**Machine Learning Algorithms in Bank Account Fraud Detection**

Department of Engineering and Computing, National University

Juan Antonio Saracho

January 30, 2023

**Machine Learning Algorithms in**

**Bank Account Fraud Detection**

**Abstract**

Credit card fraud is a serious concern for people and institutions of any kind. To put things into perspective, the December 2022 Nilson Report expects that “over the next 10 years, global losses to card fraud will amount to $397.40 billion, including $165.12 billion in the US”. Thankfully, the implementation of machine learning models gives us the ability to accurately detect fraudulent credit card transactions almost instantaneously, thereby providing institutions and consumers with a safe environment to conduct monetary transactions. The objective of this study is to create a model can accurately predict bank account fraud. To accomplish this, we will train and evaluate the performance of three popular machine learning algorithms: Logistic regression, Naïve Bayes, and XGBoost.

**Introduction**

The Federal Trade Commission reported that in 2020 fraud losses had increased more than 70% (FTC.gov, 2022). With bank account fraud being one of the most prominent and fastest growing forms of fraud it is clear to see why financial institutions and governments would spend resources coming up with methods to diminish or eliminate fraud. Previous forms of fraud detection may have included signature verification and monitoring of spending patterns, both very inefficient methods. Today, machine learning has been deemed one of the branches of computer science that is remarkably efficient at helping with these types of problems. Indeed, if we look at fraud detection of any kind through the lenses of machine learning algorithms, we may find that it can be approached as a simple classification problem. Although there exists a wide variety of classification algorithms, for this study we will focus on three specific ones: Logistic Regression, Naïve Bayes, and XGBoost. Although they are all considered good methods for classification, each model has its own strengths and weaknesses, we will see which method is better suited for bank account fraud detection by comparing each model’s performance against each other.

**Methods: Data**

For this study we will make use of a portion of the Bank Account Fraud Dataset Suite (NeurIPS 2022) which can be found on Kaggle.com. The dataset is a realistic and complete test set of bank fraud related features with five hundred thousand rows and thirty one features. Our target variable is ‘fraud\_bool’ which identifies fraud as a value of 1 and non fraud as 0 ; our original data frame has 494,784 non fraud cases, and 5,216 fraud cases. Our predictor variables are a mix of both continuous and categorical features, which include: 'income', 'employment\_status', 'credit\_risk\_score', 'housing\_status', and twenty six others. This study focuses on the relationship between bank account fraud and the thirty predictive features in our dataset.

**Methods: Data Preprocessing**

In order to select the best performing model we want to train and evaluate them using different samples. For this study we will aim to create four different samples of target and predictor variables to train our models with. These four samples will consist of a normal sample, a scaled sample, a PCA sample, and an Oversampled sample.

Before we can begin fitting our models there are some steps we must take to prepare our data for modeling. First, in order for our categorical variables to be included in our regression models they must first be transformed into numerical values. To do this we employ the use of one-hot encoding techniques to create dummy variables out of our categorical variables. Dummy variables help incorporate categorical variables into regression analysis by transforming the classes in a categorical feature into their own binary variables. After applying one-hot encoding to our data, we went from having thirty predictive variables to now having fifty one predictor variables. To reference this sample, we identify our target variable as ‘X’ and our predictor variables as ‘y’.

Having dealt with our categorical variables, the second step of our data preprocessing handles our continuous variables. In order for all of our features to have equal importance in our model, we must remove the effects of scale differences. To do this we have to standardize our numerical variables, which involves transforming them to have a mean of zero and a standard deviation of 1. To reference this sample, we identify our target variable as ‘X\_scaled’ and our predictor variables as ‘y\_scaled’.

The next sample we want to build is a PCA sample, where we take our scaled sample predictor features, ‘X\_scaled’, and perform Principal Component Analysis on it. Principal Component Analysis is a dimensionality reduction technique that effectively takes all of our variables and compresses them into a new set of features, called Principal Components. Principal components help contain most of the variance of the original datasets but with fewer features. It is recommended to perform PCA on scaled data for better model performance, hence why we use our scaled sample to create our Principal Components. After doing this, we end up with a new sample of thirty nine principal components (columns) and five hundred thousand observations (rows), called ‘X\_pca’, which collectively hold 94% of the total variation of the original scaled sample.

Lastly, we want to create a new sample that is not as imbalanced as our original sample. As we mentioned earlier, all of our samples’ target variable consists of 494,784 non fraud cases, and 5,216 fraud cases. This imbalance distorts our perception of model accuracy because it means that a model can classify everything as a non-fraud case and still have 99% accuracy, however this model would not be useful at all. To balance our data we apply Synthetic Minority Oversampling Technique (SMOTE) on our original sample. The SMOTE algorithm uses K-Nearest Neighbors to create new synthetic data points based on the original data points. One of the advantages of SMOTE is that we are not duplicating records, but creating synthetic ones that are extremely similar to the original ones (Yudha, 2020). With SMOTE we can effectively balance class distribution by randomly increasing minority class examples. After applying SMOTE to our original sample, we end up with a new target variable (y\_smote) sample of 494,784 fraud cases and 494,784 non-fraud cases, along with our 51 predictor variables (X\_smote).

**Methods: Model Evaluation**

Accuracy is a popular metric of assessing performance of classification models; it is interpreted as the proportion of predictions that our model got right. However, since we are dealing with a highly imbalanced data set where the count of frauds is significantly lower than the count of non-frauds, we must choose other evaluation metrics that takes into account the importance of correctly predicting fraud cases against the importance of correctly predicting non fraud cases. Otherwise, we will run into the issue of building a model that classifies everything as non-fraud (0) and still have a 99% accuracy score. Precision is a metric that measures the number of true positives divided by the number of positive predictions; it “quantifies the number of positive class predictions that actually belong to the positive class” (Brownlee, 2020). Recall is a metric that measures the number of true positives divided by the number of actual positives in the sample; it “quantifies the number of positive class predictions made out of all positive examples in the dataset” (Brownlee, 2020). The F1 score is the combination of both precision and recall. For our purposes we will use precision and recall, along with F1 score, as our main model evaluation metrics.

**Results: Logistic Regression**

The first model that we trained was the Linear Regression model. Before fitting our model with our sample of choice, we partitioned our sample into a training and testing sets; the training set holds 80% of the sample and the testing set holds the remaining 20%. We used SciKit Learn’s Logistic Regression function to create our model and fit it with our four samples.

Logistic Regression on the normal sample (X, y) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.69 |
| macro avg | 0.51 | 0.68 | 0.43 |
| weighted avg | 0.98 | 0.69 | 0.8 |

Logistic Regression on the scaled sample (X\_scaled, y\_scaled) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.81 |
| macro avg | 0.52 | 0.81 | 0.49 |
| weighted avg | 0.99 | 0.81 | 0.88 |

Logistic Regression on the PCA sample (X\_pca, y\_pca) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.81 |
| macro avg | 0.52 | 0.81 | 0.49 |
| weighted avg | 0.99 | 0.81 | 0.88 |

Logistic Regression on the SMOTE sample (X\_smote, y\_smote) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.82 |
| macro avg | 0.82 | 0.82 | 0.82 |
| weighted avg | 0.82 | 0.82 | 0.82 |

**Results: Naïve Bayes**

The second model that we trained was the Naïve Bayes model. As before, before fitting our model with our sample of choice, we partitioned our sample into a training and testing sets; the training set holds 80% of the sample and the testing set holds the remaining 20%. We used SciKit Learn’s GaussianNB function to create our model and fit it with our four samples.

Naïve Bayes on the normal sample (X, y) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.9 |
| macro avg | 0.53 | 0.74 | 0.53 |
| weighted avg | 0.99 | 0.9 | 0.94 |

Naïve Bayes on the scaled sample (X\_scaled, y\_scaled) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.12 |
| macro avg | 0.5 | 0.54 | 0.11 |
| weighted avg | 0.99 | 0.12 | 0.2 |

Naïve Bayes on the PCA sample (X\_pca, y\_pca) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.96 |
| macro avg | 0.53 | 0.61 | 0.54 |
| weighted avg | 0.98 | 0.96 | 0.97 |

Naïve Bayes on the SMOTE sample (X\_smote, y\_smote) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.77 |
| macro avg | 0.78 | 0.77 | 0.76 |
| weighted avg | 0.78 | 0.77 | 0.76 |

**Results: XGBoost**

The third model that we trained was the XGBoost model. As before, before fitting our model with our sample of choice, we partitioned our sample into a training and testing sets; the training set holds 80% of the sample and the testing set holds the remaining 20%. We used xgboost’s XGBClassifier function to create our model and fit it with our four samples.

XGBoost on the normal sample (X, y) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.99 |
| macro avg | 0.76 | 0.53 | 0.54 |
| weighted avg | 0.99 | 0.99 | 0.99 |

XGBoost on the scaled sample (X\_scaled, y\_scaled) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.99 |
| macro avg | 0.76 | 0.53 | 0.54 |
| weighted avg | 0.99 | 0.99 | 0.99 |

XGBoost on the PCA sample (X\_pca, y\_pca) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.99 |
| macro avg | 0.72 | 0.51 | 0.52 |
| weighted avg | 0.98 | 0.99 | 0.98 |

XGBoost on the SMOTE sample (X\_smote, y\_smote) gave the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | recall | F1 |
| accuracy |  |  | 0.99 |
| macro avg | 0.99 | 0.99 | 0.99 |
| weighted avg | 0.99 | 0.99 | 0.99 |

**Conclusion**

The objective of this study was to create a model that can accurately predict bank account fraud. To accomplish this, trained and evaluate the performance of three models: Logistic regression, Naïve Bayes, and XGBoost. To evaluate the best model performance we will focus on the weighted average F1 score, which is calculated by taking the mean of all per-class F1 scores while considering each class's support. Based off this metric alone, the XGBoost classifier is the best performing model, with an almost 99% score for all samples. However, to truly evaluate which model performed the best we are better off analyzing all metrics of the classification report all together (note that we are leaving out the confusion matrix and regression summary for simplicity of this paper) and combine those results with any domain knowledge we have on the subject at hand. Essentially, we would need to understand the costs associated with correctly classifying frauds versus incorrectly classifying them, or the costs associated with correctly classifying non-frauds versus incorrectly classifying them. Only then could we use our study results to make a data driven decision on which model better suits our needs.

**References**

Federal Trade Commission, (2022, February 22). *New Data Shows FTC Received 2.8 Million Fraud Reports from Consumers in 2021*[Press Release]. <https://www.ftc.gov/news-events/news/press-releases/2022/02/new-data-shows-ftc-received-28-million-fraud-reports-consumers-2021-0>

Brownlee Jason, PhD. (2020). How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification. Machine Learning Master. <https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/#:~:text=Specifically%2C%20you%20learned%3A-,Precision%20quantifies%20the%20number%20of%20positive%20class%20predictions,belong%20to%20the%20positive%20class>.

Yudha Cornellius. (2020). 5 SMOTE Techniques for Oversampling your Imbalanced Data. Towards Data Science. <https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5>