What combination of sampling technique and machine learning algorithm results in the highest accuracy rate for detection of fraudulent transactions?

By Nikita Golubov, Olivia Dickson

School of Mathematics, Statistics and Actuarial Science

19/05/2022

Supervised by Dr Jian Zhang

Contents

Introduction.........................................................................................4

Methodology........................................................................................6

Results..................................................................................................15

Discussion.............................................................................................23

Conclusion............................................................................................25

References............................................................................................26

Appendices...........................................................................................26

**Figures**

Figure 1: Table Summarizing Data Set Variables and their Functions..6

Figure 2: Number of Data Set Missing Values……………………………………7

Figure 3: Number of Fraudulent vs Non-Fraudulent transactions……..7

Figure 4: Histogram and Box Plot for Transactional Amount Variable.8

Figure 5: Ranges of Principle Component Variables………………………….8

Figure 6: Density Plots for some PCA variables………………………………….9

Figure 7: Correlation Plot………………………………………………………………….10

Figure 8: Train Sample Decision Tree………………………………………………..12

Figure 9: Majority Voting Visualized………………………………………………….13

Figure 10: Punnett Square Illustration……………………………………………….14

Figure 11: Decision Tree Punnet Square for Normal Sample……………..15

Figure 12: Decision Tree Punnet Square for Under Sample……………….15

Figure 13: Decision Tree Punnet Square for Up Sample…………………….15

Figure 14: Decision Tree Punnet Square for SMOTE Sample……………..15

Figure 15: Decision Tree Sensitivity Bar Graph………………………………….16

Figure 16: Decision Tree ROC Curves………………………………………………..16

Figure 17: Random Forest Punnet Square for Normal Sample………….17

Figure 18: Random Forest Punnet Square for Under Sample……………17

Figure 19: Random Forest Punnet Square for Up Sample…………………17

Figure 20: Random Forest Punnet Square for SMOTE Sample…………….17

Figure 21: Random Forest Sensitivity Bar Graph…………………………………18

Figure 22: Random Forest ROC Curves……………………………………………….18

Figure 23: GBM Punnet Square for Normal Sample……………………….……19

Figure 24: GBM Punnet Square for Under Sample………………………………19

Figure 25: GBM Punnet Square for Up Sample……………………………………19

Figure 26: GBM Punnet Square for SMOTE Sample…………………………….19

Figure 27: GBM Sensitivity Bar Graph…………………………………………………20

Figure 28: Decision Tree ROC Curves………………………………………………….20

Figure 29: xGboost Punnet Square for Normal Sample……………………….21

Figure 30: xGboost Punnet Square for Under Sample…………………………21

Figure 31: xGboost Punnet Square for Up Sample………………………………21

Figure 32: xGboost Punnet Square for SMOTE Sample……………………….21

Figure 33: xGboost Sensitivity Bar Graph……………………………………………22

Figure 34: Decision Tree ROC Curves………………………………………………….22

Figure 35: Best Model and Sample Combination Sensitivity Bar Graph.23

**Introduction**

To begin with, our chosen dataset is Credit Card Fraud Detection (Kaggle, 2018) from the website Kaggle, a reliable site whereby its primary focus is around publishing data sets for others to use. After much research we decided this data set was appropriate for our dissertation, this was due to the fact it was very compatible and relevant to what we had studied for the past year. For example, knowledge in analysing the data and finding patterns/trends will come in very handy, as we will know the exact graphs to use in order to present our findings. Plus many more relevant skills and techniques we can use, on the other hand there are also parts that we are not too familiar with, such as analysing a confidential data set which we do not have much experience with meaning we will have to make some assumptions and piece together the data and results as best as we can which will be tricky as will have to make sure the results are coherent and logical.

In terms of the data itself, it contains transactions made by credit cards in September 2013 by European cardholders [Kaggle, 2018] over a period of two days, meaning we have 284,807 transactions where 492 of them are fraudulent. Additionally, it consists of 31 variables: Time, V1-V28, Amount and Class. In terms of what they represent, Time contains seconds elapsed between each transaction and the first transaction in the dataset…Amount is the transaction amount…[and] Class is the response variable [Kaggle, 2018] which Is also binary data 1 for fraud and 0 for not. All the data is numerical due to PCA transformation apart from Time and Amount. As noted above V1-V28 is confidential meaning we do not know what it stands for which is a problem due to us wanting to make sure the results are as accurate as possible. Another problem with the data set is that it is highly unbalanced due to only a small number of transactions (0.172%) being classed as fraudulent which might make it difficult to find an accurate machine learning algorithm to fit the data.

Despite some limitations we choose it because it is a fascinating data set, with a high number of transactions, which we can analyse and enable ourselves to develop a better understanding of machine learning algorithms. As well as the fact it is a heavily unbalanced data set, which is a perfect challenge to develop said understanding. Moreover, the dataset pertains to a common real-world scenario. As fraud is a never-ending issue meaning there will always be a need to analyse and improve measures to prevent such situations, meaning our work could help unravel answers to businesses who want to use machine learning algorithms to look out for fraud. On top of that such a project is a great addition to any data scientist’s portfolio.

Carrying on from that, credit card fraud is an increasing global issue, as technology has advanced so have the tricks and techniques of fraud for example from 2011 to 2020, payment fraud globally increased from $9.84 billion to $32.39 billion [Payment Fraud Statistics, Trends & Forecasts (2020), 2022]. Therefore, more research into this area means data scientists/analysts can prevent the increasing trauma of fraud by effectively developing new and old detection algorithms which will play a part in protecting people from falling victim.

After looking over the data set, we decided that the aim of this dissertation would be to find the best detection algorithm for credit card fraud. By comparing different models such as decision trees, random forest, GBM and xGboost in combination with samplings methods such as SMOTE we will be able to draw a conclusion about which model produces the best results. Using ROC curves we will compare sensitivity vs specificity, with links to higher sensitivity meaning a higher probability of a positive result meaning occurrence of fraud and higher specificity means a negative output and fraud would not have occurred. Additionally, the area under the curve the AUC shows how well these outputs can be detected overall, a higher AUC means better model performance.

Leading on from the aim, our hypothesis will be SMOTE combined with xGboost will be the best at predicting fraud and will produce the best ROC results.

Merchant savvy were good for providing statistics to do with the prevalence of credit card fraud. Some interesting stats to do with Europe as this is where the data set is based, in terms of the top payment fraud statistics 79% were CNP (card not present) payments, 15% POS (point of sale) terminal payments and 6% were ATM payments (Merchant Savvy, 2020). This could relate to the confidential variables which would be a good guess and go well with our results if one variable had more fraudulent activity then the others. Additionally, they also proved that France had the highest CNP fraud with 72.5% (Merchant Savvy, 2020), sprouting another idea that the variables are perhaps countries. The statistics and literature show a rising issue with fraud with predications going up year by year with a 1.8 billion total value of fraud in 2018. Therefore, our research will be useful in this field as fraud clearly is not under control.

There will also be exploration of supervised and unsupervised learning which are two different types of machine learning scenarios that can be utilised for detection of fraud. We will be splitting up the data into a training and test dataset with a split of 80% to 20% whereby the training set will act as the one we use to make the different models and the test data will be used at the end to test the accuracy of the models and to evaluate the learning algorithm.

“Foundations of Machine learning” [Mehryar, Rostamizadeh and Talwalkar, 2018] goes into great detail about these scenarios and how each one is used for different situations. In supervised learning, the algorithm performs actions based on an input-output mapping function. Initially, the algorithm is taught what the appropriate response (output) is required when presented with a classified scenario (input data) after being fed with training data [Mehryar, Rostamizadeh and Talwalkar, 2018]. This is most associated with classification, regression, and ranking problems [Mehryar, Rostamizadeh and Talwalkar, 2018].

Another good book is “Artificial Intelligence: A Modern Approach” [Russell and Novrig, 2020]. Unsupervised learning does not require data which has been manually labelled. Instead, artificial intelligence would use statistical methodologies such as cluster analysis and anomaly detection [Russell and Novrig, 2020]. The book describes how this is most common in tasks to do with clustering as no labels are needed prior compared to supervised. Both books give a good indication of how to use each learning scenario and the type of results it will produce overall allowing us to pick the one best suited for our hypotheses.

One website suggests, like us, that SMOTE is the best sampling technique to use when working with an imbalanced data set. This is because it takes the minority class...and adds new examples to the data set until the quantity of the two classes are equal (Clarke, 2021). Meaning it over samples the minority in a balanced way and creates synthetic data which would only be slightly different from the primary data. Overall, creating a more balanced data set. Additionally, this combined with the xGboost which is used for gradient boosting will help our algorithm make more accurate predictions. The best model that Matt Clarke used was the xGboost as it achieved an average precision of 0.664627 (Clarke, 2021) suggesting a highly reliable and accurate model.

**Methodology**

Exploratory Analysis

The primary exploratory analysis step is to get familiar with the variables contained in our data set and their purpose:

A picture containing graphical user interface

Description automatically generated

|  |  |
| --- | --- |
| **Variable Name** | **Variable Function** |
| Time | Represents the number of seconds that have elapsed after the first data set transaction for each of the 284,807 transactions. |
| V1 to V28 | These variables represent various forms of relevant customer transaction information. Due to confidentiality concerns, these variables were converted into Principle Components through principal component analysis. |
| Amount | Denotes the magnitude of monetary value for each transaction. However, it is not disclosed what currency or currencies this data is reported in. |
| Class | Binary Classifier for fraudulent and non-fraudulent transactions. Where 0 is an instance of non-fraud and 1 is an instance of fraud. |

**Figure 1: Table Summarizing Data Set Variables and their Functions**

Notably, the “Time” variable is just a timestamp which serves a similar purpose to an “ID” variable often found in data sets. This purpose is to provide a system by which the data set records can be ordered. As such, this variable does not have any meaningful connection to the rest of the data set and will be removed as to not skew detection algorithm training.

Our next step is to check for any missing values within the data set:

Text

Description automatically generated with medium confidence

**Figure 2: Number of Data Set Missing Values**

There are no missing values within this data set. The data source does not report whether any ever existed and were removed. Consequently, imputation of data is not required.

Chart

Description automatically generated

**Figure 3: Number of Fraudulent vs Non-Fraudulent transactions**

Figure 1 visualizes the massive discrepancy between transactions classified as fraudulent vs non-fraudulent. Just 0.13% of all data set transactions are classified as fraudulent. Of course, this is natural for occurrences of fraud. However, as illustrated by Figure 1, this means the data set is highly imbalanced which will ultimately require a more sophisticated approach to training fraud detection algorithms.

Chart

Description automatically generated with low confidence

**Figure 4: Histogram and Box Plot for Transactional Amount Variable**

Figure 4 shows that most transactions in the data set were of a small amount (near 0). However, some transactions held much larger monetary values with six exceeding 100,000 units of currency and one exceeded 200,000. Additionally, the box plot in Figure 4 interestingly highlights that no fraudulent transactions exceeded the value of 50,000.

Chart, bar chart

Description automatically generated

**Figure 5: Ranges of Principle Component Variables**

Figure 5 reveals that all remaining explanatory variables (V1 to V28) have a much smaller range than the Amount variable. As such, it is necessary to scale all the explanatory variables to avoid distorting the influence of Amount on the training of the detection algorithms. Additionally, black bars on Figure 5 represent the mean values for each variable suggesting they are not normally distributed. However, further testing is required to confirm this hypothesis so we will take some of the principal component variables and run further testing:

Diagram, box and whisker chart

Description automatically generated

**Figure 6: Density Plots for some PCA variables**

Figure 6 confirms our suspicion that the Principal Component variables do not follow a normal distribution similarly to the Amount variable. As such it is inefficient to employ the use of parametric data analysis methods and detection algorithm models [Gupta et al., 2019].

Chart

Description automatically generated

**Figure 7: Correlation Plot**

Figure 6 shows intervariable relationships using the Pearson Correlation Coefficient method. The plot shows that the dependent variable Class is only weakly correlated to most other variables within the data set. However, the results of this correlation matrix cannot be fully relied upon as none of the variables within the data set are not normally distributed which is a requirement for parametric analysis methodologies such as the Pearson Correlation Coefficient metric.

Sampling Methods

To combat the highly imbalanced nature of the dataset we can utilize a variety of sampling techniques in the hopes of improving the accuracy of our detection algorithms. However, before setting up any of the sampling methods we must first create a training data set and test data set. The train data (80% of the original data set) will be used to create all other samples and will be used as a control in training the detection algorithm models. The test data (20% of original data set) will be used to create predictions for every model and training sample combination. This allows for principled comparison of algorithms performance in accurately detecting previously unseen cases of fraud from the same initial data set. Both train and test data will be selected from the original data through random sampling.

Under sampling is the first method that will be used to attempt to balance the training data set. This method aims to balance its target data set by reducing the number of observations from the majority class (Non-Fraudulent transactions in this case). It does this by removing records until both classes have an identical number of instances. There are two types of under sampling: random and informative. The first of which removes records at random while the informative method removes data entries using pre-specified selection criteria until the data set is balanced [Imbalanced Classification Problems in R, 2022]. In this case, we utilized the random method. A disadvantage of this method is that it causes massive data loss to balance the data set. However, due to the reduced sample size, runtimes of detection algorithms are much faster.

Oversampling functions inversely by increasing the number of observations from the minority class (fraudulent transactions). It functions by replicating records from the underrepresented class until the data set is balanced. Standard under-sampling is random replication of existing records. As a result, models using oversampling often suffer from overfitting as they contain multiple identical records from the minority class. However, unlike under sampling, this method does not cause any data loss.

Synthetic Minority Oversampling Technique (SMOTE) is an informative version of oversampling. Unlike normal oversampling, SMOTE does not replicate records from the minority class but instead generate artificial version of what they could be until the data set is balanced. It functions by slightly taking the difference between a newly generated record’s nearest neighbour and a randomly selected feature vector (average statistics of a sample taken from the data set). It then multiplies this calculated difference by a number between 0 and 1 and adding the result to the feature vector. This method avoids the overfitting usually associated with standard oversampling.

Modelling Methods

It was important to choose methods of non-parametric modeling for binary classification as the data set variables are not normally distributed. Initially, we planned to use xGboost, lightGBM and Artificial Neural Network. However, we faced some challenges with the attempted implementation of the latter two methodologies. Artificial Neural Networks had unreasonably long run times due to their complex process of binary classification. LightGBM consistently produced extremely overfitted models where all transactions were predicted as non-fraudulent. Instead, we utilized Decision Tree, Random Forest, GBM and xGboost modeling as detection algorithms.

Decision trees are a simple yet surprisingly effective learning algorithm which can be used for both classification and regression modeling. Classification trees are a supervised learning algorithm which consists of nodes, branches and leaves. It functions via a top-down approach of splitting data through recursive evaluation. At the top, a decision tree selects the optimal variable for splitting data evenly to become the root node. From the root node, two branches consisting of “yes” or “no” determine the path that a particular record will follow which depends on a certain condition for the root node variable such as x > 12. This process is then repeated through intermediary decision nodes which present conditions for other select data set variables until we reach the leaves of the model. Each leaf represents the final output reached for that record. In our case, the leaves denote whether a data entry is considered as a fraudulent or a non-fraudulent transaction (Chauhan, 2022).

Diagram

Description automatically generated

**Figure 8: Train Sample Decision Tree**

Figure 8 demonstrates how a binary classification decision tree functions with a model built on our training data set. Seemingly, all pathways have 0% probability of being followed except for one which has 100%. However, this is a misrepresentation which occurred because all other pathways have extremely small probabilities like 0.01% which would not fit on the nodes and leaves represented in Figure 8 unless they were rounded down.

A random forest model functions by combining multiple decision trees for an optimal result. It does this by selecting a specific number of smaller samples from the original training data set and building a decision tree for each one (such as the classification tree depicted in Figure 8). This process is known as Bootstrap Aggregation (Bagging), where sampling from the original train set is performed with replacement (meaning the same records can be selected as sample more than once). When a test sample record is put through the random forest model, it decides on the final binary classification output through majority voting (see Figure 9) [Sruthi R, 2022]. Random forest models can also be used for regression modeling.

Diagram

Description automatically generated

**Figure 9: Majority Voting Visualized [Sruthi R, 2022]**

Generalized Boosted Regression Model or Gradient Boosting Machine (GBM) is a more advanced version of the random forest model. Just like the random forest model, GBM runs multiple decision trees on random samples derived from the training data set. However, unlike the random forest model, GBM creates its weak learners (decision trees) sequentially meaning that it uses the errors in the predictions of previous classification trees to build more accurate future decision trees [Singh, 2022]. It does this by applying a higher weighting to incorrect predictions and a lower weighting to correct predictions of previous weak learners. It then utilizes a loss function which is minimized using an additive model to create more accurate weak future weak learners [Brownlee, 2022].

Where a loss function is an additive model is an algebraic equation which can be rearranged to minimise a particular component (algebraic term). Within economics, a common loss function is a production function which can be minimised through differentiation to minimise production cost by providing an expression with optimal levels of capital and labour. Within GBM, incorrect predictions of previous weak learners are minimised to create a strong model consisting of many weak learners. This additive model is also known as a Gradient Descent Algorithm which is where the name Gradient Boosting originates from.

xGboost stands for Extreme Gradient Boosting and functions in a very similar manner to normal GBM. However, this modelling method differs in how it builds its consecutive weak learners (classification decision trees). Instead of building them sequentially, xGboost uses parallel boosting to build them simultaneously reducing prediction errors in all weak learners and decreasing run times. Additionally, xGboost is far more effective at avoiding overfitting as it has more regularization than its predecessor GBM. This algorithm even can deal with missing values without prior imputation [Morde, 2022].

Cross Validation

The purpose of this report is to compare combinations of sampling methods with various machine learning detection algorithms. Naturally, we require metrics that measure the accuracy of their predictions which can be used for model cross-examination (cross validation). For this purpose, we will be employing the use of Punnett Squares to allow for the derivation of total accuracy, specificity and sensitivity. Additionally, we will be employing the use of the Receiver Characteristic Operator Curves (ROC).

Punnett Squares, often referred to as a confusion matrix, is a table that displays the number of accurate and false predictions. It consists of the number of true positive, false positive, false negative and true negative predictions. Where true positive predictions are instances which were correctly predicted as belonging to binary class 1 (fraudulent transaction in this case). Thereby false positive instances are predictions classified as belonging to class 0 but belong to class 1. The opposite is true for false negative and true negative predictions (see Figure 10 for a visual representation of a Punnett Square).

A picture containing chart

Description automatically generated

**Figure 10: Punnett Square Illustration [Bhandari, 2022]**

Sensitivity is the percentage of the positive class (fraud) that was correctly predicted and can be referred to as the true positive rate. Specificity is the percentage of the negative class (not fraud) that was correctly predicted also known as the true negative rate. The ROC curve plots Sensitivity against the false negative ratee (1 – Specificity). In the case of transaction fraud, we would definitely prefer a higher Sensitivity which signals how well the algorithm is able to detect fraud. A human operator is likely to investigate each case marked as fraudulent so false positive results would not be greatly detrimental unless there was an overwhelming amount of them.

Therefore, Area Under the ROC Curve (AUC) is the metric that measures how well the machine learning detection algorithm’s predictions are able to correctly distinguish between the positive and negative class. Where and AUC of 1 means a model can perfectly distinguish between cases of fraud and non-fraud. On the other hand, an AUC of 0 would mean that all positive results are predicted as negative, and all negative results are predicted as positive.

**Results**

As criteria for the optimal model and sample combination the highest Sensitivity scorers will be chosen as this signifies that they are the most effective at predicting occurrences of fraud correctly. However, any model with a Specificity of less that 97% will be disqualified as this would mean this would signify that over 1700 of false positive cases. This would be an unreasonable number of transactions for a human operator to check manually.

Decision Tree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 11: Decision Tree Punnet Square for Normal Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56859 | 23 | 99.99% | 76.53% |
| **1 (Fraud)** | 6 | 75 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.95% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 12: Decision Tree Punnet Square for Under Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 54666 | 15 | 96.13% | 84.69% |
| **1 (Fraud)** | 2199 | 83 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 96.11% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 13: Decision Tree Punnet Square for Up Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 54338 | 17 | 95.56% | 82.65% |
| **1 (Fraud)** | 2527 | 81 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 95.53% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 14: Decision Tree Punnet Square for SMOTE Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56139 | 18 | 98.72% | 81.63% |
| **1 (Fraud)** | 726 | 80 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.69% | | |

Chart, bar chart

Description automatically generated

**Figure 15: Decision Tree Sensitivity Bar Graph**

Line chart

Description automatically generated

**Figure 16: Decision Tree ROC Curves**

All decision tree models were fairly weak with an average Sensitivity of just 81.37% (see Figure 15). This means it predicted 81.37% of fraudulent transactions in the test data set correctly. The SMOTE sample produced the best combination with the Decision Tree model even though it had the third highest Sensitivity after under sample and up sample. However, both Up and Under samples had a Specificity of less than 97%. The normal sample has a very impressive Specifcity of 99.99% but it’s lower AUC of 0.908 (see Figure 16) suggests that this may be due to over fitting as the model is not as good at distinguishing between fraudulent and non-fraudulent cases.

Random Forest

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 17: Random Forest Punnet Square for Normal Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56864 | 24 | 99.98% | 75.51% |
| **1 (Fraud)** | 1 | 74 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.96% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 18: Random Forest Punnet Square for Under Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 55968 | 9 | 98.42% | 90.82% |
| **1 (Fraud)** | 897 | 89 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.41% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 19: Random Forest Punnet Square for Up Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56864 | 26 | 99.98% | 73.47% |
| **1 (Fraud)** | 1 | 72 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.95% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 20: Random Forest Punnet Square for SMOTE Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56464 | 26 | 99.29% | 88.78% |
| **1 (Fraud)** | 401 | 87 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.28 | | |

Chart, bar chart

Description automatically generated

**Figure 21: Random Forest Sensitivity Bar Graph**

Chart, line chart

Description automatically generated

**Figure 22: Random Forest ROC Curves**

On average, random forest has a slightly higher Sensitivity on average than the decision tree models at 82.15% (See Figure 21). However, its under sample and SMOTE sample models significantly outperformed decision tree models with the same samples. Under sample and random combination produced an impressive Sensitivity score of 90.82% meaning it mis-identified less than 10% of fraudulent cases in the test data. This was accompanied by an oustounding AUC of 0.974 (see Figure 22) showing that this combination was great at distinguishing between positive and negative class instances.

GBM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 23: GBM Punnet Square for Normal Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56860 | 28 | 99.99% | 71.43% |
| **1 (Fraud)** | 5 | 70 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.94% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 24: GBM Punnet Square for Under Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56235 | 12 | 98.89% | 87.76% |
| **1 (Fraud)** | 630 | 86 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.87% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 25: GBM Punnet Square for Up Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56126 | 13 | 98.70% | 86.73% |
| **1 (Fraud)** | 739 | 85 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.68% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 26: GBM Punnet Square for SMOTE Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56208 | 12 | 98.84% | 87.76% |
| **1 (Fraud)** | 657 | 86 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.83% | | |

Chart, bar chart

Description automatically generated

**Figure 27: GBM Sensitivity Bar Graph**

Graphical user interface, line chart

Description automatically generated

**Figure 28: Decision Tree ROC Curves**

GBM produced very consistent high Sensitivity rates across the board except for with the normal sample (see Figure 27). It scored an average Sensitivity of 83.42% with it’s best two models (which used under and SMOTE samples) producing an identical rate of 87.76%. However, the SMOTE sample slightly beat the under sample combination with GBM as it had a slightly higher AUC of 0.978 compared to 0.972 (see Figure 28). Which means that the GBM and SMOTE combination was better at distinguishing between fraudulent and non-fraudulent transaction.

xGboost

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 29: xGboost Punnet Square for Normal Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56863 | 23 | 99.996% | 76.53% |
| **1 (Fraud)** | 2 | 75 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.96% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 30: xGboost Punnet Square for Under Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 55384 | 8 | 97.40% | 91.84% |
| **1 (Fraud)** | 1481 | 90 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 97.39% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 31: xGboost Punnet Square for Up Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56863 | 21 | 99.996% | 78.57% |
| **1 (Fraud)** | 2 | 77 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 99.96% | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure 32: xGboost Punnet Square for SMOTE Sample** | | | | |
|  | **Actual** | | **Accuracy** | |
| **Predicted** | **0 (Not Fraud)** | **1 (Fraud)** | **Not Fraud** | **Fraud** |
| **0 (Not Fraud)** | 56274 | 13 | 98.96% | 86.73% |
| **1 (Fraud)** | 591 | 85 |
| **Total Accuracy** | | | **Specificity** | **Sensitivity** |
| 98.94% | | |

Chart, bar chart

Description automatically generated

**Figure 33: xGboost Sensitivity Bar Graph**

Diagram, line chart

Description automatically generated

**Figure 34: Decision Tree ROC Curves**

Under wins

The xGboost models also had an average sensitivity of 83.42%. However, the xGboost and under sample combination significantly outperformed all GBM models with a Sensitivity rate of 91.84% (see Figure 33). However, it did have a comparatively underwhelming AUC of 0.969 (see Figure 34). In fact, this model almost did not meet the selection of best model criteria as it had a Specificity rate of 97.4 (see Figure 30).

Best Model

Chart, bar chart

Description automatically generated

**Figure 35: Best Model and Sample Combination Sensitivity Bar Graph**

Figure 35 shows that under sample and xGboost combination produced the machine learning algorithm with the highest accuracy in fraud detection (Sensitivity) of 91.84%. In second place, we have the combination of random forest and under sampling at 90.82%. However, xGboost and GBM were the best at accurately predicting fraud across the board with an average Sensitivity rate of 83.42%.

**Discussion**

Overall, our results have proven our hypothesis cannot be accepted because our best combination was xGboost with under sampling as shown in Figure 35. By testing out different sampling methods it enabled us to see which worked the best with an imbalanced data set which gave us new skills and knowledge. Additionally, GBM and xGboost SMOTE also had identical average Sensitivity rate showing another variation of models and samples which works well for this data set. We definitely met our aim as we tested out most of the models we planned to and used the Punnett Square to calculate the sensitivity and specificity as well as use the ROC cure to look at the AUC to check model performance.

The implications of the results are extraordinarily strong as no other R code on Kaggle compares as many models in combination with as many sampling methods as this report, suggesting that we have well founded results which has more precision due to considering more options. With regard to the literature, our results contradict the article that suggests SMOTE combined with xGboost would be the best model, even though we thought it would be as well, our findings show that xGboost combined with under sampling was the best. Therefore, we had the same idea with xGboost and clearly is a good model to use however our sampling technique had higher results with 98.4% and theirs producing 83% (Clarke, 2021).

The strengths of our work outweigh the limitations, due to such a thorough and extensive analysis on the chosen data set which in turn has provided us with accurate and practical results. As shown through our data above such as Figure 35 which represents the sensitivity of the top 4 models and sample combinations, with our top being XGboost with under sampling which correctly predicted fraud cases with 91.84% accuracy. You can see as well that all the models build on each other in terms of complexity and they all had quite a high accuracy percentage meaning the methods we used did go well with the data and produce successful results. This also resulted with an in-depth comparison of model performances to identify the best combination of model and sampling method.

Moreover, a wide use of sampling methods such as SMOTE, under sampling and up sampling, has meant that we were able to test the data against multiple methods to see which one gave the best results and lead to a higher reliability rate of our results. Leading on from that, despite the data being extremely imbalanced, models used such as xGboost were able to build a fairly accurate detection algorithm for the data meaning the results would have been more accurate with it.

Our strengths also lie with our graphical summaries and producing plenty of different graphs such as bar graphs, correlation plots, density plots, ROC curves, crossbar graphs, histogram and finally boxplot meaning there was plenty of aesthetically pleasing summaries to look at that explain the data well and show all the information someone would need to see to show what the data means and its results without having to know much about it.

Normally, multiple training and test data sets are created to re-run detection algorithms to assess how consistent their prediction results are. However, in this case, it was unreasonable to use more than one train and test data split as this would have produced an overwhelming number of results due to the combination of four different sampling techniques with four different detection algorithms. Although, this was not a massive issue for the focus of the report (comparing effectiveness of different algorithms) as all models were trained and tested on the same sample data sets allowing for consistent evaluation.

Limitations of the actual data were that it only provided numerical input variables and no background information, meaning we cannot apply our derived machine learning algorithm to real life credit card fraud cases. As well as there being a low rate of transactions in the data which were fraudulent meaning it was highly imbalanced and even though we used methods to help make it more accurate there will always be some inaccuracy due to the data sets nature.

Lastly, we also deviated from the original plan which was use neural net however that had extremely long training times and Light GBM which produced overfitted results predicting that all instances were non-fraud.

In terms of future work, more work could be done on the code to enhance the results even more and perhaps see what else it produces, such as multiple training and test data set samples to improve the performance by picking out more variables and seeing if that affects the accuracy of the model more so than the first set. Additionally, there could be an adjustment on the number of trees in random forest, GBM and xGboost depending on the sampling method used as increasing and decreasing will allow you to see how much the results differ as well as ensuring you tune the parameter to avoid over-fitting

Other ways this could be used for future work is utilising more advanced methods of under-sampling within R such as EasyEnsamble and BalanceCascade. These are forms of informative under sampling techniques (where records are removed based on pre-specified criteria) which would have reduced the negative effects of data loss which is associated with all under sampling methods. This will improve the validity of under-sampling even more and see whether these two options are better to use then our chosen method, which will lead to even more accurate fraud detection algorithms.

Lastly, we could utilise Cost sensitive learning which is a subfield of machine learning that takes the costs of prediction errors into account when training a machine learning model (Brownlee, 2020). It is great at dealing with imbalanced data like ours and classification methods, this would be interesting to investigate further and see if our models vary with it.

**Conclusion**

In summary, this whole project has taught us that you do not necessarily need all the information to make valid and accurate results, it is all about understanding the data and deciding what models and methods go best with it, and I feel that has been accomplished within this highly outstanding report.

**Bibliography**

Analytics Vidhya. 2022. *Imbalanced Classification Problems in R*. [online] Available at: <https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/#:~:text=1.-,Undersampling,run%20time%20and%20storage%20troubles.> [Accessed 8 May 2022].

Brownlee, J., 2022. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/> [Accessed 27 March 2022].

Brownlee, J. 2020. Cost-Sensitive Learning for Imbalanced Classification. Available at: https://machinelearningmastery.com/cost-sensitive-learning-for-imbalanced-classification/

Clarke, M. 2021. *How to use SMOTE for imbalanced classification*. Available at: https://practicaldatascience.co.uk/machine-learning/how-to-use-smote-for-imbalanced-classification

Gupta, A., Mishra, P., Pandey, C., Singh, U., Sahu, C. and Keshri, A., 2019. Descriptive statistics and normality tests for statistical data. *Annals of Cardiac Anaesthesia*, 22(1), p.67.

Kaggle. 2018. *Credit Card Fraud Detection.* Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download

Mehryar, M., Rostamizadeh, A. and Talwalkar, A., 2018. Foundations of Machine Learning. 2nd ed. The MIT Press, p.6.

Merchant savvy. 2020. *Global Payment Fraud Statistics, Trends & Forecasts.* Available at: https://www.merchantsavvy.co.uk/payment-fraud-statistics/

Morde, V., 2022. *XGBoost Algorithm: Long May She Reign!*. [online] Medium. Available at: <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d> [Accessed 15 May 2022].

Russell, S. and Novrig, P., 2020. *Artificial Intelligence: A Modern Approach*. 4th ed.

Sruthi R, E., 2022. *Random Forest | Introduction to Random Forest Algorithm*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/#:~:text=Random%20forest%20is%20a%20Supervised,average%20in%20case%20of%20regression.> [Accessed 15 May 2022].

**Appendix (R Code)**

########################################

#Setting working directory instructions#

########################################

#Before running the following code it is important to first set the working directory to the folder where the original

#data file is stored. This can be done in the following ways:

#Navigate to: Session > Set Working Directory > Choose Directory

#Alternatively you can press: Ctrl + Shift + H

#Or you can type the code: setwd("") and type the folder directory into the quotation marks

setwd("C:\\Users\\EpiNick\\Desktop\\Data Analytics\\MAST5957")

##############################################

#Installing and calling all relevant packages#

##############################################

if (!("tidyverse" %in% installed.packages())) {

install.packages("tidyverse")}

library(tidyverse)

if (!("dplyr" %in% installed.packages())) {

install.packages("dplyr")}

library(dplyr)

if (!("grid" %in% installed.packages())) {

install.packages("grid")}

library(grid)

if (!("gridExtra" %in% installed.packages())) {

install.packages("gridExtra")}

library(gridExtra)

if (!("corrplot" %in% installed.packages())) {

install.packages("corrplot")}

library(corrplot)

if (!("tidyr" %in% installed.packages())) {

install.packages("tidyr")}

library(tidyr)

if (!("ggplot2" %in% installed.packages())) {

install.packages("ggplot2")}

library(ggplot2)

if (!("RColorBrewer" %in% installed.packages())) {

install.packages("RColorBrewer")}

library(RColorBrewer)

if (!("factoextra" %in% installed.packages())) {

install.packages("factoextra")}

library(factoextra)

if (!("cluster" %in% installed.packages())) {

install.packages("cluster")}

library(cluster)

if (!("dendextend" %in% installed.packages())) {

install.packages("dendextend")}

library(dendextend)

if (!("ranger" %in% installed.packages())) {

install.packages("ranger")}

library(ranger)

if (!("caret" %in% installed.packages())) {

install.packages("caret")}

library(caret)

if (!("data.table" %in% installed.packages())) {

install.packages("data.table")}

library(data.table)

if (!("scales" %in% installed.packages())) {

install.packages("scales")}

library(scales)

if (!("rpart" %in% installed.packages())) {

install.packages("rpart")}

library(rpart)

if (!("rpart.plot" %in% installed.packages())) {

install.packages("rpart.plot")}

library(rpart.plot)

if (!("caTools" %in% installed.packages())) {

install.packages("caTools")}

library(caTools)

if (!("Metrics" %in% installed.packages())) {

install.packages("Metrics")}

library(Metrics)

if (!("gbm" %in% installed.packages())) {

install.packages("gbm")}

library(gbm)

if (!("pROC" %in% installed.packages())) {

install.packages("pROC")}

library(pROC)

if (!("xgboost" %in% installed.packages())) {

install.packages("xgboost")}

library(xgboost)

if (!("cvms" %in% installed.packages())) {

install.packages("cvms")}

library(cvms)

if (!("randomForest" %in% installed.packages())) {

install.packages("randomForest")}

library(randomForest)

if (!("ggpubr" %in% installed.packages())) {

install.packages("ggpubr")}

library(ggpubr)

#Dependencies for DMwR package:

if (!("xts" %in% installed.packages())) {

install.packages("xts")}

library(xts)

if (!("quantmod" %in% installed.packages())) {

install.packages("quantmod")}

library(quantmod)

if (!("zoo" %in% installed.packages())) {

install.packages("zoo")}

library(zoo)

if (!("ROCR" %in% installed.packages())) {

install.packages("ROCR")}

library(ROCR)

#Package DMwR which contains the SMOTE function has been removed from the CRAN library.

#However, it can be installed from the CRAN archives from the following link:

#https://cran.r-project.org/src/contrib/Archive/DMwR/

#After downloading the latest version you will need to provide the directory path to it

#in the following code:

if (!("DMwR" %in% installed.packages())) {

install.packages( "C:\\Users\\EpiNick\\Desktop\\Data Analytics\\MAST5957\\DMwR\_0.4.1.tar.gz", repos=NULL, type="source" )} #Change this line to make it function

library(DMwR)

###############

#Data Cleaning#

###############

data <- read.csv("creditcard.csv") #Import Data Set

#Exploratory Analysis:

head(data) #Show's variable names as well as the first 6 data entries for said variables.

sum(is.na(data)) #Check for missing values

table(data$Class) #Compares how many variables fall into is fraud and is not fraud categories.

#Interestingly, only around 0.13% of transactions are fraudulent.

#######################

#Bar Chart for isFraud#

#######################

#Create Data frame contaning count stats for instances of Fraud and Non-Fraud:

df\_Class <- data.frame(TransactionType = c("Fraud", "Not\_Fraud"), Count = c(492, 284315))

#Formats the axis for all upcoming line graphs. Makes all axis text grey so that axis and plot titles can stand out:

theme\_set(theme(

axis.line = element\_line(colour = "grey87"),

axis.ticks = element\_line(colour = "grey87"),

axis.text = element\_text(colour = "grey40", size = 12),

))

#Bar chart for fraudelent vs non-fraudelent transactions:

ggplot(data=df\_Class, aes(x = TransactionType, y = Count)) +

geom\_bar(stat = "identity", fill = "steelblue") + #Formats the bars.

scale\_y\_continuous("Count", labels = scales::comma) + #Scales the Y-axis.

geom\_text(aes(label = Count), vjust = -0.2) + #Adds data labels just above each bar.

labs(

title = "Number of fraudulent vs non-fraudulent transactions", #Sets a plot title.

Count = "Count", #Changes y-axis title.

TransactionType = "Type of Transaction" #Changes x-axis title.

) +

theme(

plot.title = element\_text(size = 16, face = "bold", hjust = 0.5), #Formats plot title.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#Box Plot for Amount by Class

Box\_plot1 <- ggplot(data, aes(y = Amount, x = factor(Class))) + geom\_boxplot() +

labs(

title = "Distribution of Transaction Amount by Class", #Sets a plot title.

y = "Amount", #Changes y-axis title.

x = "Class" #Changes x-axis title.

) +

theme(

plot.title = element\_text(size = 16, face = "bold", hjust = 0.5), #Formats plot title.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#Histogram for Amount

Histogram1 <- ggplot(data, aes(x = Amount)) +

geom\_histogram(fill = "steelblue", binwidth = 1000) + #Formats the bars.

scale\_y\_continuous("Count", labels = scales::comma) + #Scales the Y-axis.

labs(

title = "Number of Transactions by Amount", #Sets a plot title.

y = "Count", #Changes y-axis title.

x = "Type of Transaction" #Changes x-axis title.

) +

theme(

plot.title = element\_text(size = 16, face = "bold", hjust = 0.5), #Formats plot title.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

grid.arrange(Box\_plot1, Histogram1) #Arrange Histogram and Box plot for Amount on the Same plot

attach(data) #Attach data to create the following data frame

#Create Data from for the Ranges and mean values of all PCA variables:

df\_PCA <- data.frame (Names = c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8",

"V9", "V10", "V11", "V12", "V13", "V14", "V15",

"V16", "V17", "V18", "V19", "V20", "V21", "V22",

"V23", "V24", "V25", "V26", "V27", "V28"),

Start = c(min(V1), min(V2), min(V3), min(V4), min(V5),

min(V6), min(V7), min(V8), min(V9), min(V10),

min(V11), min(V12), min(V13), min(V14), min(V15),

min(V16), min(V17), min(V18), min(V19), min(V20),

min(V21), min(V22), min(V23), min(V24), min(V25),

min(V26), min(V27), min(V28)),

End = c(max(V1), max(V2), max(V3), max(V4), max(V5),

max(V6), max(V7), max(V8), max(V9), max(V10),

max(V11), max(V12), max(V13), max(V14), max(V15),

max(V16), max(V17), max(V18), max(V19), max(V20),

max(V21), max(V22), max(V23), max(V24), max(V25),

max(V26), max(V27), max(V28)),

Mean = c(mean(V1), mean(V2), mean(V3), mean(V4), mean(V5),

mean(V6), mean(V7), mean(V8), mean(V9), mean(V10),

mean(V11), mean(V12), mean(V13), mean(V14), mean(V15),

mean(V16), mean(V17), mean(V18), mean(V19), mean(V20),

mean(V21), mean(V22), mean(V23), mean(V24), mean(V25),

mean(V26), mean(V27), mean(V28))

)

positions <- c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", #Creates data frame which will be used to order variables on the

"V9", "V10", "V11", "V12", "V13", "V14", "V15", #cross-bar plot

"V16", "V17", "V18", "V19", "V20", "V21", "V22",

"V23", "V24", "V25", "V26", "V27", "V28")

ggplot(data = df\_PCA, aes(x = Names, y = Mean)) + #GGplot setup

geom\_crossbar(aes(ymin = Start, ymax = End), width = 0.5, fill = "skyblue") + #Create Cross-bar plot for PCA variable Ranges

scale\_x\_discrete(limits = positions) + #Ensures Bars are ordered by their names

coord\_flip() +

labs(

title = "Range and Mean of PCA Variables", #Sets a plot title.

y = "Value", #Changes y-axis title.

x = "Variable Name" #Changes x-axis title.

) +

theme(

plot.title = element\_text(size = 16, face = "bold", hjust = 0.5), #Formats plot title.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#Creating denisty plots for select PCA variables to test for normality:

V1\_Density <- ggdensity(data$V1,

main = "Density plot of V1",

xlab = "V1 Value")

V7\_Density <- ggdensity(data$V7,

main = "Density plot of V7",

xlab = "V7 Value")

V21\_Density <- ggdensity(data$V21,

main = "Density plot of V21",

xlab = "V21 Value")

V28\_Density <- ggdensity(data$V28,

main = "Density plot of V28",

xlab = "V28 Value")

grid.arrange(V1\_Density, V7\_Density, V21\_Density, V28\_Density)

df <- data[,-1] #Create data frame to preserve original data set. Time variable was removed as it is only used to order the data and messes with some of modelling techniques due to it's large values

df[,-30] <- scale(df[,-30]) #All PCA numeric variables were scaled to ensure their magnitudes do not skew the detection algorithms importance of them

#Correlations

correlation <- cor(df , method = "pearson") #Runs correlations for the data set variables

corrplot(correlation, number.cex = 1.5, method = "square", type = "full", tl.cex = 0.8, tl.col = "black", #Plots correlation matrix

col = colorRampPalette(c("azure3", "dodgerblue", "dodgerblue4","navyblue"))(100), cl.lim = c(0, 1)) #Sets custom colour scheme for the correlation matrix

##############################

#Creating Test and Train data#

##############################

df <- data[,-1] #Create data frame to preserve original data set. Time variable was removed as it is only used to order the data and messes with some of modelling techniques due to it's large values

df[,-30] <- scale(df[,-30]) #All PCA numeric variables were scaled to ensure their magnitudes do not skew the detection algorithms importance of them

df$Class <- as.factor(df$Class) #Convert Class variable into factor to allow for up and down sampling

set.seed(120) #Ensures all following random procedures can be replicated when code is re-run

sample<-sample.split(df, SplitRatio = 0.8) #This first code is used to split the data into 80% and 20%

train <- subset(df, sample== TRUE) #80% is put into the train data to test and run

test <- subset(df, sample== FALSE) #20% is put into the test data to predict later on

dim(train) #Returns the number of elements in the data

dim(test) #Returns the number of elements in the data

#Create under-sampled train model

train\_down <- downSample(x = train [, -ncol(train)], y = train$Class)

table(train\_down$Class)

#Create Up-sampled train data

train\_up <- upSample(x = train[, -ncol(train)], y = train$Class)

table(train\_up$Class)

#Create synthetically over sampled minority class train data (using SMOTE):

train\_smote <- SMOTE(Class ~ ., data = train)

table(train\_smote$Class)

###############

#Decision Tree#

###############

#Annotation of code for all models was limited to the first instance where the model was run

#with a normal sample as every all other training code was identical to this original.

set.seed(120)

#Decision Tree Normal Sample

decisionTree\_model\_norm <- rpart(Class ~ . , train, method = 'class') #Creates regression decision tree model. Method "class" is suitable for binary classification

rpart.plot(decisionTree\_model\_norm) #Plots decision tree model

tree\_norm\_pred <- predict(decisionTree\_model\_norm, newdata = test, type = 'class') #Predicts values using the decision tree model with test data

table(predicted = tree\_norm\_pred, actual = test$Class) #Create a Punnet Square for Decision Tree prediction results

tree\_norm\_prob <- predict(decisionTree\_model\_norm, newdata = test, type = "prob") #Predicts the probabilities of cases being Fraud or not Fraud using decision tree model

roc(test$Class, tree\_norm\_prob[, 2], plot = TRUE, print.auc = TRUE, #Creates and stores ROC Curve for our Decision Tree model prediction

main = "Decision Tree Normal Sample")

#Decision Tree Down Sample

decisionTree\_model\_down <- rpart(Class ~ . , train\_down, method = 'class')

tree\_down\_pred <- predict(decisionTree\_model\_down, newdata = test, type = 'class')

table(predicted = tree\_down\_pred, actual = test$Class)

tree\_down\_prob <- predict(decisionTree\_model\_down, newdata = test, type = "prob")

roc(test$Class, tree\_down\_prob[, 2], plot = TRUE,

print.auc = TRUE, main = "Decision Tree Under Sample")

#Decision Tree Up Sample

decisionTree\_model\_up <- rpart(Class ~ . , train\_up, method = 'class')

tree\_up\_pred <- predict(decisionTree\_model\_up, newdata = test, type = 'class')

table(predicted = tree\_up\_pred, actual = test$Class)

tree\_up\_prob <- predict(decisionTree\_model\_up, newdata = test, type = "prob")

roc\_tree\_up <- roc(test$Class,tree\_up\_prob[,2], plot = TRUE,

print.auc = TRUE, main = "Decision Tree Up Sample")

#Decision Tree SMOTE Sample

decisionTree\_model\_smote <- rpart(Class ~ . , train\_smote, method = 'class')

tree\_smote\_pred <- predict(decisionTree\_model\_smote, newdata = test, type = 'class')

table(predicted = tree\_smote\_pred, actual = test$Class)

tree\_smote\_prob <- predict(decisionTree\_model\_smote, newdata = test, type = "prob")

roc\_tree\_smote <- roc(test$Class, tree\_smote\_prob[, 2], plot = TRUE,

print.auc = TRUE, main = "Decision Tree SMOTE Sample")

#Create Data Frame for Decision Tree Sensitivity Bar Chart:

df\_Tree <- data.frame (Sample = c("Normal","Under", "Up", "SMOTE"),

Sensitivity = c(76.53, 84.69, 82.65, 81.63)

)

order\_tree <- c("Under", "Up", "SMOTE", "Normal") #Create variable to order bars

ggplot(data=df\_Tree, aes(x = Sample, y = Sensitivity)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = Sensitivity), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_tree) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "Decision Tree Sensitivity as Percentage for each Sample", #Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

###############

#Random Forest#

###############

set.seed(120)

#Random Forest Normal Sample

randomForest\_norm <- randomForest(Class~., #Creates random forest model

data = train, #Selects data set which is used to train the model

minNode = 20, #Sets the minimum number of nodes contained within each decision tree

maxLeaf = 13, #Sets the maximum number of leaves (outputs) each tree can have

ntree=500) #Tells the model how many trees to use to train the model.

forest\_norm\_pred <- predict(randomForest\_norm, test) #Predicts values using the random forest model with test data

table(forest\_norm\_pred, test$Class) #Creates Punnet Square for Random Forest predictions.

forest\_norm\_prob <- predict(randomForest\_norm, newdata = test, type = "prob") #Calculated prediction probabilities to be used in ROC curve creation

roc\_tree\_norm <- roc(test$Class, forest\_norm\_prob[, 2], plot = TRUE, #Creates and stores ROC curve for the model

print.auc = TRUE, main = "Random Forest Normal Sample")

#Random Forest Down Sample

randomForest\_down <- randomForest(Class~.,

data = train\_down,

minNode = 20,

maxLeaf = 13,

ntree=500)

forest\_down\_pred <- predict(randomForest\_down, test)

table(forest\_down\_pred, test$Class)

forest\_down\_prob <- predict(randomForest\_down, newdata = test, type = "prob")

roc\_tree\_down <- roc(test$Class, forest\_down\_prob[, 2], plot = TRUE,

print.auc = TRUE, main = "Random Forest Under Sample")

#Random Forest Up Sample

randomForest\_up <- randomForest(Class~.,

data = train\_up,

minNode = 20,

maxLeaf = 13,

ntree=500)

forest\_up\_pred <- predict(randomForest\_up, test)

table(forest\_up\_pred, test$Class)

forest\_up\_prob <- predict(randomForest\_up, newdata = test, type = "prob")

roc\_tree\_up <- roc(test$Class, forest\_up\_prob[, 2], plot = TRUE,

print.auc = TRUE, main = "Random Forest Up Sample")

#Random Forest SMOTE Sample

randomForest\_smote <- randomForest(Class~.,

data = train\_smote,

minNode = 20,

maxLeaf = 13,

ntree=500)

forest\_smote\_pred <- predict(randomForest\_smote, test)

table(forest\_smote\_pred, test$Class)

forest\_smote\_prob <- predict(randomForest\_smote, newdata = test, type = "prob")

roc\_tree\_smote <- roc(test$Class, forest\_smote\_prob[, 2], plot = TRUE,

print.auc = TRUE, main = "Random Forest SMOTE Sample")

#Create Data Frame for Random Forest Sensitivity Bar Chart:

df\_forest <- data.frame (Sample = c("Normal","Under", "Up", "SMOTE"),

Sensitivity = c(75.51, 90.82, 73.47, 88.78)

)

order\_forest <- c("Under", "SMOTE", "Normal", "Up") #Create variable to order bars

ggplot(data=df\_forest, aes(x = Sample, y = Sensitivity)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = Sensitivity), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_forest) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "Random Forest Sensitivity as Percentage for each Sample", #Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#####

#GBM#

#####

set.seed(120)

#GBM Normal Sample

gbm\_norm <- gbm(as.character(Class) ~ ., #Creates GBM model with Class as the dependent variable

distribution = "bernoulli", #Sets the statistical distribution. Bernoulli is used for logistic regression

data = rbind(train, test), #Lists the data set used to train and test the model

n.trees = 500, #Sets number of decision trees used to train the model at 500

interaction.depth = 3, #Sets maximum depth of each decision tree which effects how it interacts with other trees through the additive model

n.minobsinnode = 100, #Sets the minimum number of observations within each decision tree

shrinkage = 0.01, #Sets the learning rate for each tree. Smaller learning rates require more trees

bag.fraction = 0.5, #Fraction of training set observations randomly selected to propose the next tree in the expansion. This is not random in this case as we used set.seed

train.fraction = nrow(train) / (nrow(train) + nrow(test)) #Signals training set to be used to fit the GBM model and the combination of training and test data

) #to be used for computing out-of-sample estimates of the loss function.

gbm\_norm\_iter = gbm.perf(gbm\_norm, method = "test") #Calculates the optimal number of trees needed to train the GBM which will be used to make predcitions

gbm\_norm\_pred <- predict(object = gbm\_norm, newdata = test, n.trees = gbm\_norm\_iter) #Makes predictions using the GBM model

table(as.factor(gbm\_norm\_pred > 0.50), test$Class) #Prints out Punnet Square for GBM predictions

gbm\_norm\_prob <- predict(gbm\_norm, newdata = test, type = "response") #Calculates probabilities for GBM predictions used to plot ROC curves

roc\_gbm\_norm <- roc(test$Class, gbm\_norm\_prob, plot = TRUE, #Creates and stores ROC curve

print.auc = TRUE, main = "GBM Normal Sample")

#GBM Down Sample

gbm\_down <- gbm(as.character(Class) ~ .,

distribution = "bernoulli",

data = rbind(train\_down, test),

n.trees = 500,

interaction.depth = 3,

n.minobsinnode = 100,

shrinkage = 0.01,

bag.fraction = 0.5,

train.fraction = nrow(train\_down) / (nrow(train\_down) + nrow(test))

)

gbm\_down\_iter = gbm.perf(gbm\_down, method = "test")

gbm\_down\_pred <- predict(object = gbm\_down, newdata = test, n.trees = gbm\_down\_iter)

table(as.factor(gbm\_down\_pred > 0.50), test$Class)

gbm\_down\_prob <- predict(gbm\_down, newdata = test, type = "response")

roc\_gbm\_down <- roc(test$Class, gbm\_down\_prob, plot = TRUE,

print.auc = TRUE, main = "GBM Under Sample")

#GBM Up Sample

gbm\_up <- gbm(as.character(Class) ~ .,

distribution = "bernoulli",

data = rbind(train\_up, test),

n.trees = 500,

interaction.depth = 3,

n.minobsinnode = 100,

shrinkage = 0.01,

bag.fraction = 0.5,

train.fraction = nrow(train\_up) / (nrow(train\_up) + nrow(test))

)

gbm\_up\_iter = gbm.perf(gbm\_up, method = "test")

gbm\_up\_pred <- predict(object = gbm\_up, newdata = test, n.trees = gbm\_up\_iter)

table(as.factor(gbm\_up\_pred > 0.50), test$Class)

gbm\_up\_prob <- predict(gbm\_up, newdata = test, type = "response")

roc\_gbm\_up <- roc(test$Class, gbm\_up\_prob, plot = TRUE,

print.auc = TRUE, main = "GBM Up Sample")

#GBM SMOTE Sample

gbm\_smote <- gbm(as.character(Class) ~ .,

distribution = "bernoulli",

data = rbind(train\_smote, test),

n.trees = 500,

interaction.depth = 3,

n.minobsinnode = 100,

shrinkage = 0.01,

bag.fraction = 0.5,

train.fraction = nrow(train\_smote) / (nrow(train\_smote) + nrow(test))

)

gbm\_smote\_iter = gbm.perf(gbm\_smote, method = "test")

gbm\_smote\_pred <- predict(object = gbm\_smote, newdata = test, n.trees = gbm\_smote\_iter)

table(as.factor(gbm\_smote\_pred > 0.50), test$Class)

gbm\_smote\_prob <- predict(gbm\_smote, newdata = test, type = "response")

roc\_smote\_up <- roc(test$Class, gbm\_smote\_prob, plot = TRUE,

print.auc = TRUE, main = "GBM SMOTE Sample")

#Create Data Frame for GBM Sensitivity Bar Chart:

df\_gbm <- data.frame (Sample = c("Normal","Under", "Up", "SMOTE"),

Sensitivity = c(71.43, 87.76, 86.73, 87.76)

)

order\_gbm <- c("Under", "SMOTE", "Up", "Normal") #Create variable to order bars

ggplot(data=df\_gbm, aes(x = Sample, y = Sensitivity)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = Sensitivity), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_gbm) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "GBM Sensitivity as Percentage for each Sample", #Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#########

#xGboost#

#########

#Setting Labels for xGboost:

train\_x <- train

test\_x <- test

train\_x\_down <- train\_down

train\_x\_up <- train\_up

train\_x\_smote <- train\_smote

levels(train\_x$Class) <- c("Not\_Fraud", "Fraud")

levels(test\_x$Class) <- c("Not\_Fraud", "Fraud")

levels(train\_x\_down$Class) <- c("Not\_Fraud", "Fraud")

levels(train\_x\_up$Class) <- c("Not\_Fraud", "Fraud")

levels(train\_x\_smote$Class) <- c("Not\_Fraud", "Fraud")

labels\_norm <- train\_x$Class

labels\_down <- train\_x\_down$Class

labels\_up <- train\_x\_up$Class

labels\_smote <- train\_x\_smote$Class

x\_norm <- recode(labels\_norm, 'Not\_Fraud' = 0, "Fraud" = 1)

x\_down <- recode(labels\_down, 'Not\_Fraud' = 0, "Fraud" = 1)

x\_up <- recode(labels\_up, 'Not\_Fraud' = 0, "Fraud" = 1)

x\_smote <- recode(labels\_smote, 'Not\_Fraud' = 0, "Fraud" = 1)

set.seed(120)

#Note: Number of trees was reduced to 300 to avoid over fitting.

#xGboost Normal Sample

xgboost\_norm <- xgboost(data = data.matrix(train\_x[, -30]), #Sets data set used to train the model. Outcome has to be removed to test and train xGboost models

label = x\_norm, #Sets the labels for training the model. This information contains the outcomes for each record

eta = 0.1, #Sets the learning rate for the model.

gamma = 0.1, #Tells the model the minimum loss reduction required to split decision tree nodes

max\_depth = 10, #Sets maximum depth of each decision tree

nrounds = 300, #Tells the model to use 300 weak learners for training

objective = "binary:logistic", #Informs the model that it is solving a binary classification problem

colsample\_bytree = 0.6, #Sub sample ratio of columns to be used when creating each tree

verbose = 0, #Tells the model not to print performance stats

nthread = 7, #Sets number of threads that can be used for model training (processor threads)

)

xgb\_norm\_pred <- predict(xgboost\_norm, newdata = data.matrix(test\_x[, -30])) #Predicts results for test data set

table(as.numeric(xgb\_norm\_pred > 0.50), test\_x$Class) #Calculates probabilities for model predictions

roc\_xgb\_norm <- roc(test$Class, xgb\_norm\_pred, plot = TRUE, #Creates and stores ROC curve for model

print.auc = TRUE, main = "xGboost Normal Sample")

#xGboost Down Sample

xgboost\_down <- xgboost(data = data.matrix(train\_x\_down[, -30]),

label = x\_down,

eta = 0.1,

gamma = 0.1,

max\_depth = 10,

nrounds = 300,

objective = "binary:logistic",

colsample\_bytree = 0.6,

verbose = 0,

nthread = 7,

)

xgb\_down\_pred <- predict(xgboost\_down, newdata = data.matrix(test\_x[, -30]))

table(as.numeric(xgb\_down\_pred > 0.50), test\_x$Class)

roc\_xgb\_down <- roc(test$Class, xgb\_down\_pred, plot = TRUE,

print.auc = TRUE, main = "xGboost Under Sample")

#xGboost Up Sample

xgboost\_up <- xgboost(data = data.matrix(train\_x\_up[, -30]),

label = x\_up,

eta = 0.1,

gamma = 0.1,

max\_depth = 10,

nrounds = 300,

objective = "binary:logistic",

colsample\_bytree = 0.6,

verbose = 0,

nthread = 7,

)

xgb\_up\_pred <- predict(xgboost\_up, newdata = data.matrix(test\_x[, -30]))

table(as.numeric(xgb\_up\_pred > 0.50), test\_x$Class)

roc\_xgb\_up <- roc(test$Class, xgb\_norm\_pred, plot = TRUE,

print.auc = TRUE, main = "xGboost Up Sample")

#xGboost Smote Sample

xgboost\_smote <- xgboost(data = data.matrix(train\_x\_smote[, -30]),

label = x\_smote,

eta = 0.1,

gamma = 0.1,

max\_depth = 10,

nrounds = 300,

objective = "binary:logistic",

colsample\_bytree = 0.6,

verbose = 0,

nthread = 7,

)

xgb\_smote\_pred <- predict(xgboost\_smote, newdata = data.matrix(test\_x[, -30]))

table(as.numeric(xgb\_smote\_pred > 0.50), test\_x$Class)

roc\_xgb\_smote <- roc(test$Class, xgb\_smote\_pred, plot = TRUE,

print.auc = TRUE, main = "xGboost SMOTE Sample")

#Create Data Frame for xGboost Sensitivity Bar Chart:

df\_xgb <- data.frame (Sample = c("Normal","Under", "Up", "SMOTE"),

Sensitivity = c(76.53, 91.84, 78.57, 86.73)

)

order\_xgb <- c("Under", "SMOTE", "Up", "Normal") #Create variable to order bars

ggplot(data=df\_xgb, aes(x = Sample, y = Sensitivity)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = Sensitivity), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_xgb) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "xGboost Sensitivity as Percentage for each Sample", #Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#Create Data Frame for Sensitivities of best models and sample combos Bar Chart:

df\_best <- data.frame (Name = c("Tree\_SMOTE","Forest\_Under", "GBM\_SMOTE", "xGboost\_Under"),

Sensitivity = c(81.63, 90.82, 87.76, 91.84)

)

order\_best <- c("xGboost\_Under", "Forest\_Under", "GBM\_SMOTE", "Tree\_SMOTE") #Create variable to order bars

ggplot(data=df\_best, aes(x = Name, y = Sensitivity)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = Sensitivity), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_best) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "Best Model and Sample Combination Sensitivity Comparison",#Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)

#Create Data Frame for AUC's of best models and sample combos Bar Chart:

df\_bauc <- data.frame (Name = c("Tree\_SMOTE","Forest\_Under", "GBM\_SMOTE", "xGboost\_Under"),

AUC = c(0.836, 0.974, 0.978, 0.969)

)

order\_bauc <- c("GBM\_SMOTE", "Forest\_Under", "xGboost\_Under", "Tree\_SMOTE") #Create variable to order bars

ggplot(data=df\_bauc, aes(x = Name, y = AUC)) +

geom\_bar(stat="identity") + #Plots Bar Chart

geom\_col() +

geom\_text(aes(label = AUC), vjust = 1.5, colour = "white") + #Adds data labels to each bar

scale\_x\_discrete(limits = order\_bauc) + #Orders bars by height instead of in alphabetical order of sample type

labs(

title = "Best Model and Sample Combination AUC Comparison", #Sets a plot title.

x = "Sample Type", #Changes x-axis title.

y = "Sensitivity %" #Changes y-axis title.

) +

theme(

plot.title = element\_text(size = 18, face = "bold", hjust = 0.5), #Formats plot title.

plot.subtitle = element\_text(size = 12, face = "bold", hjust = 0.5), #Formats plot subtitle.

axis.title.x = element\_text(size = 12, face = "bold", colour = "grey40"), #Formats x-axis title.

axis.title.y = element\_text(angle = 90, size = 12, face = "bold", colour = "grey40") #Formats y-axis title.

)