University College London

Department of Computer Science

Cost-sensitive Learning Approach In Motor Insurance Fraud Detection

Author:
Rik Harng Loong

Supervisor:
Dr Geoffrey Goodell

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science in Computational Finance

October 18, 2019

Abstract

Motor insurance fraud is the most common and costly type of fraud for insurance companies. Although there is a lot of research done on various kinds of fraud, relatively fewer studies have been undertaken to tackle motor insurance fraud. This paper aims to fill this gap by first discussing about different sampling methods such as undersampling, oversampling and hybrid strategy to deal with the imbalanced dataset containing 11565 accident cases. Out of these cases, only 685 of them are labelled as fraudulent. Then these methods have been implemented along with predictive models from simple logistic regression to more advanced ensemble techniques, such as bagging and boosting. The performance of the model is evaluated based on precision, recall, F_1 score, and receiver operating characteristics (ROC) curve. The investigation showed that random over-sampling appeared to be the best sampling technique among the rest as it topped most of the metrics for our models. It was also found that XGBoost performed the best among all models after hyperparameter tuning. Finally, cost-sensitive learning approach was taken to consider different costs associated with the misclassified labels. The experimental result showed that Random Forest is the best model to implement for business, incurring the lowest costs, though at the expense of lower recall.

Acknowledgements

First of all, I would like to show my sincere appreciation to my academic supervisor, Dr. Geoffrey Goodell, for his patient guidance and useful advice throughout the development of my research project. I would also like to thank my industrial supervisor, Kristian Feldborg from Vesuvio Labs, who has provided me with the dataset. Finally, I must thank immensely my colleagues Ruofan and Jiaqi, who made my time at UCL a memorable one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tackling insurance fraud remains one of the top priorities for insurance companies. The latest figures released from Cifas, the UK's leading fraud prevention service, has reported a nationwide increase of 27 % in fraudulent insurance claims between 2017 and 2018. Of the estimated increase, fraudulent claims in motor insurance accounted for 45 %, with the majority of the fraudsters between 21 and 30 years old [1]. The objective of committing motor fraud is to earn an easy money without harming others. However, people are often unaware of the serious consequences by making false insurance claims. First, it is a crime and can impact one's life and career by preventing an individual from purchasing any insurance during his/her lifetime, which may even lead to imprisonment. Second, committing fraud hurts almost every party and the UK as a whole. It takes insurers a long time to review insurance claims and policy requests in order to detect fraud, which ultimately leads to an increase in premiums and thus everyone loses out.

1.1 Motivation

Motor insurance fraud can be categorized into two types: either opportunistic or professional occurring through group cooperation [2]. The opportunistic fraud is the more prevalent form of fraud. It arises when people encounter an opportunity to exaggerate claims or provide false information during an accident to receive more money than they are entitled to. On the other hand, there are highly organized gangs who get into motor crash on purpose to claim the insurance money. The second type often results in a greater financial loss. Since fraud is inevitable, it has to be detected as soon as possible, so that necessary actions can be taken against it. For large-scale processing of insurance claims, automated fraud detection systems are required, since it is not possible for humans to check manually every claim one by one for potential of fraud. This thesis will mainly focus on detecting fraudulent claims (opportunistic fraud) using state-of-the-art machine learning techniques, whereas professional fraud is more challenging to deal with, due to the strategic planning of the fraudulent groups.

1.2 Challenges

Building a fraud detection system is not straightforward. The practitioner is required to determine which learning approach to take (e.g supervised or unsupervised learning), which algorithms to employ (e.g logistic regression, random forests, etc), which features to select, and most importantly how to address the imbalanced class problem (where legitimate cases dominate over fraud cases in our dataset). It is a key concern as most machine learning algorithms are capable of only classifying equally distributed data. In the case of imbalanced data, models tend to be biased towards the majority samples, causing poor classification of the minority samples. Overlapping features for both legitimate and fraud cases due to the limited information provided from the past claim records also pose a problem, as the classes are not easily separable. Most machine learning algorithms would underperform in this scenario. In the reality, full automated detection of motor insurance fraud is not feasible. The fraud detection system only aids the investigation process by flagging suspicious claims to humans for further assessment. This costly follow-up makes it harder to provide feedback to the system promptly to improve its performance. After all, it is difficult to get the real-world datasets in the financial industry, especially in the insurance sector, where companies rarely disclose customer data due to confidentiality issues. This is one of the major challenges in fraud detection research work.

1.3 Objectives

The primary objective of this thesis is to perform predictive analysis on the motor insurance claim dataset from Vesuvio Labs in order to speed up decision-making process in handling motor insurance claims. We first perform hypothesis testing to determine which features are significant for detecting fraud. Then, we implement various machine learning models coupled with different sampling techniques to tackle the class imbalance problem. Finally, we adopt a cost-sensitive learning approach by taking misclassification costs into consideration, and report all the results.

1.4 Outlines

In Chapter 1, we will give an overview of the motor insurance fraud, its impact on the society, challenges involved in fraud detection, and also the proposed method to build a fraud detection system.

In Chapter 2, we will discuss about the previous research works performed on the motor insurance fraud detection, and then move on to the preliminaries of machine learning.

In Chapter 3, we will describe the methodology that we use for the analysis of our datasets.

In Chapter 4, we will study the experimental comparison of several models coupled with different sampling techniques for the unbalanced data streams and the total costs incurred associated with the misclassified labels.

In Chapter 5, we will summarize our findings and discuss the implication, as well as some recommendations for future research.

Chapter 2

Background and Preliminaries

2.1 Related Work

Since motor insurance fraud has become a serious and escalating issue, the industry has invested significant resources to develop resilient fraud detection systems. The use of practical models to settle insurance claims for fraud investigation started to emerge in the 1990s [3]. In this chapter, we will look into some of the previous research activities in this field.

Weisberg and Derrig (1998) [4] have applied multiple linear regression analysis on the fraud indicators to determine the suspicion rating of the automobile insurance claims, which takes the value of 0 (None), 1-3 (Slight), 4-6 (Moderate) and 7+ (Strong). All possible regression were performed using different subset of fraud indicators. The resulting models were then ranked in terms of the value of R^2 coefficient. Belhadji, Dionne, and Tarkhani (2000) [5] have used a probit model to help claim adjusters to estimate the probability of fraud in each insurance claim studied.

In another study, Brockett, Xia, and Derrig (1995) [6] have suggested the use of neural networks on a set of fraud indicators to classify automobile claims by the level of suspicion. They reported that this technique achieves a better performance than the fraud assessment made by both insurance adjuster and investigator with respect to accuracy and consistency. Viaene, Derrig, Baesens, and Dedene (2002) [7] have demonstrated that logistic regression showed excellent predictive capability in detecting automobile insurance fraud. They also concluded that the performance difference in terms of mean area under the receiver operating characteristic (AUROC) curve between many algorithm types are minor, except for the C4.5 decision tree whose performance is rather disappointing.

In a recent study, Nian, Zhang, Tayal, Coleman, and Li (2016) [8] have proposed a motor fraud detection method by separating fraud and genuine based on the distance between claim attributes using the unsupervised spectral ranking method. Bodaghi and Teimourpour (2018) [9] have performed social network analysis for identification and analysis of organized fraudulent groups in automobile insurance. Cycle detection algorithms (using both DFS, BFS trees) were first applied to detect the suspicious groups, then the

probability of being fraudulent for these components was examined to identify fraudulent groups using maximum likelihood estimation, and their reviews were prioritized. It was stated that this method is more effective and faster compared to other existing methods for finding such groups.

What we have discussed so far is that the fraud detection model considered only aim to minimize the error rate rather than the cost of classification. In this thesis, we show that focusing on cost rather than the error of classification is a more profitable approach to the business, and this was pointed out by Dionne, Giuliano, and Picard (2003) [10]. In other words, we will compare the predictive performance of algorithms under different sampling methods, as well as the expected classification costs incurred.

2.2 Preliminaries

Before looking further into the domain-specific application, it is better to become familiar with some of the machine learning theories. This chapter will assist the reader in understanding the concept behind the proposed fraud detection model. In particular, we will mainly focus on the learning approach called Supervised Learning.

2.2.1 Supervised Learning

Supervised Learning can be defined as the process in which the model will be trained based on previously labelled data. In supervised learning, the idea is to learn the mapping function f from the input X to the output Y

$$Y = f(X) \tag{2.1}$$

as best as possible such that when an unseen new input is given to the mapping function, it is able to predict the output correctly. Moreover, supervised learning can be divided into two parts: regression and classification. In a regression task, the output variable is a real value (e.g stock price, blood pressure, etc). In a classification task, the output variable is a category, (e.g. fraud or genuine, boy or girl, etc). This thesis will only deal with classification task.

2.2.2 Classification

In machine learning, classification task is the process of predicting the class label of given data points. For instance, fraud detection can be identified as a classification task. In our case, the goal is to predict if the claim is fraudulent or genuine. Generally, there are three types of classification: binary classification, where only two class labels are involved (e.g., classifying a claim which may be fraud or genuine), multiclass classification, where there are more than two class labels (e.g classifying the flower Iris species which may be setosa or versicolor or virginica) and multilabel classification, where multiple labels may be assigned to each instance (e.g classifying a movie genre in which the class labels can be both horror

and thriller). This thesis will focus on only binary classification problem where the class label is either genuine or fraud.

2.2.3 Class Imbalance Problem

Imbalanced classes are a common problem in machine learning classification where one of the classes forms a majority and dominates other classes. The best example is the fraud detection task, as it is very likely that the number of fraudulent claims in comparison to genuine claims will be much less. Most machine learning algorithms perform poorly in the presence of imbalanced class distribution where they tend to classify wrongly the minority sample as the majority example [11]. In the next section, we aim to address some of the questions that may arise: (1) How to tackle the class imbalance problem? (2) Which machine learning algorithms should be implemented in the presence of imbalanced class distribution? (3) What evaluation metrics should we choose to measure the performance of a predictive model with regard to the imbalanced data?

2.2.4 Tackling Class Imbalance Problem

This section discusses several methods that are used to tackle the class imbalance problem. The methods can be grouped into three categories based on their approaches: namely data-level (resampling), algorithmic level (ensemble-based approach) and cost-sensitive learning approach. This thesis will deal with all three approaches mentioned above.

2.2.4.1 Resampling Method

Most of the predictive models work poorly when dealing with imbalanced data. Therefore, data preprocessing needs to be carried out before providing our input data to the model. It is performed using the so called data-level approach called resampling. In general, there are three types of resampling methods: undersampling, oversampling, and hybrid.

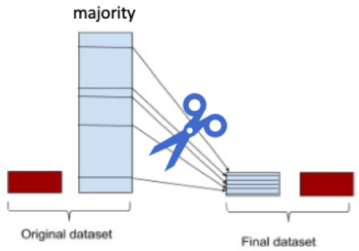


Figure 2.1: Undersampling [12]

Figure 2.2: Oversampling [13]

The undersampling method consists of removing samples from the majority class in order to make a balanced dataset, whereas oversampling method is exactly the opposite, it replicates samples from the minority class. Both methods are shown in figure 2.1 and figure 2.2 above. At last, the hybrid approach combines both undersampling and oversampling method for rebalancing purpose. Several resampling approaches will be discussed more in depth in the following subsections.

2.2.4.1.1 Random Undersampling

This method randomly eliminates samples from the majority class in order to make a balanced dataset. However, it may cause loss of useful information during the process. This method works particularly well for large amounts of training data. By reducing samples from the majority class, it also improves runtime performance and reduces data storage [14].

2.2.4.1.2 NearMiss

This method is proposed to resolve the problem of potential information loss in undersampling. The idea of this method is that it selects a set of majority class instances that have the smallest average distance to those in the minority class, in order to better represent the decision boundary. For instance, suppose that m instances of the majority class are selected and there are n instances in the minority class, then the "nearest neighbor" method will then produce $m \times n$ instances of the majority class [12].

2.2.4.1.3 Random Oversampling

This method randomly replicates samples from the minority class in order to balance the dataset. The runtime performance deteriorates due to the increased size of the original training data. Despite not having the problem of information loss, it may increase the chance of overfitting of the training data [14].

2.2.4.1.4 Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE is a well-known technique used to rebalance the dataset and it was established by Chawla [15]. This method synthesizes new minority instances by interpolating between nearest existing minority instances, which is shown in figure 2.3. The difference between random oversampling and SMOTE is that the former just increases the size of the minority samples through repetition of original samples, while the latter not only increases the size of minority samples, but also increases their variety. As a result, it diminishes the chances of data overfitting. The nearest neighbors of minority samples are randomly selected subject to the extent of oversampling required.

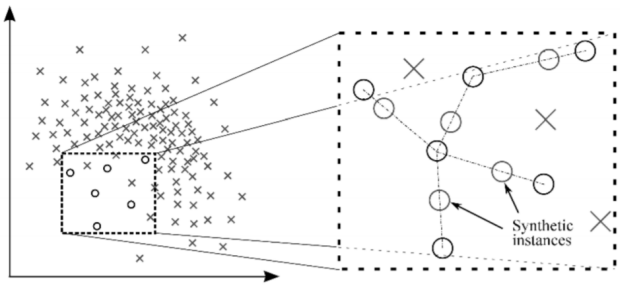


Figure 2.3: Generation of synthetic minority instances using SMOTE [14]

2.2.4.1.5 SMOTEENN (Combination of Oversampling and Undersampling)

It is used as an undersampling method or a data cleaning method after SMOTE. ENN stands for Edited Nearest Neighbor which eliminates the instances of the majority class whose prediction made by K-nearest neighbor (KNN) method is different from the majority class. Basically, it helps to eliminate the noisy examples or borderline examples, creating a smoother decision boundary between the two classes [16].

2.2.4.2 Ensemble Approach

In the previous section, the data level approach was discussed in which resampling method is used to address the class imbalance issue. In this section, we will discuss an algorithmic level approach called the ensemble approach. Ensemble approach tackles the class imbalance issue by combining various existing classification algorithms. In general, ensemble approach is a learning paradigm where multiple base models (often called "weak learners") are trained to make predictions for each instance. The final prediction is then based on the majority votes from each of the base models, as shown in figure 2.4. Typically, there are two types of ensemble approach: bagging and boosting.

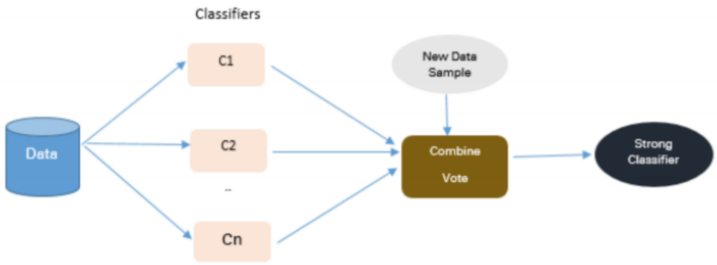


Figure 2.4: Ensemble Approach [14]

Before discussing about bagging, let us first understand an essential concept of bootstrapping that is used in bagging algorithms. Bootstrapping refers to random sampling of the training data with replacement (each element drawn from the training data is replaced back into the training data), in the way that each bootstrap sample will have different varieties as shown in figure 2.5. This helps boost the predictive performance of the model during training on these samples, as it can learn various aspects of the training data.

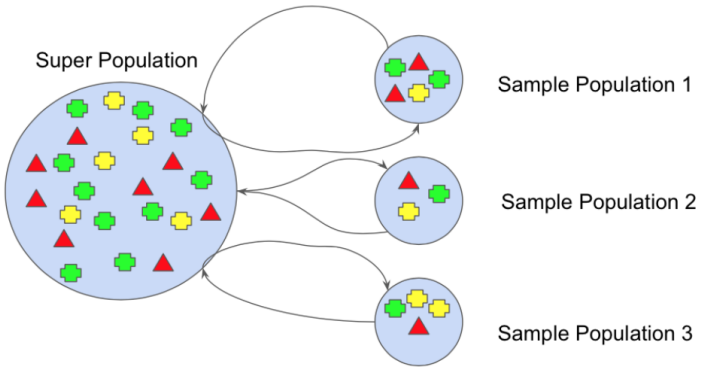


Figure 2.5: Bootstrapping [17]

2.2.4.2.1 Bagging

Bagging (Bootstrap Aggregation) is a simple yet very powerful ensemble method. This method involves bootstrapping that first generates random samples from the training data with replacement. These bootstrap samples are used to fit several independent models. The predictions made by each model are then aggregated by averaging over them in order to obtain a consensus prediction. Figure 2.6 gives a clear picture of bagging. This method can help to reduce the variance of the overall prediction and also control overfitting. Although decision trees are typically used as a base model in bagging, other types of models can also be considered. This thesis deals with the random forest algorithm as a bagging approach.

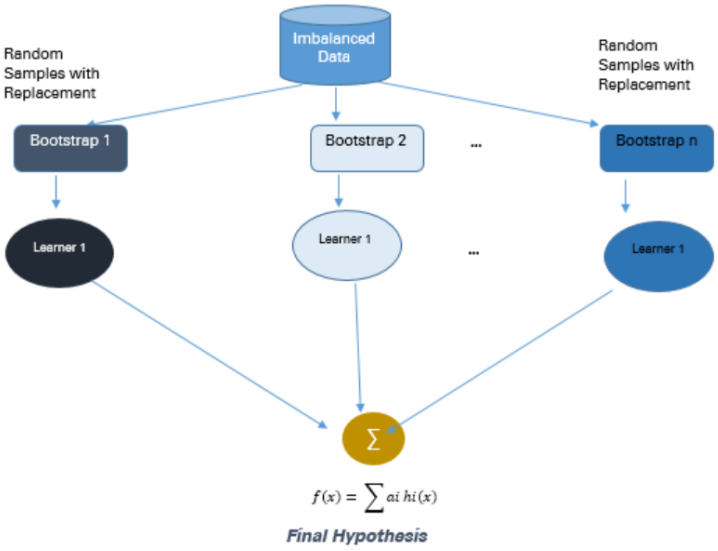


Figure 2.6: Overview of Bagging [14]

2.2.4.2.2 Boosting

Boosting is another very powerful ensemble method. Unlike bagging that had each model trained independently and the outputs are aggregated at the end, boosting is a sequential technique, in which the initial model is trained on the entire dataset and then the subsequent models are built by fitting the residuals of the initial model, thus placing more weight to those cases that were wrongly predicted by the previous model. Intuitively, it consists of creating a set of weak learners each of which might not perform well for the entire dataset, but might perform well for some parts of the dataset. Thus, each model will help to improve the performance of the ensemble. Figure 2.7 gives a clear picture of boosting. Since boosting also involves bootstrapping, it reduces variance and avoids overfitting. There are many types of boosting algorithms, but the most common ones are AdaBoost, Gradient Boosting, and XGBoost [18]. This thesis will only deal with XGBoost.

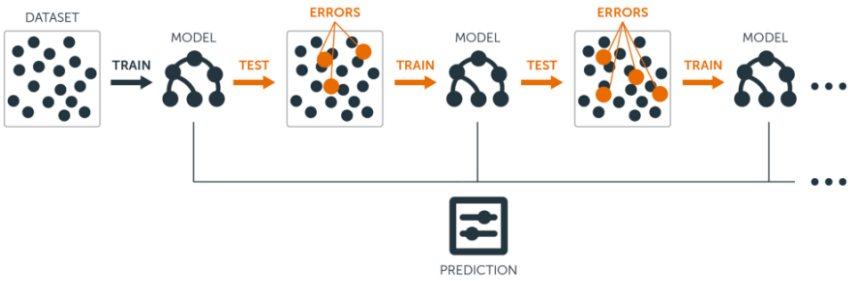


Figure 2.7: Overview of Boosting [19]

2.2.5 Predictive Models

In this section, we will discuss about the predictive models used for the analysis for our dataset. We choose three state-of-the-art classification algorithms, namely logistic regression, random forest and XGBoost.

2.2.5.1 Logistic Regression

Logistic regression is the most famous machine learning algorithms for binary classification. Unlike linear regression whose outputs are real values, logistic regression squashes the predicted real values using the sigmoid function to return a probability value between the range of 0 and 1. The equation can be written as follow:

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

In order to map the probability value to a discrete class, the threshold value is selected above which we classify values as 1 (fraud) and below which we classify values as 0 (genuine). By default, logistic regression uses a threshold of 0.5. However, this threshold can be adjusted according to our needs. Let us consider an example: we wish to classify if the student passes or fails the exam based on two independent variables *Study* and *Slept*.

$$P(Y = 1) = \phi(\beta_0 + \beta_1 Study + \beta_2 Slept) \tag{2.3}$$

where ϕ is the equation (2.2) above, β_0, β_1 and β_2 are the parameters of the logistics regression model that need to be learned during training. With a threshold of 0.5, the predicted output can be represented as follows.

$$Y = 1 \ if \ P(Y = 1) \geq 0.5 \tag{2.4}$$

$$Y = 0 \ if \ P(Y = 1) < 0.5 \tag{2.5}$$

For example, if the sigmoid function predicted 0.3 it indicates that there is only a 30% chance of passing. If our decision boundary (threshold) was 0.5, we would classify it as a fail. In contrast, if the sigmoid function predicted 0.7 which is higher than 0.5, we would classify it as a pass. Figure 2.8 shows how the sigmoid function looks like.

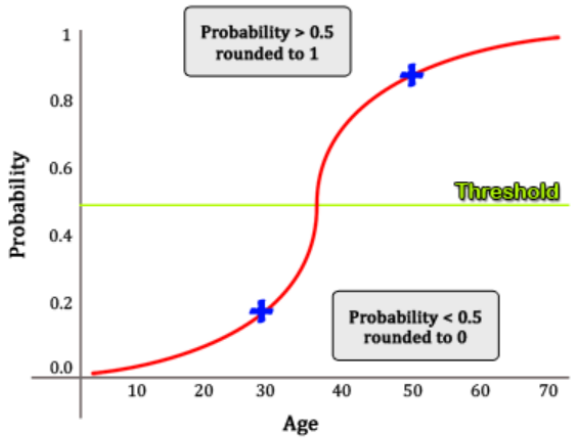


Figure 2.8: Sigmoid function with threshold of 0.5 [20]

2.2.5.2 Random Forest

Random forest is an ensemble model using decision trees as the weak learners in the bagging approach. Let us first understand the decision tree, which is the basic building block of a random forest. Decision tree is a non-parametric supervised learning algorithm used for both regression and classification. However, it is mostly used in classification task. It is composed of internal nodes each of which represents a test on a feature (e.g whether tomorrow's weather is sunny or rainy), each branch in the tree represents the outcome of the test, and finally each leaf node represents a class label [21]. Figure 2.9 gives an overview of decision tree for deciding whether to play tennis tomorrow or not.

It starts with the root node (Outlook) which consists of three possibilities: Sunny, Overcast and Rain. If tomorrow's weather is sunny, then check the humidity. If the humidity is high, we decide not to play. If the humidity is low, then we decide to play. If tomorrow's weather is overcast, we decide to play. If tomorrow's weather is rainy, check if the wind is strong or weak. If it is strong, we decide not to play; if it is weak, we decide to play. This example shows that decision tree involves a series of if-then conditions in order to reach to the final decision.

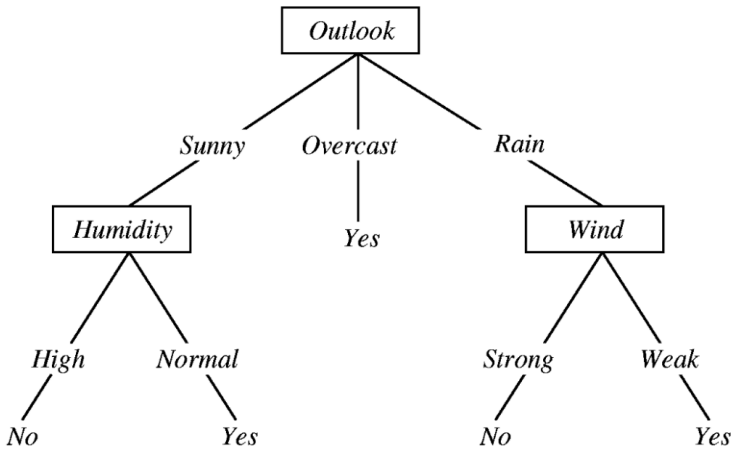


Figure 2.9: Decision tree for tennis playing tomorrow [22]

Decision tree is built by splitting the training data into smaller subsets based on some rules on the classification features. Generally, it determines the best split on each node based on the metrics such as Information Gain and Gini Impurity. Information Gain helps to determine what is the best feature to split at each node when building the tree. The split with the highest information gain will be chosen as the root node, and the operation continues till we reach the leaf node (class label). Gini Impurity measures the probability of an incorrect classification of instances, if those instances were randomly classified according to the class distribution in the dataset. The formula for Gini Impurity is given as follow [23]:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \tag{2.6}$$

where n is the number of classes, p_i is the probability of an instance being classified to a particular class. In our case, $n = 2$ (either fraud or genuine). When building the decision tree, the feature with the lowest Gini impurity will be chosen as the root node, then the similar procedure is repeated to determine the sub-nodes of the decision tree.

Although decision tree is easily interpretable, it is prone to overfitting because of the greedy approach for always selecting the best feature at each level without considering the broader assumption about the dataset. Decision tree tends to have a high variance due to its sensitivity of where and how it splits. Therefore, even a small change in the value of one input variable could give rise to a very different tree structure.

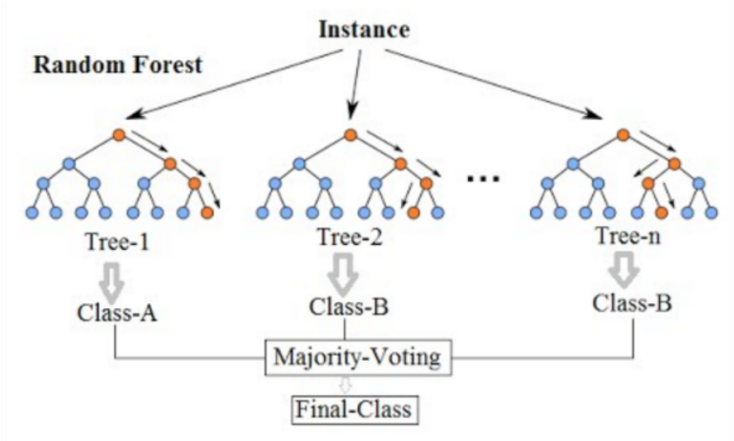


Figure 2.10: Random Forest Model [24]

Random forest is comprised of multiple decision trees. Each decision tree considers a random subset of features from the training data. This increases the overall diversity in the forest, which leads to a more robust prediction than an individual tree. In the classification task where targets are discrete class labels (e.g sunny or rainy), the random forest combines the individual decision tree predictions by taking a majority vote for the predicted class. This is illustrated in figure 2.10.

2.2.5.3 XGBoost

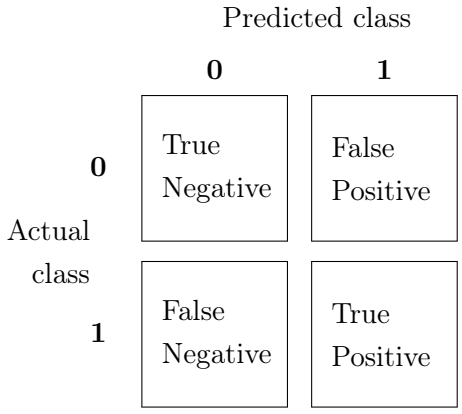
XGBoost stands for eXtreme Gradient Boosting. It is a decision-tree based ensemble method that uses a gradient boosting framework, which was proposed by Chen and Guestrin [25]. Gradient boosting is a special case of boosting, where errors are minimized in the sequential model by employing gradient descent algorithm. However, XGBoost has many advantages over the standard gradient boosting approach. XGBoost has a better regularization in order to avoid overfitting. XGBoost has the in-built capability to handle missing values. XGBoost also comes with in-built cross-validation features to easily determine the number of boosting rounds at each iteration. Moreover, it allows parallel processing designed for speed and performance. Despite its superior quality, there is a fairly large number of hyperparameters that need to be tuned in order to get the best result from XGBoost. In fact, XGBoost model performs better than other algorithms in terms of both predictive performance and processing time. This is the reason why XGBoost has been credited with winning numerous Kaggle competitions for solving either regression or classification problem [26].

2.2.6 Evaluation Metrics

After training the data with our predictive models, it is time to evaluate their performances in terms of how the model generalizes on the new unseen data. In this section, we will discuss about some evaluation metrics that are used for measuring the performance of the model in a classification task.

2.2.6.1 Confusion Matrix

The confusion matrix is the most widely used metric used to measure the performance of the classification algorithm. It is simple to understand and can be used to derive other useful metrics such as accuracy, precision, recall and F_1 score. For binary classification, it can be represented as a 2×2 matrix as shown in the following diagram:



We can think of the two possible predicted classes above as **0 : Genuine** and **1 : Fraud**, for instance in our case detecting the potential of fraud in motor insurance claims. The confusion matrix consists of four important statistics: True Negative (TN), True Positive (TP), False Negative (FN) and False Positive (FP), which are computed based on the counts of correct and incorrect classification.

True Negative (TN) is a case where the actual outcome was genuine, and the predicted outcome is also genuine.

False Negative (FN) is a case where the actual outcome was fraud, but the predicted outcome is genuine.

True Positive (TP) is a case where the actual outcome was fraud, and the predicted outcome is also fraud.

False Positive (FP) is a case where the actual outcome was genuine, but the predicted outcome is fraud.

2.2.6.2 Accuracy

Accuracy is perhaps the most simple metric used to measure the proportion of correctness in a classifier. It can be obtained using the following formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.7}$$

Let us consider an example: suppose that our dataset had 95 % genuine claims and 5 % frauds, and a classifier is built that predicted all emails as genuine. Then this classifier would achieve a 95 % accuracy, but terrible at classifying fraud. This is called an Accuracy Paradox where accuracy may not be a good metric for predictive models in the classification task. Therefore, it is important to have other metrics to evaluate the performance of the classifier.

2.2.6.3 Precision

Precision is defined as the fraction of the number of true positives over the total number of true positives and false positives, Basically, it expresses the proportion of observations that our model predicted was fraud actually were fraud.

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

2.2.6.4 Recall

Recall, also known as sensitivity, is the fraction of the number of true positives to the actual positive cases. In our case, it denotes the ability to find all the fraud cases in the dataset.

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

2.2.6.5 $\mathbf{F_1}$ Score

The F_1 score is the weighted average of precision and recall. Thus, this score takes into account both false positives and false negatives. Accuracy is a good metric to consider if false positives and false negatives cost equally. However, they are not the same in our case as will be seen in section 3.7 later. So, F_1 score is a more reliable metric than accuracy here.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.10}$$

Note that the weighted average is used rather than a simple average because it can punish extreme values. For instance, a classifier with a precision of 1.0 and a recall of 0.0 would have an average of 0.5, but F_1 score of 0 [27]. F_1 score has the range of value between 0 and 1, where 0 is considered the worst and 1 is considered the best.

2.2.6.6 Area Under Receiver Operating Characteristic (ROC) curve

A good visible way to evaluate the performance of a binary classifier is by looking at the area under ROC curve. It reveals how the performance of the classifier changes as the threshold varies. The threshold is the probability above which we classify the observation as fraud. By default, the threshold of 0.5 is typically used in the classification task. The ROC curve is a graphical plot showing the true positive rate (TPR), also known as sensitivity, against the false positive rate (FPR) at different threshold settings. FPR can also be computed as 1 - Specificity. The formulas for sensitivity and specificity are given in the equations below:

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.11}$$

$$Specificity = \frac{TN}{TN + FP} \tag{2.12}$$

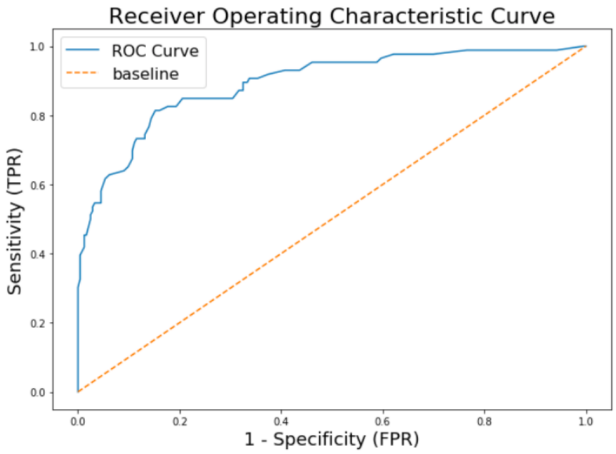


Figure 2.11: ROC Curve [28]

Let us understand the ROC curve above in steps. Suppose that the threshold is 0, all claims are classified as fraud. This implies that no fraud claims are classified as genuine and so there are no FN $\implies$ TPR (or recall) is 1. At the same time, this also means that no genuine claims are classified as genuine, so there are no TN $\implies$ FPR is also 1. This corresponds to the top right section of the curve. Suppose now that the threshold is 1, all the claims are classified as genuine. Then there are no TP and FP. This implies that both TPR and FPR are 0. This corresponds to the bottom left section of the curve. The other parts of the curve correspond to the threshold between 0 and 1, from the top-right to the bottom-left of the curve. Note that the curve approaches, but does not reach one point of the curve where TPR is 1 and FPR is 0. This is the point of perfect classification, at which no fraud claims are classified as genuine and no genuine claims are classified as fraud.

Chapter 3

Methodology

3.1 Data visualization

Datasets are an essential part in the field of machine learning. However, collecting data can be a painful procedure, especially in the financial domain such as motor insurance fraud. The dataset used in this thesis is provided by Vesuvio Labs. It contains the records of the motor insurance claims made by the insureds from year 1994 to 1996. The dataset contains 11565 observations out of which only 685 are fraudulent. Thus the dataset is very imbalanced with fraud cases only account for 5.92 % of the total observations. The imbalanced class distribution is shown in the pie chart below:

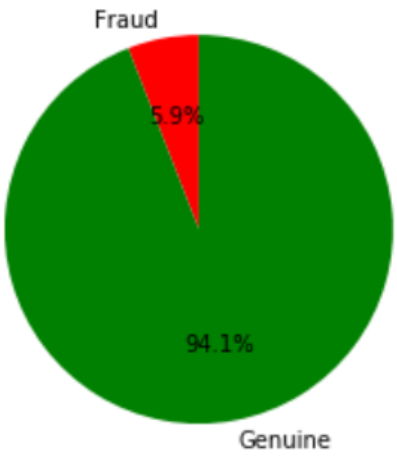


Figure 3.1: Data visualization of imbalanced class distribution

The dataset contains both numerical and categorical features with the latter being the majority. There is a total of 31 features out of which only 6 features are numerical values. There are also 11 observations with missing values in some features, and since it only forms a small percentage of the entire dataset, it is reasonable to drop them. Before diving into the implementation part, let us visualize the data in terms of correlation between the features. Correlation is a statistical term which measures the mutual relationship between two or more variables. The correlation matrix is plotted below with the light/deep colours indicate high/low correlation.

One important note is that features with high correlation tend to have the same effect on the dependent variable, which is either fraud or genuine. So, if two features have high correlation, we can drop one of the two features. In the next section, we will compare the correlation between features and drop one of the two features that have a correlation higher than 0.8.

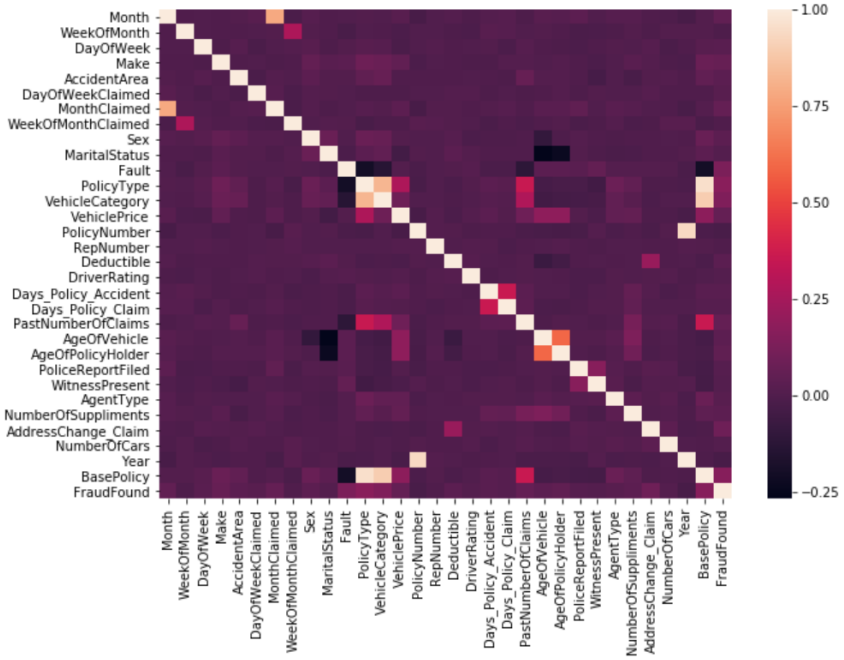


Figure 3.2: Correlation Matrix

3.2 Feature Selection

Feature selection plays an important role in building a machine learning model. It is the process where features are carefully selected such that they can contribute most to the predicted output. Having irrelevant features in the dataset can make the model learns based on them, thus reducing the overall performance of the model. In this section, we will explore two ways to select the right features in our dataset. The first way is to discard one of the two features that have correlation higher than 0.8 as discussed previously. Based on the correlation matrix above, we discard the feature **"MonthClaimed"** which highly correlates with other feature **"Month"**. On the other hand, we also discard **"BasePolicy"** and **"VehicleCategory"** as both of these features highly correlates with **"PolicyType"**. In fact, the feature **"PolicyType"** indicates the combination of both base policy and the vehicle category. Lastly, we discard **"PolicyNumber"** which highly correlates with **"Year"**.

Now we are left only with features that have correlation less than 0.8. Hypothesis testing is then performed to identify which features are significant in detecting fraud. This would be the second way to select the right features in our dataset. We will first consider Independent Samples T-Test which determines whether there is a statistically significant difference between one continuous variable and our target variable, either fraud

or genuine. We then consider Pearson's Chi-square Test which determines whether there is any association between two categorical variables being considered. The null hypothesis in both tests is that there is no relationship between the independent variable and target variable. If the p-value is less than 0.05 (95 % confidence interval in our case), the null hypothesis is rejected, indicating the continuous/categorical variable is associated with fraud and vice-versa. The results are as follows:

	Features	p-value
1	Month	0.0013
2	Make	0.0002
3	AccidentArea	0.0009
4	Sex	0.0003
5	Fault	5.8968e-45
6	VehicleCategory	1.7461e-49
7	BasePolicy	1.8325e-66
8	VehiclePrice	3.5901e-08
9	Deductible	1.1073e-12
10	PastNumberOfClaims	2.1991e-09
11	AgeOfVehicle	0.0278
12	AgeOfPolicyHolder	0.0044
13	AgentType	0.0066
14	NumberOfSuppliments	0.0027
15	AddressChange_Claim	1.9425e-19
16	Year	0.0053

Table 3.1: Significant predictors

	Features	p-value
1	WeekOfMonth	0.5252
2	DayOfWeek	0.2227
3	DayOfWeekClaimed	0.6675
4	WeekOfMonthClaimed	0.2344
5	MaritalStatus	0.7580
6	RepNumber	0.8809
7	DriverRating	0.5110
8	Days_Policy_Accident	0.1023
9	Days_Policy_Claim	0.4372
10	WitnessPresent	0.4303
11	NumberOfCars	0.9308
12	PoliceReportFiled	0.1549

Table 3.2: Insignificant predictors

From the tables above, we concluded that out of the 28 predictors, 17 predictors are identified as significant and will be used for training our models. In order to flag the fraudulent claims, we need to further look into the significant predictors and better understand why there is a significant difference between fraudulent and genuine claims for these predictors. The summary below shows the key findings from the analysis of some of our significant predictors:

Fraudulent Claim Demographic Characteristics

- Year: Fraud cases tend to happen within the first two years (1994 and 1995) rather than later.
- Month: Fraud cases tend to happen in January, March, June, July, October or December.
- AgeOfPolicyHolder: Fraudsters' age tend to be between 30 and 50.
- NumberOfSuppliments: Fraudulent claims are more likely to have no supplements.
- Deductible: Claims which have deductibles amounted to 400 are more likely to be fraudulent.

- AddressChange_Claim: Fraudsters tend to have their address changed more frequently.
- AccidentArea: Fraudulent claims mostly occur in urban areas.
- Sex: Males tend to commit fraud more often than their female counterparts.
- Fault: Policy holders are more likely to commit fraud.
- PastNumberOfClaims: Fraudsters tend to have the number of past claims between 2 and 4.

Fraudulent Claims Vehicle Type

- Make: Vehicles made by Toyota, Honda and Pontiac tend to have the most fraudulent claims.
- VehicleCategory: Sedan vehicles are more commonly to be involved in fraudulent claims.
- VehiclePrice: Vehicles of low price (below 30,000) are more commonly to be involved in fraudulent claims.
- AgeOfVehicle: Vehicles of more than 5 years are more commonly to be involved in fraudulent claims.

Fraudulent Claims Policy Type

- BasePolicy: Fraudulent claims are potentially to be either Collision or All Perlis.
- AgentType: Fraudulent claims are potentially to be handled by an external agent.

3.3 Data splitting

Before proceeding with the resampling technique, we split our data into 80 % training set and 20 % testing set. This is to ensure that the model is tested on the original testing set and not on the testing set created by the resampling technique. The training set is then used for resampling, hyperparameter tuning, and model evaluation which will be discussed in the sections 3.4, 3.5 and 3.6 respectively. The performance of the trained model is evaluated on the testing set. While splitting the data, the random seed is specified (any random number) just to ensure the same data splits every time when the program executed.

3.4 Data resampling

As discussed earlier, the dataset is very imbalanced. The number of genuine claims outnumbers the number of fraudulent claims. If we fit our model to this dataset, the model tends to be biased towards the genuine claims, causing a poor classification of the fraudulent claims on the unseen data. To address this problem, resampling techniques are used such as random undersampling, NearMiss, random oversampling and SMOTE in order to balance our training data.

3.5 Hyperparameter tuning using 5-fold cross validation

Hyperparameters are model-specific features that are 'fixed' or need to be set manually by the practitioner even before training or testing the model on the data. The value of hyperparameter cannot be estimated from the data during the training process. It should not be confused with the model parameter whose value can be estimated during the training process. For instance, the model parameters of a simple linear regression:

$$y = \beta_0 x_1 + \beta_1 x_2 + \beta_2 x_3 + ... \tag{3.1}$$

are the βs (coefficient of x) that can be estimated after fitting our model to the training data. The right set of hyperparameters needs to be searched in order to yield the best model performance. We will optimize the hyperparameter of our models using the **RandomizedSearchCV**, an abbreviation for randomized search function with cross-validation, which is contained inside the python package called **scikit-learn** [29]. We use 5-fold cross validation here which is significant enough for robustness of our model performance metric. Note that randomized search is used here instead of grid search. Although both methods search the same exact scope of parameters and the outcome in their parameter settings are fairly similar, the runtime for randomized search is significantly lower, but at the expense of slightly worse performance compared to that of grid search [30].

3.5.1 Hyperparameter tuning in Logistic Regression

In the logistic regression, the coefficient C is an important hyperparameter that plays a role in the following objective function that we try to minimize [31]:

$$J(X, y, w) = \mathcal{L} + \frac{1}{C}\|w\|^2 \quad \text{and} \quad \lambda = \frac{1}{C} \tag{3.2}$$

where $\mathcal{L}$ is the logistic loss function and C is the inverse regularization parameter. For small value of C, the regularization strength (λ) increases which penalizes the weights $\|w\|$ by making them small. Therefore, it creates a simpler model but may also underfit the data. For large value of C, the regularization strength (λ) decreases which creates a more complicated model that may overfit the data. As a result, **RandomizedSearchCV** was implemented on the resampled training data with an initial list of C values in order to identify the optimal C for the logistic regression model.

3.5.2 Hyperparameter tuning in Random Forest

We performed randomized search with 5-fold cross validation on the resampled training data to search for the best hyperparameters:

n_estimators denotes the number of trees to be used in the forest.

max_features denotes the maximum number of features to consider while looking for the best split of a node.

max_depth denotes the maximum depth of the tree.

criterion denotes the metric of the quality of a split, takes either "gini" or "entropy".

3.5.3 Hyperparameter tuning in XGBoost

There are many important hyperparameters in the XGBoost model. We performed randomized search with 5-fold cross validation on the resampled training data to search for the best values of the following hyperparameters:

min_child_weight denotes the minimum number of samples that can be represented by a node in order to split further. If the sample size is fewer than this number at a particular node, that node becomes a leaf and no longer be splitted. This can reduce the model complexity and prevent overfitting.

max_depth denotes the maximum depth of the tree. Increasing the parameter value makes the tree becomes more complex and more prone to overfitting.

gamma denotes the minimum loss reduction required to make a further split at the tree node.

subsample denotes the fraction of training data to be randomly sampled prior to constructing trees, and this will prevent overfitting.

colsample_bytree denotes the fraction of columns to be randomly sampled when constructing each tree.

alpha denotes the regularization parameter which controls overfitting.

3.6 Performance Evaluation of Models

After tuning the hyperparameters for each model, we evaluated the performance of the models based on precision, recall and F_1 score. Note that the accuracy is not used as a metric here as it often gives a misleading result in the presence of imbalanced class distribution as discussed earlier [27]. Since this is the fraud detection task, we wish our model can detect as many fraudulent claims as possible, implying that the model recall to be as high as possible. However, we should not neglect the model precision, as we do not want our model to always predict the claim as fraudulent even if it is not. Hence the F_1 score is a good metric to consider here as it takes both previous metrics into account. The area under ROC curve (AUC) is also an essential metric to be computed in order to visualize the performance of the model at different thresholds. For instance, if the AUC is 0.8, it means there is a 80 % chance that the model is able to differentiate between fraud and genuine claims. Basically, the higher the AUC score, the better the performance of the model.

3.7 Cost-sensitive Learning

Insurance fraud detection can be viewed as a cost-sensitive problem, as the cost associated with a false positive is usually different from that of a false negative.

- **False positives:** Failing to detect genuine claims and predict those as fraudulent instead, there would be an administrative cost for follow-up and also the legal cost incurred by the insurance company.
- **False negatives:** Failing to detect fraudulent claims, the full claim amount is lost to the fraudsters.

		Predicted class	
		0	**1**
Actual class	**0**	0	$C_{FP_i} = C_a + C_l$
	1	$C_{FN_i} = Amt_i$	$C_{TP_i} = C_a$

Suppose that **0 : Genuine** and **1 : Fraud**. From the cost matrix above, we can define the expected cost function as follows [32]:

$$Cost = \sum_{i=1}^{n} t_i(p_i C_a + (1 - p_i)Amt_i) + (1 - t_i)p_i(C_a + C_l) \tag{3.3}$$

t_i denotes the actual class which takes value of either 0 or 1
p_i denotes the predicted probability of fraud
C_a denotes the administrative cost of dealing with fraud
C_l denotes the legal cost of predicting fraud, but in fact the claim is genuine
Amt_i denotes the claim amount lost

Note that the claim amount varies quite significantly across each claim. However, due to the lack of information about both the administrative cost and legal cost in insurance claims, we assumed that these costs are held constant. In this case, we set $C_a = 50$ and $C_l = 1000$. In reality, these costs can vary depending upon the claim conditions. The amount of claim eventually paid by the insurance company to the insured under genuine cases is ignored here, as we are more concerned about the amount loss due to the fraudulent claims. That being said, the cost of FN (Amt_i) is often higher than that of FP ($C_a + C_l$). In the next section, the expected costs incurred by each model will be presented together with their respective precision and recall.

Chapter 4

Evaluation

4.1 Model Performance under Different Sampling Techniques

The result below shows our model performance on the testing set after hyperparameter tuning using 5-fold cross validation on the training set with different sampling techniques. The sampling type 'base' denotes no sampling is performed, whereas weighted base denotes the class weighting that is supported by our model for balancing classes. The 'balanced' mode automatically adjusts weights that are inversely proportional to the class frequencies.

Base

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.0290	0.6667	0.0148	0.5072	[[2175,1],[133,2]]
Random Forest	0.0292	1.0000	0.0148	0.5074	[[2176,0],[133,2]]
XGBoost	0.2222	0.9444	0.1260	0.5627	[[2175,1],[118,17]]

Table 4.1: Base Performance

SMOTE

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2409	0.1390	0.9037	0.7781	[[1420,756],[13,122]]
Random Forest	0.2171	0.1953	0.2444	0.5910	[[2040,136],[102,33]]
XGBoost	0.1096	0.7273	0.0593	0.5289	[[2173,3],[127,8]]

Table 4.2: SMOTE Performance

RandomOverSampler

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2378	0.1366	0.9190	0.7791	[[1392,784],[11,124]]
Random Forest	0.2636	0.1535	0.9333	0.8070	[[1481,695],[9,126]]
XGBoost	0.4389	0.2884	0.9185	0.8890	[[1870,306],[11,124]]

Table 4.3: RandomOverSampler Performance

NearMiss

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2351	0.1348	0.9185	0.7764	[[1380,796],[11,124]]
Random Forest	0.2306	0.1309	0.9704	0.7853	[[1306,870],[4,131]]
XGBoost	0.2594	0.1497	0.9704	0.8142	[[1432,744],[4,131]]

Table 4.4: NearMiss Performance

RandomUnderSampler

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2285	0.1310	0.8963	0.7636	[[1373,803],[14,121]]
Random Forest	0.2314	0.1310	0.9926	0.7920	[[1287,889],[1,134]]
XGBoost	0.3108	0.1870	0.9185	0.8354	[[1637,539],[11,124]]

Table 4.5: RandomUnderSampler Performance

SMOTEENN

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2367	0.1387	0.8074	0.7481	[[1499,677],[26,109]]
Random Forest	0.2746	0.2112	0.3926	0.6508	[[1978,198],[82,53]]
XGBoost	0.2121	0.3333	0.1556	0.5681	[[2134,42],[114,21]]

Table 4.6: SMOTEENN Performance

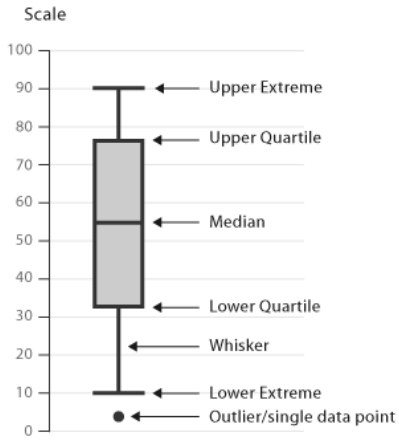
weighted base

Model	F_1	precision	recall	AUC	confusion_matrix
Logistic Regression	0.2361	0.1350	0.9407	0.7833	[[1362,814],[8,127]]
Random Forest	0.2542	0.1466	0.9556	0.8052	[[1425,751],[6,129]]
XGBoost	0.2772	0.1628	0.9333	0.8178	[[1528,648],[9,126]]

Table 4.7: weighted base Performance

It is expected that all of our models have the lowest AUC under no-sampling scenario, as they often classify the claims as genuine, although possessed high precision, but failed to detect most of the fraudulent claims. Class weighting seemed to produce comparable results with other sampling techniques for all of the three models. RandomOverSampler appeared to be the best sampling technique among the rest as it topped most of the metrics for our models.

Logistic regression seemed to produce a more consistent result, while the tree classifiers (RF and XGBoost) suffered from both SMOTE and SMOTEENN sampling techniques. However, it is clear that XGBoost has the best overall performance compared to other models, as indicated by the highest score achieved in all of the metrics above other than SMOTE and SMOTEENN. It can also be visualized by the Box and Whisker Plots (or Box Plots) in Figure 4.1 below showing the comparison of performance of several models on the entire dataset using 5-fold cross validation. The Box Plot is a convenient way of displaying the distribution of the estimate of AUC scores through their quartiles. Below also shows the anatomy of the Box Plot:



Model abbreviation for Figure 4.1

LR : Logistic Regression

KNN : K-nearest neighbor

SVC : Support Vector Classifier

DT : Decision Tree

BRF : Balanced Random Forest

XGB : XGBoost

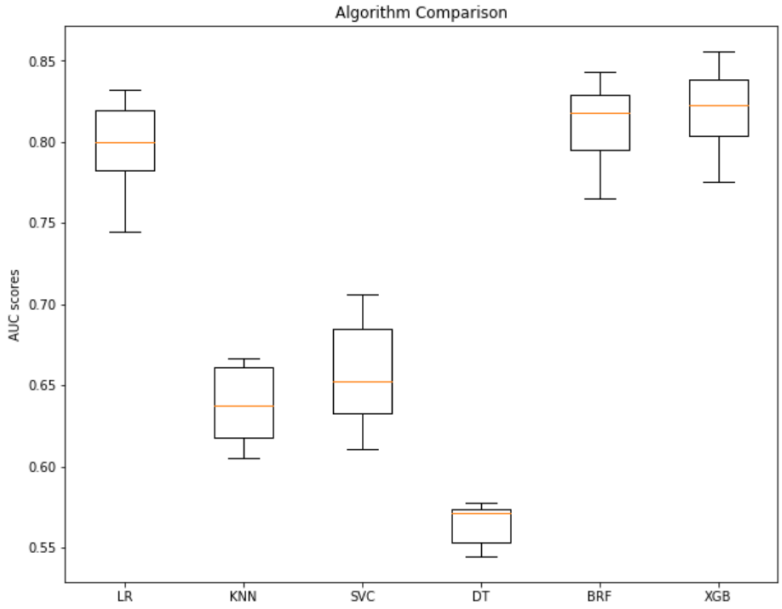


Figure 4.1: Comparison of models with class weighting on the entire dataset

4.2 Automated Machine Learning (AutoML)

The dataset was also fitted on the Microsoft Azure AutoML Service using 5-fold cross validation. In general, AutoML allows one to automate the tasks of feature engineering, model selection and even hyperparameter tuning, thus saving an enormous amount of time it takes to build a machine learning model. The approach is based on the combination of ideas from collaborative filtering and Bayesian optimization to search a large space of possible machine learning pipelines intellectually and efficiently [33]. The highest AUC score attained is 0.8547 for a reasonably 50 iterations (the dots below represent each iteration), again with XGBoost came up on top.

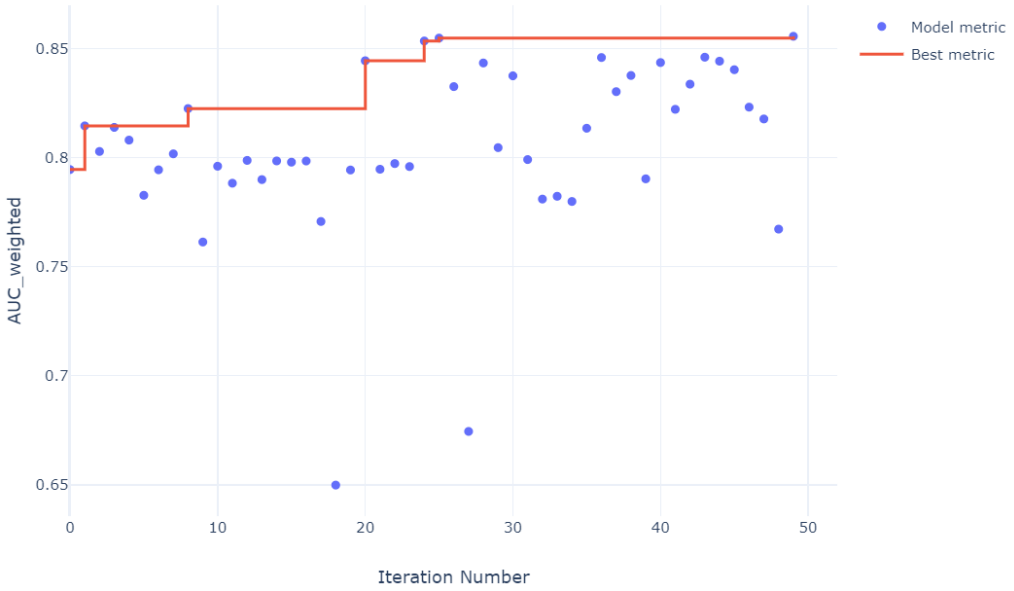


Figure 4.2: AUC score attained using AutoML for 50 iterations

4.3 Expected Cost Incurred

Up until now, the classification algorithms that we have considered only aimed to minimize the error rate: the proportion of incorrect prediction of class labels. However, we have neglected the consequence of different types of misclassification errors. In other words, we implicitly assumed that all missclassification errors cost equally. In many real-world applications, like the problem we face in motor insurance fraud, this assumption is not true. In our case, the consequence of predicting genuine given a fraudulent claim is definitely more severe than that of predicting fraud given a genuine claim.

In this section, we compare our models in terms of the expected cost incurred based on the cost matrix mentioned earlier. We then identify the threshold and construct the confusion matrix for each model in which case the expected cost is minimized on each of the testing set. Recall that the threshold is the value above which we classify the claim as fraudulent, and below which we classify it as genuine. The figures and tables below show the respective results:

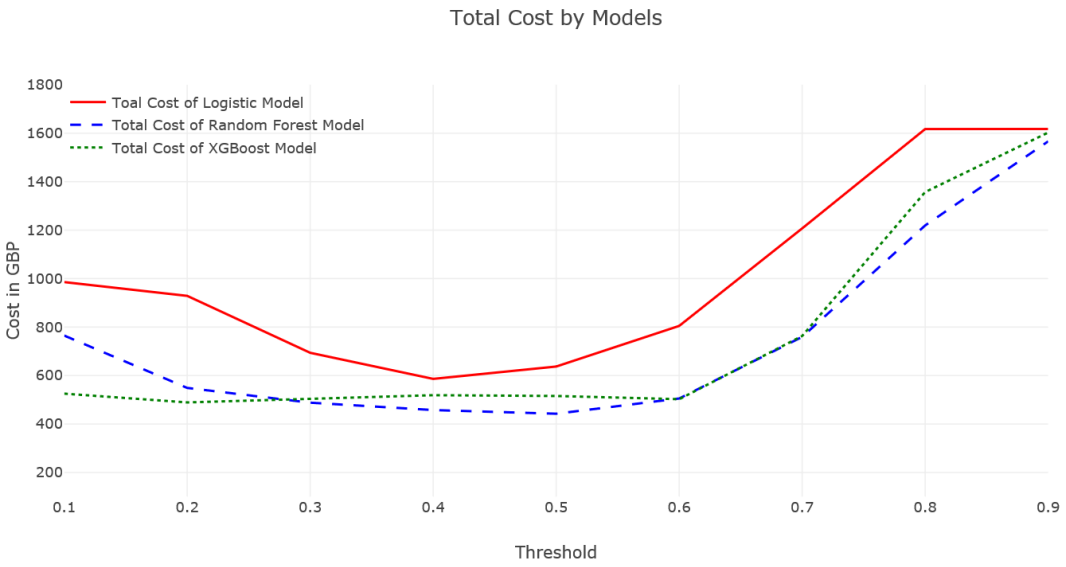


Figure 4.3: Expected cost incurred on 1st testing set

Model	threshold	recall	precision	cost
Logistic Regression	0.4	0.9463	0.1095	586.1712
Random Forest	0.5	0.9128	0.1598	442.4498
XGBoost	0.2	0.9396	0.1313	489.1902

Table 4.8: Model threshold at which total cost is minimized on 1st testing set

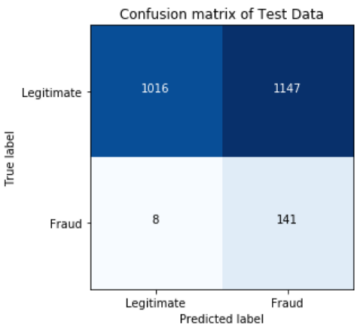


Figure 4.4: Confusion matrix of LR at threshold 0.4

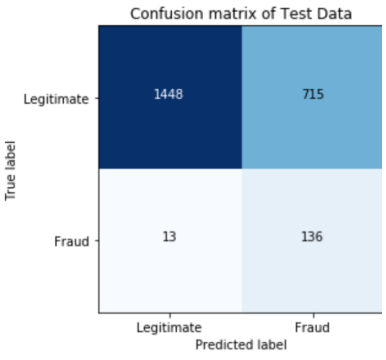


Figure 4.5: Confusion matrix of RF at threshold 0.5

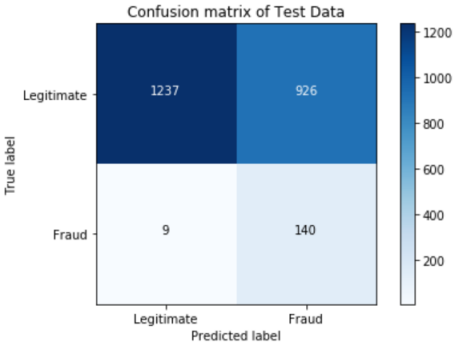


Figure 4.6: Confusion matrix of XGB at threshold 0.2

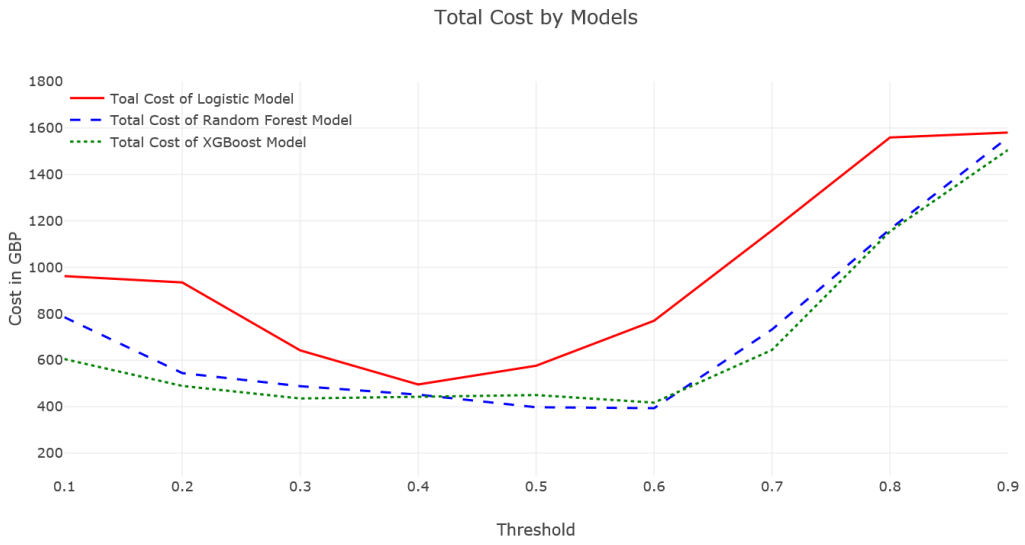


Figure 4.7: Expected cost incurred on 2nd testing set

Model	threshold	recall	precision	cost
Logistic Regression	0.4	0.9779	0.1172	495.6784
Random Forest	0.6	0.8750	0.1922	393.7789
XGBoost	0.3	0.9706	0.1320	435.4947

Table 4.9: Model threshold at which total cost is minimized on 2nd testing set

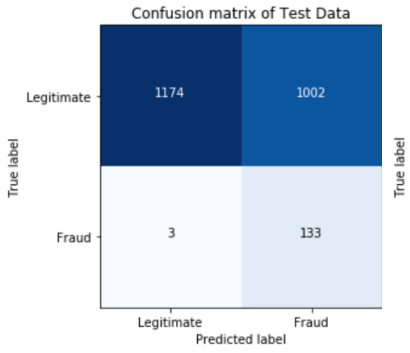


Figure 4.8: Confusion matrix of LR at threshold 0.4

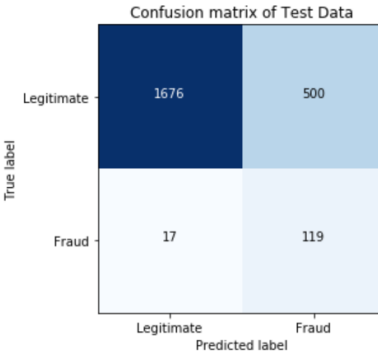


Figure 4.9: Confusion matrix of RF at threshold 0.6

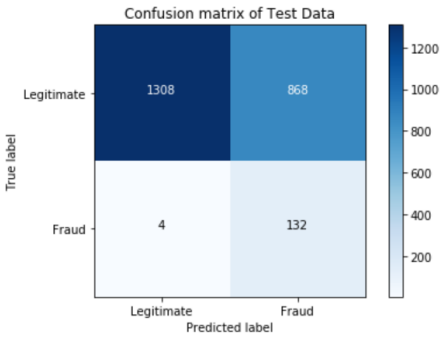


Figure 4.10: Confusion matrix of XGB at threshold 0.3

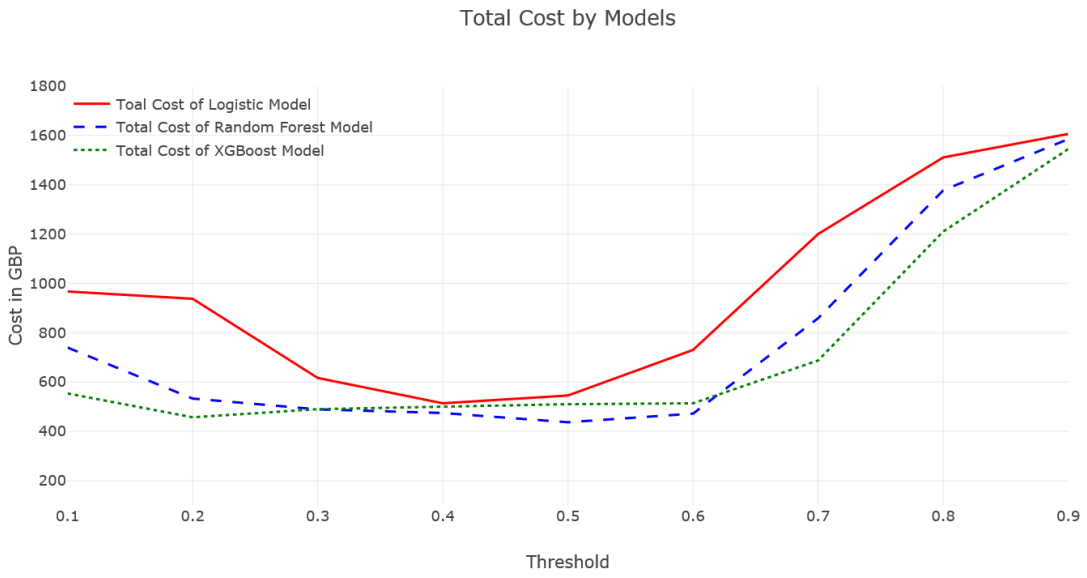


Figure 4.11: Expected cost incurred on 3rd testing set

Model	threshold	recall	precision	cost
Logistic Regression	0.4	0.9353	0.1212	513.8100
Random Forest	0.5	0.8921	0.1435	436.9660
XGBoost	0.2	0.9424	0.1309	457.0751

Table 4.10: Model threshold at which total cost is minimized on 3rd testing set

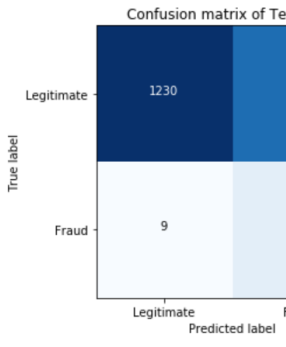


Figure 4.12: Confusion matrix of LR at threshold 0.4

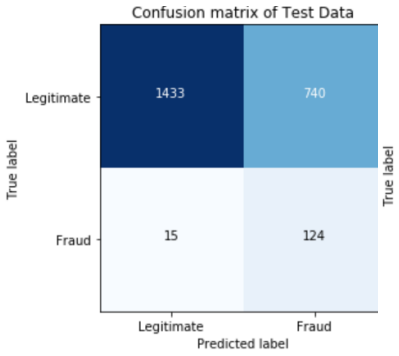


Figure 4.13: Confusion matrix of RF at threshold 0.5

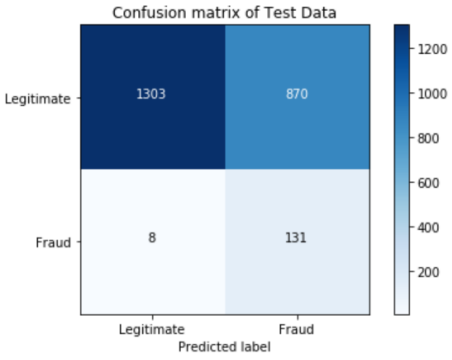


Figure 4.14: Confusion matrix of XGB at threshold 0.2

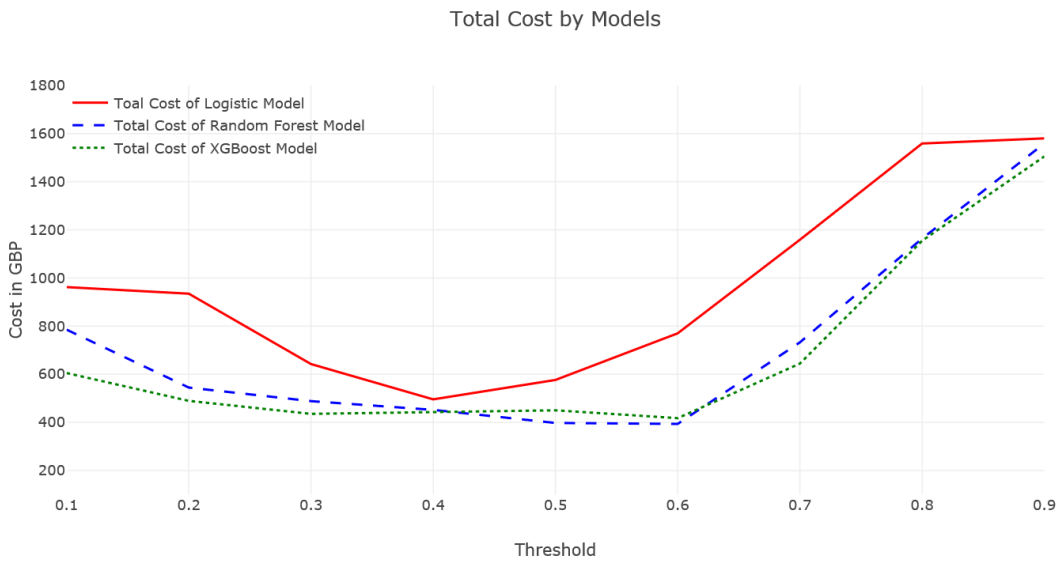


Figure 4.15: Expected cost incurred on 4th testing set

Model	threshold	recall	precision	cost
Logistic Regression	0.4	0.9558	0.0924	564.6765
Random Forest	0.5	0.9381	0.1202	460.5238
XGBoost	0.3	0.9469	0.1069	489.5058

Table 4.11: Model threshold at which total cost is minimized on 4th testing set

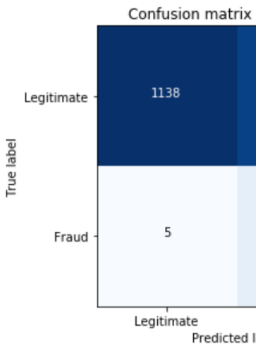


Figure 4.16: Confusion matrix of LR at threshold 0.4

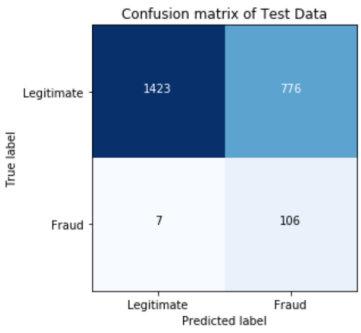


Figure 4.17: Confusion matrix of RF at threshold 0.5

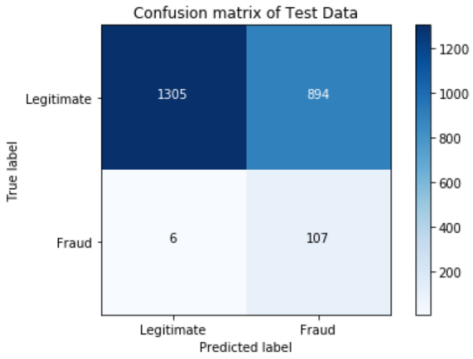


Figure 4.18: Confusion matrix of XGB at threshold 0.3

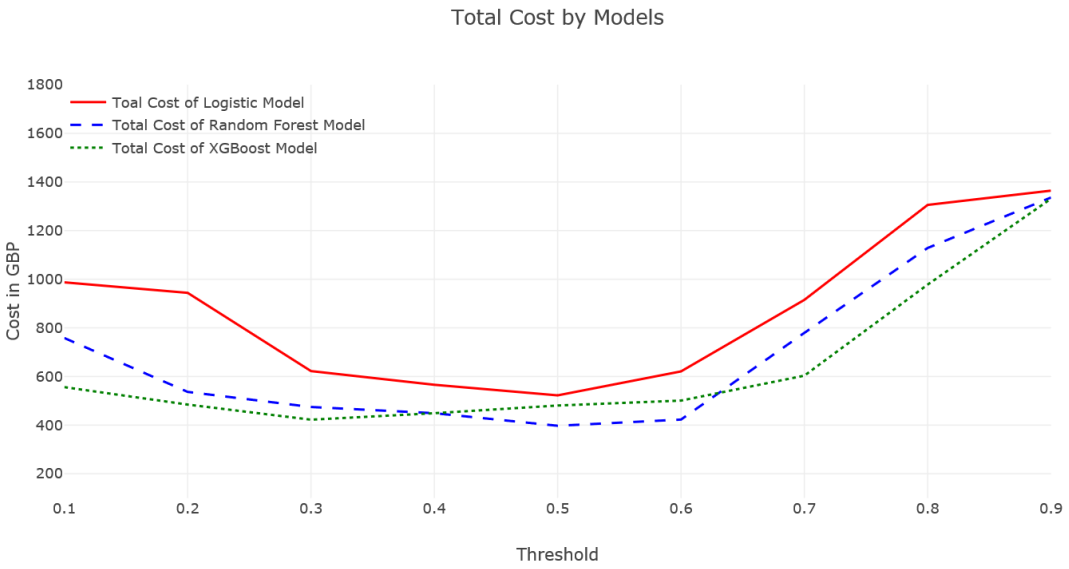


Figure 4.19: Expected cost incurred on 5th testing set

Model	threshold	recall	precision	cost
Logistic Regression	0.5	0.7174	0.1358	522.2884
Random Forest	0.5	0.9275	0.1435	397.2605
XGBoost	0.3	0.9493	0.1366	422.3030

Table 4.12: Model threshold at which total cost is minimized on 5th testing set

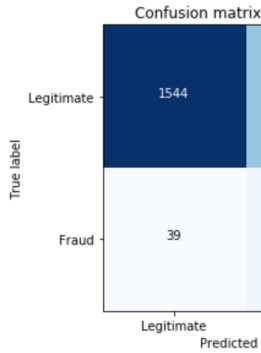


Figure 4.20: Confusion matrix of LR at threshold 0.5

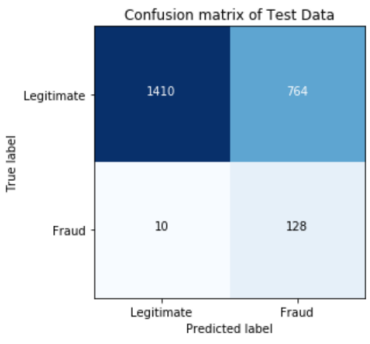


Figure 4.21: Confusion matrix of RF at threshold 0.5

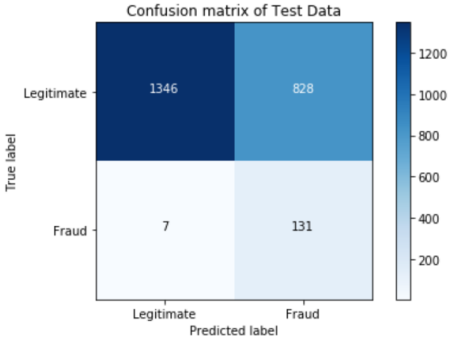


Figure 4.22: Confusion matrix of XGB at threshold 0.3

The expected cost graphs above clearly show that both Random Forest and XGBoost incur less cost than logistic regression on each testing set at all different thresholds. From the tables above, it is clear that logistic regression has a high recall but low precision on most of the testing sets, and is able to detect roughly the same number of fraudulent claims as XGBoost (except the 5th testing set), as indicated by the confusion matrix. However, logistic regression often failed to classify a large number of genuine claims correctly (except

the 5th testing set), predicting those claims as fraud instead, thus incurred both very high administrative and legal costs. On the other hand, logistic regression could detect the genuine claims pretty well on the 5th testing set, but failed to correctly classify the most number of fraudulent claims.

Despite XGBoost having a higher recall than Random Forest on each testing set, it incurred a slightly higher cost. This is due to the lower precision of XGBoost, failing to correctly classify a large number of genuine claims compared to random forest, thus incurred both the administrative and legal costs. These combined costs are higher than the claim amount loss due to only a few misclassified fraud cases in Random Forest, compared to that of XGBoost. After all, the best threshold to set for each model is depicted as follows:

Model	best threshold
Logistic Regression	0.4 - 0.5
Random Forest	0.5 - 0.6
XGBoost	0.2 - 0.3

Table 4.13: Best model threshold

4.4 Business Use

The range of thresholds found in Table 4.13 can be used as a reference when choosing a particular model for the future prediction of the motor insurance claims in Vesuvio Labs. Ultimately, random forest would be the best model to consider alongside with its best range of threshold between 0.4 and 0.5, since it incurred the lowest expected cost compared to other models. One important note is that we have assumed that under the genuine cases in which our model predicted as fraud, those claims would be flagged and sent to the experts for further investigation, and the claim amount would be ultimately paid by the insurance company to the insured, which makes sense in reality.

Chapter 5

Discussion and Future work

5.1 Summary of Results

In this thesis, machine learning techniques were used to classify the fraudulent claims in the presence of highly imbalanced class distribution. First, we implemented a data-level approach which includes various resampling techniques such as random undersampling, NearMiss, random oversampling, SMOTE and SMOTEENN. Besides that, we also implemented an algorithmic approach where bagging and boosting methods are used to tackle the class imbalanced problem. For this, Random Forest was selected as a bagging method and XGBoost as a boosting method. These models were then compared with the well-known classification algorithm, logistic regression with or without resampling techniques. The experimental result showed that XGBoost performed better than other models, and this was also backed up by the model selection process in Microsoft Azure AutoML service. In addition, cost-sensitive learning approach was taken to take into account different misclassification costs. It was found that Random Forest incurred a slightly lower cost than XGBoost, though at the expense of lower recall but higher precision. The best threshold to set for each model is also reported and can be used as a reference for the future prediction.

5.2 Discussion

One should also take into account the trade-off between expected cost and recall or precision. Although the recall of XGBoost is high, it incurred a higher cost than Random Forest due to its lower model precision. The low model precision might tarnish the reputation of an insurance company because of the delay of paying out the claim amount. On the other hand, despite Random Forest incurring a slightly lower cost, the low model recall might lead to an increase in the fraud cases in the future due to the undetected ones in the past. Consequently, it may lead to an even higher cost in the log run. However, the practitioner must opts for the latter approach, because one can always find ways to improve the fraud detection model whereas it is difficult to restore a company reputation once it becomes bad.

5.3 Future work

For future work, deep learning model can be implemented to tackle the class imbalanced problem provided that more data is available, to see whether it will outperform the models considered here in terms of both expected costs incurred and model recall or precision. Given more time, one should also try using Grid Search rather than Randomized Search for model hyperparameter exploration in order to further improve the model recall or precision and thus lower down the expected cost, though the difference is likely to be insignificant. Moreover, obtaining labelled datasets in fraud detection task is laborious and expensive, if not practically infeasible. Due to the confidentiality issue, there is even a shortage of unlabelled datasets. Thus, unsupervised learning technique such as spectral ranking method can be implemented to detect anomaly (fraud cases) without considering any labels. This method is found to surpass many existing outlier-based fraud detection methods. Lastly, social networks can be used such as cycle detection algorithms (using both DFS, BFS trees) in order to identify and analyze the organized fraudulent groups in automobile insurance.

Bibliography

[1] Cifas.org.uk. (2019). Cifas members report a 27% rise in false insurance claims across the UK in the past year, with spikes in household and motor insurance| Cifas. [online] Available at: https://www.cifas.org.uk/newsroom/household-motor-insurance [Accessed 20 Aug. 2019].

[2] Combating Insurance Claims Fraud How to Recognize and Reduce Opportunistic and Organized Claims Fraud. (n.d.). [online] Available at: https://support.sas.com/resources/papers/proceedings12/105573_0212.pdf [Accessed 20 Aug. 2019].

[3] Derrig, R.A., 2002. Insurance fraud. Journal of Risk and Insurance, 69(3), pp.271-287.

[4] Weisberg, H.I. and Derrig, R.A., 1998. Quantitative methods for detecting fraudulent automobile bodily injury claims. Risques, 35(July–September), pp.75-99.

[5] Belhadji, E.B., Dionne, G. and Tarkhani, F., 2000. A model for the detection of insurance fraud. The Geneva Papers on Risk and Insurance-Issues and Practice, 25(4), pp.517-538.

[6] Brockett, P.L., Xia, X. and Derrig, R.A., 1998. Using Kohonen's self-organizing feature map to uncover automobile bodily injury claims fraud. Journal of Risk and Insurance, pp.245-274.

[7] Viaene, S., Derrig, R.A., Baesens, B. and Dedene, G., 2002. A comparison of state-of-the-art classification techniques for expert automobile insurance claim fraud detection. Journal of Risk and Insurance, 69(3), pp.373-421.

[8] Nian, K., Zhang, H., Tayal, A., Coleman, T. and Li, Y., 2016. Auto insurance fraud detection using unsupervised spectral ranking for anomaly. The Journal of Finance and Data Science, 2(1), pp.58-75.

[9] Bodaghi, A. and Teimourpour, B., 2018. Automobile Insurance Fraud Detection Using Social Network Analysis. In Applications of Data Management and Analysis (pp. 11-16). Springer, Cham.

[10] Dionne, G., Giuliano, F. and Picard, P., 2003. Optimal auditing for insurance fraud.

[11] Gonzalo Ferreiro Volpi (2019). Class Imbalance: a classification headache. [online] Medium. Available at: https://towardsdatascience.com/class-imbalance-a-classification-headache-1939297ff4a4 [Accessed 2 Sep. 2019].

[12] Dataman (2018). Using Under-Sampling Techniques for Extremely Imbalanced Data. [online] Medium. Available at: https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-part-i-under-sampling-a8dbc3d8d6d8 [Accessed 23 Aug. 2019].

[13] https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-part-ii-over-sampling-d61b43bc4879 Paul, S. (2018). Diving Deep with Imbalanced Data. [online] DataCamp Community. Available at: https://www.datacamp.com/community/tutorials/diving-deep-imbalanced-data [Accessed 28 Aug. 2019].

[14] Analytics Vidhya (2019). How to handle Imbalanced Classification Problems in machine learning? [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/ [Accessed 28 Aug. 2019].

[15] Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, pp.321-357.

[16] Datasciencecentral.com. (2011). Handling imbalanced dataset in supervised learning using family of SMOTE algorithm. [online] Available at: https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family [Accessed 23 Aug. 2019].

[17] SeattleDataGuy (2017). Boosting and Bagging: How To Develop A Robust Machine Learning Algorithm. [online] Medium. Available at: https://medium.com/better-programming/how-to-develop-a-robust-algorithm-c38e08f32201 [Accessed 28 Aug. 2019].

[18] Hackernoon.com. (2018). Boosting Algorithms: AdaBoost, Gradient Boosting and XGBoost. [online] Available at: https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c [Accessed 28 Aug. 2019].

[19] mariajesusbigml (2017). Introduction to Boosted Trees. [online] The Official Blog of BigML.com. Available at: https://blog.bigml.com/2017/03/14/introduction-to-boosted-trees/ [Accessed 28 Aug. 2019].

[20] Hetal Vinchhi (2018). Machine Learning[U+202F]: Introduction to Logistic regression in Python. [online] Assignment Solution Guru. Available at: http://assignmentsolutionguru.com/article/intro-to-logistic-regression-in-python [Accessed 28 Aug. 2019].

[21] Built In. (2019). A complete guide to the random forest algorithm. [online] Available at: https://builtin.com/data-science/random-forest-algorithm [Accessed 23 Aug. 2019].

[22] View all posts by HV (2017). A Tutorial to Understand Decision Tree ID3 Learning Algorithm. [online] The Null Pointer Exception. Available at: https://nulpointerexception.com/2017/12/16/a-tutorial-to-understand-decision-tree-id3-learning-algorithm/ [Accessed 28 Aug. 2019].

[23] QuantInsti. (2019). Gini Index For Decision Trees. [online] Available at: https://blog.quantinsti.com/gini-index/ [Accessed 1 Sep. 2019].

[24] Koehrsen, W. (2017). Random Forest Simple Explanation. [online] Medium. Available at: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d [Accessed 28 Aug. 2019]

[25] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

[26] Vishal Morde (2019). XGBoost Algorithm: Long May She Reign! [online] Medium. Available at: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost- algorithm-long-she-may-rein-edd9f99be63d.

[27] Koehrsen, W. (2018). Beyond Accuracy: Precision and Recall. [online] Medium. Available at: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da0bea9f6c [Accessed 23 Aug. 2019].

[28] Antony Paulson Chazhoor (2019). ROC curve in machine learning. [online] Medium. Available at: https://towardsdatascience.com/roc-curve-in-machine-learning-fea29b14d133 [Accessed 28 Aug. 2019].

[29] Scikit-learn.org. (2019). sklearn.model_selection.RandomizedSearchCV — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html [Accessed 1 Sep. 2019].

[30] Scikit-learn.org. (2019). Comparing randomized search and grid search for hyperparameter estimation — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html [Accessed 1 Sep. 2019].

[31] Mlcourse.ai. (2011). topic-4-linear-models-part-3-regularization. [online] Available at: https://mlcourse.ai/articles/topic4-part3-regularization/ [Accessed 1 Sep. 2019].

[32] Bahnsen, A.C., Stojanovic, A., Aouada, D. and Ottersten, B., 2013, December. Cost sensitive credit card fraud detection using Bayes minimum risk. In 2013 12th international conference on machine learning and applications (Vol. 1, pp. 333-338). IEEE.

[33] Anumalasetty, K. (2018). New automated machine learning capabilities in Azure Machine Learning service. [online] Microsoft.com. Available at: https://azure.microsoft.com/en-us/blog/new-automated-machine-learning-capabilities-in-azure-machine-learning-service/WT.mc_id=oreilly-webinar-lazzeri [Accessed 23 Aug. 2019].