Machine Learning Methods for the Detection of Fraudulent Insurance Claims

Sisheng Zhao

A Thesis

in

The Department

of

Mathematics and Statistics

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Arts (Mathematics) at

Concordia University

Montreal, Quebec, Canada

March 2020

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Sisheng Zhao

Entitled: Machine Learning Methods for the Detection of Fraudulent Insurance Claims

and submitted in partial fulfillment of the requirements for the degree of

Master of Arts (Mathematics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Dr. Cody Hyndman

_____ Examiner

Dr. Arusharka Sen

_____ Examiner

Dr. Yogendra P. Chaubey

_____Supervisor

Dr. Jose Garrido

Approved by _____

Galia Dafni, Graduate Program Director

____March 24___ 2020 _____

Dr. André Roy, Dean, Faculty of Arts and Science

ABSTRACT

Machine Learning Methods for the Detection of Fraudulent Insurance Claims

Sisheng Zhao, M.A.

This thesis focuses on automotive fraudulent claims detection, a particular Property and Casualty (P&C) insurance product. By analyzing the customer's information, we try to define a model to determine if one customer has filed a fraudulent claim.

Two datasets used in this thesis. One of them is very imbalanced, as 6.1% of policyholders file fraudulent claims (coded as *1*) and 93.9% of policyholders file normal claims (coded as *0*). So, we need to deal with the imbalanced classes, by using rebalanced methods such as SMOTE and under-sampling. Then we use classical methods (naïve Bayes and logistic regression) and new data science methods (random forest and gradient boosting) to detect the fraudulent claims. During the process, we compare these methods to find which one performs better for this application.

In addition, the combination of SMOTE and clustering is also used to these two datasets, which is unusual in fraud detection. But the results have been improved a lot for all these four classification models. What is more, link analysis method has also been mentioned in the conclusion.

These methods have also been used to another dataset, which is not that imbalanced, with 24.7% of fraudulent claims and 75.3% of normal claims. The reason for using two datasets is to see if the degree of imbalance affects the performance of the oversampling, undersampling and different models. If so, then these methodologies will be more convincing. If not, we can dig deeper to find the reason.

Acknowledgments

First of all, I want to thank my supervisor, Prof. Jose Garrido. I thank him for being my professor, giving me the chance to study Actuarial Science at Concordia University. I really appreciate this chance because that is a big turning point in my life, which allowed me to learn more about mathematics and actuarial science, changing my career path and future life. I also want to thank him for his patience, support, guidance on my study and this thesis, which is very valuable and important.

I would also thank Dr. Arusharka Sen and Ms. Debbie Arless for helping me registering for courses and answering my questions. My gratitude also to Prof. Yogendra P. Chaubey and Dr. Arusharka Sen for their questions and comments at the thesis defence, which helped improved this final version of the thesis.

Also, I thank Dong Qiu, Shuangning Cao and other friends for the help in my daily life and in finding me internships. Thanks to IACO and my manager Jesper at IATA, they gave me the chance to improve my abilities in an all-round way.

What is more, I want to thank Dr. Jose Garrido and the Department of Mathematics and Statistics in Concordia University for the financial support they provided, which is very important for me.

I thank my parents for supporting all the decisions I made and letting me to try everything I want to do.

Finally, thanks for everything I have been through.

To my parents: Guxiang and Hualing.

Contents

List of Figures

List of Tables

Introduction

The insurance industry is undergoing a major transformation due to the need to improve customer experience and rapid claims processing. Insurers operate in a highly competitive environment, and each additional cost can seriously affect their profitability.

Organized fraudsters often use multiple product lines, like using fake identities to remain undiscovered, and often collude with the employees and suppliers. Insurance companies are also often seen as acceptable and easy targets for opportunity fraud. In the current environment, it is necessary to detect more fraud. In the particular Property and Casualty (P&C) insurance is subjected to fraud, hence insurers try to detect, investigate and prevent fraud in claims, while minimizing the impact on real claimants. And most of detection methods used in insurers are the Machine learning (ML) classification models.

It is known that ML is about more than just using computers for fast calculation and data retrieval. Combining these two capabilities of a computer system makes it seem to learn and make rational decisions based on previously observed conditions and previous actions or reactions, rather than just acting on a fixed program. ML is used not only for search engines and stock market analysis, but also for classification of DNA sequencing, medical diagnostics, speech and handwriting recognition, and robotics. Machine learning technology can be used in a wide range of applications, and more uses are discovered over time. It allows computer systems to be improved in a dynamic environment where the input signal is unknown, and the best decisions can only be learned from historical data.

In this situation, data scientists came up with ML uses to solve problems in insurance companies. One such example is fraudulent insurance claim detection, where a policy-holder's attributes are related to the response variable; fraudulent claims equal to *1*; and *0* otherwise.

Chapter 1

Literature Review

1.1 Overview

Fraud detection is a topic that applies to many industries, including banking and finance, insurance, government agencies and law enforcement agencies. In recent years, fraudulent attempts have increased dramatically, making fraud detection more important. Despite the efforts of different institutions, large amounts of money are lost each year due to fraud. The detection of these frauds is difficult because the percentage of fraudulent activities is very small (see [14, 15, 18]).

In insurance, 25% of claims include different kinds of fraud, resulting in approximately 10% of insurance expenses. The scope of fraud ranges from exaggerated losses to accidents that result in expenditures. Because of the different methods to fraud, it becomes more difficult to identify them (see [18]).

Data mining and statistics help predict and quickly detect fraud and take immediate action to minimize the cost. By using sophisticated data mining tools, one can search millions of claims to discover patterns and detect fraudulent claims.

An important early step in fraud detection is to identify factors that are related to fraud. Once these phenomena and characteristics are identified, it is easier to manage and detect fraud. Then the next step is to use some predictive models to identify the fraudulent claims.

1.2 Existing Methods

There are some existing models used to detect the fraudulent claims, such as naïve Bayes, logistic regression and random forest.

Ridgeway (1998) used the evidence reconstruction formula of the naive Bayesian scoring to diagnose insurance claim fraud. This method combined the advantages of boosting and representative attractiveness of the probability weights of the evidence scoring framework. They

presented the results of an experimental comparison, focusing on the discriminative power and probability estimates' calibration. The dataset evaluated for this method included a representative set of closed personal injury protection auto insurance claims of accidents in Massachusetts in 1993. The results show that this method has a valuable contribution to an effective, efficient and easy fraud detection.

With the increase of credit card transactions, credit card fraud has become more and more common in recent years. Fraud is a serious problem faced by credit card issuers. In 2004, credit card transactions in the United States caused a total loss of fraud of $800 million. The same year in UK, credit card fraud caused losses of 425 million pounds ($750 million). In China, the lag in risk management has become one of the biggest obstacles to business growth and profitability. So, for researchers in the private finance business of some banks, credit card risk management has become one of the most important topics. In this situation, Sahin and Duman (2001) proposed to use logistic regression to detect the credit card fraud, which also achieved great improvements for the fraud detection.

What is more, auto insurance fraud is spreading all over the world, and detecting the automobile insurance fraud is more and more important to the society and insurance company (see [23, 28]). Due to the imbalanced dataset (classes of dataset are not represented equally) of actual auto insurance claims and the real data of auto insurance company being selected, a random forest fraud model was established to detect the auto insurance fraud (Li, Yan, Liu and Li, 2016). The error of the model is analyzed, and then the method is verified by empirical analysis. The empirical results show that compared with the traditional model, the auto insurance fraud detection model (random forest) is suitable for large and imbalanced dataset. It can be better used for the classifying the auto insurance claims and detecting fraudulent claims. Besides, it also has good accuracy and robustness.

1.3 Discussion About Existing Methods

There is no doubt that before using these models, the first step is to clean the dataset such as dealing with those variables with missing data and unbalanced data.

In terms of dealing with unbalanced dataset, one common technique is the SMOTE method to rebalance the dataset; while others use a combination of SMOTE and undersampling to rebalance the dataset. When using a combination of SMOTE and undersampling methods, it is important to calibrate the ratio of SMOTE over undersampling. People normally suggest to use a 2:1 ratio (SMOTE:undersampling). For different datasets, maybe different ratios will be better; for instance, after testing for this thesis, a 3:2 ratio was seen to be better than 2:1.

After cleaning the dataset, it is common to choose logistic regression, because in many cases, it provides better model sensitivity than a naïve Bayes. Naïve Bayes is a simple probability calculator.

Before using logistic regression, data analysts usually check assumptions, such as multicollinearity or continuous independent variables being linearly related to dependent variables. Then the related variables can serve as input into the model to get a confusion matrix that measures the quality of logistic regression prediction accuracy.

In addition, as mentioned before, some analysts use decision trees or random forest to detect fraudulent claims. These two methods are relatively easier to use than logistic regression, because there is no need to check for assumptions before implementing them. The computations are also very fast. What is more, the theories behind these two methods is simple. For decision trees, at each root node, if the *Gini* index is small, then the classification at that root node is good. For random forest, which is a combination of several resampled decision trees, the result is the same as that of most decision trees.

To check the quality of different models, some analysts use the *accuracy rate,* although I personally think it is not that good. Here our main objective is to find fraudulent claims, so we need to focus on the *recall* to see what percentage of fraudulent claims have been detected. This will be discussed in detail later in *Section 3.3*.

Chapter 2

Theories of Clustering and Different Classification Methods

2.1 The Naïve Bayes Classifier

The naïve Bayes classifier is a probabilistic classifier based on Bayes' theorem (see Murphy, 2006). The latter describes the relation between conditional probabilities of a hypothesis and observations as given in *Eq. (2.1 invisible)*. Assume that h represents the hypothesis and O represents the observation made.

$$P(h|O) = \frac{P(O|h)P(h)}{P(O)}, \tag{2.1}$$

where:

- $P(h)$ = prior probability of the hypothesis,
- $P(O)$ = prior probability of observations O,
- $P(h|O)$ = probability of hypothesis given O (posterior probability),
- $P(O|h)$ = probability of O given hypothesis (likelihood).

Typically, the most probable hypothesis or the maximum *a posteriori* hypothesis needs to be identified. The maximum posterior (h_{MAP}) is given by *Eq. (2.2)*:

$$h_{MAP} = \arg\max P(h|O) = \arg\max \frac{P(O|h)P(h)}{P(O)} = \arg\max P(O|h)P(h). \tag{2.2}$$

Now, let $H = h_j \epsilon \{h_1, h_2, \dots, h_m\}$ be the hypotheses, assuming that hypotheses are *mutually exclusive* and *exhaustive* and $\langle O_1 = o_1, O_2 = o_2, \dots, O_n = o_n \rangle$ be the various observations made. Then the most probable hypothesis is given by *Eq. (2.3)*:

$$\begin{aligned} h_{MAP} &= \arg max_H P(h_j | o_1, o_{2\,,\dots,}\, o_n) \\ &= \arg max_H \frac{P(o_1, o_{2\,,\dots,}\, o_n | h_j) P(h_j)}{P_{o_1, o_{2\,,\dots,}\, o_n}} \\ &= \arg max_H P(o_1, o_{2\,,\dots,}\, o_n | h_j) P(h_j). \end{aligned} \tag{2.3}$$

The naïve Bayes classifier assumes that the conditional probability of observations given a hypothesis equals to the production of conditional probabilities of each observation given the hypothesis according to *Eq. (2.4)*:

$$P(o_1, o_{2\,,\ldots,}\, o_n | h_j) = \prod_i P(o_i | h_j). \tag{2.4}$$

A substitution of $P(o_1, o_{2\,,\ldots,}\, o_n | h_j)$ by $\prod_i P(o_i | h_j)$ in *Eq. (2.3)* shows that the naive Bayes classifier is given by *Eq. (2.5)*:

$$h_{NB} = \arg max_H\, P(h_j) \prod_i P(o_i | h_j). \tag{2.5}$$

The naïve Bayes classifier is a supervised learning algorithm, which means it needs to be trained before it can be classified. Therefore, it must have a training set that contains several observations and categories of classification. For example, the training set shown in *Table 2.1.1* contains four parameters (*T, L, H, P*) and one class (*Fraud*) values, where various parameter value sequences are classified.

T	*L*	*H*	*P*	*Fraud*
A	B	A	C	NO
A	A	B	A	NO
B	B	A	A	NO
A	A	C	C	NO
B	A	B	C	YES
B	C	C	A	YES
B	A	B	B	YES

Table 2.1.1: A Training Set Containing Labeled Data Rows.
(Source: Murphy, 2006, p11)

The purpose of the naive Bayes classifier is to classify an unobserved sequence of parameter values into a class in the training set. Suppose the values to be classified are B, A, A, C. The classifier must classify this sequence of va‘giveslues into one of the fraud categories: YES or NO. According to *Eq. (2.5)*, you must choose a hypothesis with a greater likelihood. The classifier needs to reference the training set to calculate the probability of each class based on the probability distribution in the training set. To calculate the probability of class NO, when the data row is

classified as NO, the classifier must calculate the number of data rows with *T* equal to B. The standard has 1 data row; there are 3 data rows, where *T* is equal to B, and the data row is classified as YES. Therefore, the conditional probability of *T* is equal to B, and given NO is equal to 1/4. The classifier calculates all conditional probabilities.

As shown in *Table 2.1.2,* raw values of smartphone sensors are numeric (*i.e.* continuous) whereas, the input data of the naive Bayes classifier should be nominal. Thus, a method is required for converting numeric data to nominal data.

Temperature (℃)	Light (lux)	Humidity (Percent)	Pressure (mbar)
23	350	33	989.5
22.5	400	32	1001.5
23	410	33	1000.5
23	510	35	993.9
24	71	33	998.5
24	55	32	100.4

Table 2.1.2: An Example of Sensor Values Captured from Smartphone Sensors.
(Source: Murphy, 2006, p12)

2.2 The Logistic Regression Classifier

2.2.1 Introduction

Multivariate statistical analysis methods commonly appear in many fields. The terms "multivariable analysis" and "multivariate analysis" are often used in the literature. Strictly speaking, multivariate analysis refers to the simultaneous prediction of multiple outcomes. Multivariate analysis uses several variables to predict one outcome. The multivariable approach explores the relationship between more than one independent variable and the dependent variable. Then we obtain the coefficients that give the best fit for the certain model. The coefficients represent the effects of independent variables on the dependent variable.

The model has two purposes: (1) it predicts the dependent variable for the new value of the independent variable, and (2) it can help to show the contribution of each independent variable to the dependent variable and control the other independent variables of the influencing factor.

The four multivariable methods (linear regression, logistic regression, discriminant analysis, and proportional hazard regression) have many mathematical similarities, but the dependent variables are expressed and formatted differently. In linear regression, for example, in health science, the dependent variable is continuous, such as blood pressure. In logistic regression, dependent variables are usually binary events such as 'alive' versus 'dead'.

2.2.2 Concepts About Logistic Regression

According to Park (2013), logistic regression refers to the logistic model, analyzing the relationship between several independent variables and the categorical dependent variable, and estimating the probability of occurrence of the event by fitting a logistic curve to the data.

2.2.2.1 Odds

The odds mean the ratio of the event occur probability to the probability that it does not occur. If the probability of occurring is p, the probability of not occurring is (1 - p). Then the corresponding odds is given by:

$$odds = \frac{P}{1-P}\,.$$

Since the probability of a logistic regression calculation event occurring exceeds the probability that the event did not occur, the effect of the independent variable is usually explained by the odds. Using logistic regression, the average response variable p in terms of the explanatory variable x is modeled relating p and x by the equation $p = \alpha + \beta x$.

Unfortunately, this is not a good model because the extremum of x gives a value of $\alpha + \beta x$, which does not necessarily fall between 0 and 1. The logistic regression solution to this problem is to use the natural logarithmic to transform the odds. Using logistic regression, we can get the natural log odds as a linear function:

$$\text{logit(y)} = \ln(\text{odds}) = \ln\left(\frac{p}{1-p}\right) = \alpha + \beta x\,. \qquad (2.6)$$

This is a simple logistic model. If we take the antilog of *Eq. (2.6)* on both sides, we can derive an equation for the prediction of the occurrence probability of the outcome as

$$p = P(Y = interested\ outcome \mid X = x, a\ specific\ value)$$

$$= \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}} = \frac{1}{1 + e^{-(\alpha+\beta x)}}.$$

If we extend the simple logistic regression to multiple predictors, we can get a complex logistic regression as

$$\text{logit(y)} = \ln\left(\frac{p}{1-p}\right) = \alpha + \alpha_1 x_1 + \cdots + \alpha_k x_k .$$

Therefore,

$$p = P(Y = interested\ outcome \mid X_1 = x_1, \ldots, X_k = x_k)$$

$$= \frac{e^{\alpha + \beta_1 x_1 + \cdots + \beta_k x_k}}{1 + e^{\alpha + \beta_1 x_1 + \cdots + \beta_k x_k}} = \frac{1}{e^{-(\alpha + \beta_1 x_1 + \cdots + \beta_k x_k)}}.$$

2.2.2.2 The Logistic Curve

When *y* contains binary code (0, 1-- failed, successful), logistic regression is a method to fit the regression curve, $y = f(x)$. When the dependent variable is binary and *x* is a numerical value, logistic regression fits the logistic curve between *x* and *y*. Logistic curves are "*S*"-shaped or sigmoid curves that are commonly used to model population growth.

A basic logistic function is defined by:

$$f(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}},$$

which is graphed in *Figure 2.2.1*.

To provide flexibility, the above function can be extended to the form:

$$f(x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} = \frac{e^{\alpha+\beta x}}{1+e^{-(\alpha+\beta x)}},$$

where α and β mean the logistic intercept and slope.

Figure 2.2.1 shows the logistic function with α and β being 0 and 1, respectively. The logistic function is used to transform the *S*-shaped curve into an approximate line and change the scale from 0 - 1 to -∞ - +∞ as

$$\text{logit}(p) = \ln(odds) = \ln\left(\frac{p}{1-p}\right) = \alpha + \beta x \,,$$

where p is the probability of interested outcome, α is the intercept parameter, β is a regression coefficient, and χ is a predictor.

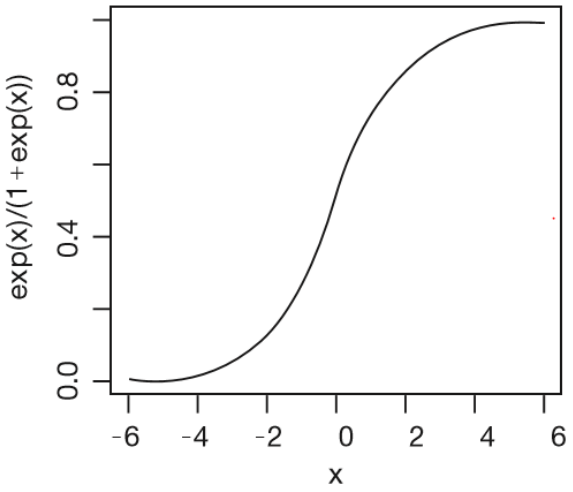


Figure 2.2.1: Graph of Logistic Curve Where α = 0 and β = 1 (Source: Park, 2013, p16)

2.2.2.3 Assumptions of Logistic Regression

Logistic regression does not require some of the main assumptions of linear regression models, especially on the linear relationships between independent variables and the dependent variable, the normality of the error and the homoskedasticity of the error. Logistic regression can deal with the nonlinear relationship between the dependent variable and the independent variable because it applies a nonlinear logarithmic transformation to linear regression. The error terms (residual) do not need to have a multivariate normal distribution - although multivariate normality produces a more stable solution. For each level of independent variable, the variance of the error can be heteroscedastic.

Generally speaking, it uses maximum likelihood estimation to predict group membership. However, in order to accurately interpret the predictions of group members, a preliminary analysis of the data set needs to be conducted to check if the assumptions of logistic regression are met.

2.2.2.3.1 Absence of Multicollinearity

The limitation of logistic regression is that it is sensitive to variables that have very high correlations with each other. Highly collinear variables usually produce very large standard errors and expanded regression estimates. Therefore, it is necessary to observe the collinearity between the independent variables in the model. The standard procedure that allows this is to calculate the tolerance of each variable. The tolerance statistic is the calculation of the variance of each independent variable in the model, not the interpretation of all other independent variables in the model. Higher tolerance values indicate lower collinearity levels. Menard (2010) believes that tolerances less than 0.2 are alarming. Although logistic regression software usually does not provide a tolerance function, we can calculate the model as linear regression to observe the relationship between independent variables.

2.2.2.3.2 Independence

Logistic regression also requires that dependent variables only have mutually exclusive categories. This requirement is met in this thesis because the customer's claim is either fraudulent or reasonable. In addition, each of the clients' claims come from a different unrelated case so there are no dependencies of the responses.

2.2.2.3.3 Lack of Outliers (Logistic Regression)

An outlier is a value that is very different from the other data values in a data set. This can skew results. Outliers often have a significant effect on the sample mean and standard deviation. Because of this, we must take steps to first remove outliers from our data sets before performing any analysis.

2.3 The Random Forest Classifier

Before talking about the random forest algorithm, we need to know the concept of decision tree, because the random forest is just the combination of decision trees, which is based on Breiman (2001).

2.3.1 Introduction to Decision Trees

In machine learning, a decision tree can be used to visually and explicitly represent decision making. As the name implies, it uses a tree-like model to make decisions. Decision trees have decision nodes and branches. The decision node is a point where a choice must be made; it is shown as an oval in *Figure 2.3.1*. The branches extending from a decision node are decision branches, each branch representing one of the possible alternatives or courses of action available at that point. The set of alternatives must be mutually exclusive (if one is chosen, the others cannot be chosen) and collectively exhaustive (all possible alternatives must be included in the set).

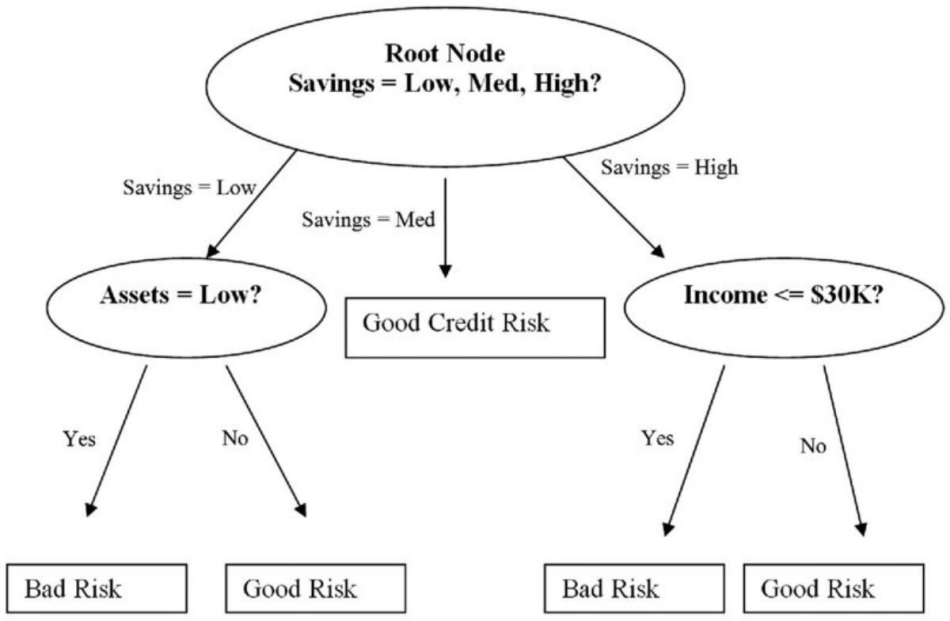


Figure 2.3.1: Decision Tree Example
(Source: Bird, 2018, p4)

The decision tree can be constructed based on the *Gini* index (*G*), which is calculated by subtracting the sum of the squared probabilities of each class from one:

$$G = 1 - \sum_{i=1}^{C}(p)^2 ,$$

Where p is the probability of the ooutcome of interest, as above.

For example, out of 14 instances, say yes = 9 and no = 5. Then

$$G = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 .$$

If the G index is smaller, then the classification in that root node is better.

The dataset contains a large set of features, which results in many splits, in turn producing a very large tree. Such trees are complex and can lead to overfitting.

2.3.2 Introduction to Random Forest

Random forest is one of the most popular and powerful machine learning algorithms. The difference between the random forest algorithm and the decision tree algorithm is that in the random forest, the process of finding the root node and segmenting the feature nodes will run randomly. Bagging (Bootstrap Aggregating) is a technique for converting a single decision tree with poor prediction capabilities into a more accurate prediction function. However, bagging is often affected by tree correlation. The random forest is a modification of the bagging technique, which builds many decorrelated trees. Then get the random forest result from those decision trees. For instance, out of 10 trees, if 6 trees show “yes” and 4 trees show “no”. The random forest output will be “yes”, because “yes” takes a larger percentage.

Often, there is a direct relationship between the number of trees in the forest and the results available: the more trees there are, the more accurate the results will be. For imbalanced dependent variables, when we focus on a certain dependent variable, many trees may lead to bad accuracy, so random forest is not necessarily an improvement.

2.4 The Gradient Boosting Classifier

It is a machine learning method that produces a predictive model (usually decision trees). It builds models in stages like other enhancement methods and promotes these models by allowing the optimization of arbitrary differentiable loss functions (Friedman, 2001).

The gradient boosting contains two techniques: boosting combined with gradient descent, which is also called the steepest descent method. In order to introduce these two concepts, we first need to discuss some theoretical background notions. Therefore, the technical framework will be explained in the following sections before this section focuses directly on gradient boosting.

2.4.1 Predictive Model Framework

Before we introduce machine learning methods, the comparisons with traditional methods are important. *Figure 2.4.1* shows the original situation and compares basic statistical methods with machine learning methods. Here, *Figure 2.4.1a* shows the relationships between input *x* and output *y*, also known as the data generation process, as shown in *Figure 2.4.1b*. Classical statistical methods attempt to describe this relationship through interpretable models. These models usually follow several assumptions that the data may or may not satisfy. If not, these models need to be questioned. By contrast, machine learning does not build relationships directly. Instead, it treats the connection as a black box function, using the learning algorithm to learn *x* and *y* as close as possible (see *Figure 2.4.1c*), which is based on Coors (2018).

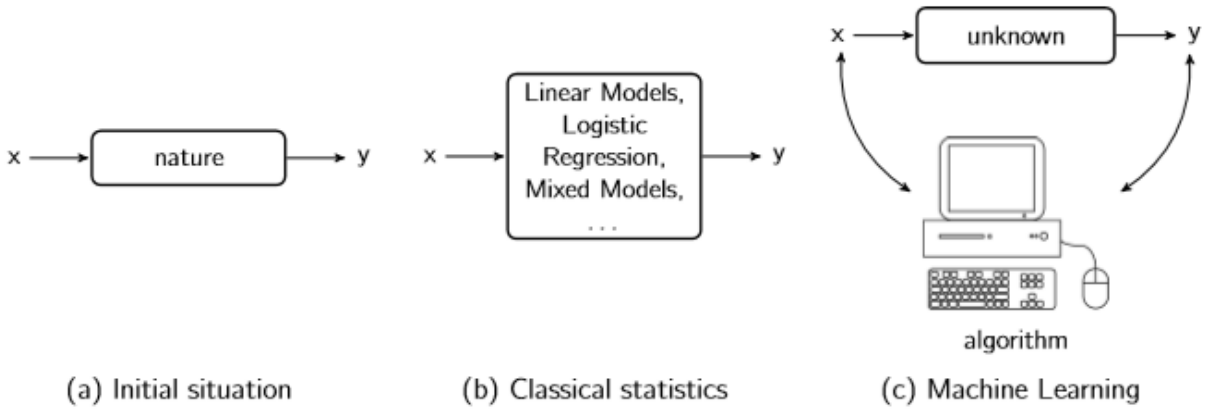


Figure 2.4.1: Comparisons Between the Classical Statistical Method and the Machine Learning Method. (Source: Coors, 2018, p10)

Therefore, the machine learning method is always good at building a good model at the expense of interpretability. Therefore, data scientists usually check whether there is a need to interpret the impact of a single independent variable and choose traditional statistical methods, or instead using machine learning methods. Typical predictive model setup includes a system with a *d*-dimensional random response vector y $\in R^d$ and a set of features $x = \{x_1, x_2, \dots, x_n\}$.

The feature *x* is called explanatory variables and *y* is called result. Then, the aim of predictive model is to use the training dataset that contains tuples (x_i, y_i) for i = 1,...,*n,* to estimate the unknown dataset system by using the function $f()$, such as:

$$f(x) = y. \tag{2.7}$$

The goodness of predictive model is measured by a *loss function* $L(y, f(x))$ and its expected value, which is called *risk*:

$$\mathcal{R}\big(f(x)\big) = \mathbb{E}\left[L\big(y, f(x)\big)\right] = \int L\big(y, f(x)\big) d\mathbb{P}_{xy}\,. \tag{2.8}$$

From this function, we can see that the loss is calculated by the point-by-point deviation of the estimated model $\hat{f}(x)$ from the actual data point *y*. Normally, the loss function can be chosen arbitrarily. But most of the loss functions used are the *least-squares loss*:

$$L\big(y, f(x)\big) = \big(y - f(x)\big)^2, -\frac{\partial L}{\partial f(x)} = 2\big(y - f(x)\big)\,. \tag{2.9}$$

This is equivalent to the maximum likelihood method of the normal distribution error and is therefore sometimes referred to as *Gaussian Loss*. As shown in *Figure 2.4.2*, the quadratic loss is gradually weighted to the point where the distance $\hat{f}(x)$ is the highest. Therefore, it is not robust to outliers. Another type of loss function is called *absolute loss*:

$$L\big(\mathrm{y}, f(x)\big) = |y - f(x)|, -\frac{\partial L}{\partial f(x)} = sign\big(y - f(x)\big), \quad \text{for x} \neq 0.$$

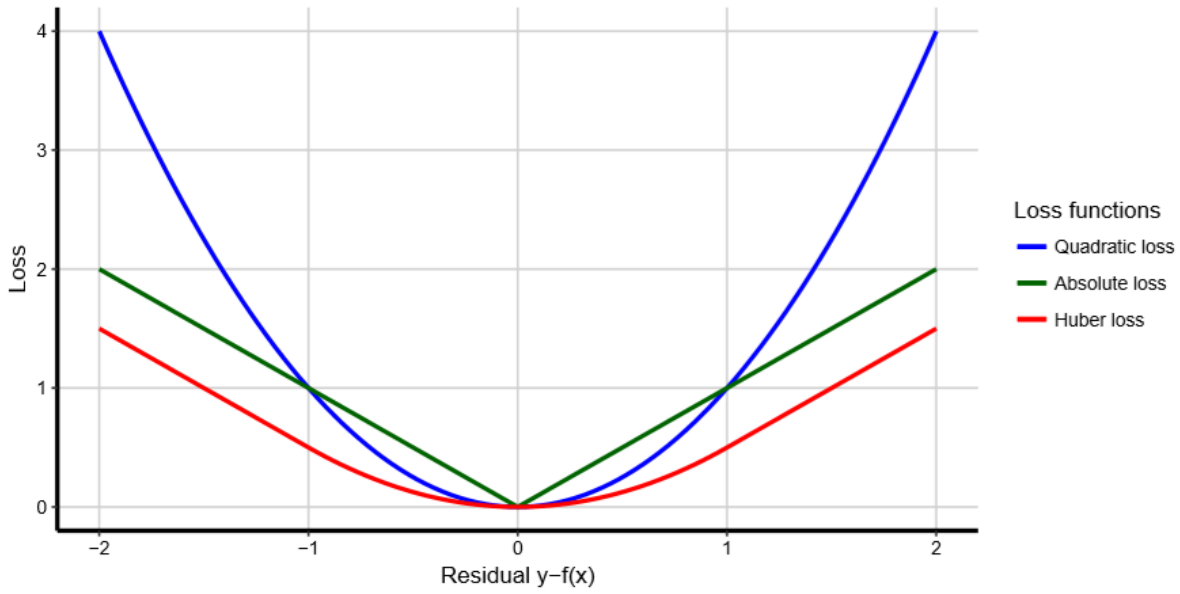


Figure 2.4.2: Common Loss Functions.

(Source: Coors, 2018, p11)

This type of loss function is robust, which can be seen in *Figure 2.4.2* (Coors, 2018).

While the *Huber loss* contains the advantages of both loss functions mentioned before because it combines both of them. For any $\delta > 0$, the *Huber loss* is defined as

$$L(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{for } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases},$$

hence

$$-\frac{\partial L}{\partial f(x)} = \begin{cases} y - f(x), & \text{for } |y - f(x)| \leq \delta \\ sign(y - f(x)), & \text{otherwise} \end{cases}.$$

Huber loss is not only differentiable but also robust because it is quadratic in the interval around 0, followed by linear continuity.

When switching to other tasks like the binary classification, some other types of loss functions will also be used because the regression function is not reasonable. But the *Zero-One* loss function is not totally smooth, so it is not suitable for optimization, as shown in *Figure 2.4.3*. By contrast, loss functions with smooth and convex characteristics are commonly used for the classification, as shown in *Figure 2.4.3* (Coors, 2018). The popular example is an exponential loss, defined as

$$L\big(y, f(x)\big) = \begin{cases} \exp\big(-yf(x)\big) & \text{for y } \in \{-1, +1\} \\ \exp\big(-(2y-1)f(x)\big) & \text{for y } \in \{0, 1\} \end{cases}.$$

Compared to the following methods, it is less robust to observations of strong misclassifications due to the exponential increase in negative values. In addition, the *truncated hinge loss* is also suitable for classification work, which is shown as below:

$$L\big(y, f(x)\big) = \max\big(0, 1 - yf(x)\big) = \ |1 - yf(x)|_+ .$$

It is more robust due to the linearity of negative values. Another possibility has the same rate of return. It is called *binomial loss*:

$$L\big(y, f(x)\big) = \begin{cases} \ln\,(1 + \ \exp(-2yf(x))) & \text{for y } \in \{-1, +1\} \\ -yf(x) + \ln\big(1 + exp\big(f(x)\big)\big) & \text{for y } \in \{0, 1\} \end{cases}. \qquad (2.10)$$

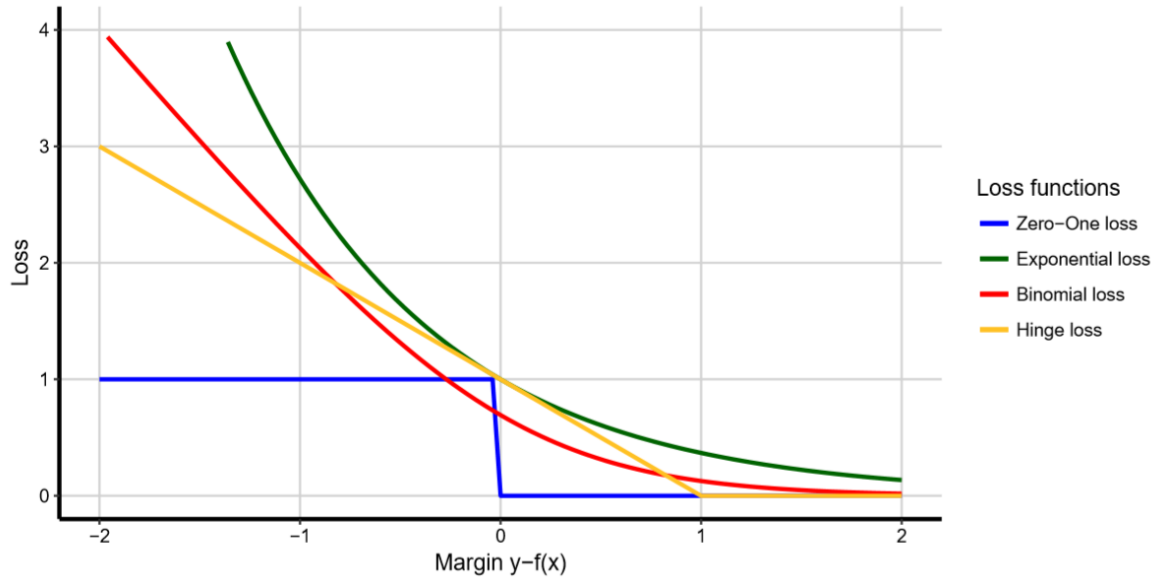


Figure 2.4.3: Common Loss Functions for Regression.
(Source: Coors, 2018, p11)

A similar approach is possible for all maximum likelihood models, which leads to the use of negative log-likelihood as a loss function. By assuming a multinomial model, the binomial loss function can be simply extended to a multi-class classification problem, which is also known as the *softmax* function. This function produces the probability of every data point belonging to which class, and is also used as the objective function for multi-class classification, which is called *softprob*.

Obviously, to finding an optimal estimate $\hat{f}(x)$, which minimizes the risk $\mathcal{R}(f(\mathrm{x}))$ over the joint distribution of the training set, then $\hat{f}(x)$ is determined by

$$\begin{aligned} \hat{f}(x) &= \arg\min \mathcal{R}(f(\mathrm{x})) \\ &= \arg\min \mathbb{E}\left[L(y, f(x))\right] \\ &= \arg\min \mathbb{E}_{x,y}\left[L(y, f(x))\right] \\ &= \arg\min \mathbb{E}_{x}\left[\mathbb{E}_{y}\left(L(y, f(x))\right)|x\right] . \end{aligned} \tag{2.11}$$

2.4.2 Gradient-descent optimization

When it comes to focusing on optimization problems, the stochastic methods can be separated. Deterministic methods are usually faster than random methods, however, the risk of being trapped at a local minimum is significantly higher. Some random methods of hyperparameters and threshold adjustments are described later in this article. However, gradient descent method is a deterministic nonparametric iterative method for numerical function optimization that is often proposed to minimize empirical risk. We consider the case of the *Eq. (2.11)* with an arbitrary, differentiable target function $f(\mathrm{x})$. From Coors (2018), the *gradient* $\nabla f(x)$ can be seen as a pointer, which is always displayed in the *steepest ascent* direction. Similarly, $-\nabla f(\mathrm{x})$ points to the *steepest descent* in $f(\mathrm{x})$. Thus, then gradient means the graph's tangemt slope, which is very similar to derivative.

But compared with the scalar-valued derivative, the gradient is a value that contains the above directions and depends on the underlying space.

So, for $f : \mathbb{R}^n \longrightarrow \mathbb{R}$:

$$\nabla f(x) = grad\ f(x) = \left(\frac{\partial f(x)}{x_1}, \dots, \frac{\partial f(x)}{x_n}\right)^T. \tag{2.12}$$

Then, we select a starting point $x^{(0)}$ as the initial guess. This point can be improved, *i.e.* we can also select the next point $x^{(1)}$ such as:

$$x^{(1)} = x^{(0)} - \upsilon \nabla f\left(x^{(0)}\right),$$

so in general for iteration m,

$$x^{(m)} = x^{(m-1)} - \upsilon \nabla f\left(x^{(m-1)}\right) \text{ for } m = 1, \dots, M, \tag{2.13}$$

where υ controls the *step size* in the steepest descent direction. The optimal υ is able to change in each iteration. And the choice is to minimize the objective function:

$$v^{(m)} = \arg\min f\left(x^{(m-1)} - v\nabla f\left(x^{(m)}\right)\right), \qquad v > 0\,. \tag{2.14}$$

Eq. (2.14) is called *line search*. If the algorithm reaches an $x^{(m)} \in \mathbb{R}^n$ with $\nabla f\left(x^{(m)}\right) = 0 \in \mathbb{R}^n$, then a local minimum can be reached. For the *Figure 2.4*, it illustrates the procedure for a two-dimensional function $f(x, y) = 2x^2 + y^2$.

2.4.3 Boosting

The property of boosting is that the performance of *weak learner* can be improved by adding additional learners. So, boosting means the *stagewise additive models*:

$$f(x) = \sum_{m=1}^{M} f_m(x) = \sum_{m=1}^{M} \beta_m h(x, \theta_m)\,. \tag{2.15}$$

To minimize the empirical risk for *Eq. (2.8)*:

$$\mathcal{R} = \sum_{i=1}^{n} L\big(y_i, f(x_i)\big) = \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m h(x_i, \theta_m)\right), \tag{2.16}$$

that depends on the function $h(\mathrm{x}, \theta_m)$ and especially the β_m and θ_m. Hence, $\mathcal{R}$ needs to be minimized with regard to parameters $(\beta, \theta) = \big((\beta_1, \theta_1), \ldots, (\beta_M, \theta_M)\big)$ which can be difficult if we depend on the chosen loss function L. Therefore, optimization can be reached by using the iterative *"greedy" forward stagewise additive model* approach. Thus, for optimizing

$$(\beta^*, \theta^*) = \arg\min \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m h(x_i, \theta_m)\right), \tag{2.17}$$

we can use

$$(\beta^*, \theta^*) = \arg\min \sum_{i=1}^{n} L\left(y_i, f_{m-1}(x_i) + \beta h(x_i, \theta)\right), \tag{2.18}$$

in order to get

$$f_m(\mathrm{x}) = f_{m-1}(x) + \beta_m h(x_i, \theta_m), \tag{2.19}$$

with

$$f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j h(x_i, \theta_j). \quad (2.20)$$

Here, adding each component step by step means that the previous model is fixed and therefore will not be readjusted. This strategy is called enhancement in a machine learning context.

The typical *weak learner* $h(x, \theta)$, also known as the *base function*, is *tree stumps*, which is a decision tree with few splitting points. These tree stumps bring some advantages for the decision trees, including support for classification features and missing values or robustness with respect to outliers. In addition, the training tree is faster than training other algorithms.

What is more, boosting can greatly improve prediction performance when compared with just training one single tree. However, it is clear to see that they lose some interpretability when combining some trees. These advantages are like random forest methods, which use *bootstrap aggregation* to combine several decision trees for modeling.

2.4.4 Gradient Boosting algorithm

The gradient boosting method contains the gradient descent algorithm in *Section 2.4.2* with the boosting method described in *Section 2.4.3* above. This means that the gradient boosting uses a phased additional model whose empirical risk $\mathcal{R}$ is minimised by gradient descent.

The additive model for *Eq. (2.15)*, we want to find a combination of the parameters (β_m^*, θ_m^*), as shown in *Eq. (2.18)*; that is, we want to find the new additive component $\beta_m h(x_i, \theta_m)$ of *Eq. (2.19)* for iteration m. Here, *Eq. (2.19)*, β_m is the step size of the gradient descent, and the former part is also expressed as υ, also known as the *learning rate*. If $0 < \beta_m \ll 1$, only a small number of base learners are considered in the *m-th* iteration. This helps prevent overfitting of additive models. Cross-validation can be used to select an appropriate learning rate υ in a given application; repeatedly fitting a model for different values of υ to select the one that produces the fitted model with smallest loss,

2.4.4.1 Regression

First of all, the nonparametric model is considered in which each individual observation x_i of the n observations of the training dataset can be arbitrarily predicted. This leads to n parameters $f(x_i)$, but there is no generalization of the whole space x. By gradient descent, we can get the gradient with *Eq. (2.12)* for a loss function L at the point x_j by:

$$\nabla \mathcal{R}|_{x_j} = \frac{\partial \mathcal{R}}{\partial f(x_j)} = \frac{\partial \sum_{i=1}^{n} L(y_i, f(x_i))}{\partial f(x_j)} = \frac{\partial L(y_i, f(x_j))}{\partial f(x_j)}. \tag{2.21}$$

Hence, the update for iteration m by gradient descent is

$$f_m(x_j) \leftarrow f_{m-1}(x_j) - \beta \frac{\partial \sum_{i=1}^{n} L\big(y_i, f_{m-1}(x_i)\big)}{\partial f_{m-1}(x_j)}. \tag{2.22}$$

Consequently, we can determine the steepest descent direction for each x_i and also define these as *pseudo residuals* r_{im}:

$$r_{im} = -\left[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\right]_{f(x)= f_{m-1}(x)}. \tag{2.23}$$

Thus, the optimal weight β_m for iteration m can obtain by setting $r_{im} = h(x_i, \theta_m)$ in *Eq. (2.18)*:

$$\beta_m = \arg\min \sum_{i=1}^{n} L\left(y_i, f_{m-1}(x_i) - \beta \left[\frac{\partial L\big(y_i, f_{m-1}(x_i)\big)}{\partial f_{m-1}(x_j)}\right]\right). \tag{2.24}$$

However, as mentioned above, this only applies to a single observation x_j of the training set. Therefore, it is necessary for the generalization of all $x \in \chi$, which can be achieved by using the regression model to approximate the negative gradient as well as possible. The regression model is called the *base function* or the *weak learner* in *Section 2.4.3*.

$$h(x, \theta_m) = -r_m = -\left[\frac{\partial L\left(y_j, f(x_j)\right)}{\partial f(x_j)}\right] = -\left[\frac{\partial \sum_{i=1}^{n} L\big(y_i, f(x_i)\big)}{\partial f(x_j)}\right]. \tag{2.25}$$

And, minimizing the risk

$$\mathcal{R}\big(h(\mathrm{x},\theta_m)\big) = L(h(x,\theta_m), r_m) \tag{2.26}$$

leads to

$$\theta_m = \arg\min \sum_{i=1}^{n} L\big(r_m, h(x_i,\theta)\big)\,. \tag{2.27}$$

This gives us the best parameter θ for the *Eq. (2.18)*. Finally, the new entire additive portion that contains the weak learner $h(\mathrm{x},\theta_m)$ can be interpreted as a component that improves the model towards the maximum reduction in loss, where β_m is determined by *Eq. (2.24)*, indicating the step size of this move. By using the least squares loss function, *Eq. (2.27)* reduced to

$$\theta_m = \arg\min \sum_{i=1}^{n} \big(r_m, h(x_i,\theta)\big)^2. \tag{2.28}$$

Each learner can be fitted by a quadratic loss. In addition, the solution is numerically efficient.

As mentioned earlier, the choice of decision trees as a basic learner has advantages, which makes them the first choice for autoxgboost:

$$h(x,b,R) = \sum_{j=1}^{J} b_j \mathbb{I}\big(x \,\epsilon\, R_j\big), \tag{2.29}$$

where R_j is the disjoint regions, defined by the tree's terminal nodes with the corresponding means γ_j.

Algorithm 1: Gradient Boosting Algorithm.

Initialize: $\boldsymbol{f_0(x) = arg\ min\ \sum_{i=1}^{n} L(y_i, \theta_0)}$

1. for *m = 1* ⟶ *M* **do**

2. **for** *all i* **do**

3. *Calculate* $\boldsymbol{r_{im}} = -[\frac{\partial L(y, f(x))}{\partial f(x_i)}]_{\boldsymbol{f(x_i) = f_{m-1}(x_i)}}$

4. **end**

5. *Fit regression base learner to the pseudo-residuals* r_{im}:

6. $\theta_m = arg\ min \sum_{i=1}^{n} (r_{im} - h(x_i, \theta))^2$

7. *Find via line search:*

8. $\beta_m = \arg\min \sum_{i=1}^{n} L(y_i, f_{m-1}(x) + \beta h(x, \theta_m))$

9. Update $f_m(x) = f_{m-1}(x) + \beta_m h(x, \theta_m)$

10. end

Output: $\hat{f}(x) = f_M(x)$

Putting *Eq. (2.29)* into *Eq. (2.19)* leads to

$$f_m(x) = f_{m-1}(x) + \beta_m \sum_{j=1}^{J_m} b_{jm} \mathbb{I}(x \in R_{jm}), \tag{2.30}$$

which can be reduced to

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm}). \tag{2.31}$$

when setting $\gamma_j m = \beta_m b_{jm}$, where like before, β_m is determined by line search. Again, minimizing the loss function provides the optimal coefficients for γ_{jm} which is done by

$$\gamma_{jm} = \arg\min \sum_{i}^{n} L\left(y_i, f_{m-1}(x_i) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm})\right). \tag{2.32}$$

Algorithm 2: Gradient Tree Boosting Algorithm.

Initialize: $\boldsymbol{f_0(x) = \arg\min \sum_{i=1}^{n} L(y_i, \theta_0)}$

1. for *m = 1* $\longrightarrow$ *M* **do**

2. **for** *all i* **do**

3. *Calculate* $\boldsymbol{r_{im}} = -[\frac{\partial L(y, f(x))}{\partial f(x_i)}]_{\boldsymbol{f(x_i) = f_{m-1}(x_i)}}$

4. **end**

5. *Fit regression tree to the pseudo-residuals* r_{im} *given terminal regions* $R_{jm}, j = 1, \dots, J_m$:

6. **for** $j = 1 \rightarrow J_m$ **do**

7. $\gamma_{jm} = \arg\min \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$

8. **end**

9. Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm})$

10. end

Output: $\hat{f}(x) = f_M(x)$

Because the R_j are disjoint, we can also get:

$$\gamma_{jm} = \arg\min \sum_{x_i \epsilon R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma). \tag{2.33}$$

For the loss function L, this result is the optimal update given the function f_{m-1}. It can also be determined directly within each terminal region. So, *Algorithm 1* is changed to *Algorithm 2* above.

2.4.4.2 Classification

The general gradient boosting in *Algorithm 1* depends entirely on the loss function L. For the regression, we choose the least squares loss, which equals to the maximum likelihood method of the normal distribution error. In terms of classification, we have seen the appropriate loss functions in *Section 2.4.1*, and we hope to discuss their mathematical derivation in more detail. The first limiting binary classification means that our target variable does not contain contiguous but two classification levels, for example $y \in \{0, 1\}$. If the output of the model is mapped on real values, the positive values can be treated as an indication of class 1 and a negative value of class 0, respectively. Therefore, we obtain a discrete prediction by $\mathbb{I}(f(x) > 0)$. Or, we convert the model so that its function value is in the interval $[0, 1]$. This can be achieved by applying a logical distribution function:

$$logit(\eta) = \frac{\exp(\eta)}{1+\exp(\eta)}. \tag{2.34}$$

where η is called *link function*, because it related to prediction probabilities:

$$\pi_{i1} = \mathbb{P}(y_i = 1|x_{i1}, \dots, x_{ik}) = logit(\eta_i) = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}, \tag{2.35}$$

where

$$\eta_i = x_i^T \beta_i, \qquad x_i = x_{i1}, \dots, x_{ik}\,.$$

Hence, probability x_{i1} is indirectly modeled by the *logit* function

$$\eta_i = logit(y_i = 1|x_{i1}, \dots, x_{ik}) = ln\frac{\pi_{i1}}{1-\pi_{i1}} = x_i^T \beta_i\,. \tag{2.36}$$

Getting the log-likelihood by applying the maximum likelihood method:

$$\sum_{i=1}^{n} (y_i ln\pi_{i1} + (1-y_i)\ln(1-\pi_{i1}))$$
$$= \sum_{i=1}^{n} \big(y_i f(x_i) - \ln\big(1+\exp\big(f(x_i)\big)\big)\big). \tag{2.37}$$

For $f(x_i) = x_i^T \beta_i$. Defining the negative log-likelihood of *Eq. (2.31)* as new loss function which has been metioned in *Eq. (2.10)* in *Section 2.4.1*. That is

$$L\big(\mathrm{y}, f(\mathrm{x})\big) = -yf(x) + \ln\big(1+\exp\big(f(x)\big)\big), \text{ with } -\frac{\partial L}{\partial f(x)} = y - \pi_1(x)\,,$$

where $\pi_1(x)$ is the prediction for the posterior probability of class 1, that is

$$\hat{\mathbb{p}}(y = 1|x) = logit(\hat{f}(x)).$$

By using the maximum likelihood methods, we obtain as loss function:

$$L\big(y_k, f_k(\mathrm{x})\big) = -\sum_{k=1}^{K} y_k ln\pi_k(x)\,, \tag{2.38}$$

where $y_k = \mathbb{I}(y = k)$ for class *k*. So, the posterior probability of class *k* is given by

$$\pi_k(x) = \hat{\mathbb{p}}(y = k|x) = \frac{\exp\,(f_k(x))}{\sum_{j=1}^{J} \exp\,(f_j(x))}\,, \tag{2.39}$$

taking the first derivatives:

$$r_{ik,m} = -\left[\frac{\partial L\left(y_{ik}, f_{k,m}(x_i)\right)}{\partial f_{k,m}(x_i)}\right]_{f_{k,m}(x) = f_{k,m-1}(x)} = y_{ik} - \pi_{k,m-1}(x_i), \tag{2.40}$$

where $\pi_{k,m-1}(x_i)$ is derived from *Eq. (2.39)* for $f_{k,m-1}$. We see that *K* models (trees) are fitted in each iteration *m* to predict the pseudo residuals $r_{ik,m}$. Every single tree has *J* terminal nodes with regions $\{R_{1k,m}, \ldots, R_{Jk,m}\}$:

$$\gamma_{ik,m} = \arg\min \sum_{i=1}^{n} \sum_{k=1}^{K} \phi\left(y_{ik}, f_{k,m-1}(x_i) + \sum_{j=1}^{J} \gamma_{jk}\, \mathbb{I}\big(x_i \in R_{j,m}\big)\right), \tag{2.41}$$

with $\phi\big(y_k, f_k(\mathrm{x})\big) = -y_k \ln\, \pi_k(x)$ from *Eq. (2.38)*.

Based on a single *Newton-Raphson* step, it can be separated into a single calculation for each terminal node:

$$\gamma_{jk,m} = \frac{K-1}{K} \frac{\sum_{x_i \epsilon R_{jk,m}} \gamma_{ik,m}}{\sum_{x_i \epsilon R_{jk,m}} |\gamma_{ik,m}|(1 - |\gamma_{ik,m}|)}\,, \tag{6.36}$$

which serves for the update

$$f_{k,m}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jk,m} \mathbb{I}\big(x \in R_{jk,m}\big). \tag{6.37}$$

Algorithm 3: K-class Classification Gradient Tree Boosting Algorithm.

Initialize: $f_{k,0}(x) = 0, k = 1, \dots, K$

1. for $m = 1 \longrightarrow M$ **do**

2. *Set* $\pi_k(x) = \frac{\exp(f_k(x))}{\sum_{j=1}^{J} \exp(j(x))}$ **for** $k = 1 \longrightarrow M$ ***do***

3. *Calculate* $r_{ik,m} = y_{ik} - \pi_{k,m-1}(x_i), i = 1, \dots, n.$

4. *Fit regression tree to the pseudo-residuals* $r_{ik,m}$ *given terminal regions* $R_{jk,m}, j = 1, \dots, m$:

5. **for** $j = 1 \rightarrow J_m$ **do**

6. $\gamma_{jk,m} = \frac{K-1}{K} \frac{\sum_{x_i \epsilon R_{jk,m}} \gamma_{ik,m}}{\sum_{x_i \epsilon R_{jk,m}} |\gamma_{ik,m}|(1 - |\gamma_{ik,m}|)}$

7. **end**

8. Update $f_{k,m}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jk,m} \mathbb{I}(x \epsilon R_{jk,m})$

9. **end**

10. end

Output: $\hat{f}(x) = f_{k,M}(x)$

Finally, after M steps, $f_{k,M}(x)$ is returned as a final model, as shown in *Algorithm 3*.

2.5 The K-Means Clustering

K-means clustering is a vector quantization method, which is very popular in cluster analysis of data mining. The k-means clustering aims to divide ***n*** observations into ***k*** clusters, where each observation belongs to the cluster with the nearest mean.

For example, given a set of observations $\{x_1, x_2, x_3, \dots, x_n\}$, where each observation is a ***d***-dimensional real vector, k-means clustering aims to divide these ***n*** observations into ***k*** ($\leqslant$n) sets S = $\{S_1, S_2, S_3, \dots, S_n\}$ in order to minimize the sum of variance within the cluster. In other words, the goal is to find:

$$\arg\min \sum_{i=1}^{k} \sum_{x \epsilon S_t} \|x - \mu_i\|^2 = \arg\min \sum_{i-1}^{k} |S_i| Var(S_i),$$

Where μ_i is the mean of different points in set S_i. And the above equation equals to minimizing the pairwise squared deviations of the distances:

$$\arg\min \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{x,y \epsilon S_t} \|x - y\|^2$$

And the above equation can be deduced from $\sum_{x \epsilon S_i} \|x - \mu_i\|^2 = \sum_{x \neq y \epsilon S_i} (x - \mu_i)(\mu_i - y)$. This is because the total variance will not change, which equals to the sum of squared deviations between points in different clusters.

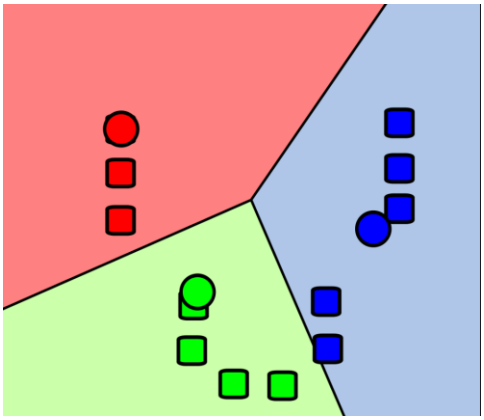


Figure 2.5: K-Means Clustering Example
(Source: Wikipedia)

Chapter 3

Performance Measures

3.1 ROC Curve

In the Receiver Operating Characteristic (ROC) curve, the true positive rate is plotted as a function of the false positive rate for different cut-off points. Each point on the ROC curve represents a sensitivity/specificity pair. A test with perfect discrimination has ROC curve across the upper left corner. Therefore, the closer the ROC curve is to the upper left corner, the higher the overall accuracy of the test.

Since the area under the ROC curve is typically a measure of test usefulness. In other words, a larger area means a more useful test, and the area under the ROC curve is also used to compare the usefulness of the test (Narkhede, 2018).

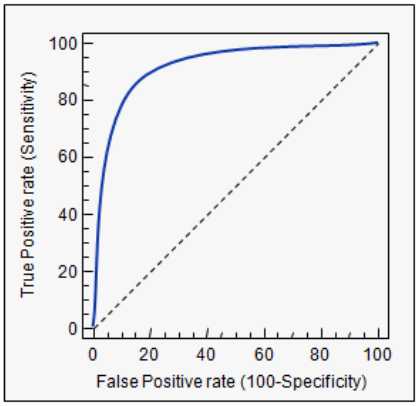


Figure 3.1.1: ROC Curve
(Source: Narkhede, 2018, p2)

3.2 Confusion Matrix

In the field of machine learning, especially statistical classification problems, the confusion matrix is also known as an error matrix, which takes a specific table layout that allows the visualization

of an algorithm's performance. Each row of the matrix represents an instance in the predictive class, and each column represents an instance in the actual class. This name stems from the fact that it makes it easy to see if the system confuses two classes.

		Actual class	
		Cat	**Non-cat**
Predicted class	**Cat**	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	3 True Negatives

Figure 3.2.1: Confusion Matrix Sample
(Source: Narkhede, 2018, p2)

3.3 Recall and Precision

The performance of machine learning algorithms is usually evaluated by the confusion matrix, as shown in *Figure 3.3.1*. The column is the predicted class and the row is the actual class. In the confusion matrix, TN (True Negative) is the number of examples of correct negative (as *0*) classification, FP (False Positives) is the number of negative (as *0*) misclassified as positive (as *1*), and FN (False Negative) is the number of positive (as *1*) that are misclassified as negative (as *0*), while TP (True Positives) is the number of positive (as *1*) correctly classified.

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Table 3.3.1: Confusion Matrix

Predictive accuracy is a measure of the performance of machine learning algorithms and is defined as Accuracy = (TP + TN) / (TP + FP + TN + FN). For balanced data sets and equal error costs, it is reasonable to use error rates as performance metrics. The Error Rate is $(1 - \text{Accuracy})$.In the case of an imbalanced dataset with unequal error costs, it is more appropriate to use ROC curves or other similar measures. The ROC curve can be thought of as the best decision boundary family representing the relative cost of TP and FP. On the ROC curve, the X-axis represents %FP = FP / (TN + FP) and the Y-axis represents %TP = TP / (TP + FN). The ideal point of the ROC curve is (0, 1.0); that is, all positive cases are correctly classified and no negative examples are misclassified as positive. One way in which the ROC curve can be swept is by manipulating the balance of the training samples for each class in the training set. In the ROC curve, the line y = x represents the scenario of the random guess class. The Area under the ROC Curve (AUC) is a useful measure of the classifier performance because it is independent of the chosen criterion and prior probability. AUC can be used for comparisons between different classifiers.

For some examples, the AUC method is not that useful to see the accuracy of the classifier. For instance, The Information Retrieval (IR) domain also faces the problem of imbalances in the data set. Take a document or web page that is converted to a word bag representation; that is, a feature vector reflecting the appearance of the word in the page is constructed. Often, there are very few instances of interest categories in text categorization. In information retrieval problems, the excessive performance of negative categories may lead to problems in assessing classification performance. Since the error rate is not a good indicator for skewed data sets, the classification performance of algorithms in information retrieval is usually measured by *recall* and *precision*:

$$recall = \frac{TP}{TP+FN},$$

$$precision = \frac{TP}{TP+FP}.$$

In terms of this thesis, we need to detect insurance fraud, which means we need to correctly find the insurance frauds among real fraudulent claims. Instead of using *accuarcy rate*, we need to use *recall* to measure the quality of models, because from the above equation, we can see that *recall*

means what percentage of fraudulent claims can be detected, which is the main objective. Therefore, the first and most important step is to improve the performance of *recall* to measure most of the real fraudulent claims that have been detected. And then we can see if the *precision* is good enough. Because *precision* stands for what percentage of legitimate claims will not be mistook for fraudulent claims, which is not as important as *recall*.

Chapter 4

Dataset 1 Description and Manipulation

4.1 Description

4.1.1 Dependent Variable

The first dataset studied is about car insurance claims and it can be found at Kaggle (url: https://www.kaggle.com/srikanthmalyala/exleq/data). It consists of 11,554 observations with 32 variables. The response variable named the "FraudFound_P" is either *1* for a driver who filed a fraudulent claim or *0* for car insurance claims that are legitimate. There are also 31 independent variables being the information of each driver (sex, age of policyholder, age of vehicle, vehicle price, claim size, etc.).

count	11,554
mean	22,966
std	26,995
min	0
25%	4,149
0%	8,131
75%	46,490
max	141,394

Table 4.1.1: Claim Severity Summary Statistics

As shown in *Table 4.1*, 75% of claims are small than $46,490. The mean claim amount is $22,966, and the median claim amount is $8,131. Also, from the common sense, we know the minimum of claim is 0, which means there is no accident for certain clients.

The dependent variable for fraudulent claims, yes or no, is bivariate (*1* or *0*), and it is clear from *Figure 4.1.1* that this dependent variable is very imbalanced (6.1% if *1*s and 93.9% if *0*s). This is quite common in practice, because only a few drivers filed fraudulent claims.

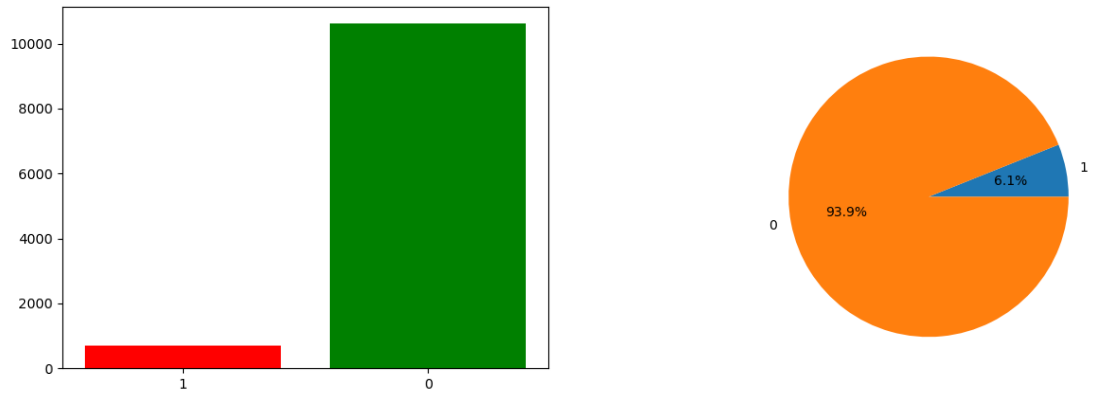


Figure 4.1.1: Bar Chart and Pie Chart of the Response Variable

4.1.2 Correlation Matrices

Correlation matrices are central to understanding our data, because we want to know what features would influence the detection of fraud. Instead of using the dataset to get correlation matrix directly, it is important to use the adjusted dataset (subsample) to get a more obvious relationship among different features and the fraudulent claims.

Before doing that, an adjustment is needed as the dependent variable is very unbalanced. So, we use a subsample in our correlation matrix; otherwise, our correlation matrix will be affected by the high imbalance between the classes, due to the high unbalance in the original dataset.

Once we determine how many instances are considered as fraudulent claim (Fraud = "1"), we need to reduce the non-fraudulent claims frequency to the same number as fraudulent transactions (assuming we want a 1/1 ratio), this will be equivalent to 659 cases of fraudulent and 659 cases of non-fraudulent transactions. After implementing this subsample technique, we get a sub-sample of our dataset with a 1/1 ratio with regards to our classes, and then use it to get the correlation matrix.

The main issue with "random under-sampling" is the risk that our classification models will not perform as accurately as we expect since there is a great deal of information loss (reducing to 659 non-fraudulent transactions from 10,641).

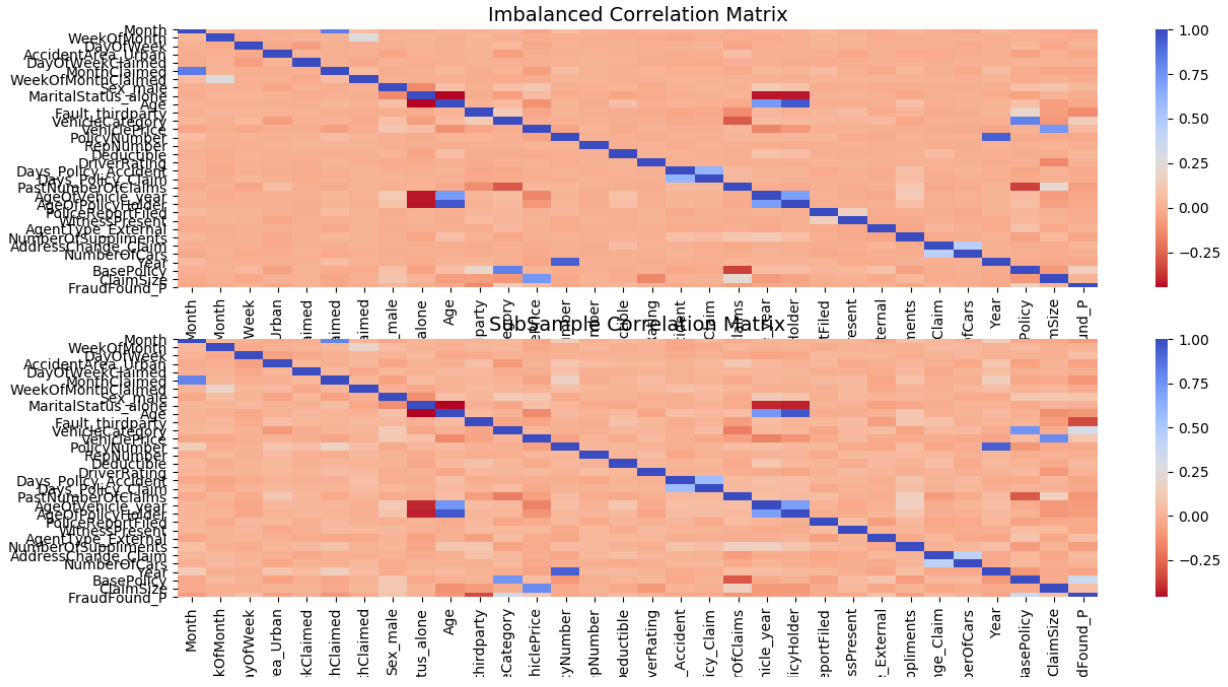


Figure 4.1.2: Imbalanced Correlation Matrix and Sub-Sample Correlation Matrix

From the output, we can see that the correlations are more obvious after using the subsample in the correlation matrix. And we can also see that the “VehicleCategory” and “BasePolicy" tend to have a positive relationship with the response variable, while the “Fault_thirdparty” has a negative relationship with the dependent value (fraudulent claim).

A note of caution, the coefficient of correlation might not be a good mesure of depence when applied to a binary response, like we have here, and to independent variables that can be binary, categorical or discrete.

4.2 Data Manipulations

In this part, we detail some common data manipulations that were carried out across all the analyses, any specific manipulation required for certain analyses will be stated in this section. To be able to perform cross-validation, the dataset was first divided into two parts. The training dataset contains 9,243 observations, and the testing set contains 2,311 observations.

4.2.1 Features Selection

Before fitting any models, it is important to check the relation between all the features, there are two main reasons:

• If two features are highly linearly correlated, the data may have multicollinearity effects, especially for the logistic regression.

• By reducing the number of input variables, we may have fewer parameters requiring tuning when fitting models and consequently reduce the computational load.

4.2.2 Standardization and Scaling

Since our features are expressed in different measurement scales, we standardize or scale the features based on the following:

- AccidentArea: Change to 1 (Urban) and 0 (Rural)
- Sex: Change to 1 (Male) and 0 (Female)
- MaritalStatus: Change to 1 (Single) and 0 (Married)
- Fault: Change to 1 (Third Party) and 0 (Policy Holder)
- Witness: Change to 1 (Yes) and 0 (No)
- AgentType: Change to 1 (External) and 0 (Internal)
- PolicyRepordField: Change to 1 (Yes) and 0 (No)
- BasePolicy, VehicleCategory: Dummy variables

4.2.3 Missing Data

In terms of the missing data, there are several situations:

- If the data is missing randomly and more than 70% data of certain feature is missing, then delete this feature directly;
- If the feature with missing data has a trend based on time, then fill in the missing data by time;
- Check if those features with missing data have some relationships with other features. If so, using other related variables as independent variables and using the missing data as the dependent variable to build a model in order to predict those missing data.

4.3 Unbalanced Dependent Variable

4.3.1 SMOTE

SMOTE stands for the Synthetic Minority Over-Sampling Technique. We propose an oversampling method that oversamples a few classes by creating a "synthetic" sample instead of replacing oversampling. This approach was inspired by a technique for success in handwritten character recognition. The idea is to create additional training data by performing certain operations on real data. In this case, operations such as rotation and tilting are natural ways to disrupt training data. We generate synthetic examples in a less application-specific way by operating in the "feature space" instead of the "data space." The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining all the k minority class nearest neighbors. The values of the k nearest neighbors are randomly selected according to the required oversampling amount. The composite sample is generated as follows: the difference between the considered feature vector (sample) and its nearest neighbor. Multiply this difference by a random number between 0 and 1 and add it to the feature vector under consideration. This results in the selection of random points along with the line segments between two features (Chawla et al, 2002).

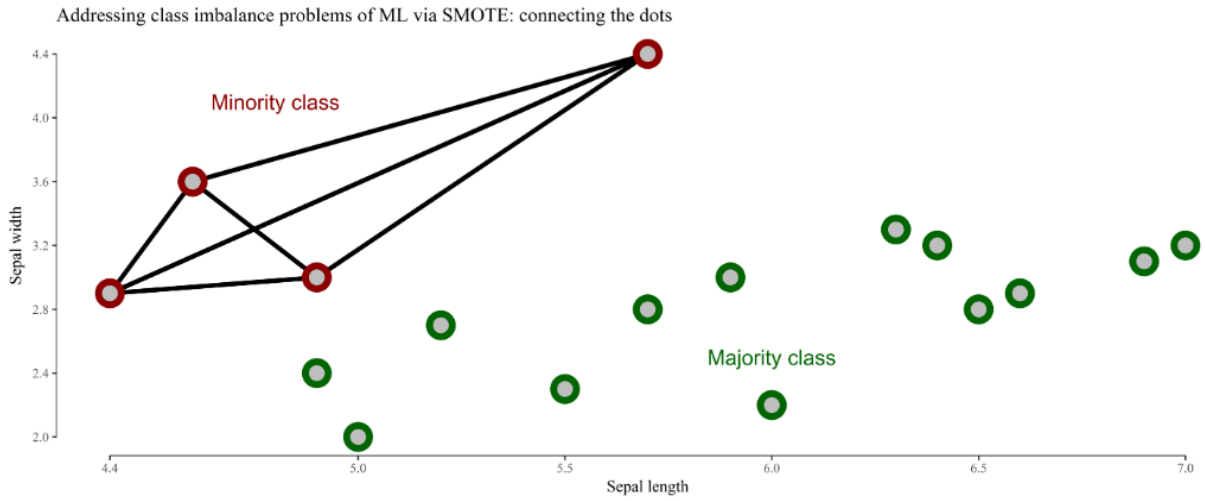


Figure 4.3.1: Connecting the Dots
(Chawla et al, 2002, pp321-357)

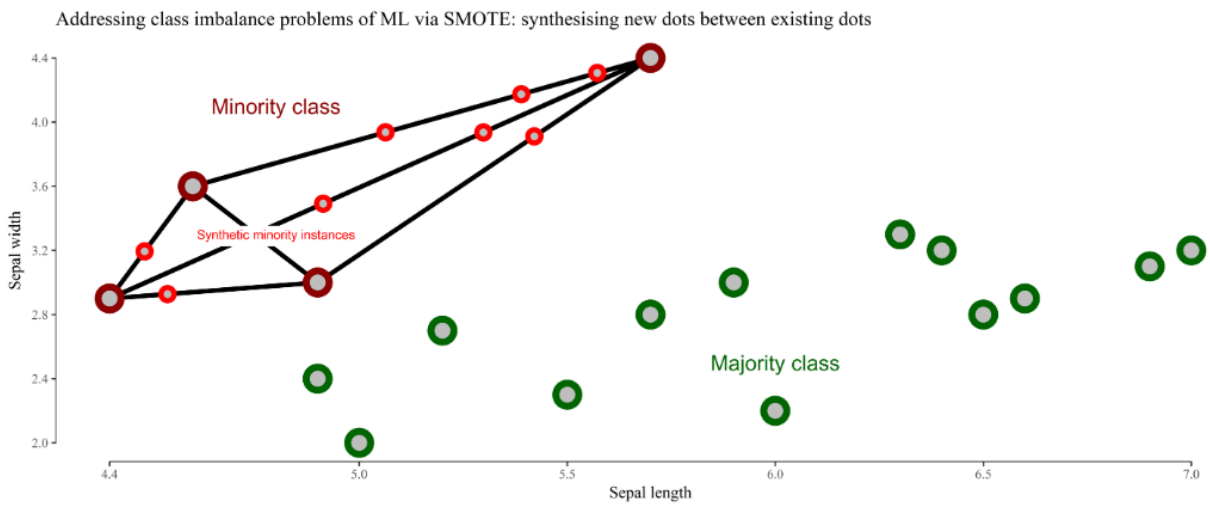


Figure 4.3.2: Synthesizing New Dots Between Existing Dots (Chawla, 2002, pp321-357)

For each instance x_i in the minority class, SMOTE searches its k nearest neighbors and one neighbor is randomly selected as x' (we call instances x_i and x', the seed sample). Then a random number δ between [0, 1] is generated. The new artificial sample x_{new} (from *Figure 4.3.1* to *Figure 4.3.2*) is created as:

$$x_{new} = x_i + (x' - x_i) \times \delta\,.$$

This approach effectively forces minority decision-making areas to become more common.

Note that SMOTE is bound to become a popular method for fraud detection, as it is particularly suited for large datasets where the proportion of fraudulent records is very small (highly unbalanced large datasets).

4.3.2 Under-Sampling

The majority class is under-sampled by removing samples randomly from the majority class until the minority class becomes a specified percentage of the majority class. This forces learners to experience varying degrees of under-sampling, and in a higher degree of under-sampling, minority groups have a greater percentage in the training set.

Again, this method has to be used with care in small data sets, where undersampling might fix the unbalce problem at the cost of producing training datasets that are too small to be fitted any model with an acceptable precision.

4.3.3 Combine SMOTE and Under-Sampling

Instead of using under-sampling or oversampling separately, we also consider combining these two sampling methods. By applying a combination of under-sampling and oversampling, the learner's initial bias for the minority class is reversed to a majority class. The classifier is learned on a data set that is influenced by “SMOTE” the minority and under-sampling the majority. In this thesis, we use different sampling methods to test each model in order to get a better result.

The first method is to “SMOTE” the minority class into 4,000 (around 1:2); the second method is to “SMOTE” the minority class into 9,000 (1:1); the third method is “SMOTE” the minority class into 9,000 and under-sample the majority class into 6,000 (3:2).

4.3.4 A Common Mistake

There is a common mistake implementing these methods; if you want to undersample or oversample your data you should not do it before cross-validation (Altini, 2015). Because if you get the minority class (“Fraud” in our case) and create the synthetic points before cross-validation, you have a certain influence on the "validation set" of the cross-validation process. But remember how cross-validation works; let us assume we are splitting the data into 5 batches, then 4/5 of the dataset will be the training set and 1/5 of the dataset will be the validation set. The test set should not be touched. For that reason, we should do the creation of synthetic data points before cross-validation, just like below:

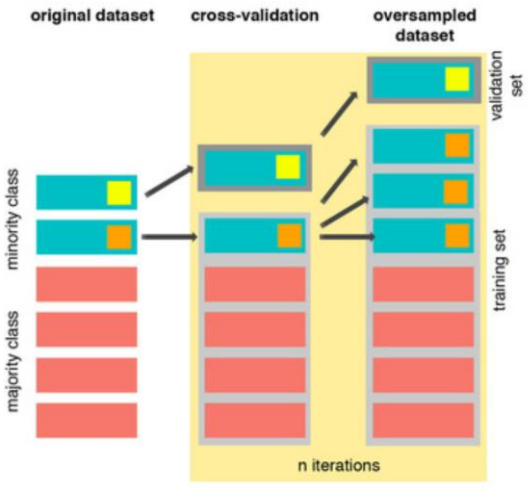


Figure 4.3.3: SMOTE Process
(Source: Altini, 2015, p3)

4.4 Results

4.4.1 Naïve Bayes Classifier

4.4.1.1 Confusion Matrix

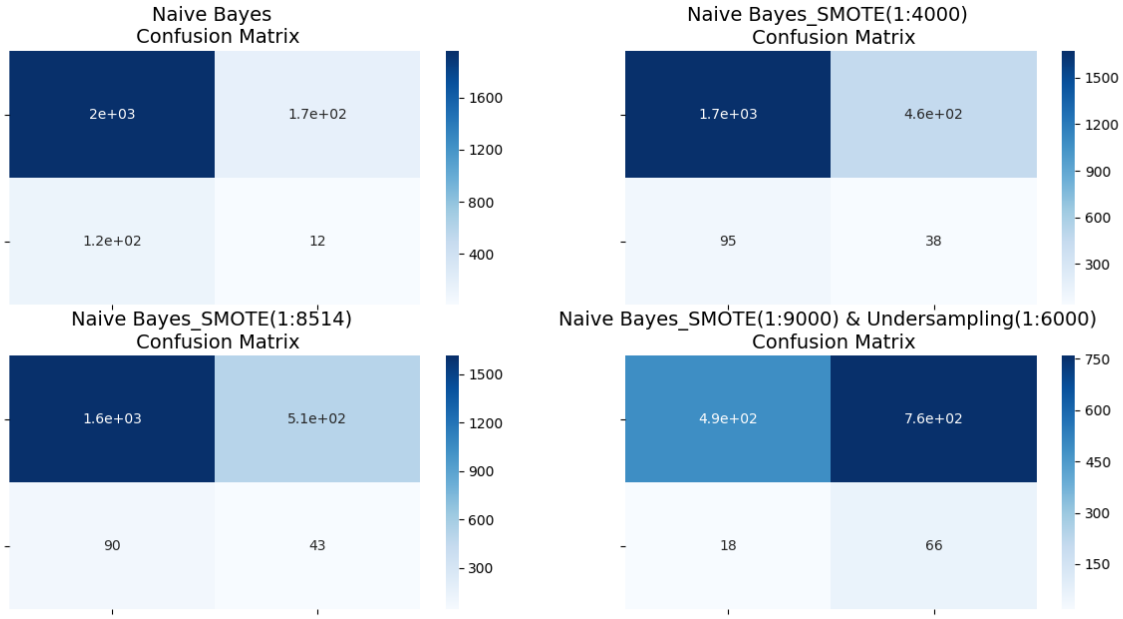


Figure 4.4.1: Confusion Matrix - Naive Bayes

From the above confusion matrix, we can clearly see the performance of an algorithm by numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Naïve Bayes	Recall	Precision
No SMOTE **No undersampling**	0.09	0.065
SMOTE (4,000) **No undersampling**	0.29	0.077
SMOTE (9,000) **No undersampling**	0.41	0.078
SMOTE (9,000) **Undersampling (6,000)**	0.79	0.08

Table 4.4.1: Recall and Precision Table - Naive Bayes

From the above table, it is clear that the *recall* will increase when the SMOTE sample increases. While at the same time, the *precision* will decrease if we just increase the SMOTE sample from 4,000 to 9,000. If we combine the SMOTE and the undersampling methods, the *recall* improves substantially from 0.41 to 0.79, but the *precision* almost keeps the same. In terms of the combination of SMOTE (9,000) and undersampling (6,000), a *recall* of 0.79 means 79% fraudulent claims can be detected; a *precision* of 0.08 means among all the predicted fraudulent claims, 8% claims were really fraudulent.

4.4.1.2 Discussion

4.4.1.2.1 Advantages

The naive Bayesian algorithm assumes that the dataset attributes are independent of each other, so the logic of the algorithm is very simple and the algorithm is relatively stable. When the data exhibits different characteristics, the classification performance of naïve Bayes is not greatly different. In other words, the naive Bayes algorithm is more robust and does not show much difference for different types of data sets. The naive Bayesian classification algorithm performs well when the relationships between dataset attributes are relatively independent.

4.4.1.2.2 Disadvantages

The condition of independence of attributes is also a disadvantage of the naive Bayes classifier. The independence of dataset attributes is difficult to satisfy in many cases, because the attributes of datasets are often related to each other. If such problems occur in the classification process, the classification performance will be greatly affected.

4.4.2 Logistic Regression

4.4.2.1 Check the Assumptions

4.4.2.1.1 Continuous Independent Variables (IVs) being Linearly Related to the LOG ODDS

Logistic regression does not require continuous independent variables (IVs) to be linearly related to dependent variables (DVs). But it requires the continuous IVs to be linearly related to the log odds of the DVs. One way to test this is to use the graph and look for an *S-shaped curve*. Sometimes the *S-shaped curve* will not be obvious. The figure should have a flat or flattish top and bottom with an increase or decrease in the middle.

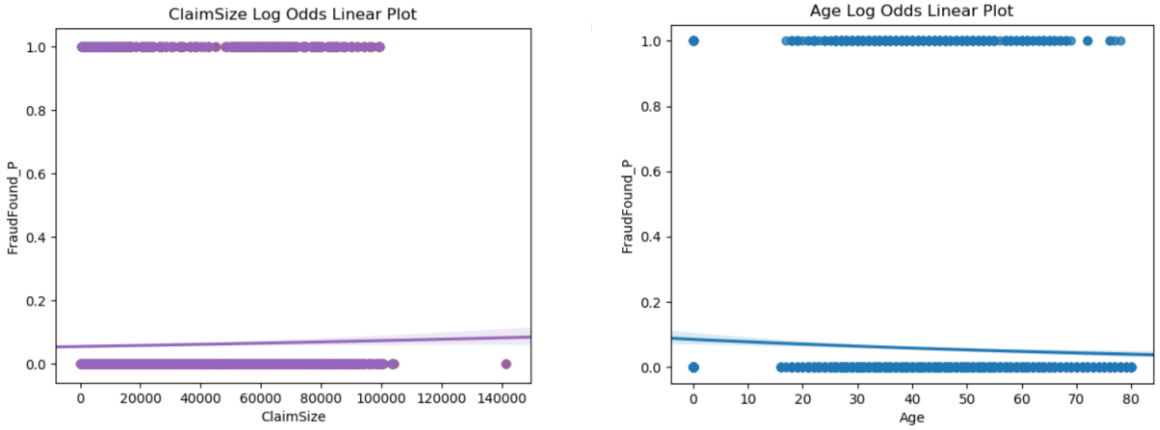


Figure 4.4.2: Curve of ClaimSize and Age

So, from the output, it is obvious that both "ClaimSize" and "Age" are linearly related to the dependent value ("FraudFound_P"), which satisfies the first assumption.

4.4.2.1.2 Absence of Multicollinearity

A simple approach to check multicollinearity is to use the correlation matrix to find any highly correlated variables. If there are variables that are highly correlated, then we need to drop one of them because they are measuring the same or similar things.

From *Figure 4.4.2*, we can see there is some multicollinearity among the variables, such as

"MonthClaimed" and "Month", "MaritalStatus_alone" and "AgeOfVehicle_year", "MaritalStatus_alone" and "AgeOfPolicyHolder". What is needed here is to delete one of these variables.

4.4.2.1.3 Lack of Outliers (Logistic Regression)

The assumption of lack of outliers is an easy one to check. One can get a feel of this with the descriptive statistics provided by the "*.describe()*" function in R. It is also very easy to check for outliers by using a box plot. Since there is a large difference between the values used to measure ClaimSize and Age, two separate box plots are generated.

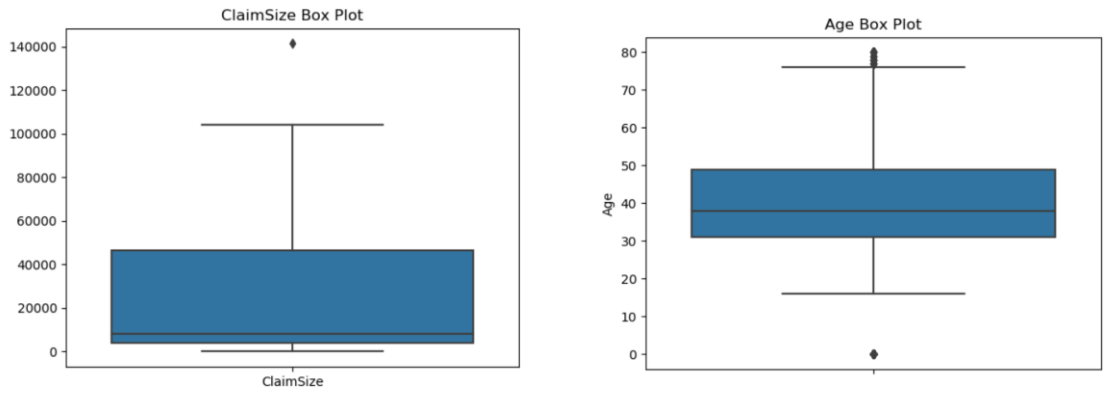


Figure 4.4.3: Box Plot of ClaimSize and Age

From the two outputs, we can see that there are some outliers in both "ClaimSize" and "Age". For the variable "ClaimSize", there is one claim at about $140, 000, which is much larger than the most claims. But we cannot directly delete this variable, because some “ClaimSize” can be much bigger than the mean value, which is reasonable. In terms of "Age", we can see there are some points around 0. But due to the common sense, we know it is impossible that the age of a driver is about 0, so we could judge these points as outliers and then delete them.

4.4.2.2 Results

4.4.2.2.1 ROC Curve

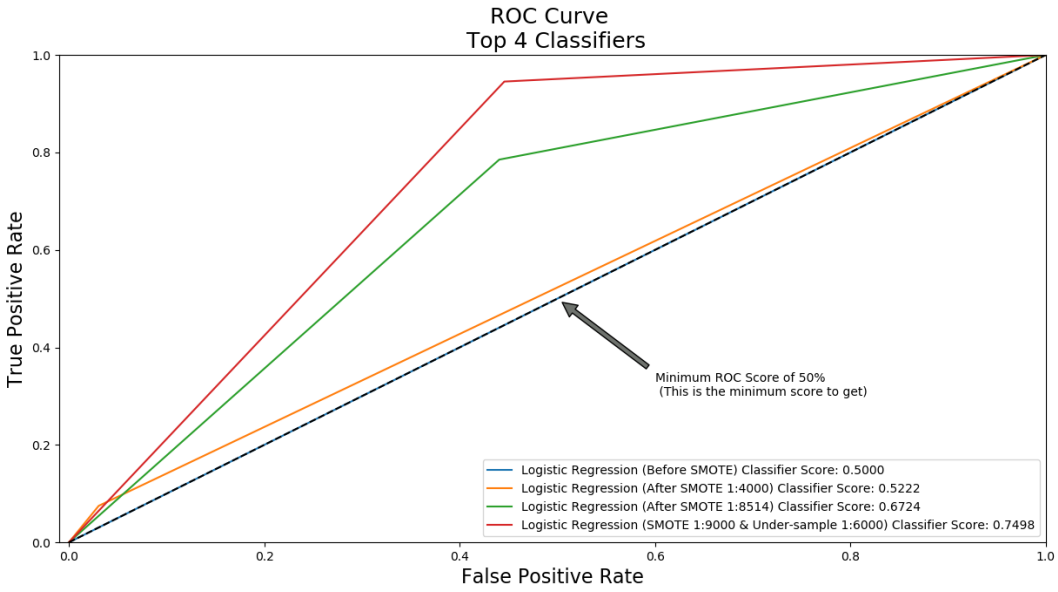


Figure 4.4.3: ROC Curve - Logistic Regression

From the ROC Curve of logistic regression, we can see that for the imbalanced dataset, the overall accuracy rate of the combination of SMOTE and undersampling is the highest; while the sampling method without SMOTE and undersampling is the lowest.

4.4.2.2.2 Confusion Matrix

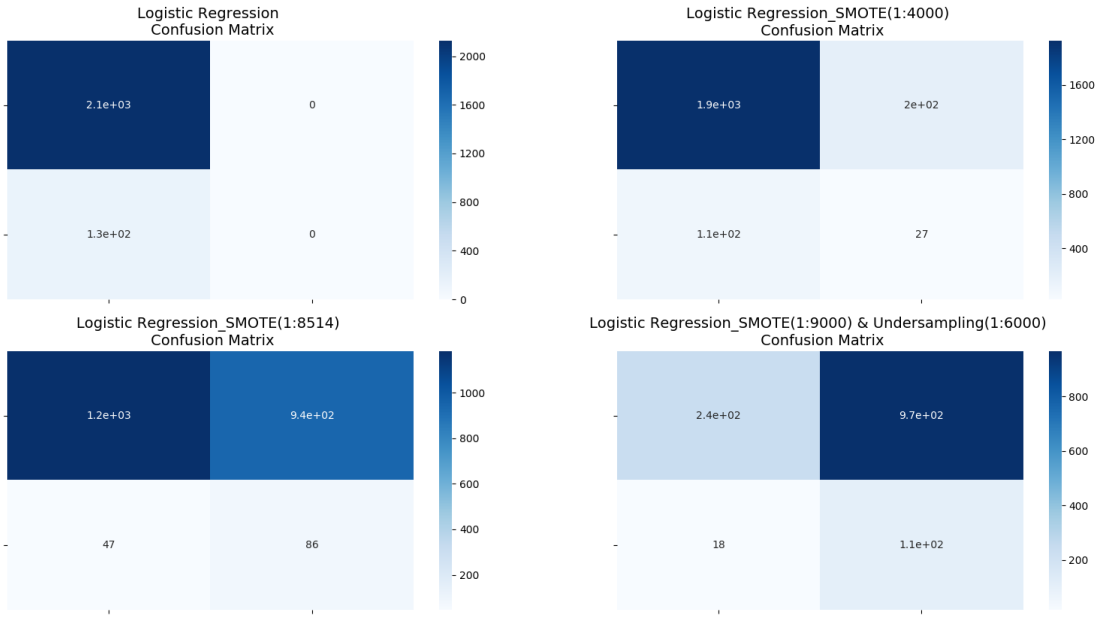


Figure 4.4.4: Confusion Matrix - Logistic Regression

From the above confusion matrix, we can clearly see the performance of an algorithm as summarized in a few numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Logistic Regression	Recall	Precision
No SMOTE **No undersampling**	0.0	0.0
SMOTE (4,000) **No undersampling**	0.203	0.117
SMOTE (9,000) **No undersampling**	0.647	0.08
SMOTE (9,000) **Undersampling (6,000)**	0.856	0.1

Table 4.4.2: Recall and Precision Table - Logistic Regression

From *Table 4.4.2*, it is clear that the *recall* will increase when the SMOTE sample increases. While at the same time, the *precision* will decrease if we just increase the SMOTE. But, if we combine SMOTE and the undersampling methods, then both *recall* and *precision* improve a lot. In terms of the combination of SMOTE (9,000) and Undersampling (6,000), *recall* (0.856) means 85.6% fraudulent claims can be detected; while *precision* (0.1) means among all the predicted fraudulent claims, 10% claims are really fraudulent. In practice, this model can be useful.

4.4.2.3 Discussion

4.4.2.3.1 Advantages

It is a widely used technology because it is very efficient, does not require too much computing resources, it is highly interpretable, it does not require scaling input; and it does not require any tuning. Like linear regression, logistic regression is very efficient when you remove attributes that are not related to the response variable and attributes that are very similar (correlated) to each other. Therefore, data cleaning plays an important role in the performance of logistic regression. What is more, logistic regression is very easy to implement and trains very effectively.

4.4.2.3.2 Disadvantages

A disadvantage of logistic regression is that it cannot solve non-linear problems since its decision surface is linear. Also, to use logistic regression, the data should satisfy many assumptions, that in practice, are either not be satisfied or difficult to verify.

4.4.3 Random Forest

4.4.3.1 ROC Curve

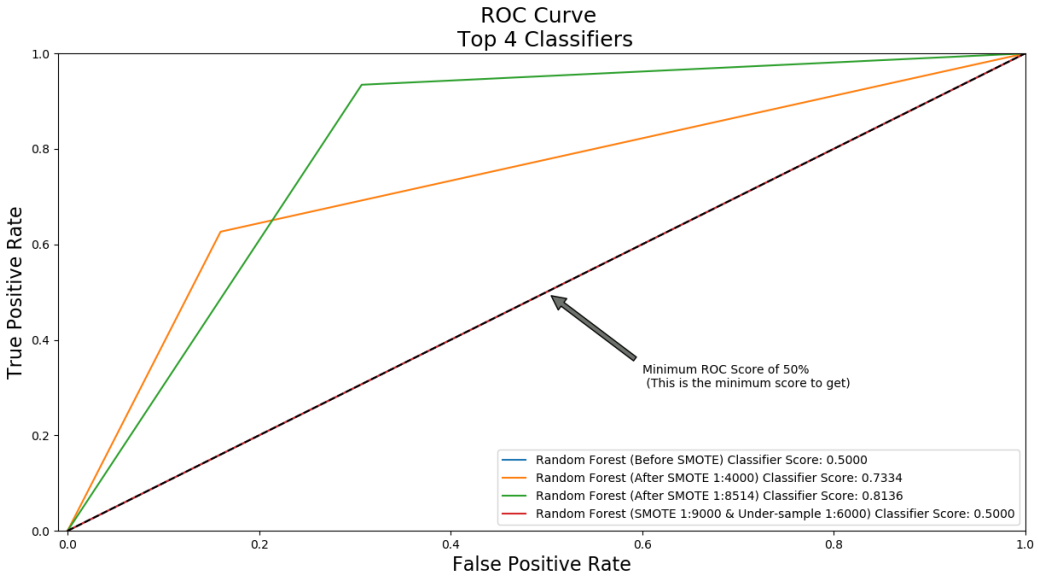


Figure 4.4.5: ROC Curve – Random Forest

For the Random Forest classification method, the overall accuracy rate of SMOTE (9,000) is the best and is even better than that of the combination of SMOTE (9,000) and undersampling (6,000). The performance of SMOTE (4,000) and non-SMOTE methods are poor, with an accuracy rate of 0.5.

4.4.3.2 Confusion Matrix

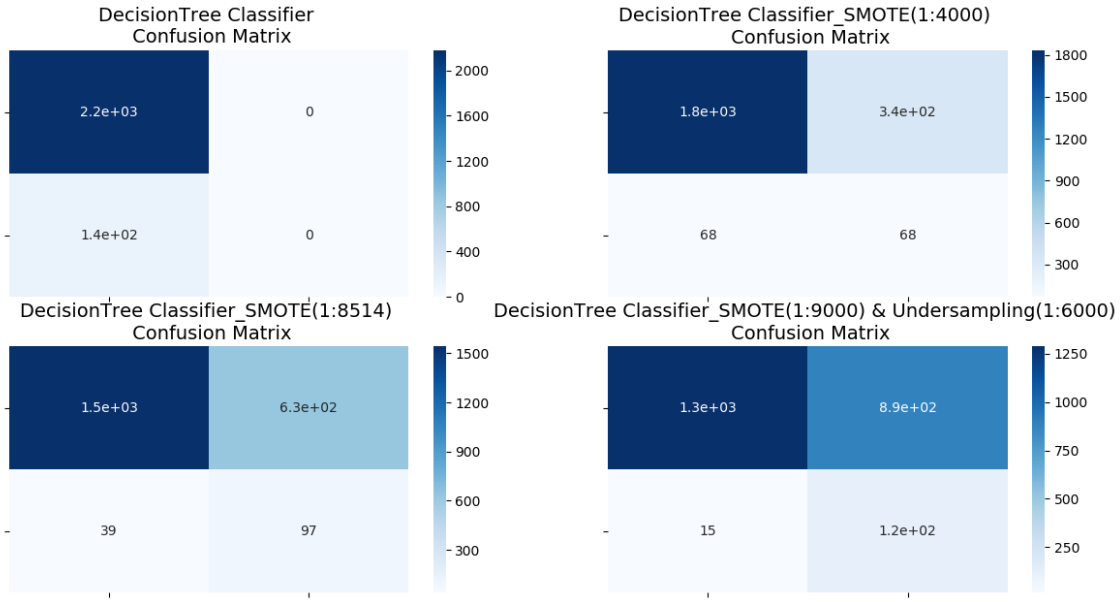


Figure 4.4.6: Confusion Matrix – Random Forest

From the above confusion matrix, we can also clearly see the performance of the random forest :

Random Forest	Recall	Precision
No SMOTE **No undersampling**	0.0	0.0
SMOTE (4,000) **No undersampling**	0.5	0.165
SMOTE (9,000) **No undersampling**	0.713	0.133
SMOTE (9,000) **Undersampling (6,000)**	0.89	0.12

Table 4.4.3: Recall and Precision Table – Random Forest

From the *Table 4.4.3*, it is clear that the *recall* will also increase when the SMOTE increases. While at the same time, the *precision* will decrease from 0.165 to 0.133 if we just increase SMOTE.

If we combine the SMOTE and undersampling methods, then the *recall* improves from 0.713 to 0.89 and *precision* decreases from 0.133 to 012. In terms of the combination of SMOTE (9,000) and undersampling (6,000), *recall* (0.89) means 89% fraudulent claims can be detected; *precision* (0.12) means among all those predicted fraudulent claims, 12% claims are really fraudulent. In practice, the model can also be useful.

4.4.4 Gradient Boosting

4.4.4.1 ROC Curve

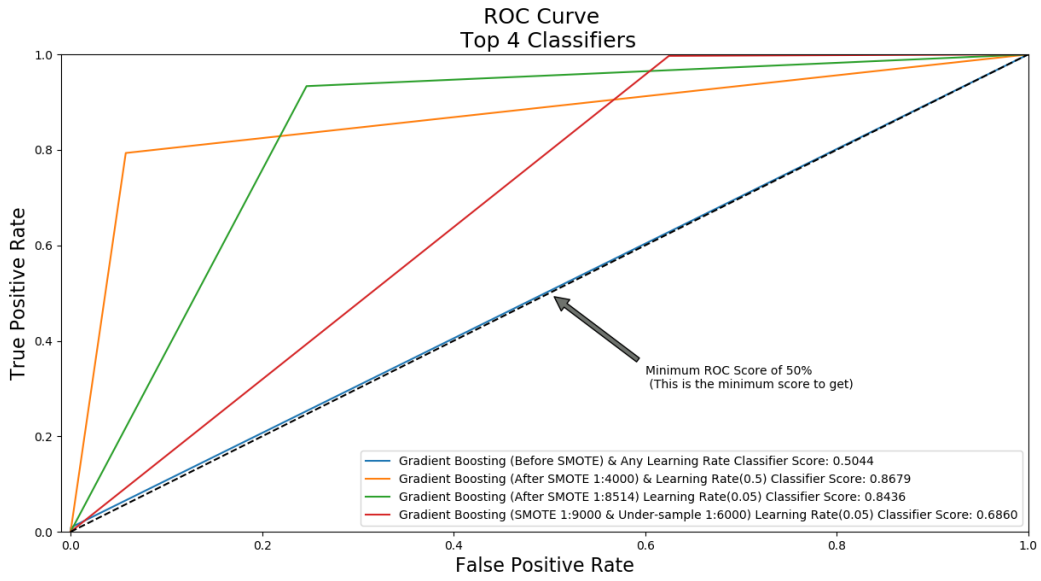


Figure 4.4.7: ROC Curve - Gradient Boosting

For the random forest classification method, the overall accuracy rate of SMOTE (4,000) is the best and is even better than that of the combination of SMOTE (9,000), undersampling (6,000) and learning rate (0.05), with classifier score being 0.8679. The performance of non-SMOTE method is poor, with an accuracy rate of about 0.5. For gradient boosting, the combination of SMOTE (9,000), undersampling (6,000) and learning rate (0.05) do not perform that well, with the classifier score just being 0.6860.

4.4.4.2 Confusion Matrix

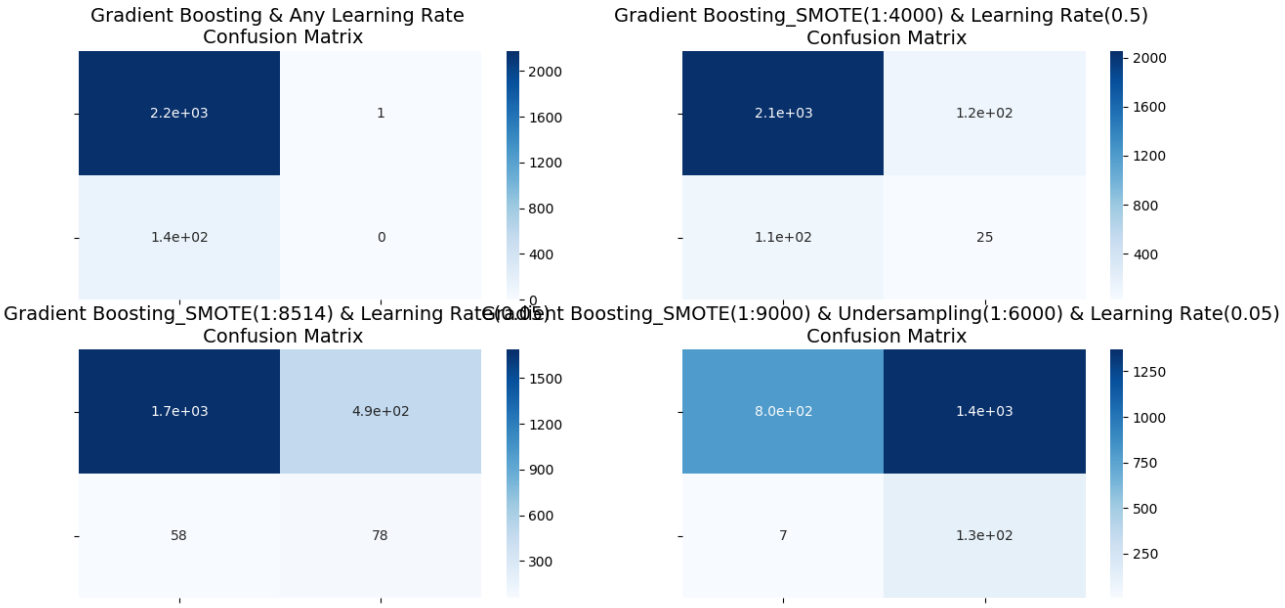


Figure 4.4.8: Confusion Matrix - Gradient Boosting

From the above confusion matrix, we can also clearly see the performance of an algorithm by numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Gradient Boosting	**Recall**	**Precision**
No SMOTE **Any learning rate**	0.0	0.0
SMOTE (4,000) **Any learning rate (0.5)**	0.18	0.17
SMOTE (9,000) **Any learning rate (0.5)**	0.57	0.14
SMOTE (9,000) **Undersampling (6,000)** **Any learning rate (0.5)**	0.95	0.09

Table 4.4.4: Recall and Precision Table - Gradient Boosting

From *Table 4.4.4*, it is clear that *recall* will also increase when the SMOTE increases. At the same time, the *precision* will increase from 0 to 0.17 if we just increase SMOTE from 0 to 4,000. But different from the above methods, the *precision* will decrease from 0.17 to 0.4 by increasing the SMOTE from 4,000 to 9,000. If we combine the SMOTE and undersampling methods, then the *recall* has been improved significantly from 0.57 to 0.95 and *precision* decreases from 0.14 to 0.09. In terms of combination of SMOTE (9,000) and undersampling (6,000), *recall* (0.95) means 95% fraudulent claims can be detected; *precision* (0.09) means among all those predicted fraudulent claims, 9% claims are really fraudulent. In practice, the model is also useful.

4.5 Conclusion

In general, random forest and gradient boosting classifiers are easy to train. We do not need to consider missing values or independence; while for naïve Bayes and logistic regression, we need to care about these conditions.

What is more, in order to compare different classification methods, we just focus on the combination of SMOTE (9,000) and undersampling (6,000), because in this situation, the classification performance is the best. As shown below:

	Recall	Precision
Naive Bayes	0.79	0.08
Logistic Regression	0.856	0.1
Random Forest	0.89	0.12
Gradient Boosting	0.95	0.09

Table 4.5.1: Overall Recall and Precision Table

From *Table 4.5.1,* it is clear to see that for the combination of SMOTE (9,000) and undersampling (6,000), gradient boosting has the highest *recall*, which means it can detect the most percentage of the fraudulent claims. Although the random forest method has better *precision* (percentage of predicted fraudulent claims that are actual fraudulent claims), gradient boosting is also the best classification method, because compared to random forest, the *recall* of gradient boosting increases by 0.06 (from 0.89 to 0.95), but the *precision* just decreases 0.03 (from 0.12 to 0.09), which is acceptable. Remember that *recall* meaures the percentage of fraudulent claims that can

be detected, which is the quantity of interest here, to be maximized. The *precision* error is a less important decision variable in this application.

In terms of each classification method, we can see that the *recall* will be improved by increasing the SMOTE index, but the *precision* will decrease at the same time.

Even though the *recalls* of gradient boosting and random forest are very good, there are still some things to improve in fraudulent claim detection. From *Table 4.5* above, we see that the precision is very low, which means many real-claim customers will be bothered when we classify them as fraudulent claims. Therefore, we need to find a way to improve recall and precision simultaneously.

4.6 Original Idea

Reaching this step, we can also use a combination of clustering and classification models. Starting form the SMOTE (9,000) step, the first method is to use *k*-means clustering to divide the whole dataset into three clusters, and only then apply the proper classification model to these three clusters. By using this method, the performance can also be improved. However, this method cannot be applied to small datasets, because there would not be enough observations in all small clusters to train the classification model.

What is more, the method of changing the order of SMOTE (9,000) and *k*-means clustering has also been tested, which means using *k*-means clustering to group the data and then using SMOTE to resample each cluster. The results were almost the same. After checking the data in each clustering, by using SMOTE (9,000) and *k*-means clustering, all these three clusters are very balanced (around 1:1, 5:4 and 4:5). Maybe this is why the order does not change the final results.

4.6.1 Performances of Different Models in Three Clusters

	first cluster	second cluster	third cluster
random forest	0.846938776	0.72	0.684210526
gradient boosting	0.571428571	0.96	0.842105263
logistic regression	0.785714286	0.88	0.789473684
naïve bayes	0.357142857	0.92	0.789473684

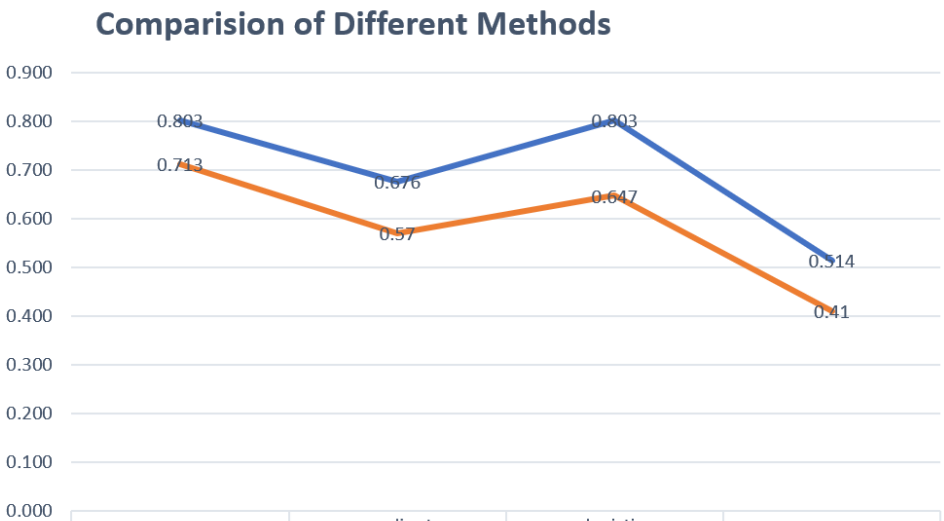
CLUSTERS

Table 4.6.1: Overall Recall Table for Clustered Data

From the table above, we can see that in the first cluster, random forest is the best method, with *recall* being 0.8469, while in second and third clusters, gradient boosting is the best, with recall being 0.96 and 0.84 respectively.

4.6.2 The Effect of Using Clustering

But when comes to the overall performance of different models, it is clear to see from the below graph that randon forest and logistic regression are the best. By using the combination method of SMOTE and clustering, the performances (*recall*) of all these four models have been improved a lot.



	random forest	gradient boosting	logistic regression	naïve bayes
recall - SMOTE & clustering	0.803	0.676	0.803	0.514
recall - SMOTE	0.713	0.57	0.647	0.41

Table 4.6.2: Overall Recall Table For the Best Methods

What is more, gradient boosting is almost the best model in all situations, but in the combination method of clustering and classification models, it is not an ideal model, because the theory of gradient boosting is to build a tree first, and then iteratively build other trees to improve the error of the previous tree, which means the number of observations will influence the performance of gradient boosting. Dividing the dataset into clusters means smaller number of observations in each cluster, which will lead to a worse performance.

Chapter 5

Dataset 2 Description and Manipulation

5.1 Description

5.1.1 Dependent Variable

The second dataset analyzed is also about car insurance claims, which can be found at Kaggle (url: https://www.kaggle.com/roshansharma/insurance-claim). It consists of 1,000 observations with 39 variables. The response variable named the "fraud_reported" is either *1* for a driver who filed a fraudulent claim or *0* for car insurance claims that are legitimate. There are 38 explanatory variables with information on each driver (sex, age, education level, claim amount or claim time, for example).

The dependent variable is binary (*1* or *0*), and it is clear from the following figure that this dependent variable is somewhat imbalanced (24.7% if *1*s and 75.3% if *0*s). So for this dataset, a transformation of the dependent variable to correct the imbalance may not be necessary, which means the result without the SMOTE method may be acceptable.

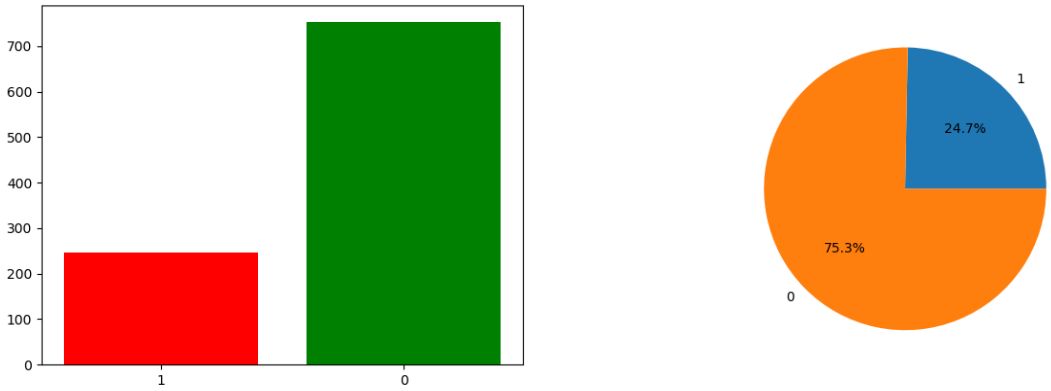


Figure 5.1.1: Bar Chart and Pie Chart of the Response Variable

5.1.2 Correlation Matrices

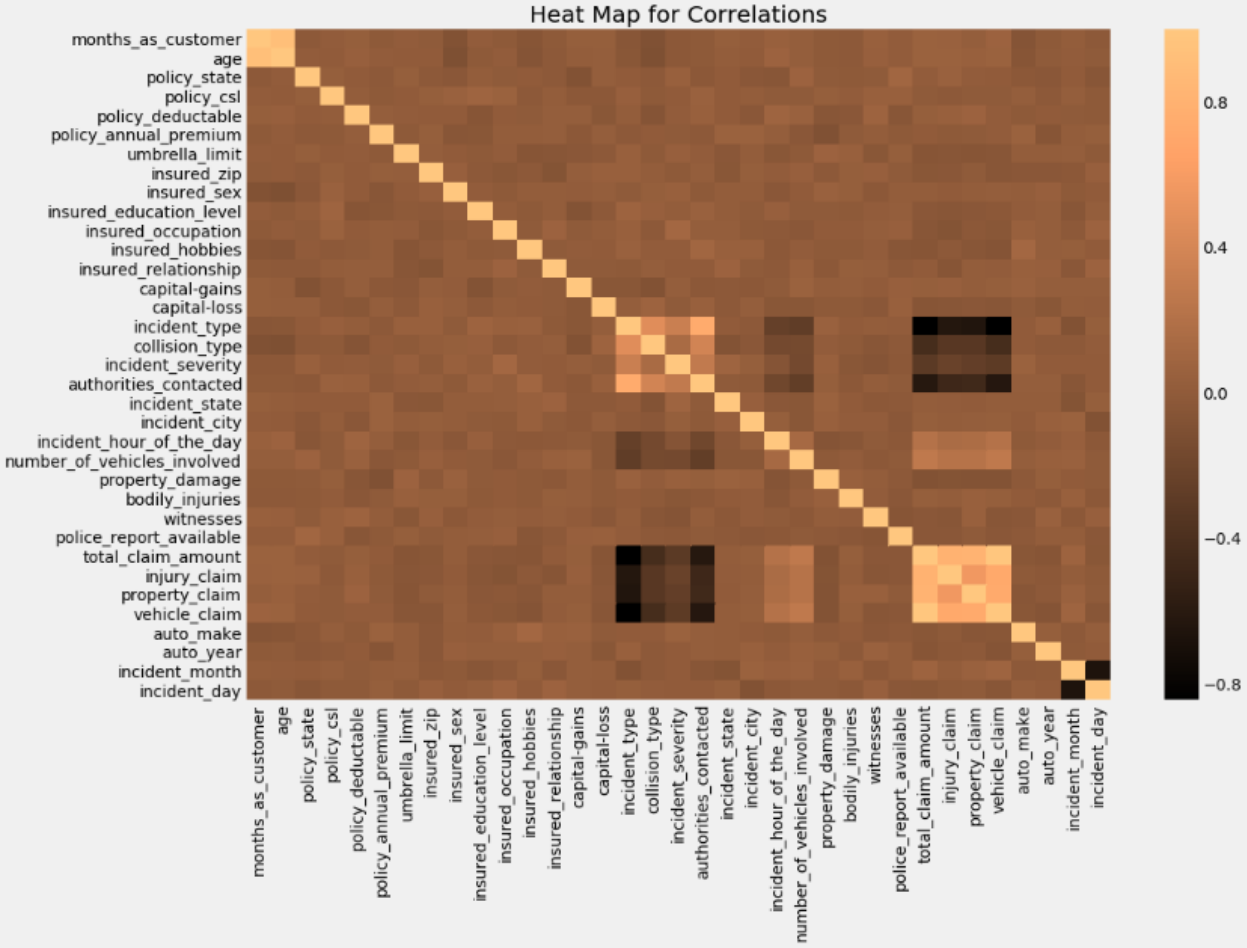


Figure 5.1.2: Correlation Matrix

From the output, we can easily see the "incident_type" tends to have negative a negative relationship with "total_claim_amount", "vehicle_claim", "injury_claim" and "property_claim"; while the "vehicle_claim" has a positive relationship with "total_claim_amount", "vehicle_claim" and "property_claim". So, when we use a logistic regression model, it is important to check these instances of multicollinearity.

5.2 Data Manipulations

In this section, we detail some common data manipulations that are carried out for all the analyses, plus any specific manipulation required for certain. To be able to perform cross-validation, the dataset was first divided into two parts. The training dataset contains 700 observations, and the

testing dataset contains 300 observations, for a total of 1,000 rows, which is not a large dataset. Compared with the first dataset, we need to increase the percentage of testing set (30%).

5.2.1 Features Selection

Before fitting any models, it is important to check the relation between all the features, for two main reasons:

• If two features are highly linearly correlated, the data may show multicollinearity effects, especially for the logistic regression.

• By reducing the number of input variables, we may have fewer parameters requiring tuning when fitting the model and consequently reduce the computational load.

5.2.2 Standardization and Scaling

Since our features are expressed in different measurement scales, we standardize or scale the following features:

- fraud_reported: Change to 1 (Yes) and 0 (No).
- Sex: Change to 1 (Male) and 0 (Female).
- MaritalStatus: Change to 1 (Single) and 0 (Married).
- Collision_type: Replace the missing data with the most common collision type (Back Collision).
- Witness: Change to 1 (Yes) and 0 (No).
- Property_damage: Replace the missing data with “No”, because we just treat missing data as no response for the property damage.
- Police_report_available: Replace the missing data with “No”, because we just treat missing data as no police report.
- Insured_education_level, incident_type, insured_relationship, insured_hobbies, etc.: Target Encoding.
- Policy_number, policy_bind_date, incident_date, incident_location: delete these variables because they do not contain any useful information for the fraud detection.

5.2.3 Missing Data

In terms of the missing data, as mentioned before, there are also several standards:

- If the data is missing randomly and more than 70% data of certain feature is missing, then delete this feature directly;
- If the feature with missing data has a trend based on time, then fill in the missing data by time;
- Check if those features with missing data have some relationships with other features. If so, treat other related variables as independent variables and treat the missing data as the dependent variable to build a model in order to predict those missing data.

5.2.4 Target Encoding

The second dataset contains some categorical variables that have more than two categories. In this case, instead of using one-hot encoding (used in Dataset 1) method to deal with them, target encoding method is more convenient and efficient. Because if there are many categorical variables having multiple categories, one-hot encoding method will produce many columns, which may lead to memory issues.

The main idea of target encoding method is to average the value by category. For example, there is a categorical variable $\boldsymbol{x}$ and a dependent variable $\boldsymbol{y}$ ($\boldsymbol{y}$ can be binary or continuous). For each distinct element in x_i, we can replace each x_i by computing the average of the corresponding values in $\boldsymbol{y}$. All of this calculation is pretty easy in "***pandas***" library of Python. This means it can help to produce categorical variables with little effort.

5.2.5 Combine SMOTE and Under-Sampling

By applying a combination of undersampling and oversampling, the learner's initial bias for the minority class is reversed to a majority class. The classifier is learned on a data set that is influenced by "SMOTE" the minority and under-sampling the majority.

In this thesis, we use different sampling methods to test each model in order to get a better result. The first method is to "SMOTE" the minority class into 376 (around 1:2); the second method is to "SMOTE" the minority class into 753 (1:1); the third method is "SMOTE" the minority class into 753 and undersample the majority class into 502 (3:2).

5.3 Results

5.3.1 Naïve Bayes Classifier

5.3.1.1 ROC Curve

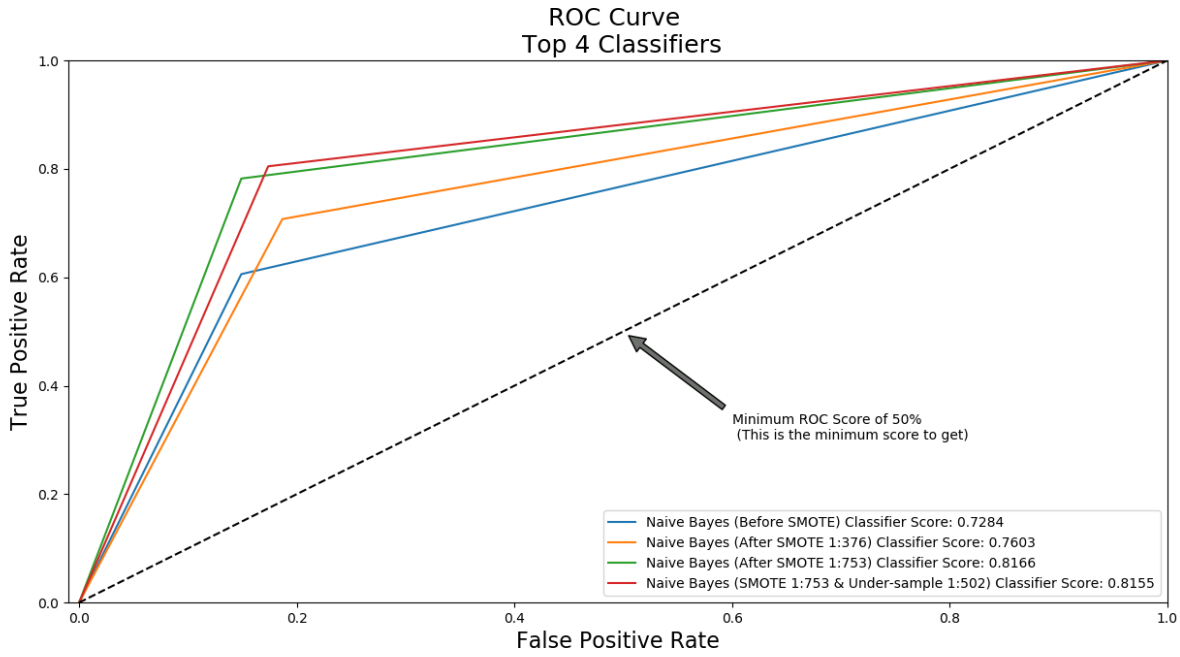


Figure 5.3.1: ROC Curve – Naïve Bayes

From the ROC Curve of logistic regression, we can see that for the imbalanced dataset, the overall accuracy rate of the combination of SMOTE and undersampling is the highest; while the sampling method without SMOTE and undersampling is the lowest.

5.3.1.2 Confusion Matrix

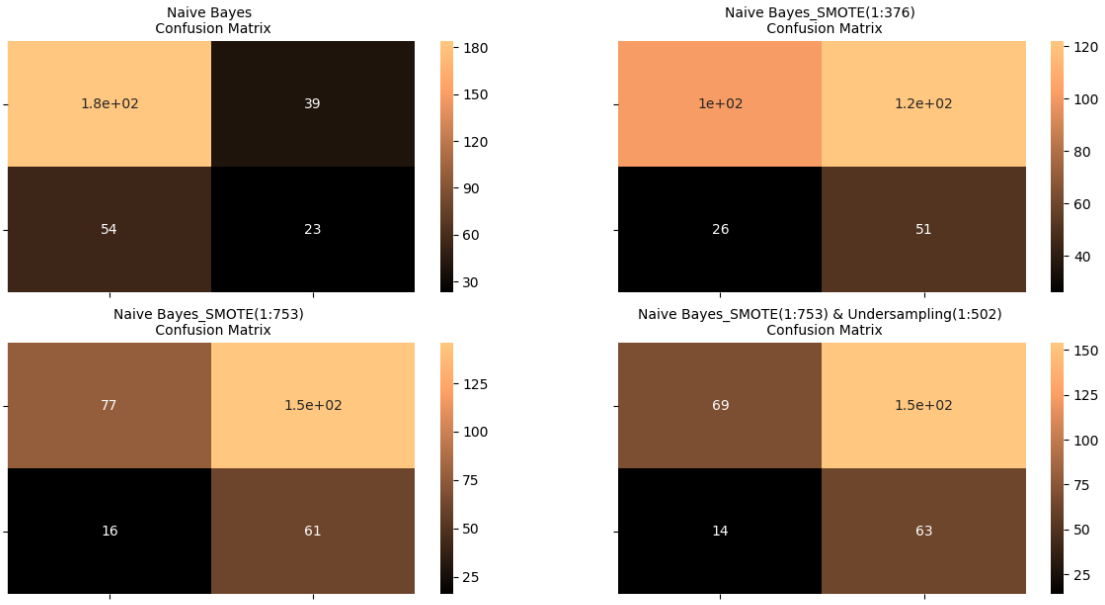


Figure 5.3.2: Confusion Matrix – Naïve Bayes

From the above confusion matrix, we can clearly see the performance of an algorithm as summarized in a few numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Naïve Bayes	Recall	Precision
No SMOTE **No undersampling**	0.30	0.37
SMOTE (376) **No undersampling**	0.66	0.29
SMOTE (753) **No undersampling**	0.79	0.29
SMOTE (753) **Undersampling (502)**	0.82	0.29

Table 5.3.1: Recall and Precision Table - Naive Bayes

From the above table, it is clear that the *recall* will increase when the SMOTE sample increases. While at the same time, the *precision* will decrease from 0.37 to 0.29 and then remain the same if we just increase the SMOTE sample from 376 to 753. But if we combine the SMOTE and the undersampling methods, the *recall* improves a lot, and the *precision* also does not change. In terms of the combination of SMOTE (753) and undersampling (502), *recall* (0.84) means 82% fraudulent claims can be detected; *precision* (0.29) means among all the predicted fraudulent claims, 29% claims were really fraudulent.

5.3.2 Logistic Regression

5.3.2.1 Check the Assumptions

5.3.2.1.1 Continuous Independent Variables (IVs) being Linearly Related to the LOG ODDS

Logistic regression does not require continuous independent variables (IVs) to be linearly related to dependent variables (DVs). But it requires the continuous IVs be linearly related to the log odds of the DVs. One way to test this is to use the graph and look for an *S-shaped curve*. Sometimes the *S-shaped curve* will not be obvious. The figure should have a flat or flattish top and bottom with an increase or decrease in the middle.

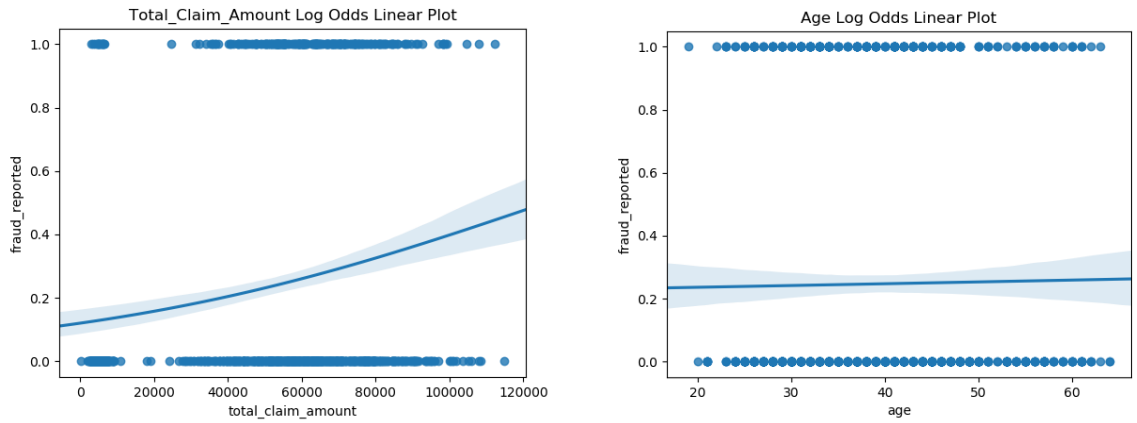


Figure 5.3.3: S-shaped Curve of Total Claim Amount and Age

So, from the output, it is obvious that both "total_claim_amount" and "age" are linearly related to the dependent value ("fraud_reported"), which satisfies the first assumption.

5.3.2.1.2 Absence of Multicollinearity

A simple approach to check multicollinearity is to use the correlation matrix to find any highly correlated variables. If there are variables that are highly correlated, then we need to drop one of them because they are measuring the same or similar things.

From *Figure 5.1.2*, we can see there is some multicollinearity among the variables, such as "incident_type" and "total_claim_amount". What we need to do is delete "total_claim_amount" because the sum of "injury_claim", "property_claim" and "vehicle_claim" equals to "total_claim_amount". So, we can delete "total_claim_amount" without losing any information.

5.3.2.1.3 Lack of Outliers (Logistic Regression)

The assumption of lack of outliers is an easy one to check. One can get a feel of this with the descriptive statistics provided by the *".describe()"* function in R. But it is also very easy to check the outliers by using a box plot. Since there is a huge difference between the values used to measure "vehicle_claim" and "injury_claim", two separate box plots are generated. And from the two outputs, we can see that there is no outliers in "vehicle_claim" and "injury_claim".

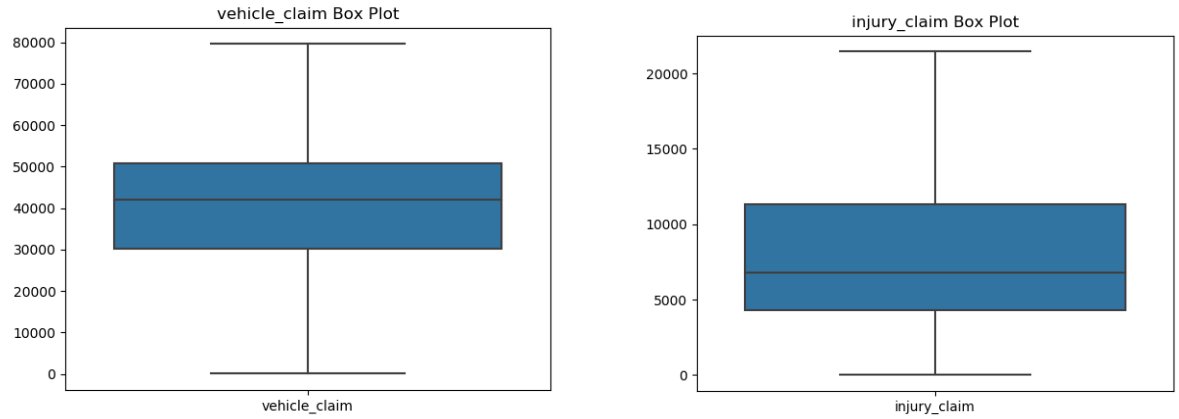


Figure 5.3.4: Box Plot of Vehicle_Claim and Injury_Claim

5.3.2.2 Results

5.3.2.2.1 ROC Curve

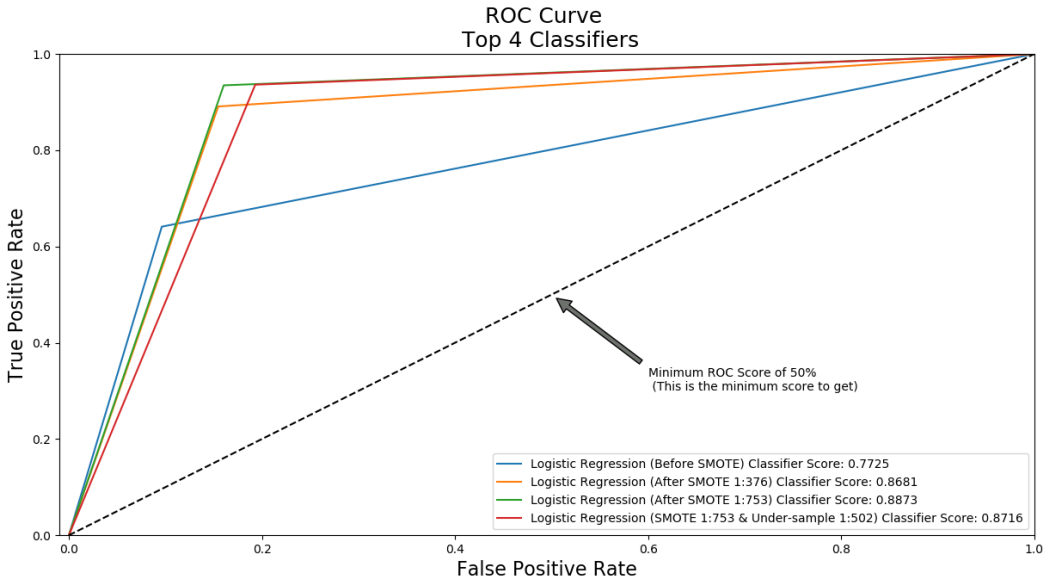


Figure 5.3.5: ROC Curve - Logistic Regression

From the ROC Curve of logistic regression, we can see that for the imbalanced dataset, the overall accuracy rate of the SMOTE (1:753) is the highest and very close to the combination of SMOTE and undersampling; while the sampling method without SMOTE and undersampling is the lowest.

5.3.2.2.2 Confusion Matrix

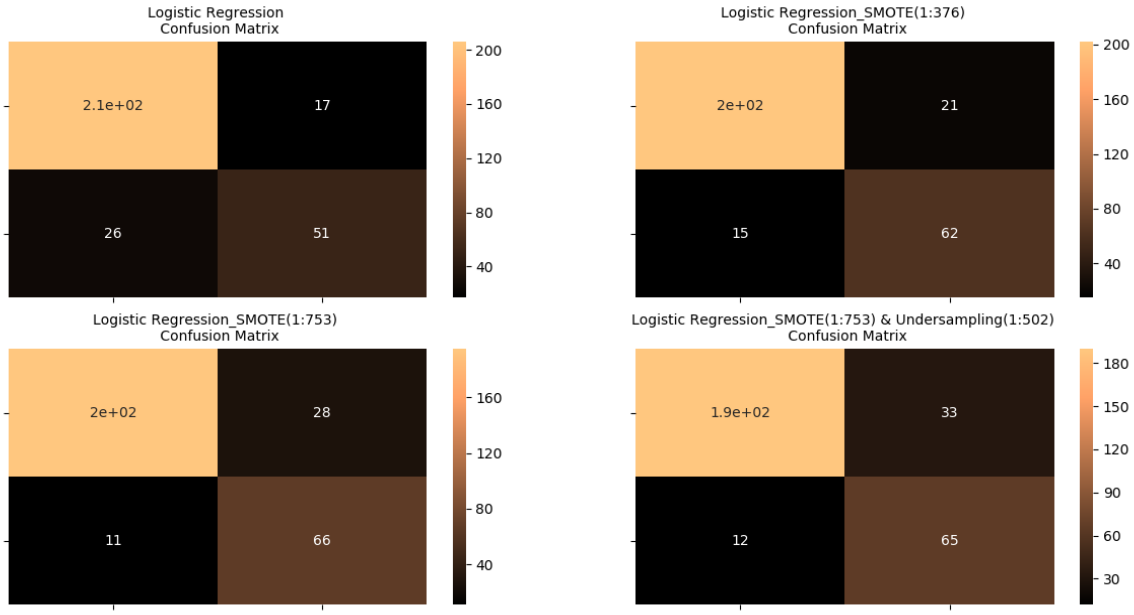


Figure 5.3.6: Confusion Matrix - Logistic Regression

From the above confusion matrix, we can clearly see the performance of an algorithm as summarized in a few numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Logistic Regression	Recall	Precision
No SMOTE **No undersampling**	0.66	0.75
SMOTE (376) **No undersampling**	0.81	0.74
SMOTE (753) **No undersampling**	0.86	0.70
SMOTE (753) **Undersampling (502)**	0.84	0.66

Table 5.3.2: Recall and Precision table – Logistic Regression

From the *Table 5.3.2*, it is clear that the *recall* will also increase when the SMOTE increases. But at the same time, the *precision* will decrease from 0.75 to 0.70 if we just increase the SMOTE from 0 to 753. But different from the above methods, the *precision* will decrease from 0.70 to 0.66 by using a combination method of SMOTE (753) and undersampling (502), while the *recall* just increases from 0.86 to 0.84. Then we can say, in this case, for logistic regression, the combination method does not work well. In this case, choosing SMOTE (753) is the best, because this method more sense in practice. The *recall* (0.86) means 86% fraudulent claims can be detected; *precision* (0.7) means among all those predicted fraudulent claims, 70% claims are really fraudulent.

5.3.3 Random Forest

5.3.3.1 ROC Curve

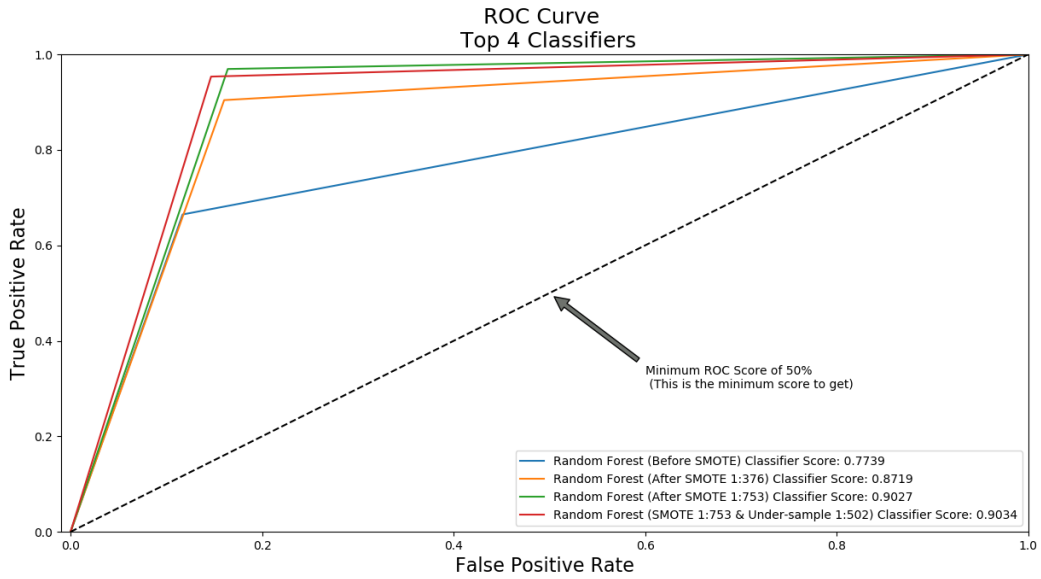


Figure 5.3.7: ROC Curve – Random Forest

For the random forest classification method, the overall accuracy rate of the combination of SMOTE (753) and undersampling (502) is the best, with *classifier score* of 0.5. The performance of raw dataset is poor.

5.3.3.2 Confusion Matrix

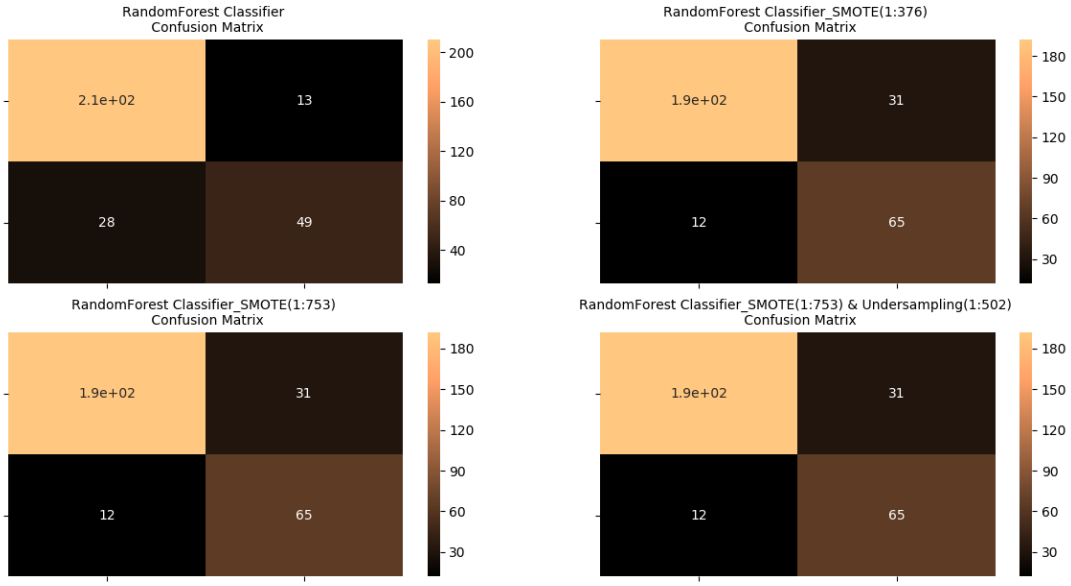


Figure 5.3.8: Confusion Matrix – Random Forest

From the above confusion matrix, we can also clearly see the performance of the random forest :

Random Forest	Recall	Precision
No SMOTE **No undersampling**	0.64	0.79
SMOTE (376) **No undersampling**	0.84	0.68
SMOTE (753) **No undersampling**	0.84	0.68
SMOTE (753) **Undersampling (502)**	0.84	0.68

Table 5.3.3: Recall and Precision Table – Random Forest

From the *Table 5.3.3*, it is clear that the *recall* will also increase from 0.64 to 0.84 and the *precision* will decrease from 0.79 to 0.68, and then remain unchanged when the SMOTE increases from 0 to 376 and then to 753, even though we combine the undersampling method at the same time. Maybe for the dataset with not that imbalanced dependent variable, when we use random forest model, the SMOTE ratios and undersampling methods do not have a lot influence to the result.

In terms of the combination of SMOTE (753) and undersampling (502), a *recall* (0.84) means 84% fraudulent claims can be detected; a *precision* (0.68) means among all those predicted fraudulent claims, 68% claims are really fraudulent. In practice, the model can also be useful.

5.3.4 Gradient Boosting

5.3.4.1 ROC Curve

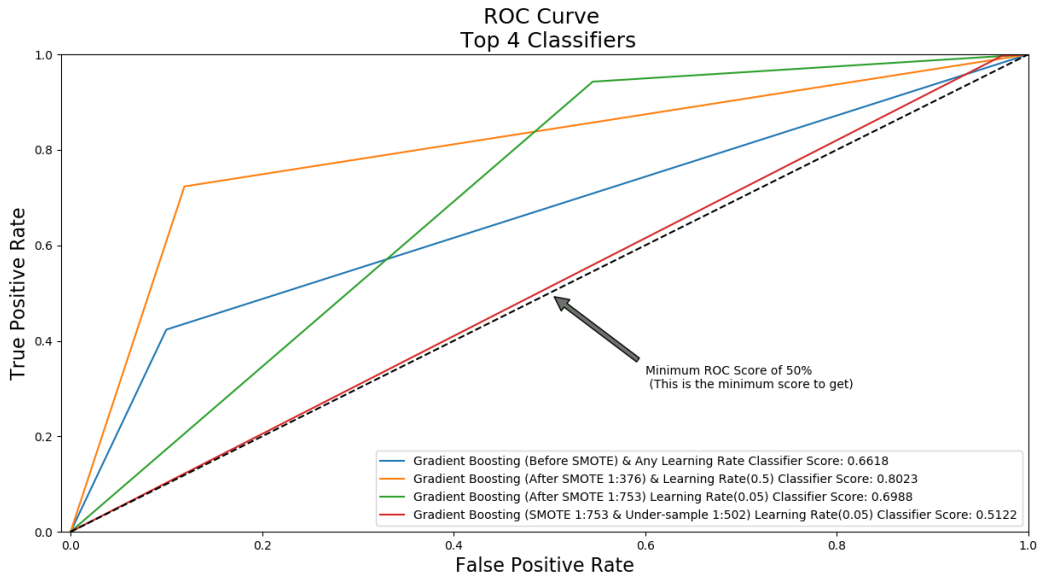


Figure 5.3.9: ROC Curve - Gradient Boosting

For the random forest classification method, the overall accuracy rate of SMOTE (376) is the, with classifier score being 0.8023. But for gradient boosting model, the performance of the combination of SMOTE (753), undersampling (376) and learning rate (0.05) is the lowest, with classifier score being only 0.5122.

5.3.4.2 Confusion Matrix

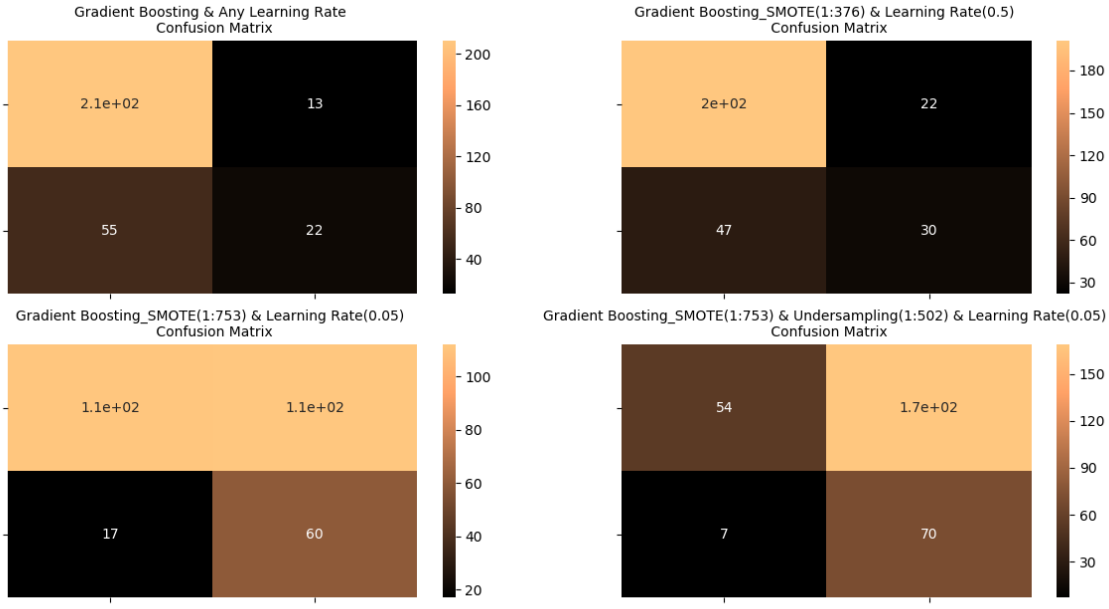


Figure 5.3.10: Confusion Matrix - Gradient Boosting

From the above confusion matrix, we can also clearly see the performance of an algorithm by numbers. In addition, we can also get the *recall* and *precision* from the confusion matrix:

Gradient Boosting	Recall	Precision
No SMOTE **Any learning rate**	0.29	0.63
SMOTE (376) **Any learning rate (0.5)**	0.39	0.58
SMOTE (753) **Any learning rate (0.5)**	0.78	0.35
SMOTE (753) **Undersampling (502)** **Any learning rate (0.5)**	0.91	0.29

Table 5.4.1: Recall and Precision table - Gradient Boosting

From the *Table 5.3.4*, it is clear that the *recall* will also increase from 0.29 to 0.78 when the SMOTE increases. At the same time, the *precision* will decrease constantly from 0.63 to 0.35 if we just increase the SMOTE from 0 to 753. If we combine the SMOTE and undersampling methods, then the *recall* has been improved significantly from 0.78 to 0.91 and *precision* decreases from 0.35 to 0.26. In terms of combination of SMOTE (753) and undersampling (502), a *recall* (0.91) means 91% fraudulent claims can be detected; *precision* (0.29) means among all those predicted fraudulent claims, 29% claims are really fraudulent. In practice, the model is also useful.

5.4 Conclusion

For this dataset, in order to compare different classification methods, we also just focus on the combination of SMOTE (753) and undersampling (502), because in this situation, the performance of classification is the best. As shown below:

	Recall	Precision
Naive Bayes	0.82	0.29
Logistic Regression	0.86	0.70
Random Forest	0.84	0.68
Gradient Boosting	0.91	0.29

Table 5.4: Overall Recall and Precision Table

From *Table 5.4.1,* it is clear to see that for the combination of SMOTE and undersampling (3:2), gradient boosting also has the highest *recall*, which means it can detect the most percentage of the fraudulent claims. Compared to logistic regression, the *recall* of gradient boosting increases by 0.5 (from 0.86 to 0.91), while the precision decreases from 0.70 to 0.29. If we just focus on the *recall*, we regard the combination of SMOTE (753) and undersampling (502) methods as the best one; but if we care about both *recall* and *precision*, then logistic regression is the best model, because the *recall* of logistic regression is just 5% less than gradient boosting, but the *precision* is 41% higher than the *precision* of gradient boosting.

In terms of each classification method, we can see that the *recall* will be improved by increasing the SMOTE index, but the *precision* will decrease at the same time. Therefore, in practice, if we want to increase the *precision* and we do not need *recall* to be that high, then we can decrease the

ratio between SMOTE and undersampling to get a better *precision*. Because in this case, less noise will be introduced to the model.

Conclusions

The two datasets studied here are totally separate, although both are about car insurance claims fraud detection. We analysed these two datasets in order to get more general conclusions, that not only pertain to a single example, so they are more convincing.

Comparing all the SMOTE, undersampling methods and models, we can comfortably say that when increasing the SMOTE ratio, the *recall* increases. And for most of these models, the *recall* is even better if we combine the SMOTE and undersampling methods.

Focusing just on the combination of SMOTE and undersampling methods, it seems that, based on these two samples of auto insurance, gradient boosting is the best model to maximize the fraudulent claims detected, because the *recall* of gradient boosting is the highest, which is the primary objective.

In addition, it is easy to see from *Figure 4.1.1* and *Figure 5.1.1* that these two datasets are unbalanced at different degrees (6.1% to 93.9% and 24.7% to 75.3%). From the output, it is clear that for the datasets with different unbalance ratios, the performances of different models and ratios between SMOTE and under-sampling will also change.

For the very unbalanced dataset (6.1% to 93.9%), choosing a higher ratio of SMOTE and undersampling (3:2) seems reasonable, which can improve the *recall*, while the *precision* does not decrease significantly. The best model is gradient boosting, but for the dataset that is slightly less unbalanced (24.7% to 75.3%), choosing the SMOTE (753) method is the best, maximizing *recall* without *precision* decreasing much at the same time. Also, for this second dataset, logistic regression is the best model if a high *precision*is seeked, even though its *recall* is not the highest.

In addition, for this second dataset with a smaller unbalance ratio (24.7% to 75.3%), it also yields a higher *precision*. For example, for both datasets, if logistic regression is used, the *recalls* of two datasets are almost the same, 0.856 and 0.86, respectively, while the *precisions* are 0.1 and 0.7. This means that the less unbalanced dataset tends to have a much better *precision*.

There is also another method to deal with unbalanced datasets, using a combination of clustering and SMOTE methods, to then apply separate models for each cluster. From the graph in *Figure 4.6.2*, it is clear that the *recall* of all these four classification models have improved significantly.

In conclusion, as mentioned before, even though *recall* is high enough for some models, we also see that the *precision* is a quite low, which means that many legitimate-claim customers will be bothered when investigated for fraudulent claims. Therefore, there is still a need to find methods that maintain a high *precision* while maximizing *recall*.

Finally, we can also use link analyses to improve fraud detection. Using other related datasets for the same policy holders, such as their education background, their residential address, information on their friends and social networks, the time at which the car accident occured or their financial credit, can help improve fraud detection.

In this thesis we did not consider these factors, like accident time, because the public datasets used do not include such detailed personal information. Insurance companies have detailed longitudinal records for each policyholder, for more than a year. In this case, the time factor can also be considered. For example, one client buys several policies in a short time period, and then this policyholder files a large insurance claim. This behaviour should be detected.

Therefore, in practice, insurance data scientists can also use the time factor as an important variable to detect insurance fraud.

References

[1] Altini, M. "Dealing With Imbalanced Data: Undersampling, Oversampling And Proper Cross-Validation", Aug. 2015, https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation.

[2] Amjad, M. "Naive Bayes Classifier-Based Fire Detection Using Smartphone Sensors", *Master Thesis*, Dept. Engineering and Science, University of Agder, Grimstad, June 2014.

[3] Breiman, L. "Random Forests", Dept. Statistics, University of California, Berkeley, CA, January 2001.

[4] Brid, Rajesh S. "Decision Trees a Simple Way to Visualize a Decision", https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

[5] Brockett, L., Golden, L. and Guillen, M. "A Robust Unsupervised Method for Fraud Rate Estimation", *Journal of Risk & Insurance* 80(1): 121-143, Jan. 2011.

[6] Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. "SMOTE: synthetic minority over-sampling technique", *Journal of Artificial Intelligence Research* 16, 321-357.

[7] Coors, S. "Automatic Gradient Boosting", *Master's Thesis*, Dept. Statistics, Ludwig Maximilians University, Munich, March 2018.

[8] Efimov, D. "Gradient Boosting Trees: Theory and Applications". Nov. 2016. http://efimov-ml.com/pdfs/Lvov_xgboost_2016.pdf

[9] Friedman, H. "Greedy Function Approximation: A Gradient Boosting Machine", *Annals of Statististics* 29 (2001), no. 5, 1189--1232. https://projecteuclid.org/euclid.aos/1013203451.

[10] Li, Y., Yan, C., Liu, W. and Li, M. "Research and application of random forest model in mining automobile insurance fraud", *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Aug. 2016.

[11] Murphy, P. "Naive Bayes Classifiers", Oct. 24, 2006, https://www.ic.unicamp.br/~rocha/teaching/2011s1/mc906/aulas/naive-bayes.pdf.

[12] Narkhede, S. "Understanding AUC - ROC Curve", *Towards Data Science*, https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[13] Oates, S. "A Logistic Regression Analysis of Score Sending and College Matching Among High School Students", *Ph.D. Thesis*, Dept. Philosophy, University of Iowa, Fall 2015.

[14] Park, H. "An Introduction to Logistic Regression: From Basic Concepts to Interpretation with Particular Attention to Nursing Domain", *J Korean Acad Nurs, Vol.43, No.2,* Apr. 2013.

[15] Shen, A., Tong, R. and Deng, Y. "Application of Classification Models on Credit Card Fraud Detection", *IEEE*, June 2007.

[16] Sahin, Y. and Duman, E. "Detecting credit card fraud by ANN and logistic regression", *2011 International Symposium on Innovations in Intelligent Systems and Applications,* Jun. 2001.

[17] Schapire, E. "A Brief Introduction to Boosting", *IJCAI'99 Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, USA, 1998.

[18] Shubham, J. "Naïve Bayes Theorem", June 2018. https://becominghuman.ai/naive-bayes-theorem-d8854a41ea08.

[19] Viaene, S., Derrig, S. and Dedene, G. "Boosting Naive Bayes for Claim Fraud Diagnosis", *International Conference on Data Warehousing and Knowledge Discovery*, Sep. 2002.

[20] Zheng, Z., Cai, Y. and Li, Y. "Oversampling Method for Imbalanced Classification", *Computing and Informatics*, Vol. 34, 1017–1037, 2015.

[21] "Cross Validated, Area Under Curve of ROC vs. Overall Accuracy". https://stats.stackexchange.com/questions/68893/area-under-curve-of-roc-vs-overall-accuracy.

[22] "Kaggle, Credit Card Fraud Detection". https://www.kaggle.com/joparga3/in-depth-skewed-data-classif-93-recall-acc-now.

[23] "Kaggle, Fraud Detection in Insurance Claims". https://www.kaggle.com/roshansharma/fraud-detection-in-insurance-claims/notebook#Data-Processing

[24] "Kaggle, Fraud Detection with Naive Bayes Classifier". https://www.kaggle.com/lovedeepsaini/ fraud-detection-with-naive-bayes-classifier.

[25] "Kaggle, Prediction with Gradient Boosting classifier". https://www.kaggle.com/beagle01/ prediction-with-gradient-boosting-classifier.

[26] "Logistic Regression Theory: An Overview". https://dzone.com/articles/logistic-regression-theory

[27] "Python for Data Science, Logistic Regression". https://pythonfordatascience.org/logistic-regression-python/

[28] Ravelin, "Link Analysis For Fraud Detection". https://www.ravelin.com/insights/link-analysis-and-graph-database-for-fraud-detection?utm_term=fraud%20analysis&utm_campaign=Online+fraud&utm_source=adwords&utm_medium=ppc&hsa_ver=3&hsa_grp=70183435564&hsa_kw=fraud%20analysis&hsa_tgt=kwd-296702529729&hsa_mt=b&hsa_acc=6420205522&hsa_net=adwords&hsa_ad=355069767789&hsa_src=g&hsa_cam=1746763886&gclid=Cj0KCQiAvc_xBRCYARIsAC5QT9kvgFZiBNHTZPXDx-f7willl87Uu5J63uXRf1qLwkC27G-D4T1X3ZFwaAvyLEALw_wcB.

[29] Wikipedia, "Gradient Boosting". https://en.wikipedia.org/wiki/Gradient_boosting.

Appendix A

Code for the Analysis of Dataset 1

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import recall_score, precision_score, accuracy_score, f1_score
from itertools import cycle
from sklearn.metrics import confusion_matrix,precision_recall_curve,auc,roc_auc_score,roc_curve,recall_score,classification_report
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
```

```
from sklearn.model_selection import cross_val_predict
from sklearn.cluster import KMeans



data = pd.read_csv('C:/Users/Jason/Desktop/Thesis/Data.csv')
df2 = pd.read_csv('C:/Users/Jason/Desktop/Thesis/Data_Cleaned.csv')
df = data

data.head()

data.FraudFound_P.value_counts()

# bar chart
from matplotlib import pyplot as plt
name_list = ['1', '0']
num_list = [data['FraudFound_P'].sum(), 11300-data['FraudFound_P'].sum()]
plt.bar(range(len(num_list)), num_list, color = 'rgb', tick_label = name_list)

# pie chart
labels = '1', '0'
sizes = [data['FraudFound_P'].sum(), 11300-data['FraudFound_P'].sum()]
plt.pie(sizes, labels = labels, autopct = '%1.1f%%', shadow = False)

# check the correlation
data['FraudFound_P'].corr(data['DriverRating'])

## Clustering & SMOTE
df5 = df.sample(frac=1)
X5 = df5.iloc[:, :-1]
y5 = df5.iloc[:, -1]
```

```
X5_train, X5_test, y5_train, y5_test = train_test_split(X5, y5, test_size=0.2, random_state=42)

sm = SMOTE(ratio={1: 9000},random_state=42)
X5sm_train, y5sm_train = sm.fit_sample(X5_train, y5_train)
X5sm_train = pd.DataFrame(X5sm_train)
y5sm_train = pd.DataFrame(y5sm_train)

# sum(y5sm_train==1)

# X5sm_train, y5sm_train = X5_train, y5_train
train = X5sm_train
train['test'] = 0
train['fraud'] = y5sm_train

test = X5_test
test['test'] = 1
test['fraud'] = y5_test
train.columns = test.columns
data77 = pd.concat([train, test])
data99 = data77.iloc[:,:-2]
data99['fraud'] = data77.iloc[:, 31:32]
data99['test'] = data77.iloc[:, 32:33]
clf=KMeans(n_clusters=3)
clf=clf.fit(data99)

# clf.cluster_centers_
data99['label']=clf.labels_
data99['test']=data77.iloc[:,-2].values
data99['fraud']=data77.iloc[:,-1].values
data0=data99.loc[data99["label"] == 0]
```

```
data1=data99.loc[data99["label"] == 1]
data2=data99.loc[data99["label"] == 2]

### data0 ###############
data0=data99.loc[data99["label"] == 0]
data0_test = data0.loc[data0['test']==1]
data0_test = data0_test.drop(['test', 'label'],axis=1)
X_test0 = data0_test.iloc[:,:-1]
y_test0 = data0_test.iloc[:,-1]

data0_train = data0.loc[data0['test']==0]
# sm = SMOTE(ratio={1: 9000},random_state=42)
# X5sm_train, y5sm_train = sm.fit_sample(X5_train, y5_train)
# X5sm_train = pd.DataFrame(X5sm_train)
# y5sm_train = pd.DataFrame(y5sm_train)

data0_train = data0_train.drop(['test', 'label'],axis=1)
X_train0 = data0_train.iloc[:,:-1]
y_train0 = data0_train.iloc[:,-1]

# Randon Forest Classifier
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train0, y_train0)
# tree best estimator
tree_clf = grid_tree.best_estimator_
y_pred_tree0 = tree_clf.predict(X_test0)
```

```
recall_score(y_test0, y_pred_tree0, average='binary') # 0.8469387755102041
precision_score(y_test0, y_pred_tree0) # 0.10863874345549739

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)



### Naive Bayes
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import QuantileTransformer

pipeline = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())
pipeline.fit(X_train0, y_train0)
y_pred_tree0 = pipeline.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.35714285714285715
precision_score(y_test0, y_pred_tree0) # 0.13307984790874525

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)



# LR Classifier
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train0, y_train0)
```

```
log_reg = grid_log_reg.best_estimator_
y_pred_log = log_reg.predict(X_test0)

recall_score(y_test0, y_pred_log, average='binary') # 0.8088235294117647  0.7857142857142857
precision_score(y_test0, y_pred_log) # 0.1323024054982818

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_log, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)

# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
      max_depth = 2, random_state = 0)
    gb4.fit(X_train0, y_train0)

gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth = 2,
    random_state = 0)   ### 0.05 => 0.95  0.09
gb4.fit(X_train0, y_train0)
y_pred_tree0 = gb4.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.5714285714285714
precision_score(y_test0, y_pred_tree0) # 0.14583333333333334

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

```
### data1 ################
data0 = data1
data0_test = data0.loc[data0['test']==1]
data0_test = data0_test.drop(['test', 'label'],axis=1)
X_test0 = data0_test.iloc[:,:-1]
y_test0 = data0_test.iloc[:,-1]

data0_train = data0.loc[data0['test']==0]
data0_train = data0_train.drop(['test', 'label'],axis=1)
X_train0 = data0_train.iloc[:,:-1]
y_train0 = data0_train.iloc[:,-1]



# Randon Forest Classifier
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train0, y_train0)
# tree best estimator
tree_clf = grid_tree.best_estimator_
y_pred_tree0 = tree_clf.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.72
precision_score(y_test0, y_pred_tree0) # 0.16822429906542055

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

```
# NB
pipeline = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())
pipeline.fit(X_train0, y_train0)
y_pred_tree0 = pipeline.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.92
precision_score(y_test0, y_pred_tree0) # 0.09787234042553192

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)

# LR Classifier
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train0, y_train0)
log_reg = grid_log_reg.best_estimator_
y_pred_log = log_reg.predict(X_test0)

recall_score(y_test0, y_pred_log, average='binary') # 0.88
precision_score(y_test0, y_pred_log) # 0.15602836879432624

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_log, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)

# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
```

```
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in learning_rates:

   gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
     max_depth = 2, random_state = 0)

   gb4.fit(X_train0, y_train0)


gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth = 2,
     random_state = 0)   ### 0.05 => 0.95  0.09

gb4.fit(X_train0, y_train0)

y_pred_tree0 = gb4.predict(X_test0)


recall_score(y_test0, y_pred_tree0, average='binary') # 0.96

precision_score(y_test0, y_pred_tree0) # 0.125


import seaborn as sn

confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix, annot=True)


### data2 ################

data0 = data2

data0_test = data0.loc[data0['test']==1]

data0_test = data0_test.drop(['test', 'label'],axis=1)

X_test0 = data0_test.iloc[:,:-1]

y_test0 = data0_test.iloc[:,-1]


data0_train = data0.loc[data0['test']==0]

data0_train = data0_train.drop(['test', 'label'],axis=1)

X_train0 = data0_train.iloc[:,:-1]

y_train0 = data0_train.iloc[:,-1]
```

```
# Randon Forest Classifier
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train0, y_train0)
# tree best estimator
tree_clf = grid_tree.best_estimator_
y_pred_tree0 = tree_clf.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.6842105263157895
precision_score(y_test0, y_pred_tree0) # 0.16666666666666666

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)



# NB
pipeline = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())
pipeline.fit(X_train0, y_train0)
y_pred_tree0 = pipeline.predict(X_test0)

recall_score(y_test0, y_pred_tree0, average='binary') # 0.7894736842105263
precision_score(y_test0, y_pred_tree0) # 0.13157894736842105

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

```
# LR Classifier
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train0, y_train0)
log_reg = grid_log_reg.best_estimator_
y_pred_log = log_reg.predict(X_test0)

recall_score(y_test0, y_pred_log, average='binary')
precision_score(y_test0, y_pred_log)

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_log, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)



# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
      max_depth = 2, random_state = 0)
    gb4.fit(X_train0, y_train0)

gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth = 2,
      random_state = 0)   ### 0.05 => 0.95  0.09
gb4.fit(X_train0, y_train0)
y_pred_tree0 = gb4.predict(X_test0)
```

```
recall_score(y_test0, y_pred_tree0, average='binary') # 0.15789473684210525
precision_score(y_test0, y_pred_tree0) # 0.10714285714285714

import seaborn as sn
confusion_matrix = pd.crosstab(y_test0, y_pred_tree0, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)

from sklearn.model_selection import train_test_split
X = np.array(data.ix[:, data.columns != 'FraudFound_P'])
y = np.array(data.ix[:, data.columns == 'FraudFound_P'])

X1 = df2.iloc[:, :-1]
y1 = df2.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=42)

# Standardization
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
# X_train = pd.DataFrame(X_train)
X_test = sc.transform(X_test)

X1_train = sc.fit_transform(X1_train)
X1_test = sc.transform(X1_test)

# Correlation Matrices
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))
```

```
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix", fontsize=14)

# New
df = df.sample(frac=1)
non_fraud_df = df.loc[df['FraudFound_P'] == 0][:685]
fraud_df = df.loc[df['FraudFound_P'] == 1]
normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

new_df = normal_distributed_df.sample(frac=1, random_state=42)
sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix', fontsize=14)
plt.show()

# Check Assumptions of LR
# ASSUMPTION OF CONTINUOUS IVS BEING LINEARLY RELATED TO THE LOG ODDS
import statsmodels.formula.api as smf
C_S = sns.regplot(x= 'ClaimSize', y= 'FraudFound_P', data= df, logistic= True).set_title("ClaimSize Log Odds
    Linear Plot")
C_S = sns.regplot(x= 'Age', y= 'FraudFound_P', data= df, logistic= True).set_title("Age Log Odds Linear Plot")

# ASSUMPTION OF ABSENCE OF MULTICOLLINEARITY
df.corr()
# Delete the MULTICOLLINEARITY Variables
columns = ['Month', 'AgeOfVehicle_year', 'AgeOfPolicyHolder', 'Year', 'BasePolicy', 'VehiclePrice',
    'VehicleCategory', 'PolicyNumber']
df1 = df.drop(columns, axis=1)

# ASSUMPTION OF LOCK OF OUTLIERS
```

```
ClaimSize_box = sns.boxplot(data= df[['ClaimSize']]).set_title("ClaimSize Box Plot")

from sklearn.model_selection import cross_val_score
log_reg_score = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Logistic Regression Cross Validation Score: ', round(log_reg_score.mean() * 100, 2).astype(str) + '%')



# LR
# SMOTE
from imblearn.over_sampling import SMOTE
# SMOTE Technique (OverSampling) After splitting and Cross Validating
sm = SMOTE(ratio={1: 4000},random_state=42)
Xsm_train, ysm_train = sm.fit_sample(X_train, y_train)
X1sm_train, y1sm_train = sm.fit_sample(X1_train, y1_train)

sm1 = SMOTE(ratio={1: 8514},random_state=42)
X3sm_train, y3sm_train = sm1.fit_sample(X_train, y_train)
X2sm_train, y2sm_train = sm1.fit_sample(X1_train, y1_train)



# Under - Smaple & SMOTE for LR
df3 = df2.sample(frac=1)
non_fraud_df = df3.loc[df['FraudFound_P'] == 0][:6000]
fraud_df = df3.loc[df['FraudFound_P'] == 1]
normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
new_df1 = normal_distributed_df.sample(frac=1, random_state=42)

X8 = new_df1.iloc[:, :-1]
y8 = new_df1.iloc[:, -1]
X8_train, X8_test, y8_train, y8_test = train_test_split(X8, y8, test_size=0.2, random_state=42)
```

```
sm = SMOTE(ratio={1: 9000},random_state=42)
X8sm_train, y8sm_train = sm.fit_sample(X8_train, y8_train)



# Logistic Regression
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X1_train, y1_train)
log_reg = grid_log_reg.best_estimator_
y_pred_log = log_reg.predict(X1_test)

# Logistic Regression After SMOTE (1:4000)
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg2 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg2.fit(X1sm_train, y1sm_train)
log_reg2 = grid_log_reg2.best_estimator_
y1sm_pred_log = log_reg2.predict(X1_test)

# Logistic Regression After SMOTE (1:9000)
grid_log_reg3 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg3.fit(X2sm_train, y2sm_train)
log_reg3 = grid_log_reg3.best_estimator_
y2sm_pred_log = log_reg3.predict(X1_test)

# Undersampling & SMOTE
grid_log_reg8 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg8.fit(X8sm_train, y8sm_train)
log_reg8 = grid_log_reg8.best_estimator_
y8sm_pred_log = log_reg8.predict(X8_test)
```

```
# ROC Curve
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

log_reg_pred = cross_val_predict(log_reg, X1_train, y1_train, cv=5)
log_fpr, log_tpr, log_thresold = roc_curve(y1_train, log_reg_pred)

log_reg_pred2 = cross_val_predict(log_reg2, X1sm_train, y1sm_train, cv=5)
log_fpr2, log_tpr2, log_thresold2 = roc_curve(y1sm_train, log_reg_pred2)

log_reg_pred3 = cross_val_predict(log_reg3, X2sm_train, y2sm_train, cv=5)
log_fpr3, log_tpr3, log_thresold3 = roc_curve(y2sm_train, log_reg_pred3)

log_reg_pred8 = cross_val_predict(log_reg8, X8sm_train, y8sm_train, cv=5)
log_fpr4, log_tpr4, log_thresold4 = roc_curve(y8sm_train, log_reg_pred8)

def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(log_fpr, log_tpr, label='Logistic Regression (Before SMOTE) Classifier Score:
      {:.4f}'.format(roc_auc_score(y1_train, log_reg_pred)))
    plt.plot(log_fpr2, log_tpr2, label='Logistic Regression (After SMOTE 1:4000) Classifier Score:
      {:.4f}'.format(roc_auc_score(y1sm_train, log_reg_pred2)))
    plt.plot(log_fpr3, log_tpr3, label='Logistic Regression (After SMOTE 1:8514) Classifier Score:
      {:.4f}'.format(roc_auc_score(y2sm_train, log_reg_pred3)))
    plt.plot(log_fpr4, log_tpr4, label='Logistic Regression (SMOTE 1:9000 & Under-sample 1:6000) Classifier
      Score: {:.4f}'.format(roc_auc_score(y8sm_train, log_reg_pred8)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
```

```
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
      xytext=(0.6, 0.3),
            arrowprops=dict(facecolor='#6E726D', shrink=0.05),
            )
    plt.legend()


import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)
plt.show()

# Randon Forest Classifier
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
            "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train, y_train)
# tree best estimator
tree_clf = grid_tree.best_estimator_
y_pred_tree = tree_clf.predict(X_test)

# Decision Tree SMOTE 4000
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
            "min_samples_leaf": list(range(5,7,1))}
grid_tree_sm1 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm1.fit(Xsm_train, ysm_train)
log_reg_sm1 = grid_tree_sm1.best_estimator_
ysm_pred_tree1 = log_reg_sm1.predict(X_test)
```

```
# SMOTE 8514
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree_sm2 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm2.fit(X3sm_train, y3sm_train)
log_reg_sm2 = grid_tree_sm2.best_estimator_
ysm_pred_tree2 = log_reg_sm2.predict(X_test)

# Undersampling & SMOTE
df5 = df.sample(frac=1)
non_fraud_df5 = df5.loc[df5['FraudFound_P'] == 0][:6000]
fraud_df5 = df5.loc[df5['FraudFound_P'] == 1]
normal_distributed_df5 = pd.concat([fraud_df5, non_fraud_df5])
new_df5 = normal_distributed_df5.sample(frac=1, random_state=42)

X5 = new_df5.iloc[:, :-1]
y5 = new_df5.iloc[:, -1]
X5_train, X5_test, y5_train, y5_test = train_test_split(X5, y5, test_size=0.2, random_state=42)

sm = SMOTE(ratio={1: 9000},random_state=42)
X5sm_train, y5sm_train = sm.fit_sample(X5_train, y5_train)

grid_tree_sm5 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm5.fit(X5sm_train, y5sm_train)
log_reg_sm5 = grid_tree_sm5.best_estimator_
ysm_pred_tree5 = log_reg_sm5.predict(X_test)

# recall  --  RF
from sklearn.metrics import recall_score
```

```
# UNDER-SAMPLE (1:6000) and SMOTE (1:9000)
recall_score(y_test, ysm_pred_tree5, average='binary') # 0.8897058823529411
precision_score(y_test, ysm_pred_tree5)  # 0.12015888778550149

# Logistic Regression After SMOTE (1:9000)
recall_score(y_test, ysm_pred_tree2, average='binary') # 0.7132352941176471
precision_score(y_test, ysm_pred_tree2)  # 0.13324175824175824

# Logistic Regression After SMOTE (1:4000)
recall_score(y_test, ysm_pred_tree1, average='binary')  # 0.5
precision_score(y_test, ysm_pred_tree1)  # 0.1650485436893204

# No SMOTE
recall_score(y_test, y_pred_tree, average='binary') # 0
precision_score(y_test, y_pred_tree) # 0

# Tree Report
from sklearn.metrics import classification_report

y_pred_tree = tree_clf.predict(X_test)
print(classification_report(y_test, y_pred_tree))

ysm_pred_tree = log_reg_sm2.predict(X_test)
print(classification_report(y_test, ysm_pred_tree))

# confusion_matrix Logistic Regression
from sklearn.metrics import confusion_matrix

log_cf = confusion_matrix(y_test, y_pred_log)
```

```
log_cf_sm = confusion_matrix(y_test, ysm_pred_log)

# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
tree_cf = confusion_matrix(y_test, y_pred_tree)
tree_cf_sm = confusion_matrix(y_test, ysm_pred_tree)

# ROC Curve
from sklearn.metrics import roc_curve

# Check cross validation of the Decision Tree   DecisionTree Classifier Cross Validation Score 94.06%
tree_score = cross_val_score(tree_clf, X_train, y_train, cv=5)
print('DecisionTree Classifier Cross Validation Score', round(tree_score.mean() * 100, 2).astype(str) + '%')

from sklearn.model_selection import cross_val_predict
# Create a DataFrame with all the scores and the classifiers names.

log_reg_pred = cross_val_predict(log_reg, X_train, y_train, cv=5, method="decision_function")
svc_pred = cross_val_predict(svc, X_train, y_train, cv=5, method="decision_function")

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
print('Logistic Regression: ', roc_auc_score(y_train, log_reg_pred))
print('Decision Tree Classifier: ', roc_auc_score(y_train, tree_pred))

# recall  -- LR
from sklearn.metrics import recall_score

# UNDER-SAMPLE (1:6000) and SMOTE (1:9000)
recall_score(y8_test, y8sm_pred_log, average='binary')
```

```
precision_score(y8_test, y8sm_pred_log)  # 0.1

# Logistic Regression After SMOTE (1:8514)
recall_score(y1_test, y2sm_pred_log, average='binary')
precision_score(y1_test, y2sm_pred_log)

# Logistic Regression After SMOTE (1:4000)
recall_score(y1_test, y1sm_pred_log, average='binary')
precision_score(y1_test, y1sm_pred_log)

# Logistic Regression
recall_score(y1_test, y_pred_log, average='binary')
precision_score(y1_test, y_pred_log)

# confusion_matrix LR
from sklearn.metrics import confusion_matrix
LR_cf1 = confusion_matrix(y1_test, y_pred_log)
LR_cf2 = confusion_matrix(y1_test, y1sm_pred_log)
LR_cf3 = confusion_matrix(y1_test, y2sm_pred_log)
LR_cf4 = confusion_matrix(y8_test, y8sm_pred_log)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(LR_cf1, ax=ax[0][0], annot=True, cmap=plt.cm.Blues)
ax[0, 0].set_title("Logistic Regression \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(LR_cf2, ax=ax[0][1], annot=True, cmap=plt.cm.Blues)
ax[0][1].set_title("Logistic Regression_SMOTE(1:4000) \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
```

```
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(LR_cf3, ax=ax[1][0], annot=True, cmap=plt.cm.Blues)
ax[1][0].set_title("Logistic Regression_SMOTE(1:8514) \n Confusion Matrix", fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(LR_cf4, ax=ax[1][1], annot=True, cmap=plt.cm.Blues)
ax[1][1].set_title("Logistic Regression_SMOTE(1:9000) & Undersampling(1:6000) \n Confusion Matrix",
    fontsize=14)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


# RF
recall_score(y_test, y_pred_tree, average='binary')
precision_score(y_test, y_pred_tree)


recall_score(y_test, ysm_pred_tree1, average='binary')
precision_score(y_test, ysm_pred_tree1)


recall_score(y_test, ysm_pred_tree2, average='binary')
precision_score(y_test, ysm_pred_tree2)
recall_score(y_test, ysm_pred_tree5, average='binary')
precision_score(y_test, ysm_pred_tree5)


# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
tree_cf1 = confusion_matrix(y_test, y_pred_tree)
tree_cf2 = confusion_matrix(y_test, ysm_pred_tree1)
tree_cf3 = confusion_matrix(y_test, ysm_pred_tree2)
```

```
tree_cf4 = confusion_matrix(y_test, ysm_pred_tree5)
from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(tree_cf1, ax=ax[0][0], annot=True, cmap=plt.cm.Blues)
ax[0, 0].set_title("Random Forest Classifier \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf2, ax=ax[0][1], annot=True, cmap=plt.cm.Blues)
ax[0][1].set_title("Random Forest Classifier_SMOTE(1:4000) \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf3, ax=ax[1][0], annot=True, cmap=plt.cm.Blues)
ax[1][0].set_title("Random Forest Classifier_SMOTE(1:8514) \n Confusion Matrix", fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf4, ax=ax[1][1], annot=True, cmap=plt.cm.Blues)
ax[1][1].set_title("Random Forest Classifier_SMOTE(1:9000) & Undersampling(1:6000) \n Confusion
    Matrix", fontsize=14)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

rf_reg_pred = cross_val_predict(tree_clf, X_train, y_train, cv=5)
rf_fpr, rf_tpr, rf_thresold = roc_curve(y_train, rf_reg_pred)
```

```
rf_reg_pred2 = cross_val_predict(log_reg_sm1, Xsm_train, ysm_train, cv=5)
rf_fpr2, rf_tpr2, rf_thresold2 = roc_curve(ysm_train, rf_reg_pred2)

rf_reg_pred3 = cross_val_predict(log_reg_sm2, X3sm_train, y3sm_train, cv=5)
rf_fpr3, rf_tpr3, rf_thresold3 = roc_curve(y3sm_train, rf_reg_pred3)

rf_reg_pred4 = cross_val_predict(log_reg_sm5, X5_train, y5_train, cv=5)
rf_fpr4, rf_tpr4, rf_thresold4 = roc_curve(y5_train, rf_reg_pred4)

def graph_roc_curve_multiple(rf_fpr, rf_tpr, rf_fpr2, rf_tpr2):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(rf_fpr, rf_tpr, label='Random Forest (Before SMOTE) Classifier Score: {:.4f}'.format(roc_auc_score(y_train, rf_reg_pred)))
    plt.plot(rf_fpr2, rf_tpr2, label='Random Forest (After SMOTE 1:4000) Classifier Score: {:.4f}'.format(roc_auc_score(ysm_train, rf_reg_pred2)))
    plt.plot(rf_fpr3, rf_tpr3, label='Random Forest (After SMOTE 1:8514) Classifier Score: {:.4f}'.format(roc_auc_score(y3sm_train, rf_reg_pred3)))
    plt.plot(rf_fpr4, rf_tpr4, label='Random Forest (SMOTE 1:9000 & Under-sample 1:6000) Classifier Score: {:.4f}'.format(roc_auc_score(y5_train, rf_reg_pred4)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3),
                 arrowprops=dict(facecolor='#6E726D', shrink=0.05),)
    plt.legend()

import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
graph_roc_curve_multiple(rf_fpr, rf_tpr, rf_fpr2, rf_tpr2)
```

```
plt.show()


from itertools import cycle
from sklearn.metrics import confusion_matrix,precision_recall_curve,auc,roc_auc_score,roc_curve,recall_score,classification_report


lr = LogisticRegression(C = 0.01, penalty = 'l1')
lr.fit(X_train, y_train)
y_pred_undersample_proba = lr.predict_proba(X_test)
thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal', 'red', 'yellow', 'green', 'blue','black'])


plt.figure(figsize=(5,5))


j = 1
for i, color in zip(thresholds, colors):
   y_test_predictions_prob = y_pred_undersample_proba[:, 1] > i
   precision, recall, thresholds = precision_recall_curve(y4_test, y_test_predictions_prob)


   # Plot Precision-Recall curve
   plt.plot(recall, precision, color=color,
   label='Threshold: %s' % i)
   plt.xlabel('Recall')
   plt.ylabel('Precision')
   plt.ylim([0.0, 1.05])
   plt.xlim([0.0, 1.0])
   plt.title('Precision-Recall example')
   plt.legend(loc="lower left")
```

```
# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb1 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
      max_depth = 2, random_state = 0)
    gb1.fit(X_train, y_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb1.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb1.score(X_test, y_test)))
    print()

gb1 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.5, max_features=2, max_depth = 2,
      random_state = 0)    ### 0  0
gb1.fit(X_train, y_train)
predictions1 = gb1.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))
print()
print("Classification Report")
print(classification_report(y_test, predictions))

##### SMOTE
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb2 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
      max_depth = 2, random_state = 0)
    gb2.fit(Xsm_train, ysm_train)
```

```
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb2.score(Xsm_train, ysm_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb2.score(X_test, y_test)))
    print()

gb2 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.5, max_features=2, max_depth = 2,
      random_state = 0)   ### 0.5 => 0.18  0.17
gb2.fit(Xsm_train, ysm_train)
predictions2 = gb2.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions2))
print()
print("Classification Report")
print(classification_report(y_test, predictions2))

##### SMOTE (9000)
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb3 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
      max_depth = 2, random_state = 0)
    gb3.fit(X3sm_train, y3sm_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb3.score(X3sm_train, y3sm_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb3.score(X_test, y_test)))
    print()

gb3 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth = 2,
      random_state = 0)   ### 0.05 => 0.57  0.14
gb3.fit(X3sm_train, y3sm_train)
predictions3 = gb3.predict(X_test)
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, predictions))

print()

print("Classification Report")

print(classification_report(y_test, predictions))


########## SMOTE & Undersampling

learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in learning_rates:

   gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
     max_depth = 2, random_state = 0)

   gb4.fit(X4sm_train, y4sm_train)

   print("Learning rate: ", learning_rate)

   print("Accuracy score (training): {0:.3f}".format(gb4.score(X4sm_train, y4sm_train)))

   print("Accuracy score (validation): {0:.3f}".format(gb4.score(X_test, y_test)))

   print()


gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth = 2,
     random_state = 0)   ### 0.05 => 0.95  0.09

gb4.fit(X4sm_train, y4sm_train)

predictions4 = gb4.predict(X_test)

print("Confusion Matrix:")

print(confusion_matrix(y_test, predictions))

print()

print("Classification Report")

print(classification_report(y_test, predictions))


# ROC Curve

from sklearn.metrics import roc_curve

from sklearn.model_selection import cross_val_predict


log_reg_pred = cross_val_predict(gb1, X_train, y_train, cv=5)
```

```
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)


log_reg_pred2 = cross_val_predict(gb2, Xsm_train, ysm_train, cv=5)
log_fpr2, log_tpr2, log_thresold2 = roc_curve(ysm_train, log_reg_pred2)


log_reg_pred3 = cross_val_predict(gb3, X3sm_train, y3sm_train, cv=5)
log_fpr3, log_tpr3, log_thresold3 = roc_curve(y3sm_train, log_reg_pred3)


log_reg_pred4 = cross_val_predict(gb4, X4sm_train, y4sm_train, cv=5)
log_fpr4, log_tpr4, log_thresold4 = roc_curve(y4sm_train, log_reg_pred4)


def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(log_fpr, log_tpr, label='Gradient Boosting (Before SMOTE) & Any Learning Rate Classifier Score: {:.4f}'.format(roc_auc_score(y_train, log_reg_pred)))
    plt.plot(log_fpr2, log_tpr2, label='Gradient Boosting (After SMOTE 1:4000) & Learning Rate(0.5) Classifier Score: {:.4f}'.format(roc_auc_score(ysm_train, log_reg_pred2)))
    plt.plot(log_fpr3, log_tpr3, label='Gradient Boosting (After SMOTE 1:8514) Learning Rate(0.05) Classifier Score: {:.4f}'.format(roc_auc_score(y3sm_train, log_reg_pred3)))
    plt.plot(log_fpr4, log_tpr4, label='Gradient Boosting (SMOTE 1:9000 & Under-sample 1:6000) Learning Rate(0.05) Classifier Score: {:.4f}'.format(roc_auc_score(y4sm_train, log_reg_pred4)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3), arrowprops=dict(facecolor='#6E726D', shrink=0.05),)
    plt.legend()


import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
```

```
graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)
plt.show()

# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
tree_cf1 = confusion_matrix(y_test, predictions1)
tree_cf2 = confusion_matrix(y_test, predictions2)
tree_cf3 = confusion_matrix(y_test, predictions3)
tree_cf4 = confusion_matrix(y_test, predictions4)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(tree_cf1, ax=ax[0][0], annot=True, cmap=plt.cm.Blues)
ax[0, 0].set_title("Gradient Boosting & Any Learning Rate \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf2, ax=ax[0][1], annot=True, cmap=plt.cm.Blues)
ax[0][1].set_title("Gradient Boosting_SMOTE(1:4000) & Learning Rate(0.5) \n Confusion Matrix",
    fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf3, ax=ax[1][0], annot=True, cmap=plt.cm.Blues)
ax[1][0].set_title("Gradient Boosting_SMOTE(1:8514) & Learning Rate(0.05) \n Confusion Matrix",
    fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf4, ax=ax[1][1], annot=True, cmap=plt.cm.Blues)
```

```
ax[1][1].set_title("Gradient Boosting_SMOTE(1:9000) & Undersampling(1:6000) & Learning Rate(0.05) \n
    Confusion Matrix", fontsize=14)

ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


##### Naive Bayes

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make_pipeline

from sklearn.naive_bayes import GaussianNB


from sklearn.preprocessing import QuantileTransformer

pipeline = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline.fit(X1_train, y1_train)

y_pred66 = pipeline.predict(X1_test)

y_pred66_prob = pipeline.predict_proba(X1_test)

recall_score(y1_test,y_pred66) # 0.09

NB1 = confusion_matrix(y1_test,y_pred66)


pipeline11 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline11.fit(X1sm_train, y1sm_train)

y_pred6 = pipeline11.predict(X1_test)

y_pred6_prob = pipeline11.predict_proba(X1_test)

recall_score(y1_test,y_pred6) # 0.2857142857142857

NB2 = confusion_matrix(y1_test,y_pred6)


pipeline7 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline7.fit(X2sm_train, y2sm_train)

y_pred7 = pipeline7.predict(X1_test)

recall_score(y1_test,y_pred7) # 0.406

NB3 = confusion_matrix(y1_test,y_pred7)
```

```
pipeline8 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())
pipeline8.fit(X8sm_train, y8sm_train)
y_pred8 = pipeline8.predict(X8_test)
recall_score(y8_test,y_pred8) # 0.7857142857142857
NB4 = confusion_matrix(y8_test,y_pred8)

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import QuantileTransformer

def get_predictions(clf, X_train, y_train, X_test):
    # create classifier
    clf = clf
    # fit it to training data
    clf.fit(X_train,y_train)
    # predict using test data
    y_pred = clf.predict(X_test)
    # Compute predicted probabilities: y_pred_prob
    y_pred_prob = clf.predict_proba(X_test)
    #for fun: train-set predictions
    train_pred = clf.predict(X_train)
    print('train-set confusion matrix:\n', confusion_matrix(y_train,train_pred))
    return y_pred, y_pred_prob
def print_scores(y_test,y_pred,y_pred_prob):
    print('test-set confusion matrix:\n', confusion_matrix(y_test,y_pred))
    print("recall score: ", recall_score(y_test,y_pred))
    print("precision score: ", precision_score(y_test,y_pred))
    print("f1 score: ", f1_score(y_test,y_pred))
```

```
    print("accuracy score: ", accuracy_score(y_test,y_pred))
    print("ROC AUC: {}".format(roc_auc_score(y_test, y_pred_prob[:,1])))

y_pred, y_pred_prob = get_predictions(GaussianNB(), X1_train, y1_train, X1_test)
print_scores(y1_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X1sm_train, y1sm_train, X1_test)
print_scores(y1_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X2sm_train, y2sm_train, X1_test)
print_scores(y1_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X8sm_train, y8sm_train, X8_test)
print_scores(y8_test,y_pred,y_pred_prob)

# ROC Curve
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

log_reg_pred = cross_val_predict(log_reg, X1_train, y1_train, cv=5)
log_fpr, log_tpr, log_thresold = roc_curve(y1_train, log_reg_pred)

log_reg_pred2 = cross_val_predict(log_reg2, X1sm_train, y1sm_train, cv=5)
log_fpr2, log_tpr2, log_thresold2 = roc_curve(y1sm_train, log_reg_pred2)

log_reg_pred3 = cross_val_predict(log_reg3, X2sm_train, y2sm_train, cv=5)
log_fpr3, log_tpr3, log_thresold3 = roc_curve(y2sm_train, log_reg_pred3)

log_reg_pred8 = cross_val_predict(log_reg8, X8sm_train, y8sm_train, cv=5)
log_fpr4, log_tpr4, log_thresold4 = roc_curve(y8sm_train, log_reg_pred8)
```

```
def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(log_fpr, log_tpr, label='Logistic Regression (Before SMOTE) Classifier Score: {:.4f}'.format(roc_auc_score(y1_train, log_reg_pred)))
    plt.plot(log_fpr2, log_tpr2, label='Logistic Regression (After SMOTE 1:4000) Classifier Score: {:.4f}'.format(roc_auc_score(y1sm_train, log_reg_pred2)))
    plt.plot(log_fpr3, log_tpr3, label='Logistic Regression (After SMOTE 1:8514) Classifier Score: {:.4f}'.format(roc_auc_score(y2sm_train, log_reg_pred3)))
    plt.plot(log_fpr4, log_tpr4, label='Logistic Regression (SMOTE 1:9000 & Under-sample 1:6000) Classifier Score: {:.4f}'.format(roc_auc_score(y8sm_train, log_reg_pred8)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3),
                arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                )
    plt.legend()


import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)
plt.show()


# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
tree_cf1 = confusion_matrix(y_test, y_pred_tree)
tree_cf2 = confusion_matrix(y_test, ysm_pred_tree1)
tree_cf3 = confusion_matrix(y_test, ysm_pred_tree2)
```

```
tree_cf4 = confusion_matrix(y_test, ysm_pred_tree5)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(tree_cf1, ax=ax[0][0], annot=True, cmap=plt.cm.Blues)
ax[0, 0].set_title("RandomForest Classifier \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf2, ax=ax[0][1], annot=True, cmap=plt.cm.Blues)
ax[0][1].set_title("RandomForest Classifier_SMOTE(1:4000) \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf3, ax=ax[1][0], annot=True, cmap=plt.cm.Blues)
ax[1][0].set_title("RandomForest Classifier_SMOTE(1:8514) \n Confusion Matrix", fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf4, ax=ax[1][1], annot=True, cmap=plt.cm.Blues)
ax[1][1].set_title("RandomForest Classifier_SMOTE(1:9000) & Undersampling(1:6000) \n Confusion
Matrix", fontsize=14)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(NB1, ax=ax[0][0], annot=True, cmap=plt.cm.Blues)
ax[0, 0].set_title("Naive Bayes \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
```

```
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(NB2, ax=ax[0][1], annot=True, cmap=plt.cm.Blues)

ax[0][1].set_title("Naive Bayes_SMOTE(1:4000) \n Confusion Matrix", fontsize=14)

ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(NB3, ax=ax[1][0], annot=True, cmap=plt.cm.Blues)

ax[1][0].set_title("Naive Bayes_SMOTE(1:8514) \n Confusion Matrix", fontsize=14)

ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(NB4, ax=ax[1][1], annot=True, cmap=plt.cm.Blues)

ax[1][1].set_title("Naive Bayes_SMOTE(1:9000) & Undersampling(1:6000) \n Confusion Matrix",
fontsize=14)

ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)
```

Appendix B

Code for the Analysis of Dataset 2

```
data = pd.read_csv('C:/Users/Jason/Desktop/Thesis/insurance_claims_original.csv')

df2 = pd.read_csv('C:/Users/Jason/Desktop/Thesis/insurance_claims_original.csv')

data = data.drop(['policy_number','policy_bind_date', 'incident_date','incident_location','auto_model'],
axis = 1)


# fill in missing data

# check missing data

data.isnull().any().any()

data = data.replace('?',np.NaN)

data['collision_type'].fillna(data['collision_type'].mode()[0], inplace = True)

data['property_damage'].fillna('NO', inplace = True)

data['police_report_available'].fillna('NO', inplace = True)

data['fraud_reported'] = data['fraud_reported'].replace(('Y','N'),(1,0))


# bar chart

from matplotlib import pyplot as plt

name_list = ['1', '0']

num_list = [data['fraud_reported'].sum(), 1000-data['fraud_reported'].sum()]

plt.bar(range(len(num_list)), num_list, color = 'rgb', tick_label = name_list)


# pie chart

labels = '1', '0'

sizes = [data['fraud_reported'].sum(), 1000-data['fraud_reported'].sum()]
```

```
plt.pie(sizes, labels = labels, autopct = '%1.1f%%', shadow = False)


# Check Assumptions of LR

# ASSUMPTION OF CONTINUOUS IVS BEING LINEARLY RELATED TO THE LOG ODDS

import statsmodels.formula.api as smf

C_S = sns.regplot(x= 'total_claim_amount', y= 'fraud_reported', data= df, logistic= True).set_title("Total_Claim_Amount Log Odds Linear Plot")

C_S = sns.regplot(x= 'age', y= 'fraud_reported', data= df, logistic= True).set_title("Age Log Odds Linear Plot")

C_S = sns.regplot(x= 'months_as_customer', y= 'fraud_reported', data= df, logistic= True).set_title("Months_As_Customers Log Odds Linear Plot")

C_S = sns.regplot(x= 'policy_annual_premium', y= 'fraud_reported', data= df, logistic= True).set_title("Policy_Annual_Premium Log Odds Linear Plot")


# ASSUMPTION OF ABSENCE OF MULTICOLLINEARITY

df.corr()

# Delete the MULTICOLLINEARITY Variables

columns = ['Month', 'AgeOfVehicle_year', 'AgeOfPolicyHolder', 'Year', 'BasePolicy', 'VehiclePrice', 'VehicleCategory', 'PolicyNumber']

df1 = df.drop(columns, axis=1)


# ASSUMPTION OF LOCK OF OUTLIERS

ClaimSize_box = sns.boxplot(data= df[['vehicle_claim']]).set_title("vehicle_claim Box Plot")

ClaimSize_box = sns.boxplot(data= df[['injury_claim']]).set_title("injury_claim Box Plot")


# let's check the correlation auto make with the target

data['incident_type'] = data['incident_type'].replace(('Vehicle Theft','Parked Car','Multi-vehicle Collision', 'Single Vehicle Collision'),(0.09, 0.10, 0.28,0.30))


data['insured_sex'] = data['insured_sex'].replace(('FEMALE','MALE'),(0.24,0.27))

data['policy_csl'] = data['policy_csl'].replace(('500/1000','100/300','250/500'),(0.22,0.26,0.27))

data['policy_state'] = data['policy_state'].replace(('IL','IN','OH'),(0.23,0.255,0.26))
```

```
data['insured_education_level'] = data['insured_education_level'].replace(('Masters', 'High
School','Associate', 'JD','College', 'MD','PhD'),(0.22,0.23,0.24,0.26,0.27,0.28,0.29))

data['police_report_available'] = data['police_report_available'].replace(('NO','YES'),(0.23,0.26))


data[['auto_make','fraud_reported']].groupby(['auto_make'], as_index = False).mean().sort_values(by =
'fraud_reported', ascending = False)


data['auto_make'] = data['auto_make'].replace(('Jeep','Nissan','Toyota','Accura','Saab','Suburu',

'Dodge','Honda','Chevrolet','BMW','Volkswagen','Audi','Ford','Mercedes'),
(0.17,0.18,0.19,0.19,0.23,0.24,0.25,0.26,0.27,0.28,0.28,0.30,0.31,0.36))


data[['incident_city','fraud_reported']].groupby(['incident_city'],as_index = False).mean().sort_values(by
= 'fraud_reported', ascending = False)


data['incident_city'] = data['incident_city'].replace(('Northbrook','Riverwood','Northbend','Springfield',

'Hillsdale','Columbus','Arlington'),(0.22,0.22,0.23,0.24,0.25,0.26,0.29))


data[['incident_state','fraud_reported']].groupby(['incident_state'], as_index =
False).mean().sort_values(by = 'fraud_reported', ascending = False)


# let's perform target encoding for incident state

data['incident_state'] = data['incident_state'].replace(('WV','NY','VA','PA','SC','NC','OH'),

                                        (0.18,0.22,0.23,0.27,0.29,0.31,0.43))

data[['authorities_contacted','fraud_reported']].groupby(['authorities_contacted'],

            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)

data['authorities_contacted'] =
data['authorities_contacted'].replace(('None','Police','Fire','Ambulance','Other'),
(0.07,0.21,0.27,0.29,0.31))

data[['insured_relationship','fraud_reported']].groupby(['insured_relationship'], as_index =
False).mean().sort_values(by = 'fraud_reported', ascending = False)

data['insured_relationship'] = data['insured_relationship'].replace(('husband','own-child','unmarried',

                        'not-in-family','wife','other-relative'),(0.20,0.21,0.24,0.26,0.27,0.29))
```

```
data[['insured_hobbies','fraud_reported']].groupby(['insured_hobbies'],
            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)
data['insured_hobbies'] = data['insured_hobbies'].replace(('camping', 'kayaking', 'golf','dancing',
      'bungie-jumping','movies', 'basketball','exercise','sleeping','video-games','skydiving','paintball',
        'hiking','base-jumping','reading','polo','board-games','yachting', 'cross-fit','chess'),(0.09, 0.09,
            0.11, 0.12,0.16,0.16,0.18,0.19,0.19,0.20,0.22,0.23,0.24,0.27,0.27,0.28,0.29,0.30,0.74,0.83))

data[['insured_occupation','fraud_reported']].groupby(['insured_occupation'],
            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)
data['insured_occupation'] = data['insured_occupation'].replace(('other-service','priv-house-serv',
                  'adm-clerical','handlers-cleaners','prof-specialty','protective-serv',
            'machine-op-inspct','armed-forces','sales','tech-support','transport-moving','craft-repair',
              'farming-fishing','exec-managerial'),(0.16, 0.17,0.17, 0.21,0.22,0.23,0.24,0.25,0.28,0.29,
                                        0.291,0.297,0.30,0.37))

data[['property_damage','fraud_reported']].groupby(['property_damage'],
            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)

data['property_damage'] = data['property_damage'].replace(('NO','YES'),(0.24,0.26))

data[['collision_type','fraud_reported']].groupby(['collision_type'],
            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)
data['collision_type'] = data['collision_type'].replace(('Rear Collision', 'Side Collision', 'Front Collision'),
                                        (0.31,0.25,0.28))

data[['incident_severity','fraud_reported']].groupby(['incident_severity'],
            as_index = False).mean().sort_values(by = 'fraud_reported', ascending = False)
data['incident_severity'] = data['incident_severity'].replace(('Trivial Damage','Minor Damage','Total
Loss', 'Major Damage'),(0.06,0.11,0.13,0.61))
```

```
data[['authorities_contacted','fraud_reported']].groupby(['authorities_contacted'], as_index =
False).mean().sort_values(by = 'fraud_reported', ascending = False)

data['authorities_contacted'] =
data['authorities_contacted'].replace(('None','Police','Fire','Ambulance','Other'),( 0.06,0.21,0.27,0.30,0.3
2))


x = data.drop(['fraud_reported'], axis = 1)

y = data['fraud_reported']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)


# Correlation Matrix

plt.rcParams['figure.figsize'] = (15, 10)

sns.heatmap(X_train.corr(), cmap = 'copper')

plt.title('Heat Map for Correlations', fontsize = 20)

plt.show()


# SMOTE

from imblearn.over_sampling import SMOTE

# SMOTE Technique (OverSampling) After splitting and Cross Validating

sm = SMOTE(ratio={1: 376},random_state=42)

Xsm_train, ysm_train = sm.fit_sample(X_train, y_train)

X1sm_train, y1sm_train = sm.fit_sample(X_train, y_train)


sm1 = SMOTE(ratio={1: 753},random_state=42)

X3sm_train, y3sm_train = sm1.fit_sample(X_train, y_train)

X2sm_train, y2sm_train = sm1.fit_sample(X_train, y_train)


# Under - Smaple & SMOTE for LR

df = data

df3 = data.sample(frac=1)

non_fraud_df = df3.loc[df['fraud_reported'] == 0][:502]
```

```
fraud_df = df3.loc[df['fraud_reported'] == 1]
normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
new_df1 = normal_distributed_df.sample(frac=1, random_state=42)

X8 = new_df1.iloc[:, :-1]
y8 = new_df1.iloc[:, -1]
X8_train, X8_test, y8_train, y8_test = train_test_split(X8, y8, test_size=0.3, random_state=42)

sm = SMOTE(ratio={1: 753}, random_state=42)
X8sm_train, y8sm_train = sm.fit_sample(X8_train, y8_train)
from collections import Counter

# Logistic Regression
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train, y_train)
log_reg = grid_log_reg.best_estimator_
y_pred_log = log_reg.predict(X_test)

# Logistic Regression After SMOTE (1:376)
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg2 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg2.fit(X1sm_train, y1sm_train)
log_reg2 = grid_log_reg2.best_estimator_
y1sm_pred_log = log_reg2.predict(X_test)

# Logistic Regression After SMOTE (1:753)
grid_log_reg3 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg3.fit(X2sm_train, y2sm_train)
log_reg3 = grid_log_reg3.best_estimator_
```

```
y2sm_pred_log = log_reg3.predict(X_test)

# Undersampling & SMOTE
grid_log_reg8 = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg8.fit(X8sm_train, y8sm_train)
log_reg8 = grid_log_reg8.best_estimator_
y8sm_pred_log = log_reg8.predict(X_test)

# ROC Curve
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

log_reg_pred = cross_val_predict(log_reg, X_train, y_train, cv=5)
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)

log_reg_pred2 = cross_val_predict(log_reg2, X1sm_train, y1sm_train, cv=5)
log_fpr2, log_tpr2, log_thresold2 = roc_curve(y1sm_train, log_reg_pred2)

log_reg_pred3 = cross_val_predict(log_reg3, X2sm_train, y2sm_train, cv=5)
log_fpr3, log_tpr3, log_thresold3 = roc_curve(y2sm_train, log_reg_pred3)

log_reg_pred8 = cross_val_predict(log_reg8, X8sm_train, y8sm_train, cv=5)
log_fpr4, log_tpr4, log_thresold4 = roc_curve(y8sm_train, log_reg_pred8)

def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):
   plt.figure(figsize=(16,8))
   plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
   plt.plot(log_fpr, log_tpr, label='Logistic Regression (Before SMOTE) Classifier Score:
{:.4f}'.format(roc_auc_score(y_train, log_reg_pred)))
   plt.plot(log_fpr2, log_tpr2, label='Logistic Regression (After SMOTE 1:376) Classifier Score:
{:.4f}'.format(roc_auc_score(y1sm_train, log_reg_pred2)))
```

```
    plt.plot(log_fpr3, log_tpr3, label='Logistic Regression (After SMOTE 1:753) Classifier Score: {:.4f}'.format(roc_auc_score(y2sm_train, log_reg_pred3)))

    plt.plot(log_fpr4, log_tpr4, label='Logistic Regression (SMOTE 1:753 & Under-sample 1:502) Classifier Score: {:.4f}'.format(roc_auc_score(y8sm_train, log_reg_pred8)))

    plt.plot([0, 1], [0, 1], 'k--')

    plt.axis([-0.01, 1, 0, 1])

    plt.xlabel('False Positive Rate', fontsize=16)

    plt.ylabel('True Positive Rate', fontsize=16)

    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3), arrowprops=dict(facecolor='#6E726D', shrink=0.05))

plt.legend()


import matplotlib.pyplot as plt

from sklearn.metrics import roc_auc_score

graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)

plt.show()


# DecisionTree Classifier

from sklearn.model_selection import GridSearchCV

tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),

          "min_samples_leaf": list(range(5,7,1))}

grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)

grid_tree.fit(X_train, y_train)

# tree best estimator

tree_clf = grid_tree.best_estimator_

y_pred_tree = tree_clf.predict(X_test)


# Decision Tree SMOTE

from sklearn.model_selection import GridSearchCV

tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),

          "min_samples_leaf": list(range(5,7,1))}
```

```
grid_tree_sm1 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm1.fit(X1sm_train, y1sm_train)
log_reg_sm1 = grid_tree_sm1.best_estimator_
ysm_pred_tree1 = log_reg_sm1.predict(X_test)

# SMOTE
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree_sm2 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm2.fit(X2sm_train, y2sm_train)
log_reg_sm2 = grid_tree_sm2.best_estimator_
ysm_pred_tree2 = log_reg_sm2.predict(X_test)



# Undersampling & SMOTE
from sklearn.model_selection import GridSearchCV
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
          "min_samples_leaf": list(range(5,7,1))}
grid_tree_sm5 = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree_sm5.fit(X8sm_train, y8sm_train)
log_reg_sm5 = grid_tree_sm5.best_estimator_
ysm_pred_tree5 = log_reg_sm5.predict(X_test)

# recall  -- LR
from sklearn.metrics import recall_score
# UNDER-SAMPLE (1:6000) and SMOTE
recall_score(y_test, y8sm_pred_log, average='binary')
precision_score(y_test, y8sm_pred_log)
```

```
# Logistic Regression After SMOTE
recall_score(y_test, y2sm_pred_log, average='binary')
precision_score(y_test, y2sm_pred_log)

# Logistic Regression After SMOTE
recall_score(y_test, y1sm_pred_log, average='binary')
precision_score(y_test, y1sm_pred_log)

# Logistic Regression
recall_score(y_test, y_pred_log, average='binary')
precision_score(y_test, y_pred_log)

# confusion_matrix LR
from sklearn.metrics import confusion_matrix
LR_cf1 = confusion_matrix(y_test, y_pred_log)
LR_cf2 = confusion_matrix(y_test, y1sm_pred_log)
LR_cf3 = confusion_matrix(y_test, y2sm_pred_log)
LR_cf4 = confusion_matrix(y_test, y8sm_pred_log)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(LR_cf1, ax=ax[0][0], annot=True, cmap='copper')
ax[0, 0].set_title("Logistic Regression \n Confusion Matrix", fontsize=10)
ax[0, 0].set_xticklabels(['', ''], fontsize=10, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=10, rotation=360)

sns.heatmap(LR_cf2, ax=ax[0][1], annot=True, cmap='copper')
ax[0][1].set_title("Logistic Regression_SMOTE(1:376) \n Confusion Matrix", fontsize=10)
ax[0][1].set_xticklabels(['', ''], fontsize=10, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=10, rotation=360)
```

```
sns.heatmap(LR_cf3, ax=ax[1][0], annot=True, cmap='copper')
ax[1][0].set_title("Logistic Regression_SMOTE(1:753) \n Confusion Matrix", fontsize=10)
ax[1][0].set_xticklabels(['', ''], fontsize=10, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=10, rotation=360)

sns.heatmap(LR_cf4, ax=ax[1][1], annot=True, cmap='copper')
ax[1][1].set_title("Logistic Regression_SMOTE(1:753) & Undersampling(1:502) \n Confusion Matrix",
fontsize=10)
ax[1][1].set_xticklabels(['', ''], fontsize=10, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=10, rotation=360)

# RF
# UNDER-SAMPLE (1:6000) and SMOTE (1:9000)
recall_score(y_test, ysm_pred_tree5, average='binary')
precision_score(y_test, ysm_pred_tree5)

# Logistic Regression After SMOTE
recall_score(y_test, ysm_pred_tree2, average='binary')
precision_score(y_test, ysm_pred_tree2)

# Logistic Regression After SMOTE
recall_score(y_test, ysm_pred_tree1, average='binary')
precision_score(y_test, ysm_pred_tree1)

# No SMOTE
recall_score(y_test, y_pred_tree, average='binary')
precision_score(y_test, y_pred_tree)

# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
```

```
tree_cf1 = confusion_matrix(y_test, y_pred_tree)
tree_cf2 = confusion_matrix(y_test, ysm_pred_tree1)
tree_cf3 = confusion_matrix(y_test, ysm_pred_tree2)
tree_cf4 = confusion_matrix(y_test, ysm_pred_tree5)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(tree_cf1, ax=ax[0][0], annot=True, cmap='copper')
ax[0, 0].set_title("RandomForest Classifier \n Confusion Matrix", fontsize=10)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf2, ax=ax[0][1], annot=True, cmap='copper')
ax[0][1].set_title("RandomForest Classifier_SMOTE(1:376) \n Confusion Matrix", fontsize=10)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf3, ax=ax[1][0], annot=True, cmap='copper')
ax[1][0].set_title("RandomForest Classifier_SMOTE(1:753) \n Confusion Matrix", fontsize=10)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf4, ax=ax[1][1], annot=True, cmap='copper')
ax[1][1].set_title("RandomForest Classifier_SMOTE(1:753) & Undersampling(1:502) \n Confusion
Matrix", fontsize=10)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
```

```
rf_reg_pred = cross_val_predict(tree_clf, X_train, y_train, cv=5)

rf_fpr, rf_tpr, rf_thresold = roc_curve(y_train, rf_reg_pred)


rf_reg_pred2 = cross_val_predict(log_reg_sm1, X1sm_train, y1sm_train, cv=5)

rf_fpr2, rf_tpr2, rf_thresold2 = roc_curve(y1sm_train, rf_reg_pred2)


rf_reg_pred3 = cross_val_predict(log_reg_sm2, X2sm_train, y2sm_train, cv=5)

rf_fpr3, rf_tpr3, rf_thresold3 = roc_curve(y2sm_train, rf_reg_pred3)


rf_reg_pred4 = cross_val_predict(log_reg_sm5, X8sm_train, y8sm_train, cv=5)

rf_fpr4, rf_tpr4, rf_thresold4 = roc_curve(y8sm_train, rf_reg_pred4)


def graph_roc_curve_multiple(rf_fpr, rf_tpr, rf_fpr2, rf_tpr2):

   plt.figure(figsize=(16,8))

   plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)

   plt.plot(rf_fpr, rf_tpr, label='Random Forest (Before SMOTE) Classifier Score:
{:.4f}'.format(roc_auc_score(y_train, rf_reg_pred)))

   plt.plot(rf_fpr2, rf_tpr2, label='Random Forest (After SMOTE 1:376) Classifier Score:
{:.4f}'.format(roc_auc_score(y1sm_train, rf_reg_pred2)))

   plt.plot(rf_fpr3, rf_tpr3, label='Random Forest (After SMOTE 1:753) Classifier Score:
{:.4f}'.format(roc_auc_score(y2sm_train, rf_reg_pred3)))

   plt.plot(rf_fpr4, rf_tpr4, label='Random Forest (SMOTE 1:753 & Under-sample 1:502) Classifier Score:
{:.4f}'.format(roc_auc_score(y8sm_train, rf_reg_pred4)))

   plt.plot([0, 1], [0, 1], 'k--')

   plt.axis([-0.01, 1, 0, 1])

   plt.xlabel('False Positive Rate', fontsize=16)

   plt.ylabel('True Positive Rate', fontsize=16)

   plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
xytext=(0.6, 0.3), arrowprops=dict(facecolor='#6E726D', shrink=0.05),)

   plt.legend()
```

```
import matplotlib.pyplot as plt

from sklearn.metrics import roc_auc_score

graph_roc_curve_multiple(rf_fpr, rf_tpr, rf_fpr2, rf_tpr2)

plt.show()


from itertools import cycle

from sklearn.metrics import
confusion_matrix,precision_recall_curve,auc,roc_auc_score,roc_curve,recall_score,classification_report


lr = LogisticRegression(C = 0.01, penalty = 'l1')

lr.fit(X_train, y_train)

y_pred_undersample_proba = lr.predict_proba(X_test)

thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal', 'red', 'yellow', 'green',
'blue','black'])


plt.figure(figsize=(5,5))


j = 1

for i, color in zip(thresholds, colors):

   y_test_predictions_prob = y_pred_undersample_proba[:, 1] > i

   precision, recall, thresholds = precision_recall_curve(y4_test, y_test_predictions_prob)


   # Plot Precision-Recall curve

   plt.plot(recall, precision, color=color,

         label='Threshold: %s' % i)

   plt.xlabel('Recall')

   plt.ylabel('Precision')

   plt.ylim([0.0, 1.05])

   plt.xlim([0.0, 1.0])

   plt.title('Precision-Recall example')
```

```
    plt.legend(loc="lower left")


# import machine learning algorithms
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc


learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb1 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
max_depth = 2, random_state = 0)
    gb1.fit(X_train, y_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb1.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb1.score(X_test, y_test)))
    print()
gb1 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.5, max_features=2, max_depth = 2,
random_state = 0)    ### 0  0
gb1.fit(X_train, y_train)
predictions1 = gb1.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions1))
print()
print("Classification Report")
print(classification_report(y_test, predictions1))


###SMOTE
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb2 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
max_depth = 2, random_state = 0)
    gb2.fit(Xsm_train, ysm_train)
```

```
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb2.score(X1sm_train, y1sm_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb2.score(X_test, y_test)))
    print()

gb2 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.5, max_features=2, max_depth = 2,
random_state = 0)   ### 0.5 => 0.18  0.17
gb2.fit(X1sm_train, y1sm_train)
predictions2 = gb2.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions2))
print()
print("Classification Report")
print(classification_report(y_test, predictions2))

#####SMOTE
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb3 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
max_depth = 2, random_state = 0)
    gb3.fit(X2sm_train, y2sm_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb3.score(X2sm_train, y2sm_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb3.score(X_test, y_test)))
    print()

gb3 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth =
2, random_state = 0)   ### 0.05 => 0.57  0.14
gb3.fit(X2sm_train, y2sm_train)
predictions3 = gb3.predict(X_test)
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, predictions3))

print()

print("Classification Report")

print(classification_report(y_test, predictions3))


#### SMOTE & Undersampling

learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in learning_rates:

   gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2,
max_depth = 2, random_state = 0)

   gb4.fit(X8sm_train, y8sm_train)

   print("Learning rate: ", learning_rate)

   print("Accuracy score (training): {0:.3f}".format(gb4.score(X8sm_train, y8sm_train)))

   print("Accuracy score (validation): {0:.3f}".format(gb4.score(X_test, y_test)))

   print()


gb4 = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.05, max_features=2, max_depth =
2, random_state = 0)   ### 0.05 => 0.95  0.09

gb4.fit(X8sm_train, y8sm_train)

predictions4 = gb4.predict(X_test)

print("Confusion Matrix:")

print(confusion_matrix(y_test, predictions4))

print()

print("Classification Report")

print(classification_report(y_test, predictions4))


# ROC Curve

from sklearn.metrics import roc_curve

from sklearn.model_selection import cross_val_predict


log_reg_pred = cross_val_predict(gb1, X_train, y_train, cv=5)
```

```
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)


log_reg_pred2 = cross_val_predict(gb2, X1sm_train, y1sm_train, cv=5)

log_fpr2, log_tpr2, log_thresold2 = roc_curve(y1sm_train, log_reg_pred2)


log_reg_pred3 = cross_val_predict(gb3, X2sm_train, y2sm_train, cv=5)

log_fpr3, log_tpr3, log_thresold3 = roc_curve(y2sm_train, log_reg_pred3)


log_reg_pred4 = cross_val_predict(gb4, X8sm_train, y8sm_train, cv=5)

log_fpr4, log_tpr4, log_thresold4 = roc_curve(y8sm_train, log_reg_pred4)


def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):

   plt.figure(figsize=(16,8))

   plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)

   plt.plot(log_fpr, log_tpr, label='Gradient Boosting (Before SMOTE) & Any Learning Rate Classifier
Score: {:.4f}'.format(roc_auc_score(y_train, log_reg_pred)))

   plt.plot(log_fpr2, log_tpr2, label='Gradient Boosting (After SMOTE 1:376) & Learning Rate(0.5)
Classifier Score: {:.4f}'.format(roc_auc_score(y1sm_train, log_reg_pred2)))

   plt.plot(log_fpr3, log_tpr3, label='Gradient Boosting (After SMOTE 1:753) Learning Rate(0.05)
Classifier Score: {:.4f}'.format(roc_auc_score(y2sm_train, log_reg_pred3)))

   plt.plot(log_fpr4, log_tpr4, label='Gradient Boosting (SMOTE 1:753 & Under-sample 1:502) Learning
Rate(0.05) Classifier Score: {:.4f}'.format(roc_auc_score(y8sm_train, log_reg_pred4)))

   plt.plot([0, 1], [0, 1], 'k--')

   plt.axis([-0.01, 1, 0, 1])

   plt.xlabel('False Positive Rate', fontsize=16)

   plt.ylabel('True Positive Rate', fontsize=16)

   plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
xytext=(0.6, 0.3), arrowprops=dict(facecolor='#6E726D', shrink=0.05),)

   plt.legend()


import matplotlib.pyplot as plt

from sklearn.metrics import roc_auc_score
```

```
graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)
plt.show()

# confusion_matrix Tree
from sklearn.metrics import confusion_matrix
tree_cf1 = confusion_matrix(y_test, predictions1)
tree_cf2 = confusion_matrix(y_test, predictions2)
tree_cf3 = confusion_matrix(y_test, predictions3)
tree_cf4 = confusion_matrix(y_test, predictions4)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(tree_cf1, ax=ax[0][0], annot=True, cmap='copper')
ax[0, 0].set_title("Gradient Boosting & Any Learning Rate \n Confusion Matrix", fontsize=10)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf2, ax=ax[0][1], annot=True, cmap='copper')
ax[0][1].set_title("Gradient Boosting_SMOTE(1:376) & Learning Rate(0.5) \n Confusion Matrix",
fontsize=10)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf3, ax=ax[1][0], annot=True, cmap='copper')
ax[1][0].set_title("Gradient Boosting_SMOTE(1:753) & Learning Rate(0.05) \n Confusion Matrix",
fontsize=10)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf4, ax=ax[1][1], annot=True, cmap='copper')
```

```
ax[1][1].set_title("Gradient Boosting_SMOTE(1:753) & Undersampling(1:502) & Learning Rate(0.05) \n Confusion Matrix", fontsize=10)

ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


#### Naive Bayes

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make_pipeline

from sklearn.naive_bayes import GaussianNB


from sklearn.preprocessing import QuantileTransformer

pipeline = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())


pipeline.fit(X_train, y_train)

y_pred6 = pipeline.predict(X_test)

y_pred6_prob = pipeline.predict_proba(X_test)

recall_score(y_test,y_pred6)


pipeline67 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline67.fit(X2sm_train, y2sm_train)

y_pred67 = pipeline67.predict(X_test)

recall_score(y_test,y_pred67)


pipeline7 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline7.fit(X2sm_train, y2sm_train)

y_pred7 = pipeline7.predict(X_test)

recall_score(y_test,y_pred7)


pipeline8 = make_pipeline(QuantileTransformer(output_distribution='normal'), GaussianNB())

pipeline8.fit(X8sm_train, y8sm_train)
```

```
y_pred8 = pipeline8.predict(X_test)
recall_score(y_test,y_pred8)

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import QuantileTransformer

def get_predictions(clf, X_train, y_train, X_test):
    # create classifier
    clf = clf
    # fit it to training data
    clf.fit(X_train,y_train)
    # predict using test data
    y_pred = clf.predict(X_test)
    # Compute predicted probabilities: y_pred_prob
    y_pred_prob = clf.predict_proba(X_test)
    #for fun: train-set predictions
    train_pred = clf.predict(X_train)
    print('train-set confusion matrix:\n', confusion_matrix(y_train,train_pred))
    return y_pred, y_pred_prob

def print_scores(y_test,y_pred,y_pred_prob):
    print('test-set confusion matrix:\n', confusion_matrix(y_test,y_pred))
    print("recall score: ", recall_score(y_test,y_pred))
    print("precision score: ", precision_score(y_test,y_pred))
    print("f1 score: ", f1_score(y_test,y_pred))
    print("accuracy score: ", accuracy_score(y_test,y_pred))
    print("ROC AUC: {}".format(roc_auc_score(y_test, y_pred_prob[:,1])))
```

```
y_pred, y_pred_prob = get_predictions(GaussianNB(), X_train, y_train, X_test)
print_scores(y_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X1sm_train, y1sm_train, X_test)
print_scores(y_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X2sm_train, y2sm_train, X_test)
print_scores(y_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X8sm_train, y8sm_train, X8_test)
print_scores(y8_test,y_pred,y_pred_prob)

y_pred, y_pred_prob = get_predictions(GaussianNB(), X8sm_train, y8sm_train, X_test)
print_scores(y_test,y_pred,y_pred_prob)

# ROC Curve
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

log_reg_pred = cross_val_predict(pipeline, X_train, y_train, cv=5)
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)

log_reg_pred2 = cross_val_predict(pipeline67, X1sm_train, y1sm_train, cv=5)
log_fpr2, log_tpr2, log_thresold2 = roc_curve(y1sm_train, log_reg_pred2)

log_reg_pred3 = cross_val_predict(pipeline7, X2sm_train, y2sm_train, cv=5)
log_fpr3, log_tpr3, log_thresold3 = roc_curve(y2sm_train, log_reg_pred3)

log_reg_pred8 = cross_val_predict(pipeline8, X8sm_train, y8sm_train, cv=5)
log_fpr4, log_tpr4, log_thresold4 = roc_curve(y8sm_train, log_reg_pred8)
```

```
def graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2):
    plt.figure(figsize=(16,8))
    plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
    plt.plot(log_fpr, log_tpr, label='Naive Bayes (Before SMOTE) Classifier Score:
{:.4f}'.format(roc_auc_score(y_train, log_reg_pred)))
    plt.plot(log_fpr2, log_tpr2, label='Naive Bayes (After SMOTE 1:376) Classifier Score:
{:.4f}'.format(roc_auc_score(y1sm_train, log_reg_pred2)))
    plt.plot(log_fpr3, log_tpr3, label='Naive Bayes (After SMOTE 1:753) Classifier Score:
{:.4f}'.format(roc_auc_score(y2sm_train, log_reg_pred3)))
    plt.plot(log_fpr4, log_tpr4, label='Naive Bayes (SMOTE 1:753 & Under-sample 1:502) Classifier Score:
{:.4f}'.format(roc_auc_score(y8sm_train, log_reg_pred8)))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.01, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
xytext=(0.6, 0.3),
                arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                )
    plt.legend()


import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
graph_roc_curve_multiple(log_fpr, log_tpr, log_fpr2, log_tpr2)
plt.show()


from sklearn.metrics import confusion_matrix
clf1 = GaussianNB()
clf1.fit(X_train, y_train)
y_pred11 = clf1.predict(X_test)
train_pred11 = clf1.predict(X_test)
NB1 = confusion_matrix(y_test,train_pred11)
```

```
recall_score(y_test,y_pred11)

clf2 = GaussianNB()
clf2.fit(X1sm_train, y1sm_train)
y_pred12 = clf2.predict(X_test)
train_pred12 = clf2.predict(X_test)
NB2 = confusion_matrix(y_test,train_pred12)
recall_score(y_test,y_pred12)

clf3 = GaussianNB()
clf3.fit(X2sm_train, y2sm_train)
y_pred13 = clf3.predict(X_test)
train_pred13 = clf3.predict(X_test)
NB3 = confusion_matrix(y_test,train_pred13)
recall_score(y_test,y_pred13)

clf4 = GaussianNB()
clf4.fit(X8sm_train, y8sm_train)
y_pred14 = clf4.predict(X_test)
train_pred14 = clf4.predict(X_test)
NB4 = confusion_matrix(y_test,train_pred14)
recall_score(y_test,y_pred14)

from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(NB1, ax=ax[0][0], annot=True, cmap='copper')
ax[0, 0].set_title("Naive Bayes \n Confusion Matrix", fontsize=10)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)
```

```
sns.heatmap(NB2, ax=ax[0][1], annot=True, cmap='copper')

ax[0][1].set_title("Naive Bayes_SMOTE(1:376) \n Confusion Matrix", fontsize=10)

ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(NB3, ax=ax[1][0], annot=True, cmap='copper')

ax[1][0].set_title("Naive Bayes_SMOTE(1:753) \n Confusion Matrix", fontsize=10)

ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)


sns.heatmap(NB4, ax=ax[1][1], annot=True, cmap='copper')

ax[1][1].set_title("Naive Bayes_SMOTE(1:753) & Undersampling(1:502) \n Confusion Matrix",
fontsize=10)

ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)

ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)
```