MACHINE LEARNING APPROACHES ON MOTOR INSURANCE FRAUD DETECTION

E.N.R. Fernando

2021

MACHINE LEARNING APPROACHES ON MOTOR INSURANCE FRAUD DETECTION

A dissertation submitted for the Degree of Master of Business Analytics

E.N.R. Fernando

University of Colombo School of Computing

2021

Declaration

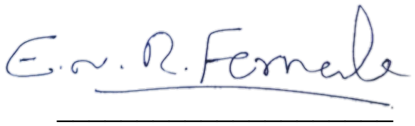
The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: E. N. R. Fernando

Registration Number: 2018/BA/012

Index Number: 18880129



Signature: Date: 14/09/2021

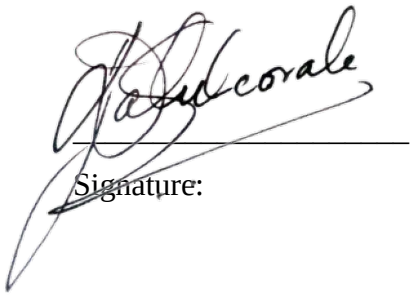
This is to certify that this thesis is based on the work of

Mr. E. N. R. Fernando

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Dr. D. A. S. Athukorale



Signature: Date: 14/09/2021

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FORMULAS

LIST OF FIGURES

ABSTRACT

Insurance companies are one of the most important components of the financial sector for any country. One of the main challenges faced by insurance companies in current market environment are the fraud claims, especially in motor insurance domain. The number of fraud claims are expected to increase in the future, since claim counts are also increasing. Therefore, many researchers in the motor insurance field around the world are trying to find methods on detecting fraudulent claims as early by using machine learning algorithms. Motor claims fraud detection is a complex task since the fraud behavior different for each claim and the detected fraud cases are much low compared to the normal claims. This research aims to develop a motor insurance fraud detection model using classification algorithms and proposed a best model by using some evaluation criteria's. The research includes in its scope motor claim data from Sri Lanka Insurance. Dataset contains 30098 claims and out of these claims 3112 claims are labeled as fraudulent. Dataset is imbalanced since fraud claims also known as positive cases only accounts 10% of total cases. Past claim data are analyzed with underwriting details. Artificial Neural Network, Random Forest and XGBoost algorithms are used as the classifiers to detect a claim is fraudulent or not. These algorithms are analyzed and evaluated by dividing the data set into training, validating and testing. However, when giving input data of an imbalanced class variable to the machine learning model, it is biased towards the majority class. Then it misclassified a fraudulent claim as a normal claim. Oversampling method called Synthetic Minority Oversampling Technique (SMOTE) is applied along with ensemble models to address this problem. Model performance is evaluated based on evaluation criteria's such as recall, precision, f1-score, precision-recall (PR) curve, and receiver operating characteristics (ROC) curve. Since Random Forest and XGBoost classifier model contains parameters that need to be decided by the researcher, hyperparameter tuning is also applied and evaluated. It was found that Random forest and XGBoost models are perform better compared to neural network model. There were not much difference between random forest models and XGBoost models, however, Random forest model with tuned hyperparameters perform slightly better than other models.

1. INTRODUCTION

1.1. The background of the problem

Vehicle is considered as essential product in today's world and people give much higher value for their vehicle safety. So people always tend to buy a vehicle insurance when they purchase a vehicle. Due to its higher demand, auto insurance is considered as major filed in insurance industry. However, like other industries frauds are inevitable in auto insurance due to high demand of claims. A fraud occurs in auto insurance when the customer purposely tries to gets an additional benefit or advantage over insurance claim which is not due. However unlike other crimes, auto insurance fraud claims are not visible or detectable. So exact amount of money stolen from fraud claims are hard to predict. Insurance companies forced to increase its premiums due to the large amount of money lost in this scenario and it will affect their overall performance of the company. Due to these reasons researches on fraud claim detection in auto insurance is considered as one of the most important and interesting research topic in auto insurance industry. There are some fraud detection methods developed by manually, however due to complexity and undetectable nature, the success ratio of them are quite low. Therefore, machine learning solutions are more important in this scenario. If it can implement a machine learning approach for fraud claim detection many benefit can be obtained like reduce human intervention, identify risky claims early etc,..and it will results reduction of monetary losses.

1.2. The Problem Domain

According to the FBI in United States ("Insurance Fraud," n.d.), the insurance companies collect more than $1 trillion premium for year. Further it states that cost of insurance fraud is estimated more than $4 billion per year, which mean on average 4% of insurance premium is lost due to the insurance frauds. There for most of the insurers in global believe that fraud is number one threat to the industry.

In this research it is mainly focused on the Sri Lankan insurance industry domain. Therefore, first it will critically analyze the performance of Sri Lankan Insurance Industry.

1.2.1. Insurance Domain in Sri Lanka

According to Annual Report 2018 ("IBSL-AR-English-2018-Fullset.pdf," n.d.) of Insurance Regulatory Commission of Sri Lanka (IRCSL), it provides relevant legal framework for the supervision and regulate insurance companies. 25 insurance companies, 63 insurance broker companies, 44919 insurance agents, 10 individual loss adjusters are providing insurance services within 773 Grama niladari divisions in 12 divisional secretarial. 9 insurance companies listed in Colombo Stock Exchange (CSE). 11 insurance companies provide only General insurances, 12 insurance companies only prove life (long term) insurances and 2 insurance companies provide life and general insurances to people who lives in Sri Lanka.

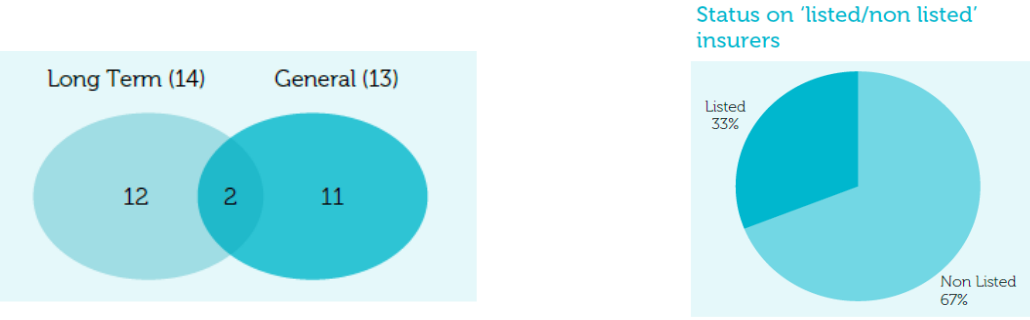


Figure 1.2-1 Insurance Companies Distribution in Sri Lanka

Total assets of insurance industry recorded as Rs. 623,477 Mn in year 2018 and it is 7.18% improvement. 2018 Gross written premium (GWP) is Rs. 181,506 Mn and it is 10.03% of improvement compare with year 2017. 8.88% positive gain shows in insurance density and 2.44% of positive change in penetration as a percentage of GDP.

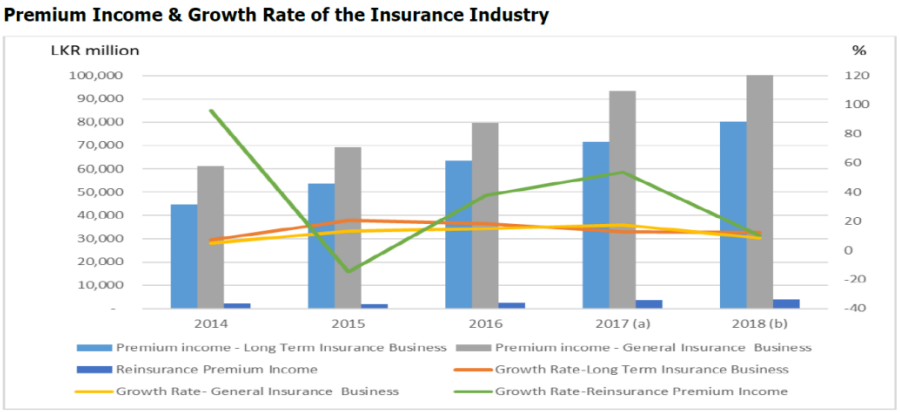


Figure 1.2-2 insurance Growth by each year

Total Assets of Insurance Companies

	2014	2015	2016	2017 (a)	2018 (b)
Long Term Insurance (LKR millions)	247,061	312,713	345,589	391,890	429,706
General Insurance (LKR millions)	174,588	151,177	173,985	185,583	190,088
Reinsurance (LKR millions)	3,065	3,417	5,755	4,212	3,683
Total (LKR millions)	***422,031**	***466,519**	**525,329**	**581,685**	**623,477**
* Inter segment transactions have been eliminated					

Figure 1.2-3 Total Assets of Insurance Companies

General insurance sector is mostly effected sector in insurance industry. Sub components in the general insurance are Fire, Motor, health insurance sectors. Around 61% of general insurance total GWP covered by motor insurance sector. All general insurances are short term insurances. Need to renew insurance yearly. 6,492,003 general insurance policies issued in 2018 and it is5.94% of growth. In motor insurance section there have two type of insurance policies and call these policies as 3rd party insurance policy and Comprehensive insurance policy. Higher amount of contribution policy type is motor 3rd party insurance policy and positive growth rate of 14.6. But 7% of negative growth shown in motor comprehensive policies.

Company –wise Market Share of Gross Written Premium- General Insurance Business for the year ended 31st December 2018

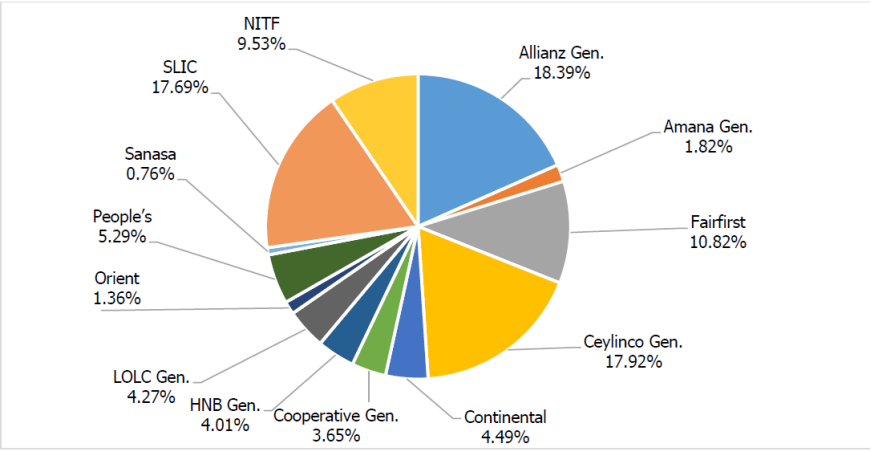


Figure 1.2-4 Market Share of General Insurance Companies

But life insurance is long term insurance policy type. It is shows 12% positive growth rate. 712,013 new long term life policies issued and 10.65% of growth is achieved compared to tear 2017. 3,215,911 life insurance policies are reinforced and growth percentage compared to 2017 is 4.79%.

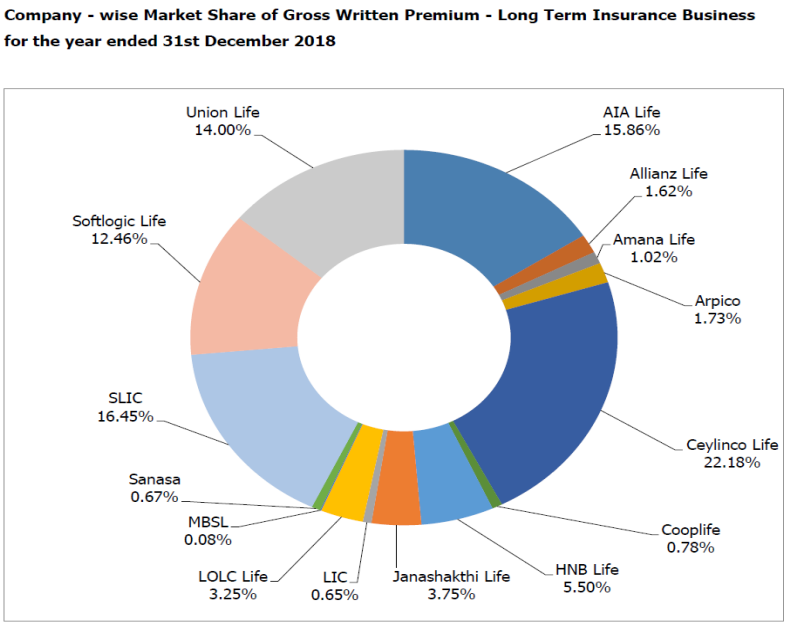


Figure 1.2-5 Market Share of Life Insurance Companies

Industry profitability shows Rs. 49,084 Mn drop in 2017 and Rs. 37,017 Mn drop in 2018. Running modernization projects to enhance efficiency, IFRS 17 implementation, implement wider power insurance regulatory act, supervision and product developments are underway in the current situation.

1.2.2. Fraud Claims in Sri Lankan Domain

On a daily basis every insurance company in Sri Lanka gets alerts on possible fraudulent activities related to claims and underwriting across all portfolios. However, detecting these fraud claims is not easy as most of them are not straightforward. The domain of this research is mainly focus on to the motor claims in Sri Lanka as there were not many researches done on this domain. The data were collected from Sri Lanka Insurance, the largest insurance company in Sri Lanka.

1.3. Problem Statement

The continuous fraud claim submission in motor insurance is one of major problem faced by Insurance companies. Since this is hard to detect it is directly impacted to the financial stability, trustworthy of customers to the company, target customer base of insurance companies. This study is to identify what are the major determinants of motor fraud claims and proposed a machine learning solution for detecting fraud motor claims.

1.4. Motivation

The research idea is inspired through working at Insurance field for past seven years. It was observed that there were many manual works are involved in motor claim handling process. The most common approach for detecting fraud claims depend on human experts. However, this takes time and effort of the work force which is a huge cost to the insurance company. Therefore, it is important to have an effective way to identify motor fraud claims and inform to relevant authorities. It was found that only a few studies concerned with motor fraud detection using machine learning techniques were done in Sri Lankan context. Therefore, applying machine learning concepts in Sri Lankan context is one of the main objective in this research.

1.5. Exact Machine Learning Problem

Insurance Companies use manual procedures to detect the motor fraud claims and these existing methods have low success rate in detecting fraud claims. The most common approach for fraud detection depends on expert intervention. However Existing method doesn't have proper mechanism to identify which attributes are more related to fraud claims. The problem with manual systems is that creating fraudulent claims list takes time. However, data mining algorithms would help to solve these problems since they can be trained with data and the model can be improved over time. This could save insurance companies time and require less human intervention. Proposed methodology can be used as a machine learning solution for an insurance company.

1.6. Scope

This research aims to identify what are the major determinants of motor fraud claims and proposed a machine learning solution for detecting fraud motor claims. The research includes in its scope motor claim data in Sri Lanka Insurance. Past claim data will be analyzed with underwriting details. Those data were imported into Python as a data frame. The imported dataset was separated into Training, Validating and testing datasets and applied the machine learning algorithm.

1.7. Objectives

The main goal of the research consists on developing affective and accurate machine learning model which can be used for detecting motor fraud claims. To accomplish it, objectives are summarized as follows.

- To study and understand about the Motor Fraud Claim Detection.
- To study the different methods used in Motor Fraud Claims.
- To study how the Machine Learning can be used to fraud claim detection.
- To compare and identify different machine learning algorithms for fraud claim detection.
- To research about Random Forest, Extreme Gradient Boosting (XGBoost) and Neural Networks machine learning classifiers.
- Study different evaluation methods used in classification problems.
- Learn how to use Python programming to implement fraud detection solution.
- To propose machine learning solution to identify a fraud claim with high accuracy.

1.8. Document Overview

The chapter 01 of this project covers introduction of insurance fraud detection. It will explain insurance industry in Sri Lanka, objectives and flow of this project.

Chapter 02 covers the literature survey of insurance fraud detection. It will discuss about history of fraud detection, different techniques used in fraud detection and previous researches that have been done on fraud detection. Research articles are summarized with its data mining technique used in the research.

In chapter 03, it contains the research methodology. It included the detail information regarding the dataset, preprocessing methods used and data mining techniques used for the research.

Chapter 04 involves in design and implements a models based on Classifiers discussed in chapter 03. Also testing and validating the models will be discussed in this chapter.

Finally, in chapter 05, conclusion and further development of the project will be discussed.

2. LITERATURE REVIEW

The first section of the literature review discusses the Insurance fraud detection with specially mention about motor fraud detection. It also contains the review of insurance fraud detection literature. Section two contains with machine learning techniques and data mining classifiers used in this research with justification for using them.

2.1. Fraud Claims and Fraudsters

There can be various ways that people commit frauds. The oxford dictionary ("Fraud - Oxford Reference," n.d.) defined the fraud as "wrongful of criminal deception intended to result in financial or personal gain". Fraud occurs in wide variety of industries. According to the (Ngai et al., 2011) financial frauds can be summarized as figure 2.1.1. However unlike other industries, frauds are inevitable in motor insurance due to high demand of claims. According to the Insurance Information Institute of USA ("Background on: Insurance fraud | III," n.d.), Insurance fraud claim is a deliberate deception perpetrated by a customer or agent for the purpose of additional benefit or advantage which is not due. (Morley et al., n.d.) defined that fraud as ''knowingly making a fictitious claim, inflating a claim or adding extra items to a claim, or being in any way dishonest with the intention of gaining more than legitimate entitlement''. Fraud can be committed at different points in the claim process by applicants, customers, or professionals like brokers who provide services to claimants. Insurance agents and employees may also commit insurance fraud. Common frauds include padding (increasing the amount of the claim by fixed amount), inflating claims (submits exaggerated or false information), misrepresenting facts on an insurance application, submitting claims for injuries or damage that never occurred and staging accidents. According to the Insurance Regulatory Commission of Sri Lanka - Annual Report 2018 ("IBSL-AR-English-2018-Fullset.pdf," n.d.), fraud may involve collusion, forgery, intentional omissions, misrepresentations, or the override of internal controls.

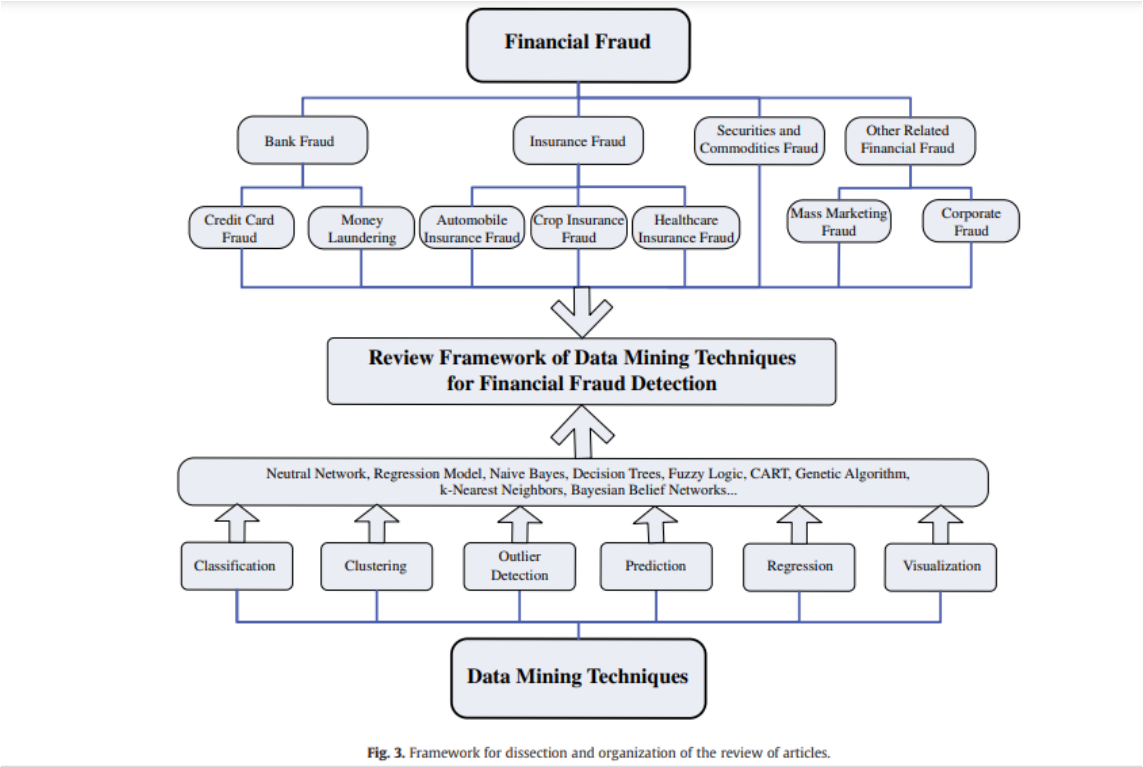


Fig. 3. Framework for dissection and organization of the review of articles.

Figure 2.1-1 Financial Fraud Breakdown

There are also many types of fraudster in current motor insurance industry. (Morley et al., n.d.) classified fraudsters as the opportunist, the amateur and the professional. The opportunist ones take advantage from a genuine loss to commit fraud, for example, by claiming alongside genuine losses for items not broken in an accident. The amateur may take a step further to opportunistic fraudster, for example, submitting a claim in an accident that never took place. The professional, most serious type of fraudster, takes frauds in both individually and in organized networks, for example staging claims. Whether these organized networks can be found in all insurance domains is unclear and very hard. However, there are evidences that they exist in motor insurance and the potential for the spread of organized networks further. (Phua et al., n.d.) categorized these fraudsters into three main groups as figure 2.1.2, which are Manager, External Group and Employee. They explained that traditionally the business is always vulnerable for internal fraud from its management and employees and in addition it is vulnerable for external fraud.

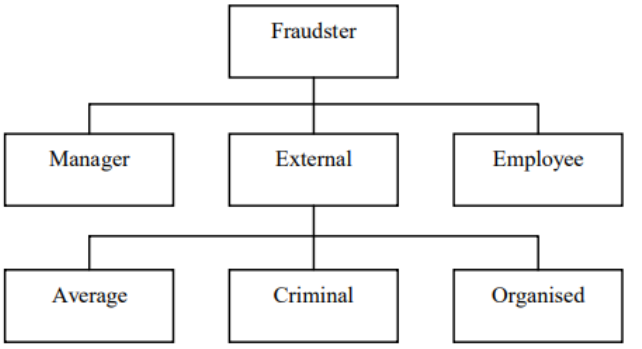


Figure 2.1: Hierarchy chart of white-collar crime perpetrators from both firm-level and community-level perspectives.

Figure 2.1-2 Fraudster Hierarchy view

2.2. Approaches to detecting potential motor fraud

Insurance fraud detection is very important for insurance companies since it is directly involving their reputation and financial stability. In simple terms Insurance fraud claim detection is nothing but distinguishing fraudulent claims from genuine claims. However unlike other crimes, motor insurance fraud claims are not visible or detectable. So exact amount of money stolen from fraud claims are hard to predict. (Phua et al., n.d.) explained that fraud is refers to abuse of organization's system without direct legal consequences. Insurance companies forced to increase its premiums due to the large amount of money lost in this scenario and it will affect their overall performance of the company.

The responsibility of detecting fraud claims in an insurance company rests with staff at the claims handling process. Claims handlers are often experienced, but however Insurance companies like Infinilytics estimates the rate at of fraudulent claims (Health claims specially) are detected are as low as 10%, suggests that huge numbers of fraudulent cases remain undetected. According to the Atlas Magazine – Insurance News around the world nearly 54% insurers believe that fraud stand the number one threat for insurance companies. Cost of fraud has been estimated 10% of the total claims around the Europe region and according to them this figure is remarkably high in Asian countries. There should be a mechanism for spotting suspect claims in place, to discriminate quickly between claims that carry a high level of fraud probability and false positives, that is, genuine claims that are suspicious to an inexperienced staff member.

In order to solve fraud claims problem effectively, insurance companies may face many hardships internally and externally. According to the Atlas magazine ("Insurance fraud detection and cost to industry," n.d.) these issues centered around lack of data and inadequate response mechanism. A survey has showed that main problem that insurance companies face is problems with internal data quality.

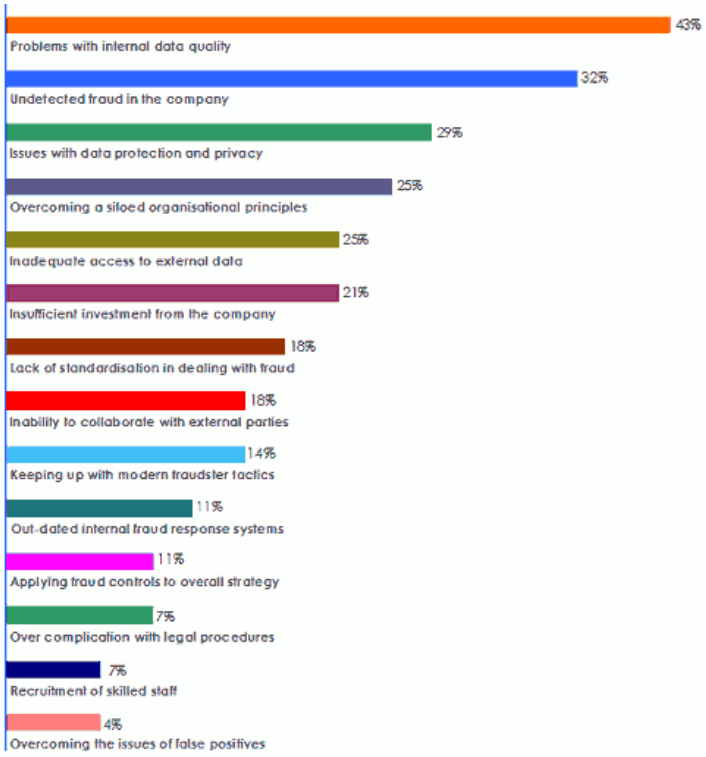


Figure 2.2-1 Problems faced by insurance companies

In order to increase the probability of detecting fraudulent claims, insurance companies are moving to new technologies. Researches on fraud claim detection in motor insurance is considered as one of the most important and interesting research topic in motor insurance industry. There are some fraud detection methods developed by manually, however due to complexity and undetectable nature, the success ratio of them are quite low. Therefore, machine learning solutions are more important in this scenario. If it can implement a machine learning approach for fraud claim detection, many benefit can be obtained like reduce human intervention, identify risky claims early etc, and it will result reduction of monetary losses.

(Subudhi and Panigrahi, 2018) proposed a new fraud detection methodology for auto insurance based on an approach named as adaptive synthetic sampling. This is an imbalanced learning approach which replicates points which are harder to learn rather than easier to learn. The adaptive synthetic sampling was used to remove the class imbalance in the insurance data. The data set had 33 features which include class label mentioning the claim is fraud or not. The researcher has used three different supervised classifiers for detecting fraud claims which is namely, Support Vector Machine, Decision Tree, and Multi-layer perceptron. To get the best classifier model the 10-fold cross validation method has been used. It was identified that using adaptive synthetic sampling method, the system has performed far better than imbalanced one. Further it was observed that rate of detecting fraud claims by using Support Vector Machines or by using Decision Trees is high.

(Dhieb et al., 2019) developed an automated fraud detection method for auto insurance based on extreme gradient boosting algorithm. The aim of the framework was to predict and classify auto insurance claims into different fraud types. The performance of the system was evaluated by comparing other classifiers namely, Decision Tree algorithm, naïvebayes and nearest neighbor algorithm. Features like age of the customer, Gender, marital status, sum insured etc.. were used to develop the model. Final model classified the claims into three categories namely Invalid kind of loss, Fraudulent claim amount and No premium but has claim. The results ensured that extreme gradient boosting algorithm has the best accuracy together with classification of fraud claims into different types. But when comparing with the training and evaluation time, extreme gradient boosting algorithm takes more time which need to address.

(Kalvihura, et.al 2020) ("AUTO-INSURANCE FRAUD DETECTION," 2020) proposed a data preprocessing technique which is feature engineering approach to improve the performance of prediction model. RFM which is Recency, Frequency, Monetary based features together with ensemble feature selection technique was used to detecting auto insurance frauds. To capture the behavior of features from claim toward their fraud, RFM based features were used and the ensemble feature selection was used to get the best candidate features. Finally Bootstrapped Random Forest classification was used to classify claims based on the selected candidate features. To capture relevant behavior analysis for insurance claims the researcher proposed a new feature engineering methodology call HOBA which categories homogeneous claims into one group. Initially there were 32 features to select and random forest was used as the stratified ensemble

classifier. The results showed that the proposed feature ensemble model shows a significant improvement in the overall performance.

(S. Patil, 2018) conducted a survey on machine learning approaches for detecting insurance claim frauds. They have disclosed both traditional techniques and some contemporary techniques like hybrid and ensemble learning. They argued that hybrid models provide flexibility since its uses different algorithms together and these hybrid models outperformed traditional learning methods. However, they observed that ensemble learning approaches gaining more importance recently due to their reliability and flexibility. They discovered that these ensemble learning methods addresses some common problems in machine learning such as class imbalance, over fitting and concept drift. Although ensembles are expensive in terms of time and resources, it will be a onetime investment if it successful to implement.

(Ghorbani and Farzai, 2018) proposed a data mining approach to extract hidden knowledge and patterns in auto insurance fraud claims. They have select 7 clusters according to the insurance expert's suggestions and applied K mean clustering using Euclidean distance as similarity and dissimilarity measure for fraud claim data. The results showed significant accuracy in comparison with real statistics. They have succeeded in extracting patterns which can be used to detect fraud claims in next accident cases.

Apart from the auto insurance there were researches done for detecting fraud claims from other insurance fields. Although auto insurance claims and health claims are different in fields, they have some similarities with respect to insurance fraud behavior. (Vineela, et.al 2020) ("Fraud Detection in Health Insurance Claims using Machine Learning Algorithms," 2020) has conducted a research by applying machine learning algorithms on fraud detection in health claims. They have applied both unsupervised algorithms like K –mean clustering and Hierarchical Clustering and supervised algorithms like decision trees and regression to identify and classify fraud claims. It was discovered that Decision Trees – supervised learning method gave the better outcome. In banking sector Credit card frauds are common and a serious problem. Machine learning algorithms were used to detect fraud claims even though there were limitations in researches due to confidentiality. (Randhawa et al., 2018) has done a study on credit card frauds using different machine learning algorithms. Some standard machine learning models like Naïve bayes, support vector machines, Decision Trees have been applied for a publicaly available data set. The models were evaluated using individual models and hybrid models; which include adaBoost and majority

voting combination method. The majority voting based method offered the best performance when each model was evaluated. (Batra and Kundra, 2019) has proposed a naïve bayes classification approach for fraud detection of insurance claims. Execution time and Accuracy of the model compared with the voting classifier and it was identified that naïve bayes had high accuracy and low execution time compared to voting classifier.

2.3. Data Mining Techniques in Fraud Detection

Data mining can be recognized as discovering hidden knowledge and patterns from large databases. Insurance companies provide verity of services and with the development with technology, their stored databases are growing rapidly. So the data mining techniques can be applied to discover hidden patterns of fraud claims information in insurance companies. (Ngai et al., 2011) classified data mining techniques in fraud detection into six different classes.

2.3.1. Classification

The most common mining technique used in fraud detection can be identified as classification. Classification is the process of define a model that distinguish classes and use it to predict unknown class label. (Zhang et.al 2011) described that classification can be used to identifying common features and models that distinguish data classes or concepts. Artificial Neural Networks, Support Vector Machines, Decision Trees and Naïve Bayes are the mostly used classifies in this technique in fraud detection.

2.3.2. Clustering

Another common data mining technique used in fraud detection is clustering. Clustering is used to group similar objects where there is no clear idea about the class of the objects. The objects are clustered such that intra cluster similarity is maximized and inter cluster similarity is minimized. (Ghorbani and Farzai, 2018) explained that clustering facilitate taxonomy which organization of objects into classes that group similar events together. Common clustering techniques in fraud detection can be classified as K-mean, K-nearest neighbor and Self Organize Maps.

2.3.3. Regression

Regression is another data mining technique that can be identified as a method of fraud detection. This is a statistical technique which is used to explain the relationship between dependent variable and independent variables. Since fraud detection can be identified as a binary classification problem, logit regression is used most often in researches.

2.3.4. Outlier Detection

Outlier or anomaly detection is another data mining technique used in fraud detection. Objects that have different behaviors from rest of the data are called outliers. According to the (Agyemang et al., 2006), outlier mining is applied to identifying outliers from huge data repositories. In fraud detection outlier may indicate fraudulent activity since it is different from rest of the population. Many data mining algorithms try to minimize the influence the effect of outliers from the data, however in fraud detection the main objective is to find the outliers which are the fraudulent activities.

2.4. Summary of Published papers according to Data Mining Techniques

The articles analyzed are summarized into a table. These articles summarized based on the data mining techniques used for the research.

Table 2.4-1 Summary of Research Articles

Title of Article	Technique used	Algorithm
Effect of Class Imbalanceness in Detecting Automobile Insurance Fraud(Subudhi and Panigrahi, 2018)	Classification	Support Vector Machine, Decision Tree and Multi Layer Perceptron
Fraud Detection in Automobile Insurance using a Data Mining(Ghorbani and Farzai, 2018)	Clustering	K-Means
Fraud Detection in Health Insurance Claims using Machine Learning Algorithms (“Fraud Detection in Health Insurance Claims using Machine Learning Algorithms,” 2020)	Classification and Clustering	Decision Trees, Naive Bayes, K means

Detecting Fraudulent Claims – A Machine Learning Approach("Detecting Fraudulent Claims - A machine learning approch.pdf," n.d.)	Classification	Generalized Linear Models (GLM), Gradient Boosting Machines (GBM, an ensemble of decision trees) and Neural Networks
Decision Support System (DSS) for Fraud Detection in Health Insurance Claims Using Genetic Support Vector Machines (GSVMs)(Sowah et al., 2019)	Classification	Genetic support vector machines
Naïve Classification Approach for Insurance Fraud Prediction (Batra and Kundra, 2019)	Classification	Naive Bayes
An Efficient Classification Model for Analyzing Skewed Data to Detect Frauds in the Financial Sector (Makki, n.d.)	Classification	Cost-Sensitive Cosine Similarity K-Nearest Neighbor (CoSKNN), K-modes Imbalance Classification Hybrid Approach (K-MICHA)
Research on Integrated Learning Fraud Detection Method Based on Combination Classifier Fusion (THBagging) (Gong et al., 2020)	Classification	tree hybrid bagging
Credit Card Fraud Detection Using AdaBoost and Majority Voting (Randhawa et al., 2018)	Classification, Regression	Naïve Bayes, Decision Tree, Random Forest, MLP network, Feed-Forward Neural Network, Linear Regression, SVM
Auto-Insurance Fraud Detection: A Behavioral Feature Engineering Approach ("AUTO-INSURANCE FRAUD DETECTION," 2020)	Classification	Random Forest

Extreme Gradient Boosting Machine Learning Algorithm For Safe Auto Insurance Operations (Dhieb et al., 2019)	Classification	Extreme Gradient Boosting Algorithm

2.5. Summary and Research Gap

This chapter provided an overview of the existing literatures on fraud detection research area. Also it gives brief explation about different data mining techniques applied in the research domain. Through this litreture review analysis it was learned that there are many data mining techniques like classification, regression, clustering and outlier detections were applied in fraud detection domain with different algorithms. Some new data mining algorithms like XGBoost are applied in fraud detection domain to optimize the fraud detection rate. However, by analyzing these literatures it was identified that lack of data which are tag as fraud claims is the biggest challenge when applying machine learning techniques in this domain. Some researchers have suggested data imbalance methods to solve this problem, however addressing this insufficient data problem will be the main research gap in fraud detection domain.

3. RESEARCH METHODOLOGY

This chapter will present the approaches and techniques that were used to the proposed insurance fraud detection model. The research methodology is divided into five sections. First the data preparation is explained with data set description and preprocessing methods used. Then the Fraud claim detection process will be explained. After that a brief description of data mining algorithms which will be used in the research will be explained. Then the software tools that will be used in the research will be explained. Finally, The chapter ends with the model evaluation methods; which evaluate and measure the model performance will be explained.

3.1. Dataset Description

In this research the dataset was taken from Sri Lanka Insurance motor claims which comprise 19 features and one target variable. It consists of 30100 records in which 26987 are normal clams and 3113 are fraud claims.

Table 3.1-1 Data summary

Category	Count
Paid Claims	26,986
Fraud Claims	3,112

In the data set the attributes can be divided into three categories; personal attributes, Policy Attributes and Claim Characteristics. Following table shows the description of attributes.

Table 3.1-2 Attribute Description

Attribute Name	Type
ACCIDENT_TYPPE	Claim Characteristic
TOTAL_LOST	Claim Characteristic
ACCIDENT_MONTH	Policy Characteristic
ACCIDENT_WEEK	Claim Characteristic
ACCIDENT_WEEK_DAY	Claim Characteristic
ACCIDENT_TIME	Claim Characteristic
CLAIM_MONTH	Claim Characteristic
CLAIM_WEEK	Claim Characteristic
CLAIM_WEEK_DAY	Claim Characteristic
CLAIM_TIME	Claim Characteristic
GAP_IN_DAYS	Claim Characteristic
ESTIMATED_AMOUNT	Claim Characteristic
VEHICLE_CATEGORY	Policy Characteristic
PURPOSE_OF_USE	Policy Characteristic
MAKE	Policy Characteristic
SUM_INSURED	Policy Characteristic
PREMIUM	Policy Characteristic
STATUS	Policy Characteristic
MONTHS_AS_CUSTOMER	Policy Characteristic
FRAUD_OR_NOT	Class Variable

3.2. Motor Fraud Claim Detection Process

When a claim is informed, Claim Department assessed and labeled the claims as normal or rejected. Reasons for rejected claims could be not paid premiums, expire policies, suspicious policies or fraud etc… If the claim is a fraud claim, then it will be labeled as 'Not Consistency'. However, these rejected and fraud claims are small in number with compared to all the claim records, due to complexity nature of detecting them. The claim process is summarized as follows.

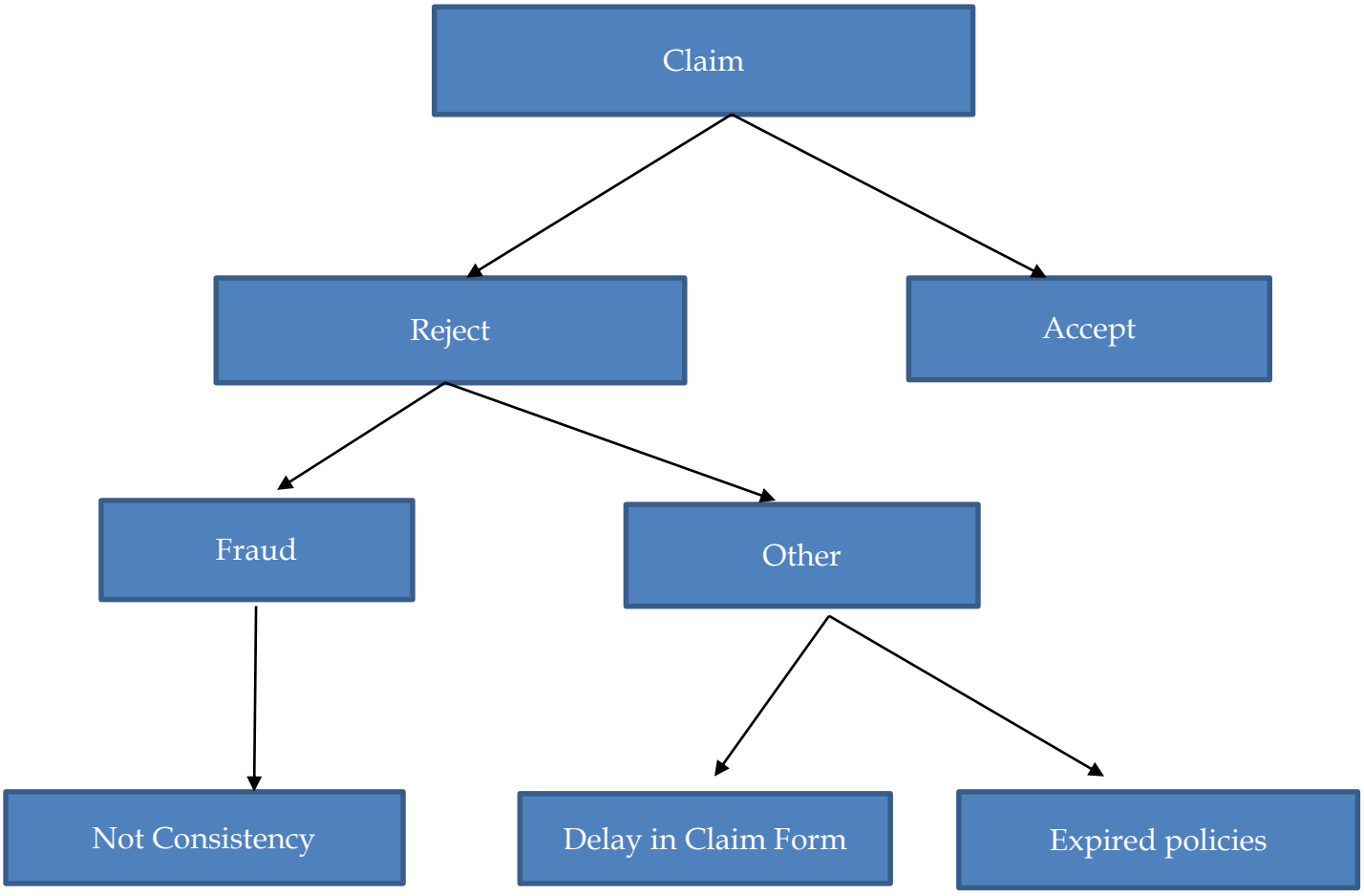


Figure 3.2-1 SLIC Claim Process

3.2.1. Motor Fraud Claim Detection Process as a classification problem

Fraud detection process is a the task of identifying a motor claim as a “FRAUD” or “NOT FRAUD” after comparing it with the already stored similar fraud claims and not fraud claims and in advance, if the detected claim is a fraud , informing relevant stake holders .

Actually fraud detection is a classification task performed specifically on claims. If we can identify attributes which are more related to the class variable, some of classification algorithms can be applied to classify the claim as ‘fraud’ or ‘not fraud’. Following diagram explain the high level procedure in identifying a fraud claim.

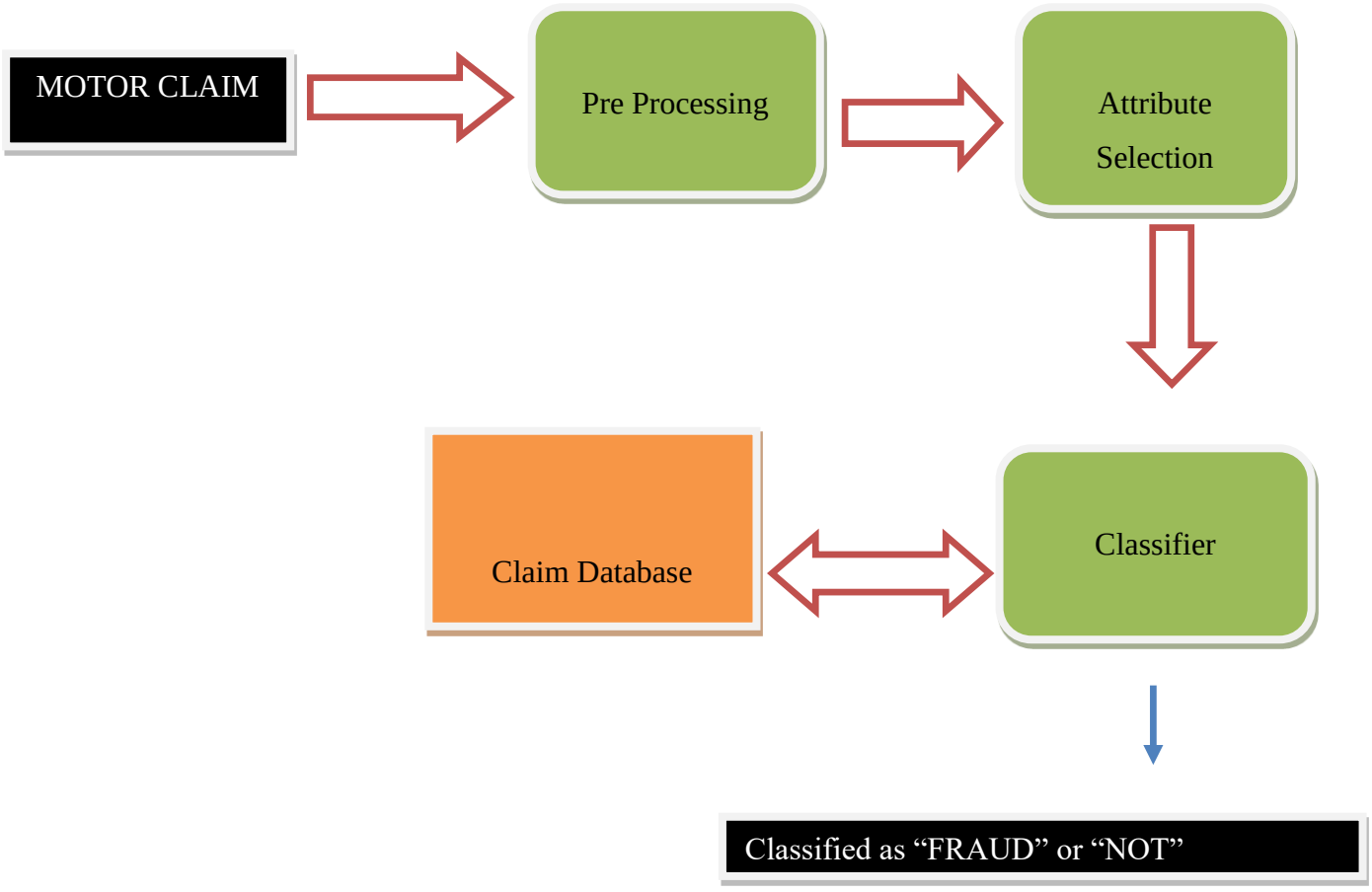


Figure 3.2-2 High-level architecture for identify a fraud claim

3.3. Research Process

Raw data will be cleansed and anonymized by applying techniques with categorization and generalization. After transforming raw data into features, feature selection methods will be applied to select main features that affect to fraud claims. Also domain knowledge is critical in identify what are the features that might be relevant for detecting fraud claims and it is assumed that the researcher's working experience at Sri Lanka Insurance Corporation as an employee will be helpful to gain thorough domain knowledge in this area and applied it in several stages.

Motor claim Fraud detection can be viewed as a classification problem. Random Forest, XGBoost and Artificial Neural Network classifiers will be used to detect claims are fraudulent or not and categorized them into different types of fraud. These algorithms will be analyzed and evaluated by dividing the data set into training, validating and testing. However, the objective is to propose a machine learning solution with high accuracy. Therefore model performances will be evaluated and compared by applying some critical evaluation criteria of recall, precision, ROC curve and training time. Finaly it is assumed to propose the best model among them using these techniques. The high level architecture is shown in the following diagram.

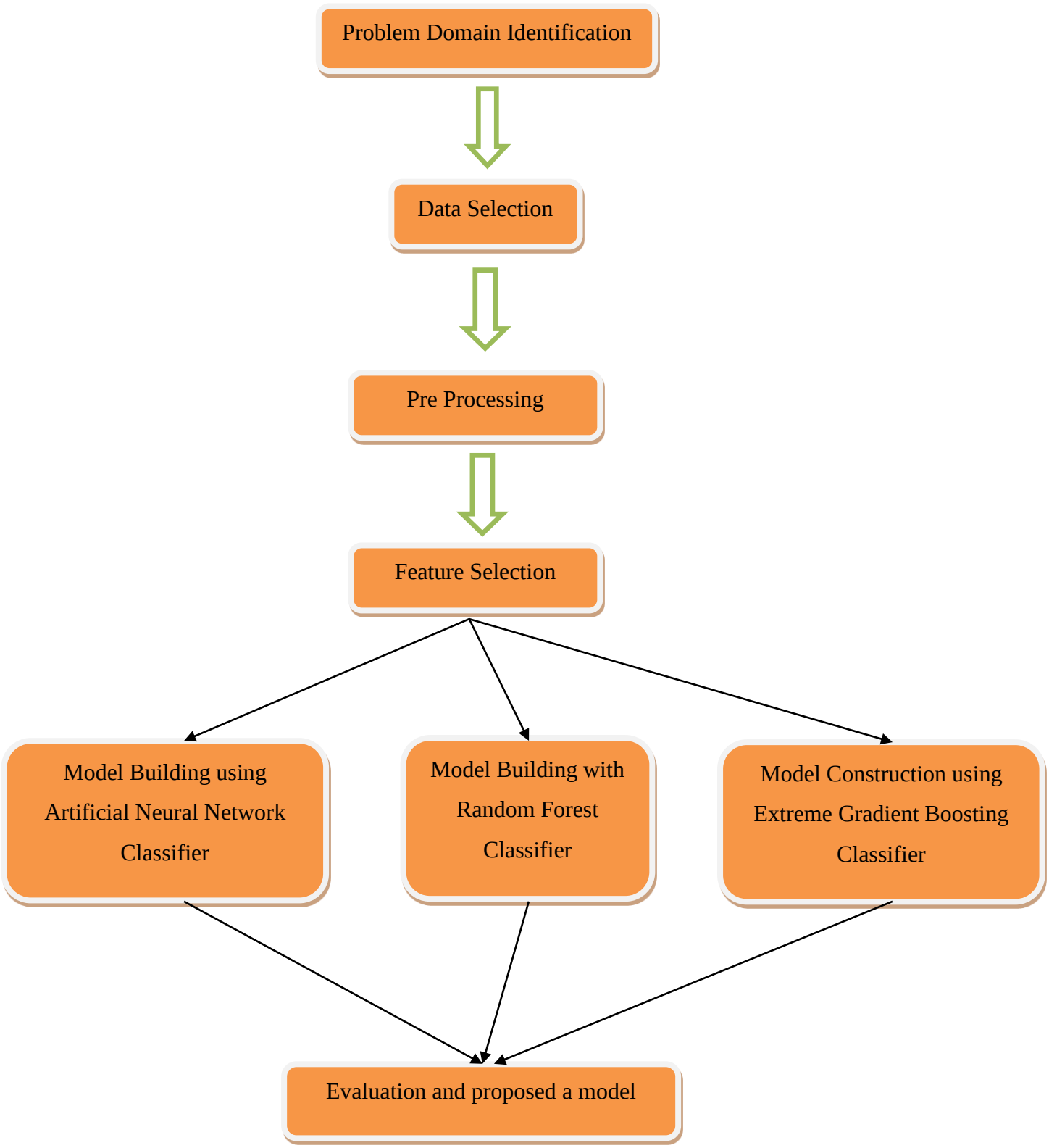


Figure 3.3-1 Research Process

3.4. Fraud Classifier Models

Artificial Neural Network, Random Forest and XGBoost algorithms will be used as classifiers in this research. A brief description about these classifiers and how it used in fraud detection will be explained in this section.

3.4.1. Random Forest Classifier

In order to apply random forest classifier, first it should be mentioned about the decision tree classifier. Decision Tree is a predictive model that is using a set of binary rules to predict a target value. It can be also used for regression purposes as classification purposes. Decision trees are relying on partitioning the feature vector over class variable for each feature set. This can be constructed using a tree structure, which the algorithm takes the form of a tree. Followings are main components of a decision tree.

- Decision node : single attribute which decide by the algorithm
- Leaf Node : target attribute value
- Edge : Split the feature
- Path : Explainthe final decision

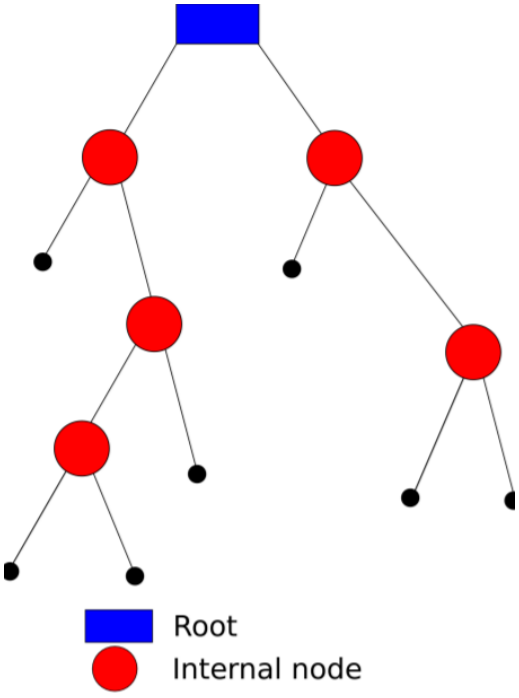


Figure 3.4-1 Decision Tree Architecture

In decision tree, different algorithms are used to determine the best splitting feature at a node. ID3 (Iterative Dichotomiser 3) is a simple and efficient algorithm which is used by the decision tree to make the split. It uses two concepts which is Entropy and Information Gain when creating the decision tree from top to bottom. Entropy can be viewed as the measure of uncertainty where higher the entropy, higher the uncertainty. Entropy is used by Information gain to findout what attribute is the best to split at a node.

The main advantage in Decision tree is its easiness in interpreting the decision rules. Another advantage is its robustness with regard to outliers in training data. However, its main disadvantage is overfitting the data which will give unexpected results at the end. Following figure shows a data set and its constructed decision tree using ID3 algorithm.

Day	Outlook	Temp.	Humid.	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No
D15	Sunny	Cool	Normal	Weak	Yes
D16	Rain	Mild	Normal	Strong	Yes
D17	Sunny	Mild	Normal	Strong	Yes
D18	Overcast	Mild	High	Strong	Yes
D19	Overcast	Hot	Normal	Weak	Yes
D20	Rain	Mild	High	Strong	No

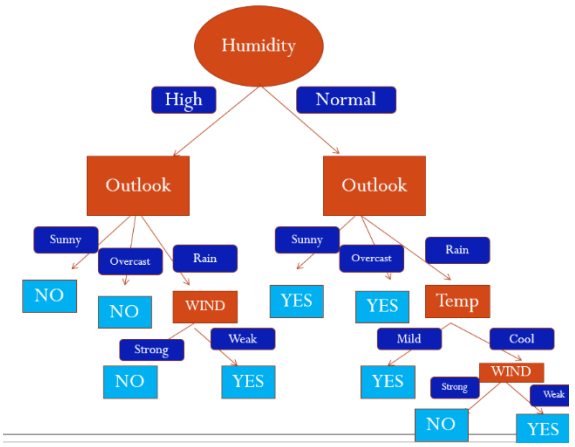


Figure 3.4-2 Decision Tree Example

By considering the limitations in decision trees, an extension of the decision trees is considered for classification. Random forest can be considered as such kind of an extension or improvement of the decision trees. Random forest is an ensemble model which combines the results of different decision trees to predict the final classification. Random forest also a Bootstrap Aggregating method, also known as bagging, which can be classified as a combination of homogeneous weak learner models that learns independently in parallel and then combined for determining the model average. Random forest models mitigate the overfitting problem encountered in decision trees by building multiple decision trees and combining them such that final performance of the model is

improved. Random forest uses boostraping sampling technique which will select training data for each decision tree with replacement technique. So random forest will utilize boostraping method by each decision contains different subsets. Also random forest choses only certain number of features to train each decision tree.

Once the outputs of each decision tree is collected, the output of each decision tree is aggregated to for getting the final result using the voting method. Therefore, this method can be classified as a bagging method since it uses majority voting technique to select the final result. Since Random forest uses multiple decision trees to predict the results and it uses different subset of data to trained the model it ensures generalization which improve the efficiency of the model. Following figure shows building of the random forest from the given dataset.

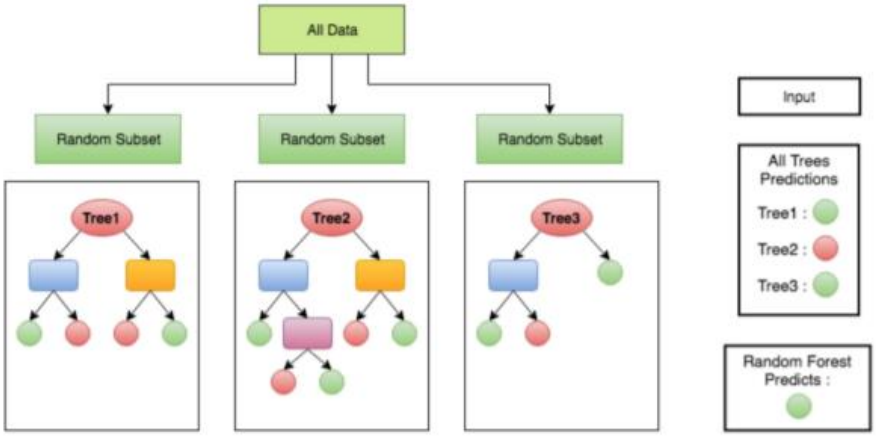


Figure 3.4-3 Random Forest Architecture

3.4.2. XGBoost Classifier

Boosting Algorithm

Boosting is also an ensemble learning method which build a strong classifier from multiple weak classifiers in a model. However, this is a sequential learning model where first it builds a model from training data and based on that model it builds another model which tries to reduce the errors present in first model. Every new training data sample contains the elements that were misclassified by previous model. In this process, for any incorrect misclassified samples are assigned by larger weights and correctly classified samples are assigned by lower weights. Boosting will not change the previous model and only corrects the next model by learning from its mistakes. Boosting is greedy algorithm, therefore it is better to set a stopping criteria like early stopping or depth of tree in decision tree models to prevent overfitting of data. Boosting algorithms differs from the bagging algorithms such that boosting algorithms controls both variance and bias in a model whereas bagging algorithms only controls variance in the model. Following figure illustrates a boosting algorithm process.

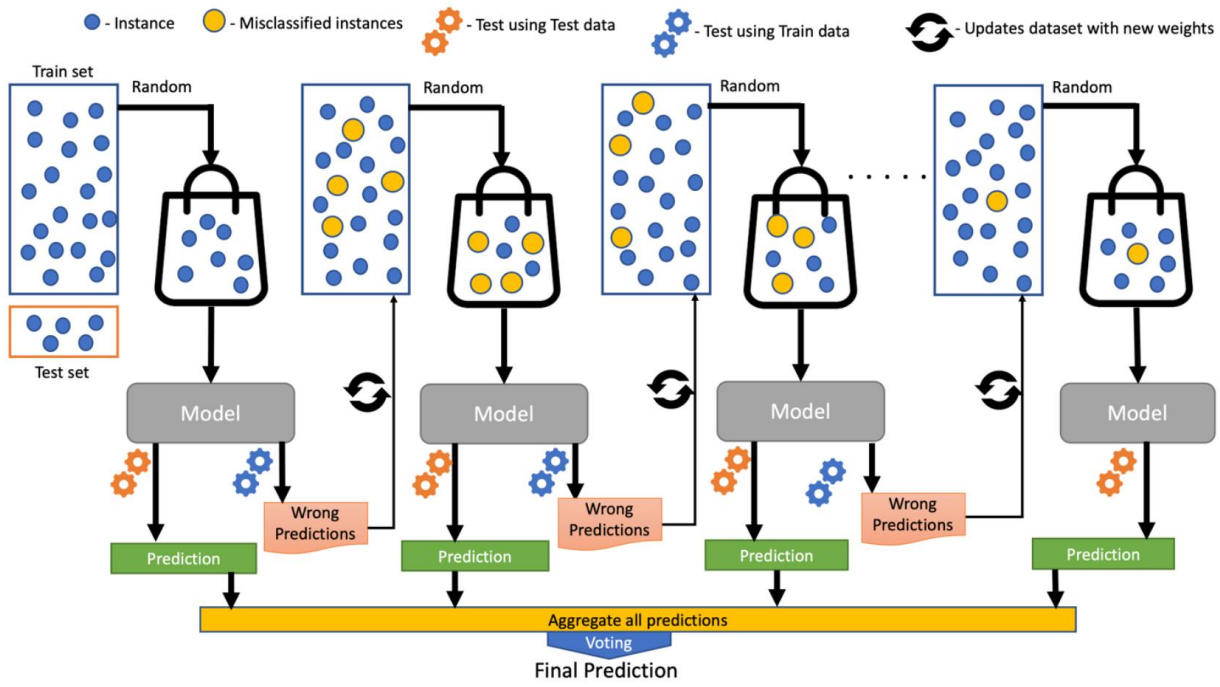


Figure 3.4-5 Boosting Algorithm Process

Adaptive Boosting

There are two main boosting algorithms applied in machine learning models which are Adaptive(Ada) Boosting and gradient boosting. In ada boosting it used multiple iterations to build a strong learner. It iteratively adds weak learners and builds a strong learner. During each iteration phase, a weak learner is added to the current model, and the weighting vector is adjusted to misclassified models in previous iterations. The final model has higher accuracy than the previous weak classifiers. Following figure illustrates the ada boosting formula and its procedure.

$$F_i(x) = F_{i-1}(x) + f_i(x)$$

F(i) is current model, F(i-1) is previous model and f(i) represents weak model

Equation 3.4-1 Adaptive Boosting Formula

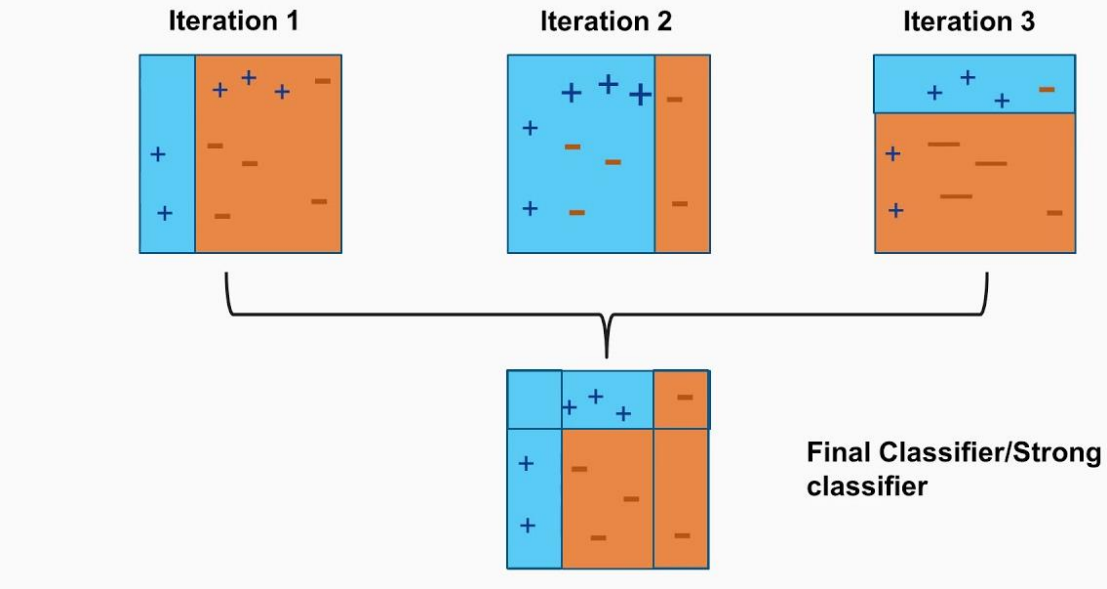


Figure 3.4-6 Ada Boosting Process

Gradient Boosting

Gradient boosting can be called as a generalization of AdaBoost. The main difference in gradient boosting is that it includes a loss function such that main objective of the algorithm is to minimize the loss function. It satisfies the objective by adding weak learners using a technique called gradient descent optimization. Since it uses a differentiable loss functions for minimizing the error, it can be considered not only for binary classification problems but also for multi class classification and regression and many more.

Gradient Boosting has three main attributes :

- **Weak Learner** : Which classify the data. These are mostly random trees however other classifiers can be used.
- **Loss Function** : Estimate how best is the model for given data.
- **Additive model** : Sequential and iterative process for adding decision trees one for each iteration. Each iteration should reduce the loss function.

Gradient boosting stops splitting the node when it found a negative loss. Therefore, it can overfit a dataset quickly and optimization methods are used to improve the algorithm by reducing overfitting.

XGBoosting

Extreme Gradient Boosting, is a supervised learning method which optimized the gradient boosting algorithm to be highly effective and efficient. It is an optimization technique for gradient boosting which was proposed by Chen and Guestrin., Decision trees are used as the weak learners and it allows parallel processing which reduce overfitting and faster than the standard gradient boosting. Main difference in Gradient boosting and XGBoost is that xgboost splits the trees up to the maximum depth which specified. Also XGBoost has a cross validation feature, therefore it is easier to identify boosting count at each run. However, several parameters need tunning to get the best results with xgboost algorithm.

XGBoost also has four main features:

Gradient Tree Boosting: The model trains as additive sequence where it optimized the model.

Regularized Learning: Selecting a model using predictive functions by reducing over-fitting.

Shrinkage: To further prevent overfitting shrinkage ecan be used. The Shrinkage technique is introduced by Friedman in which it scales newly added weights by a factor η after each step of tree boosting. Shrinkage will reduce the influence of each added tree and leaves space for another tree to improve the prediction.

Column Subsampling: Column subsampling prevents overfitting even more. It also speeds up the computations of the parallel algorithm.

XGBoost algorithm Parallelizes the tree construction using all of CPU cores of the machine during training. Also XGBoost is designed to make optimal use of hardware of the machine by allocating internal buffers in each thread, where the gradient statistics can be stored. The algorithm is adjustable to use distributed Computing for training very large datasets using a cluster of machines. In tree learning, the most time-consuming part is sorting the data. XGBoost uses column blocks in compressed format to reduce the cost of sorting. By considering these performance improvemts, XGBoost can be considered as one of the fastest machine learning algorithm at present.\

3.4.3. Artificial Neural Network Classifier

An Artificial Neural Network (ANN) is an information processing network that the architecture is inspired by biological nervous systems. Neural Networks are mostly used because its ability to learn quickly. They figure out how to perform their functions by their own. It determines their functions based by inputs. It also has the ability to generalize to the situation. Precisely, it has the ability to predict outputs for inputs that has not been taught how to deal with.

A Neural Network has three Layers of units.

Input layer	-	Raw information that input into Neural Network
Hidden Layers	-	Connect input and output layers
Output layer	-	Outputs the predictions

Fraud Detection and Neural Network

A fraud detection system depends on the selected classifier of the system. Actually Neural Network can be used as a classifier in fraud detection system. It can train to identify a given face as 'fraud' or 'not'. Some attributes from the dataset can be given to the system and it can be trained to recognize a claim for the given features.

For a given dataset, the first step is to select a set of features or attributes from the universe of features that can represent the claim. This feature vector then use as the input vector of the Neural Network. The Network will be trained such that, if the Network gets another feature vector which is close to the current vector it will recognize it and output the result '1' otherwise '0'. This is a supervised learning process and back propagation technique can be used in hidden layers.

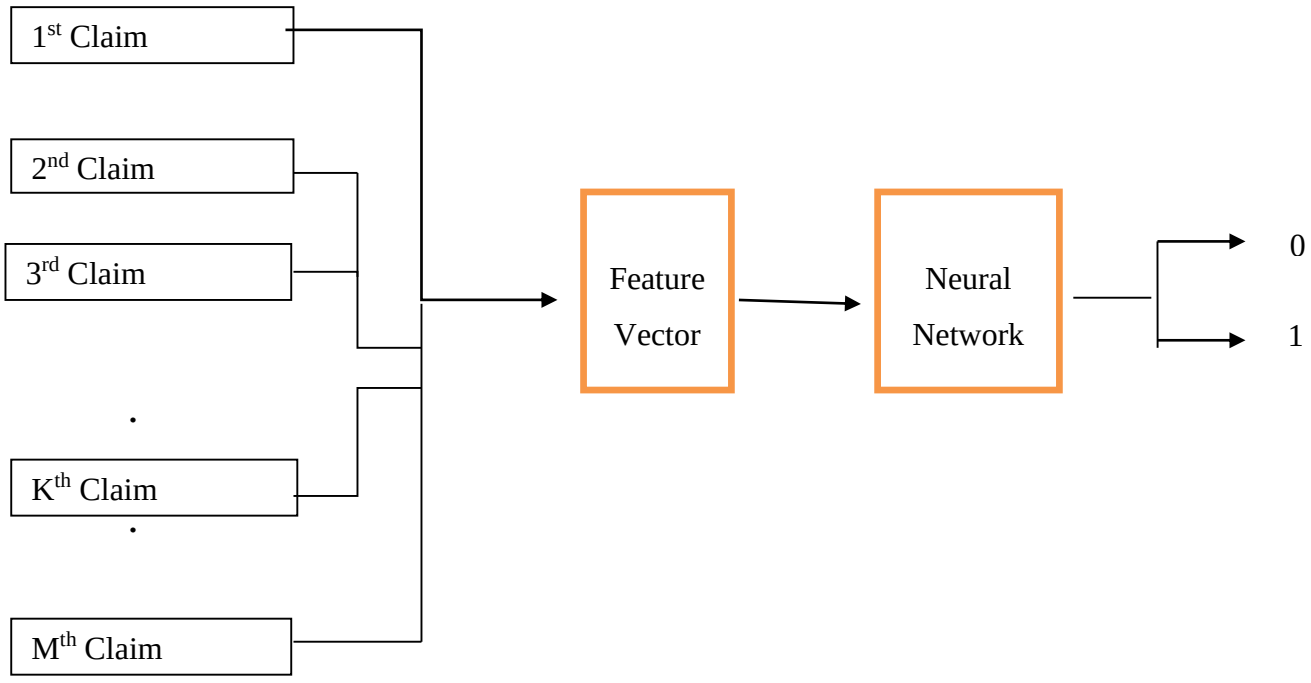


Figure 3.4-8 Fraud Detection in ANN

When a new claim is considered for classification, the claim is mapped with features. Then the claim is assign to a feature vector. Then this feature vector is fed into the trained Neural Network. Neural Network compares the new feature vector with already existing feature vectors. If it successfully identified, then it outputs '1'. Otherwise outputs '0'. Following Figure illustrates this process.

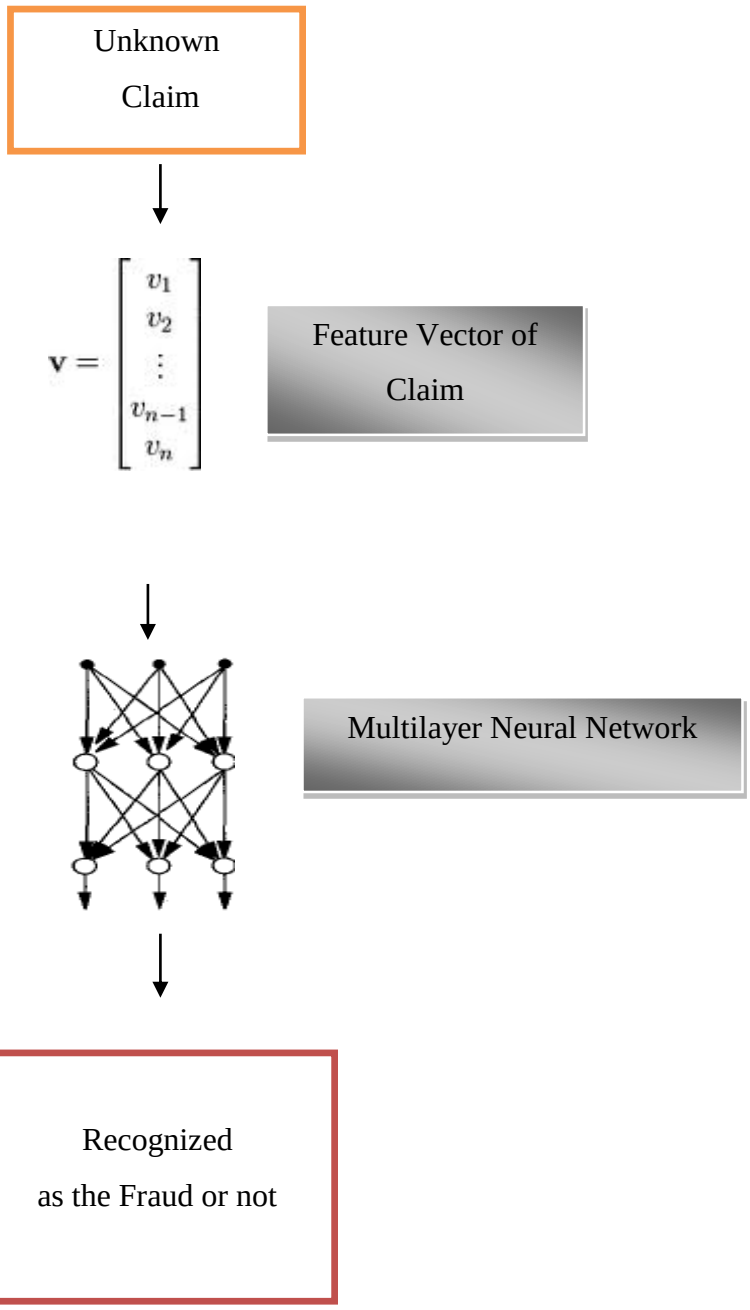


Figure 3.4-9 Unknown Claim Classification by ANN

3.5. Imbalance Problem

Imbalance dataset is a dataset that target classes of the dataset is distributed unequally. When applying machine learning algorithms for imbalance data set, problems can be created due to unequal of target class. For example, if accuracy is used when evaluating the model, it will give much high percentage value since the model will biased towards the majority class of the target variable. As mentioned in the data description section earlier, dataset for this research is also highly imbalance. Therefore, it must create a balance dataset from this imbalance data to apply the machine learning algorithms. There are few techniques used to address the imbalance problem that are used in practice and they will create an equalize representation of all classes. Following figure shows the imbalance data set and balance data set after applying a data balancing technique.

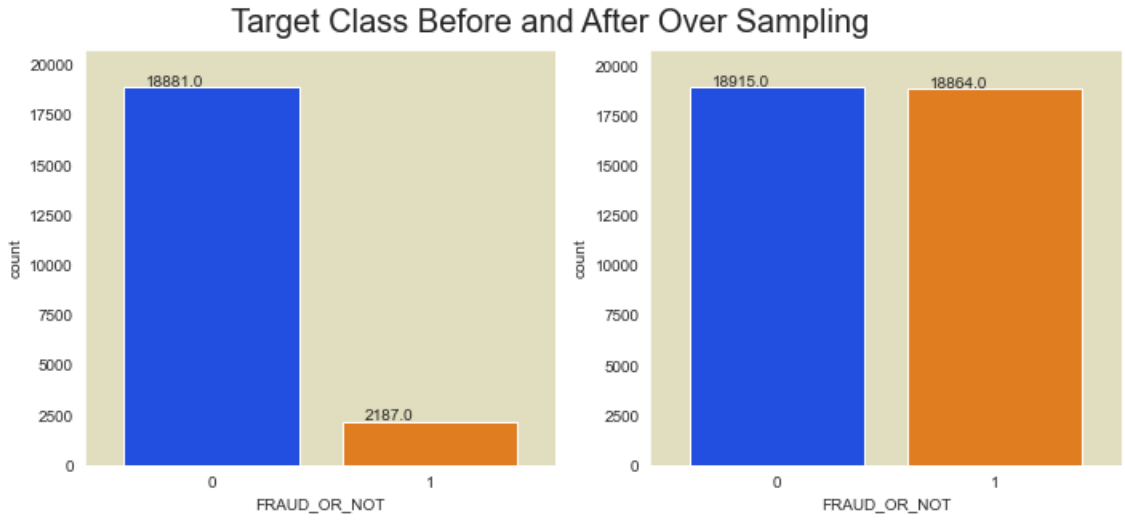


Figure 3.5-1 Target class before and after Data Balancing

3.5.1. Random Undersampling for balancing data

Undersampling technique removes data from the majority class to reduce the overrepresented gap. It randomly removes samples (with or without replacement) until the data in the majority class becomes close to the number of observations in the minority class. However, major disadvantage of undersampling is that it will be lost significant chunk of the data, which contains valuable information andas a result, not get significant results.

3.5.2. Random Oversampling for balancing data

Oversampling can be classified as resampling of the minority class data to equal the of majority data. Random oversampling makes multiple copies of minority class and increasing the number of total observations for the minority class.

3.5.3. Creating Synthetic data for balancing (SMOTE)

The problem with repeating the data in random oversampling is that it does not provide any extra information. However, by creating synthetic data using an algorithm, it will increase the minority class data and also increase the information about the data. One such technique which is used in practice is the SMOTE (Synthetic Minority Oversampling technique). This is also a oversampling technique and it will create synthetic data points for the minority class. For this reseach SMOTE technique will be applied for balancing the data.

3.6. Performance Evaluation

When building a machine learning model, first the model is trained using the training data then it tests with the testing data to evaluate model accuracy. However, not only that is enough and it is important to check the generalization capability of the model. Some evaluation metrics are available to check the capability but these evaluation metrics are depended on the type of the problem, for example whether it is a classification or clustering problem or whether it is a balance or imbalance problem. For this research, it will only consider the evaluation criteria's related to the classification problem since the fraud claim detection was identified as a classification problem.

3.6.1. Confusion matrix

The confusion matrix is constructed using statistics of True Positive-TP, True Negative-TN, False Positive-FP and False Negative-FN. These statistics are calculated using the of actual and predicted values. Confusion matrix is the most commonly used evaluation criteria in machine learning since it is easiness and it can be used to calculate other evaluation criteria's like accuracy, recall, precision, etc. This is an 2x2 matrix for binary classification problem as shown in figure.

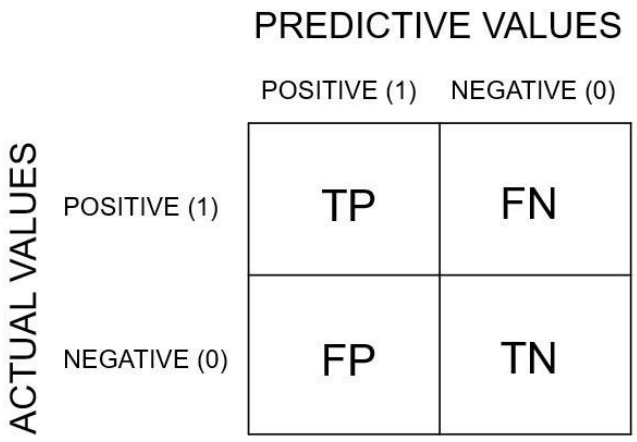


Figure 3.6-1 Confusion Matrix

True Positive (TP) - Actual output is positive and the predicted output is also positive.
False Negative (FN) – Actual output is positive but the predicted output is negative.
False Positive (FP) - Actual output is negative but the predicted output is positive.
True Negative (TN) - Actual output is negative and the predicted output is also negative.

3.6.2. Recall

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

Equation 3.6-1 Recall equation

Recall is the percentage of true positives count to the actual positive count. Actually, recall is how many of true positives recalled from the true positive count. Recall also called as sensitivity.

For example, consider the claim count of 100 where it contains 20 fraud claims. Suppose a model identifies 15 fraud claims, only 12 claims were true fraud claims (TP), while rest were normal claims (FP). Therefore, recall is 12/20.

3.6.3. Precision

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

Equation 3.6-2 Precision Equation

Precision is the fraction of true positive count over the true positives and false positives, which is shown in equation. In simple terms, precision is how many of the found positive count were true positives. In the previous example of identifying fraud claims, the precision value is 12/15.

3.6.4. F1 Score

$$F1\ score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Equation 3.6-3 F1 Score equation

F1 Score is the harmonic mean of the recall and precision. Its values ranged from 0 to 1, where 0 is considered as weak, and 1 is considered as best.

3.6.5. Area Under Receiver Operating Characteristic curve (ROC)

Area Under Receiver Operating Characteristic curve or ROC is used as most common evaluation criteria in machine learning. It gives how good a model performs when used at different probability values. In classification problems default probability threshold is set to 0.5. ROC is a plot between True positive rate, also called as sensitivity and False Positive Rate. False Positive Rate which can be calculated as (1-Specificity).

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

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

Equation 3.6-4 Sensitivity and Specificity Equations

Sensitivity and specificity are inversely related such that when decreasing the probability threshold sensitivity increases and specificity decreases and when increase the threshold, sensitivity decreases while specificity increases. Area under the curve is calculated from ROC, which is the probability that a model will rank a randomly chosen positive instance higher than a randomly chosen negative. Figure shows example of a ROC curve where the orange curve shows the ROC of the model in which AUC is 0.92 and blue dotted line shows the ROC of a random model in which AUC is 0.5.

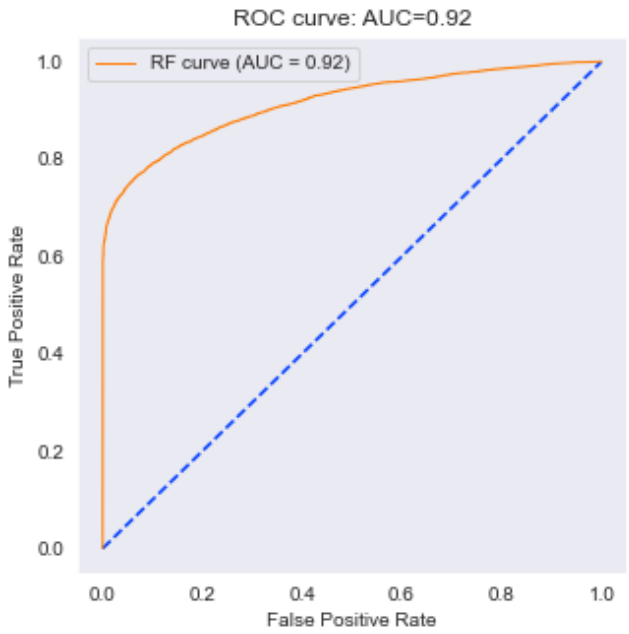


Figure 3.6-2 ROC Example

3.6.6. Precision Recall Curve

In unbalanced classification problem Precision Recall Curve is more important than ROC curve. It is based on precision and recall criteria and the plot shows the values of precision and recall at different probability values. The area under curve can be used to evaluate the performance same as ROC curve however beat for imbalance problem.

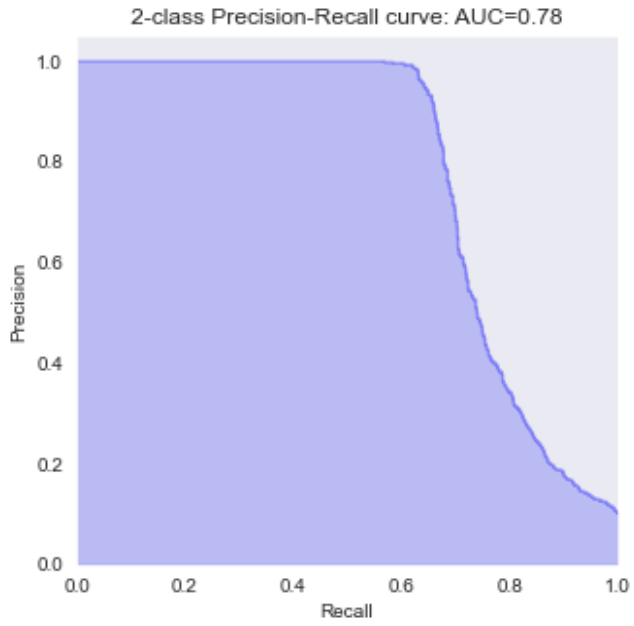


Figure 3.6-3 PR Curve Example

3.6.7. Cross Validation

Cross validation is a method to further evaluate a machine learning model for given dataset. The technique has a parameter K, which identify number of folds where given dataset is to be split into. Therefore, it is mostly called K-Fold-Cross-Validation since it is used k folds to split the data. When building a machine learning problem, first the data set is split into a training and a testing dataset. However, in K Fold CV, the training set is further split into K number of subsets or folds. Then iteratively the algorithm fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold which is called the validation dataset.

As an example, when fitting a model with K = 5, in first iteration, the training dataset is divide into 5 folds and choose the first fold as a test dataset and remaining folds as a training set. Then train the model using the last four folds and evaluate on the first The second time the model will be trained on the first, third, fourth and fifth fold and evaluate on the second fold. The process is repeated for 3 more times, each time evaluating for a different fold. At the end of training, performance is averaged for each fold and decide a final validation metric for the model. Following figure shows how to apply the 5-fold cross validation for a given dataset.

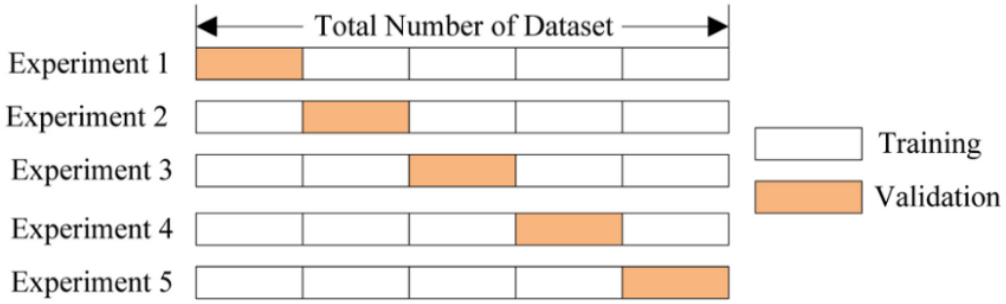


Figure 3.6-4 Cross validation Example

3.7. Hyperparameter Tuning

There are two kind of parameters can be found when training a machine learning model, which are model parameters and hyperparameters. A model parameter can be classified as a configuration variable that is internal to the model and its value can be estimated from data. They are required by the model when doing predictions and often saved as part of the learned model. Some examples of model parameters can be shown as weights of neural networks, support vectors in a SVM model etc. Hyperparameter is a configuration that is external to the model where its value cannot be estimated from the training data. Hyperparameters are oftenly used to help estimate the model parameters and are specified manually by the researcher. Since they are manually set, it is very important to be rightly tuned the hyperparameters to get the best performance from the model.

Hyperparameter tuning relies on experimental results rather than theoretical assigning, and therefore the best method to get the optimal settings is to try many different combinations of parameters and evaluate the performance of each model. However, evaluating each model only on the training dataset can lead to model overfitting, which is a model perform well for training set but perform poorly for testing data. Apparently an overfit model may look good for training set but may useless for real applications. Cross validation technique which discussed previously, can be applied to tuning the hyperparameters through optimizing for non-overfitting the model. In this research Random Forest and XGBoost algorithms are having hyperparameters and they will be tuned using cross validation method.

In hyperparameter tuning, many iterations is performed on the entire K-Fold CV, each time using different model combinations. Then all of the models will be compared, select the best one, train it on the full training set, and then evaluate on the testing set. This is a complex process where each time it needs to assess a different set of hyperparameters by splitting the training data into K fold and train and evaluate K times. If there are 5 sets of hyperparameters and are using 10 Fold CV, that represents 50 training iterations. There are two methods which are Grid Search and Random Search that can be used to simply this process in Python programming environment.

3.7.1. Grid Search for Cross Validation

In grid search method, first need to prepare a list of values of hyperparameters and then search the best combination based on cross validation score. In grid search only the prelist will be considered for the search. Grid search will give the best combination but it takes time.

3.7.2. Random Search for Cross Validation

In random search method, it tries random number of combinations from range of possible values. This is good for testing wide range of values since it reaches a good combination very fast, but have a disadvantage that it does not guarantee to give the best combination since it selects the combination randomly. For this research random Search for Cross Validation will be used since it's quicly approach to a good combination.

3.7.3. Hyperparameters tuning in Random Forest

First it is better to identify the hyperparameters used in Random Forest classifier in Python Environment. Following figure shows the available hyperparameters.

```
In [59]: from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor(random_state = 42)
         from pprint import pprint

         print('Parameters currently in use:\n')
         pprint(rf.get_params())

Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Figure 3.7-1 Random Forest Hyperparameters

However, for this research hyperparameter tuning is performed only for following hyperparameters with 3-fold cross validation since they are considered as most important parameters for the model.

- n_estimators - Number of trees in the forest
- max_features - Maximum number of features considered for splitting a node
- max_depth - Maximum number of levels in each decision tree

3.7.4. Hyperparameters Tuning in XGBoost

Following are the hyperparameters used in the XGBoost classifier in Python environment.

```
In [19]: from xgboost import XGBClassifier
         xgb = XGBClassifier(random_state = 42)
         from pprint import pprint

         print('Parameters currently in use:\n')
         pprint(xgb.get_params())

Parameters currently in use:

{'base_score': None,
 'booster': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'gamma': None,
 'gpu_id': None,
 'importance_type': 'gain',
 'interaction_constraints': None,
 'learning_rate': None,
 'max_delta_step': None,
 'max_depth': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'n_estimators': 100,
 'n_jobs': None,
 'num_parallel_tree': None,
 'objective': 'binary:logistic',
 'random_state': 42,
 'reg_alpha': None,
 'reg_lambda': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'use_label_encoder': True,
 'validate_parameters': None,
 'verbosity': None}
```

Figure 3.7-2 XGBoost Hyperparameters

For this research, hyperparameter tuning is performed only for following hyperparameters with 3-fold cross validation since they are considered as most important parameters for the model.

- learning_rate - Step size used to prevent overfitting. Range is [0,1].
- max_depth - Maximum depth of the tree.
- min_child_weight - Minimum sum of weights of all observations required in a child. It will also control the overfitting.
- gamma - Minimum loss reduction which required to make a split at a node.
- colsample bytree - Fraction of columns to be randomly sampled for each tree.

3.8. Programming Environment

Python Programming environment is used to implement the methods and processes which discussed earlier in this chapter.

4. IMPLEMENTATION AND RESULTS

This chapter will present the implementation of the methods discussed in the methodology chapter with data preparation, feature selection, model building and evaluation the models. The implementation and results chapter is divided into five sections. First the data preparation is explained with missing value imputation and encoding used to encode the string variables. Then the feature selection implementation will be carried out. After that model building with random forest, XGBoost and Artifitial Neural Network will be discussed. Finally, the chapter ends with the model evaluation by comparing the built model metrics.

4.1. Data Description

Dataset is the integral part of this research since it uses machine algorithms to build the models. The dataset which was mentioned in the methodology chapter, is uploaded to the python environment for further analysis. The figure shows the snapshot of the dataset in python environment.

```
In [2]: import pandas as pd

In [15]: data = pd.read_csv("Final Data set v3.csv")

In [16]: data.head()
```

Out[16]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_MONTH	ACCIDENT_WEEK	ACCIDENT_WEEK_DAY	ACCIDENT_TIME	CLAIM_MONTH	CLAIM_WEEK	CLAIM_WE
0	ACCIDENT	NORMAL	5	18	2	Afternoon	8	34	
1	ACCIDENT	NORMAL	3	11	3	Early Morning	4	17	
2	ACCIDENT	NORMAL	10	42	4	Night	11	45	
3	ACCIDENT	NORMAL	9	36	3	Morning	9	38	
4	ACCIDENT	NORMAL	11	46	4	Late Night	1	4	

Figure 4.1-1 Dataset snapshot

The dataset contains 30098 motor claim records out of which only 3112 are fraudulent claims. Thefore the dataset is highly imbalanced as the positive class accounts for only 10.3% of the total claims. The imbalanced class distribution can be visualized in a figure below.

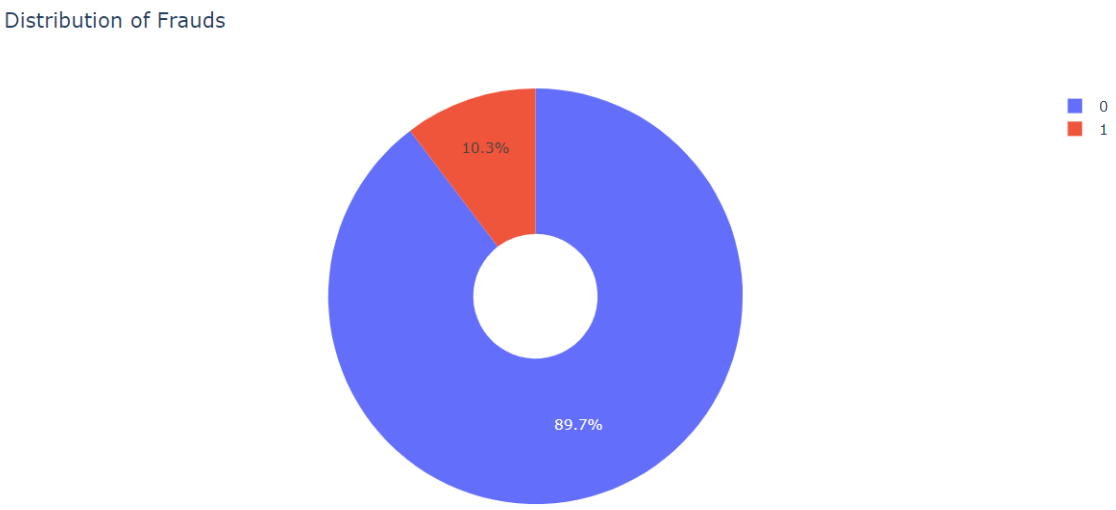


Figure 4.1-2 Pie Chart of distribution of Fraud claims

4.1.1. Missing values Imputation

After uploading the dataset, each feature is examined for missing values. Since missing values can misclassify the machine learning models, it is important to analyze missing data and correct them using suitable imputation method. Following figure shows which features contained missing values.

```
In [16]: data.isnull().any()

Out[16]: ACCIDENT_TYPPE        False
         TOTAL_LOST             True
         ACCIDENT_MONTH        False
         ACCIDENT_WEEK         False
         ACCIDENT_WEEK_DAY     False
         ACCIDENT_TIME          True
         CLAIM_MONTH           False
         CLAIM_WEEK            False
         CLAIM_WEEK_DAY        False
         CLAIM_TIME            False
         GAP_IN_DAYS           False
         ESTIMATED_AMOUNT      False
         VEHICLE_CATEGORY       True
         PURPOSE_OF_USE         True
         MAKE                  False
         SUM_INSURED            True
         PREMIUM                True
         STATUS                 True
         MONTHS_AS_CUSTOMER     True
         FRAUD_OR_NOT          False
         dtype: bool
```

Figure 4.1-3 Attributes with Null values

Missing values can be replaced with mode of the feature, 'No Category' type for string variables and 0 for numerical values. Following missing value imputation methods applied for each feature with missing values.

```
In [19]: # missing value treatment using fillna

         data['TOTAL_LOST'].fillna(data['TOTAL_LOST'].mode()[0], inplace = True)
         data['ACCIDENT_TIME'].fillna('NO', inplace = True)
         data['CLAIM_TIME'].fillna('NO', inplace = True)
         data['VEHICLE_CATEGORY'].fillna('NO', inplace = True)
         data['PURPOSE_OF_USE'].fillna('NO CATEGORY', inplace = True)
         data['MAKE'].fillna('NO CATEGORY', inplace = True)
         data['SUM_INSURED'].fillna(0, inplace = True)
         data['PREMIUM'].fillna(0, inplace = True)
         data['STATUS'].fillna('NO CATEGORY', inplace = True)
         data['MONTHS_AS_CUSTOMER'].fillna(0, inplace = True)
```

Figure 4.1-4 Missing value Imputation

After applying missing value imputation, the dataset is further checked for missing values and confirmed no missing values are available in data.

```
In [20]: data.isnull().any()

Out[20]: ACCIDENT_TYPPE        False
         TOTAL_LOST            False
         ACCIDENT_MONTH        False
         ACCIDENT_WEEK         False
         ACCIDENT_WEEK_DAY     False
         ACCIDENT_TIME         False
         CLAIM_MONTH           False
         CLAIM_WEEK            False
         CLAIM_WEEK_DAY        False
         CLAIM_TIME            False
         GAP_IN_DAYS           False
         ESTIMATED_AMOUNT      False
         VEHICLE_CATEGORY      False
         PURPOSE_OF_USE        False
         MAKE                  False
         SUM_INSURED           False
         PREMIUM               False
         STATUS                False
         MONTHS_AS_CUSTOMER    False
         FRAUD_OR_NOT          False
         dtype: bool
```

Figure 4.1-5 Checking for null values

4.1.2. String Variables Encoding

First it is important to recognized the string variables in the dataset. Following figure shows the method applied in python to recognize the string variables.

```
In [22]: cat_data = data.select_dtypes(include=['object']).copy()

In [23]: cat_data.head()
```

Out[23]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_TIME	CLAIM_TIME	VEHICLE_CATEGORY	PURPOSE_OF_USE	MAKE	STATUS
0	ACCIDENT	NORMAL	Afternoon	Morning	Dual Purpose	Private	TOYOTA	Mr
1	ACCIDENT	NORMAL	Early Morning	Evening	Car	Private Person	MITSUBISHI	Mrs
2	ACCIDENT	NORMAL	Night	Morning	Car	Private Person	MITSUBISHI	Mrs
3	ACCIDENT	NORMAL	Morning	Afternoon	Car	Private Person	TOYOTA	Prof
4	ACCIDENT	NORMAL	Late Night	Afternoon	Car	Private Person	TOYOTA	Prof

Figure 4.1-6 String Variables

When building machine learning models, all features should be in numerical form. Therefore, it is important to convert string variables into numerical variables by applying a suitable method. Python has a method which convert categorical string variables into numerical type and it is applied for the data set. Following figures shows the applied method and dataset after encoding.

```
In [25]: data.ACCIDENT_TYPPE = pd.Categorical(data.ACCIDENT_TYPPE).codes
         data.TOTAL_LOST = pd.Categorical(data.TOTAL_LOST).codes
         data.ACCIDENT_TIME = pd.Categorical(data.ACCIDENT_TIME).codes
         data.CLAIM_TIME = pd.Categorical(data.CLAIM_TIME).codes
         data.VEHICLE_CATEGORY = pd.Categorical(data.VEHICLE_CATEGORY).codes
         data.PURPOSE_OF_USE = pd.Categorical(data.PURPOSE_OF_USE).codes
         data.MAKE = pd.Categorical(data.MAKE).codes
         data.STATUS = pd.Categorical(data.STATUS).codes

In [26]: data.head()
```

Out[26]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_MONTH	ACCIDENT_WEEK	ACCIDENT_WEEK_DAY	ACCIDENT_TIME	CLAIM_MONTH	CLAIM_WEEK	CLAIM_WE
0	0	0	5	18	2	0	8	34	
1	0	0	3	11	3	1	4	17	
2	0	0	10	42	4	6	11	45	
3	0	0	9	36	3	4	9	38	
4	0	0	11	46	4	3	1	4	

```
In [24]: data.shape
Out[24]: (30098, 20)
```

Figure 4.1-7 Categorical Variable Encoding

4.2. Feature Selection

Feature selection gives most important features which is relates with class variables. Model effiency and run time can be improved by removing unwanted features from the dataset. K best method and Heatmap is used to select the best features from the dataset.

4.2.1. Heatmap for Feature Selection

Heatmap plots the correlation between variables in a visualization graph. It highlights the correlations in bold colour if it exists.

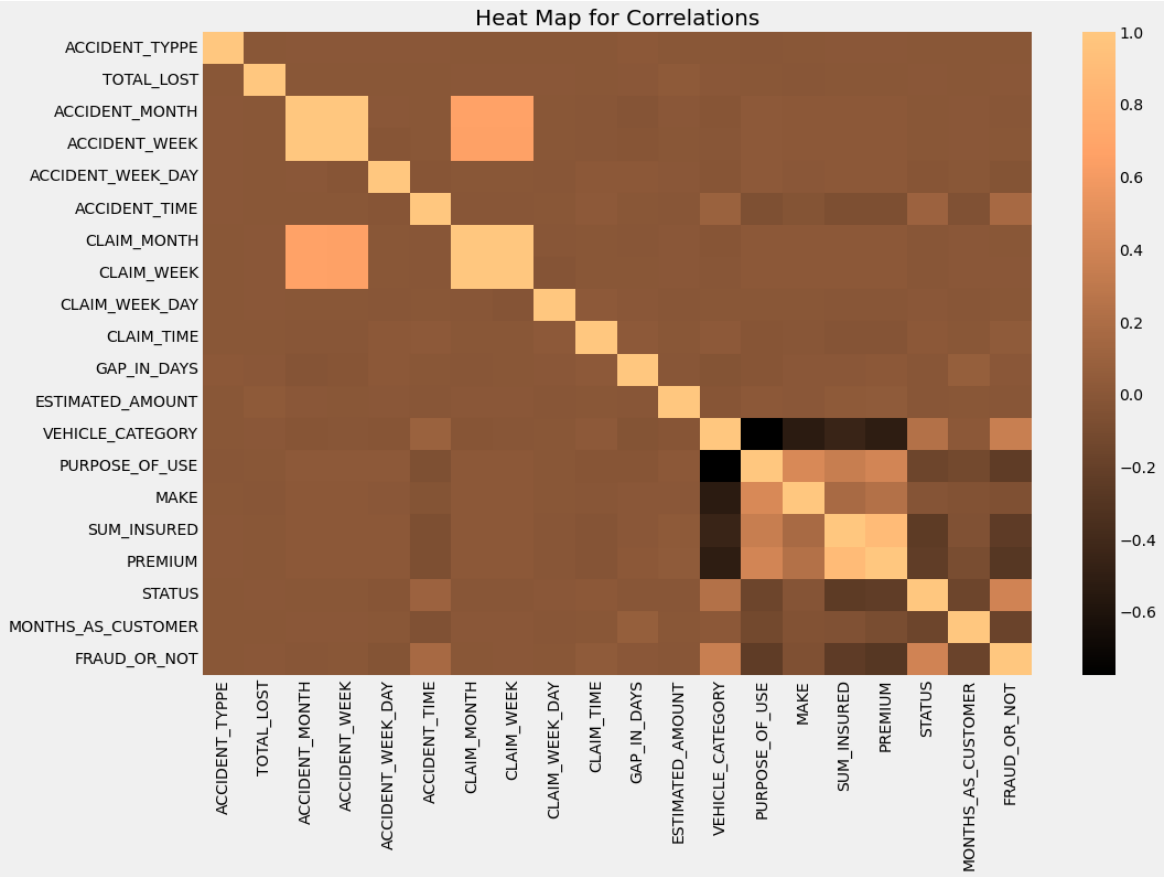


Figure 4.2-1 Heatmap for Correlations

By looking at the Heatmap, it can be identified some features which are more important to the class variable of Fraud. The features with dark colours can be identified as most important variables for the class variables.

4.2.2. Feature Selection using K Best Method

K best method is a famous method for selecting features for given dataset. It uses Chi square statistics to compare select features from weak features. Following figure shows how the feature selection using K best method was applied in python environment and finally selected features.

```
In [29]: best_features = SelectKBest(score_func = chi2, k =10)
         model = best_features.fit(x,y)
         scores = pd.DataFrame(model.scores_)

         columns  = pd.DataFrame(x.columns)
         featurescores = pd.concat ([columns, scores], axis=1)
         featurescores.columns = ["Features","Score"]
         print(featurescores.nlargest(10,'Score'))
```

	Features	Score
15	SUM_INSURED	4.304111e+09
11	ESTIMATED_AMOUNT	6.808839e+07
16	PREMIUM	6.724303e+07
18	MONTHS_AS_CUSTOMER	2.196569e+04
12	VEHICLE_CATEGORY	5.148025e+03
17	STATUS	2.360111e+03
5	ACCIDENT_TIME	1.321434e+03
13	PURPOSE_OF_USE	1.209039e+03
14	MAKE	4.595788e+02
10	GAP_IN_DAYS	1.030553e+02

Figure 4.2-2 K Best Method for feature selection

Ten features were selected by applying K Best method and heatmap criteria's. These features will be used to construct machine learning models explained in the methodology chapter. Following table shows the selected features.

Table 4.2-1 Selected Features from feature selection

Variable	Type
SUM_INSURED	FEATURE
ESTIMATED_AMOUNT	FEATURE
PREMIUM	FEATURE
MONTHS_AS_CUSTOMER	FEATURE
VEHICLE_CATEGORY	FEATURE
STATUS	FEATURE
ACCIDENT_TIME	FEATURE
PURPOSE_OF_USE	FEATURE
MAKE	FEATURE
GAP_IN_DAYS	FEATURE
FRAUD_OR_NOT	CLASS

4.3. Random Forest Models

Random Forest is the first classifier that will be used to build a machine learning model in this research. Three different models which are Default model, Oversampling with SMOTE and hyperparameter tuned model will be constructed and will be evaluated by using different evaluation metrics.

4.3.1. Default Model - No Oversampling or hyperparameter Tuning

Table 4.3-1 Evaluation metrics for Random Forest Default Model

Class	Precision	Recall	F1-Score	Support
0	0.95	1	0.98	8074
1	0.98	0.59	0.73	956
Average	0.97	0.79	0.85	9030

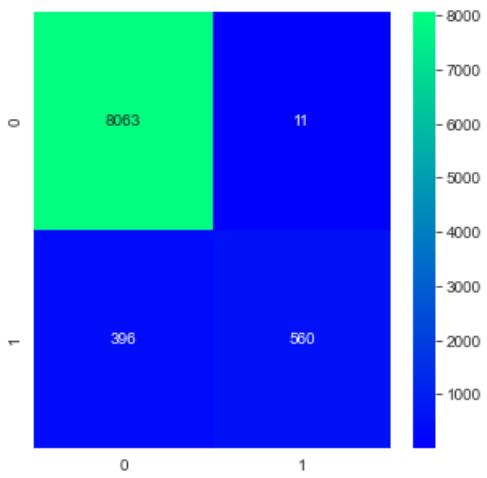


Figure 4.3-1 Confusion Matrix for Random Forest Default Model

First the model was built without any oversampling or hyperparameter tuning. The model perform well for normal claims, however, the performance is not satisfactory when dealing with fraudulent class, where the recall and f1 score were 0.59 and 0.73. This can be expected since the dataset is imbalanced.

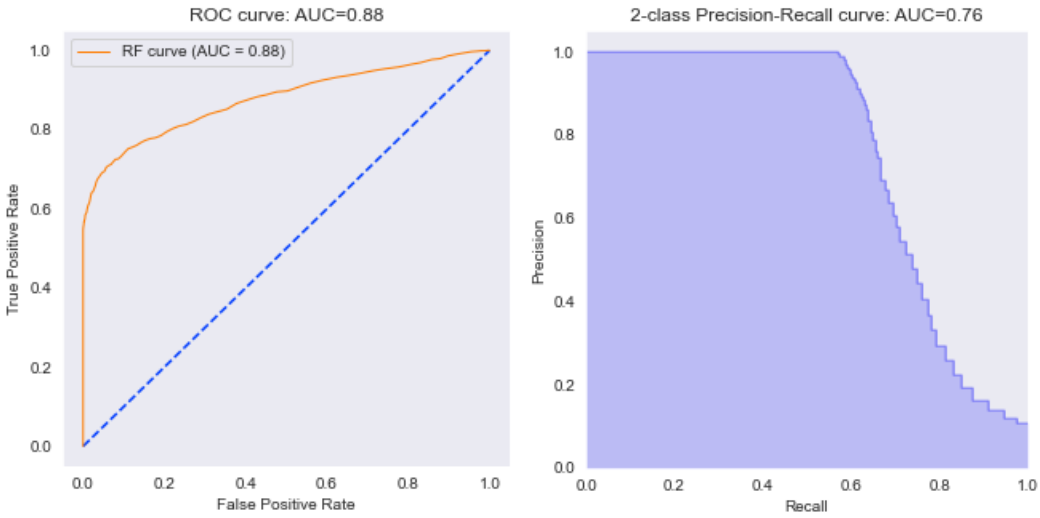


Figure 4.3-2 ROC Curve and PR Curve for Random Forest Default Model

ROC AUC (area under receiver operating characteristic curve) value is 0.88 and it gives an idea that model is good, however PR AUC (area under precision recall curve) and value 0.76, which is shown in figure is not very good. This can be expected since the dataset is imbalance.

4.3.2. Oversampling with SMOTE

The dataset is rearrange using the oversampling method SMOTE, which is used to solve the imbalance problem in the dataset. The figure shows the before and after training and test dataset distribution by applying SMOTE.

Figure 4.3-3 Training set - Target class before and after SMOTE

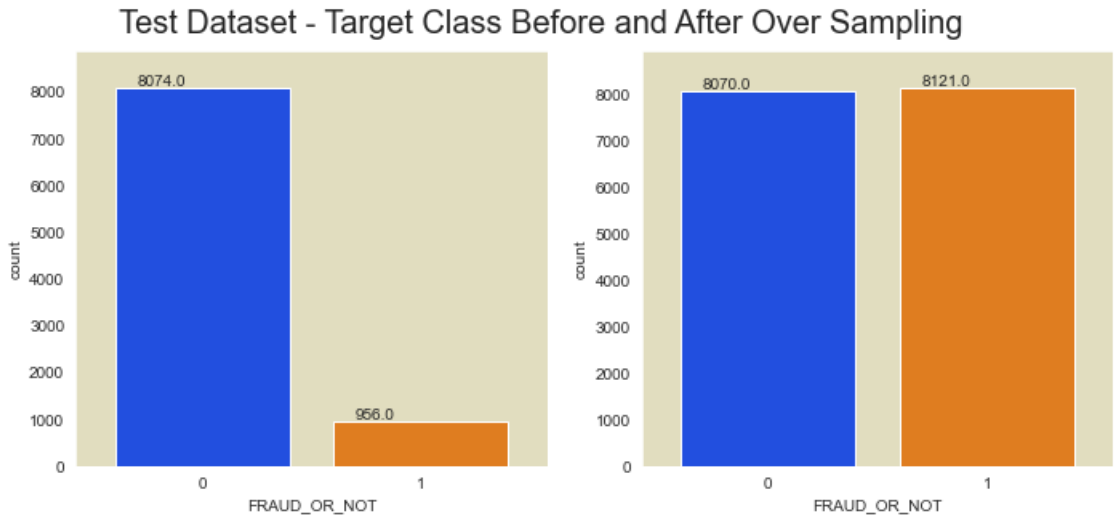


Figure 4.3-4 Testing set - Target class before and after SMOTE

Model Building after SMOTE Oversampling

After oversampling with SMOTE, random forest model was built and performance metrics are evaluated.

Table 4.3-2 Evaluation Metrics for Random Forest with SMOTE

Class	Precision	Recall	F1-Score	Support
0	0.93	0.98	0.95	8070
1	0.97	0.92	0.95	8121
Average	0.95	0.95	0.95	16191

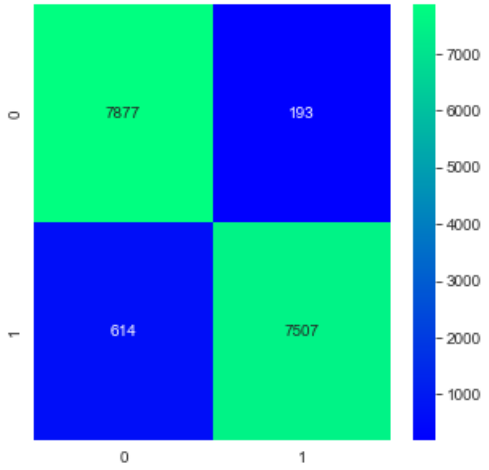


Figure 4.3-5 Confusion Matrix for Random Forest with SMOTE

Random Forest performed well in classifying the positive class, when oversampling with SMOTE was used, which is shown in table. In fraud claim class, Recall and f1 scores are increased as 0.97, 0.92 and 0.95, which can be considered as improvement from previous.

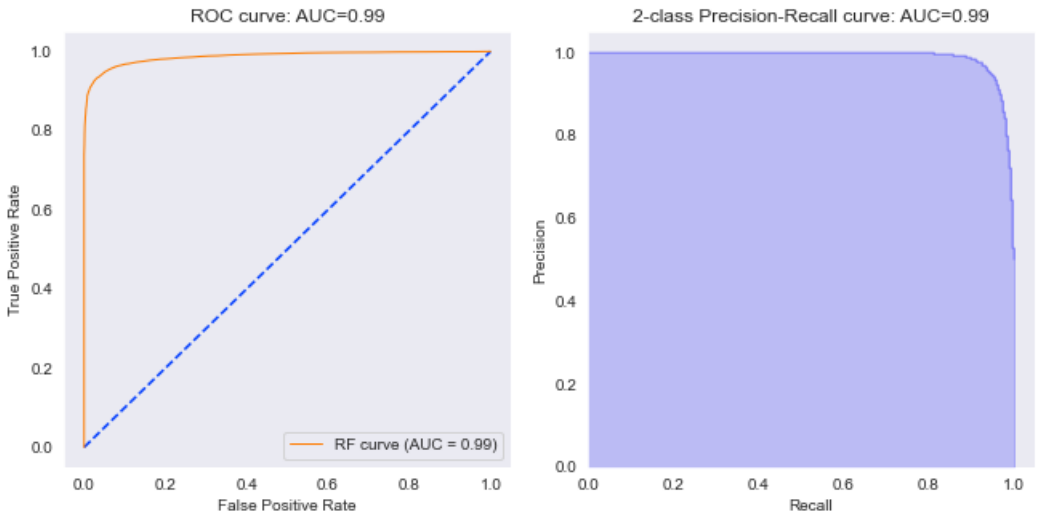


Figure shows a ROC curve in which ROC AUC is 0.99, which is very good and PR curve as shown in figure, gives AUC 0.99 which also suggest model is good for after considering the imbalance problem.

Cross Validation

```
from sklearn.model_selection import cross_val_score

rfc_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')

print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

The Random Forest model with SMOTE oversampling is further validate using cross validate technique. For cross validating the model 10-fold cross validation is used and mean AUC score was calculated and the execution code is shown above. Mean AUC is recorded as 0.86104 which shows the random forest model is good.

4.3.3. Hyperparameter Tuning

Three hyperparameters of the random forest model which are n_estimators, max_features and max_depth was tuned using random search method. Following results received for tuned hyper parameters.

Best values for the hyperparameters are:

- 'n_estimators': 1000
- 'max_features': 'auto'
- 'max_depth': 80

Random Forest Model for tuned hyperparameters

Table 4.3-3 Evaluation Metrics for Random Forest with Hyperparameter tuning

Class	Precision	Recall	F1-Score	Support
0	0.93	0.98	0.95	8070
1	0.98	0.92	0.95	8121
Average	0.95	0.95	0.95	16191

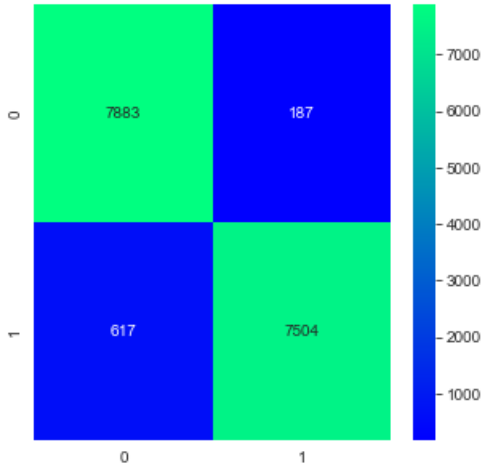


Figure 4.3-6 Confusion Matrix for Random Forest with Hyperparameter tuning

```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8666534674781788
```

By looking at the evaluation criteria's only precision is increased by 0.1 percent compared to the previous model. Mean AUC calculated after 10-fold cross validation and it is also slightly increased by 0.05comapred to previous model.

4.4. XGBoost models

XGBoost is the second classifier that will be used to build a machine learning model in this research. Same as Random Forest, three different models which are Default model, Oversampling with SMOTE and hyperparameter tuned model will be constructed and will be evaluated by using different evaluation metrics.

4.4.1. Default Model - No Oversampling or hyperparameter Tuning

Table 4.4-1 Evaluation Metrics for XGBoost with Default model

Class	Precision	Recall	F1-Score	Support
0	0.96	1	0.98	8074
1	0.95	0.64	0.76	956
Average	0.96	0.82	0.87	9030

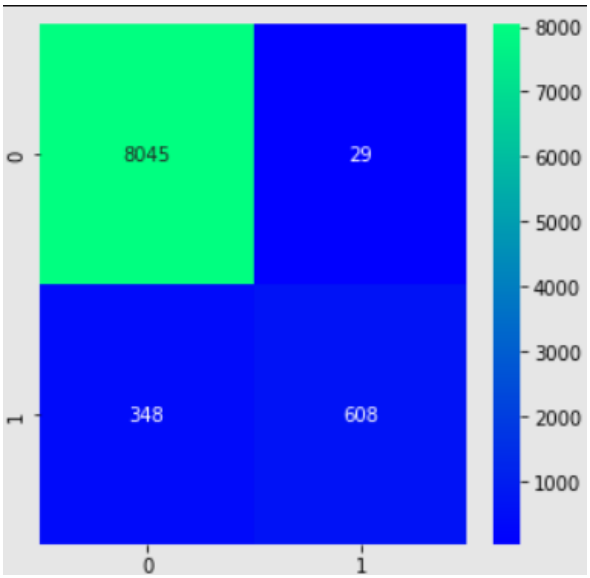


Figure 4.4-1 Confusion Matrix for XGBoost with Default Model

XGBoost model was built without any oversampling or hyperparameter tuning. Similar to Random Forest model, XGBoost model performed well in classifying the normal claims, without oversampling as can see with precision, recall and f1 scores of 0.96, 1 and 0.98 respectively as shown in table. This can be expected since the data includes an imbalanced class. However similar to random forest, the performance is not satisfactory when dealing with fraudulent class, where the recall and f1 score were 0.64 and 0.76.

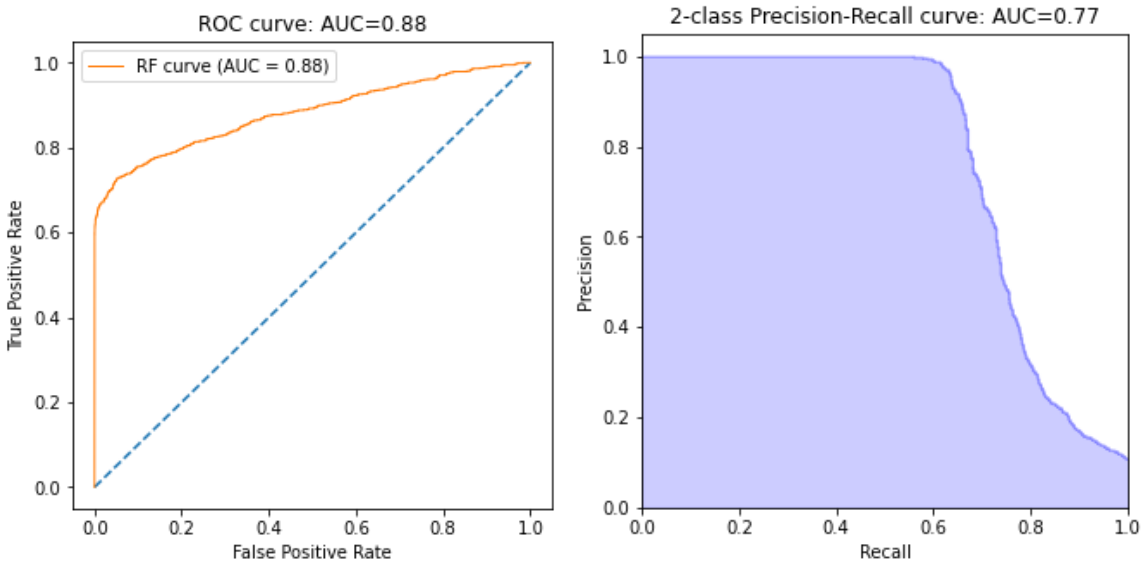


Figure 4.4-2 ROC and PR Curve for XGBoost with Default Model

ROC AUC value is 0.88 and it gives an idea that model is good, however PR AUC value is 0.77, which is shown in figure is not very good. This can be expected since the dataset is imbalance.

4.4.2. Model Building with SMOTE Oversampling

SMOTE oversampling method was applied for the dataset and XGBoost model was built and performance metrics are evaluated. The same combination of fraud claims and normal claims which applied in Random Forest will be used in here also.

Table 4.4-2 Evaluation Metrics for XGBoost with SMOTE

Class	Precision	Recall	F1-Score	Support
0	0.90	0.97	0.93	8070
1	0.96	0.89	0.93	8121
Average	0.93	0.93	0.93	16191

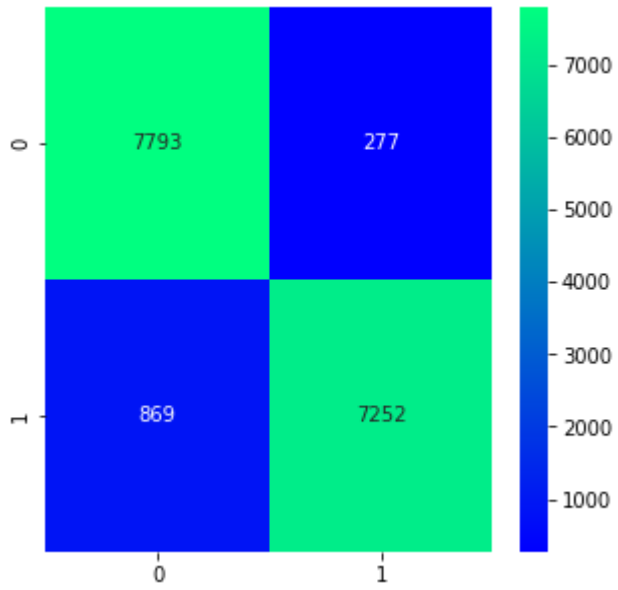


Figure 4.4-3 Confusion Matrix for XGBoost with SMOTE

Oversampling improves the XGBoost model and performed well in classifying the positive class, which is shown in table with Recall and F1 score values are improved. In the case of the positive class, the precision. Recall and f1 scores are increased as 0.96, 0.89 and 0.93.

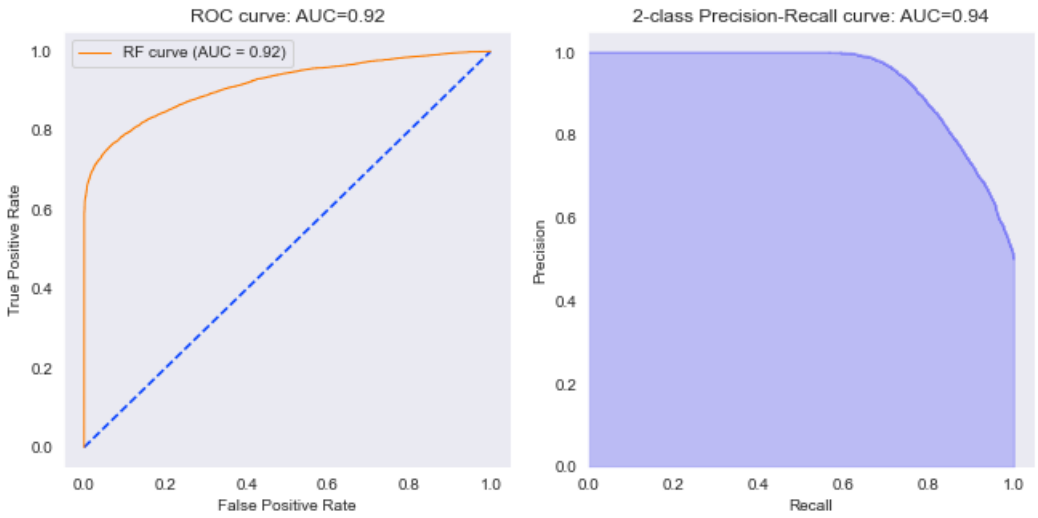


Figure 4.4-4 ROC and PR Curve for XGBoost with SMOTE

Figure shows a ROC curve in which ROC AUC is 0.92, which is good and PR curve as shown in figure, gives AUC 0.94 which also suggest model is good for after considering the imbalance problem.

Cross Validation

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

xgb_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')

print("Mean AUC Score - XGBoost: ", xgb_cv_score.mean())
```

```
Mean AUC Score - XGBoost:  0.8691560119342394
```

The XGBoost model with SMOTE oversampling is further validate using cross validate technique. For cross validating the model 10-fold cross validation is used and mean AUC score was calculated

and the execution code is shown above. Mean AUC is recorded as 0.869156 which shows the XGBoost model is improved.

4.4.3. Model with Hyperparameter Tuning

Following are the hyperparameter values of the XGBoost model which is tuned using random search method.

Best values for the hyperparameters are:

- `'min_child_weight': 1,`
- `'max_depth': 15,`
- `'learning_rate': 0.15,`
- `'gamma': 0.4,`
- `'colsample_bytree': 0.3`

XGBoost Model with tuned hyperparameters

Table 4.4-3 Evaluation Metrics for SGBoost with Hyperpaarameter Tuning

Class	Precision	Recall	F1-Score	Support
0	0.91	0.98	0.94	8070
1	0.98	0.91	0.94	8121
Average	0.94	0.94	0.94	16191

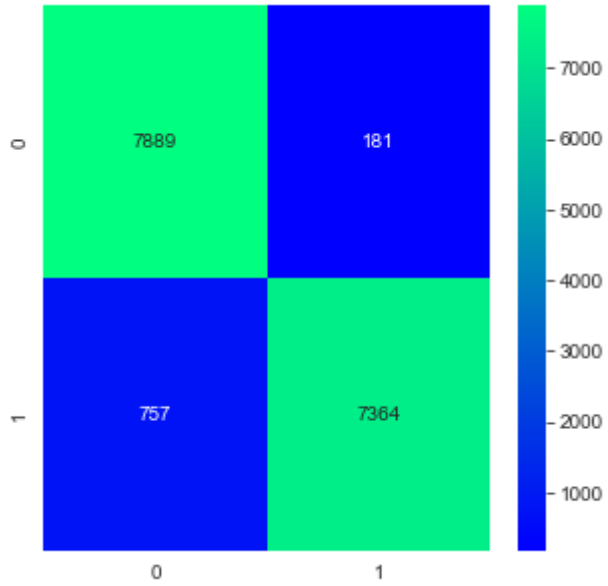


Figure 4.4-5 Confusion Matrix for SGBoost with Hyperpaaarameter Tuning

```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8684413010847196
```

Compared to the model without hyperparameter tuning, recall and F1 score are increased by 0.2 and 0.1 percent respectively. However, Mean AUC calculated after 10-fold cross validation and it is not changed significantly compared to previous model.

4.4.4. Features Importance to the model by XGBoost

Apart from building an efficiency model, XGBoost algorithm also can be used to select features that are more important to the model. XGBoost can be used to select features by counting the number of times each feature is split on boosting trees in the model, and then visualizing the result as a bar graph. I will visualize how many times they appear by ordering them in descending order. Following figure shows the feature importance of final XGBoost model.

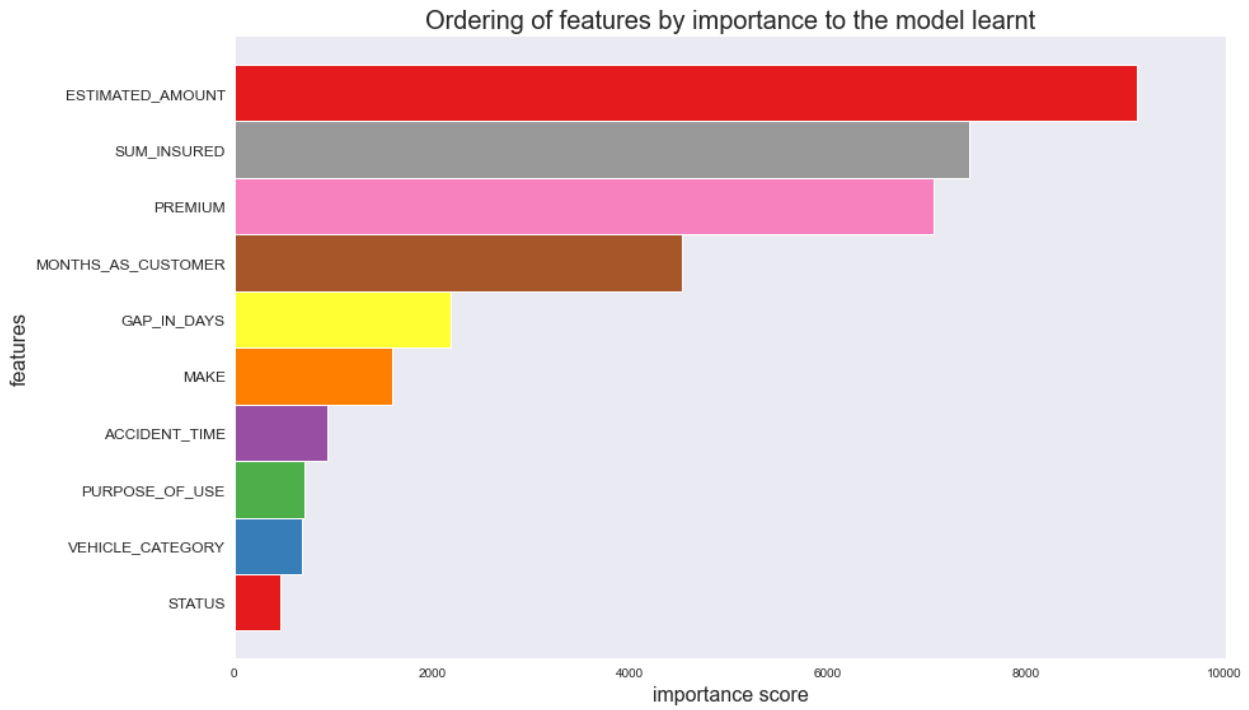


Figure 4.4-6 Important Features for the XGBoost model

4.5. Neural Network Models

Artificial Neural Network is the last classifier that will be used to build a machine learning model in this research. Two different models which are Default model and Oversampling with SMOTE will be constructed and will be evaluated by using different evaluation metrics.

4.5.1. Default Model - No Oversampling

A multilayer perceptron Neural Network model is used since this is a binary classification model and multilayer perceptron work well for binary classification. 'relu' Activation function is used for hidden layer since it is most common one for classification problems. Dense layers, which is fully connected layer is used to connect layers.

Table 4.5-1 Evaluation Metrics for Neural Network with Default model

Class	Precision	Recall	F1-Score	Support
0	0.95	0.99	0.97	8074
1	0.9	0.54	0.68	956
Average	0.92	0.77	0.82	9030

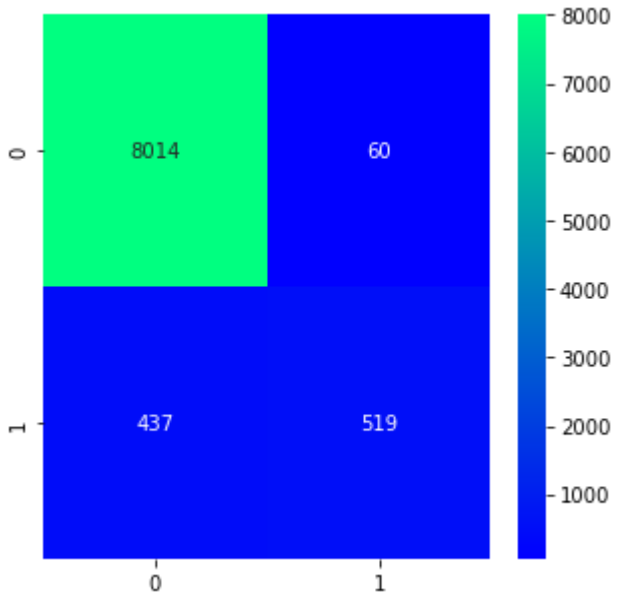


Figure 4.5-1 Confusion Matrix for Neural Network with Default Model

Neural Network mdoel was built without any oversampling with 3 hidden layers. Similar to Random Forest model and XGBoost model, Neural Network is performed well in classifying the normal claims, without oversampling as can see with precision, recall and f1 scores of 0.95, 0.99 and 0.97 respectively as shown in table. However, the performance is not satisfactory when dealing with fraudulent class, where the recall and f1 score were 0.54 and 0.68.

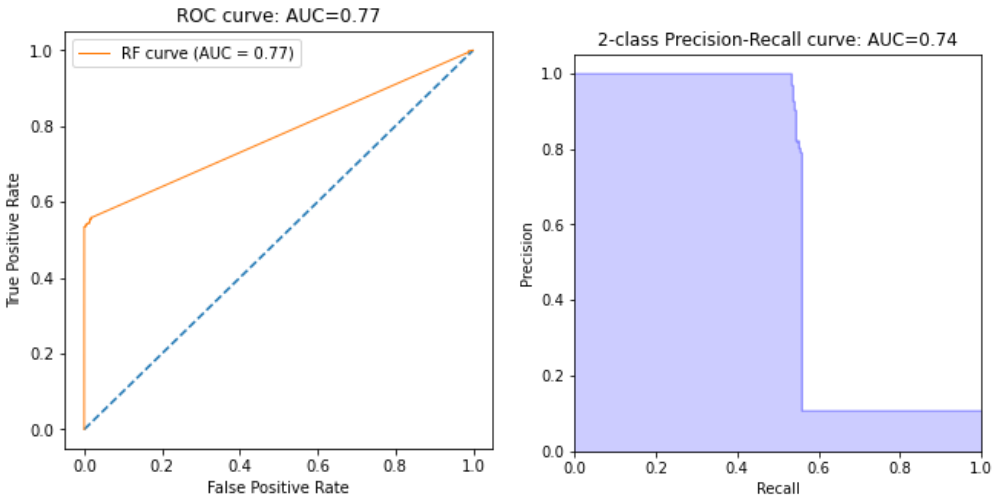


Figure-4.5-2 ROC and PR Curve for Neural Network with Default Model

ROC AUC value is 0.77 and PR AUC value is 0.74, which is shown in figure is not very good.

4.5.2. Model Building with SMOTE Oversampling

SMOTE oversampling method was applied for the dataset and Neural Network model was built and performance metrics are evaluated. The same combination of fraud claims and normal claims which applied in Random Forest and XGBoost will be used in here also.

Table 4.5-2 Evaluation Metrics for Neural Network with SMOTE

Class	Precision	Recall	F1-Score	Support
0	0.68	1	0.81	8070
1	1	0.52	0.69	8121
Average	0.84	0.76	0.75	16191

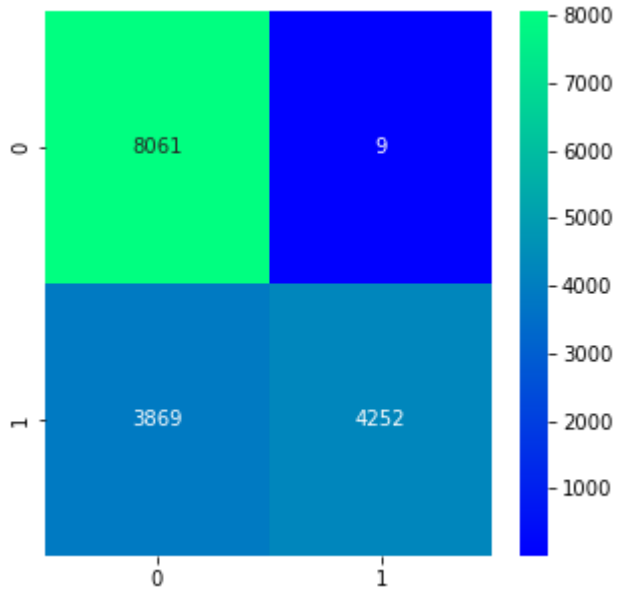


Figure 4.5-3 Confusion Matrix for Neural Network with SMOTE

Oversampling not improves the neural Network model and is not performed well in classifying the positive class, which is shown in table and confusion matrix.

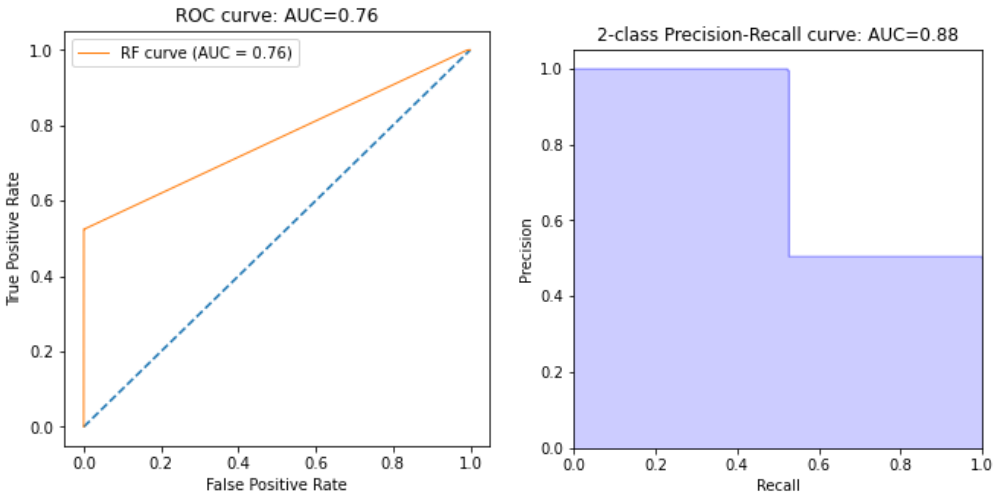


Figure 4.5-4 ROC and PR Curve for Neural Network with SMOTE

Figure shows a ROC curve in which ROC AUC is 0.76, and PR curve as shown in figure, gives AUC 0.88. ROC Performance is not in the model compared to previous model.

4.6. Result Summary

Following table summarizes fraud claim class prediction (Positive class) evaluation for each model build in this research.

Figure 4.6-1 Overall Summary of models

Classifier	Method	Precision	Recall	F1 Score	ROC	PR AUC	Model Time
Random Forest	Default	0.98	0.59	0.73	0.88	0.76	3.8 Sec
	SMOTE	0.97	0.92	0.95	0.99	0.99	5.59 Sec
	Hyperparameter Tuning	0.98	0.92	0.95	0.99	0.99	13.32 Sec
XGBoost	Default	0.95	0.64	0.76	0.88	0.77	1.58 Sec
	SMOTE	0.96	0.89	0.93	0.92	0.94	2.99 Sec
	Hyperparameter Tuning	0.98	0.91	0.94	0.92	0.94	40.5 Sec
Artificial Neural Network	Default	0.9	0.54	0.68	0.77	0.74	3.87
	SMOTE	1	0.52	0.69	0.76	0.88	6.94

Table shows that Random forest model and XGBoost model perform well compared to Neural Network model when considering all evaluation criterias. This Shows Ensemble models are well suited in Motor claim fraud detection area since their ability to covert the weak learners to strong learners.

Also When comparing the random forest model and there is not much different in each model, but random forest model with tuned parameters are slightly ahead with rest of the models. Therefore, it can be concluded that Random forest model with tuned hyperparameters and oversampled by SMOTE is best model that can be used to implement a Fraud detection system.

5. Conclusion and Recommendations

The thesis is constructed to predict whether the motor claim is fraud or not by using a machine learning model. The data used for this research is from Sri Lanka Insurance motor claim unit and it contained 26985 of normal claims and 3112 of fraud claims. Data cleansing and feature selection methods such as K best method and Heatmap are used to develop the final dataset. The final dataset presents 10 feature variables and 1 class variable. Three types of classifiers which are Random Forest, XGBoost and Neural Network classifiers are used to build models from the dataset. First the default models are constructed and evaluated. However, since the dataset is highly imbalanced, Oversampling method called Synthetic Minority Oversampling Technique (SMOTE) is used to remove the unbalanceness of the dataset. Random forest and XGBoost contains hyperpaters nd therefore hyperparameters are also tuned using cross validation technique called random search. Altogether, eight models are constructed and evaluated using precision, recall, F1score, ROC, PR Score and Model run time.

It was found that Random forest and XGBoost models are perform better compared to neural network model. There was not much difference between random forest models and XGBoost models, however, Random forest model with tuned hyperparameters perform slightly better than other models.

As a conclusion it can clearly see that ensemble models like random forest model and XGboost model perfrom better in predicting motor fraud claims, which shows the importance of converting weak learners to strong learners by ensembling techniques.

6. REFERENCES

Agyemang, M., Barker, K., Alhajj, R., 2006. A comprehensive survey of numeric and symbolic outlier mining techniques. Intell Data Anal 10, 521–538. https://doi.org/10.3233/IDA-2006-10604

AUTO-INSURANCE FRAUD DETECTION: A BEHAVIORAL FEATURE ENGINEERING APPROACH, 2020. . J. Crit. Rev. 7. https://doi.org/10.31838/jcr.07.03.23

Background on: Insurance fraud | III [WWW Document], n.d. URL https://www.iii.org/article/background-on-insurance-fraud (accessed 9.13.21).

Batra, B., Kundra, S., 2019. Naïve Classification Approach for Insurance Fraud Prediction 8, 5.

Detecting Fraudulent Claims - A machine learning approch.pdf, n.d.

Dhieb, N., Ghazzai, H., Besbes, H., Massoud, Y., 2019. Extreme Gradient Boosting Machine Learning Algorithm For Safe Auto Insurance Operations, in: 2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES). Presented at the 2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES), IEEE, Cairo, Egypt, pp. 1–5. https://doi.org/10.1109/ICVES.2019.8906396

Fraud - Oxford Reference [WWW Document], n.d. URL https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095833457 (accessed 9.13.21).

Fraud Detection in Health Insurance Claims using Machine Learning Algorithms, 2020. . Int. J. Recent Technol. Eng. 8, 2999–3004. https://doi.org/10.35940/ijrte.E6485.018520

Ghorbani, A., Farzai, S., 2018. Fraud Detection in Automobile Insurance using a Data Mining Based Approach 8, 8.

Gong, J., Zhang, H., Du, W., 2020. Research on Integrated Learning Fraud Detection Method Based on Combination Classifier Fusion (THBagging): A Case Study on the Foundational Medical Insurance Dataset. Electronics 9, 894. https://doi.org/10.3390/electronics9060894

IBSL-AR-English-2018-Fullset.pdf, n.d.

Insurance fraud detection and cost to industry [WWW Document], n.d. URL https://www.atlas-mag.net/en/article/insurance-fraud-detection-and-cost-to-industry (accessed 9.13.21).

Insurance Fraud [WWW Document], n.d. . Fed. Bur. Investig. URL https://www.fbi.gov/stats-services/publications/insurance-fraud (accessed 9.13.21).

Makki, S., n.d. An Efficient Classification Model for Analyzing Skewed Data to Detect Frauds in the Financial Sector 175.

Morley, N.J., Ball, L.J., Ormerod, T.C., n.d. How the detection of insurance fraud succeeds and fails 19.

Ngai, E.W.T., Hu, Y., Wong, Y.H., Chen, Y., Sun, X., 2011. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. Decis. Support Syst. 50, 559–569. https://doi.org/10.1016/j.dss.2010.08.006

Phua, C., Lee, V., Smith, K., Gayler, R., n.d. A Comprehensive Survey of Data Mining-based Fraud Detection Research 14.

Randhawa, K., Loo, C.K., Seera, M., Lim, C.P., Nandi, A.K., 2018. Credit Card Fraud Detection Using AdaBoost and Majority Voting. IEEE Access 6, 14277–14284. https://doi.org/10.1109/ACCESS.2018.2806420

S. Patil, K., 2018. A Survey on Machine Learning Techniques for Insurance Fraud Prediction. HELIX 8, 4358–4363. https://doi.org/10.29042/2018-4358-4363

Sowah, R.A., Kuuboore, M., Ofoli, A., Kwofie, S., Asiedu, L., Koumadi, K.M., Apeadu, K.O., 2019. Decision Support System (DSS) for Fraud Detection in Health Insurance Claims Using Genetic Support Vector Machines (GSVMs). J. Eng. 2019, 1–19. https://doi.org/10.1155/2019/1432597

Subudhi, S., Panigrahi, S., 2018. Effect of Class Imbalanceness in Detecting Automobile Insurance Fraud, in: 2018 2nd International Conference on Data Science and Business Analytics (ICDSBA). Presented at the 2018 2nd International Conference on Data Science and Business Analytics (ICDSBA), IEEE, Changsha, pp. 528–531. https://doi.org/10.1109/ICDSBA.2018.00104

7. Appendix

7.1. Data Preparation Codes and Results from Jupyter Notebook for Python

```
In [4]: import pandas as pd
```

```
In [12]: data = pd.read_csv("Final Data set v3 New.csv")
```

```
In [13]: data.head()
```

Out[13]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_MONTH	ACCIDENT_WEEK	ACCIDENT_WEEK_DAY	ACCIDENT_TIME	CLAIM_MONTH	CLAIM_WEEK	CLAIM_WE
0	ACCIDENT	NORMAL	5	18	2	Afternoon	8	34	
1	ACCIDENT	NORMAL	3	11	3	Early Morning	4	17	
2	ACCIDENT	NORMAL	10	42	4	Night	11	45	
3	ACCIDENT	NORMAL	9	36	3	Morning	9	38	
4	ACCIDENT	NORMAL	11	46	4	Late Night	1	4	

```
In [14]: data.shape
```

```
Out[14]: (30098, 20)
```

```
In [16]: data.isnull().any()
```

```
Out[16]: ACCIDENT_TYPPE        False
         TOTAL_LOST             True
         ACCIDENT_MONTH        False
         ACCIDENT_WEEK         False
         ACCIDENT_WEEK_DAY     False
         ACCIDENT_TIME          True
         CLAIM_MONTH           False
         CLAIM_WEEK            False
         CLAIM_WEEK_DAY        False
         CLAIM_TIME            False
         GAP_IN_DAYS           False
         ESTIMATED_AMOUNT      False
         VEHICLE_CATEGORY       True
         PURPOSE_OF_USE         True
         MAKE                  False
         SUM_INSURED            True
         PREMIUM                True
         STATUS                 True
         MONTHS_AS_CUSTOMER     True
         FRAUD_OR_NOT          False
         dtype: bool
```

```
In [23]: data_new = data.copy()
```

```
In [24]: data_new.head()
```

Out[24]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_MONTH	ACCIDENT_WEEK	ACCIDENT_WEEK_DAY	ACCIDENT_TIME	CLAIM_MONTH	CLAIM_WEEK	CLAIM_WE
0	ACCIDENT	NORMAL	5	18	2	Afternoon	8	34	
1	ACCIDENT	NORMAL	3	11	3	Early Morning	4	17	
2	ACCIDENT	NORMAL	10	42	4	Night	11	45	

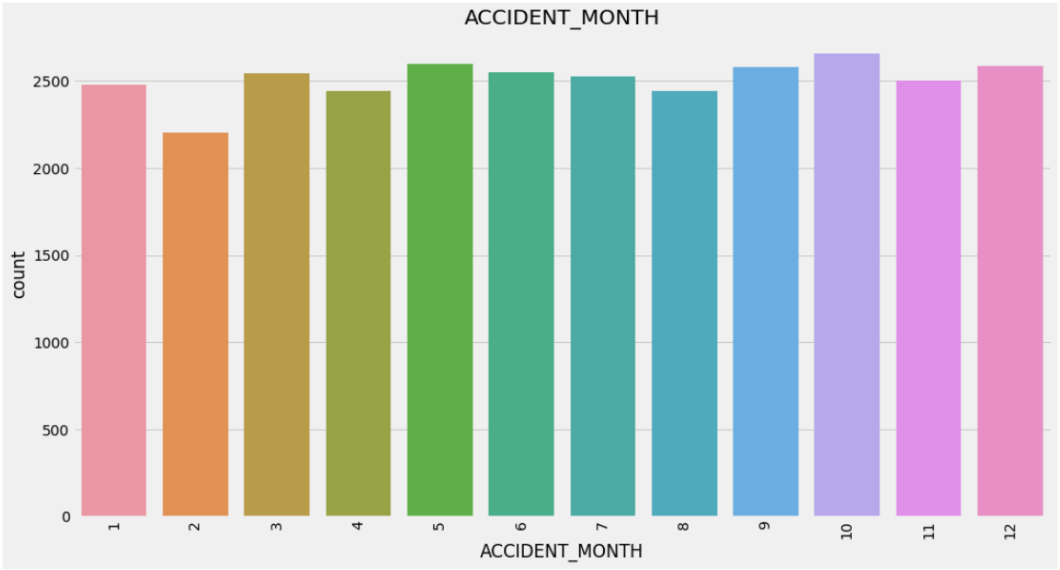
In [19]:
```
# missing value treatment using fillna


data['TOTAL_LOST'].fillna(data['TOTAL_LOST'].mode()[0], inplace = True)
data['ACCIDENT_TIME'].fillna('NO', inplace = True)
data['CLAIM_TIME'].fillna('NO', inplace = True)
data['VEHICLE_CATEGORY'].fillna('NO', inplace = True)
data['PURPOSE_OF_USE'].fillna('NO CATEGORY', inplace = True)
data['MAKE'].fillna('NO CATEGORY', inplace = True)
data['SUM_INSURED'].fillna(0, inplace = True)
data['PREMIUM'].fillna(0, inplace = True)
data['STATUS'].fillna('NO CATEGORY', inplace = True)
data['MONTHS_AS_CUSTOMER'].fillna(0, inplace = True)
```

In [20]:
```
data.isnull().any()
```

Out[20]:
```
ACCIDENT_TYPPE        False
TOTAL_LOST            False
ACCIDENT_MONTH        False
ACCIDENT_WEEK         False
ACCIDENT_WEEK_DAY     False
ACCIDENT_TIME         False
CLAIM_MONTH           False
CLAIM_WEEK            False
CLAIM_WEEK_DAY        False
CLAIM_TIME            False
GAP_IN_DAYS           False
ESTIMATED_AMOUNT      False
VEHICLE_CATEGORY      False
PURPOSE_OF_USE        False
MAKE                  False
SUM_INSURED           False
PREMIUM               False
STATUS                False
MONTHS_AS_CUSTOMER    False
FRAUD_OR_NOT          False
dtype: bool
```

In [28]:
```
cat_data.head()
```

```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (15, 8)

sns.countplot(data['ACCIDENT_MONTH'])
plt.title('ACCIDENT_MONTH', fontsize = 20)
plt.xticks(rotation = 90)
plt.show()
```



In [21]:
```
data_new2 = data.copy()
```

In [22]:
```
cat_data = data.select_dtypes(include=['object']).copy()
```

In [23]:
```
cat_data.head()
```

Out[23]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_TIME	CLAIM_TIME	VEHICLE_CATEGORY	PURPOSE_OF_USE	MAKE	STATUS
0	ACCIDENT	NORMAL	Afternoon	Morning	Dual Purpose	Private	TOYOTA	Mr
1	ACCIDENT	NORMAL	Early Morning	Evening	Car	Private Person	MITSUBISHI	Mrs
2	ACCIDENT	NORMAL	Night	Morning	Car	Private Person	MITSUBISHI	Mrs
3	ACCIDENT	NORMAL	Morning	Afternoon	Car	Private Person	TOYOTA	Prof
4	ACCIDENT	NORMAL	Late Night	Afternoon	Car	Private Person	TOYOTA	Prof

In [25]:
```
data.ACCIDENT_TYPPE = pd.Categorical(data.ACCIDENT_TYPPE).codes
data.TOTAL_LOST = pd.Categorical(data.TOTAL_LOST).codes
data.ACCIDENT_TIME = pd.Categorical(data.ACCIDENT_TIME).codes
data.CLAIM_TIME = pd.Categorical(data.CLAIM_TIME).codes
data.VEHICLE_CATEGORY = pd.Categorical(data.VEHICLE_CATEGORY).codes
data.PURPOSE_OF_USE = pd.Categorical(data.PURPOSE_OF_USE).codes
data.MAKE = pd.Categorical(data.MAKE).codes
data.STATUS = pd.Categorical(data.STATUS).codes
```

In [26]:
```
data.head()
```

Out[26]:

	ACCIDENT_TYPPE	TOTAL_LOST	ACCIDENT_MONTH	ACCIDENT_WEEK	ACCIDENT_WEEK_DAY	ACCIDENT_TIME	CLAIM_MONTH	CLAIM_WEEK	CLAIM_WE
0	0	0	5	18	2	0	8	34	
1	0	0	3	11	3	1	4	17	
2	0	0	10	42	4	6	11	45	
3	0	0	9	36	3	4	9	38	
4	0	0	11	46	4	3	1	4	

In [24]:
```
data.shape
```

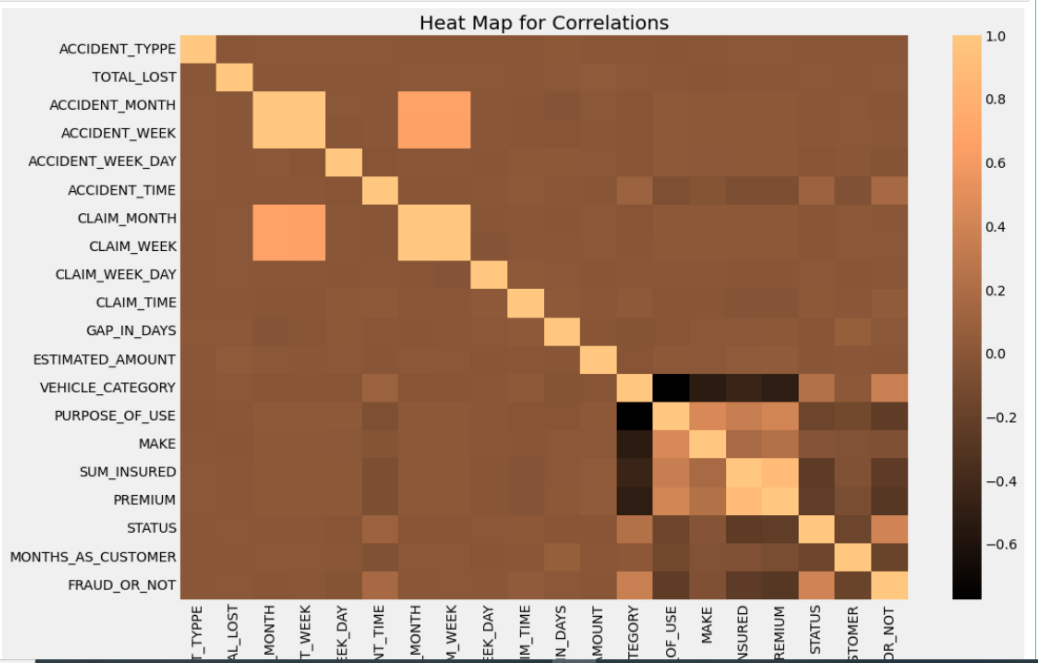
Out[24]:
```
(30098, 20)
```

In [29]:
```
best_features = SelectKBest(score_func = chi2, k =10)
model = best_features.fit(x,y)
scores = pd.DataFrame(model.scores_)

columns  = pd.DataFrame(x.columns)
featurescores = pd.concat ([columns, scores], axis=1)
featurescores.columns = ["Features","Score"]
print(featurescores.nlargest(10,'Score'))
```

```
             Features         Score
15         SUM_INSURED  4.304111e+09
11    ESTIMATED_AMOUNT  6.808839e+07
16             PREMIUM  6.724303e+07
18  MONTHS_AS_CUSTOMER  2.196569e+04
12    VEHICLE_CATEGORY  5.148025e+03
17              STATUS  2.360111e+03
5        ACCIDENT_TIME  1.321434e+03
13      PURPOSE_OF_USE  1.209039e+03
14                MAKE  4.595788e+02
10         GAP_IN_DAYS  1.030553e+02
```

In [36]:
```
plt.rcParams['figure.figsize'] = (15, 10)
sns.heatmap(data.corr(), cmap = 'copper')
plt.title('Heat Map for Correlations', fontsize = 20)
plt.show()
```

In [36]:
```
plt.rcParams['figure.figsize'] = (15, 10)
sns.heatmap(data.corr(), cmap = 'copper')
plt.title('Heat Map for Correlations', fontsize = 20)
plt.show()
```



7.2. Random Forest Model codes and Results

In [39]:
```
import warnings
warnings.filterwarnings('ignore')

# for some basic operations
import numpy as np
import pandas as pd
import joypy

# for visualizations
import matplotlib.pyplot as plt

import seaborn as sns
from pandas import plotting
from pandas.plotting import parallel_coordinates

# for interactive visualizations
import plotly
import plotly.offline as py
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from plotly import tools
init_notebook_mode(connected = True)
import plotly.figure_factory as ff

# for animated visualizations
from bubbly.bubbly import bubbleplot
import plotly_express as px


data = pd.read_csv("Final Data set v5.csv")
```

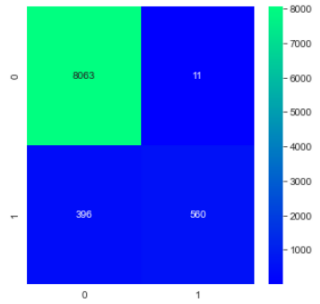
In [55]:
```
from datetime import datetime

def timer(start_time= None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now()-start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec,2)))
```

In [56]:
```
x =  data.iloc[:,0:10]
y = data.iloc[:, -1]
#y = df.iloc[:, -1]

from sklearn.model_selection import train_test_split

# iam diving the data set into trainset and test set
from sklearn.ensemble import RandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from numpy import *

np.random.seed(123)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

start_time = timer(None)
model = RandomForestClassifier()

model.fit(x_train, y_train)
y_pred_rf = model.predict(x_test)
y_score_rf = model.predict_proba(x_test)[:,-1]

print("Training Accuracy: ", model.score(x_train, y_train))
print('Testing Accuarcy: ', model.score(x_test, y_test))
```

```
# making a confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot = True, cmap = 'winter', fmt=".0f")
plt.show()
```

```
Training Accuracy:  0.9998576039491172
Testing Accuarcy:  0.9549280177187154
              precision    recall  f1-score   support

           0       0.95      1.00      0.98      8074
           1       0.98      0.59      0.73       956

    accuracy                           0.95      9030
   macro avg       0.97      0.79      0.85      9030
weighted avg       0.96      0.95      0.95      9030
```

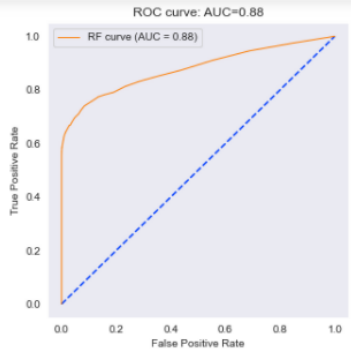
```
Time taken: 0 hours 0 minutes and 3.8 seconds.
```



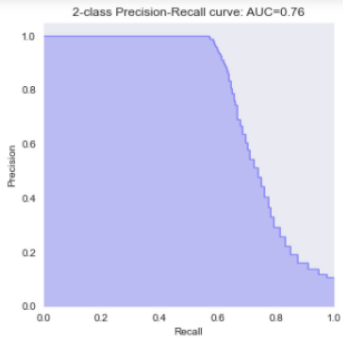
0 1

In [41]:
```
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve
```

In [42]:
```
from matplotlib import pyplot

ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model.predict_proba(x_test)
lr_probs = lr_probs[:, 1]



# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

roc_auc_rf = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_rf))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
          roc_auc_rf))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



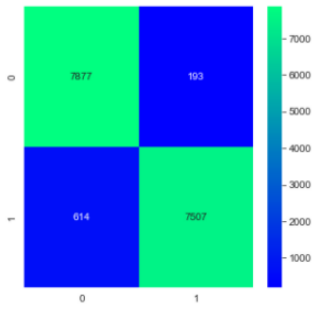
In [43]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt


average_precision = average_precision_score(y_test, y_score_rf)

print('Average precision-recall score RF: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test, y_score_rf)

pr_auc_rf = auc( recall,precision)

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_rf))
```



In [57]:
```
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE

np.random.seed(123)

x_resample, y_resample  = SMOTE().fit_resample(x, y)


x_train2, x_test2, y_train2, y_test2 = train_test_split(x_resample, y_resample, test_size = 0.3, random_state = 0)

start_time = timer(None)

model_o = RandomForestClassifier()
model_o.fit(x_train2, y_train2)

y_pred = model_o.predict(x_test2)
y_score_rf = model_o.predict_proba(x_test2)[:,-1]

print("Training Accuracy: ", model_o.score(x_train2, y_train2))
print('Testing Accuarcy: ', model_o.score(x_test2, y_test2))

# classification report
cr = classification_report(y_test2, y_pred)
print(cr)

timer(start_time)

# confusion matrix
cm = confusion_matrix(y_test2, y_pred)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'winter',fmt=".0f")
plt.show()
```

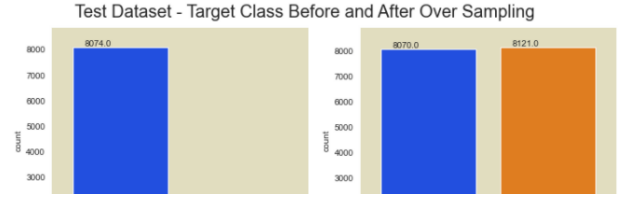
```
Training Accuracy:  0.9999470605362768
Testing Accuracy:  0.9501574949045766
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      8070
           1       0.97      0.92      0.95      8121

    accuracy                           0.95     16191
   macro avg       0.95      0.95      0.95     16191
weighted avg       0.95      0.95      0.95     16191
```

```
Time taken: 0 hours 0 minutes and 5.59 seconds.
```

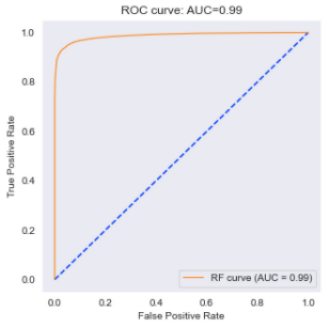


In [45]:
```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(10,4.5))
fig.subplots_adjust(bottom=0.10, left=0.10, top = 0.900, right=1.00)
fig.suptitle(' Training Dataset - Target Class Before and After Over Sampling', fontsize = 20)
sns.set_palette("bright")
sns.countplot(y_train, ax=ax1)
ax1.margins(0.1)
ax1.set_facecolor("#e1ddbf")
for p in ax1.patches:
        ax1.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_palette("bright")
sns.countplot(y_train2, ax=ax2)
ax2.margins(0.1)
ax2.set_facecolor("#e1ddbf")
for p in ax2.patches:
        ax2.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_style('dark')
```

In [46]:
```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(10,4.5))
fig.subplots_adjust(bottom=0.10, left=0.10, top = 0.900, right=1.00)
fig.suptitle(' Test Dataset - Target Class Before and After Over Sampling', fontsize = 20)
sns.set_palette("bright")
sns.countplot(y_test, ax=ax1)
ax1.margins(0.1)
ax1.set_facecolor("#e1ddbf")
for p in ax1.patches:
        ax1.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_palette("bright")
sns.countplot(y_test2, ax=ax2)
ax2.margins(0.1)
ax2.set_facecolor("#e1ddbf")
for p in ax2.patches:
        ax2.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_style('dark')
```



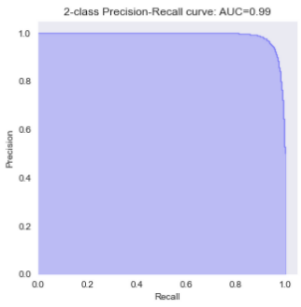
In [47]:
```
ns_probs = [0 for _ in range(len(y_test2))]
lr_probs = model_o.predict_proba(x_test2)
lr_probs = lr_probs[:, 1]
lr_probs


# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test2, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test2, lr_probs)

roc_auc_rf = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_rf))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
          roc_auc_rf))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



In [48]:
```

average_precision = average_precision_score(y_test2, y_score_rf)

print('Average precision-recall score RF: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test2,  lr_probs)

pr_auc_rf = auc(recall, precision )

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.05])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_rf))
```

```
Average precision-recall score RF: 0.9893823083150644
```

Out[48]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AUC=0.99')
```



In [49]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

rfc_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')


print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

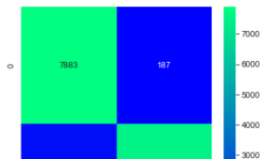
```
Mean AUC Score - Random Forest:  0.8610441507874078
```

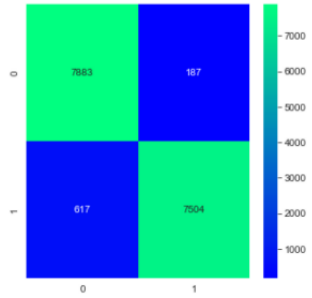
In [26]:
```
from sklearn.model_selection import RandomizedSearchCV
# number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# number of features at every split
max_features = ['auto', 'qrt']

# max depth
max_depth = [int(x) for x in np.linspace(100, 500, num = 11)]
max_depth.append(None)
# create random grid
random_grid = {
 'n_estimators': [100,200,300,1000],
 'max_features': max_features,
 'max_depth':[80,90,100,110]
 }
# Random search of parameters
rfc_random = RandomizedSearchCV(estimator = model_o, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_s
# Fit the model
rfc_random.fit(x_train2, y_train2)
# print results
print(rfc_random.best_params_)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
{'n_estimators': 1000, 'max_features': 'auto', 'max_depth': 80}
```

In [58]:
```
start_time = timer(None)

model_o = RandomForestClassifier(n_estimators=1000, max_depth=80, max_features='auto')
model_o.fit(x_train2, y_train2)

y_pred = model_o.predict(x_test2)
y_score_rf = model_o.predict_proba(x_test)[:,-1]
print("Training Accuracy: ", model_o.score(x_train2, y_train2))
print('Testing Accuarcy: ', model_o.score(x_test2, y_test2))
rfc_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')


# classification report
cr = classification_report(y_test2, y_pred)
print(cr)

timer(start_time)

# confusion matrix
cm = confusion_matrix(y_test2, y_pred)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'winter',fmt=".0f")
plt.show()


print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

```
Training Accuracy:  0.9999470605362768
Testing Accuarcy:  0.9503427830276079
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      8070
           1       0.98      0.92      0.95      8121

    accuracy                           0.95     16191
   macro avg       0.95      0.95      0.95     16191
weighted avg       0.95      0.95      0.95     16191


 Time taken: 0 hours 7 minutes and 13.32 seconds.
```






```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8666534674781788
```

In [59]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint

print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

7.3. XGBoost Codes and Results

In [3]:
```
pip install xgboost
```

```
Collecting xgboostNote: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 21.1.1; however, version 21.2.4 is available.
You should consider upgrading via the 'C:\Users\Administrator\AppData\Local\Programs\Python\Python38\python.exe -m pip install
--upgrade pip' command.


  Downloading xgboost-1.4.2-py3-none-win_amd64.whl (97.8 MB)
Requirement already satisfied: numpy in c:\users\administrator\appdata\local\programs\python\python38\lib\site-packages (from x
gboost) (1.21.2)
Requirement already satisfied: scipy in c:\users\administrator\appdata\local\programs\python\python38\lib\site-packages (from x
gboost) (1.7.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.4.2
```

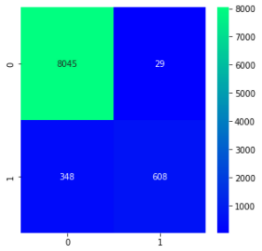
In [1]:
```
import warnings
warnings.filterwarnings('ignore')

# for some basic operations
import numpy as np
import pandas as pd
import joypy

# for visualizations
import matplotlib.pyplot as plt

import seaborn as sns
from pandas import plotting
from pandas.plotting import parallel_coordinates

# for interactive visualizations
import plotly
import plotly.offline as py
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from plotly import tools
init_notebook_mode(connected = True)
import plotly.figure_factory as ff

# for animated visualizations
from bubbly.bubbly import bubbleplot
import plotly_express as px


data = pd.read_csv("Final Data set v5.csv")
```

In [11]:
```
from datetime import datetime

def timer(start_time= None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now()-start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec,2)))
```

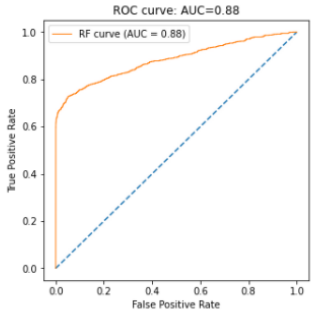
In [12]:
```
x =  data.iloc[:,0:10]
y = data.iloc[:, -1]
#y = df.iloc[:, -1]

from sklearn.model_selection import train_test_split

from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from numpy import *

np.random.seed(123)


# diving the data set into trainset and test set

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

start_time = timer(None)
model = XGBClassifier()

model.fit(x_train, y_train)
y_pred_xgb = model.predict(x_test)
y_score_xgb = model.predict_proba(x_test)[:,-1]

print("Training Accuracy: ", model.score(x_train, y_train))
print('Testing Accuarcy: ', model.score(x_test, y_test))

# making a classification report
cr = classification_report(y_test,  y_pred_xgb)
print(cr)

timer(start_time)
```

```
model.fit(x_train, y_train)
y_pred_xgb = model.predict(x_test)
y_score_xgb = model.predict_proba(x_test)[:,-1]

print("Training Accuracy: ", model.score(x_train, y_train))
print('Testing Accuarcy: ', model.score(x_test, y_test))

# making a classification report
cr = classification_report(y_test,  y_pred_xgb)
print(cr)

timer(start_time)

# making a confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
cm = confusion_matrix(y_test, y_pred_xgb)
sns.heatmap(cm, annot = True, cmap = 'winter', fmt=".0f")
plt.show()
```

```
[13:48:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Training Accuracy:  0.9737991266375546
Testing Accuarcy:  0.958250276854928
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      8074
           1       0.95      0.64      0.76       956

    accuracy                           0.96      9030
   macro avg       0.96      0.82      0.87      9030
weighted avg       0.96      0.96      0.95      9030


 Time taken: 0 hours 0 minutes and 1.58 seconds.
```



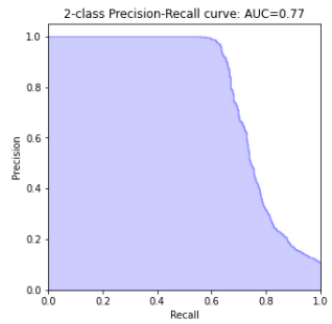
In [7]:
```
from matplotlib import pyplot

ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model.predict_proba(x_test)
lr_probs = lr_probs[:, 1]
lr_probs


# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

roc_auc_xgb = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_xgb))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
          roc_auc_rf))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



In [8]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

average_precision = average_precision_score(y_test, y_score_xgb)

print('Average precision-recall score RF: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test, y_score_xgb)

pr_auc_xgb = auc( recall,precision)

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_xgb))
```

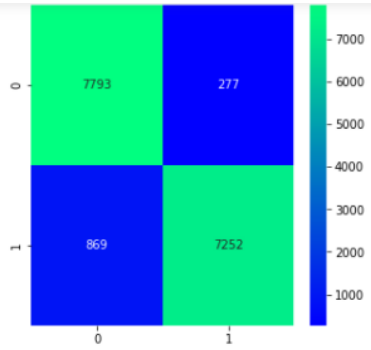
```
Average precision-recall score RF: 0.7750171582547667
```

Out[8]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AUC=0.77')
```



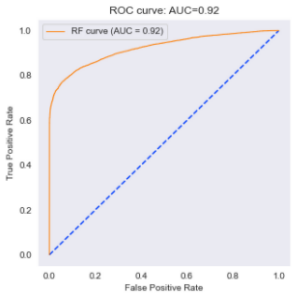
In [13]:
```
from imblearn.over_sampling import SMOTE

np.random.seed(123)


x_resample, y_resample  = SMOTE().fit_resample(x, y)


x_train2, x_test2, y_train2, y_test2 = train_test_split(x_resample, y_resample, test_size = 0.3, random_state = 0)

start_time = timer(None)

model_o = XGBClassifier()
model_o.fit(x_train2, y_train2)

y_pred = model_o.predict(x_test2)
y_score_xgb = model_o.predict_proba(x_test2)[:,-1]

print("Training Accuracy: ", model_o.score(x_train2, y_train2))
print('Testing Accuarcy: ', model_o.score(x_test2, y_test2))

# classification report
cr = classification_report(y_test2, y_pred)
print(cr)
timer(start_time)

# confusion matrix
cm = confusion_matrix(y_test2, y_pred)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'winter',fmt=".0f")
plt.show()
```

```
[13:49:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Training Accuracy:  0.9529103470181847
Testing Accuarcy:  0.9292199370020382
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      8070
           1       0.96      0.89      0.93      8121

    accuracy                           0.93     16191
   macro avg       0.93      0.93      0.93     16191
weighted avg       0.93      0.93      0.93     16191


 Time taken: 0 hours 0 minutes and 2.99 seconds.
```



In [14]:
```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(10,4.5))
fig.subplots_adjust(bottom=0.10, left=0.10, top = 0.900, right=1.00)
fig.suptitle(' Target Class Before and After Over Sampling', fontsize = 20)
sns.set_palette("bright")
sns.countplot(y_train, ax=ax1)
ax1.margins(0.1)
ax1.set_facecolor("#e1ddbf")
for p in ax1.patches:
        ax1.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_palette("bright")
sns.countplot(y_train2, ax=ax2)
ax2.margins(0.1)
ax2.set_facecolor("#e1ddbf")
for p in ax2.patches:
        ax2.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()+50))
sns.set_style('dark')
```

In [15]:
```
ns_probs = [0 for _ in range(len(y_test2))]
lr_probs = model.predict_proba(x_test2)
lr_probs = lr_probs[:, 1]
lr_probs


# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test2, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test2, lr_probs)

roc_auc_xgb = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_xgb))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
          roc_auc_xgb))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

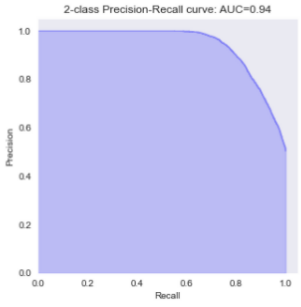


In [16]:
```
average_precision = average_precision_score(y_test2, lr_probs)

print('Average precision-recall score RF: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test2, lr_probs)

pr_auc_xgb = auc( recall,precision )

plt.step(recall, precision, color='b', alpha=0.2,
```

In [16]:
```
average_precision = average_precision_score(y_test2, lr_probs)

print('Average precision-recall score RF: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test2, lr_probs)

pr_auc_xgb = auc( recall,precision )

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.05])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_xgb))
```

```
Average precision-recall score RF: 0.9407991634991688
```

Out[16]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AUC=0.94')
```



In [17]:
```
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix

xgb_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')


print("Mean AUC Score - XGBoost: ", xgb_cv_score.mean())
```

```
[13:55:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:55:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:55:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:55:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:55:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:55:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:56:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:56:01] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:56:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[13:56:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Mean AUC Score - XGBoost:  0.8691560119342394
```

In [54]:
```
random_grid = {
             'learning_rate' : [0.05,0.1,0.15,0.2,0.25,0.3],
              'max_depth' : [3,4,5,6,8,10,12,15],
              'min_child_weight' : [1,3,5,7],
              'gamma' : [0.0,0.1,0.2,0.3,0.4],
              'colsample_bytree' : [0.3,0.4,0.5,0.7]
}
```

In [55]:
```
from sklearn.model_selection import RandomizedSearchCV

# Random search of parameters
```

In [55]:
```
from sklearn.model_selection import RandomizedSearchCV

# Random search of parameters
xgb_random = RandomizedSearchCV(estimator = model_o, param_distributions = random_grid, n_iter = 5, cv = 3, verbose=2, random_sta
# Fit the model
xgb_random.fit(x_train2, y_train2)
# print results
print(xgb_random.best_params_)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[16:10:11] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
{'min_child_weight': 1, 'max_depth': 15, 'learning_rate': 0.15, 'gamma': 0.4, 'colsample_bytree': 0.3}
```

In [56]:
```
xgb_random.best_estimator_
```

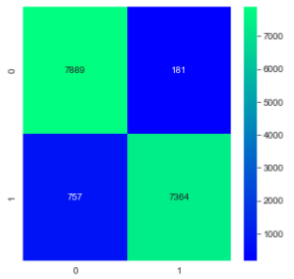
Out[56]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0.4, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [18]:
```
start_time = timer(None)

model_o = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0.4, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)


model_o.fit(x_train2, y_train2)

y_pred = model_o.predict(x_test2)
y_score_xgb = model_o.predict_proba(x_test)[:,-1]
print("Training Accuracy: ", model_o.score(x_train2, y_train2))
print('Testing Accuarcy: ', model_o.score(x_test2, y_test2))
xgb_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')


# classification report
cr = classification_report(y_test2, y_pred)
print(cr)
timer(start_time)
```

In [18]:
```
start_time = timer(None)

model_o = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0.4, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.15, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)


model_o.fit(x_train2, y_train2)

y_pred = model_o.predict(x_test2)
y_score_xgb = model_o.predict_proba(x_test)[:,-1]
print("Training Accuracy: ", model_o.score(x_train2, y_train2))
print('Testing Accuarcy: ', model_o.score(x_test2, y_test2))
xgb_cv_score = cross_val_score(model_o, x, y, cv=10, scoring='roc_auc')


# classification report
cr = classification_report(y_test2, y_pred)
print(cr)
timer(start_time)

# confusion matrix
cm = confusion_matrix(y_test2, y_pred)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'winter',fmt=".0f")
plt.show()


print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", xgb_cv_score.mean())
```

```
[14:17:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Training Accuracy:  0.9946795838958151
Testing Accuarcy:  0.9420665801988759
[14:17:47] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[14:17:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[14:17:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      8070
           1       0.98      0.91      0.94      8121

    accuracy                           0.94     16191
   macro avg       0.94      0.94      0.94     16191
weighted avg       0.94      0.94      0.94     16191


 Time taken: 0 hours 0 minutes and 40.5 seconds.
```

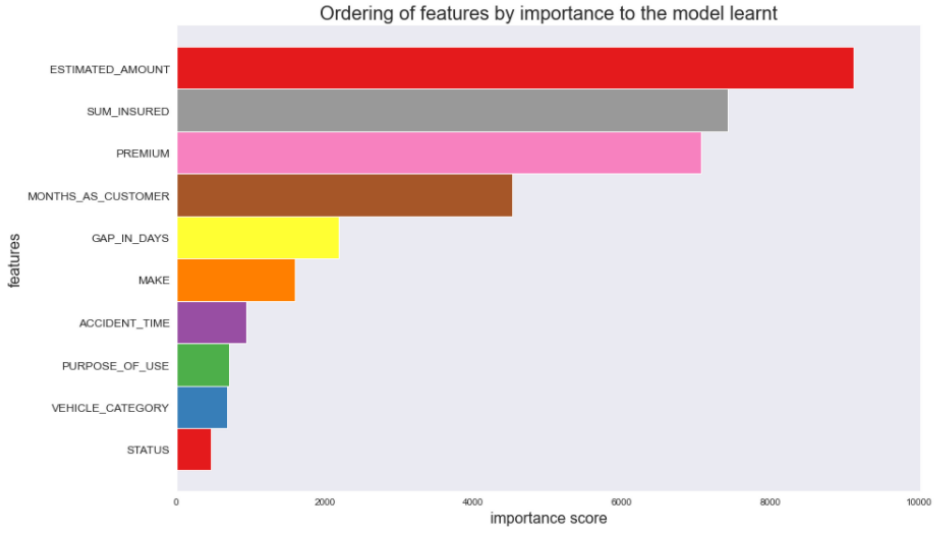



```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8684413010847196
```

In [71]:
```
fig = plt.figure(figsize = (14, 9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0, 1, 9))

ax = xgb.plot_importance(model_o, height = 1, color = colours, grid = False, \
                         show_values = False, importance_type = 'weight', ax = ax);
for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size = 16);
ax.set_ylabel('features', size = 16);
ax.set_yticklabels(ax.get_yticklabels(), size = 12);
ax.set_title('Ordering of features by importance to the model learnt', size = 20);
```

```
=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8684413010847196
```

In [71]:
```
fig = plt.figure(figsize = (14, 9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0, 1, 9))

ax = xgb.plot_importance(model_o, height = 1, color = colours, grid = False, \
                   show_values = False, importance_type = 'weight', ax = ax);
for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size = 16);
ax.set_ylabel('features', size = 16);
ax.set_yticklabels(ax.get_yticklabels(), size = 12);
ax.set_title('Ordering of features by importance to the model learnt', size = 20);
```



In [19]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(random_state = 42)
from pprint import pprint

print('Parameters currently in use:\n')
pprint(xgb.get_params())
```

```
Parameters currently in use:

{'base_score': None,
 'booster': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'gamma': None,
 'gpu_id': None,
 'importance_type': 'gain',
 'interaction_constraints': None,
 'learning_rate': None,
 'max_delta_step': None,
 'max_depth': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'n_estimators': 100,
 'n_jobs': None,
 'num_parallel_tree': None,
 'objective': 'binary:logistic',
 'random_state': 42,
 'reg_alpha': None,
 'reg_lambda': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'use_label_encoder': True,
 'validate_parameters': None,
 'verbosity': None}
```

In [ ]:

7.4. Neural Network Codes and results

In [19]:
```
# IMPORTING REQUIRED MODULES
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [20]:
```
# NEURAL NETWORKS MODULES
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

In [21]:
```
# KERAS MODULES
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from keras.callbacks import Callback
import keras.backend as kb
import tensorflow as tf
```

In [22]:
```
from datetime import datetime

def timer(start_time= None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now()-start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec,2)))
```

In [23]:
```
data = pd.read_csv("Final Data set v5.csv")
```

In [24]:
```
data.head()
```

Out[24]:

	SUM_INSURED	ESTIMATED_AMOUNT	PREMIUM	MONTHS_AS_CUSTOMER	VEHICLE_CATEGORY	STATUS	ACCIDENT_TIME	PURPOSE_OF_USE	MAKE	GA
0	1000000	48110.0	8514.85	86	3	3	0	10	35	
1	3000000	80000.0	54000.00	72	2	4	1	13	24	
2	3000000	500000.0	54000.00	55	2	4	6	13	24	
3	750000	37050.0	4792.65	184	2	7	4	13	35	
4	750000	68750.0	4792.65	184	2	7	3	13	35	

In [25]:
```
# SETTING UP THE TRAINING AND TESTING SETS
np.random.seed(123)

x =  data.iloc[:,0:10]
y = data.iloc[:, -1]
```

In [26]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [27]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout


model = Sequential([
    Dense(units=20, input_dim = x_train.shape[1], activation='relu'),
    Dense(units=24,activation='relu'),
    Dropout(0.5),
    Dense(units=20,activation='relu'),
    Dense(units=24,activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 20)                220
_________________________________________________________________
dense_12 (Dense)             (None, 24)                504
_________________________________________________________________
dropout_3 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_13 (Dense)             (None, 20)                500
_________________________________________________________________
dense_14 (Dense)             (None, 24)                504
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,753
Trainable params: 1,753
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```

start_time = timer(None)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=30, epochs=5)

timer(start_time)
```

```
Epoch 1/5
21068/21068 [==============================] - 1s 34us/step - loss: 1575.3485 - accuracy: 0.8925
```

_____

In [28]:
```

start_time = timer(None)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=30, epochs=5)

timer(start_time)
```

```
Epoch 1/5
21068/21068 [==============================] - 1s 34us/step - loss: 1575.3485 - accuracy: 0.8925
Epoch 2/5
21068/21068 [==============================] - 1s 28us/step - loss: 154.5537 - accuracy: 0.8130
Epoch 3/5
21068/21068 [==============================] - 1s 30us/step - loss: 19.8526 - accuracy: 0.8711
Epoch 4/5
21068/21068 [==============================] - 1s 33us/step - loss: 2.7980 - accuracy: 0.9356
Epoch 5/5
21068/21068 [==============================] - 1s 32us/step - loss: 8.9680 - accuracy: 0.9222

 Time taken: 0 hours 0 minutes and 3.87 seconds.
```

In [29]:
```
score = model.evaluate(x_test, y_test)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(score[1]*100,score[0]))
```

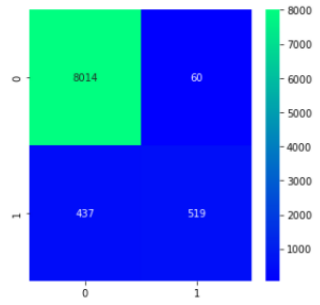
```
9030/9030 [==============================] - 0s 13us/step
Test Accuracy: 94.50%
Test Loss: 68.82941647906966
```

In [30]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(x_test)
y_test = pd.DataFrame(y_test)



# making a classification report
cr = classification_report(y_test,  y_pred.round())
print(cr)



# making a confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
cm = confusion_matrix(y_test, y_pred.round())
sns.heatmap(cm, annot = True, cmap = 'winter', fmt=".0f")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      8074
           1       0.90      0.54      0.68       956
```

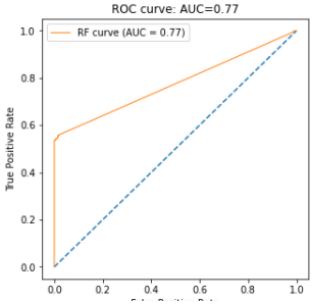
```
# making a classification report
cr = classification_report(y_test,  y_pred.round())
print(cr)


# making a confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
cm = confusion_matrix(y_test, y_pred.round())
sns.heatmap(cm, annot = True, cmap = 'winter', fmt=".0f")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      8074
           1       0.90      0.54      0.68       956

    accuracy                           0.94      9030
   macro avg       0.92      0.77      0.82      9030
weighted avg       0.94      0.94      0.94      9030
```



In [31]:

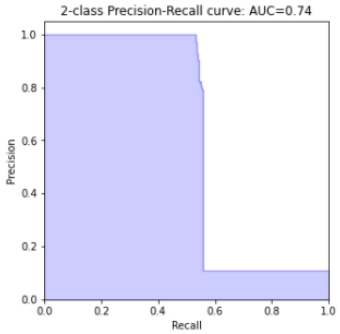
```
from sklearn.metrics import roc_curve, auc

from matplotlib import pyplot



ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model.predict(x_test).ravel()



# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

roc_auc_rf = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_rf))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
        roc_auc_rf))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



In [32]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

y_score_ANN = model.predict(x_test).ravel()
average_precision = average_precision_score(y_test, y_score_ANN)

print('Average precision-recall score ANN: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test, y_score_ANN)

pr_auc_ANN = auc( recall,precision)

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_ANN))
```

```
Average precision-recall score ANN: 0.6017715388308678
```

Out[32]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AUC=0.74')
```



In [33]:
```
from imblearn.over_sampling import SMOTE

np.random.seed(123)

x_resample, y_resample  = SMOTE().fit_resample(x, y)
```

In [34]:
```
x_train2, x_test2, y_train2, y_test2 = train_test_split(x_resample, y_resample, test_size = 0.3, random_state = 0)

model = Sequential([
    Dense(units=20, input_dim = x_train2.shape[1], activation='relu'),
    Dense(units=24,activation='relu'),
    Dropout(0.5),
    Dense(units=20,activation='relu'),
    Dense(units=24,activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 20)                220
_________________________________________________________________
dense_17 (Dense)             (None, 24)                504
_________________________________________________________________
dropout_4 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_18 (Dense)             (None, 20)                500
_________________________________________________________________
dense_19 (Dense)             (None, 24)                504
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,753
Trainable params: 1,753
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
start_time = timer(None)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train2, y_train2, batch_size=30, epochs=5)

timer(start_time)
```

```
Epoch 1/5
37779/37779 [==============================] - 1s 37us/step - loss: 4332.5200 - accuracy: 0.6777
Epoch 2/5
37779/37779 [==============================] - 1s 33us/step - loss: 175.1594 - accuracy: 0.6710
Epoch 3/5
37779/37779 [==============================] - 1s 32us/step - loss: 18.4851 - accuracy: 0.6435
Epoch 4/5
37779/37779 [==============================] - 1s 34us/step - loss: 0.6283 - accuracy: 0.7621
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 20)                220
_________________________________________________________________
dense_17 (Dense)             (None, 24)                504
_________________________________________________________________
dropout_4 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_18 (Dense)             (None, 20)                500
_________________________________________________________________
dense_19 (Dense)             (None, 24)                504
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,753
Trainable params: 1,753
Non-trainable params: 0
_________________________________________________________________
```

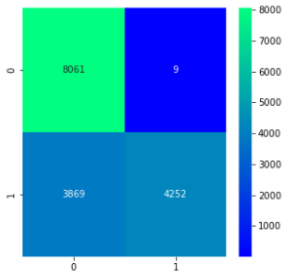
In [35]:
```
start_time = timer(None)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train2, y_train2, batch_size=30, epochs=5)

timer(start_time)
```

```
Epoch 1/5
37779/37779 [==============================] - 1s 37us/step - loss: 4332.5200 - accuracy: 0.6777
Epoch 2/5
37779/37779 [==============================] - 1s 33us/step - loss: 175.1594 - accuracy: 0.6710
Epoch 3/5
37779/37779 [==============================] - 1s 32us/step - loss: 18.4851 - accuracy: 0.6435
Epoch 4/5
37779/37779 [==============================] - 1s 34us/step - loss: 0.6283 - accuracy: 0.7621
Epoch 5/5
37779/37779 [==============================] - 1s 33us/step - loss: 0.5498 - accuracy: 0.7651

 Time taken: 0 hours 0 minutes and 6.94 seconds.
```

In [36]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(x_test2)
y_test = pd.DataFrame(y_test2)


# making a classification report
cr = classification_report(y_test2,  y_pred.round())
print(cr)
```

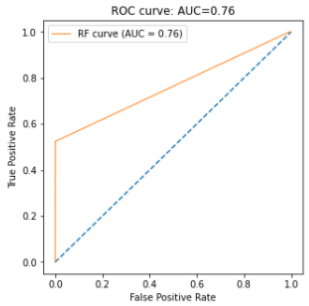
In [36]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(x_test2)
y_test = pd.DataFrame(y_test2)


# making a classification report
cr = classification_report(y_test2,  y_pred.round())
print(cr)


# making a confusion matrix
plt.rcParams['figure.figsize'] = (5, 5)
cm = confusion_matrix(y_test2, y_pred.round())
sns.heatmap(cm, annot = True, cmap = 'winter', fmt=".0f")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.68      1.00      0.81      8070
           1       1.00      0.52      0.69      8121

    accuracy                           0.76     16191
   macro avg       0.84      0.76      0.75     16191
weighted avg       0.84      0.76      0.75     16191
```



In [37]:
```
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model.predict(x_test2).ravel()
```

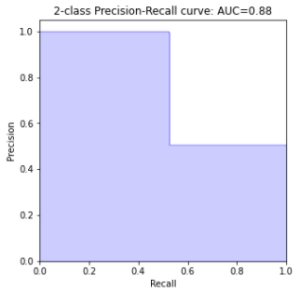
In [37]:
```
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model.predict(x_test2).ravel()



# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test2, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test2, lr_probs)

roc_auc_ANN = auc(lr_fpr, lr_tpr)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, lw=1, label='{} curve (AUC = {:0.2f})'.format('RF',roc_auc_ANN))
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
plt.title('ROC curve: AUC={0:0.2f}'.format(
          roc_auc_ANN))
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```



In [38]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

y_score_ANN = model.predict(x_test2).ravel()
average_precision = average_precision_score(y_test2, y_score_ANN)

print('Average precision-recall score ANN: {}'.format(average_precision))

precision, recall, _ = precision_recall_curve(y_test2, y_score_ANN)

pr_auc_ANN = auc( recall,precision)

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AUC={0:0.2f}'.format(
          pr_auc_ANN))
```

```
Average precision-recall score ANN: 0.7635897062509686
```

Out[38]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AUC=0.88')
```



7.5. Final Dataset

Microsoft Excel Worksheet