University of Science and Technology of Hanoi

Bachelor's Thesis in Information and Communication Technology

Application of Machine Learning in Credit Card Fraud Detection

Authors:
DANG Anh Duc
BI9068

Supervisor:
DOAN Nhat Quang
ICT Lab
ICT Deparment
Vietnam France University

*A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science*

in

Information and Communication Technology

June 18, 2021

USTH
VIETNAM FRANCE UNIVERSITY

Acknowledgement

I would like to express my sincere gratitude towards Dr. DOAN Nhat Quang for his detailed and constructive suggestions during the internship. He continuously guided me in the right direction throughout the span of this project. In addition, he also provided me with valuable knowledge, not only about the research topic, but also other important skills such as how to do a research or how to write a proper research report. I would like to extend my gratitude to Dr. TRAN Giang Son for helping us with the ICT Lab server.

I would also like to thank my friends, NGUYEN Minh Thu, TRINH Mai Phuong, NGUYEN Truong Giang and TRAN Thanh Long. Without their help, this project would not have been possible.

Abstract

Credit card fraud is an ever-growing menace in the financial world. The number of fraudulent transactions is expected to increase due to the recent trend of using non-cash payments. However, using machines to detect credit card fraud is not an easy task since the available datasets for this problem are highly imbalanced i.e. the number of genuine cases greatly outnumber the fraudulent cases, which makes the process of training a classification model harder and create inaccurate models.

In order to tackle this problem, our project suggests different techniques to resample the dataset, such as, undersampling, oversampling and hybrid strategy, which is a combination of both undersampling and oversampling. These techniques are implemented with different predictive models like Logistic Regression, Random Forest and XGBoost. Each combination between a resampling method and model is evaluated based on precision, recall, f1-score, precision-recall (PR) curve and receiver operating characteristics (ROC) curve.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Credit Card Fraud Detection

E-commerce has greatly developed in the past years and has become an essential means for most companies, corporations and government agencies to increase their productivity in global trading. One of the main reasons for this success is the easy and fast credit card transaction. However, whenever we talk about monetary transactions, we must also take financial fraud into account. As credit card transaction has become one of the most common method of payment in recent years, fraudulent activities have also been increasing rapidly.

The solution to this matter can be categorized into prevention, which involves preventing the fraudulent transaction to happen, and detection, which spots the fraudulent transaction after the transaction is made so that proper actions can be taken. In terms of prevention, there are already some common technology in place to prevent fraud such as Address Verification System (AVS) which verify that the address entered by the customer is associated with the cardholder's credit card account and Cardholder Verification Method (CVM) which verifies the authenticity of the cardholder through signature, personal identification number (PIN), etc.

However, when preventive measures fail to stop a fraudulent transaction from happening, it should be detected as soon as possible so that necessary actions can be taken timely. Credit card fraud detection is the process of detecting whether a transaction made via a credit card is legitimate or not. Due to increasing traffic of transaction data, it would be impossible for humans to manually check every transaction to find a fraud case. Therefore, an automated fraud detection system is required to solve this problem. This thesis will attempt to such system focusing on the implementation of machine learning.

1.2 Aim of the Project

In this project, our main objective is to explore different techniques to deal with an imbalanced dataset and evaluate them to see which method performs better than the others. More specifically, this project focuses on handling a credit card transaction dataset to build a model to detect fraudulent transactions by using different sampling methods along with various models. After that we chose the most well-performed model based on a range of evaluation metrics.

1.3 Overview

This section provides an overall overview of the content entailed in each section. In section 2, we discuss relevant literature in the current field of research, focusing on the methods to build a credit card fraud detection model. Section 3 presents the methodology including the data processing steps, model selection, as well as the training of the model. In Section 4, we describe the model's evaluation metrics - precision, recall, f1-score, PR curve, ROC curve and provide a detailed discussion on the results of our project. The final section 5 presents a brief conclusion of our project.

2 Literature Review

Since credit card fraud is a massive problem in finance, many institutions have invested in developing fraud detection systems. Many researchers have been actively working on different ways to tackle difficulties when building a fraud detection system such as choosing the best predictive model, changes in fraudulent behaviors, class overlapping, etc. In this chapter, we will discuss some of the related works relating to this problem.

In a study in 2014, a group of researchers from Université Libre de Bruxelles [11] focused on two different ways to detect fraud: static learning, where the data are processed all at once in a single batch, often seasonally, and incremental learning, which interprets data as a continuous stream and process new data as soon as it is available to the system. [11] concluded that the incremental approach is better since it can effectively deal the changes in fraudulent behavior over time. The researchers also proposed that Average Precision (AP) and Area Under Curve (AUC) are the best metrics for fraud detection systems. In another study by Pozzolo et al. [10], Random Forest was concluded to be the most efficient predictive model for fraud detection task.

In [2], Awoyemi et al. used K-nearest neighbors, logistic regression and Naïve Bayes to perform fraud detection on a transaction dataset that was resampled using Synthetic Minortiy Over-sampling Technique (SMOTE) to address the imbalance problem. The research suggested that K-nearest neighbors outperformed the other two in terms of precision, recall, balanced accuracy [20], specificity and Mathews correlation [9].

In 2018, a unique approach to fraud detection was proposed by Carcillo et al. [6]. They came up with a system called Scalable Real-time Fraud Finder (SCARFF), in which Big Data tools (Kafka, Spark and Cassandra) were integrated with a machine learning approach which deals with imbalance, nonstationarity and feedback latency. They mainly focus on the fact that a fraud detection system would require a real-time setting and with a massive amount of transactions data, Big Data technology holds advantages over other conventional system due to its higher scalability, fault-tolerant.

Aleskerov et al. [1] proposed using a data mining system that uses neural networks for fraud detection. A research in 2008 [19] implemented Hidden Markov Model in which the spending habit of the customers is analyzed in order to detect fraudulent transactions. Wheeler and Aitken in 2000 looked into different algorithms that could be used to build a fraud detection system.

Although, there have been multiples researches carried out on different aspects of fraud detection such as detection time, improving performance, different techniques and customers' behaviors, there are very little researches that focused on the problem of class imbalance in the dataset. Therefore, our project aims to implement various techniques in data resampling combining with different predictive model to tack the class imbalance problem.

3 Methodology

3.1 Dataset Description

For this project, we used a dataset consists of transactions made by credit cards in two days in September 2013 by European cardholders which was collected by the Machine Learning Group of Université Libre de Bruxelles (ULB) and was published on Kaggle[1]. The dataset is contains a total of 284,807 transactions, in which only 492 are fraudulent. The dataset is considered to be highly skewed as the positive class (frauds) only accounts for 0.172% of the dataset. Figure 1 visualizes the class distribution of the dataset.

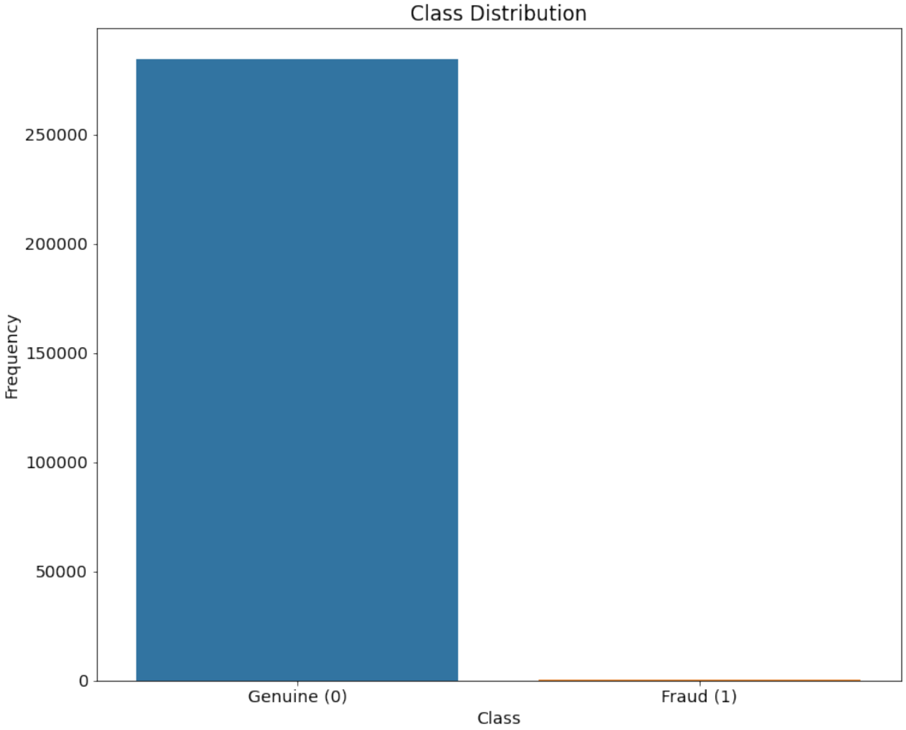


Figure 1: Dataset Class Distribution

The dataset only contains numerical values as a result of Principal Components Analysis (PCA) transformation. Due to confidentiality issue, most

[1] https://www.kaggle.com/mlg-ulb/creditcardfraud

of original attributes was not revealed. There are total 30 features, 28 of which was generated by PCA. The only features that was not transformed are '*Time*' and '*Amount*' Feature 'Class' is the target attribute and it takes value 1 in case of fraud and 0 otherwise. Table 1 gives a detailed description about the dataset's attributes.

Attribute	**Type**	**Description**
Time	Integer	Time elapsed between each transaction and the first transaction
V1	Double	First PCA component
V2	Double	Second PCA component
...	...	...
V28	Double	Last PCA component
Amount	Double	Transaction amount
Class	Integer	Target class (0 = Genuine and 1 = Fraud)

Table 1: Dataset Attribute Description

3.2 Data Pre-processing

3.2.1 Data Standardzation

It is a common requirement for machine learning techniques models that the data is normalized before training. A dataset which was not normalized before training might lead to undesirable outcomes as variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. Although for Logistic Regression and Tree based models, they are not as sensitive to the magnitude of the variables as other models such as K-Nearest Neighbors or Support Vector Machine (SVM) [15], we still decided to apply standardization for the data in order to retrieve the best possible result. Standardization can be achieved as follows:

$$z = \frac{Value - Mean}{StandardDeviation}$$

where:

$$Mean(\mu) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$StandardDeviation(\sigma) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

For this project, we standardize our data using the StandardScaler module in the scikit-learn library for convenience.

3.2.2 Data Splitting

After having standardized our data, we split the dataset into training set (70%) and test set (30%). For consistency whenever the program is executed, we assigned a constant *random_state* (14) when splitting. The training set will then be resampled and trained by different techniques and models as well as to tune each model's hyperparameter. The test set will be used to evaluate the performance of the models only.

3.3 Data Resampling

As mentioned in section 3.1, the dataset is highly unbalanced as the number of legitimate transactions outnumbers the number of fraudulent transactions. If we train our model directly on this dataset, the outcome will likely to be biased towards the genuine transactions. To tackle this problem, we used some commonly known resampling techniques such as Random Oversampling, Random Undersampling, Synthetic Minority Oversampling Technique (SMOTE) and Tomek Links Removal Undersampling.

In underampling methods, the majority class is reduced to a certain percent compared to the original data to make the number of instances between two classes balanced. Figure 2 depicts the main idea of undersampling. Undersampling is easy to implement and applying undersampling would greatly save training time and storage space when a huge dataset is used. In contrast to undersampling, in oversampling, we will work with the minority class. Instances of the minority class will be duplicated or new data points of that

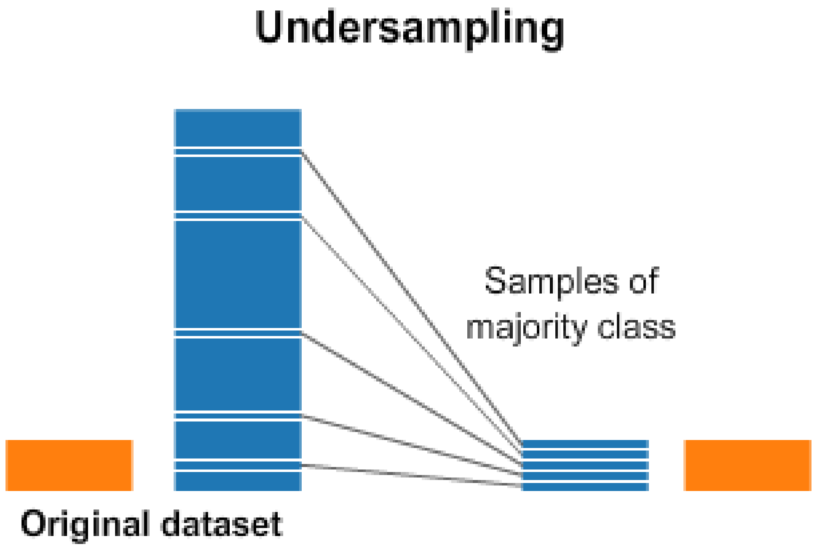


Figure 2: Undersampling Method

class will be generated to balance the ratio two classes, which is shown in Figure 3.

3.3.1 Random Oversampling

In Random Oversampling, random samples of the minority class is replicated to make the dataset balanced. Howerver, there is a high possiblity that the models would overfit the data since many instances of the same sample in the minority class is duplicated.

3.3.2 Random Undersampling

In Random Undersampling, random samples of the majority class removed from that dataset in order to balance the number of data between two classes. Unlike Random Oversampling, this method is less prone to overfitting; however during the process of elimination, useful information may be lost, resulting in inaccurate predictions by the models.

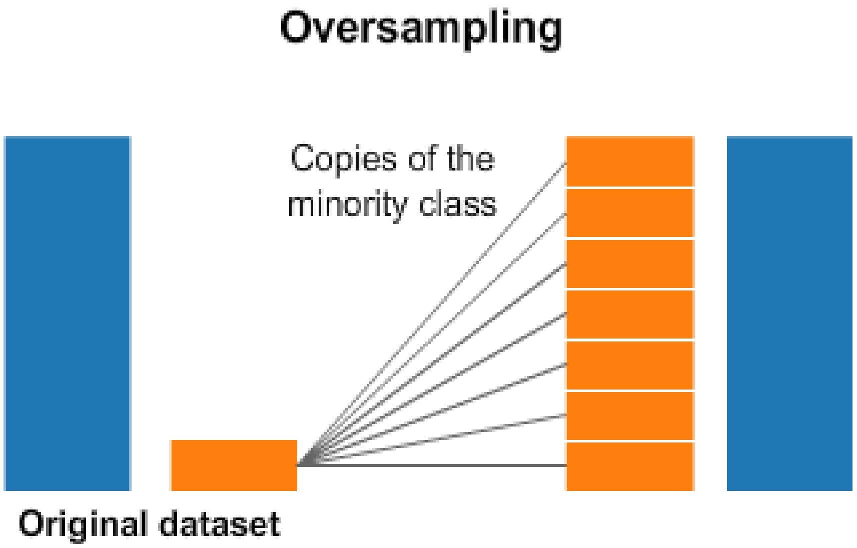


Figure 3: Oversampling Method

3.3.3 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is a popular oversampling techniques that was developed by Chawla, Bowyer, Hall and Kegelmeyer [7]. Instead of duplicating existing data like in Ranom Oversampling, SMOTE creates new data samples by interpolating between nearest minority samples. Number of chosen nearest neighbors would depend on the amount of oversampling required. By creating new data, this techniques tackled the problem of overfitting proposed by Random Oversampling. Figure 4 describes how synthetic examples are created.

3.3.4 Tomek Links Removal Undersampling

Given two samples A and B of two different classes, if there isn't a sample C such that the distance between C and A or between C and B is less than the distance A and B, the pair (A,B) is a Tomek link [12]. Removing these links by eliminating the majority samples associated to these links, we will be able to perform undersampling for the dataset and also make the boundaries between two classes clearer.

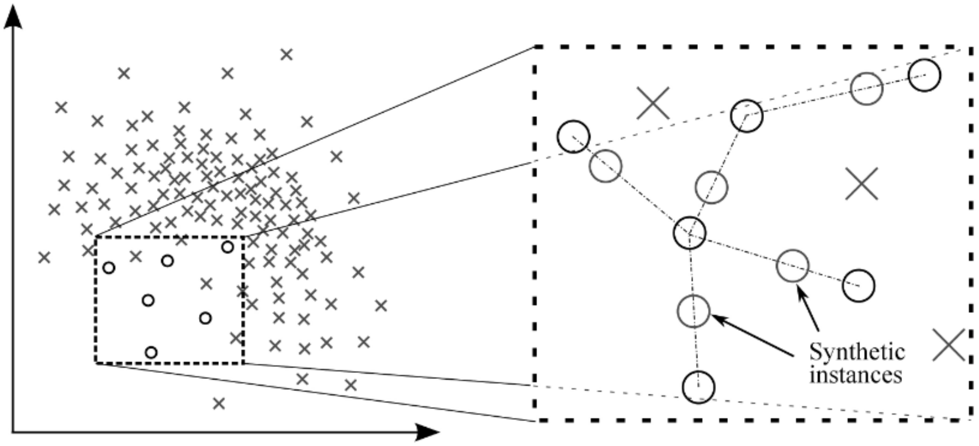


Figure 4: Synthesizing data using SMOTE [21]

3.3.5 Combination of SMOTE and Tomek Links Removal

While creating new synthetic data points, minority clusters can get mixed up with the majority samples in the data spaces. In order to mitigate this situation, both SMOTE and Tomek Links Removal can be used to balance the dataset. Tomek Links Removal will be applied after synthesizing new minority data to remove samples that have invaded the majority class space.

3.4 Models Selection

Due to the nature of classification problem, we chose to implement three very common predictive models which are Logistic Regression, Random Forest and XGBoost.

3.4.1 Logistic Regression

Linear Regression [16] is one of the most popular machine learning algorithm when it comes to classification. In Logistic Regression, the prediction is expressed as the probability of that sample belonging to each class. Logistic Regression was built on another popular which is Linear Regression. In a Linear Regression model, the output is predicted as a combination of weighted inputs. The hypothesis of Linear Regression can be expressed as:

$$y = a_0 + \sum_{i=1} a_i x_i$$

where a_0 is a constant (bias term) and a_i is the weight of input variable x_i and y is the predicted output, which can be any values possible. However, since the output of Logistice Regression is a probability, the output must be in the range [0, 1]. To tackle this problem, Logistic Regression uses sigmoid function to squash the real value between 0 and 1. The sigmoid function is defined as:

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Figure 5 is the graphical representation of the sigmoid function. In a binary classification problem, the output represent the likelihood that the example belong to the positive class. The output of a Logistic Regression can be expressed as:

$$P(y = 1) = sigmoid(a_0 + \sum_{i=1} a_i x_i)$$

By default, Logistic Regression use 0.5 as its threshold. Any probability below 0.5 is classified as 0 and any probablity above 0.5 is classified as 1.

$$y = 1 \text{ if } P(y = 1) \geq 0.5$$

$$y = 0 \text{ if } P(y = 1) < 0.5$$

However, this threshold can be modified according to different needs.

3.4.2 Random Forest

Random Forest is an extension of bagging ensemble method, which involves the combination of many weak learners to predict the outcome [4] [18]. For Random Forest, these weak learners are decision trees. We will discuss the basic of decision trees in order to understand Random Forest better.

Decision Tree is a supervised learning algorithm, composed of several internal nodes where each node represents a specific test for an attribute (e.g, whether you have work to do or not, or whether the weather is sunny or rainy). Each branch of the tree represent the outcome of the test and leaf nodes represent the outcome. Decision tree break a training set down into several subsamples. Figure 6 shows an example of a decision tree that

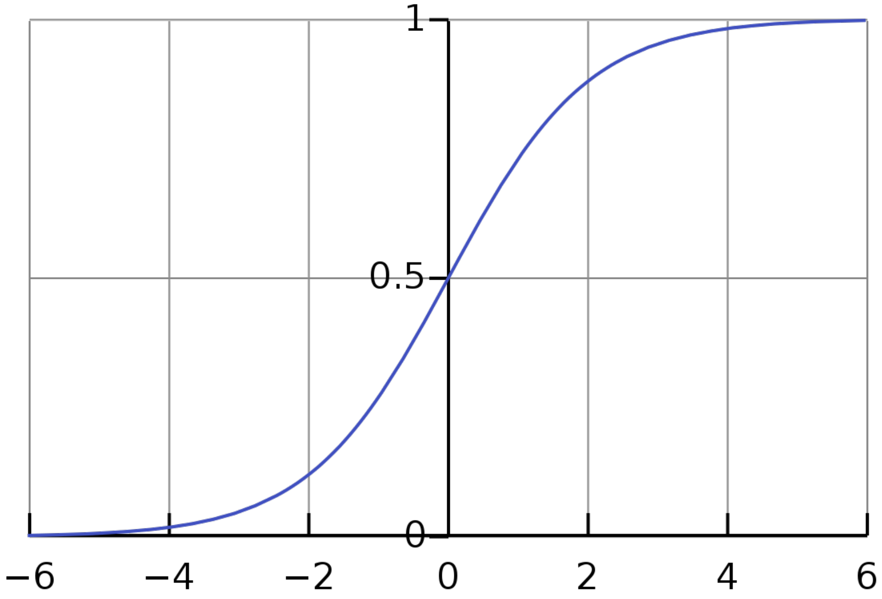


Figure 5: Sigmoid Graph [13]

tries to decide whether you should stay in or go out. It starts with a test to see if you have any work left to do. If it is true that you have work to do, the outcome will be to stay in. If the answer is no, it will perform the next test where it check the outlook condition. There are three possible choices: sunny, over-cast and rainy. If the weather is either sunny or overcast, the output of the tree will be go to the beach and go running respectively. In case the weather is rainy, the tree will perform one last test: to check if your friends are busy. If they are free, you guys can go to the movies, otherwise, you should stay home. Through this example, we can see that a decision tree uses a series of if-then conditions to make the final decision.

Although decision trees are easy to understand and perform well in some datasets, they tend to have a high variance due to their greedy approach. The trees tend to always select the best split at each level and do not have the capability to look further than the current one. Due to this reason, the decision tree pose the possibility of overfitting the data which leads to poor performance on unseen data in test sets.

Random Forest mitigates the overfitting by using bootstrap. Bootstrap-

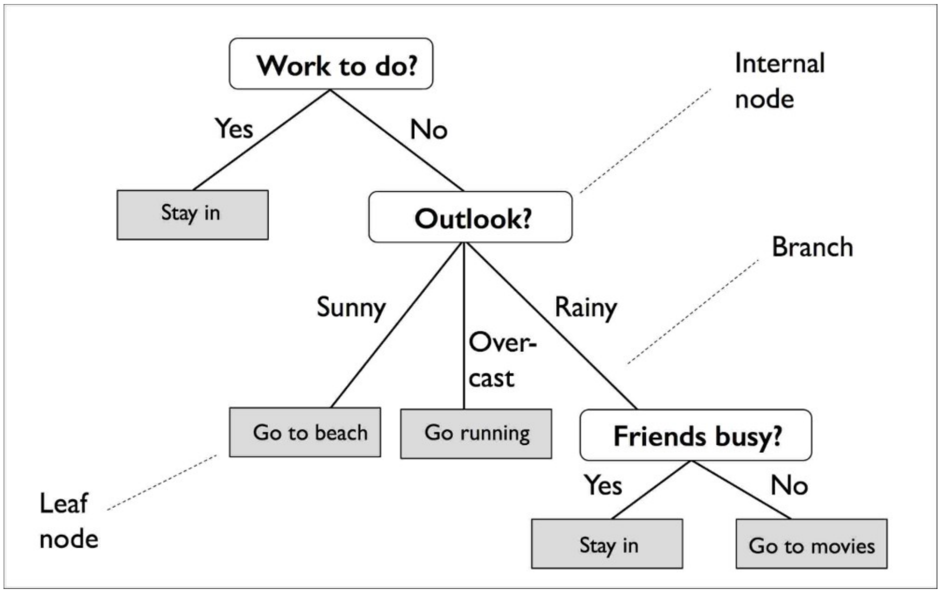


Figure 6: Decision Tree Example

ping is the process of sampling the training data randomly with replacement. Random Forest utilizes bootstrapping such that each decision tree is trained on a different subset of the training data. Moreover, Random Forest uses random subsets of attributes. For example, if there are 30 features in the data, random forest will only choose some of them to train a decision tree. Each tree will have the same number of features to train on. After building and training the decision trees, the results of each tree are aggregated to produce the final decision of the forest. The trained forest will ensure generalization since multiple decision trees are used to make the final decision and each tree is train on a different subset of the dataset. Figure 7 shows the construction of a random forest.

3.4.3 XGBoost

Unlike Bagging, where models run in parallel and the output is combined at the final stage, Boosting trains weak learners sequentially so that each learner tries to correct its predecessors by focusing and adding more weights on the samples that were misclassified.

XGBoost stands for eXtreme Gradient Boosting. XGBoost is an advance

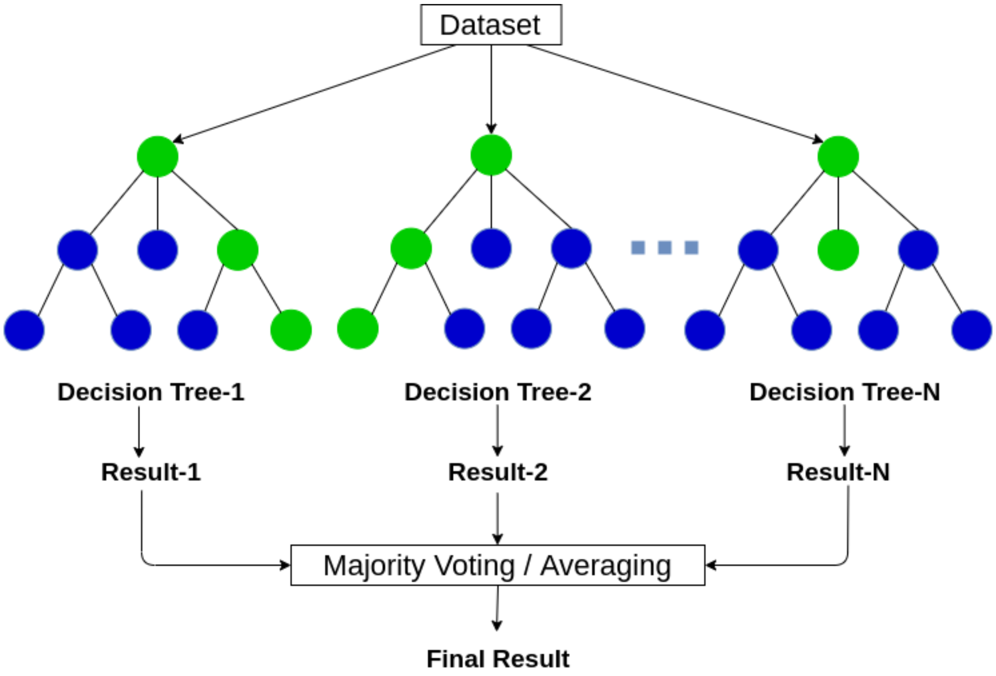


Figure 7: Random Forest Example

implementation of gradient boosted decision trees designed for speed and performance [8]. Since XGBoost is based on Gradient Boosting, we will first discus it briefly. In Gradient Boosting, at each level of the learning process, the weak learners predict the class of the observations and then calculate the difference between the real value and the predicted value of these observations (i.e. loss). Depend on the calculated loss, it creates new weak learners to train on and minimize these errors. This process keeps on going until it reach a certain threshold.

As mentioned before, XGBoost is an advance version of Gradient Boosting that uses decision trees as weak learners; therefore, it has many advantages compared to Gradient Boosting. The standard Gradient Boosting is generally slow in implementation due to sequential model training. Thus, XGBoost focuses on computational speed and model performance. XGBoost enables parallelization of tree construction for optimal speed. Gradient Boosting is a greedy algorithm that stop splitting nodes as soon as it encounters a negative loss; whereas, XGBoost will keep splitting until it reach the specified tree depth. In addition, XGBoost has a built-in cross validation function so that it is easier to number of boosting round for each run as well as others hyperparameters. Since XGBoost has quite a few important hyperparameters, we would have to tuned them all in order to get the best result from the model.

3.5 Hyperparameters Tuning Using Cross Validation

Hyperparameter of a model is the external configuration of a model that can not be estimated by looking at the data. Since the parameters can not be changed during train, they must be set manually beforehand. In order to for a model to perform at its best, its parameters have to be tuned find the best possible values. In this project, we used K-fold Cross Validation to tune the parameters for our models. More specifically, we used 10-fold Cross Validation by implementing *GridSearchCV* provided by *scikt-learn*. In 10-fold Cross Validation, we divided the training set into 10 folds, and each experiments will take 1 different fold as a validation set and the rest will be used for training. Since each model has different parameters, the hyperparameters tuning for each model was different.

3.5.1 Hyperparameters Tuning for Logistic Regression

In Logistic Regression models, the regularization parameter (C) is an important hyperparameter to keep an eye one. If C is too large, the model tends to overfit the data and vice versa. Given a list of possible of values for 'C', the *GridSearchCV* module will perform cross validation on the resampled data for all values of 'C' and return the best value for that model.

3.5.2 Hyperparameters Tuning for Random Forest

In Random Forests models, we performed 10-fold Cross Validation oon the resampled dataset in order to find the best values for *n_estimators*: the number of decision tree in the forest.

3.5.3 Hyperparameters Tuning for XGBoost

For XGBoost, there are many essential hyperparameters that needed tuning [17] [5] [23]. We performed 10-fold cross validation on the resampled datasets to find the best possible values for the following hyperparameters:

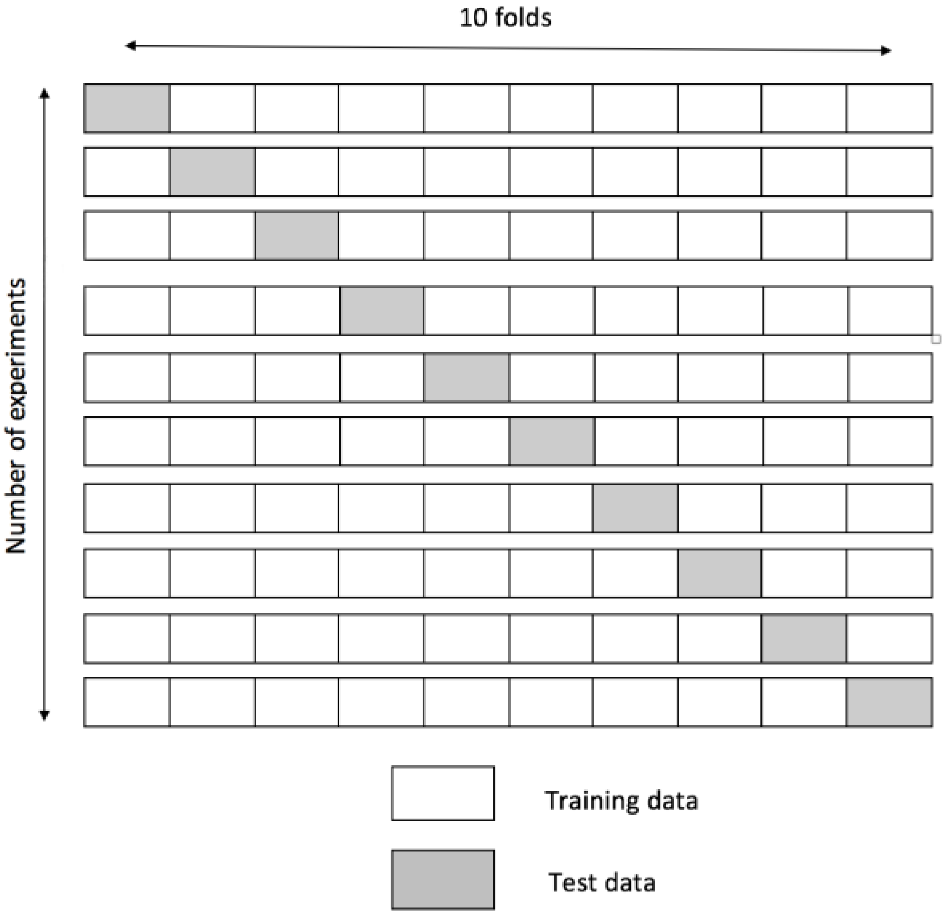


Figure 8: 10-Fold Cross Validation

- **n_estimators**: Total number of decision trees.
- **learning_rate**: The learning rate of the model.
- **subsample**: The fraction of observation to be randomly sampled by decision trees.
- **cosample_bytree**: The fraction of attributes to be randomly sampled by decision trees.

- **min_child_weight**: The minimum sum of instance weight (Hessian) needed in a child.
- **max_depth**: The maximum depth of a decision tree.
- **gamma**: The minimum loss reduction required to make a split.
- **alpha**: L1 regularization term.

However, due to hardware and time limitation, we tuned the hyperparamters one by one with 10-fold cross-validation instead tuning all 8 hyperparameters all at once.

3.6 Training and Testing Models

After having tuned the hyperparameters for each model, we set their best values to their respective model. After that we trained the model using the resampled data as training data. For three of the resampling techniques: Random Oversampling, Random Undersampling and SMOTE, we varied the resampled percentage from 10% to 100%, then choose the one that performed best based on its respective confusion matrix. Finally, the test set that we split from the beginning is used to test the performance of each models when encountering new data.

3.7 Performance Evaluation

The final step of building a model is to evaluate its performance. For this project, we use different metrics to evaluate the performance of a model: precision, recall, F1-score, PR curve, ROC curve [14]. Another commonly used metric for binary classification is *accuracy*; however accuracy does not accurately reflect the model performance when encountering an imbalanced dataset. For example, there is a dataset of 100 samples, in which only 5 samples belong to class 1. Supposedly, a model predict all 100 samples is class 0, the accuracy of the model would be 95%, which in theory is very high. However in reality, the model failed to predict all 5 samples in class 1. Due to the nature of fraud detection, we would like to correctly detect as much fraudulent cases as possible so accuracy would not be as suitable metric for our project.

Usually, ROC curve, which will be further discussed in section 4.1.3, is also a good metrics to measure the performance of a model; however, when there is a dispropotional dataset, the AUC would not be a good metric to compare different models, the PR curve would better reflect this difference. For example, there are 2 models that is trying to detect 100 fraud cases from 1 million transactions at the same threshold in a same dataset which has 100 fraud cases:
Model 1 predicted 100 transactions as fraudulent, in which only 90 is actually fraudulent.
Model 2 predicted 1000 transactions as fraudulent, in which only 90 is actually fraudulent.
In this case, model 1 would be the more preferable choice since the model 2 has a lot of false positive. At this threshold the True Positive Rate and the False Negative Rate of each model would be:
Model 1: 0.9 True Positive Rate and 0.00001 False Positive Rate
Model 2: 0.9 True Positive Rate and 0.00091 False Positive Rate
If we compare these models using AUC, the result would not be too different as they only has tiny difference in False Positive Rate ($\sim$ 0.0009). The difference between two models would be better seen when using PR curve.
Model 1: 0.9 Precision and 0.9 Recall
Model 2: 0.0833 Precision and 0.9 Recall
It is clear that model 2 is far less ideal than model 1 as there is a huge difference between the 2 models' precision score ($\sim$ 0.8167).

If the difference varies across different threshold, ROC curve would not be able to show it as well as PR curve. However, we still choose to use ROC curve as a reference and comparison to PR curve of each model.

4 Results

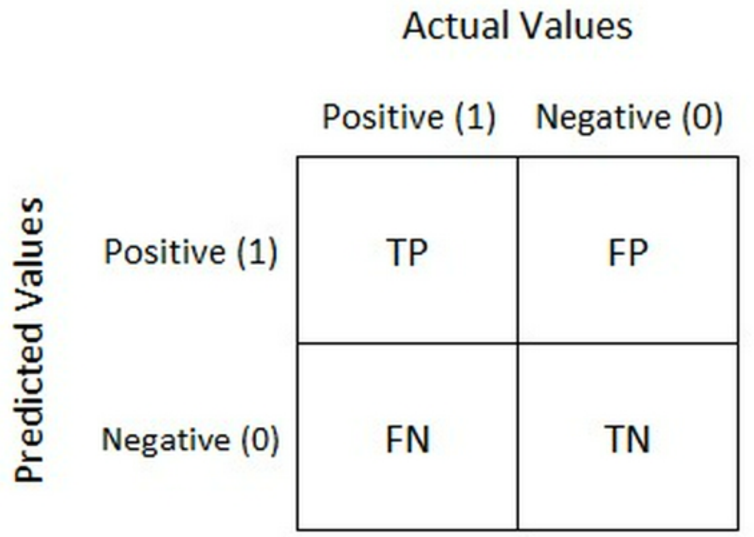
4.1 Evaluation Metrics

4.1.1 Precision, Recall and F1-score

Precision can be defined as the number of correct positive prediction over the total of positive prediction. *Recall* is the number of correct positive prediction over the total of positive ground truth. Given a confusion matrix as in Figure 9, Precision and Recall score are computed as follows:

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

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



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 9: Confusion Matrix Example

When we use both precision and recall, it is a good idea to look into *F1-score* as well since it is a function of both precision and recall. F1-score is a good metric when we look for a balance between Precision and Recall since the number of True Negative (TN) does not contribute in the calculation of F1-score, which is very suitable for skewed data that has a lot of negative sample like in this project. The formula of F1-score is as follows:

$$Precision = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

4.1.2 Precision-Recall Curve

When working with an imbalanced data, we would want to keep track of both the precision and recall of the model to make sure the model does not overfit the majority class and produce unreliable predictions. Precision-Recall (PR) curve is a useful measure of a model prediction as it shows the trade-off between precision and recall for different threshold. A model with low precision and high recall will result in a lot of positive predictions but many of them would be wrong compared to the ground truth. A model with high precision and low recall would produce few positive prediction which might leave out a lot of true positive samples but the predicted ones usually match their ground truth. Figure 10 shows and ideal PR curve with both precision and recall being 1.

In order to evaluate a model based on the PR curve, we use the Area Under the Curve (AUC), which is also known as Average Precision (AP), of the curve. The AUC of the PR curve can be calculated using integral; however we can vary the threshold by a small amount each time so that the AUC can be calculated by summing up the areas as. The AUC can be treated as a weighted sum of the precision scores. The formula to compute the AP is as follows:

$$AP = \sum_n (R_n - R_{n-1}) \times P_n$$

where P_n and R_n represent the precision and recall at threshold n^{th}.

4.1.3 Receiver Operating Characteristic Curve

Similarly to PR curve, Receiver Operating Characteristic (ROC) [3] curve illustrates the diagnostic probability of a model with varied threshold. In order to understand this metric, we must first understand the concepts of True Positive Rate (TPR) and False Positive Rate (FPR). The TPR is also

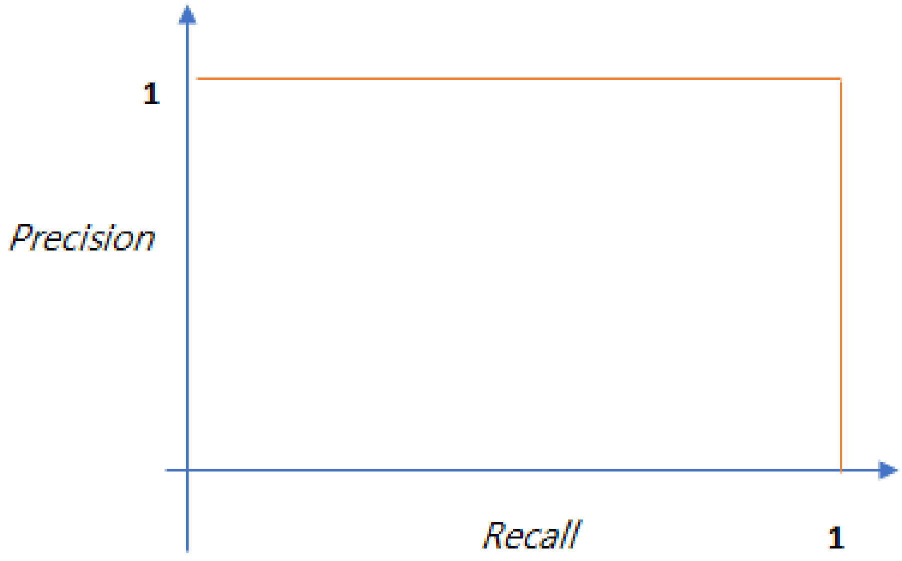


Figure 10: Ideal Precision-Recall Curve

know as Sensitivity or Recall and the FPR is also known as the inverse Specificity which is calculated as the total number of true negatives over the sum of the number of true negatives and false positives. Consider Figure 9, the FNR is computed as:

$$FPR = 1 - \frac{TN}{TN + FP}$$

4.2 Performance

4.2.1 Logistic Regression

4.2.1.1 No Resampling

After using GridsearchCV to tune the parameter for the model, the best 'C' parameter of the model is 0.1.

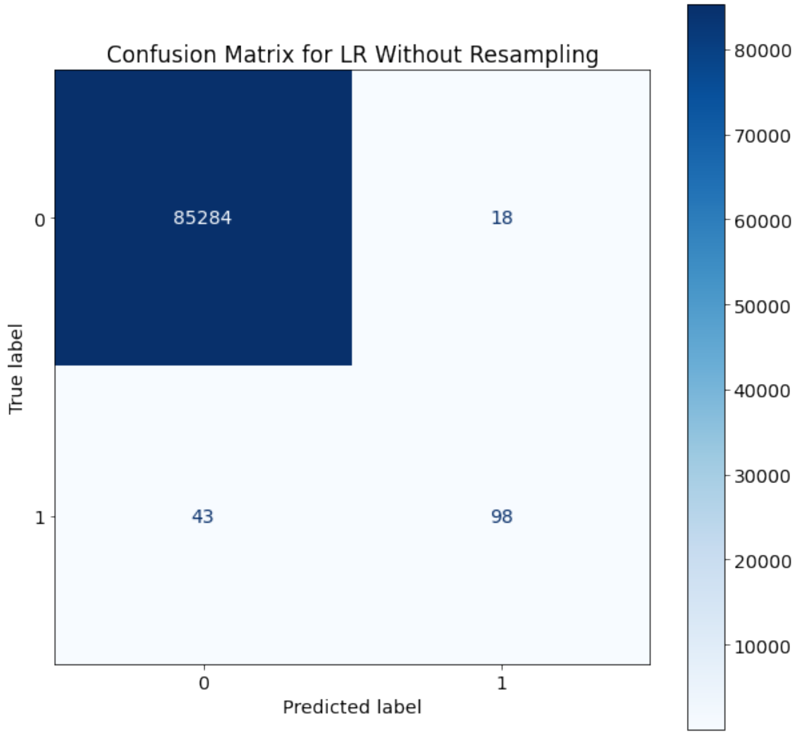


Figure 11: Confusion Matrix: LR no Resampling

Class	Precision	Recall	F1 Score
0	0.99950	0.99979	0.99964
1	0.84483	0.69504	0.76265
Avg	0.99924	0.99929	0.99925

Table 2: Classification Report: LR no Resampling

The Logistic Regression model without using any resampling techniques performed very well in classifying genuine transaction (Class 0) with the precision, recall and f1 score of 0.9995, 0.99979 and 0.99964 respectively. However, the model's performance was not good when dealing with the fraudulent transaction. Its recall and f1 score are rather low at 0.69504 and 0.76265 respectively. Figure 11 shows the confusion matrix of the Logistic Regression model with none of the resampling techniques applied. In addition, the PR curve of the model is shown in figure 12 with the AP of 0.70, which is acceptable but not too high.

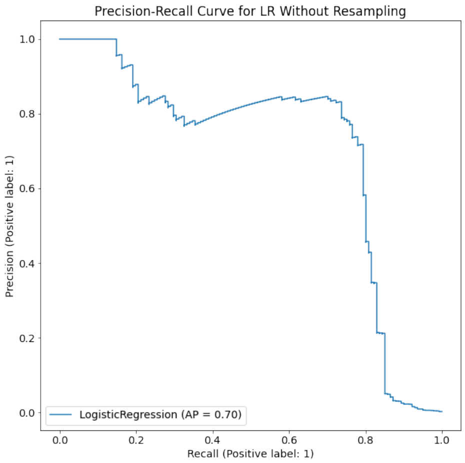


Figure 12: PR Curve: LR no Resampling

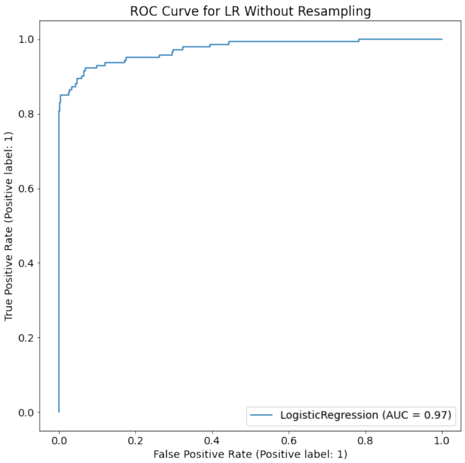


Figure 13: ROC Curve: LR no Resampling

4.2.1.2 Random Oversampling

As we varied the percentage of the number of observation to over sampled, we chose the data that gave us the highest correct number of fraudulent cases (TP), followed by the lowest misclassified positive classes (FN) then the lowest misclassified genuine transactions (FP). We noticed that when the data is oversampled using Random Oversampling technique at 70% (i.e. the final ratio of fraudulent over genuine transactions is 0.7).

Figure 14 gives us an overview look about all the confusion matrices with different ratios between two classes. The number of correctly predicted fraudulent behaviors when the data is oversampled 70% and 100% is the same; however 70% resampled data has less misclassified genuine cases so this dataset will be used to train the final model. A clearer view of its confusion matrix is shown in figure 15.

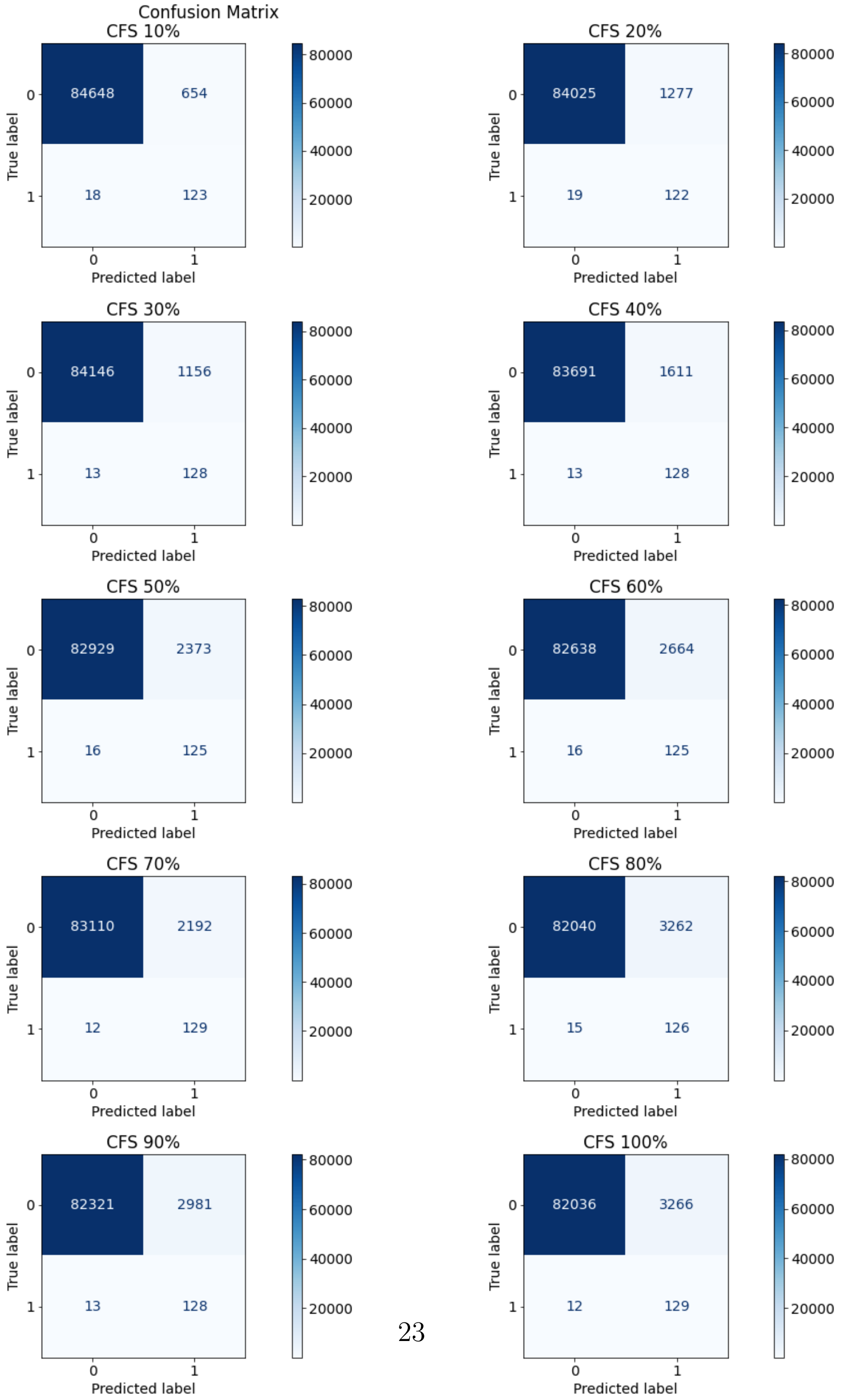


Figure 14: Confusion Matrix: LR Random Oversampling

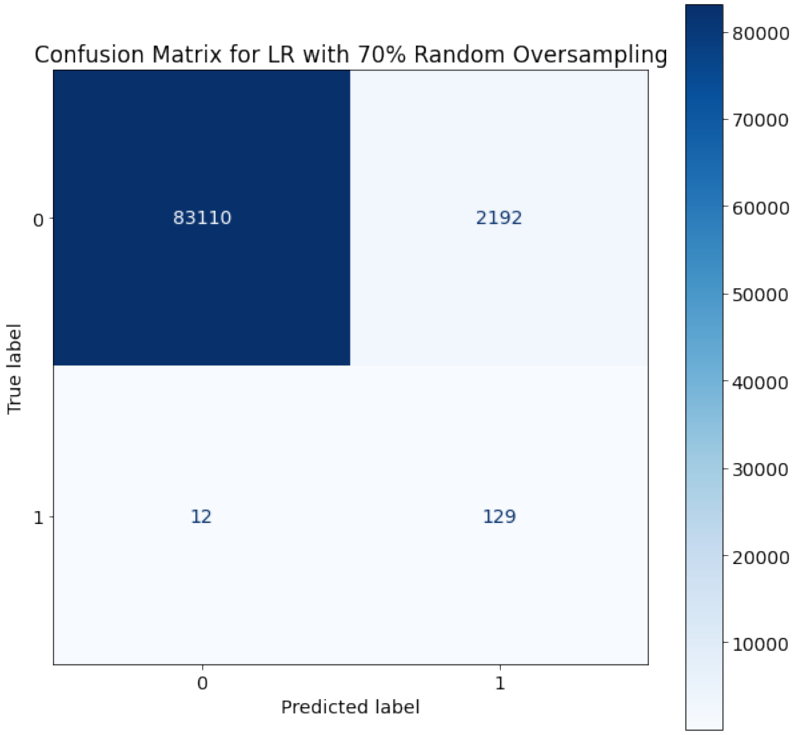


Figure 15: Confusion Matrix: LR 70% Random Oversampling

Class	Precision	Recall	F1 Score
0	0.99986	0.97430	0.98691
1	0.05558	0.91489	0.10479
Avg	0.99830	0.97421	0.98546

Table 3: Classification Report: LR 70% Random Oversampled

After using GridsearchCV to tune the parameter for the model, the best 'C' parameter of the model is 0.5.

By applying Random Oversampling to the data, we managed to increase number of correctly detected fraud compared to not using any resampling technique. However, this leads to misclassification of non-fraudulent cases which heavily impacted the model precision when dealing when fraudulent transactions. Although the model recall score was very high at 0.91489, its precision and f1 score is only 0.05558 and 0.10479 respectively, which is extremely low for a predictive model. As shown in figure 16, the AP score of the model is 0.71 which is slightly higher than the previous model where no resampling was applied. The AP score is still in an acceptable threshold; however it would be more preferable if the we can achieve a higher score as we are dealing with a sensitive issue of detecting fraud.

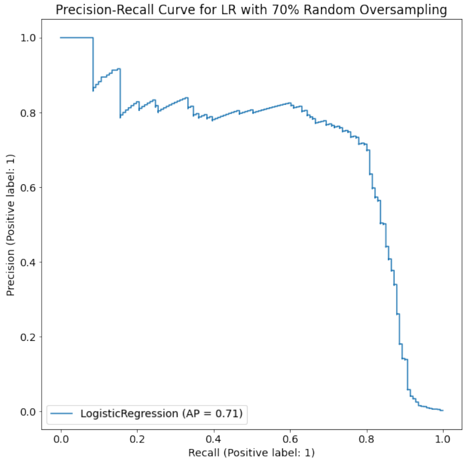


Figure 16: PR Curve: LR 70% Random Oversampling

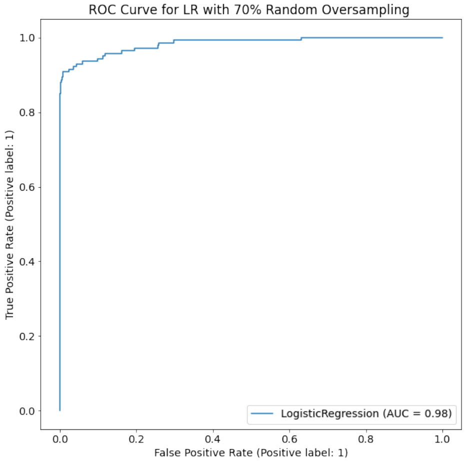


Figure 17: ROC Curve: LR 70% Random Oversampling

4.2.1.3 Random Undersampling

Similar to Random Oversampling, we also varied the percentage of the resampled date. As can be seen from figure 18, 70% undersampled would be a suitable choice for this model. However, if we look at figure 19 which gives us the overview of all the PR curves, the AP of the 70% undersampled dataset is lower than most others. With Random Undersampling technique, we decided to train the model using the dataset which has been undersampled 100% (i.e. two classes having the same number of samples) as the model has the highest correct prediction of fraud cases as well as the highest AP on the test set. The detailed confusion matrix and PR curve is depicts in figure 20 and figure 21.

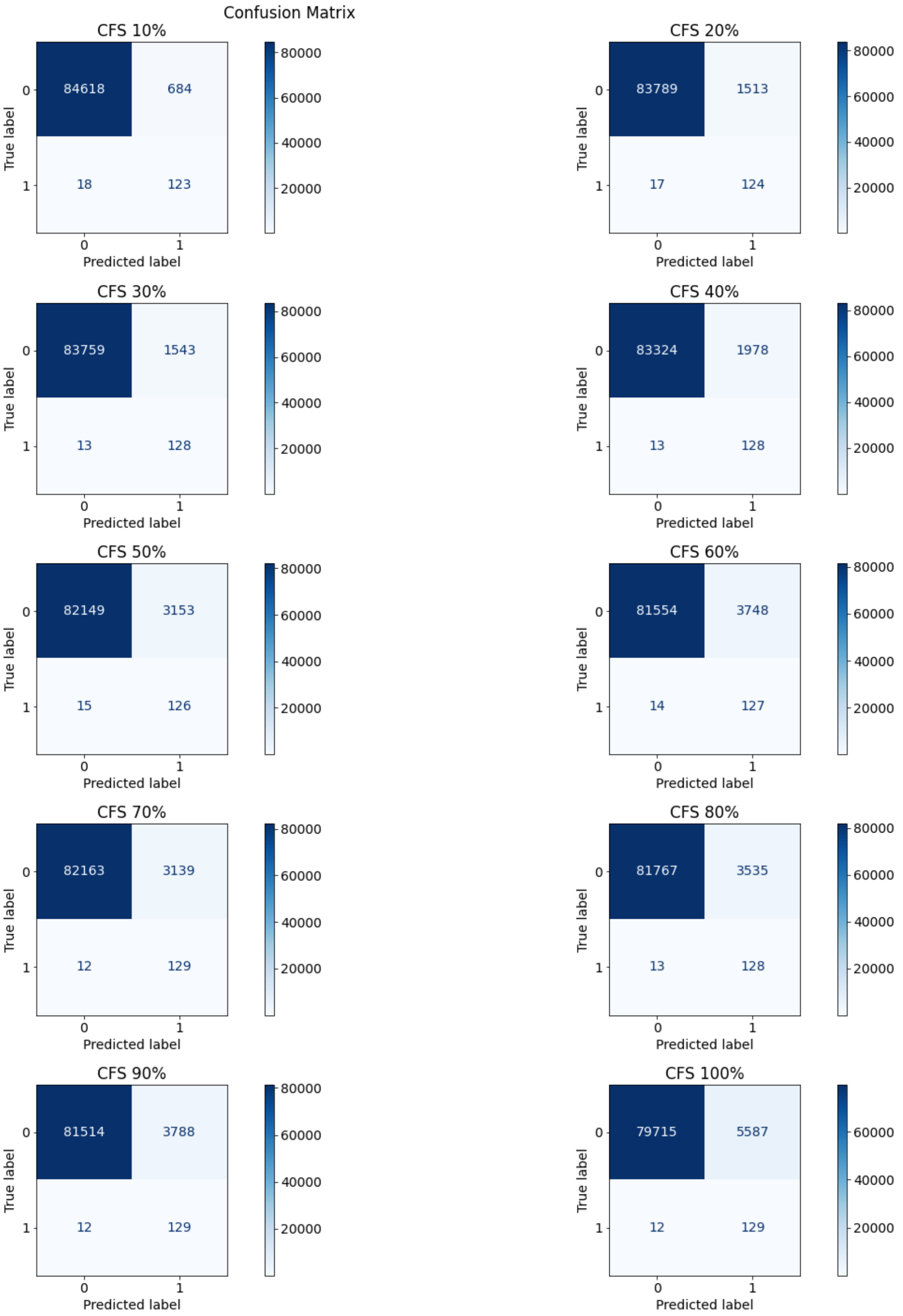


Figure 18: Confusion Matrix: LR Random Undersampling

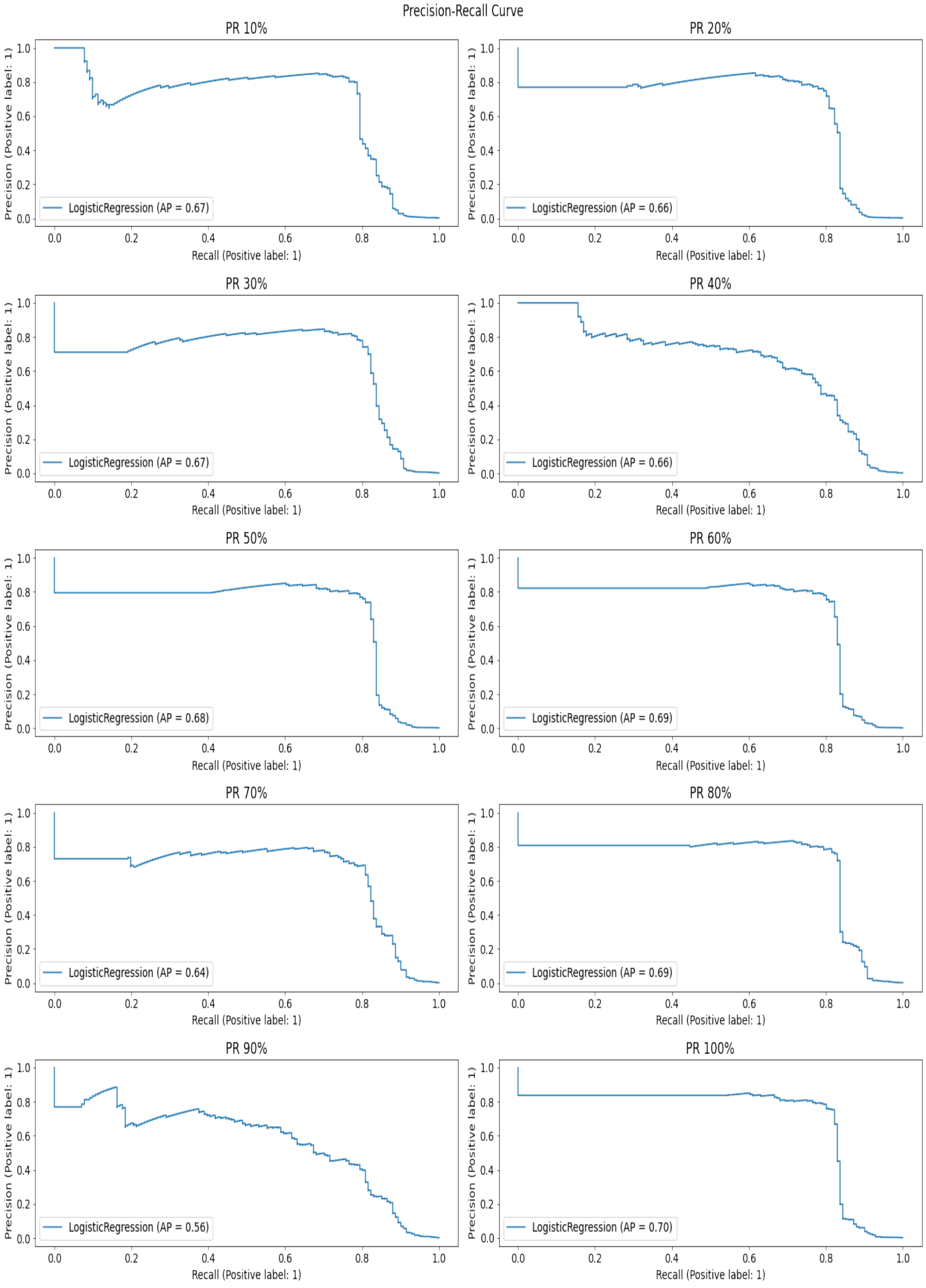


Figure 19: PR Curve: LR Random Undersampling

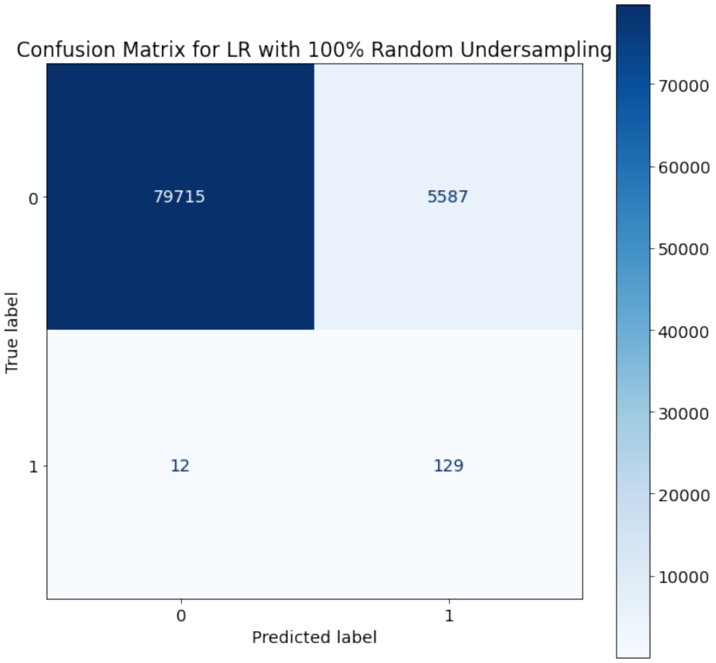


Figure 20: Confusion Matrix: LR Random Undersampling

Class	Precision	Recall	F1 Score
0	0.99985	0.93450	0.96607
1	0.02257	0.91489	0.04405
Avg	0.99824	0.93447	0.96455

Table 4: Classification Report: LR 100% Random Undersampled

After using GridsearchCV to tune the parameter for the model, the best ’C’ parameter of the model is 0.9.

As expected, the model still perform nicely in classifying the negative class with high precision (0.99986) and recall(0.9743); however, in the case of positive class, the model precision is only 0.02257. As a result, lower the f1 score down to 0.04405, which is very poor. Even though, it has a recall score over 0.9, the precision score should not be neglected. With such difference between precision and recall score, the AP score of the model only reached 0.7, the same as when no resampling technique was applied.

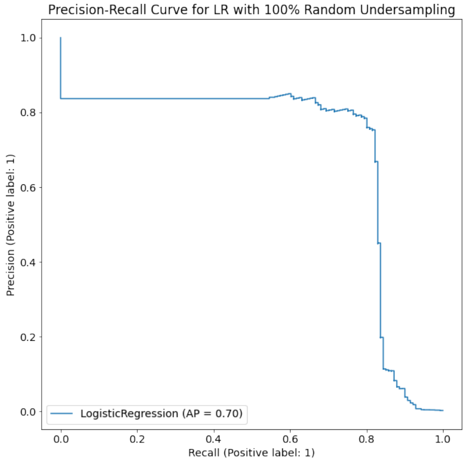


Figure 21: PR Curve: LR 100% Random Undersampling

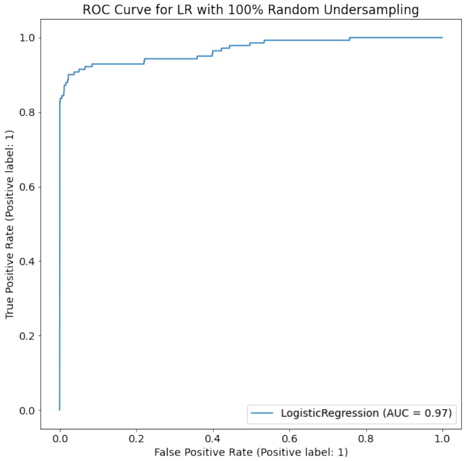


Figure 22: ROC Curve: LR 100% Random Undersampling

4.2.1.4 SMOTE

For SMOTE, the model performed best when we synthesize new training example of the positive class so that the ratio is 0.7 (70%). Looking at figure 23, most of the resampled datasets resulted in the same number of fraudulent prediction (127 True Positive and 14 False Negative). Comparing the 70% SMOTE-ed dataset to the 80% one, we noticed the number of incorrectly predicted genuine transactions are not to different from each other. In addition, the 70% one yielded a higher PR AUC (Area Under the Curve of the PR Curve) which is shown in figure 24; therefore, we decided to use the dataset that has been resampled so that the number of samples of the fraudulent class is 70% of the genuine class.

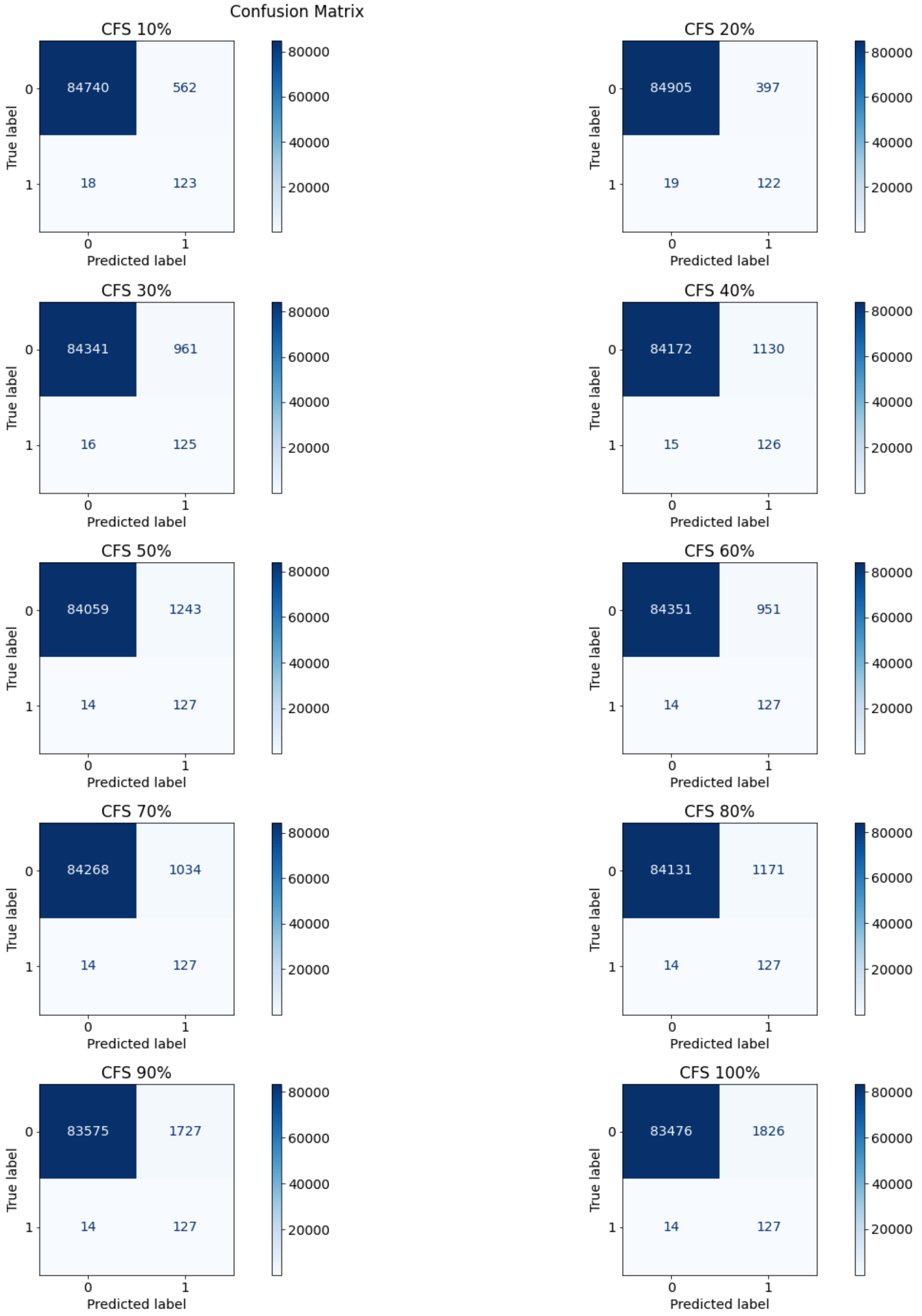


Figure 23: Confusion Matrix: LR SMOTE

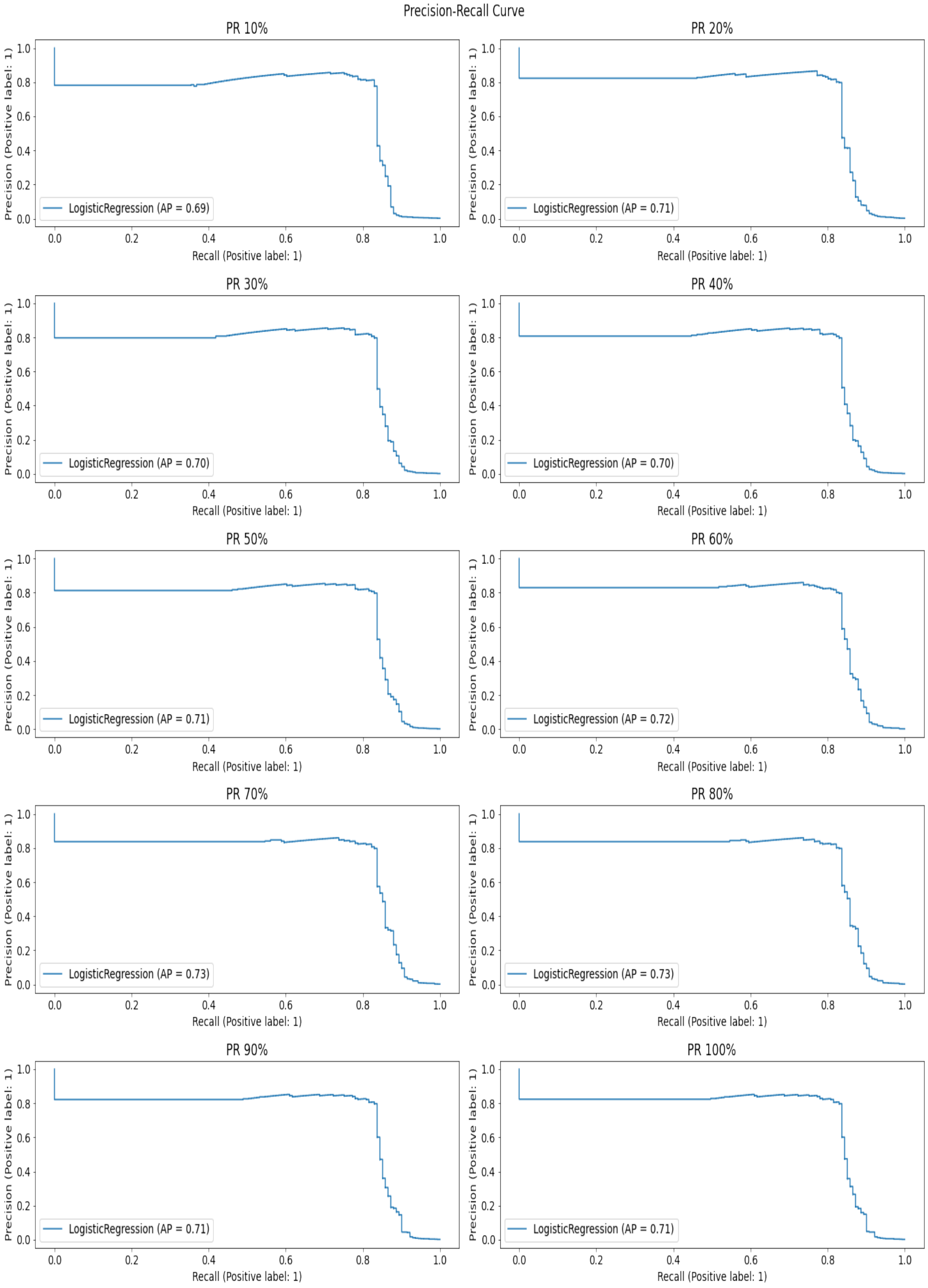


Figure 24: PR Curve: LR SMOTE

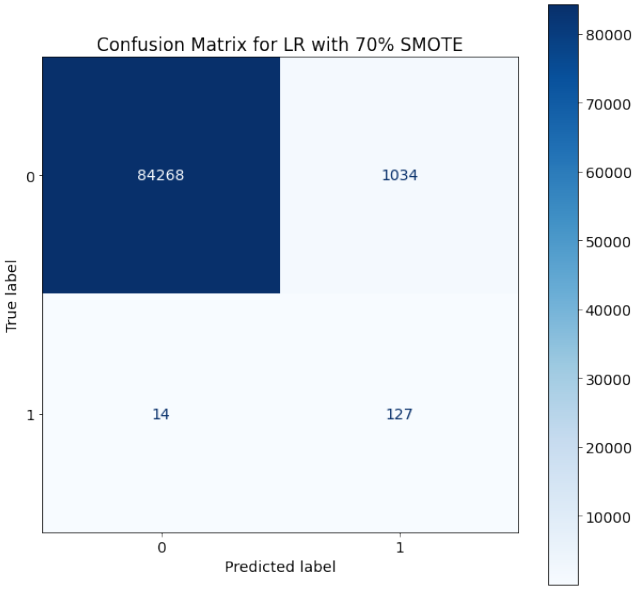


Figure 25: Confusion Matrix: LR 70% SMOTE

Class	Precision	Recall	F1 Score
0	0.99983	0.98788	0.99382
1	0.10939	0.90071	0.19508
Avg	0.99836	0.98773	0.99250

Table 5: Classification Report: LR 70% SMOTE

After using GridsearchCV to tune the parameter for the model, the best 'C' parameter of the model is 0.1.

Table 5 gives us the classification report of the model, including precision, recall and f1-score of both classes. It can be seen that by using SMOTE, the logistic regression model's performance did not improve as its f1 score when detecting fraud is only 0.19508 and its AP score is 0.73 as in figure 26.

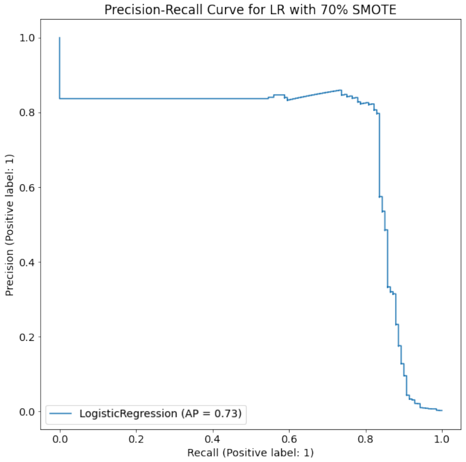


Figure 26: PR Curve: LR 70% SMOTE

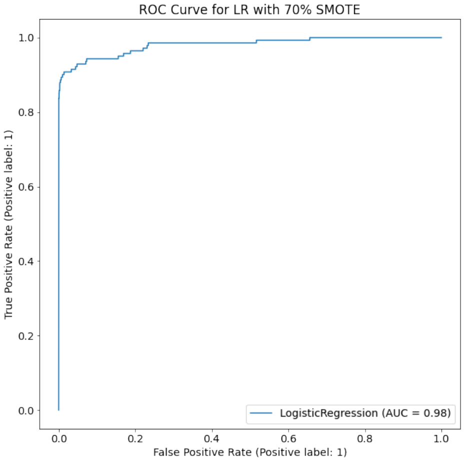


Figure 27: ROC Curve: LR 70% SMOTE

4.2.1.5 Tomek Links Removal

After using GridsearchCV to tune the parameter for the model, the best 'C' parameter of the model is 0.4.

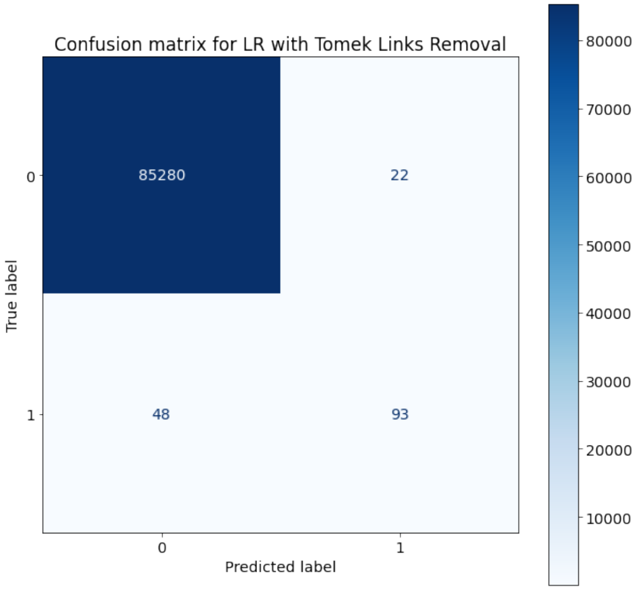


Figure 28: Confusion Matrix: LR Tomek Links Removal

Class	Precision	Recall	F1 Score
0	0.99944	0.99974	0.99959
1	0.80870	0.65957	0.72656
Avg	0.99912	0.99918	0.99914

Table 6: Classification Report: LR 70% SMOTE

Similar to previous models, this model do well when handling negative class but fail to perform when dealing with positive one. However, comparing to other resampling techniques, the precision, recall and f1 score of the model with respect to the positive class is 0.80870, 0.65957 and 0.72656 respectively. In term of AP score, the model only achive the score of 0.64, which is rather low for a predictive model.

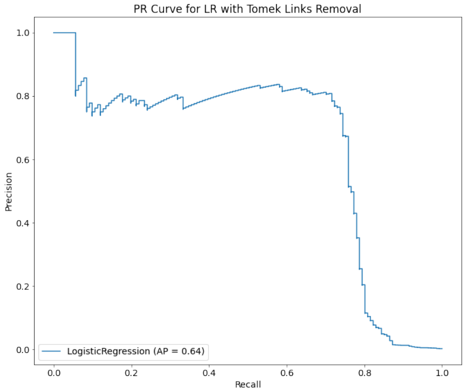


Figure 29: PR Curve: LR Tomek Links Removal

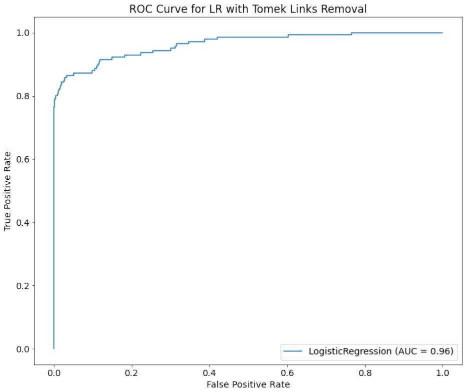


Figure 30: ROC Curve: LR Tomek Links Removal

4.2.1.6 Hybrid Resampling

As hybrid resampling technique is the combination of SMOTE and Tomek Links Removal, we decided to apply Tomek Links Removal to the best version of SMOTE-ed dataset (70%).

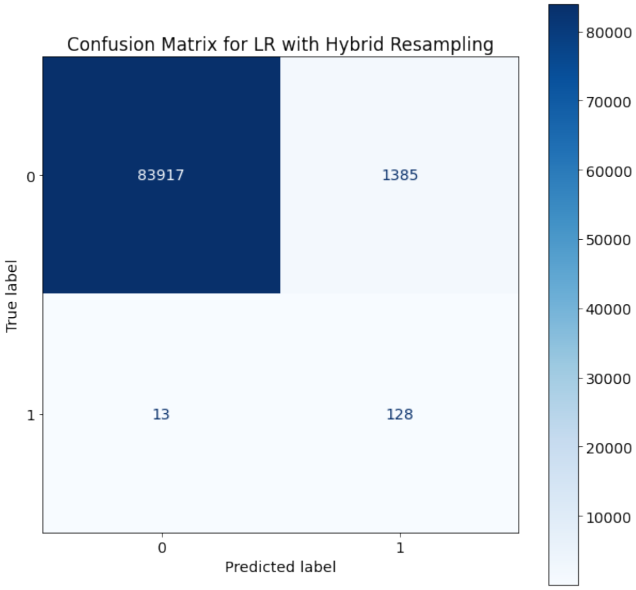


Figure 31: Confusion Matrix: LR Hybrid Resampling

Class	Precision	Recall	F1 Score
0	0.99983	0.97844	0.98902
1	0.06460	0.90071	0.12055
Avg	0.99829	0.97831	0.98759

Table 7: Classification Report: LR Hybrid Resampling

After using GridsearchCV to tune the parameter for the model, the best 'C' parameter of the model is 0.7.

With this technique, the model did not improve its performance much as it only scored a f1 score of 0.12055 as in table 7 and an AP score of 0.701 as in figure 32.

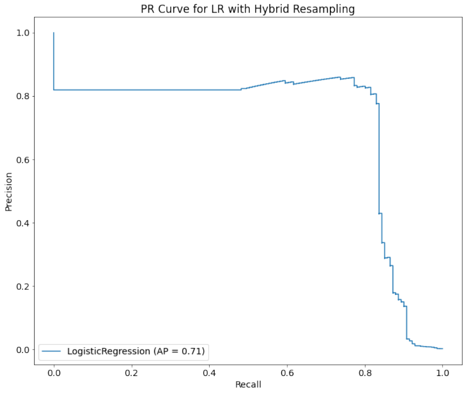


Figure 32: PR Curve: LR Hybrid Resampling

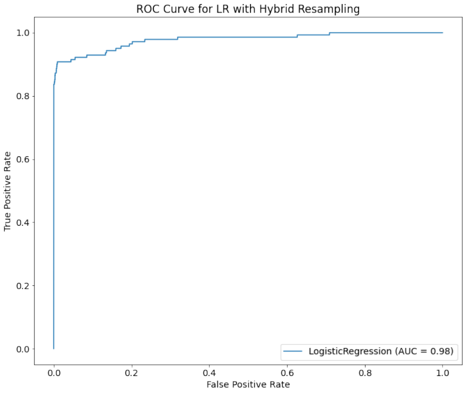


Figure 33: ROC Curve: LR Hybrid Resampling

4.2.2 Random Forest

4.2.2.1 No Resampling

After using GridsearchCV to tune the parameter for the model, the best 'n_estimators' parameter of the model is 250.

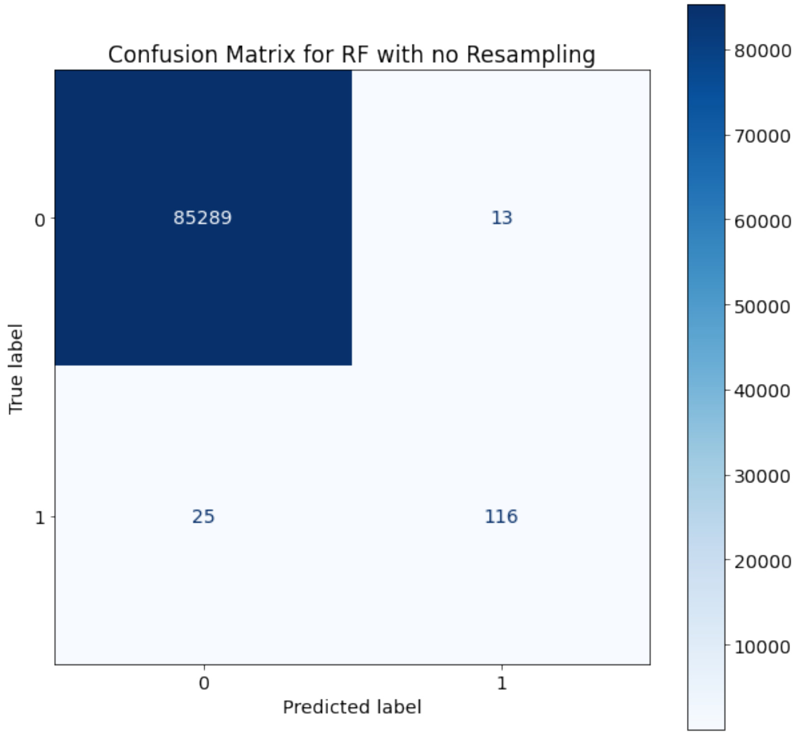


Figure 34: Confusion Matrix: RF No Resampling

Class	Precision	Recall	F1 Score
0	0.99970	0.99984	0.99977
1	0.89147	0.81560	0.85185
Avg	0.99952	0.99953	0.99952

Table 8: Classification Report: RF no Resampling

As can be seen from tabel 8, Random Forest model performed better than Logistic Regression model even when there is no resampling technique applied. The model performed well when predicting genuine transactions as all metrics scores are very high (> 0.999). In contrast to Logistic Regression model, Random Forest handled fraudulent transactions rather well as its precision, recall and f1 score are 0.8914, 0.81560 and 0.85185. In addition, comparing to to previous models, the AP score of this model is much higher at 0.86. This AP score indicates that this model is usable to detect credit card fraud.

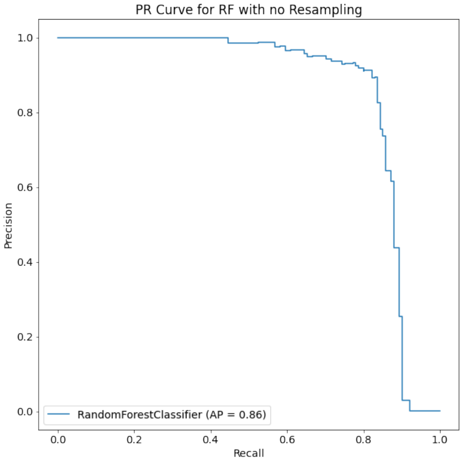


Figure 35: PR Curve: RF no Resampling

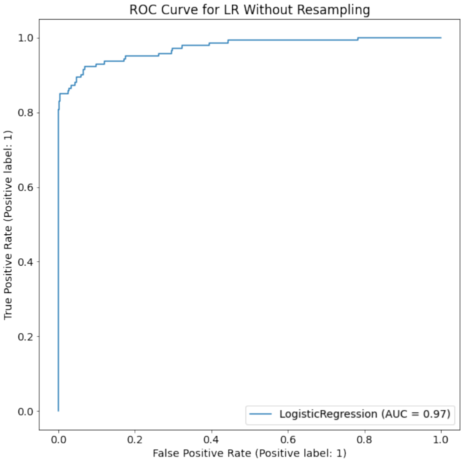


Figure 36: ROC Curve: RF no Resampling

4.2.2.2 Random Oversampling

As mentioned in section 3.6, we varied the number of samples in the minority class to choose the best percentage to use as our training dataset. Looking at figure 37 and figure 38, we noticed that most of the dataset produced the same result with 117 TP, 24 FN, 11 FP. Therefore, we can choose any dataset to train our model. We decided to go with the 70% randomly oversampled dataset. A more detailed view of its confusion matrix and PR curve is showed in figure 39 and figure 40 respectively.

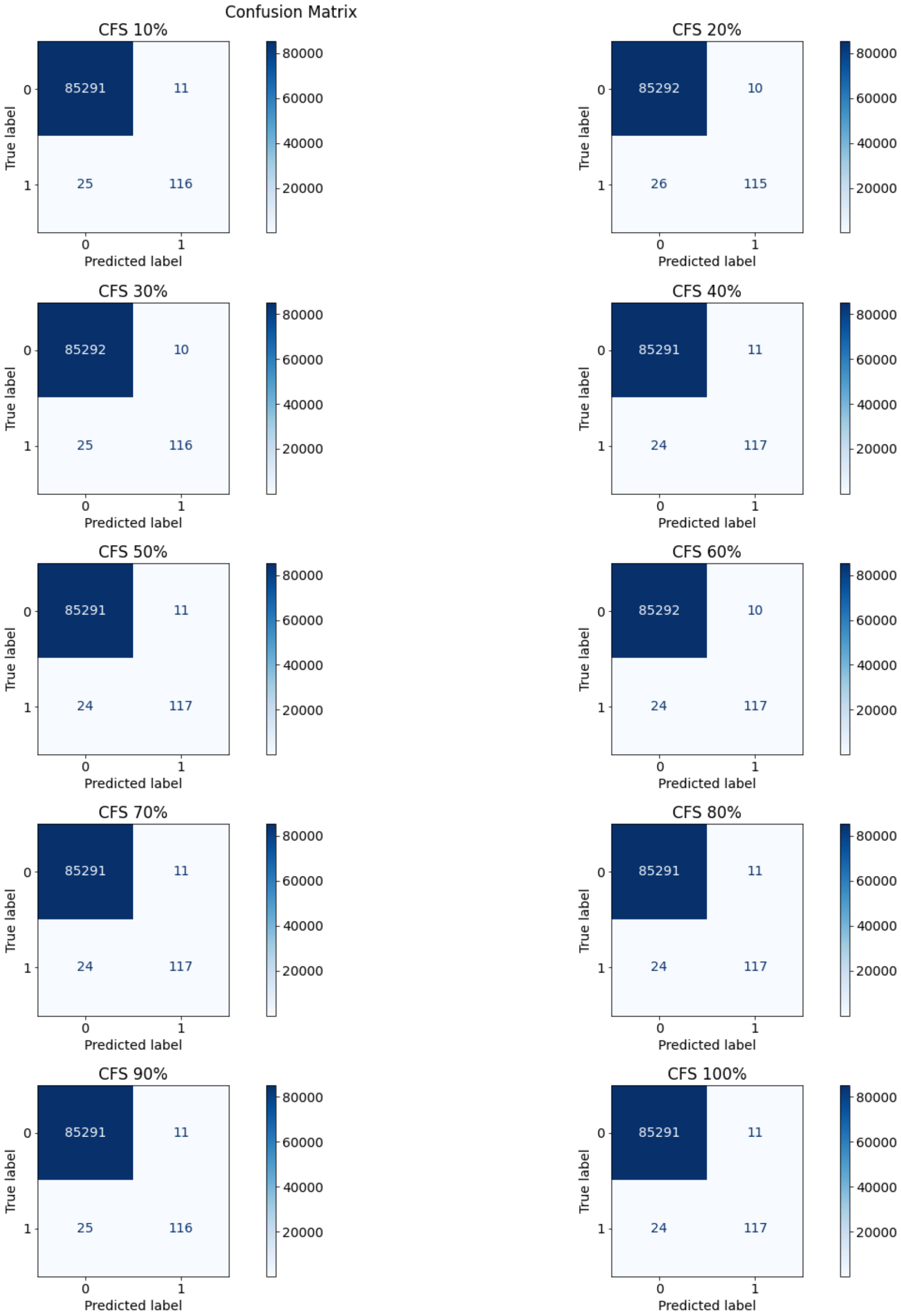


Figure 37: Confusion Matrix: RF Random Oversampling

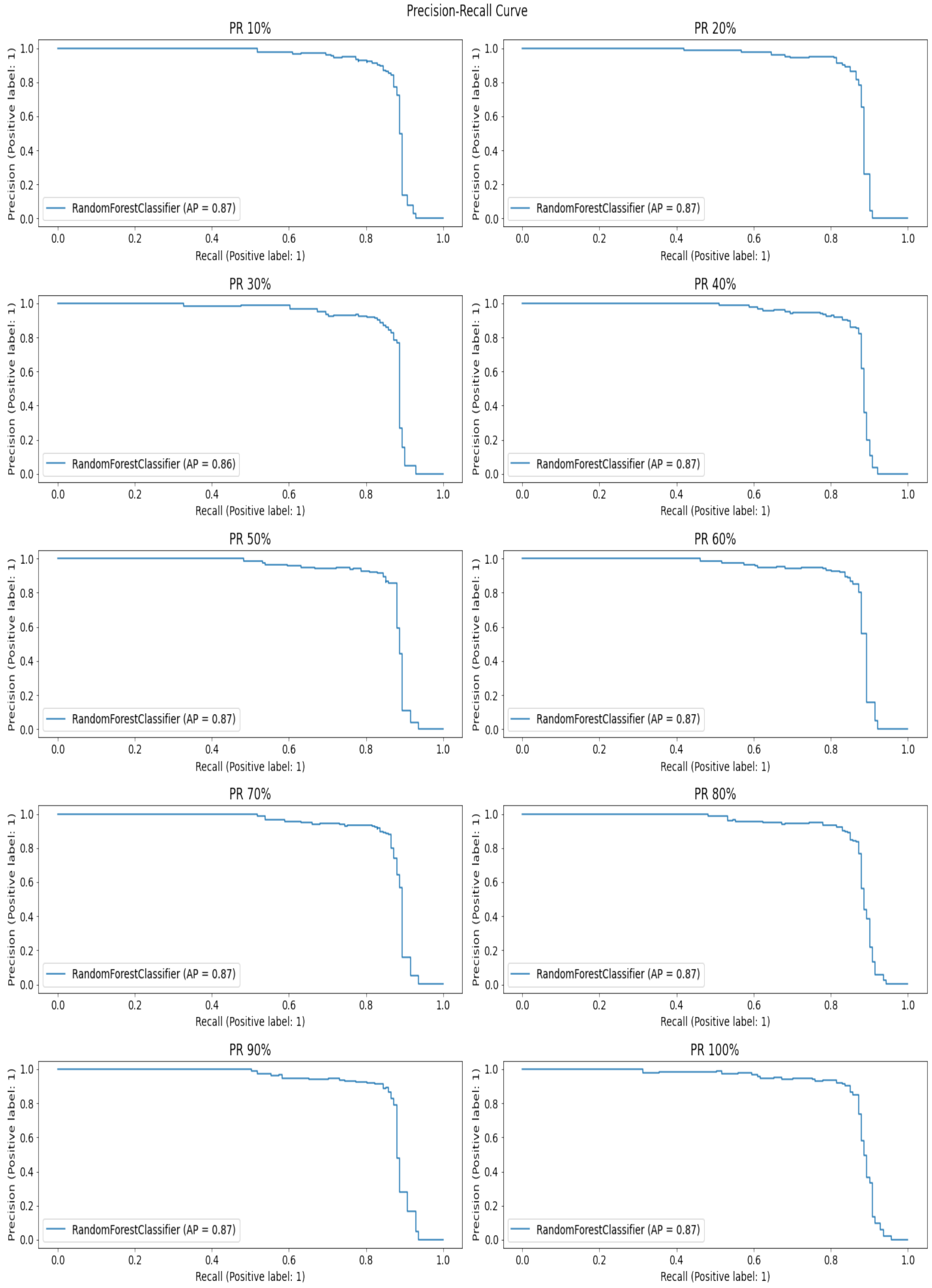


Figure 38: PR Curve: RF Random Oversampling

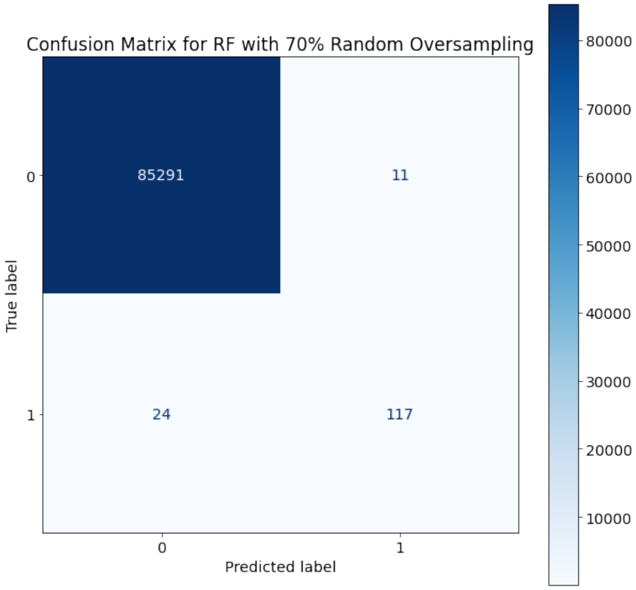


Figure 39: Confusion Matrix: RF 70% Random Oversampling

Class	Precision	Recall	F1 Score
0	0.99972	0.99987	0.99979
1	0.91406	0.82979	0.86989
Avg	0.99958	0.99959	0.99958

Table 9: Classification Report: RF no Resampling

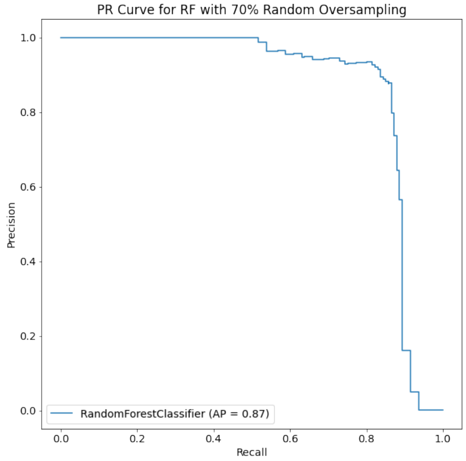


Figure 40: PR Curve: RF 70% Random Oversampling

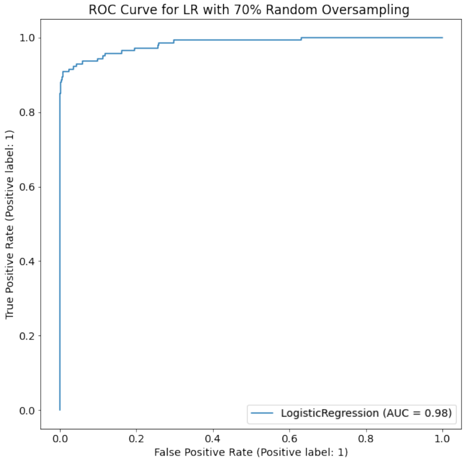


Figure 41: ROC Curve: RF 70% Random Oversampling

4.2.2.3 Random Undersampling

4.2.2.4 SMOTE

4.2.2.5 Tomek Links Removal

4.2.2.6 Hybrid Resampling

4.2.3 XGBoost

4.2.3.1 No Resampling

4.2.3.2 Random Oversampling

4.2.3.3 Random Undersampling

4.2.3.4 SMOTE

4.2.3.5 Tomek Links Removal

4.2.3.6 Hybrid Resampling

4.3 Result Summary

4.4 Discussion

5 Conclusion

6 References

[1] Aleskerov, E., Freisleben, B., & Rao, B. CARDWATCH: a neural network based database mining system for credit card fraud detection. Proceedings Of The IEEE/IAFE 1997 Computational Intelligence For Financial Engineering (Cifer). doi: 10.1109/cifer.1997.618940

[2] Awoyemi, J., Adetunmbi, A., & Oluwadare, S. (2017). Credit card fraud detection using machine learning techniques: A comparative analysis. 2017 International Conference On Computing Networking And Informatics (ICCNI). doi: 10.1109/iccni.2017.8123782

[3] Bradley, A. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition, 30(7), 1145-1159. doi: 10.1016/s0031-3203(96)00142-2

[4] Breiman, L. (2001). Random Forest. Machine Learning, 45(1), 5-32. doi: 10.1023/a:1010933404324

[5] Cambridge Spark. (2017). Hyperparameter tuning in XGBoost

[6] Carcillo, F., Dal Pozzolo, A., Le Borgne, Y., Caelen, O., Mazzer, Y., & Bontempi, G. (2018). SCARFF : A scalable framework for streaming credit card fraud detection with spark. Information Fusion, 41, 182-194. doi: 10.1016/j.inffus.2017.09.005

[7] Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal Of Artificial Intelligence Research, 16, 321-357. doi: 10.1613/jair.953

[8] Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings Of The 22Nd ACM SIGKDD International Conference On Knowledge Discovery And Data Mining. doi: 10.1145/2939672.2939785

[9] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics, 21(1). doi: 10.1186/s12864-019-6413-7

[10] Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2015). Credit card fraud detection and concept-drift adaptation with

delayed supervised information. 2015 International Joint Conference On Neural Networks (IJCNN). doi: 10.1109/ijcnn.2015.7280527

[11] Dal Pozzolo, A., Caelen, O., Le Borgne, Y., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. Expert Systems With Applications, 41(10), 4915-4928. doi: 10.1016/j.eswa.2014.02.026

[12] Elhassan, A., Aljourf, M., Al-Mohanna, F., & Shoukri, M. (2016). Classification of Imbalance Data using Tomek Link (T-Link) Combined with Random Under-sampling (RUS) as a Data Reduction Method. Global Journal Of Technology And Optimization, 01(S1). doi: 10.4172/2229-8711.s1111

[13] Gandhi, R. (2018). Introduction to Machine Learning Algorithms: Logistic Regression

[14] Hossin, M., & Sulaiman, M. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. International Journal Of Data Mining & Knowledge Management Process, 5(2), 01-11. doi: 10.5121/ijdkp.2015.5201

[15] Jaadi, Z. (2019). When and Why to Standardize Your Data?

[16] Peng, C., Lee, K., & Ingersoll, G. (2002). An Introduction to Logistic Regression Analysis and Reporting. The Journal Of Educational Research, 96(1), 3-14. doi: 10.1080/00220670209598786

[17] Restrepo, M. (2018). Doing XGBoost hyper-parameter tuning the smart way — Part 1 of 2

[18] Rocca, J. (2019). Ensemble methods: bagging, boosting and stacking

[19] Srivastava, A., Kundu, A., Sural, S., & Majumdar, A. (2008). Credit Card Fraud Detection Using Hidden Markov Model. IEEE Transactions On Dependable And Secure Computing, 5(1), 37-48. doi: 10.1109/tdsc.2007.70228

[20] Statistics Odds & Ends (2020). What is balanced accuracy?

[21] Walimbe, R. (2017). Handling imbalanced dataset in supervised learning using family of SMOTE algorithm

[22] Wheeler, R., & Aitken, S. (2000). Multiple algorithms for fraud detection. Knowledge-Based Systems, 13(2-3), 93-99. doi: 10.1016/s0950-7051(00)00050-2

[23] XGBoost Developers. Notes on Parameter Tuning. XGBoost Documentation