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 Department
Vietnam France University

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

in

Information and Communication Technology

Hanoi, July 2021

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 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, TRAN Thanh Long, and NGO Le Giang. 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 it is now a necessary tool for most businesses, corporations, and government agencies to boost their efficiency in worldwide trade. 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. In recent years, as credit card transactions have become one of the most popular payment methods, fraudulent activities have also risen rapidly.

The solution to this problem can be divided into two categories: prevention, which involves preventing the fraudulent transaction from happening, and detection, which spots the fraudulent transaction after it has occurred so that appropriate action can be taken. In terms of prevention, there are already some common technologies 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 to take timely actions. The process of determining whether a credit card transaction is authentic or not is known as credit card fraud detection. Due to the 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 build 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. More specifically, this project focuses on handling a credit card transaction dataset to build a model to detect fraudulent transactions 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 with 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 Minority 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 transaction data, Big Data technology holds advantages over other conventional systems 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 customer's spending habit is analyzed 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 multiple researches carried out on different aspects of fraud detection such as detection time, improving performance, different techniques, and customers' behaviors, there is very little research 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 models 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 contains a total of 284,807 transactions, of which only 492 are fraudulent. Therefore, 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.

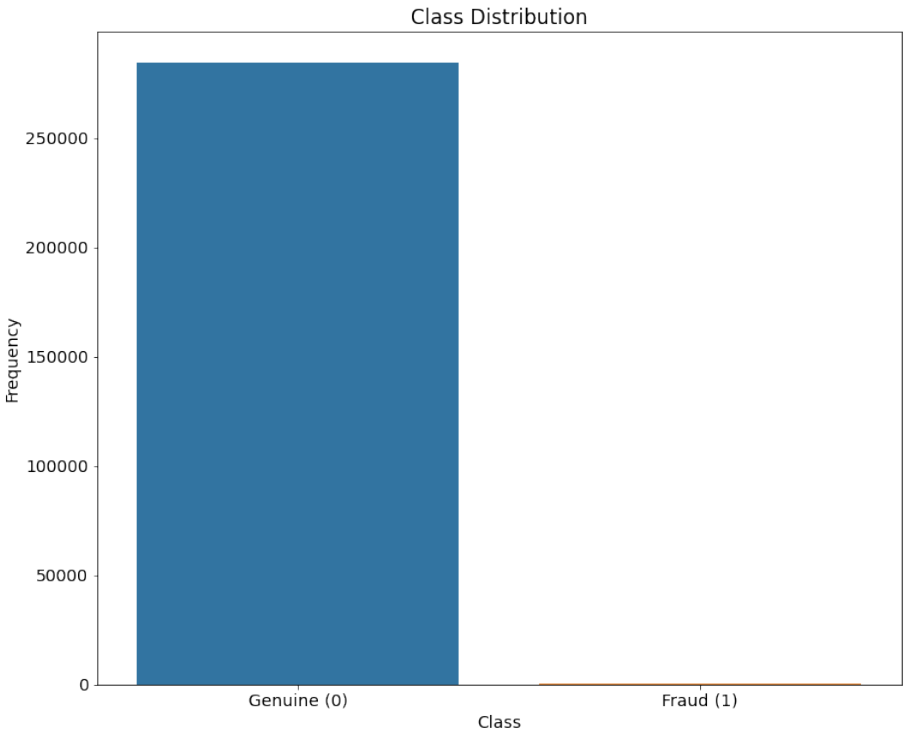

Figure 1: Dataset Class Distribution

The dataset only contains numerical values as a result of Principal Com-

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

ponents Analysis (PCA) transformation. Due to confidentiality issue, most of the original attributes was not revealed. There is a total of 30 features, 28 of which were 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 of 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 Standardization

It is a common requirement for machine learning techniques models that the data is normalized before training. A dataset that 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 Logistic Regression and Tree-based models 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 a training set (70%) and a 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 and used 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. Therefore, if we train our model directly on this dataset, the outcome will likely be biased towards 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 undersampling 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 significantly 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

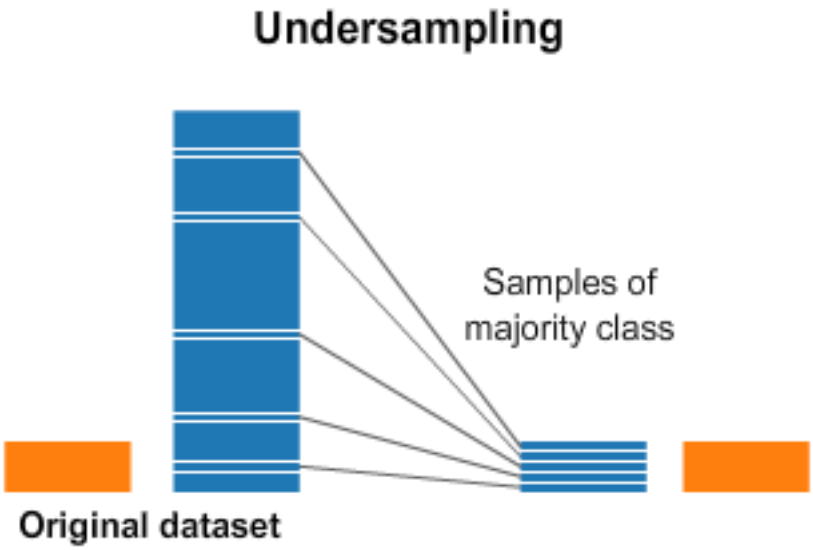

Figure 2: Undersampling Method

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

3.3.1 Random Oversampling

In Random Oversampling, random samples of the minority class are replicated to make the dataset balanced. However, there is a high possibility 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 are removed from that dataset to balance the number of data between two classes. Unlike Random Oversampling, this method is less prone to overfitting; however,helpful information may be lost during the process of elimination, , resulting in inaccurate predictions by the models.

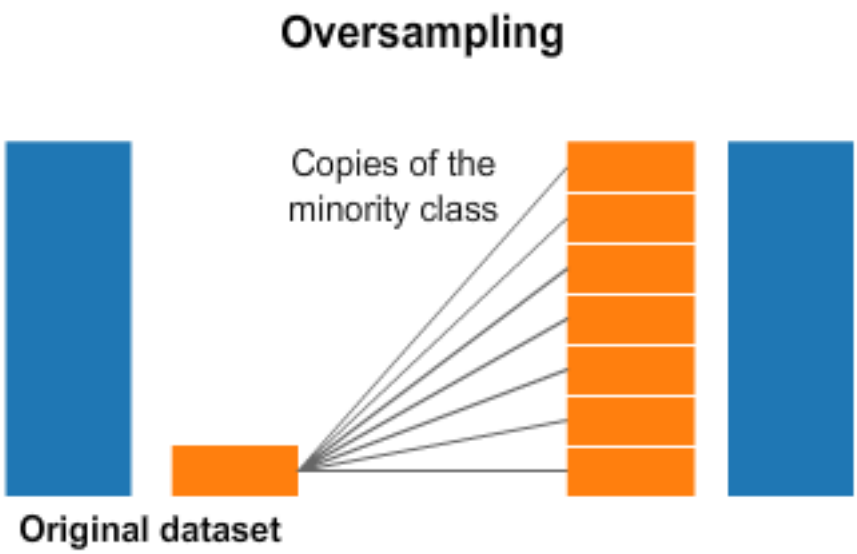

Figure 3: Oversampling Method

3.3.3 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is a popular oversampling technique developed by Chawla, Bowyer, Hall, and Kegelmeyer [7]. Instead of duplicating existing data like in Random Oversampling, SMOTE creates new data samples by interpolating between nearest minority samples. The number of chosen nearest neighbors would depend on the amount of oversampling required. By creating new data, this technique 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 is no 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 with these links, we will be able to perform undersampling for the dataset and make the boundaries between the two classes clearer.

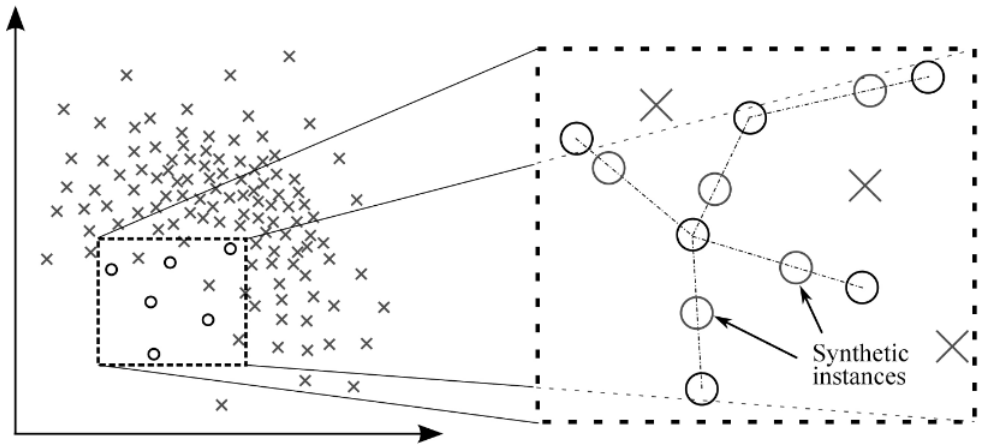

Figure 4: Synthesizing data using SMOTE [21]

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 algorithms for 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 value possible. However, since the output of Logistic Regression is a probability, the output must be in the range [0, 1]. To tackle this problem, Logistic Regression uses the 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 represents the likelihood that the example belongs 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 basics 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 represents the outcome of the test and leaf nodes represent the outcome. The decision tree breaks a training set down into several subsamples. Figure 6 shows an example of a decision tree that

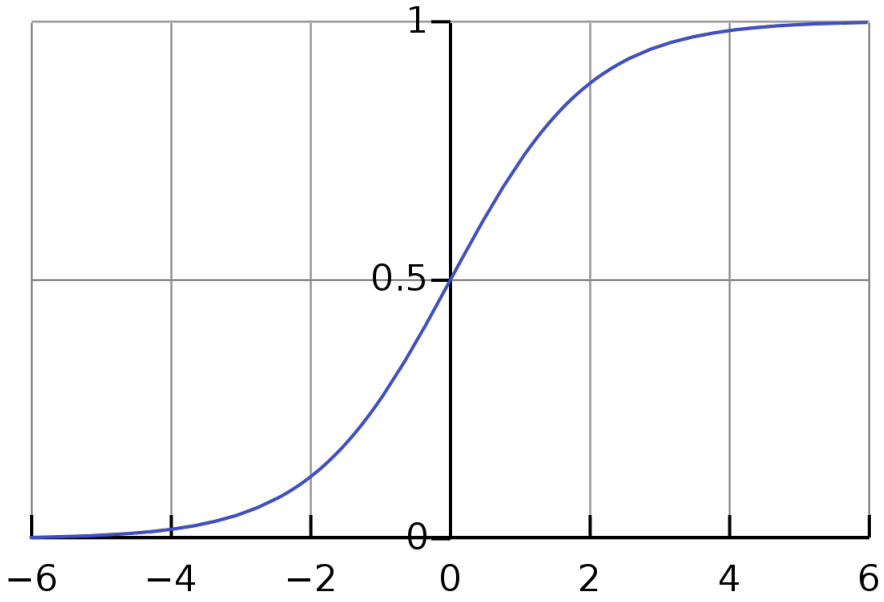

Figure 5: Sigmoid Graph [13]

tries to decide whether one should stay in or go out. It starts with a test to see if one has any work left to do. If it is true that one has work to do, the outcome will be to stay in. If the answer is no, it will perform the next test where it checks 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 to go to the beach and go running, respectively. If the weather is rainy, the tree will perform one last test: check if one's friends are busy. If they are free, ones can go to the movies with them; otherwise, one 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 simple to grasp and perform well in particular datasets, their greedy approach causes them to have a high variance. This is because 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 poses the possibility of overfitting the data, which leads to poor performance on unseen data in test sets.

Random Forest uses bootstrap to reduce overfitting. Bootstrapping is

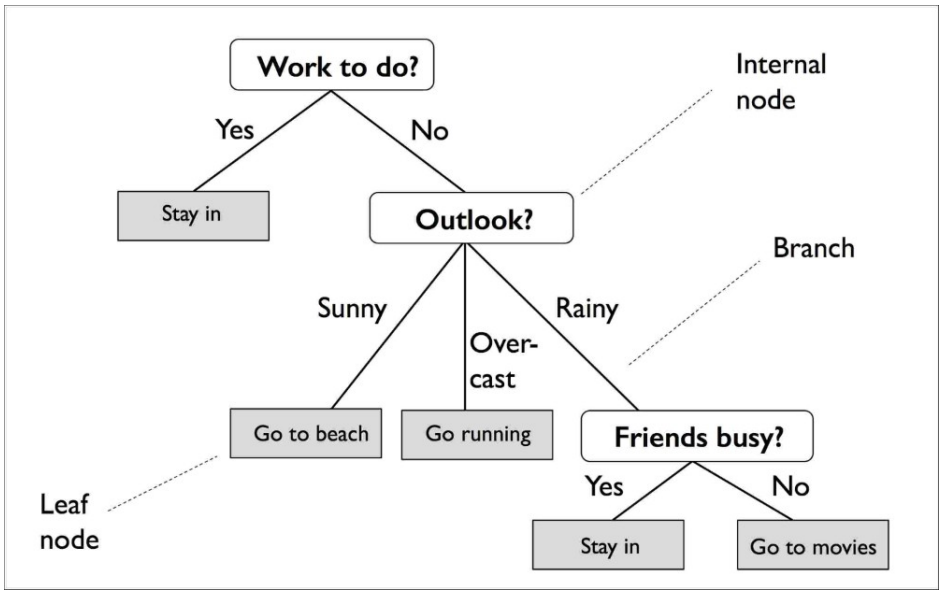

Figure 6: Decision Tree Example

the process of randomly picking and replacing training data. Random Forest makes use of bootstrapping to train each decision tree on a different portion 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 trained 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 advanced

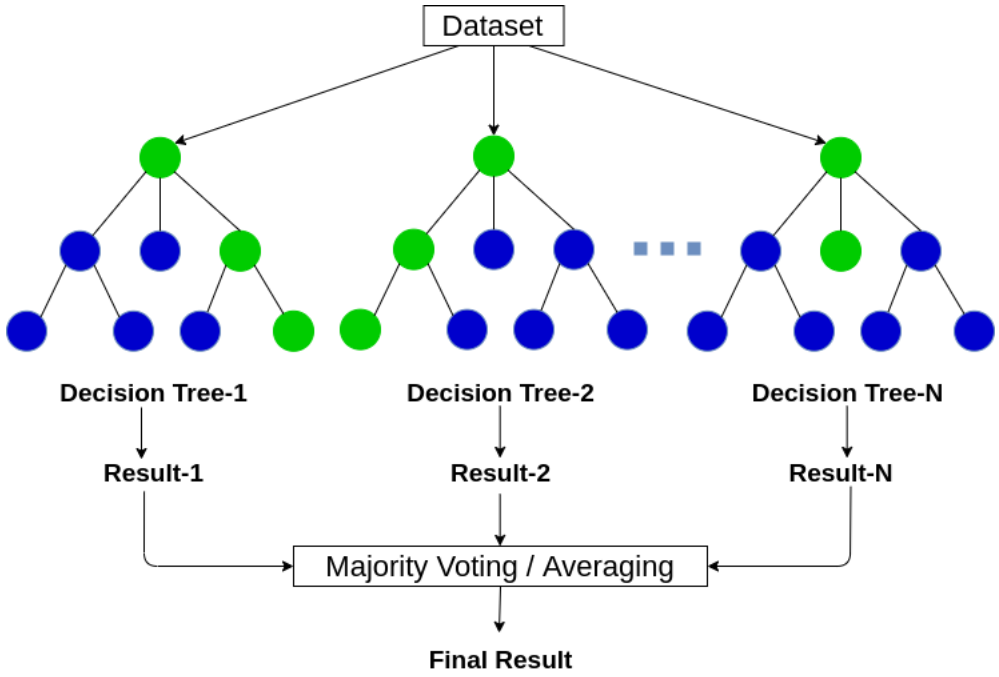

Figure 7: Random Forest Example

implementation of gradient boosted decision trees designed for speed and performance [8]. Since XGBoost is based on Gradient Boosting, we will first discuss 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 actual value and the predicted value of these observations (i.e., loss).Then, depending on the calculated loss, it creates new weak learners to train on and minimize these errors. This process keeps on going until it reaches a certain threshold.

As mentioned before, XGBoost is an advanced 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 stops splitting nodes as soon as it encounters a negative loss; whereas, XGBoost will keep splitting until it

reaches the specified tree depth. In addition, XGBoost has a built-in cross validation function so that it is easier to define the number of boosting rounds for each run as well as others hyperparameters. Since XGBoost has quite a few important hyperparameter, we would have to tune them all in order to get the best result from the model.

3.5 Hyperparameters Tuning Using Cross Validation

The 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 training, they must be set manually beforehand. In order to for a model to perform at its best, its parameters have to be tuned to 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 one 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 on. If C is too large, the model tends to overfit the data and vice versa. Given a list of possible 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 on the resampled dataset in order to find the best values for *n_estimators*: the number of decision trees 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:

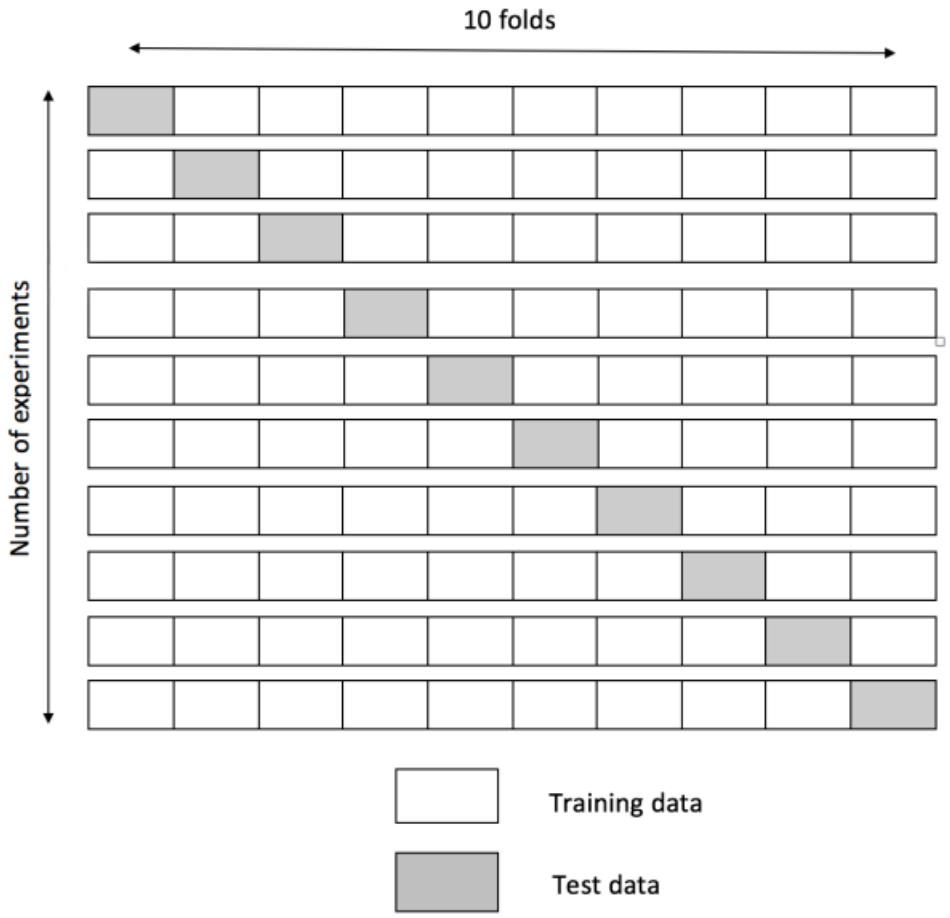

Figure 8: 10-Fold Cross Validation

- **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.
- **colsample_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 limitations, we tuned the hyperparameters one by one with 10-fold cross validation instead of tuning all eight 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 model 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 five samples belong to class 1. Supposedly, a model that predicts 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 five samples in class 1. Furthermore, due to the nature of fraud detection, we would like to correctly detect as many fraudulent cases as possible, so accuracy would not be a 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 disproportional 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 two models that are trying to detect 100 fraud cases from 1 million transactions at the same threshold in a same dataset which has 100 fraud cases:
Model A predicted 100 transactions as fraudulent, in which only 90 is actually fraudulent.
Model B predicted 1000 transactions as fraudulent, in which only 90 is actually fraudulent.
In this case, model A would be the more preferable choice since model B has a lot of false positives. At this threshold, the True Positive Rate and the False Negative Rate of each model would be:
Model A: 0.9 True Positive Rate and 0.00001 False Positive Rate
Model B: 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 have a tiny difference in False Positive Rate ($\sim$ 0.0009). The difference between the two models would be better seen when using PR curve. **Model A**: 0.9 Precision and 0.9 Recall
Model B: 0.0833 Precision and 0.9 Recall
It is clear that model B is far less ideal than model B as there is a huge difference between the two models' precision score ($\sim$ 0.8167).

If the difference varies across different thresholds, the ROC curve would not be able to show it as well as the PR curve. However, we still choose to use the ROC curve as a reference and comparison to the 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 predictions over the total of positive predictions. *Recall* is the number of correct positive predictions over the total of positive ground truth. Given a confusion matrix as in Figure 9, Precision and Recall score are computed as follows:

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

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

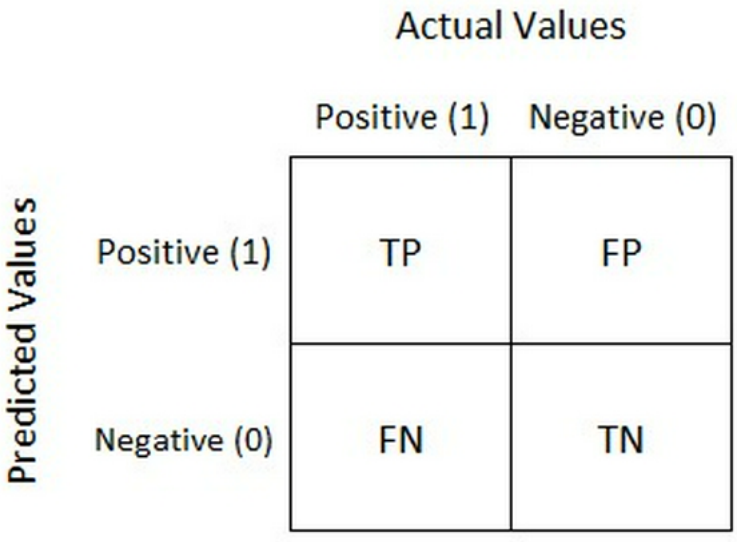

Figure 9: Confusion Matrix Example

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 Negatives (TN) does not contribute to the calculation of F1-score, which is very suitable for skewed data that has a lot of negative samples like in this project. The formula of the F1-score is as follows:

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

4.1.2 Precision-Recall Curve

When working with imbalanced data, we would want to keep track of both the precision and recall of the model to ensure 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 thresholds. A model with low precision and high recall will result in many positive predictions, but many of them would be wrong compared to the ground truth. On the other hand, a model with high precision and low recall would produce few positive predictions, which might leave out a lot of true positive samples, but the predicted ones usually match their ground truth. Figure 10 shows an 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 to calculate the AUC by summing up the areas. 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

Similar to the PR curve, Receiver Operating Characteristic (ROC) [3] curve illustrates the diagnostic probability of a model with various 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 known as Sensitivity or Recall. The FPR is also known as the inverse

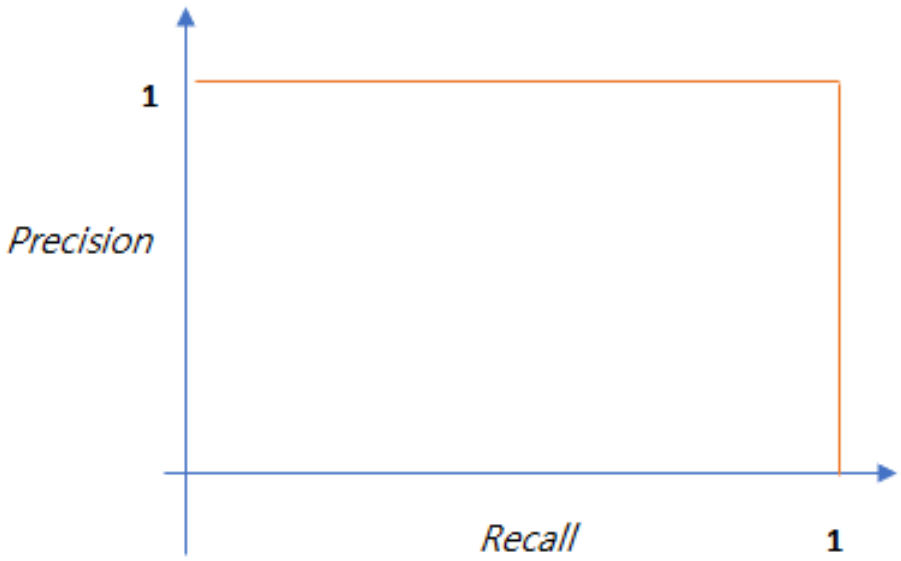

Figure 10: Ideal Precision-Recall Curve

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.

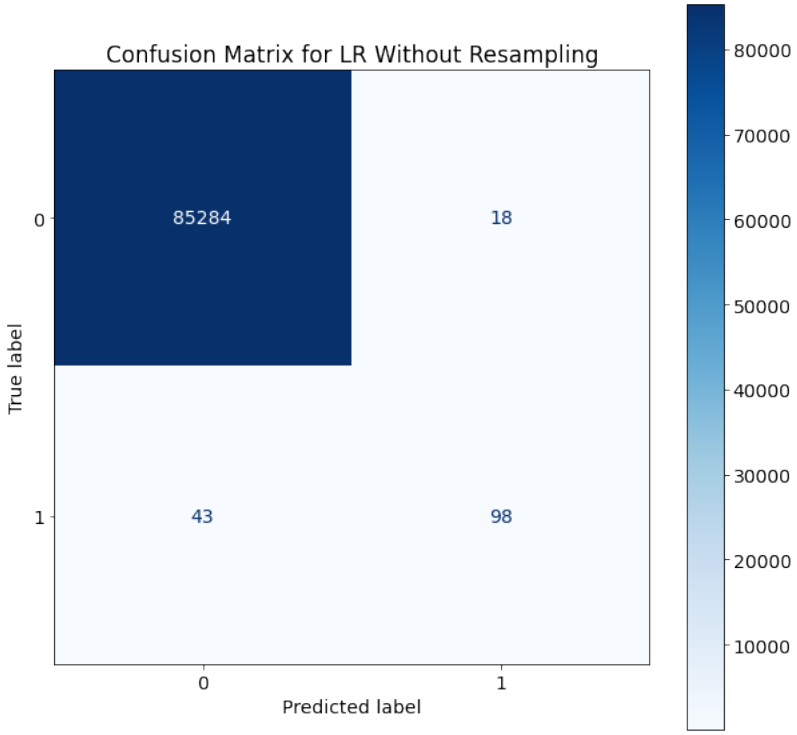

Figure 11: Confusion Matrix: LR no Resampling

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 transactions (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 relatively 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 an AP of 0.70, which is acceptable but not too high.

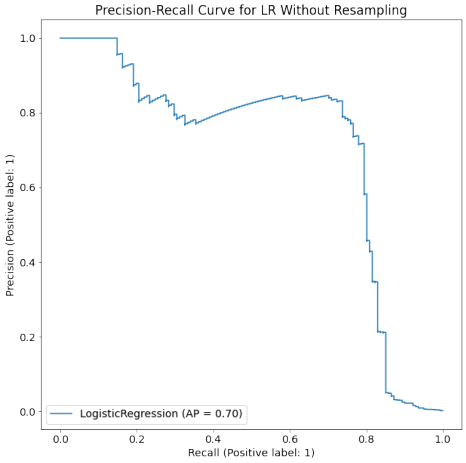

Figure 12: PR Curve: LR no Resampling

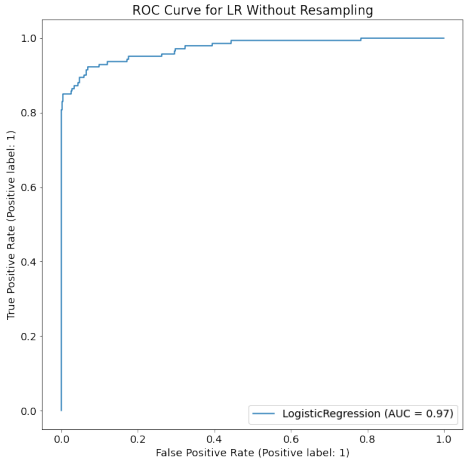

Figure 13: ROC Curve: LR no Resampling

4.2.1.2 Random Oversampling

As we varied the percentage of the number of observations to oversample, 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 the 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 at 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 fewer 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.

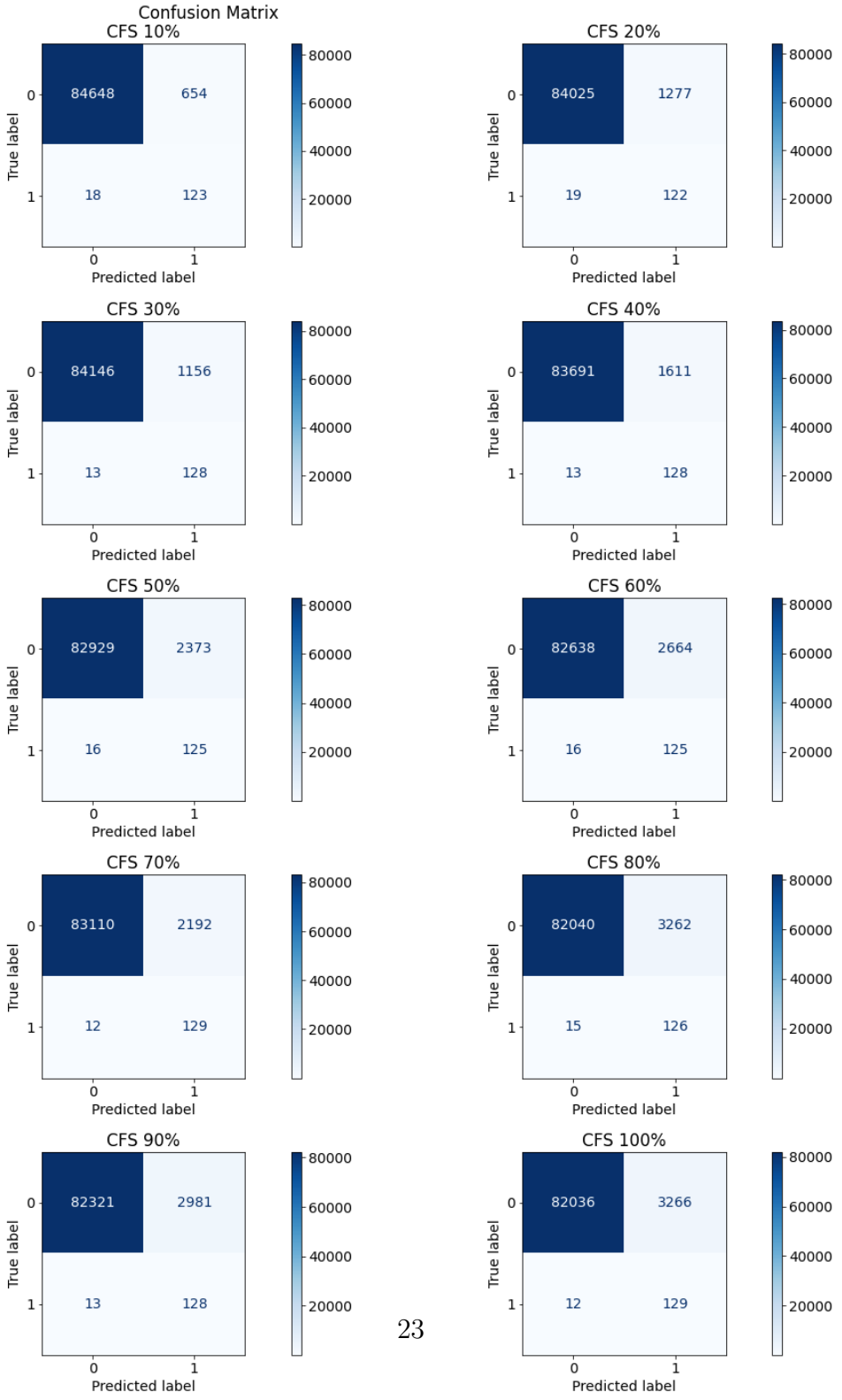

Figure 14: Confusion Matrix: LR Random Oversampling

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

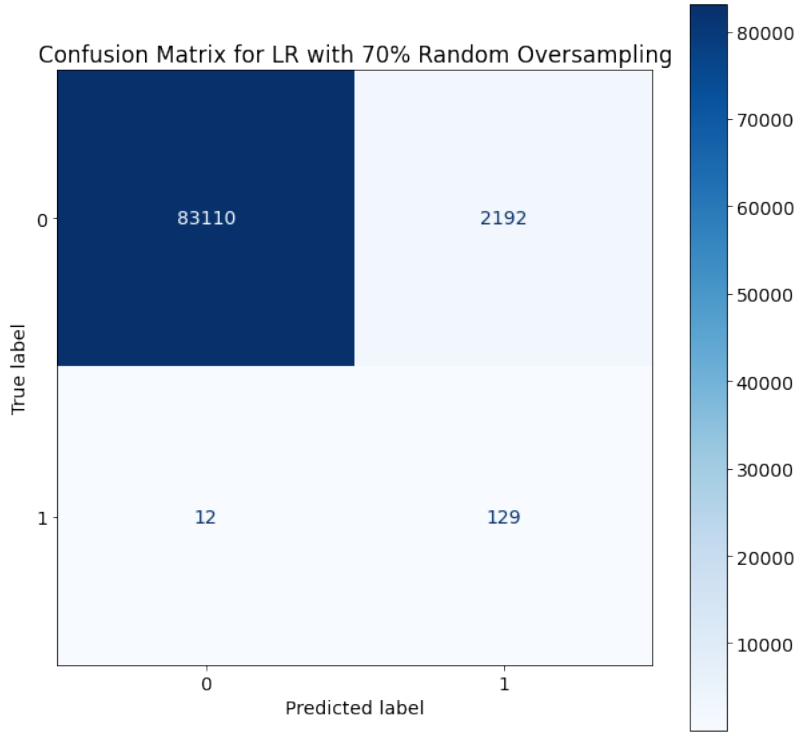

Figure 15: Confusion Matrix: LR 70% Random Oversampling

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

By applying Random Oversampling to the data, we managed to increase the number of correctly detected frauds 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 are 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 at an acceptable threshold; however, it would be more preferable if we can achieve a higher score as we are dealing with a sensitive issue of detecting fraud.

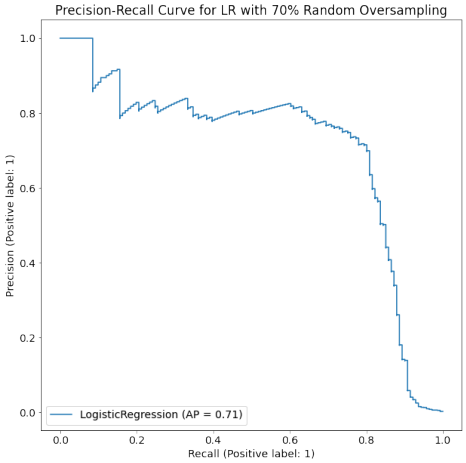

Figure 16: PR Curve: LR 70% Random Oversampling

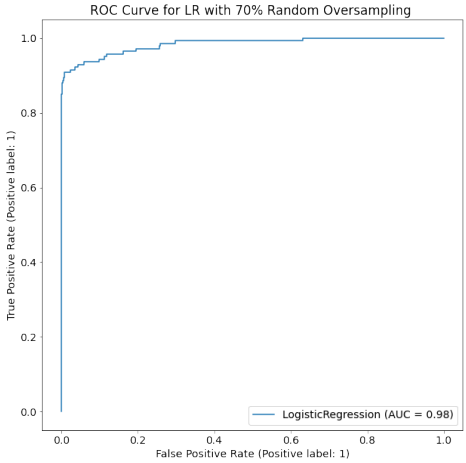

Figure 17: ROC Curve: LR 70% Random Oversampling

4.2.1.3 Random Undersampling

Similar to Random Oversampling, we also varied the percentage of the resampled data. 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 an overview of all the PR curves, the AP of the 70% undersampled dataset is lower than most others. With the Random Undersampling technique, we decided to train the model using the dataset that 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 are depicted in figure 20 and figure 21.

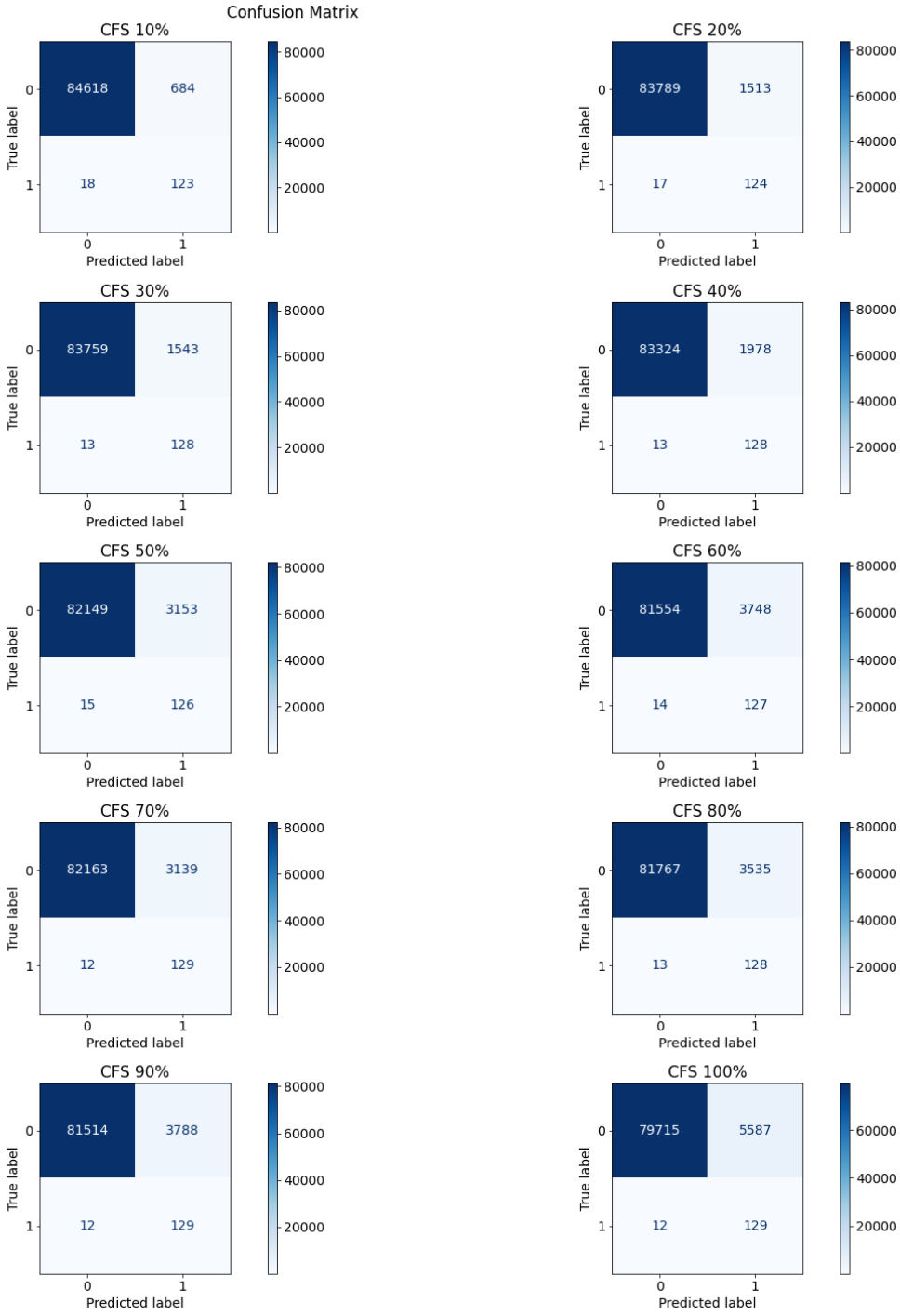

Figure 18: Confusion Matrix: LR Random Undersampling

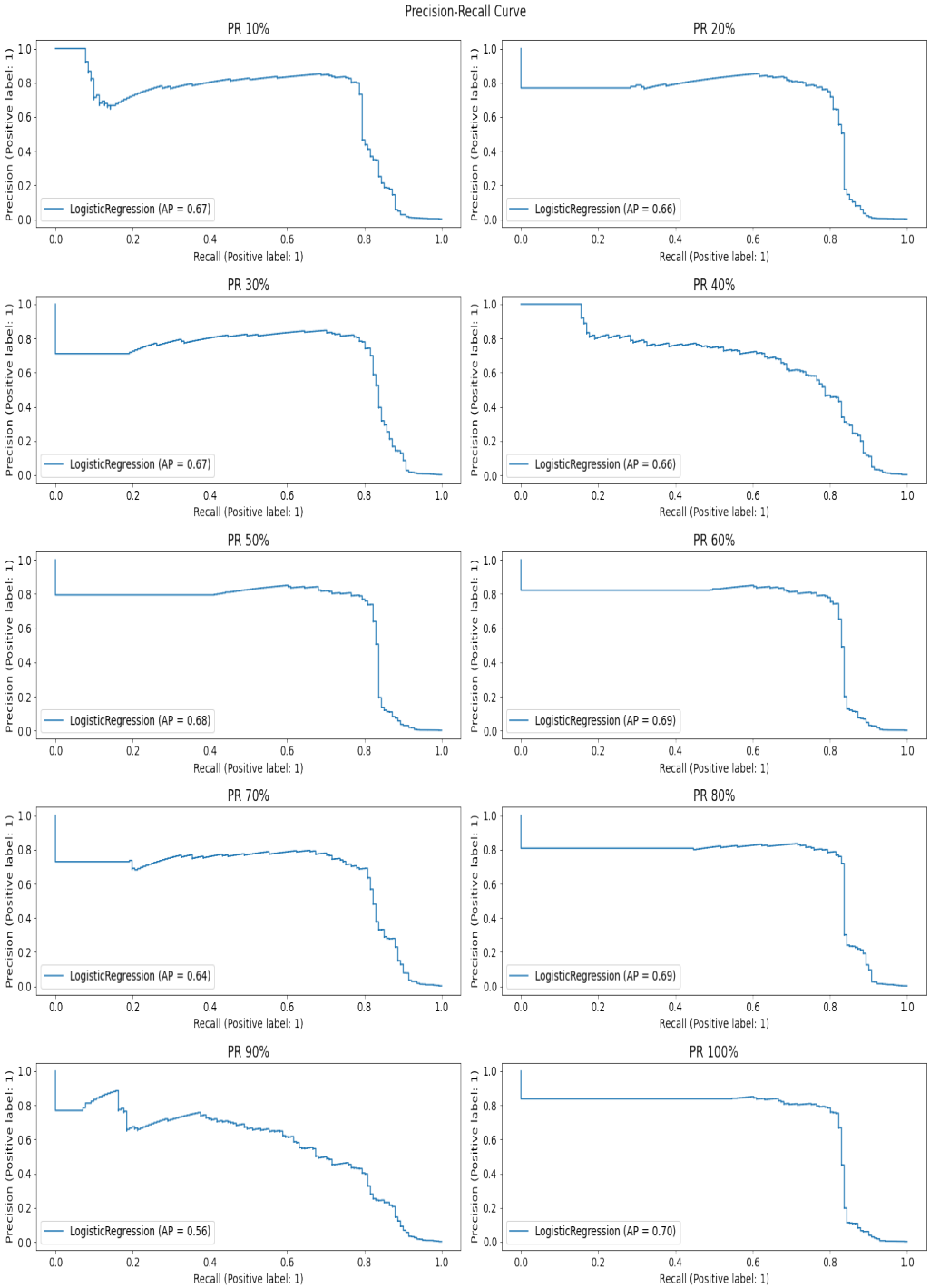

Figure 19: PR Curve: LR Random Undersampling

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

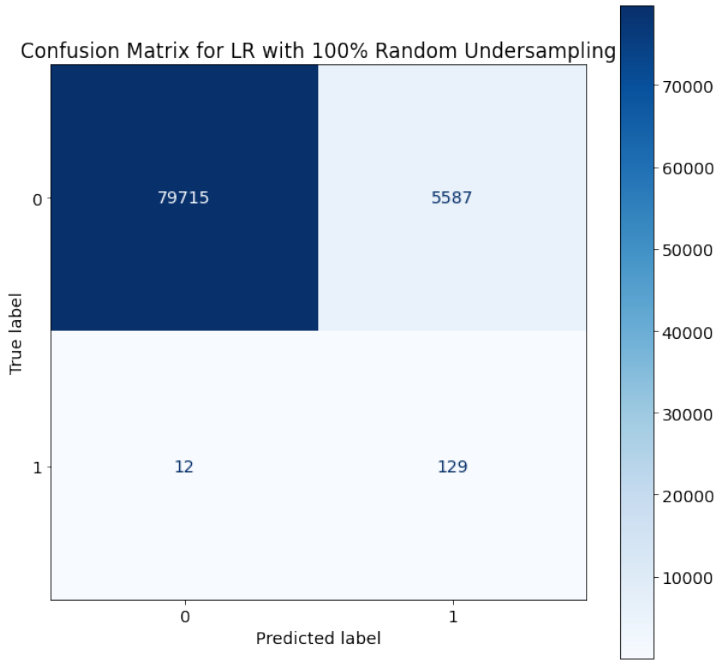

Figure 20: Confusion Matrix: LR 100% Random Undersampling

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

As expected, the model still performs 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 a 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.

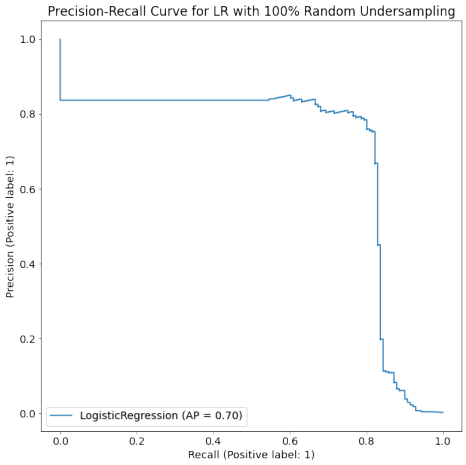

Figure 21: PR Curve: LR 100% Random Undersampling

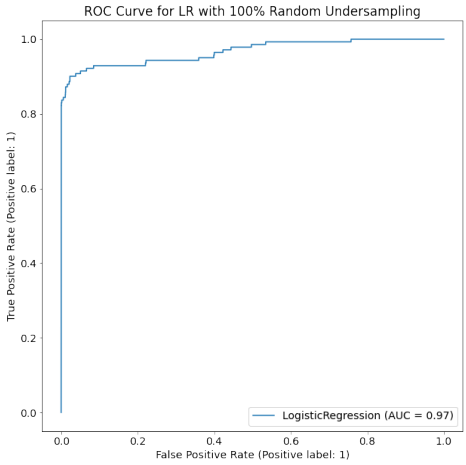

Figure 22: ROC Curve: LR 100% Random Undersampling

4.2.1.4 SMOTE

For SMOTE, the model performed best when we synthesize new training examples 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 predictions (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 too 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.

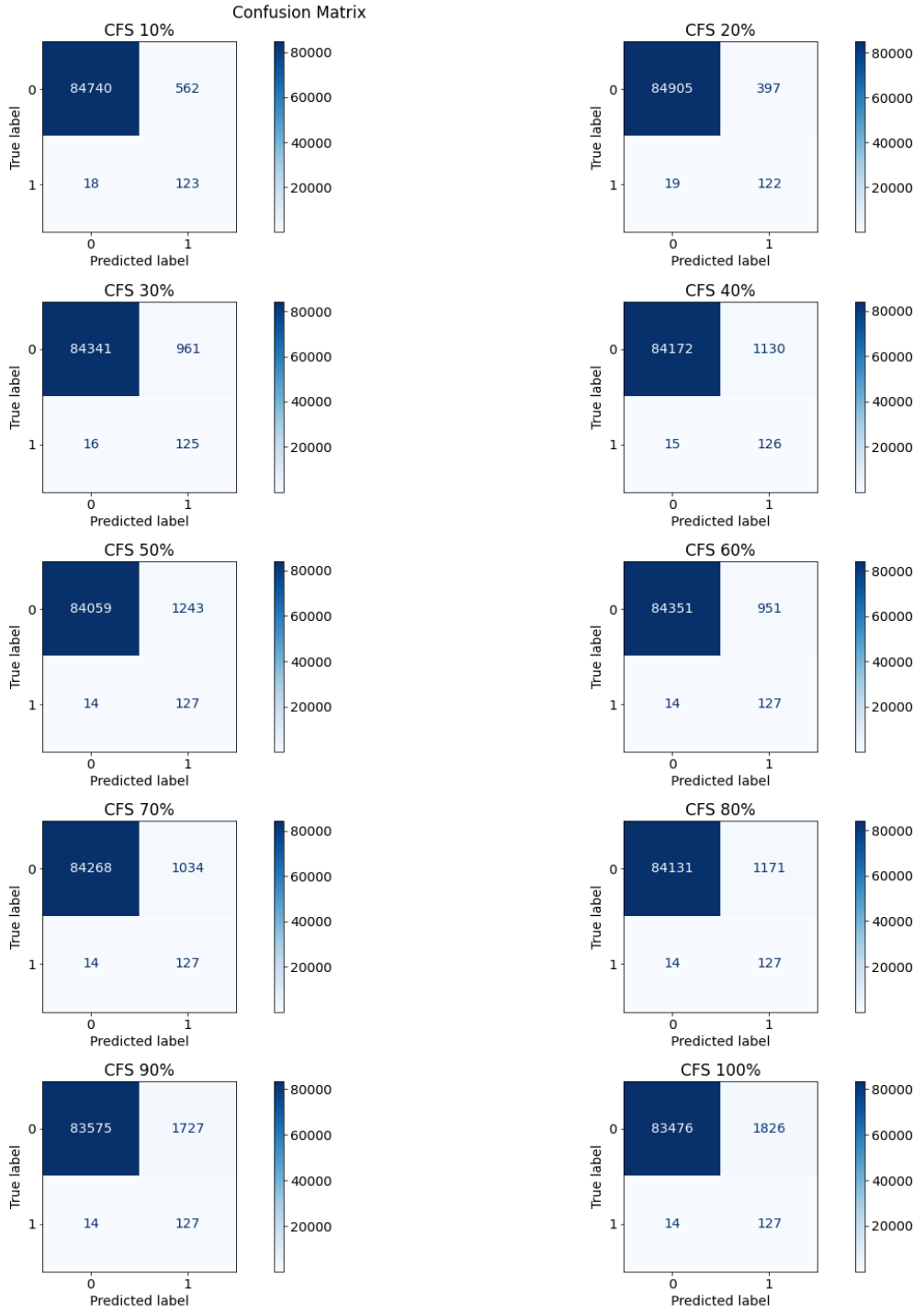

Figure 23: Confusion Matrix: LR SMOTE

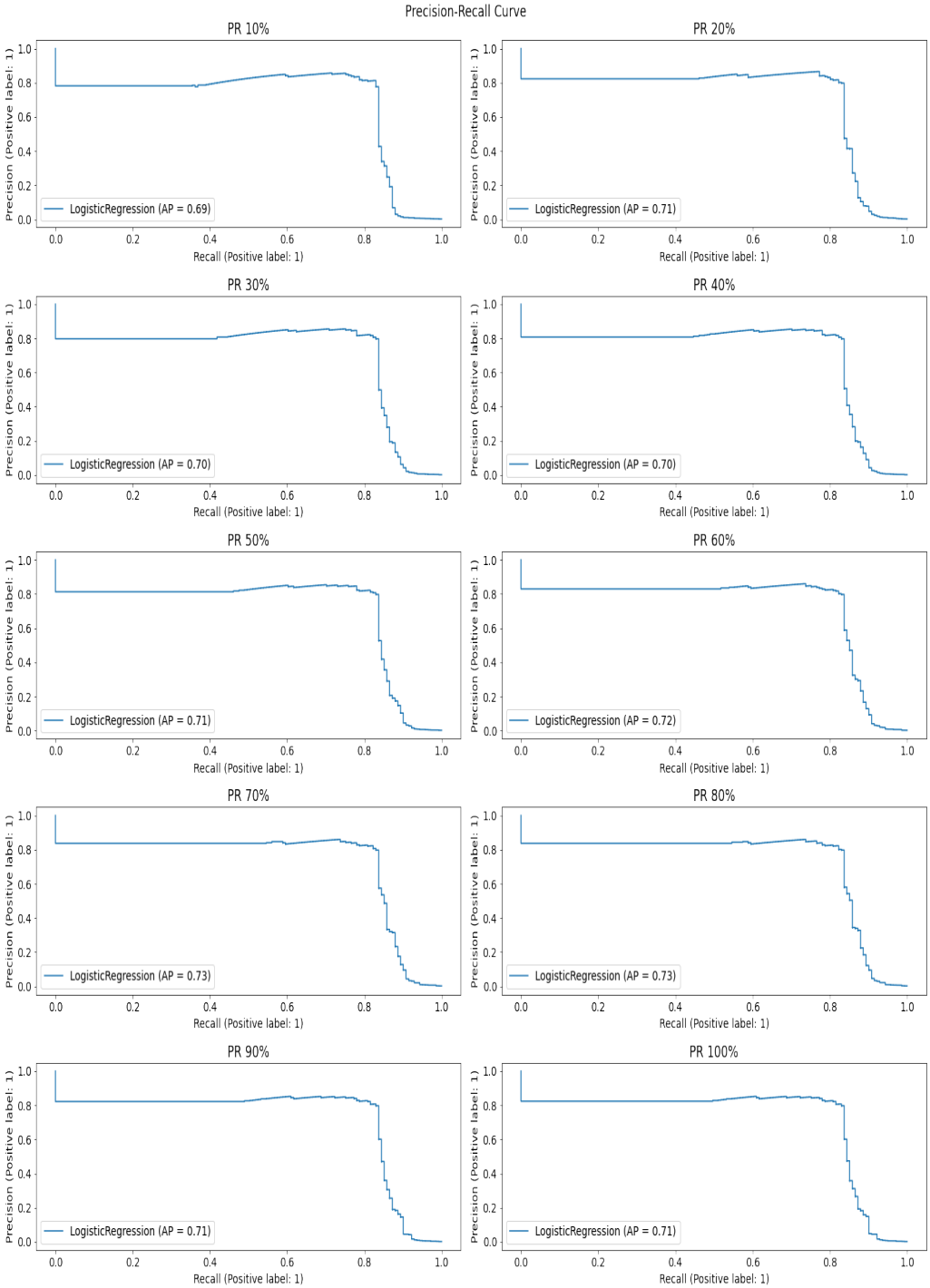

Figure 24: PR Curve: LR SMOTE

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

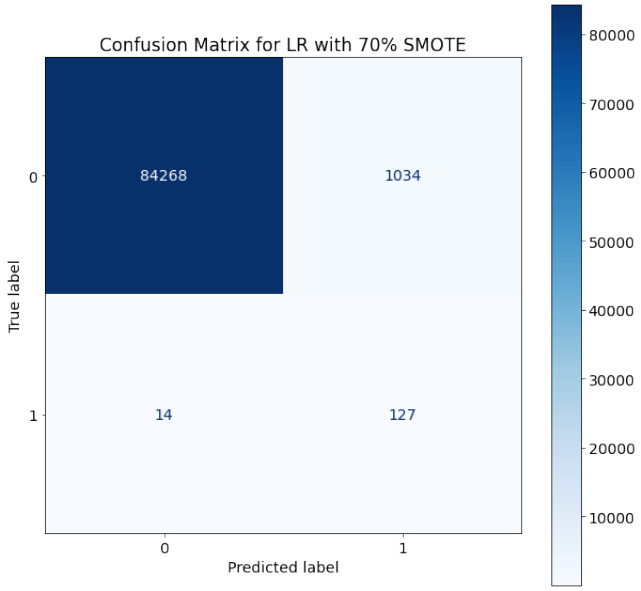

Figure 25: Confusion Matrix: LR 70% SMOTE

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

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.

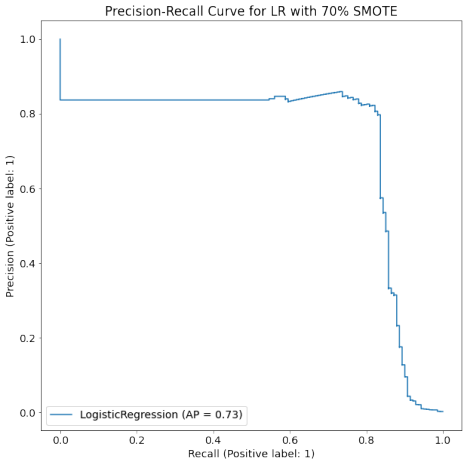

Figure 26: PR Curve: LR 70% SMOTE

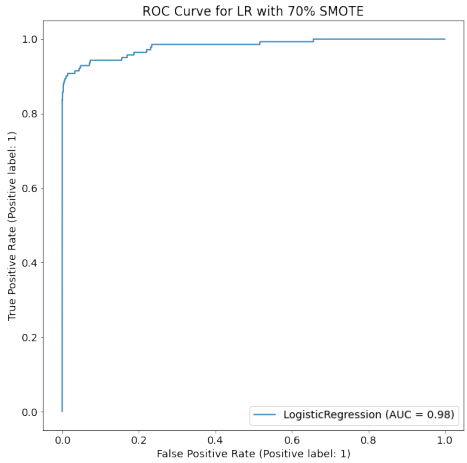

Figure 27: ROC Curve: LR 70% SMOTE

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.

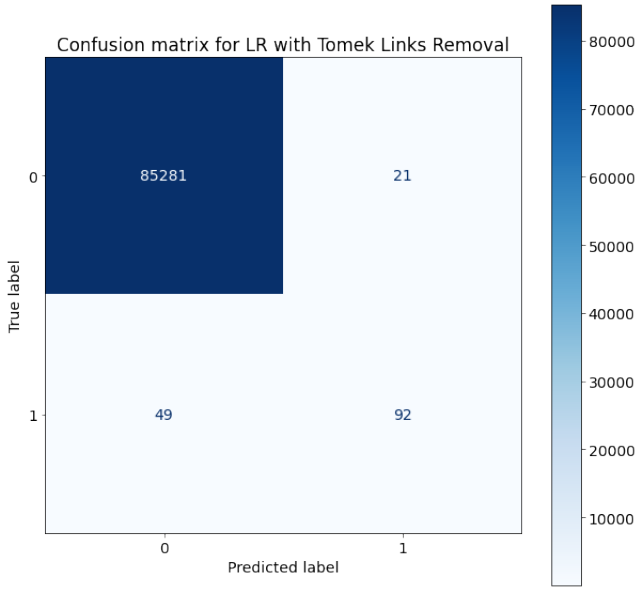

Figure 28: Confusion Matrix: LR Tomek Links Removal

Class	Precision	Recall	F1 Score
0	0.99943	0.99975	0.99959
1	0.81416	0.65248	0.72441
Avg	0.99912	0.99918	0.99914

Table 6: Classification Report: LR Tomek Links Removal

Like previous models, this model did well when handling negative class but failed to perform when dealing with positive ones. However, comparing to other resampling techniques, the precision, recall, and f1 score of the model with respect to the positive class is 0.81416, 0.65248, and 0.72441, respectively. In terms of AP score, the model only achieved a score of 0.64, which is relatively low for a predictive model.

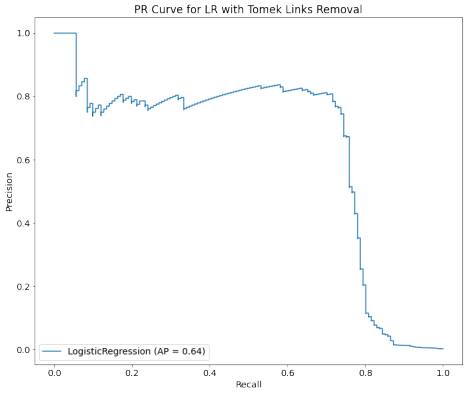

Figure 29: PR Curve: LR Tomek Links Removal

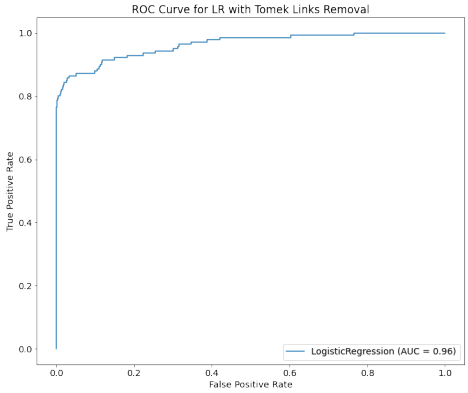

Figure 30: ROC Curve: LR Tomek Links Removal

4.2.1.6 Hybrid Resampling

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

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

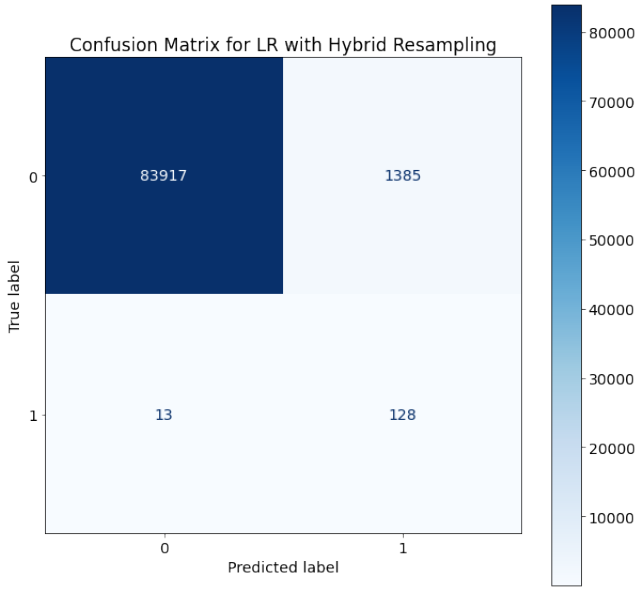

Figure 31: Confusion Matrix: LR Hybrid Resampling

Class	Precision	Recall	F1 Score
0	0.99985	0.98376	0.99174
1	0.08460	0.90780	0.15478
Avg	0.99833	0.98364	0.99036

Table 7: Classification Report: LR Hybrid Resampling

With this technique, the model did not improve its performance much as it only scored a f1 score of 0.15478 as in table 7 and an AP score of 0.71 as in figure 32.

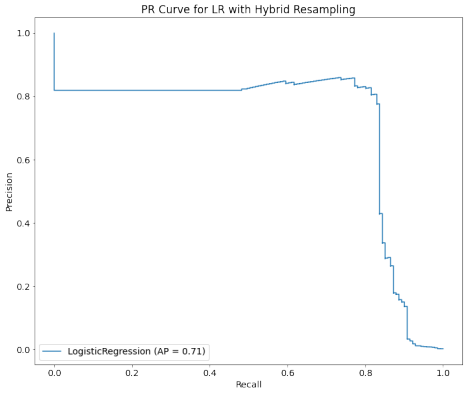

Figure 32: PR Curve: LR Hybrid Resampling

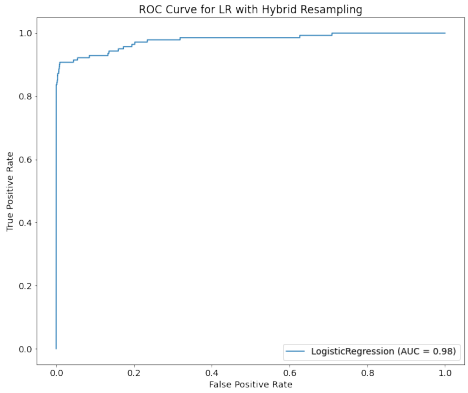

Figure 33: ROC Curve: LR Hybrid Resampling

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.

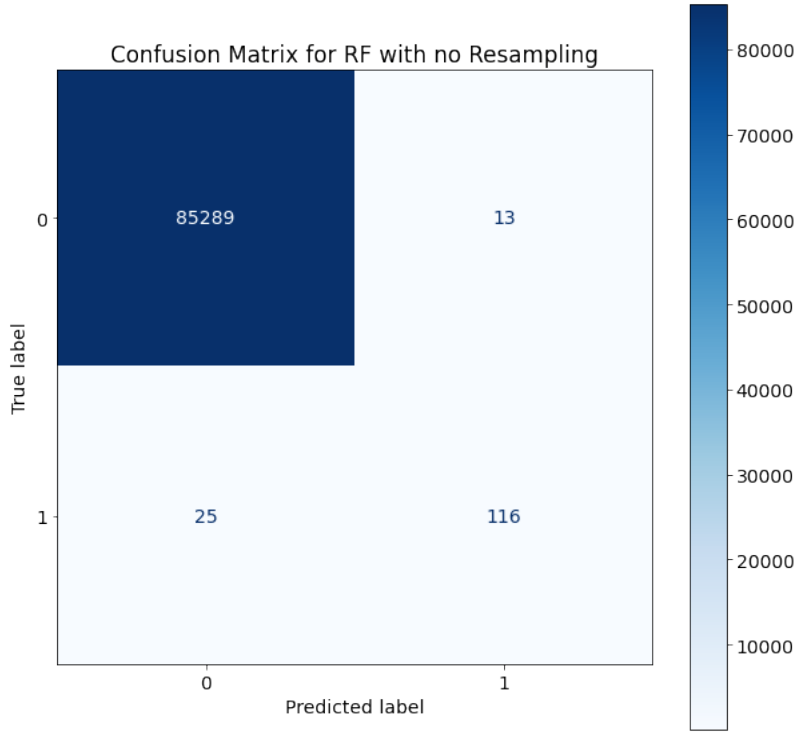

Figure 34: Confusion Matrix: RF No Resampling

Class	Precision	Recall	F1 Score
0	0.99971	0.99985	0.99978
1	0.89922	0.82270	0.85926
Avg	0.99954	0.99956	0.99955

Table 8: Classification Report: RF no Resampling

As can be seen from table 8, the Random Forest model performed better than the Logistic Regression model even when 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 0.89922, 0.82270, and 0.85926. In addition, compared 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.

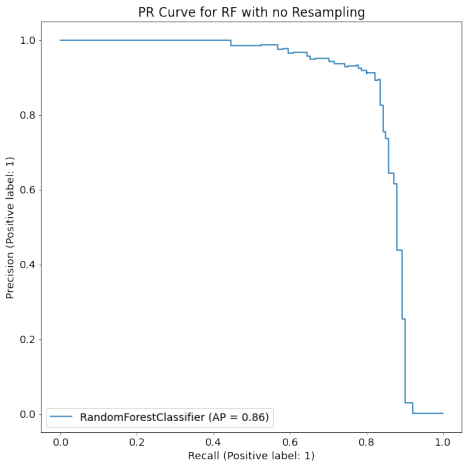

Figure 35: PR Curve: RF no Resampling

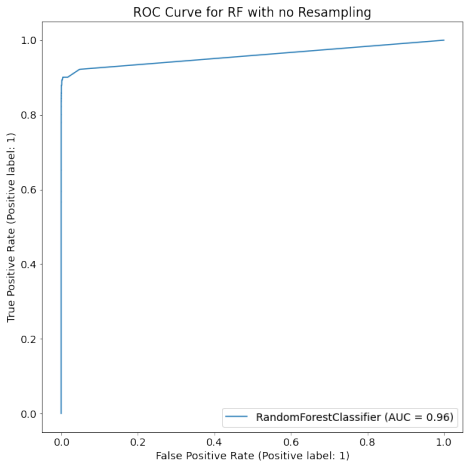

Figure 36: ROC Curve: RF no Resampling

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, 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 is shown in figure 38.

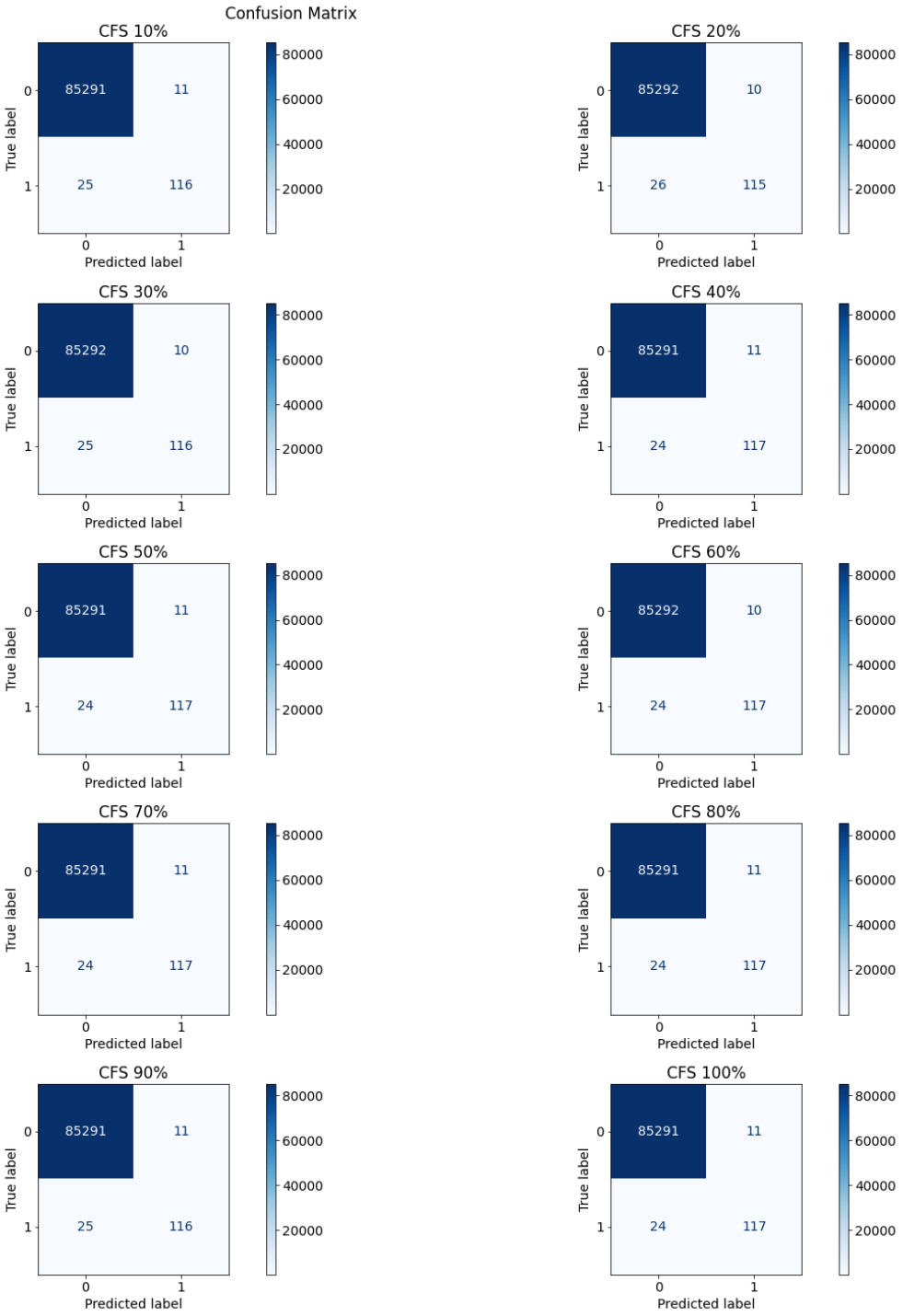

Figure 37: Confusion Matrix: RF Random Oversampling

After using GridsearchCV to tune the parameter for the model, the best 'n_estimators' parameter of the model is 100.

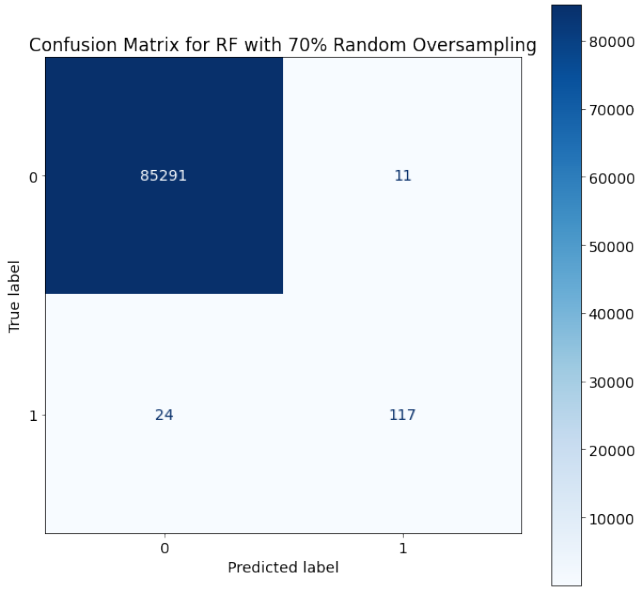

Figure 38: Confusion Matrix: RF 70% Random Oversampling

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 70% Random Oversampling

As expected, random forest with random oversampling performed well on both classes as f1 scores for class 0 and class 1 are 0.99979 and 086989, respectively, as shown in table 9. Regarding its PR AUC, the model achieved a high score of 0.87.

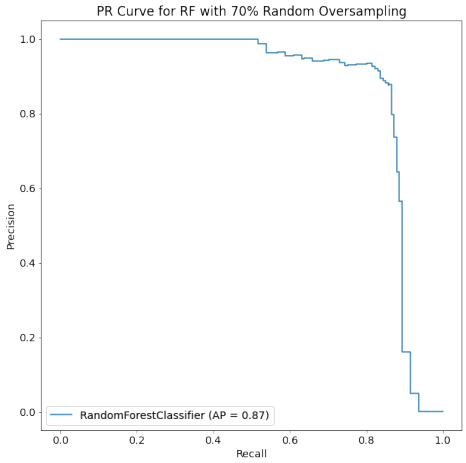

Figure 39: PR Curve: RF 70% Random Oversampling

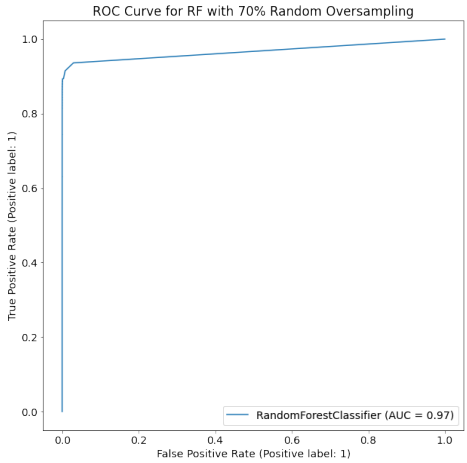

Figure 40: ROC Curve: RF 70% Random Oversampling

4.2.2.3 Random Undersampling

As we changed the ratio of the classes, we decided to use the 70% undersampled dataset as it maximized the number correctly detected fraudulent transactions and minimized the number of misclassified cases in both classes.

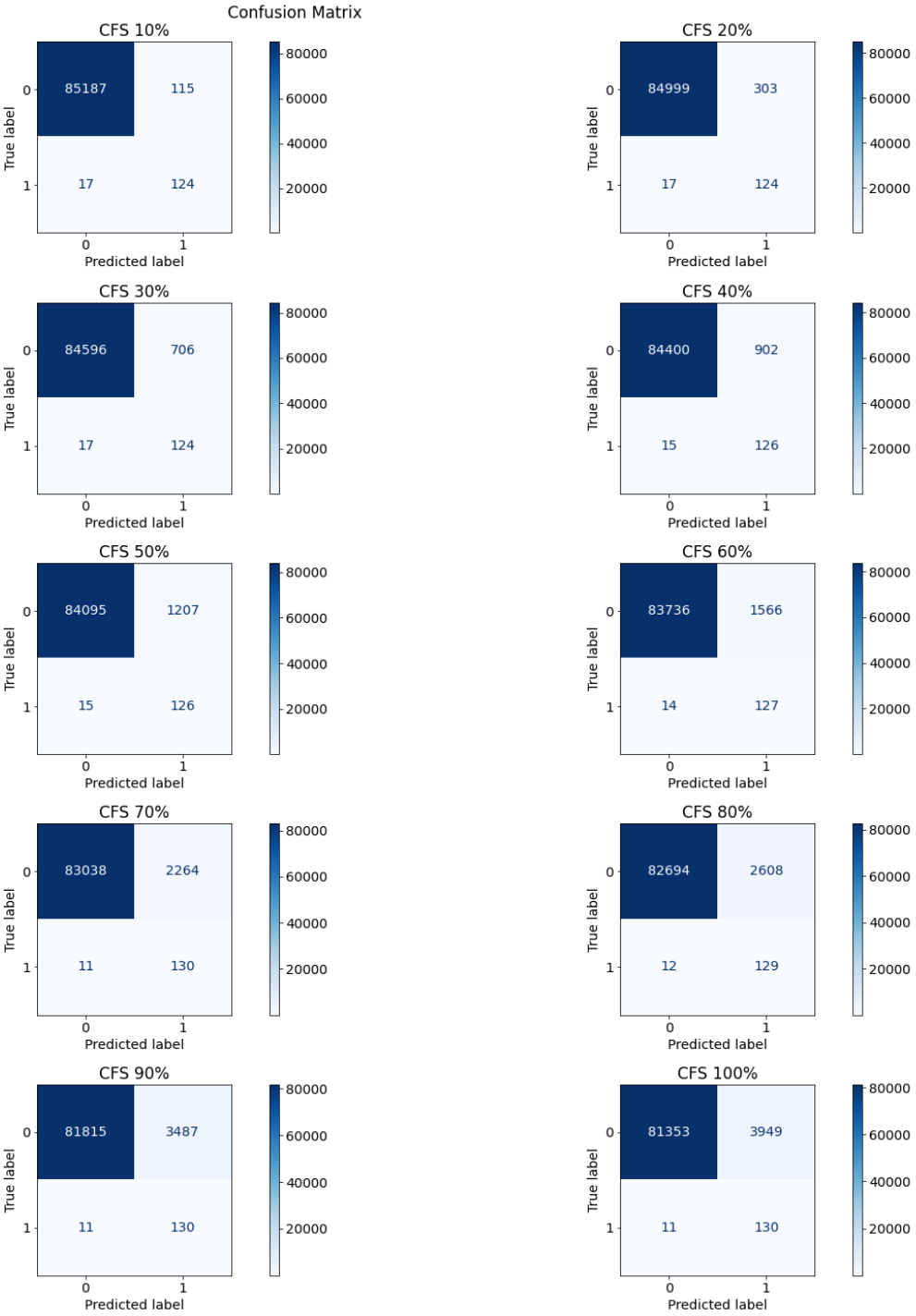

Figure 41: Confusion Matrix: RF Random Undersampling

After using GridsearchCV to tune the parameter for the model, the best 'n_estimators' parameter of the model is 200.

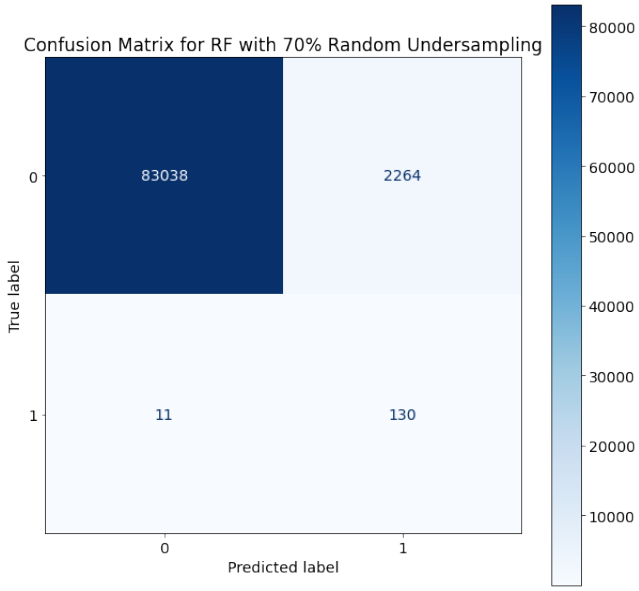

Figure 42: Confusion Matrix: RF 70% Random Undersampling

Class	Precision	Recall	F1 Score
0	0.99987	0.97346	0.98649
1	0.05430	0.92199	0.10256
Avg	0.99831	0.97337	0.98503

Table 10: Classification Report: RF 70% Random Undersampling

As we can see from table 10, the Random Forest model that used random undersampling performed poorly when handling fraudulent transactions. Although its recall score was high at 0.92199, its precision is only 0.05430 as it misclassified a lot of genuine transactions as fraud. As a result, its f1 score and AP is also rather low, 0.10256 and 0.71, respectively.

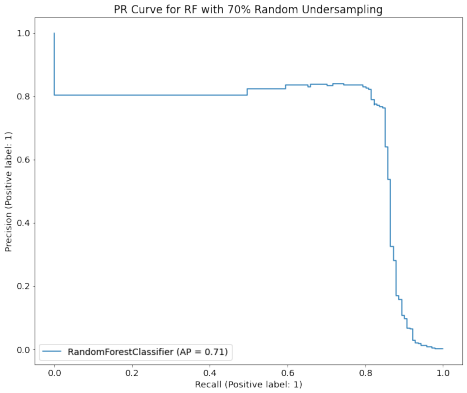

Figure 43: PR Curve: RF 70% Random Undersampling

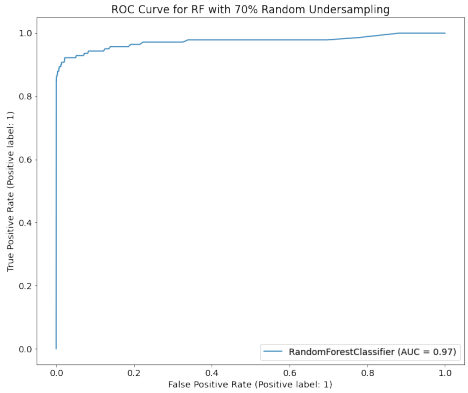

Figure 44: ROC Curve: RF 70% Random Undersampling

4.2.2.4 SMOTE

Similar to other models, different ratios between fraud and legitimate do not affect the performance too much. Most of them correctly predicted about 120 fraud cases and falsely predicted 20 fraudulent transactions as genuine. As shown in figure 45, the dataset where the number of positive class sample is half of the negative ones give us the best performance. We decided to use it as the training data to fit our predictive model. Figure 46 and figure 47 gives us a better view of the model's confusion matrix and PR curve.

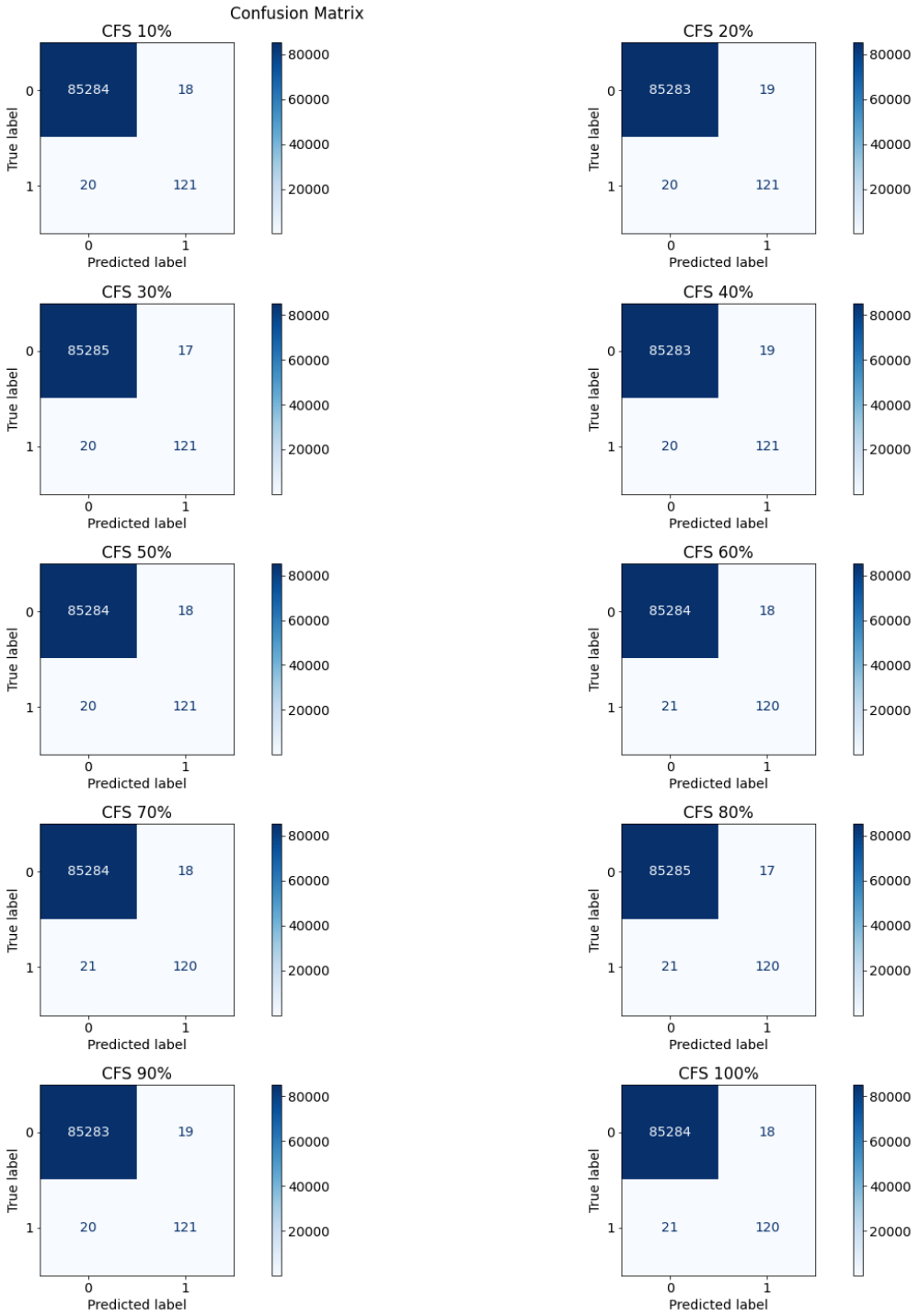

Figure 45: Confusion Matrix: RF SMOTE

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

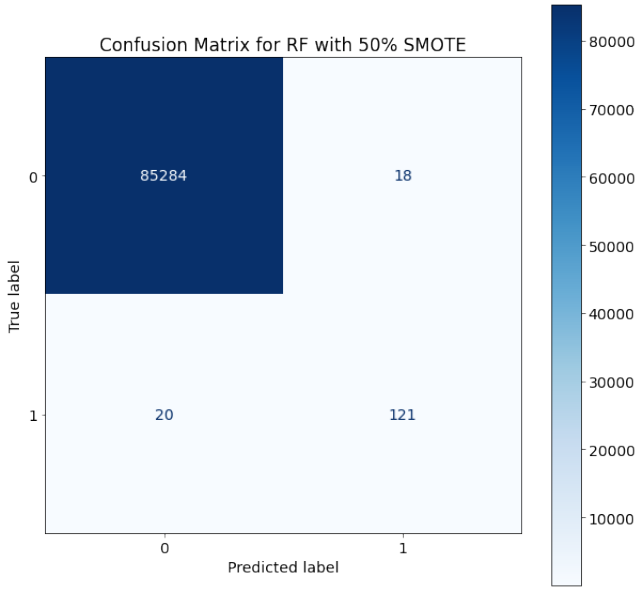

Figure 46: Confusion Matrix: RF 50% SMOTE

Class	Precision	Recall	F1 Score
0	0.99977	0.99979	0.99978
1	0.87050	0.85816	0.86429
Avg	0.99955	0.99956	0.99955

Table 11: Classification Report: RF 50% SMOTE

Table 11 shows us the metrics scores of the model. The model performed relatively well when predicting both positive and negative classes. When encountering legitimate transactions, all three precision, recall, and f1 score of the model are high, very close to 1. The model also did a good job handling illegitimate transactions as it scored acceptable scores on all metrics (0.85 - 0.87). In addition, as shown in figure 47 and figure 48, both AP score and AUC score are at 0.87 and 0.98, respectively, which is a reasonable threshold for a predictive model.

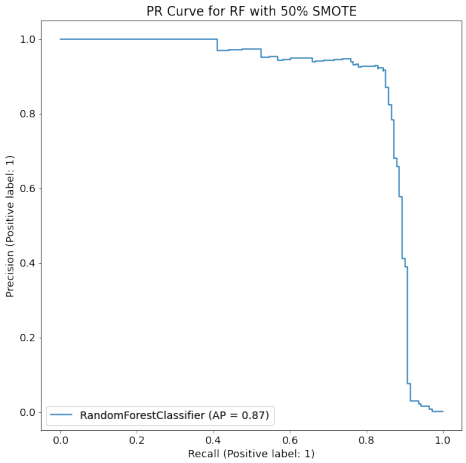

Figure 47: PR Curve: RF 50% SMOTE

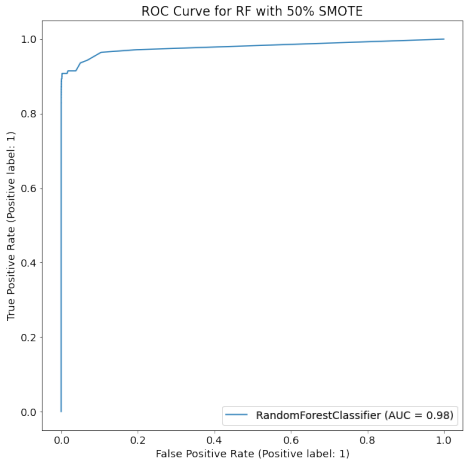

Figure 48: ROC Curve: RF 50% SMOTE

4.2.2.5 Tomek Links Removal

After using GridsearchCV to tune the parameter for the model, the best 'n_estimators' parameter of the model is 100.

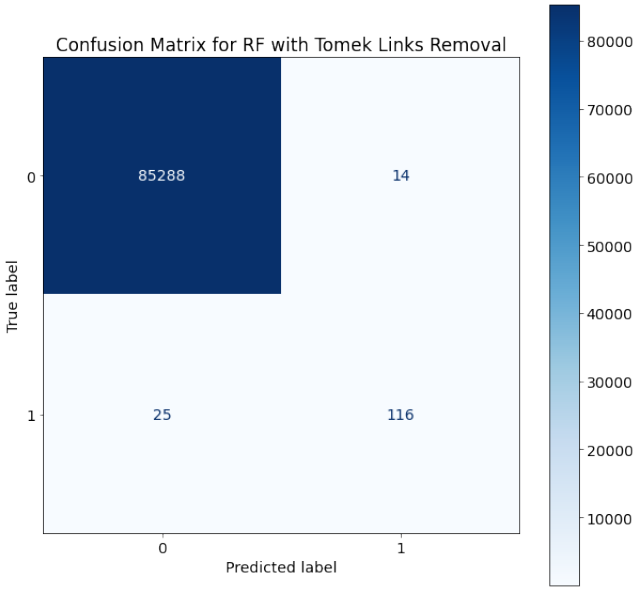

Figure 49: Confusion Matrix: RF Tomek Links Removal

Class	Precision	Recall	F1 Score
0	0.99971	0.99984	0.99977
1	0.89231	0.82270	0.85609
Avg	0.99953	0.99954	0.99953

Table 12: Classification Report: RF Tomek Links Removal

Figure 49 and table 12 give us a detailed description of the model's performance when classifying the transactions. Similar to other Random Forest models, this model still did well in detecting illegal transactions as all three of its metrics: precision, recall, and f1 score are all above 0.8. In addition, the model also achieved an AP score of 0.85, as depicted in figure 50.

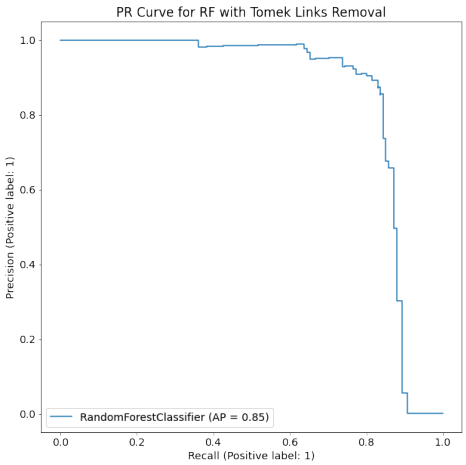

Figure 50: PR Curve: RF Tomek Links Removal

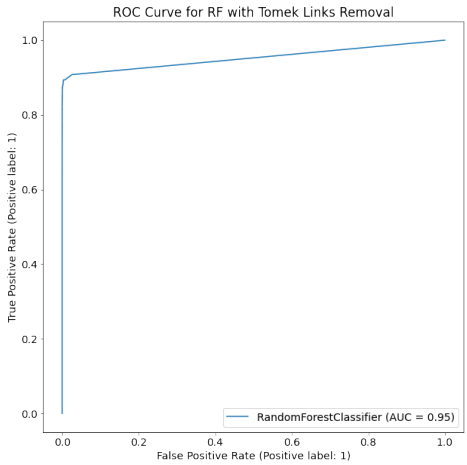

Figure 51: ROC Curve: RF Tomek Links Removal

4.2.2.6 Hybrid Resampling

For hybrid resampling technique, we used the best performed SMOTE-ed dataset and Tomek Links Removal algorithm to create our resampled training dataset. For this model, Tomek Links Removal was applied to the final dataset used in section 4.2.2.4.

After using GridsearchCV to tune the parameter for the model, the best 'n_estimators' parameter of the model is 100.

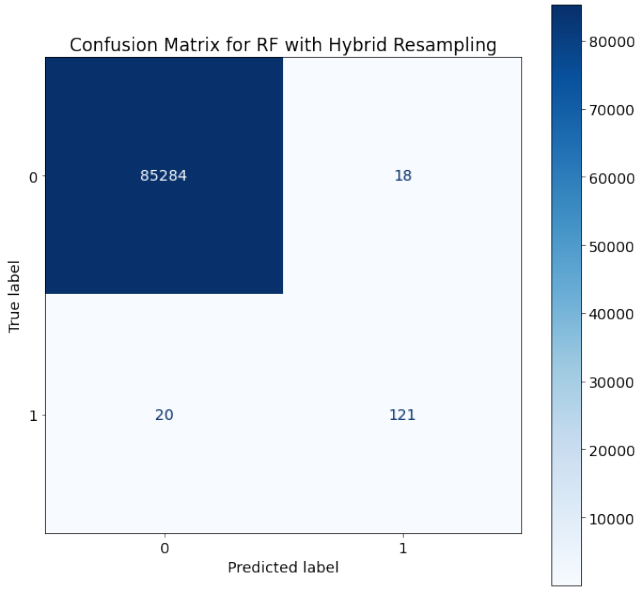

Figure 52: Confusion Matrix: RF Hybrid Resampling

Class	Precision	Recall	F1 Score
0	0.99977	0.99979	0.99978
1	0.87050	0.85816	0.86429
Avg	0.99955	0.99956	0.99955

Table 13: Classification Report: RF Hybrid Resampling

Comparing to the Random Forest model that used SMOTE, this model performance does not differ too much. The metrics scores shown in table 13 are almost identical to ones of our SMOTE model without applying Tomek Links Removal. The model achieved a high f1 score of 0.86429 and a considerably good AP and AUC of 0.85 and 0.97.

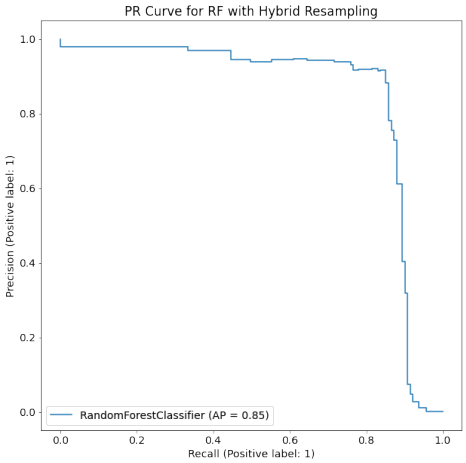

Figure 53: PR Curve: RF Hybrid Resampling

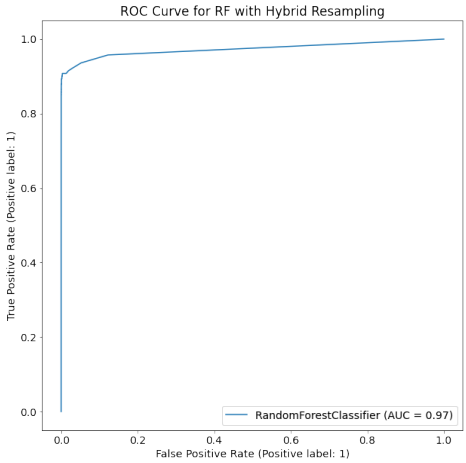

Figure 54: ROC Curve: RF Hybrid Resampling

4.2.3 XGBoost

4.2.3.1 No Resampling

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.5, colsample_bytree = 0.9, max_depth = 4, alpha = 0.0001, min_child_weight = 1,gamma = 0, n_estimators = 800, learning_rate = 0.3.

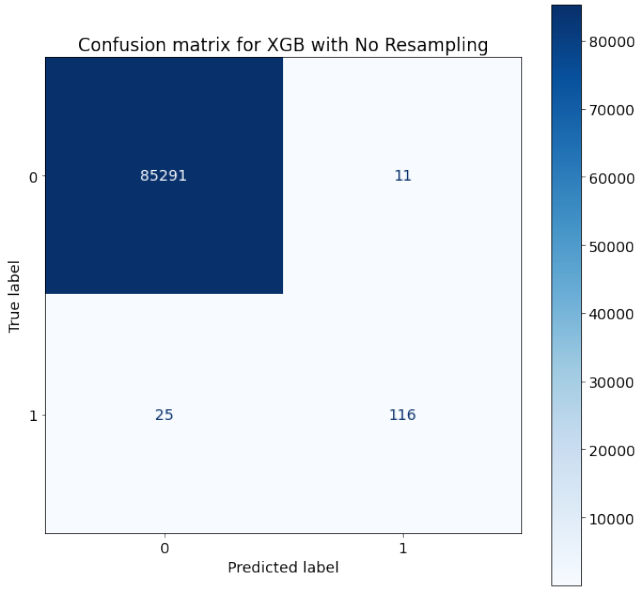

Figure 55: Confusion Matrix: XGB No Resampling

Class	Precision	Recall	F1 Score
0	0.99971	0.99987	0.99979
1	0.91339	0.82270	0.86567
Avg	0.99956	0.99958	0.99957

Table 14: Classification Report: XGB No Resampling

Similar to our Random Forest model, even without any resampling, the model still performed quite well on both classes as it scored 0.91339, 0.82270 and 0.86567 for its precision, recall, and f1 score, respectively, as depicted in table 14. In addition, the model also achieved a high AP score of 0.86.

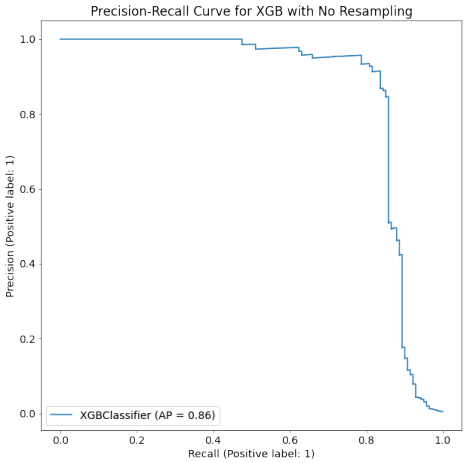

Figure 56: PR Curve: XGB No Resampling

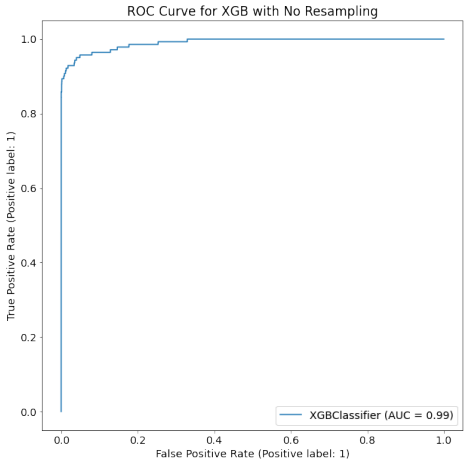

Figure 57: ROC Curve: XGB No Resampling

4.2.3.2 Random Oversampling

Figure 58 shows us that when we varied the ratio of oversampling, the result does not change too much as most of them correctly predicted 19 - 21 fraud cases. Due to the nature of fraud detection, we would choose the dataset that gives us the highest correct prediction. Hence, for this model, we used the 90% randomly oversampled dataset.

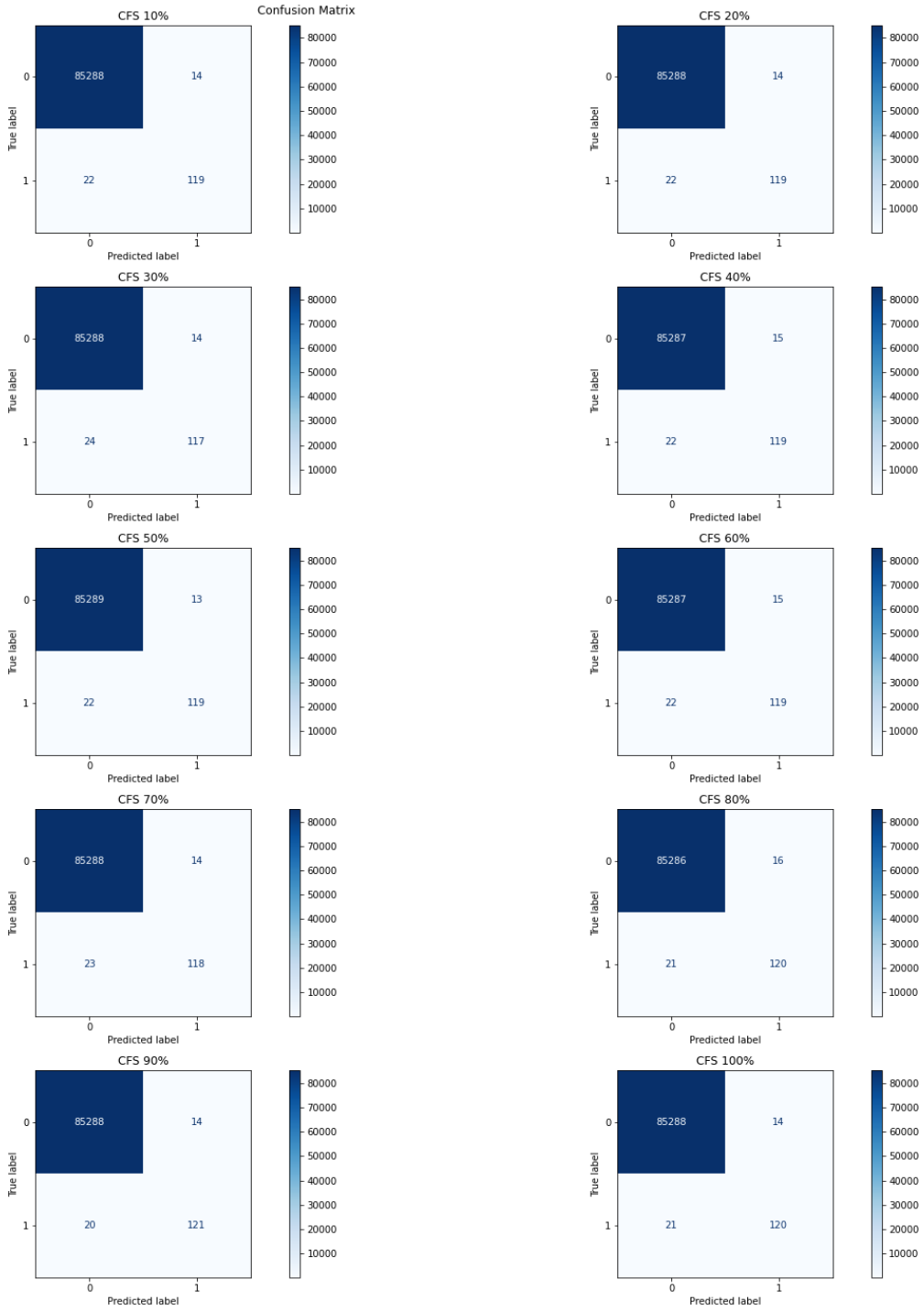

Figure 58: Confusion Matrix: XGB Random Oversampling

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.8, colsample_bytree = 0.8, max_depth = 10, alpha = 0, min_child_weight = 1,gamma = 0, n_estimators = 1000, learning_rate = 0.3.

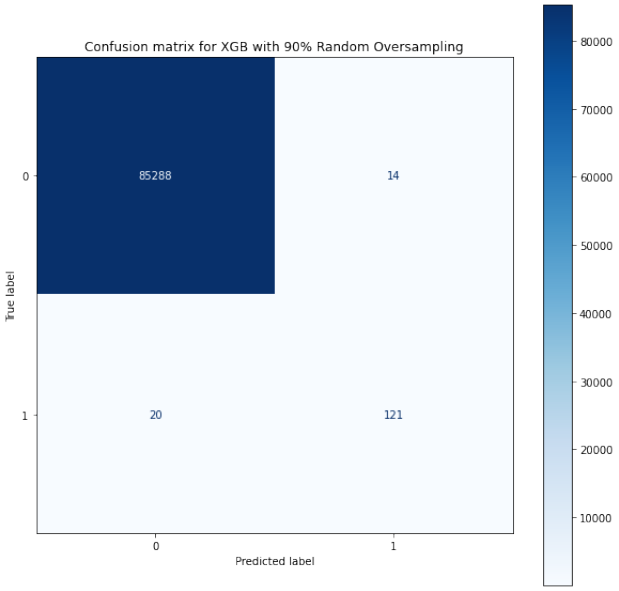

Figure 59: Confusion Matrix: XGB 90% Random Oversampling

Class	Precision	Recall	F1 Score
0	0.99977	0.99984	0.99980
1	0.89630	0.85816	0.87681
Avg	0.99959	0.99960	0.99960

Table 15: Classification Report: XGB 90% Random Oversampling

As expected, the model did an excellent job classifying both negative and positive class with high metrics scores. When handling genuine cases, the model's precision, recall and, f1 score were all over 0.999. When trying to detect fraudulent transactions, it also managed to achieve an f1 score of 0,87681, which is rather good. The model also has a high AP score of 0.87.

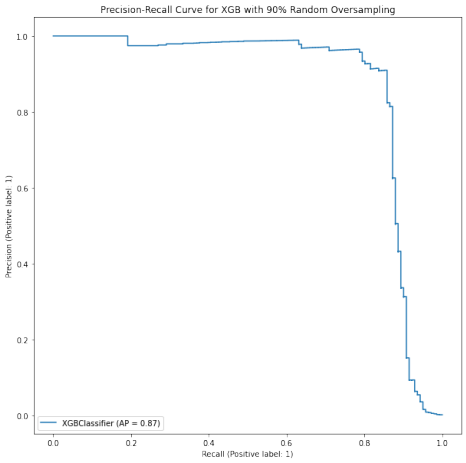

Figure 60: PR Curve: XGB 90% Random Oversampling

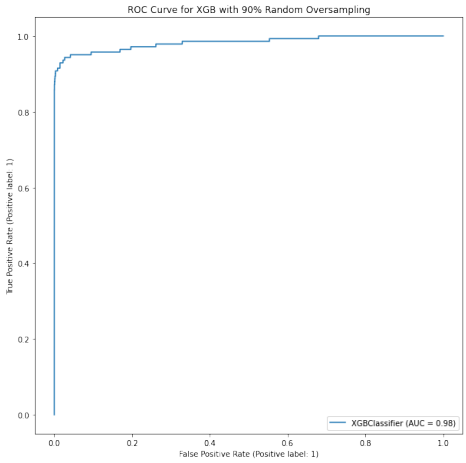

Figure 61: ROC Curve: XGB 90% Random Oversampling

4.2.3.3 Random Undersampling

As we varied our range of undersampling, we noticed that as the ratio between two classes gets closer to 1, the model seems to do a better job detecting fraud; however, it also produced more false positives since it misclassified more genuine transactions as fraud. This can be seen from figure 62, which gives us an overview of the confusion matrices from all ratios. For this model, we decided to used the dataset with the class ratio of 1 as our training data because our primary goal is to maximize the number of correctly detected fraud.

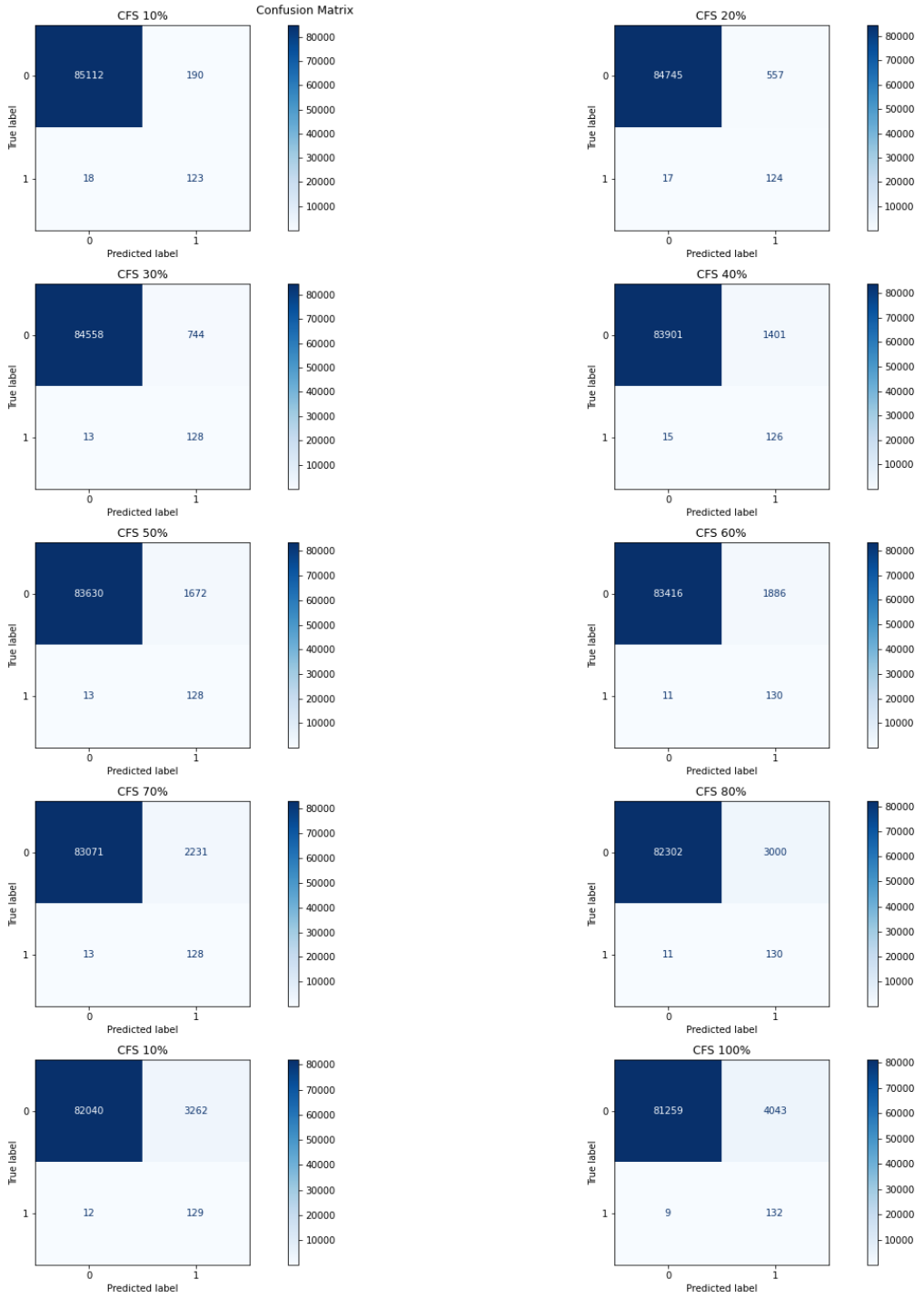

Figure 62: Confusion Matrix: XGB Random Undersampling

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.6, colsample_bytree = 0.6, max_depth = 6, alpha = 0, min_child_weight = 1, gamma = 0, learning_rate = 0.2, n_estimators = 100.

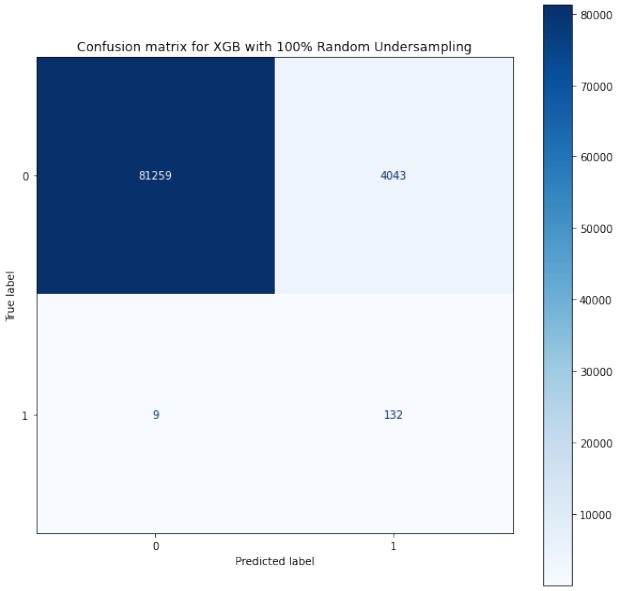

Figure 63: Confusion Matrix: XGB 100% Random Undersampling

Class	Precision	Recall	F1 Score
0	0.99989	0.95260	0.97567
1	0.03162	0.93617	0.06117
Avg	0.99829	0.95258	0.97416

Table 16: Classification Report: XGB 100% Random Undersampling

Like other models that used random undersampling, this model did a good job classifying legitimate cases but performed poorly when encountering the positive class. Its' precision score is extremely low at 0.03162 as the model misclassified a lot of genuine transactions as fraud. As a result, both the f1 score and the AP score of the model are rather low, 0,06117 and 0.73, respectively.

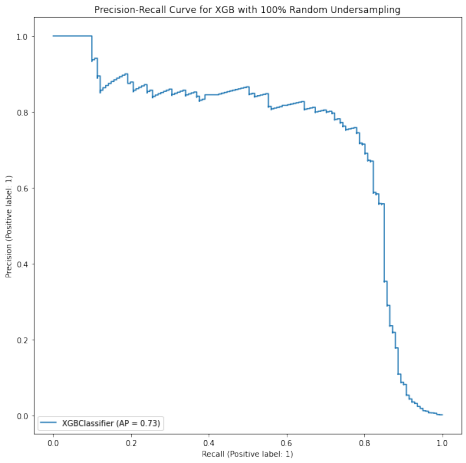

Figure 64: PR Curve: XGB 100% Random Undersampling

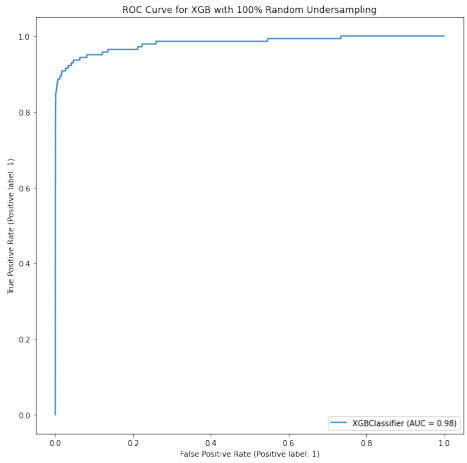

Figure 65: ROC Curve: XGB 100% Random Oversampling

4.2.3.4 SMOTE

Looking at figure 66, the 20% and 100% SMOTE-ed dataset gave us similar results in terms of true positive and false negative. However, the 100% SMOTE-ed dataset misclassified more genuine transactions as fraudulent, so we chose the 20% SMOTE-ed dataset as our training data for this model. A bigger view of this dataset's confusion matrix is shown in figure 67.

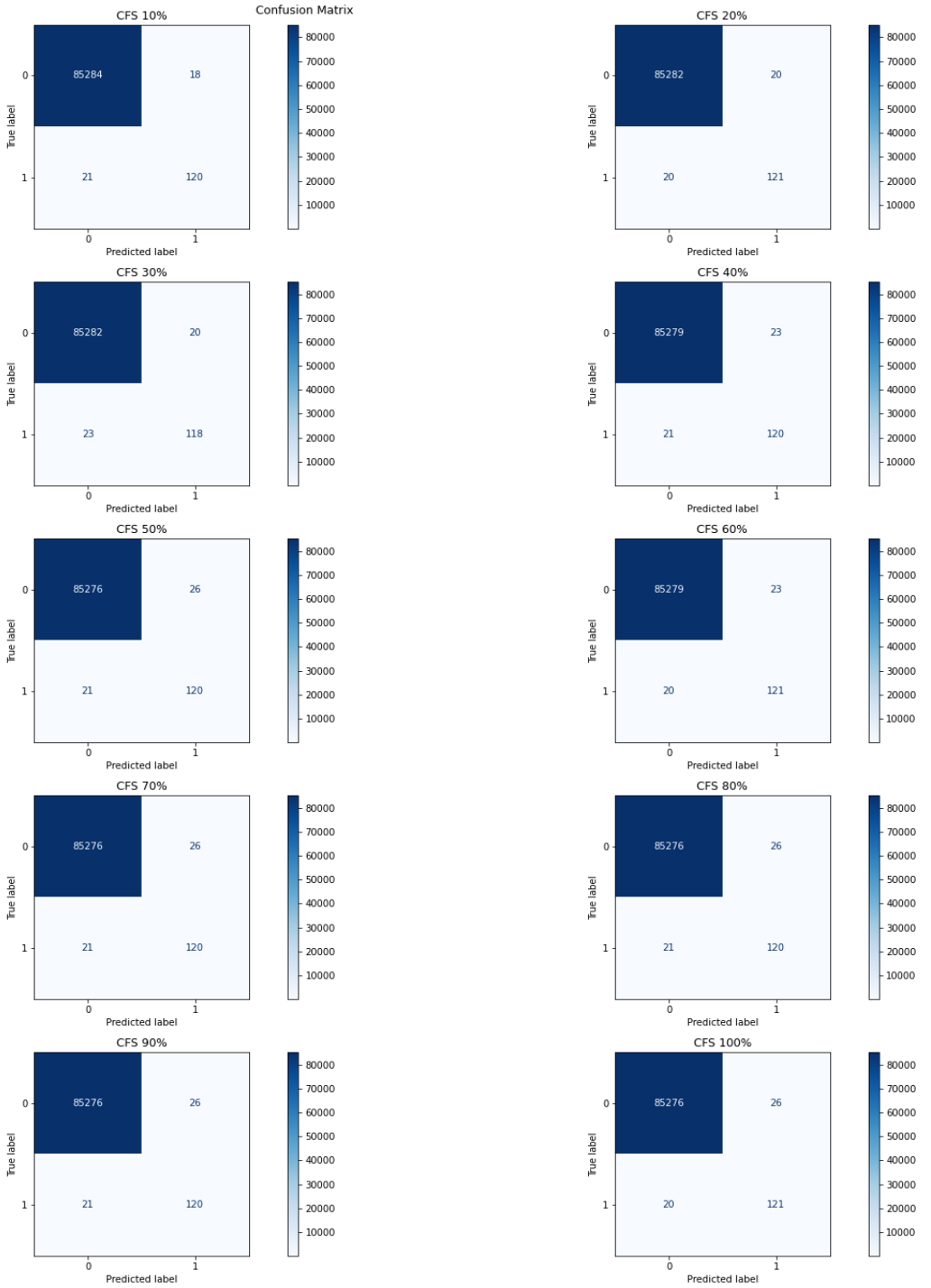

Figure 66: Confusion Matrix: XGB SMOTE

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.7, colsample_bytree = 0.9, max_depth = 10, alpha = 0, min_child_weight = 1, gamma = 0.1, n_estimators = 300, learning_rate = 0.3.

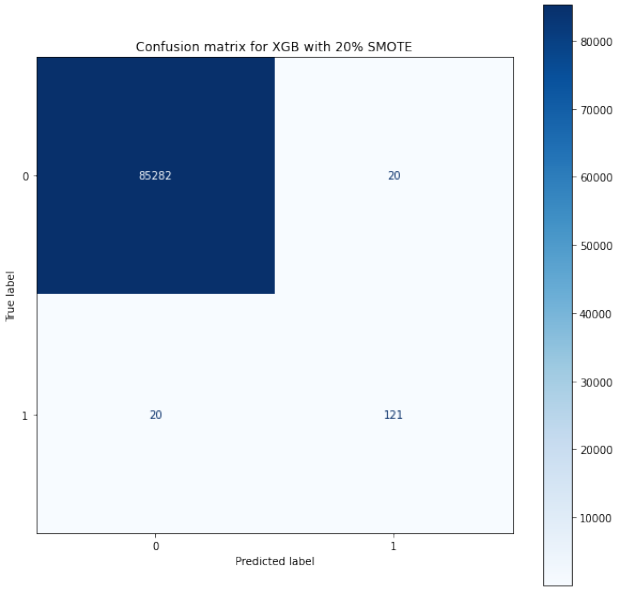

Figure 67: Confusion Matrix: XGB 20% SMOTE

Class	Precision	Recall	F1 Score
0	0.99977	0.99977	0.99977
1	0.85816	0.85816	0.85816
Avg	0.99953	0.99953	0.99953

Table 17: Classification Report: XGB 20% SMOTE

Looking at the model's metrics scores from table 17, we can see that the model did a good job in detecting fraud. It achieved high scores in all three metrics: precision, recall, and f1 score for both classes (0.99977 for class 0 and 0.85816 for class 1). The model also has an AP score of 0.88, which is considered high for a predictive model.

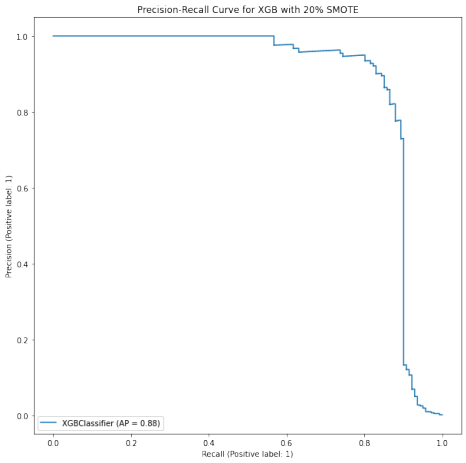

Figure 68: PR Curve: XGB 20% SMOTE

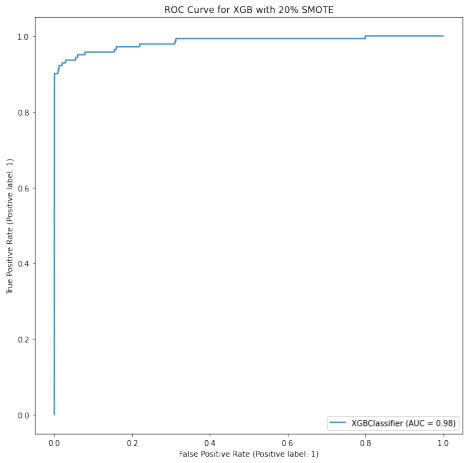

Figure 69: ROC Curve: XGB 20% SMOTE

4.2.3.5 Tomek Links Removal

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.7, colsample_bytree = 0.5, max_depth = 5, alpha = 0, min_child_weight = 1, gamma = 0.2, n_estimators = 700, learning_rate = 0.1.

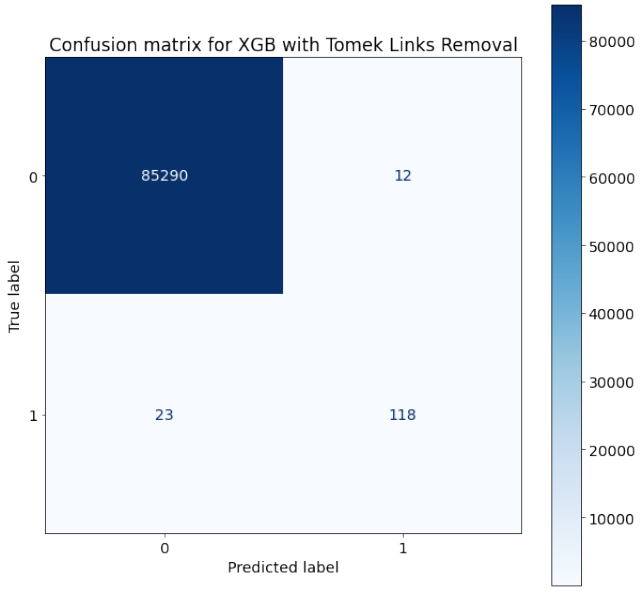

Figure 70: Confusion Matrix: XGB Tomek Links Removal

Class	Precision	Recall	F1 Score
0	0.99973	0.99986	0.99979
1	0.90769	0.83688	0.87085
Avg	0.99958	0.99959	0.99958

Table 18: Classification Report: XGB Tomek Links Removal

With the Tomek Links Removal resampling technique, the XGBoost model performed well on both classes, with the score of 0.90769, 0.83688, and 0.87085 for its precision, recall, and f1 score, respectively. In addition The model's AP and AUC are also rather at 0.87 and 0.99, as shown in figure 71 and figure 72.

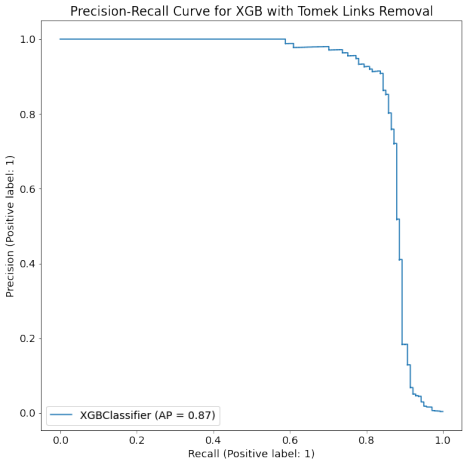

Figure 71: PR Curve: XGB Tomek Links Removal

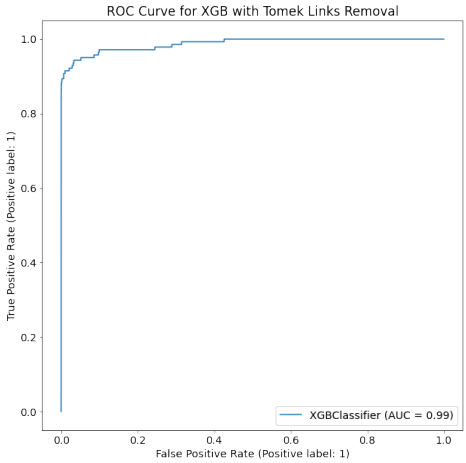

Figure 72: ROC Curve: XGB Tomek Links Removal

4.2.3.6 Hybrid Resampling

Similar to Logistic Regression and Random Forest models that used hybrid resampling, we implemented the Tomek Links Removal algorithm on the SMOTE-ed dataset that gave us the best performance, which is the 20% SMOTE-ed dataset.

After using GridsearchCV to tune the parameter for the model, the best parameters for the model are as followed: subsample = 0.5, colsample_bytree = 0.8, max_depth = 10, alpha = 0, min_child_weight = 1, gamma = 0.2, n_esimators = 700, learning_rate = 0.3.

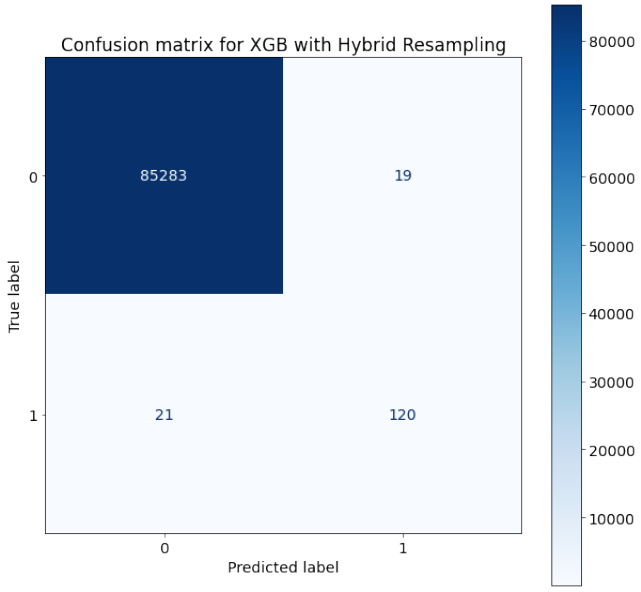

Figure 73: Confusion Matrix: XGB Hybrid Resampling

Class	Precision	Recall	F1 Score
0	0.99975	0.99978	0.99977
1	0.86331	0.85106	0.85714
Avg	0.99953	0.99953	0.99953

Table 19: Classification Report: XGB Hybrid Resampling

Table 19 shows the detailed metrics score of the model when trying to detect fraud in our test data. As we can see the model performed well in classifying both genuine and fraudulent transactions. When dealing with the negative class, the model's metrics scores were all over 0.999. When handling illegitimate transactions, the model achieved a high f1 score of 0.85714. The model also has an AP score of 0.85, as shown in figure 74.

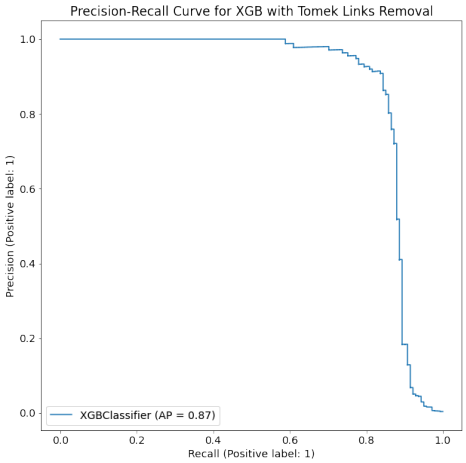

Figure 74: PR Curve: XGB Hybrid Resampling

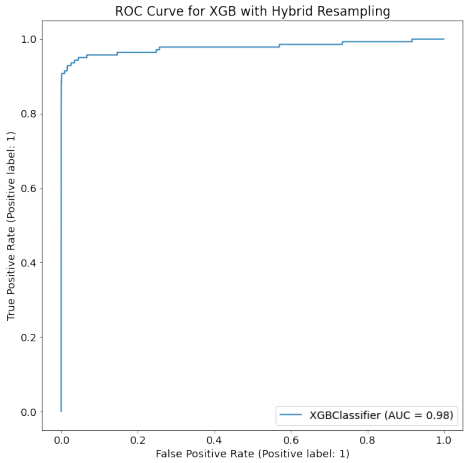

Figure 75: ROC Curve: XGB Hybrid Resampling

4.3 Result Summary

Table 20 summarizes all models' performance when detecting fraudulent transactions in terms of precision, recall, f1 score, AP, and AUC.

Classifier	Resampling Method	Precision	Recall	F1 Score	AP	AUC
Logistic Regression	No Resampling	0.84483	0.69504	0.76265	0.7	0.97
	Random Oversampling	0.05558	0.91489	0.10479	0.71	0.98
	Random Undersampling	0.02257	0.91489	0.04405	0.7	0.97
	SMOTE	0.10939	0.90071	0.19508	0.73	0.98
	Tomek Links Removal	0.81416	0.65248	0.72441	0.64	0.96
	Hybrid Resampling	0.08460	0.90780	0.15478	0.71	0.98
Random Forest	No Resampling	0.89922	0.82270	0.85926	0.86	0.96
	Random Oversampling	0.91406	0.82979	0.86989	0.87	0.97
	Random Undersampling	0.05430	0.92199	0.10256	0.71	0.97
	SMOTE	0.87050	0.85816	0.86429	0.87	0.98
	Tomek Links Removal	0.89231	0.82270	0.85609	0.85	0.95
	Hybrid Resampling	0.87050	0.85816	0.86429	0.85	0.97
XGBoost	No Resampling	0.91339	0.82270	0.86567	0.86	0.99
	Random Oversampling	0.89630	0.85816	0.87681	0.87	0.98
	Random Undersampling	0.03162	0.93617	0.06117	0.73	0.98
	SMOTE	0.85816	0.85816	0.85816	0.88	0.98
	Tomek Links Removal	0.90769	0.83688	0.87085	0.87	0.99
	Hybrid Resampling	0.86331	0.85106	0.85714	0.87	0.98

Table 20: Result Summary

4.4 Discussion

Table 20 clearly shows that Random Forest and XGBoost outperformed Logistic Regression even without any resampling technique. This indicates that ensemble models can give high performance in the presence of class imbalance. When Random Oversampling was used, both Random Forest, and XGBoost performed quite well with high metrics scores. It can be seen that, with Random Undersampling, all three models had a good recall score but failed terribly in terms of precision and f1 score. SMOTE improved recall in all models but slightly decreased the precision in Random Forest and XGBoost and Logistic Regression model's precision fell drastically. Compared with ensemble models that did not use any resampling technique, the Tomek Links Removal algorithm did not improve their performance much. Finally, when a combination of SMOTE and Tomek Links Removal was applied to the dataset, the results did not change much comparing to the models using

SMOTE. In addition, as we can see, the AUC differed very little between all models.

5 Conclusion

In this thesis, we implemented machine learning to classify whether a credit card transaction is legitimate or fraudulent. For the project, we used a public credit card dataset which was made available on Kaggle by a group of researchers from ULB. The dataset contains a total of 284,807 transactions, of which only 492 are fraudulent. The dataset is considered to be highly skewed as the positive class only accounts for 0.172% of the dataset.

Since the data is imbalanced, when a model that was trained on this dataset would be biased toward the majority class. As a result, the model can easily mistake fraudulent transactions as genuine. In order to tackle this problem, we applied various resampling techniques to the dataset, namely, Random Oversampling, Random Undersampling, SMOTE, Tomek Links Removal and a hybrid approach which is the combination of SMOTE and Tomek Links Removal. Along with these resampling techniques, we also trained different predictive models, such as Logistic Regression, Random Forest, and XGboost, to compare their performance with each other. After that, we analyzed the results of all three models with and without any resampling methods. The result indicates that Random Forest combined with SMOTE performed better than others in terms of precision and recall balance, as well as AP score.

5.1 Difficulties

During the span of the project, we encountered various difficulties, from which we learnt valuable lessons for our future projects.

The main problem that we encounter was the unexpected time consumption in the training phase. After we trained the Logistic Regression models, we assumed that the time it took to tune the parameters and train the ensemble models. As the *sklearn* module does not use GPU for training, when we trained Random Forest models, it would take up to 12 hours to tune the parameters for the model.

In addition to our main problem, we also experienced technical difficulties. As we found a way to improve the training time which is multithreading, we did not have the hardware that was capable of running all cores for a long time. We attempted to used Google Colab; however, Google Colab only provided CPU with 2 cores and it usually timed out before the training is

complete. As a results we were only to training on multiple cores occasionally.

Furthermore, as XGBoost had many parameters to tune and it would take days to tune them even with parallel training on multiple cores, we only managed to refine the hyperparameters one by one instead of all the hyperparamters combination.

Moreover, due to the sensitivity of the problem, not too many datasets are available for training. In order to keep the customer's information confidential, most attributes of the dataset that we used were masked by PCA, which made us difficult to evaluate the attributes separately.

5.2 Future Work

Credit card fraud is always changing and fraudsters are constantly coming up with new ways to attempt fraudulent activities. Therefore, it would be essential to take the users' behaviors into account, such as spending limits, spending categories, etc. For future work, a detailed research on how to includes behavior tracking into our models can be performed; however, it would require a big dataset to generalize a pattern for all the customers.

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