Data Science Final Project Report
CS-5661

(Spring – 2022)

Credit Card Fraud Detection

Neha Das (CIN – 401457144)
Jesse Rodriguez (CIN – 304679367)
Naga Siri Chandana Vankina (CIN – 401317264)
Vaibhavi Buddhiraju (CIN – 401973478)
Ashkan Aledavoud (CIN – 304333762)

Abstract

The project is proposed to detect the Credit card fraudulence so that the customers of credit card companies are not charged for items that they did not purchase. This is due to the rise in both online transactions and e-commerce platforms. Sometimes fraud and scams are in pennies, for which millions of them are not reported, so that falls into misclassified data, which makes the job more complex. In the present world, we are facing a lot of credit card problems. This project aims to focus mainly on machine learning algorithms. The algorithms used are K-Nearest Neighbour (KNN), Decision Trees Classifier, Random Forest Classification, Logistic Regression, Artificial Neural Network, Naïve Bayes and Support Vector Machine. The results of the seven algorithms are based on accuracy, precision, recall and the ROC-AUC curve. The seven algorithms are compared and the algorithm that has the greatest accuracy, precision, recall is considered as the best algorithm that is used to detect the fraud.

Introduction

Credit card fraud detection is defined as a theft or a fraud which involves stealing money from someone else credit card or purchasing goods without paying. This may be involved while using credit cards for online banking, net-banking, buying goods and services or creating investment portfolio.

Due to rapid acceleration of E-commerce, there is tremendous use of credit cards for online shopping which leads to credit card fraud and scam. In the era of digitalization, we need to detect the credit card fraud detection. We have various types of credit card fraud some of them are owners' loss or stolen cards, credit care is cloned or duplicated with a special swipe machine, card details such as card number, date of birth, etc., that are stolen from online database which are sold and used on the internet and last one is committing fraudulent application means buying new credit card with someone else's name.

Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behaviour, which consist of fraud, intrusion, and defaulting. Some of the main challenges involved in credit card fraud detection are:

1. Keeping in view of the fraudulent transactions and scams happening in real-time, huge data is needed to be experimented every day to make the model successful.
2. As per the statistics, 99.98% of transactions are reported as not fraudulent, whereas they contain scams and other frauds, which makes the fraud detection very difficult.
3. Collecting data from different resources is mostly inaccessible, since the data is very secure and most of them are private.
4. Sometimes fraud and scams are in pennies, for which millions of them are not reported, so that falls into misclassified data, which makes the job more complex.

Scammers are aware of technologies available, so our goal is to understand the highest probability of classification by using different techniques that makes the job with highest accuracy a good solution to prevent the frauds.

Machine Learning algorithm are deployed to analyse the fraudulent transaction and report the fraud. After this the professionals investigate the report and contact the cardholders to confirm whether the transaction is genuine or fraudulent.

The code work and the methods used to complete this project has been shared in a virtual link which is available here: https://github.com/NehaDas25/DataScience_CS5661.git

System Requirements

1. Collection of datasets which is of size 143 MB.
2. The dataset contains numerical inputs variables.
3. System- PC(Windows)- 16 GB RAM, 64-bit OS
4. Software – Anaconda3, Jupyter Notebook

Implementation Procedure

1. Import all the libraries.
2. Since the dataset is too huge (143 MB), the .csv file is converted to zip file.
3. In Jupyter Notebook, the .csv file is unzipped, the dataset is loaded and previewed.
4. The dataset is unbalanced. For more clarity plot contor plot of the unscaled vectors.
5. Plot the correlation table using heatmap to check the feature are scaled or not.
6. Sample the dataset using Random & NearMiss Undersampling and scale the features using MinMaxScaler.
7. Extract the feature column and the target. Split the data for X_train, y_train, X_test, y_test with test size and random state.
8. Use K-Nearest Neighbour (KNN), Decision Trees Classifier, Random Forest Classification, Logistic Regression, Artificial Neural Network, Naïve Bayes and Support Vector Machine Algorithms to fit and predict the model for both Undersampling approaches.
9. Find Accuracy, Precision, Recall and AUC- ROC curve for both Undersampling approaches to find which model best suit for predicting Fraud.
10. Plot the ROC- curves and find AUC.
11. Plot the Confusion Matrix's.
12. Observations from the project are taken to find the best algorithm for each undersampling approach through histogram.

About Dataset

The dataset is taken from Kaggle.com and as per Kaggle it is mentioned that the dataset contains transactions made by credit cards in September 2013 by European cardholders.

Unfortunately, due to confidentiality issues, they cannot provide the original features and more background information about the data. The dataset is of size 143 MB. It contains credit card transactions made by Credit cardholders. The dataset comprises of 284,807 transactions, with the features V1, V2, V3, V4, ..., V28 that are obtained using PCA (Principal Component Analysis). Time column is the feature that indicates the duration between every transaction occurring and the first transaction.

Amount column is the feature that contains the transacted amount. Class contains either 0 or 1, where 1 is referred to fraud and 0 for no fraud. The dataset has 2,84,315 genuine transaction and 492 fraud transaction which shows that the dataset is highly Imbalanced.

Data Analysis

We saw from the dataset while visualizing, there appears to be a huge difference between the number of transactions. From the description of the dataset, we already know that the dataset is imbalanced. So, let's check with how many cases of difference we see between genuine and fraud transactions through SNS countplot.

SNS countplot shows very huge difference between number of genuine and fraudulent transactions in respect to the classes as seen from the count and sns count plot. The following visualization underlines this significant contrast.

Figure 1: Plot for Genuine vs Fraud Transaction

From the above plot we could see the dataset is very much imbalanced. So now what is Imbalanced dataset? The definition states that Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e., one class label has a very high number of observations and the other has a very low number of observations.

Lets' look at the plot of Amount and Time feature:

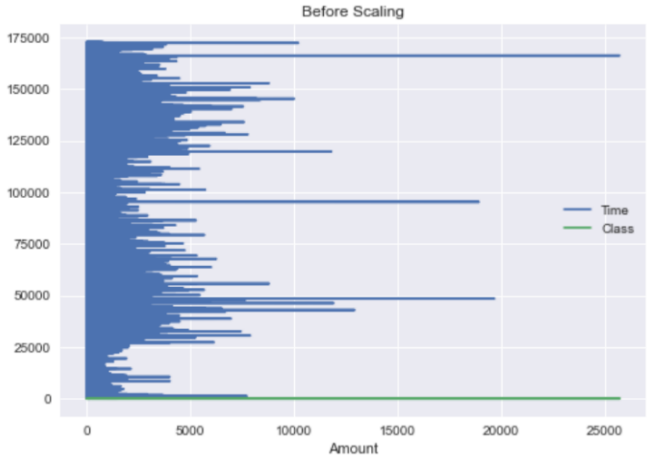


Figure 2: Plot for Time, Amount with respect to Class

From the above visualization, Feature Amount and Time are not scaled.

So now to make the two-dimensional feature data more replicative we will be using HeatMap, because HeatMap provides the best intermediary visualization of the features/vectors in the dataset and can help us in understanding the complex datasets.

It would be interesting to know if there are any significant correlations between our features. **Correlation** is a table that shows the relation between 2 features. It helps the data to understand more deeply. One of the most visually appealing ways to determine is by using a heatmap.

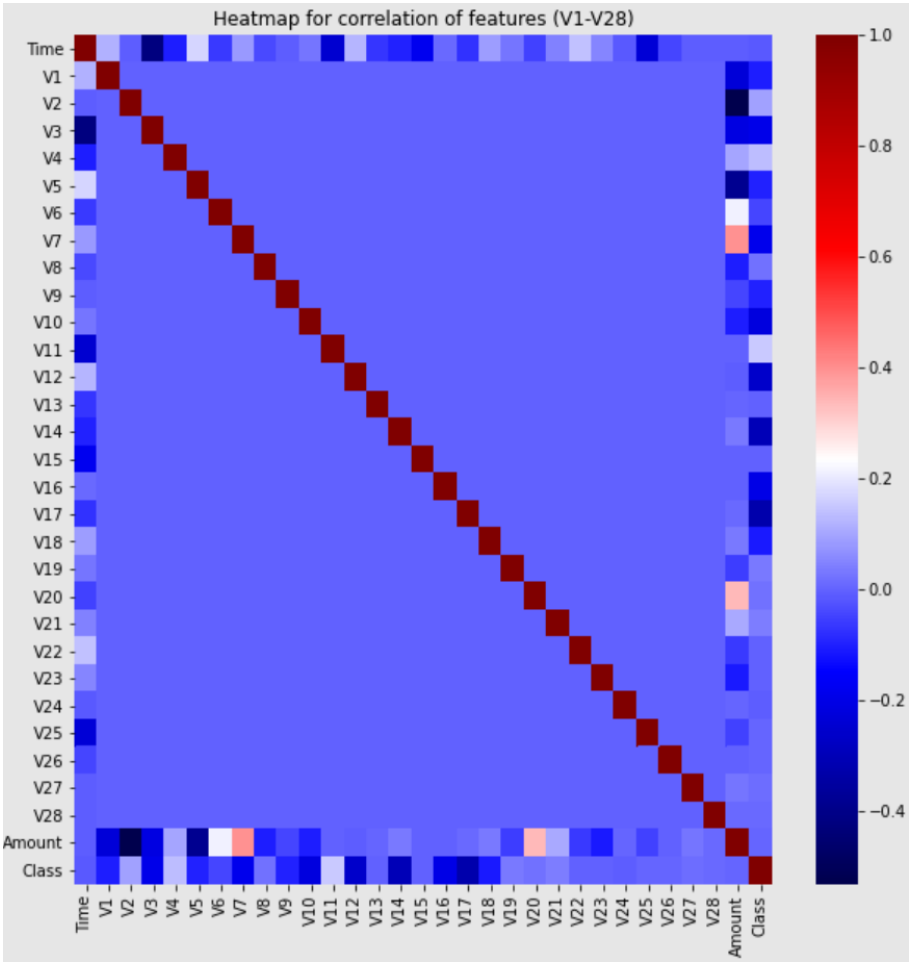


Fig: 3 Heatmap for correlation for features (V1-V28)

From the above visualization, we can observe that the highest correlations are between Time and V3[< -0.4], Amount and V2[< -0.4], Amount and V4[0.19]. Also, the features V1 - V28 somehow have close inter-connection with each other.

But the **class** feature has some positive and negative correlation with all V-features, whereas it does not have any correlation with the Feature Time and Amount. It also clearly indicates that V features are scaled but Time and Amount are not scaled.

Data Preparation

From the above analysis, it is important not to forget that the V-features are scaled and seem to centre around zero whereas feature Time and Amount are not scaled. Not scaling the data would result certain machine learning algorithm performing worse.

For this project, we considered using **MinMax scaler**, since we saw from the above results that correlation between the features is specific between [-1,1] and MinMax Scaler transforms each of the feature values by scaling them with the given range as provided. That can transform the specific features of dataset based on the range we provide.

Here, MinMax scaler is used to scale two features Amount and Time based on Minimum and Maximum value. From the above figure 2, we clearly saw the amount ranges between [0-25000], whereas Time ranges between [0-175000]. To undermine this data, we thought of a way to eliminate the median values, instead consider the minimum and maximum values of the dataset. After applying the MinMax scalar, we must drop the Time and Amount feature and renamed it as Scaled Amount and Scaled Time in the dataset.

```
minmax_scale = MinMaxScaler()

data_set['scaled_amount']=minmax_scale.fit_transform(data_set['Amount'].values.reshape(-1,1))
data_set['scaled_Time']=minmax_scale.fit_transform(data_set['Time'].values.reshape(-1,1))

data_set.drop(['Amount','Time'], axis=1,inplace=True)
```

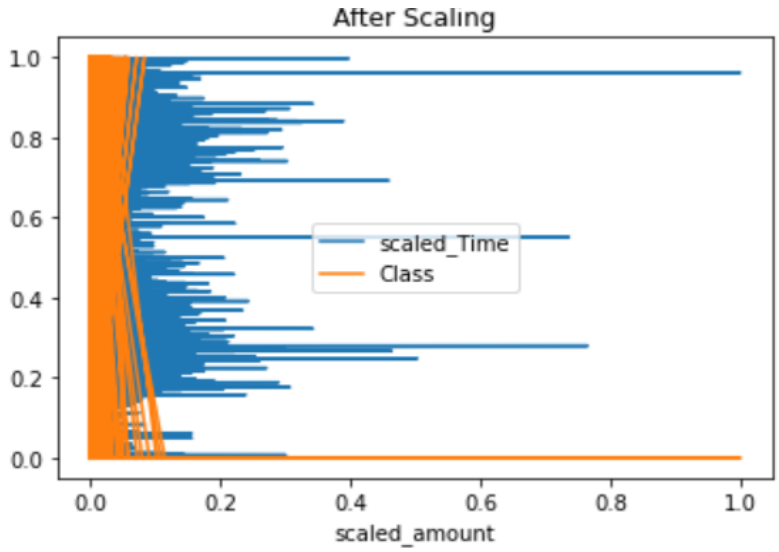


Fig 4: Plot After scaling the Time and Amount

From this plot it is seen that both time and amount are in the range between [0-1].

So, the old time and amount with the new scaled amount ands time. These two properties will be playing a vital role in handling of imbalanced dataset in the next part. Also scaling the properties gives better results while using the dataset with algorithms to predict the accuracy, precision, etc.

Let’s look at the heatmap correlation after scaling Time and Amount.

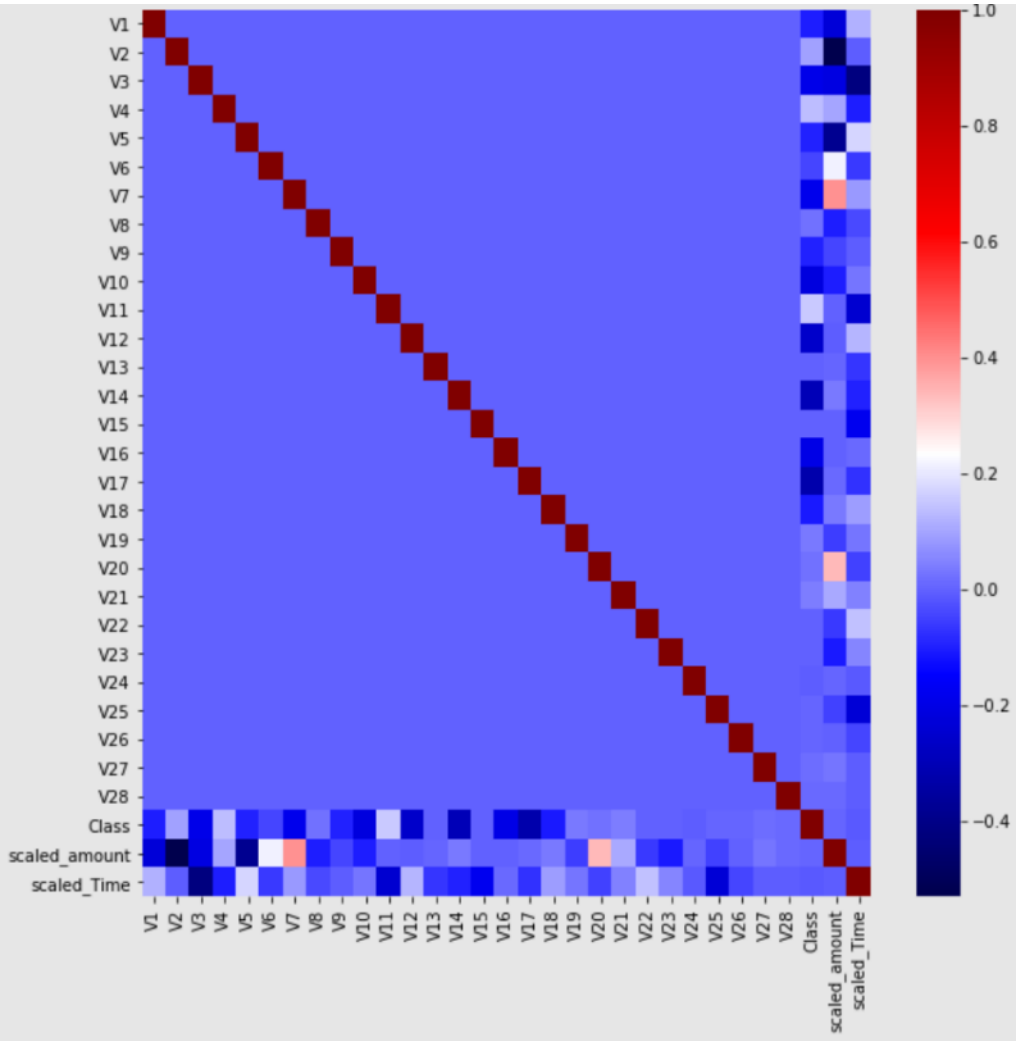


Figure 5: After Scaling Heatmap

Handling Imbalanced Dataset

Create a training dataset that will allow our algorithms to pick up the specific characteristics that make a transaction more or less likely to be fraudulent. But using the original dataset would not be the good idea as from Fig:1 we can see that there is 284315 Genuine transaction 492 Fraud transaction and for this reason we want to detect the fraudulent case and label them. There are two types of sampling Technique Undersampling and Oversampling.

Oversampling - This is the method the randomly replicates the minority samples in order to balance the dataset but there is high possibility of overfitting the data since it replicates the minority samples. In this case we are using Undersampling.

Undersampling – We will present two approaches to Undersampling.

First Method, this is a technique to balance uneven datasets by keeping all the data in the minority class and decreasing the size of the majority class. This method is best to use when the size of the training data is huge. It is one of several techniques data scientists can use to extract more accurate information from originally imbalanced datasets.

First, we will shuffle the whole dataset with random state=4 and extract fraud ones onto a different dataset which is under the variable fraud dataset as we have 492 fraud cases. Now we have to select 492 random observations from 284315 genuine observations (majority class). Then concat both fraud dataset and non_fraud_dataset as in previous steps into a normalized one which is stored in the variable called normalized_data_set. [Figure 6]

```
# Shuffle the Dataset.
shuffled_data_set = data_set.sample(frac=1,random_state=4)

# Put all the fraud class in a separate dataset.
fraud_data_set = shuffled_data_set.loc[shuffled_data_set['Class'] == 1]

#Randomly select 492 observations from the genuine (majority class)
genuine_data_set = shuffled_data_set.loc[shuffled_data_set['Class'] == 0].sample(n=492,random_state=42)

# Concatenate both dataframes again
normalized_data_set = pd.concat([fraud_data_set, genuine_data_set])
```

Figure 6: Performing Undersampling for Imbalanced dataset

Second Method for Undersampling is called NearMiss.

There are three versions of NearMiss, but for the purpose of this project we are using version 2. In simplest terms the method will balance the dataset by looking at the class distribution and removing samples from the Majority class randomly.

The NearMiss library uses KNN to sample points from the majority class (perform Undersampling).

Using version 2 of NearMiss, our dataset will be balanced by calculating the average minimum distance between the Majority class and 3 (N furthest samples) of the minority class and selects the smallest (like shown in Figure 7).

Only the instance that have the shortest distance are chosen like shown in Figure 8.

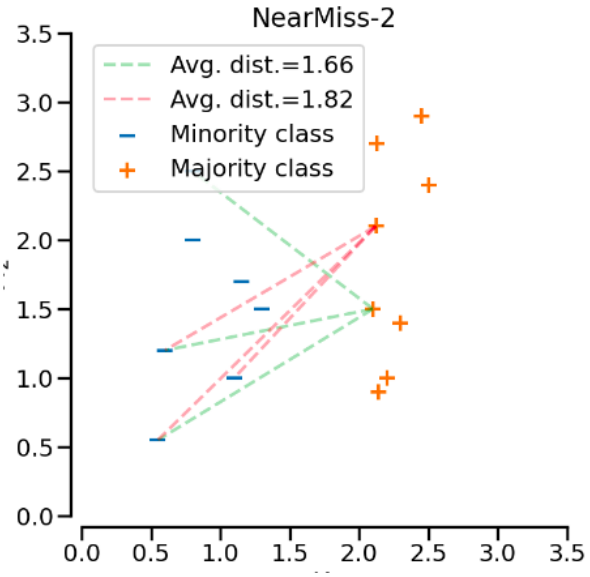


Figure 7: NearMiss-2

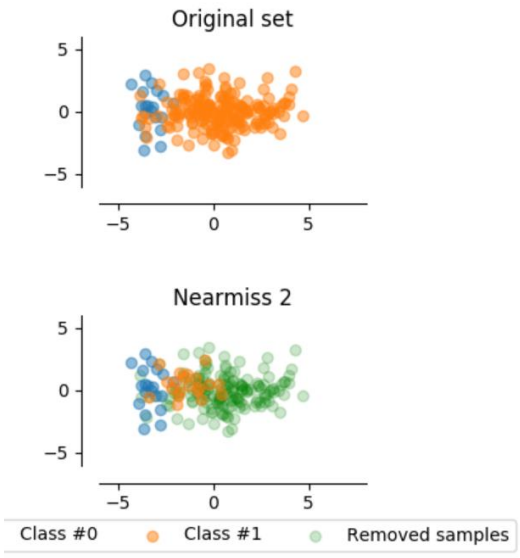


Figure 8: removing majority samples

To perform NearMiss we simply import NearMiss from the imblearn library, we then initialize NearMiss using version 2 and perform fit_resample to both the Features and the Target.

```
[31]: from imblearn.under_sampling import NearMiss
      nearmiss = NearMiss(version=2)
      X_nearmiss, y_nearmiss = nearmiss.fit_resample(X_for_nearmiss, Y_for_nearmiss)
```

Figure 9: imblearn.under_sampling library

By Plotting our data before and after Undersampling, we can see the difference in the Majority Class. The number of samples in the Minority remained the same but looking at figure 11, we can see that the instance with the largest distances were removed.

```
plt.scatter(X[y == 0, 0], X[y == 0, 1], label="Genuine", alpha=0.5, linewidth=0.2)
plt.scatter(X[y == 1, 0], X[y == 1, 1], label="fraud", alpha=0.5, linewidth=0.2, c='r')
```

Figure 10: scatter plot parameters

Before Undersampling

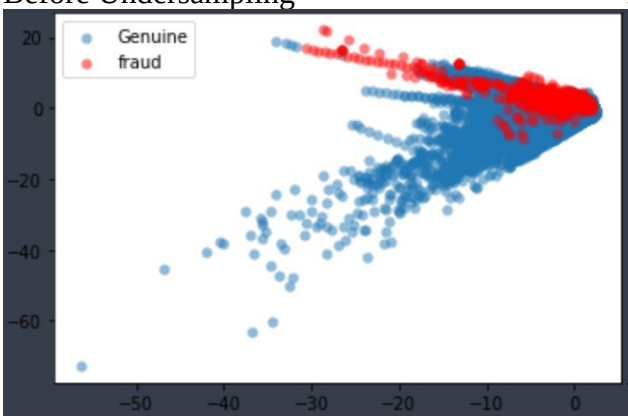


Figure 10: scatter plot before NearMiss

After NearMiss Undersampling

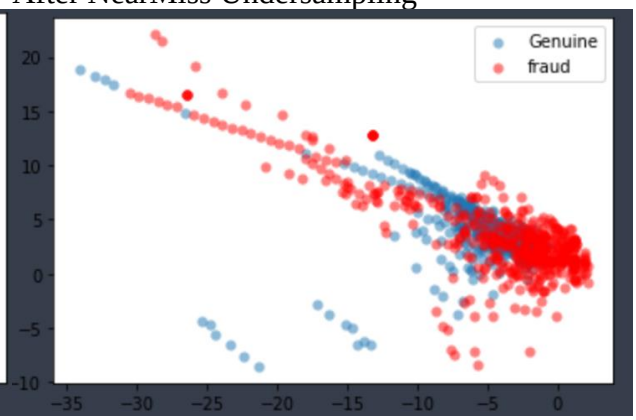


Figure 11: scatter plot After NearMiss

For both approaches the number of minority samples will be the same, hence after Undersampling both graphs are also the same.

The total number of samples after Undersampling is 984

- Number of fraud data is 492
- Number of Genuine data is 492

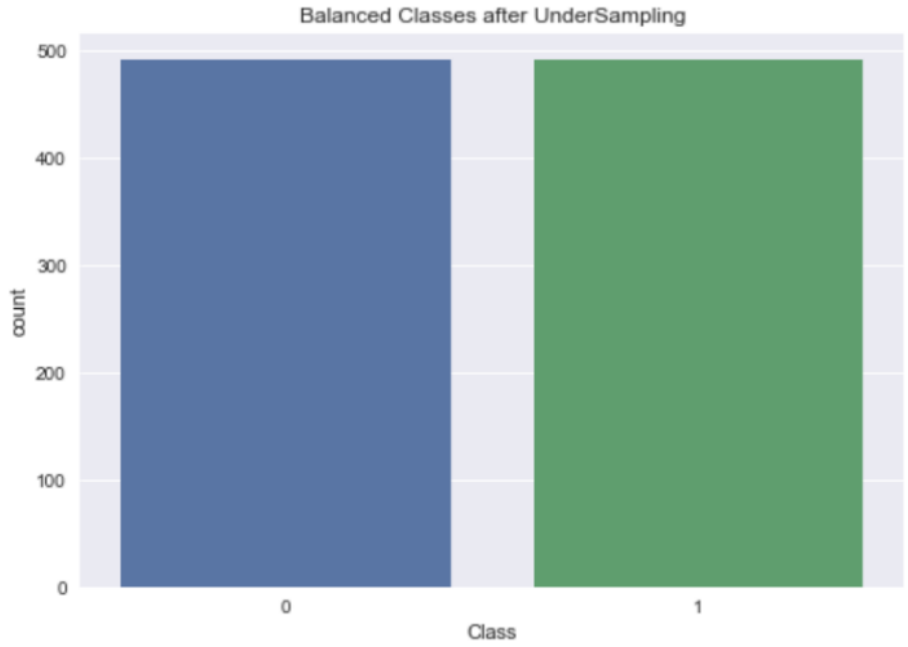


Figure 12: Plot for balanced dataset

Implementing Algorithms

- Before training our data, we must extract the Feature column and Target column from the 'normalized_data_set' and 'dataset' (NearMiss).
- We have dropped the Class column from 'normalized_data_set' and 'dataset' and considered it as target column.

Divide data into training and testing sets

- Here X is labelled as Feature and Y is labelled as Target.
- Using the machine learning library "Sklearn", we split the data into training and testing sets with test_size of 0.3 and random_state of 42.

```
# Split the data for X_train,X_test,y_train,y_test with test_size and random state
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

Figure 13: Performing the splitting of Dataset into training and testing set

Classifiers

We used 7 classifiers, for these algorithms we calculated accuracy score using metrics from sklearn. The precision_score, recall_score, and f1_score was also calculated and then we have computed the AUC, ROC-curve and Confusion matrix

Using the split data sets to train and test these algorithms, these were their score results.

1. **K-Nearest Neighbor**: The K-Nearest Neighbor is the simple, supervised machine learning machine algorithm that can be used to solve both classification problems. It classifies the object based on k closest training samples in the feature space, e.g., k=3. For every possible point in the space, we can determine its label using the KNN rule. This gives a decision boundary that partitions the space into different regions.

```
# 1. KNeighborsClassifier with k as an input to the classifier.
k = 3
knn_algorithm = KNeighborsClassifier(n_neighbors=k)
knn_algorithm.fit(X_train, y_train)
y_predict_knn = knn_algorithm.predict(X_test)
y_predict_probab_knn = knn_algorithm.predict_proba(X_test)
knn_accuracy = accuracy_score(y_test, y_predict_knn)
print("Accuracy score for KNN=",knn_accuracy*100)
pre_knn = precision_score(y_test,y_predict_knn)
print("Precision score for KNN=",pre_knn*100)
recall_knn = recall_score(y_test,y_predict_knn)
print("Recall score for KNN=",recall_knn*100)
f1_knn = f1_score(y_test,y_predict_knn)
print("F1 score for KNN=",f1_knn*100)
```

Figure 14: Performing KNN Algorithm

KNN	Accuracy score	Precision score	Recall score	F1 score
Random undersampling	93.58108%	95.80419%	91.33333%	93.51535%
NearMiss Undersampling	98.64864%	99.30555%	97.94520%	98.62068%

2. **Decision Tree Classifier:** Decision Tree is a supervised Machine learning algorithm. It uses a flowchart like a tree structure to show predictions that result from a series of feature-based splits. While training this model we need to know three things:
    (a) The structure of tree: the priority of features.
    (b) The threshold values.
    (c) The value of leaves.

    So, the best feature is one that provides the most amount information about the label. Here we need a metric to measure information called Entropy.
    Here is how we are training our dataset:

```
# 2. Decision Tree Classifier
decisiontree_algorithm = DecisionTreeClassifier()
decisiontree_algorithm.fit(X_train, y_train)
y_predict_dt = decisiontree_algorithm.predict(X_test)
y_predict_probab_dt = decisiontree_algorithm.predict_proba(X_test)
dt_accuracy = accuracy_score(y_test, y_predict_dt)
print("Accuracy score for Decision Tree=",dt_accuracy*100)
pre_dt = precision_score(y_test,y_predict_dt)
print("Precision score for Decision Tree =",pre_dt*100)
recall_dt = recall_score(y_test,y_predict_dt)
print("Recall score for Decision Tree =",recall_dt*100)
f1_dt = f1_score(y_test,y_predict_dt)
print("F1 score for Decision Tree=",f1_dt*100)
```

Figure 15: Performing Decision Tree Classifier Algorithm

Decision Tree	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	90.54054%	89.61038%	92%	91.80%
NearMiss Undersampling	91.55405%	90.32258%	93.33333%	91.80327%

3. **Random Forest Classifier:** Random Forest is a supervised learning algorithm that is used widely in Classification and Regression. It builds a forest of an ensemble of decision trees which adds more randomness while growing the trees. Here we are using the concept of Ensemble learning and the bootstrapping. It searches for best features from the random features which adds more diversity, that results in better model. After voting, the trees will compensate for each other mistakes and provide best results. Random Forest classifier will handle the missing values and maintain the accuracy of a large proportion of data**.**

```
# 3. Random Forest Classifier
# Here we will use the random state that's already used while splitting the dataset
randomForest_algorithm = RandomForestClassifier(n_estimators=19, bootstrap = True, random_state=42)
randomForest_algorithm.fit(X_train, y_train)
y_predict_rf = randomForest_algorithm.predict(X_test)
y_predict_probab_rf = randomForest_algorithm.predict_proba(X_test)
rf_accuracy = accuracy_score(y_test,y_predict_rf)
print("Accuracy score for Random Forest Classifier=",rf_accuracy*100)
pre_rf = precision_score(y_test,y_predict_rf)
print("Precision score for Random Forest=",pre_rf*100)
recall_rf = recall_score(y_test,y_predict_rf)
print("Recall score for Random Forest=",recall_rf*100)
f1_rf = f1_score(y_test,y_predict_rf)
print("F1 score for Random Forest=",f1_rf*100)
```

Figure 16: Performing Random Forest Classifier

Random Forest	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	93.24324%	93.91891%	92.66666%	93.28859%
NearMiss Undersampling	99.32432%	99.31506%	99.31506%	99.31506%

4. **Logistic Regression:** Logistic Regression is a supervised learning classification algorithm used to predict the probability of a target variable. It uses a more complex cost function; this cost function can be defined as the “Sigmoid” or also known as the ‘logistic function’ instead of a linear function. The logistic function tends it to limit the cost function between 0 and 1. If we try to use the cost function of the linear regression in ‘Logistic Regression’ then it would be of no use as it would end up being non- convex function. The cost value function can be reduced using the Gradient Descent.

```
# 4. Logistic Regression
logreg_algorithm = LogisticRegression(solver='liblinear')
logreg_algorithm.fit(X_train, y_train)
y_predict_lr = logreg_algorithm.predict(X_test)
y_predict_probab_lr = logreg_algorithm.predict_proba(X_test)
lr_accuracy = accuracy_score(y_test, y_predict_lr)
print("Accuracy score for Logistic Regression=",lr_accuracy*100)
pre_lr = precision_score(y_test,y_predict_lr)
print("Precision score for Logistic Regression=",pre_lr*100)
recall_lr = recall_score(y_test,y_predict_lr)
print("Recall score for Logistic Regression=",recall_lr*100)
f1_lr = f1_score(y_test,y_predict_lr)
print("F1 score for Logistic Regression =",f1_lr*100)
```

Figure 17: Performing Logistic Regression

Logistic Regression	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	94.59459%	96.52777%	92.66666%	94.55782%
NearMiss Undersampling	93.24324%	93.75%	92.46575%	93.10344%

5. **Artificial Neural Network:** Artificial Neural Network is a family of Machine learning systems/models, inspired by human's nervous system, especially by brain. Artificial Neural Network can learn from data and generate models that get a number of inputs and map them to desired output in an optimal way. Signals are sometimes modified at the receiving synapse and the weighted inputs are summed at the processing element. If it crosses the threshold, it goes as input to other neurons and process repeat. There are 3 main components of a Neural network: the input layer, the processing layer and the output layer. The inputs may be weighted based on various criteria. This is the output of first unit of hidden layer 1.

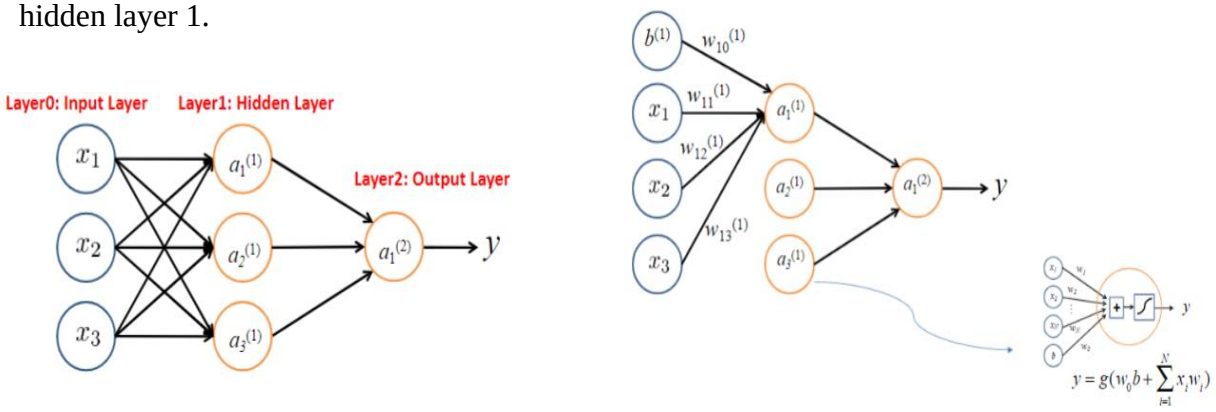


Figure 18: ANN with the 3 layers and the output of hidden layer 1

```
# 5. Artificial Neural Network
# 1 Hidden Layer with 200 neurons:
ANN_Algorithm = MLPClassifier(hidden_layer_sizes=(200), activation= 'logistic',
                    solver='sgd', alpha=1e-5, random_state=1,
                    learning_rate_init = 0.1, max_iter=10000, verbose=True, tol=0.0001)
ANN_Algorithm.fit(X_train, y_train)
# Weights:
print(ANN_Algorithm.coefs_)
# The ith element in the list represents the weight matrix corresponding to layer i.
print('\n')
# Bias weights:
print(ANN_Algorithm.intercepts_)
# The ith element in the list represents the bias vector corresponding to layer i + 1.
# Testing on the testing set:
y_predict_ann = ANN_Algorithm.predict(X_test)
y_predict_probab_ann = ANN_Algorithm.predict_proba(X_test)
print(y_predict_ann)
ann_accuracy = accuracy_score(y_test, y_predict_ann)
print("Accuracy score for Artificial Neural Network=",ann_accuracy*100)
pre_ann = precision_score(y_test,y_predict_ann)
print("Precision score for Artificial Neural Network=",pre_ann*100)
recall_ann = recall_score(y_test,y_predict_ann)
print("Recall score for Artificial Neural Network=",recall_ann*100)
f1_ann = f1_score(y_test,y_predict_ann)
print("F1 score for Artificial Neural Network=",f1_ann*100)
```

Figure 19: Performing Artificial Neural Network Algorithm

ANN	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	95.27027%	96.57534%	94.0%	95.27027%
NearMiss Undersampling	98.64864%	100.0%	97.26027%	98.61111%

6. **Naïve Bayes:** Naïve Bayes is a supervised learning algorithm based on applying Bayes' theorem with the naïve assumption of conditional independence between every pair of features given the value of class variable. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.

```
# 6. Naive Bayes
naiveBayes_algorithm = GaussianNB()
naiveBayes_algorithm.fit(X_train, y_train)
y_predict_naiveBayes  =  naiveBayes_algorithm.predict(X_test)
y_predict_probab_naiveBayes = naiveBayes_algorithm.predict_proba(X_test)
naiveBayes_accuracy = accuracy_score(y_test, y_predict_naiveBayes)
print("Accuracy score for Naive Bayes model = ", naiveBayes_accuracy*100)
pre_nb = precision_score(y_test,y_predict_naiveBayes)
print("Precision score for Naive Bayes=",pre_nb*100)
recall_nb = recall_score(y_test,y_predict_naiveBayes)
print("Recall score for Naive Bayes=",pre_nb*100)
f1_nb = f1_score(y_test,y_predict_naiveBayes)
print("F1 score for Naive Bayes=",f1_nb*100)
```

Figure 20: Performing Naïve Bayes Algorithm

Naïve Bayes	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	91.21621%	97.69230%	97.69230%	90.71428%
NearMiss Undersampling	88.17567%	92.36641%	92.36641%	87.36462%

7. **Support Vector Machine:** Support Vector Machine is a supervised machine learning algorithm that is used for both the classification and regression. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data point. This algorithm finds the closest point of lines from both the classes specified. These points are called support Vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane. Here rbf kernel is used to store the support vectors during training and not the entire dataset.

```
#7. Support Vector Machine Classifier
svm_algorithm = SVC(kernel='rbf', probability=True, random_state = 1)
svm_algorithm.fit(X_train, y_train)
y_predict_svm = svm_algorithm.predict(X_test)
y_predict_probab_svm = svm_algorithm.predict_proba(X_test)
svm_accuracy = accuracy_score(y_test, y_predict_svm)
print("Accuracy score for Support Vector Machine = ", svm_accuracy*100)
pre_svm = precision_score(y_test,y_predict_svm)
print("Precision score for Support Vector Machine=",pre_svm*100)
recall_svm = recall_score(y_test,y_predict_svm)
print("Recall score for Support Vector Machine=",recall_svm*100)
f1_svm = f1_score(y_test,y_predict_svm)
print("F1 score for Support Vector Machine=",f1_svm*100)
```

Figure 21: Performing Support Vector Machine

SVM	Accuracy score	Precision score	Recall score	F1 score
Random Undersampling	94.25675%	97.84172%	90.66666%	94.11764%
NearMiss Undersampling	96.28378%	93.54838%	99.31506%	96.34%

Confusion Matrix

Now that we trained and tested each classifier algorithms.

Now that the Classifier algorithms have been trained and tested for both approaches, we will then summarise how each method performed on the testing data.

One method is creating a Confusion Matrix for each method. A confusion / error matrix is a table layout that allows us to visualize and analyse the performance of a classifier.

The columns correlate with the predictions the current algorithm being analysed made.
The Rows correlate with the true value.

True positive (TP) = Correctly predicted as fraudulent and true value is fraudulent.
True negative (TN) = Classifier predicted not fraudulent and true value is not fraudulent.
False Negative (FN) = Classifier predicted not fraudulent but in fact it is fraudulent
False Positive (FP) = Not fraudulent but the classifier predicted as fraudulent

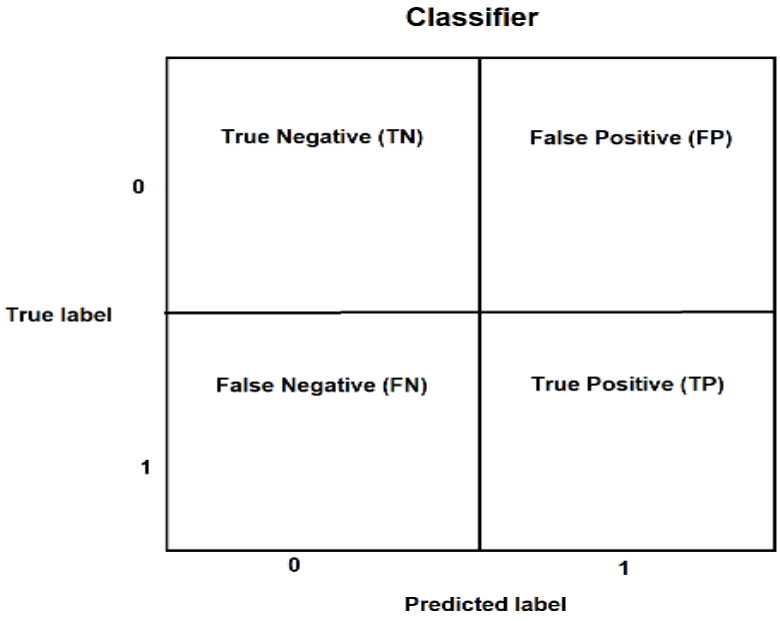
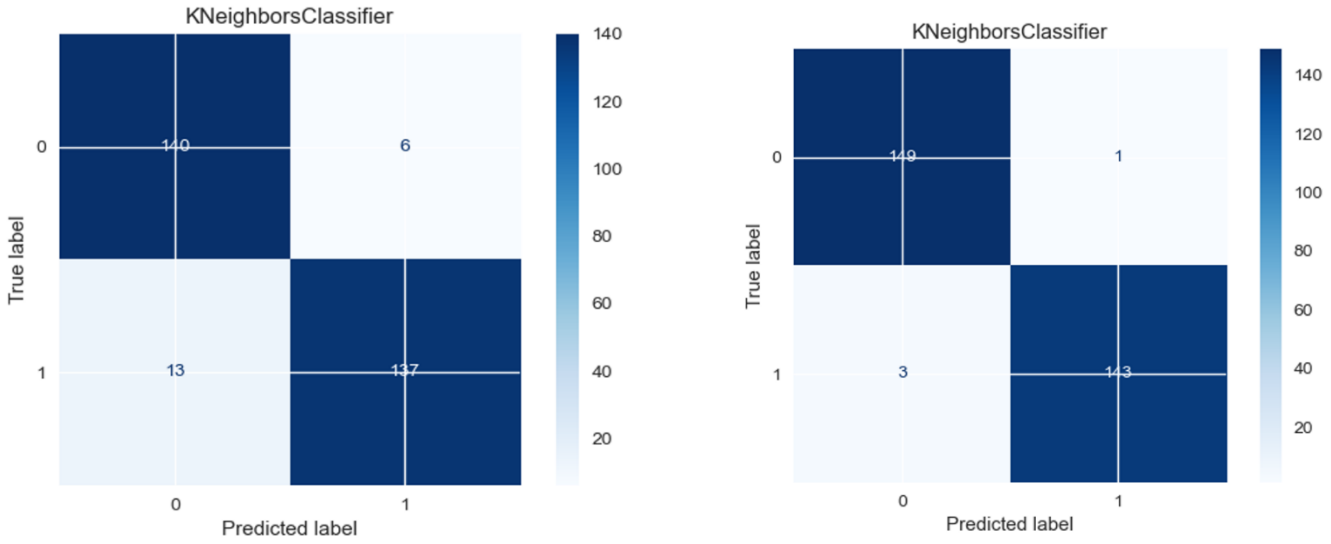


Figure 22: Confusion Matrix

Let’s check for the Confusion Matrix for the 7 algorithms that has been implemented above.

1. **K-Nearest Neighbour**
   (Random undersampling, Figure 23) correctly classified 137 as fraudulent and 140 as not fraudulent. The classifier also had some misclassifications, it predicted 13 as not fraudulent but they were fraudulent. It also incorrectly predicted 6 as fraudulent but they are not fraudulent.

   (NearMiss undersampling, Figure 23) correctly classified 143 as fraudulent and 149 as not fraudulent. The classifier also had some misclassifications, it predicted 3 as not fraudulent but they were fraudulent. It also incorrectly predicted 1 as fraudulent but they are not fraudulent.

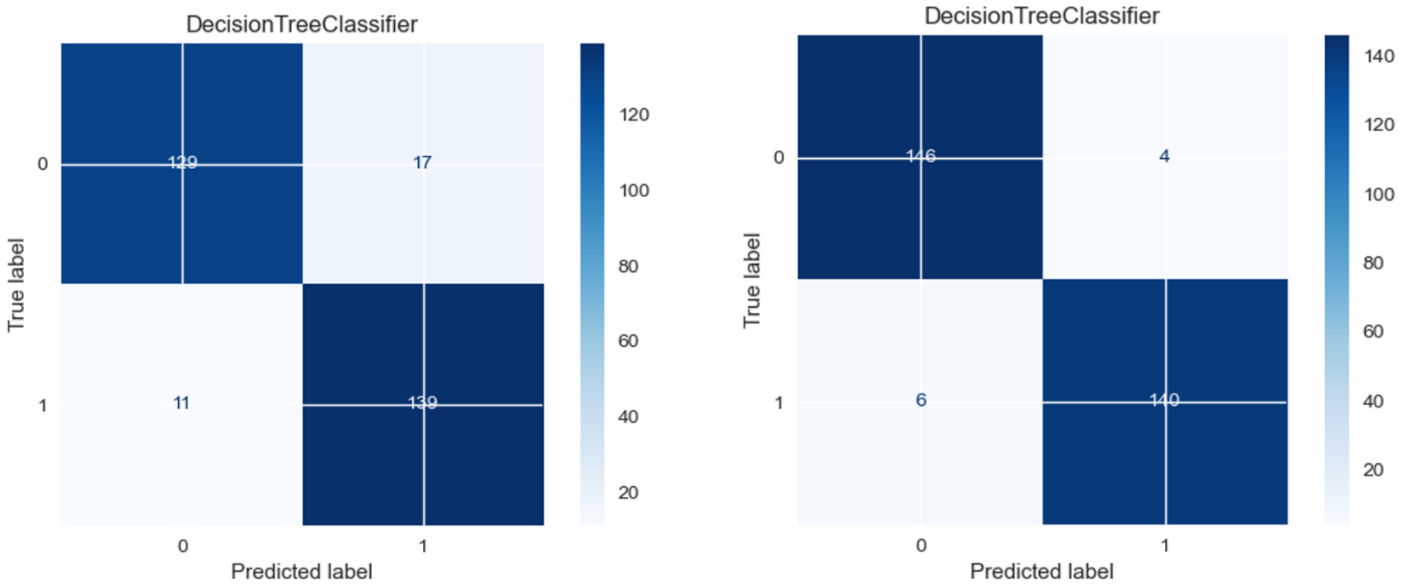


(Random undersampling) (NearMiss undersampling)

Figure 23: Confusion Matrix for KNN

2. **Decision Tree**
   (Random undersampling, Figure 24) correctly classified 139 as fraudulent and 129 as not fraudulent. The classifier also had some misclassifications, it predicted 11 as not fraudulent but they were fraudulent. It also incorrectly predicted 17 as fraudulent but they are not fraudulent.

   (NearMiss undersampling, Figure 24) correctly classified 140 as fraudulent and 146 as not fraudulent. The classifier also had some misclassifications, it predicted 6 as not fraudulent but they were fraudulent. It also incorrectly predicted 4 as fraudulent but they are not fraudulent.



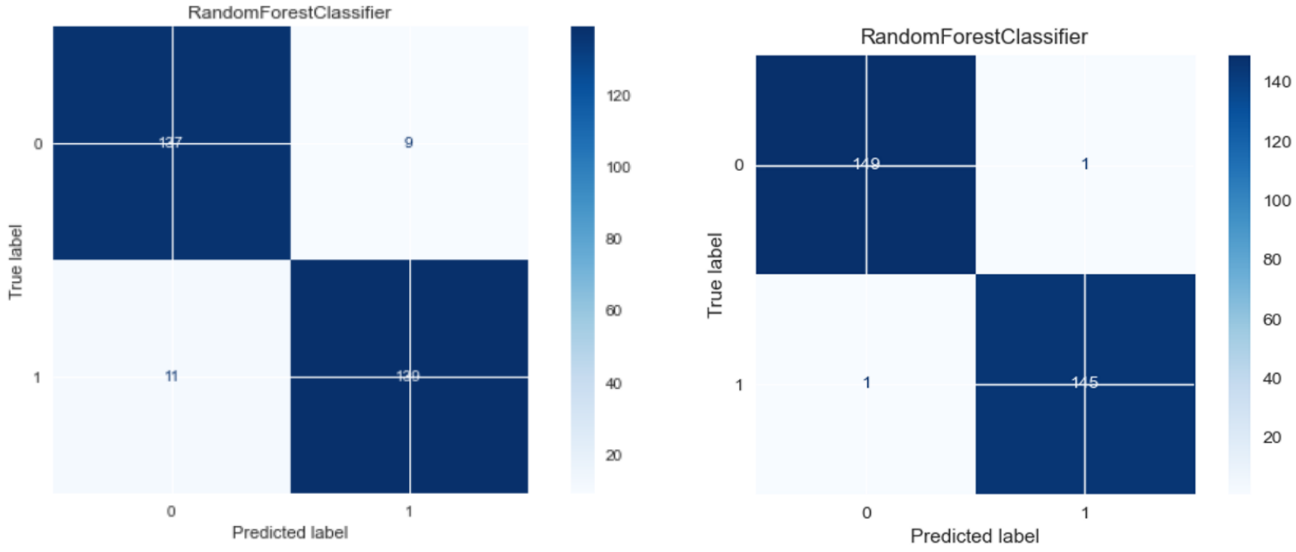
(Random undersampling) (NearMiss undersampling)

Figure 24: Confusion Matrix for Decision Tree

3. **Random Forest**

(Random undersampling) correctly classified 139 as fraudulent and 137 as not fraudulent. The classifier also had some misclassifications, it predicted 11 as not fraudulent but they were fraudulent. It also incorrectly predicted 9 as fraudulent but they are not fraudulent.

(NearMiss undersampling) correctly classified 145 as fraudulent and 149 as not fraudulent. The classifier also had some misclassifications, it predicted 1 as not fraudulent but they were fraudulent. It also incorrectly predicted 1 as fraudulent but they are not fraudulent.



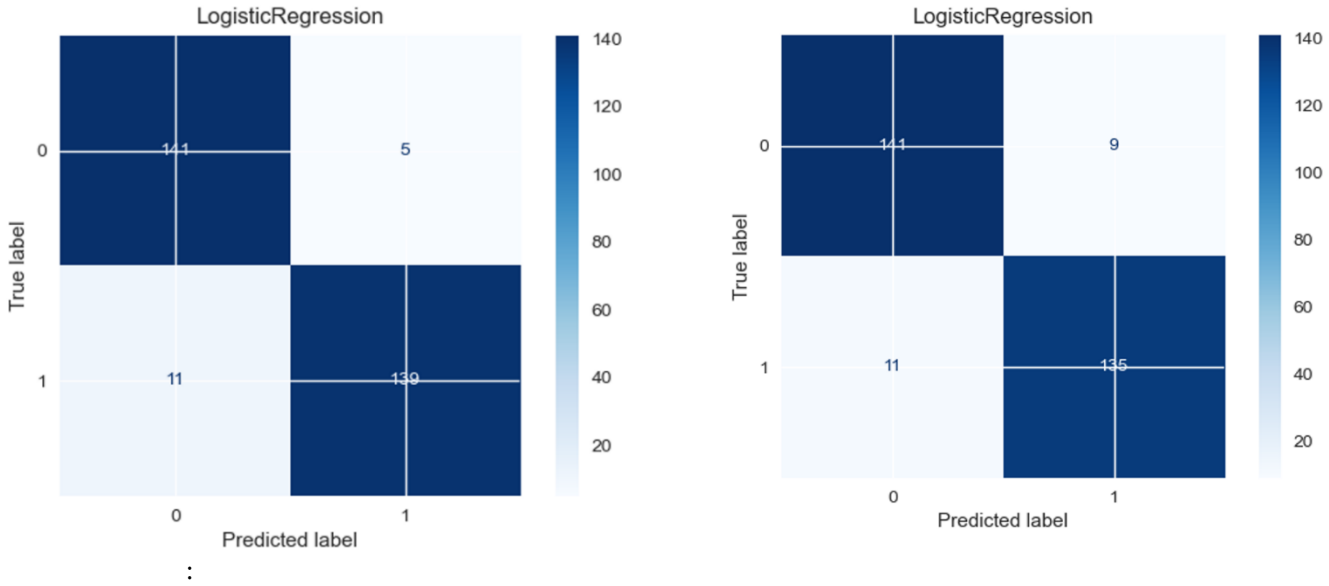
(Random undersampling) (NearMiss undersampling)

Figure 25: Confusion Matrix's for Random Forest

4. **Logistic Regression**

(Random undersampling) correctly classified 139 as fraudulent and 141 as not fraudulent. The classifier also had some misclassifications, it predicted 11 as not fraudulent but they were fraudulent. It also incorrectly predicted 5 as fraudulent but they are not fraudulent.

(NearMiss undersampling) correctly classified 135 as fraudulent and 141 as not fraudulent. The classifier also had some misclassifications, it predicted 11 as not fraudulent but they were fraudulent. It also incorrectly predicted 9 as fraudulent but they are not fraudulent.



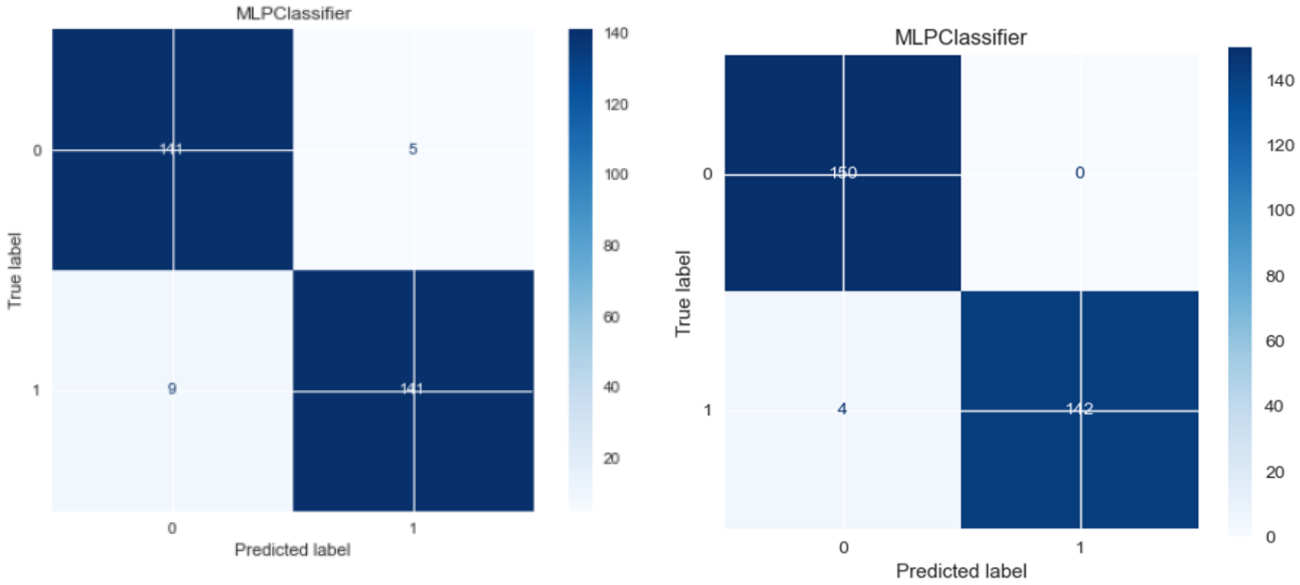
:

(Random undersampling) (NearMiss undersampling)

Figure 26: Confusion Matrix's for Logistic

5. **Artificial Neural Network**
   (Random undersampling) correctly classified 141 as fraudulent and 141 as not fraudulent. The classifier also had some misclassifications, it predicted 9 as not fraudulent but they were fraudulent. It also incorrectly predicted 5 as fraudulent but they are not fraudulent.

   (NearMiss undersampling) correctly classified 142 as fraudulent and 150 as not fraudulent. The classifier also had some misclassifications, it predicted 4 as not fraudulent but they were fraudulent. It also incorrectly predicted 0 as fraudulent but they are not fraudulent.

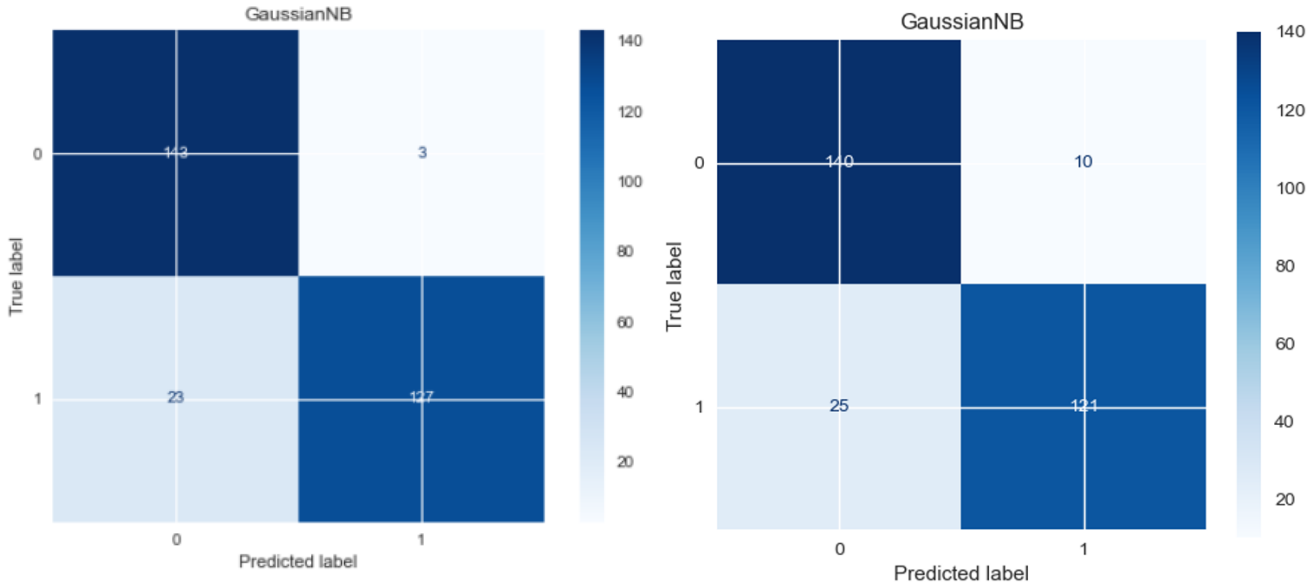


(Random undersampling) (NearMiss undersampling)

Figure 27: Confusion Matrix's for Artificial Neural Network

6. **Naive Bayes**
   (Random undersampling) correctly classified 127 as fraudulent and 143 as not fraudulent. The classifier also had some misclassifications, it predicted 23 as not fraudulent but they were fraudulent. It also incorrectly predicted 3 as fraudulent but they are not fraudulent.

   (NearMiss undersampling) correctly classified 121 as fraudulent and 140 as not fraudulent. The classifier also had some misclassifications, it predicted 25 as not fraudulent but they were fraudulent. It also incorrectly predicted 10 as fraudulent but they are not fraudulent.



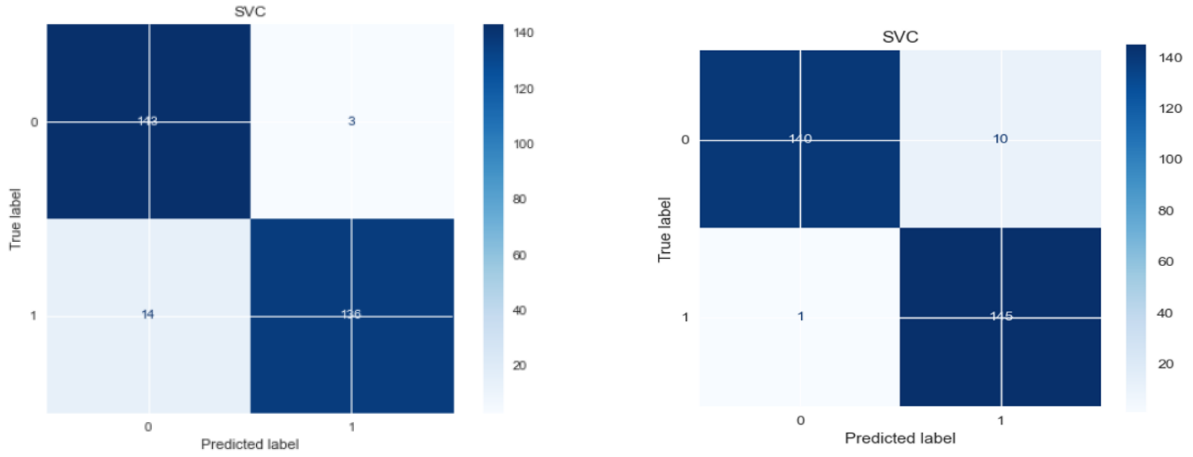
(Random undersampling) (NearMiss undersampling)

Figure 28: Confusion Matrix's for Naïve Bayes

7. **Support Vector Machine**

(Random undersampling) correctly classified 136 as fraudulent and 143 as not fraudulent. The classifier also had some misclassifications, it predicted 14 as not fraudulent but they were fraudulent. It also incorrectly predicted 3 as fraudulent but they are not fraudulent

(NearMiss undersampling) correctly classified 145 as fraudulent and 140 as not fraudulent. The classifier also had some misclassifications, it predicted 1 as not fraudulent but they were fraudulent. It also incorrectly predicted 10 as fraudulent but they are not fraudulent



(Random undersampling) (NearMiss undersampling)

Figure 29: Confusion Matrix for Decision Tree

We can use ROC and AUC to help us make a better decision.

ROC Curve

An **ROC curve** (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.

The curve is constructed by plotting the sensitivity by (1- specificity) for different classification thresholds.

The curves that are closer to 1 for y-axis (top right corner) have higher specificities associated with higher sensitivities.

The y-axis shows the true positive rate (same thing as sensitivity)

True Positive Rate = TPR = $\frac{TP}{TP + FN}$

The x-axis shows the false positive rate (same thing as 1-specificity)

False Positive Rate = FPR = $\frac{FP}{FP + TN}$

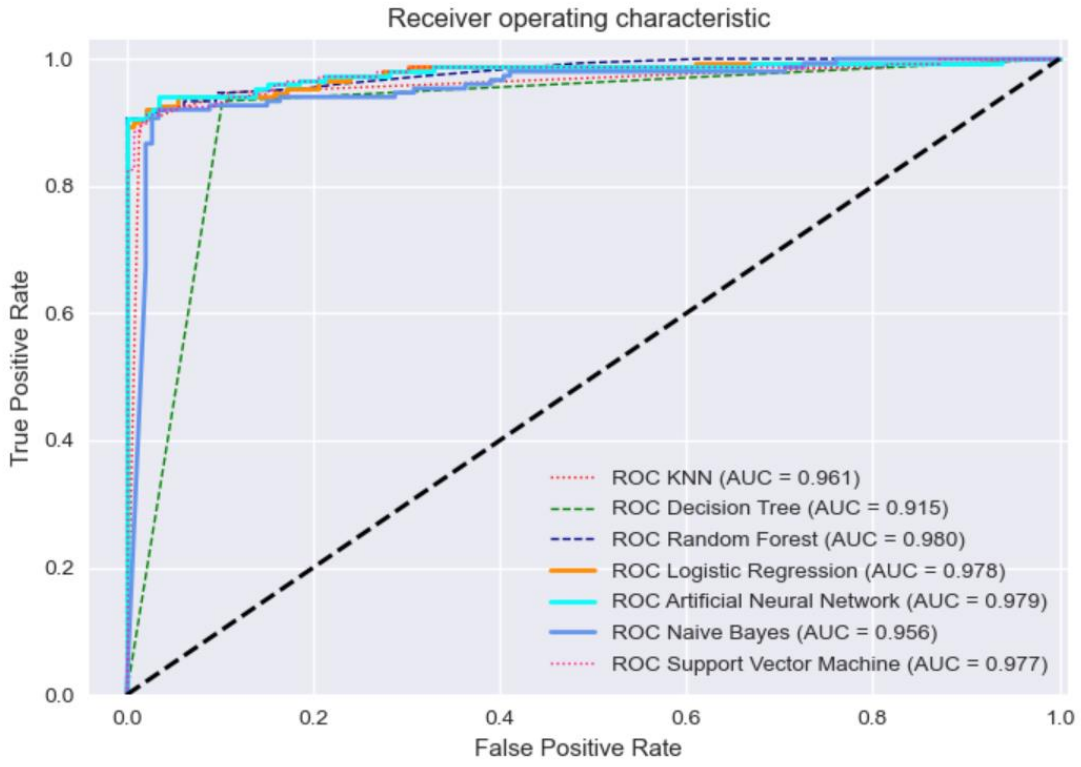


Figure 30: ROC curve for the 7 Algorithm using random Undersampling dataset

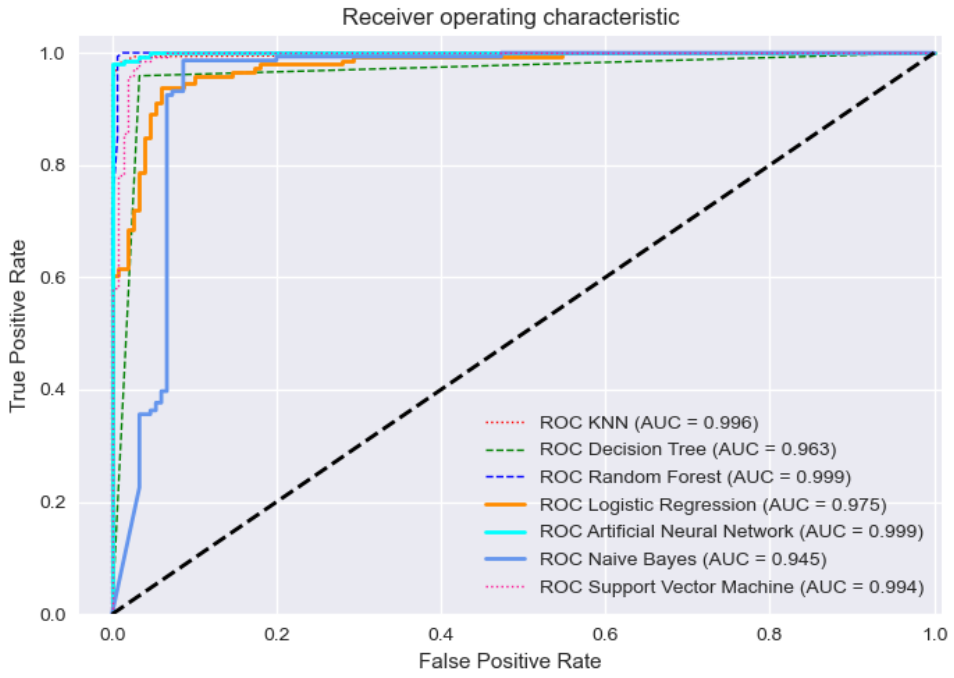


Figure 31: Zoomed ROC curve using random Undersampling dataset

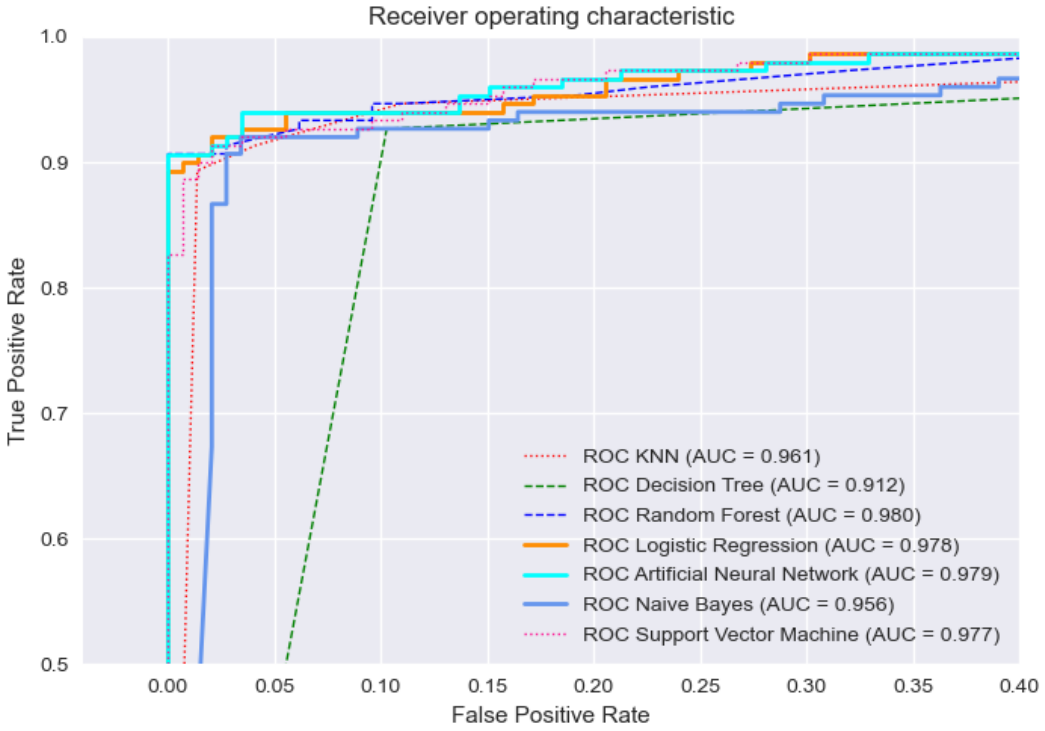


Figure 32: ROC curve for the 7 Algorithm using NearMiss Undersampling dataset

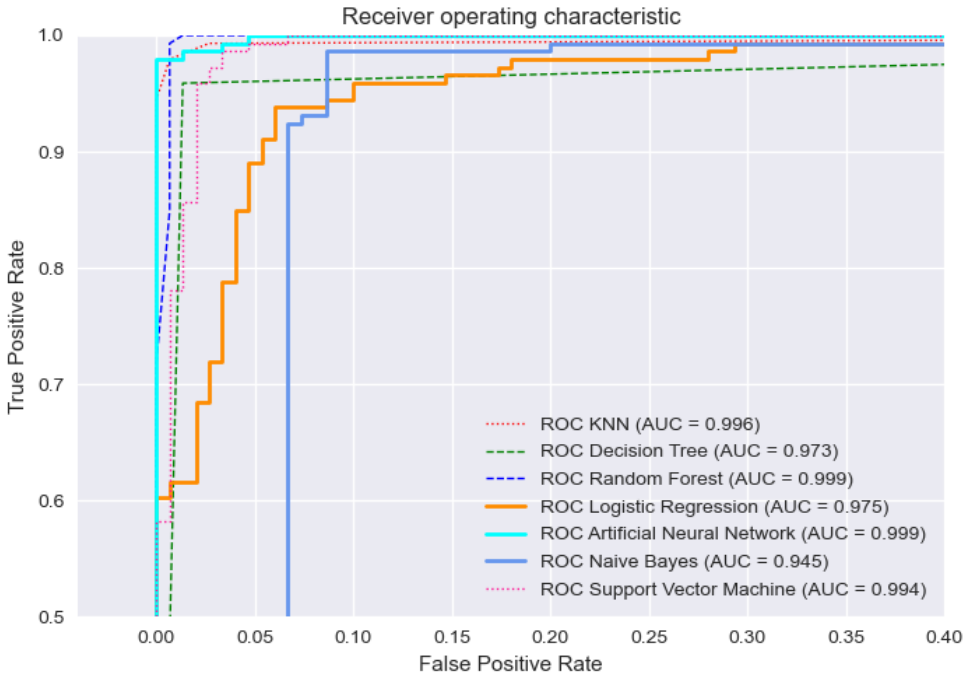


Figure 33: Zoomed ROC curve using NearMiss Undersampling dataset

When comparing roc curves the difference are hard to determine (like in curve A and B), so we rely on the Area under curve.

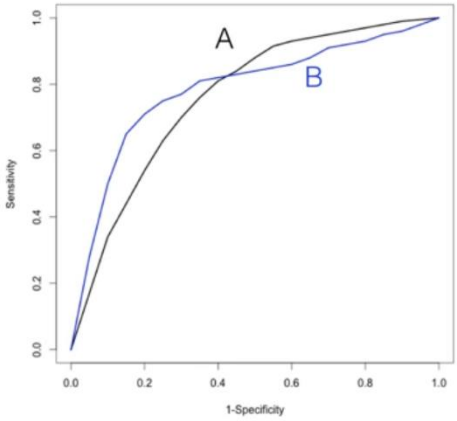


Figure 34: The ROC Curve

The model with the higher AUC will be considered the better classifier, if a classifier AUC is 1 then it should be considered a perfect model.

Area Under Curve (AUC)

The area under the curve is the measure of the ability of the Classifier to distinguish between the classes and is used as a summary for ROC. The higher is the AUC (the more area under the curve), the better is the performance of the model at distinguishing between the positive and negative class.

Below is the given AUC score for the 7 Algorithms:

```
# AUC for all 7 - models

# For KNN
AUC_KNN = metrics.auc(fpr1, tpr1)
# For Decision Tree
AUC_dt = metrics.auc(fpr2, tpr2)
# For Random Forest
AUC_rf = metrics.auc(fpr3, tpr3)
# For Logistic Regression
AUC_lr = metrics.auc(fpr4, tpr4)
# For Artificial Neural Network
AUC_ann = metrics.auc(fpr5, tpr5)
# For Naive Bayes
AUC_nb = metrics.auc(fpr6, tpr6)
# For Support Vector Machine
AUC_svm = metrics.auc(fpr7, tpr7)

print("AUC KNN: " + str(AUC_KNN))
print("AUC Decision Tree: " + str(AUC_dt))
print("AUC Random Forest: " + str(AUC_rf))
print("AUC Logistic Regression: " + str(AUC_lr))
print("AUC Artificial Neural Network: " + str(AUC_ann))
print("AUC Naive Bayes: " + str(AUC_nb))
print("AUC Support Vector Machine: " + str(AUC_svm))
```

Figure 35: Performing AUC

AUC Score for Undersampling method	KNN	Decision Tree	Random Forest	Logistic Regression	ANN	Naïve Bayes	SVM
Random	0.961	0.912	0.980	0.9780	0.979	0.956	0.977
NearMiss	0.996	0.973	0.999	0.975	0.999	0.945	0.994

Table and Observations

Below is the tables where we can find all the scores of the 7 – algorithm for both Undersampling approaches, let's us see which one gives the best among them.

Table 1: Results of all the Algorithm's calculation

Algorithms	Accuracy	Precision	Recall	ROC-AUC	F1-Score
KNN	93.58%	95.80%	91.33%	96.10%	93.51%
Decision Tree	90.54%	89.61%	92.00%	91.20%	91.80%
Random Forest	93.24%	93.91%	92.66%	98.00%	93.28%
Logistic Regression	94.59%	96.52%	92.66%	97.80%	94.55%
Artificial Neural Network	95.27%	96.57%	94.00%	97.90%	95.27%
Naïve Bayes	91.21%	97.69%	84.66%	95.60%	90.71%
Support Vector Machine	94.25%	97.84%	90.66%	97.70%	94.11%

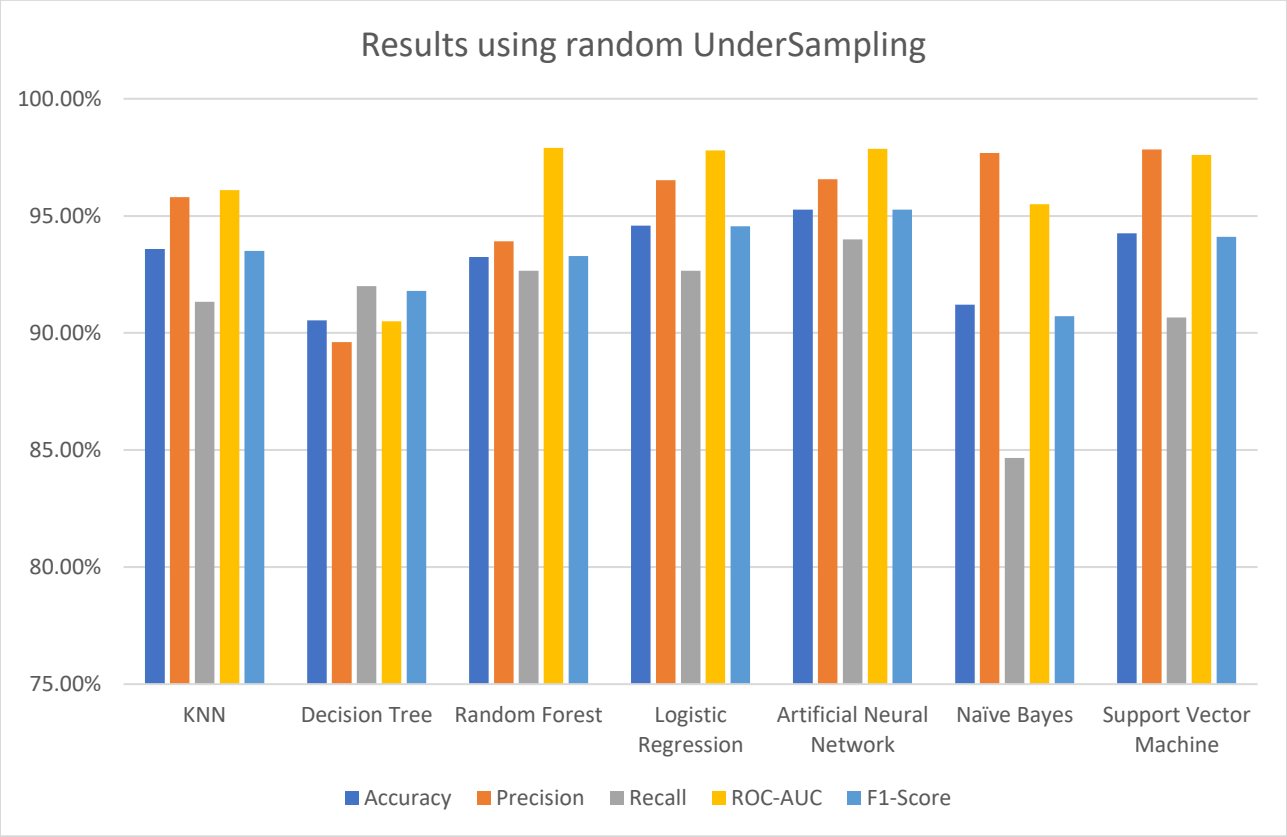


Figure 36: Histogram comparing all the 7 Algorithm using random undersampling

Table 2: Algorithm results using the NearMiss Undersampling approach

Algorithms	Accuracy	Precision	Recall	ROC-AUC	F1-Score
KNN	98.64%	99.30%	97.94%	99.60%	98.62%
Decision Tree	97.29%	98.59%	95.89%	97.30%	97.22%
Random Forest	99.32%	99.31%	99.31%	99.90%	99.31%
Logistic Regression	93.24%	93.75%	92.46%	97.50%	93.10%
Artificial Neural Network	98.64%	100%	97.26%	99.90%	98.61%
Naïve Bayes	88.17%	92.36%	82.87%	94.50a%	87.36%
Support Vector Machine	96.28%	93.54%	99.31%	99.40%	96.34%

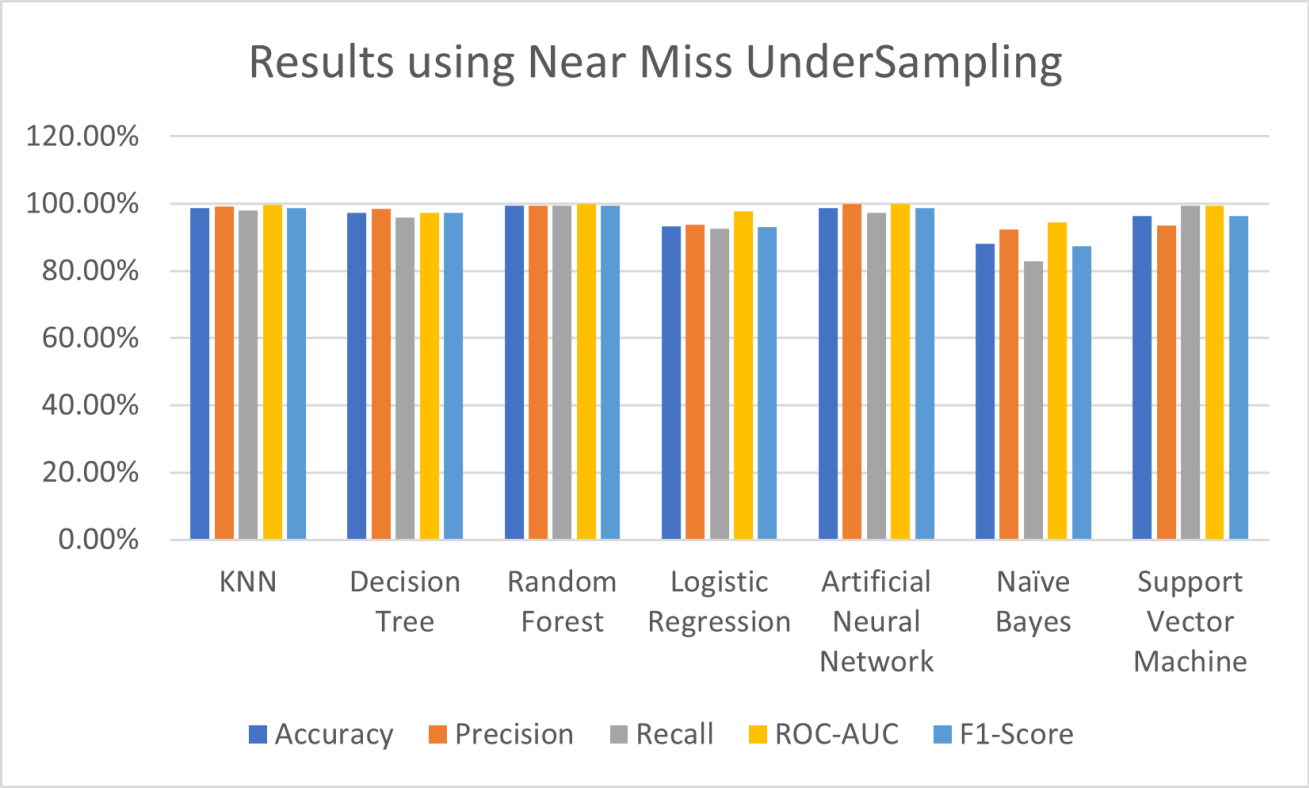


Figure 37: Histogram comparing all the 7 Algorithm using near miss undersampling

Conclusion

From this project we observed that with 284315 genuine cases, we have 492 fraudulent transactions. So anytime we get a database we shouldn't solely rely on database being optimized and scaled. This project made us understand the concepts behind handling the database that involves normalizing, scaling the features amount and time with respect to class column and lastly under-sampling the database by eighter randomly choosing 492 genuine transactions among 284315 genuine ones or using NearMiss algorithm, to pair up with fraud numbers.

From CS-4661. we had learnt how to split the dataset based on target and features using random state and test-size. Then we started applying the algorithms to both datasets (random and NearMiss undersampling datasets) in-order to find the accuracy levels.

We have applied 7-algorithms for Random undersampling dataset, and the same 7-algorithms for the NearMiss undersampling dataset.

For the data where we used Random Undersampling, in terms of accuracy, we think the best was achieved via Artificial Neural Network as 95.27%.

For the data where we used NearMiss Undersampling, terms of accuracy, we think the best was achieved via Random Forest as 99.32%

We calculated Area Under Curve (AUC) for both data balancing approaches, then plot the ROC for all 7-algorithms.

In terms of ROC-AUC, we achieved the best in Random Forest as 98% as shown in diagram for Random undersampling approach. For the NearMiss data balancing approach, we achieved the best in using both Random Forest and ANN with 99.9%, they came in a tie.

We have also calculated Precision for both data balancing approaches for all 7-algorithms, the best result was observed in Support Vector Machine as 97.80% using the random undersampling dataset. Using the NearMiss undersampling dataset, the best is Random Forest with 99.31%

We have also calculated F1-score and Recall for all 7-algorithms, the best result was observed in Artificial Neural Network as 95.27% and 94% respectively for data balancing approach 1. For approach 2 the best results were Random Forest with 99.31 for both recall and f1-score.

We have also tried to visualize the predictive analytics like using the confusion matrix since that gives direct comparisons about values of True Positives, False Positives, True Negatives and False Negatives and we also did a plot of confusion matrices. These confusion matrices are very similar which makes it harder to choose with classifier is a better fit using our current data sets. So, we have calculated the AUC and ROC curve for better result.

Now we have 7-algorithms, but we need to choose one among the 7 that gives the best accuracy, precision, recall, ROC and F1-score.

From the histogram above we observed that the best result was observed with ANN (Artificial Neural Network), with an accuracy of 95.27%, precision of 96.57%, recall of 94%, ROC of 97.87% and F1-score of 95.27% using approach 1 (Random undersampling).

If we use approach 2 (NearMiss) to balance our data the best result was observed with Random Forest with an accuracy of 99.32%, precision of 99.31%, recall of 99.31%, ROC-AUC of 99.85, and F1-Score of 99.31.

So, with this we conclude that for credit card fraud detection algorithm, the best suitable algorithm is ANN (Artificial Neural Network) if we use Random Undersampling to balance our data. If we use NearMiss to balance our data, we can conclude that the best suitable algorithm is Random Forest.

Team Members Contribution:

All the Team Members contributed equally to the project.

References

[1] http://ieeexplore.ieee.org/document/9121114/

[2] https://www.researchgate.net/publication/337591236_Credit_Card_Fraud_Detection_Using_Machine_Learning_With_Python

[3] https://www.moneyhelper.org.uk/en/blog/scams-and-fraud/what-is-credit-card-fraud-how-prevent-it

[4] https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[5] https://www.newgenapps.com/blogs/random-forest-analysis-in-ml-and-when-to-use-it-2/

[6] https://www.upgrad.com/blog/how-random-forest-algorithm-works-in-machine-learning/

[7] https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

[8] https://www.analyticsvidhya.com/blog/2021/04/artificial-neural-network-its-inspiration-and-the-working-mechanism/

[9] https://dataaspirant.com/naive-bayes-classifier-machine-learning/

[10] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[11] https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

[12] https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[13] https://calstatela.instructure.com/courses/73294/files/9752711?module_item_id=4262048- Lecture on Artificial Neural Network

[14] https://calstatela.instructure.com/courses/64721/files/8278784?module_item_id=3416914 Lecture on KNN classifier

[15] https://calstatela.instructure.com/courses/64721/files/7741637?module_item_id=3416925 Lecture on Decision tree classifier

[16] https://en.wikipedia.org/wiki/Confusion_matrix

[17] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[18] https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap-in-python/

[19] https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e

[20] https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/

[21] https://stackoverflow.com/questions/29204005/how-to-perform-under-sampling-in-scikit-learn

[22] http://lijiancheng0614.github.io/scikit-learn/modules/generated/sklearn.preprocessing.MinMaxScaler.html