NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4032 Data Analytics & Mining
Project Report

Credit Card Fraud Detection

Submitted by Group 3:

Name	Student ID	Contribution
Tan Jian Wei	U1820308E	Support Vector Machine
Kong Alson	U1822899B	Support Vector Machine
Sun Jinghan	U1722008B	Data Preprocessing, Random Forest
Ang Yong Xin	U1821708G	Random Forest
Sam Jian Shen	U1821296L	Artificial Neural Network
Li Zhaochen	U1720279L	Data Preprocessing, Logistic Regression

Contents **Pages**

1. Abstract

This project performs data mining and analysis on the credit card transactions dataset obtained from Kaggle. Some popular data mining techniques are explored and adopted to find patterns in the data and thus to detect fraudulence from transactions. We conducted the project with three stages - data preprocessing, building classification models and results evaluation. In data preprocessing, we explored the dataset and mainly performed outlier removal, feature selection and data oversampling, which provides the final training and testing sets to various classification models. In the second stage, we built classification models using four supervised machine learning algorithms respectively - Logistic Regression, Support Vector Machine (SVM), Random Forest and Artificial Neural Network (ANN), and we tested each model's performance on the testing set. Lastly, we evaluated the results from different classification models based on certain metrics and chose the best model for this task.

2. Problem Description

2.1 Motivation

With cashless payment becoming more and more popular, fraudulent credit card transactions can be inevitable. It's important to find an effective way to recognize fraud transactions so that huge losses for customers or companies can be avoided.

Data mining is a process which enables us to find important patterns from a large amount of data, and thus may use the information to predict the behavior of new data.

Therefore, we aim to use data mining techniques to build a solution for credit card fraud detection.

2.2 Problem Definition

This project aims to detect fraudulent credit card transactions using some popular data mining techniques. The dataset being used is a public dataset obtained from Kaggle. It contains 284,807 transactions made by credit card holders, among which 492 are fraudulent. The goal is to detect fraud transactions based on 30 numerical input variables, namely V1, V2, …, V28, Time and Amount. V1 to V28 are principal components transformed with PCA while Time and Amount have not been transformed.

3. Approach

The approaches used in this project can be categorized into three aspects - data preprocessing, classification models and evaluation metrics. Details are introduced below.

3.1 Data Preprocessing

Data preprocessing involves the following steps: exploratory analysis, removal of outliers, feature selection, training and testing set splitting and data oversampling.

3.2 Classification Models

Four classification algorithms are used to build different classifiers: **Logistic Regression**, **Support Vector Machine**, **Random Forest** and **Artificial Neural Network.** Each classification model is trained and tested on the same training and testing set respectively. Comparisons among these models are made based on the evaluation metrics mentioned in section 3.3.

3.3 Evaluation Metrics

Considering the high imbalance of the dataset, accuracy of the model is evaluated based on the following metrics.

➢ **Confusion Matrix**

Three scores are considered: Precision score, Recall score and F1 score.

They are computed as follows:

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

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

$$\text{F1 Score} = \frac{2 * precision * recall}{precision + recall}$$

➢ **Precision-Recall Curve**

Precision-Recall Curve is suitable for an imbalanced dataset, as it focuses more on the minority class, which in our case is the fraud transactions. And the Area-Under-Curve (AUC) score will be considered.

In addition, there is a trade-off between precision score and recall score. In this project, in order to detect as many fraud transactions as possible, we will focus more on recall than precision.

4. Implementations

For implementation, we use Python as the programming language and Jupyter Notebook as the IDE. Some popular Python libraries for data science are used, such as Pandas, Seaborn, Matplotlib, SciKit-Learn, etc.

4.1 Data Preprocessing

4.1.1 Exploratory Analysis

To get a better understanding of the dataset, we first performed exploratory analysis on the original dataset. The dataset consists of 285807 rows and 31 columns. First 30 columns are independent variables, namely, Time, V1 - V28, and Amount, and they are all floating data types. Last column is the dependent variable *Class*, which consists of two integer values, 0 for valid transactions and 1 for fraud transactions respectively. There are no missing values in the dataset. As shown in the figure below, the distribution of *Class* is highly imbalanced, skewing to valid transactions. This imbalance may favor the model in predicting the majority class and thus results in a poor performance in detecting fraud transactions. Some oversampling techniques are used to deal with this imbalance which will be introduced in Section 4.1.5.

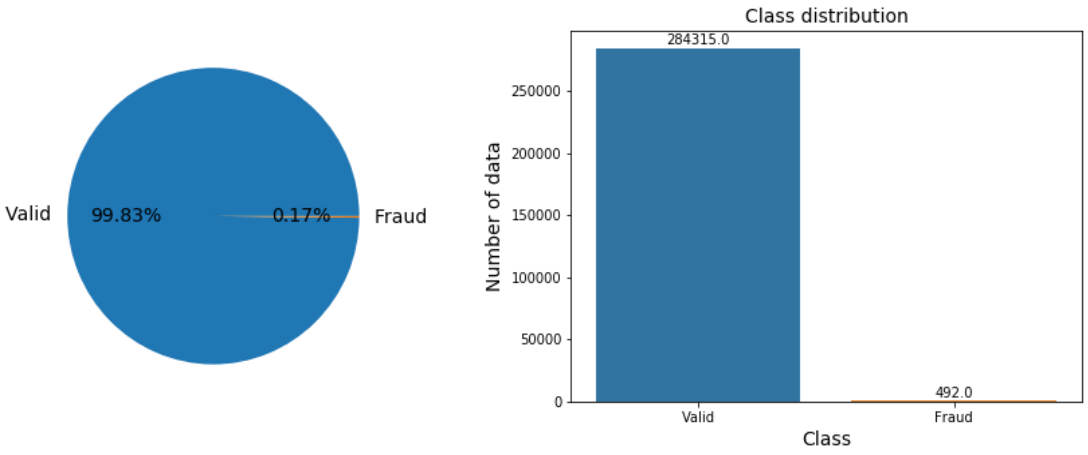

Figure 4.1.1 Class distribution in original dataset

4.1.2 Removing Outliers

To reduce noise in the dataset, we performed outlier removal. As there are only 492 fraud records originally, we only deal with outliers in *Class* 0. The following graph shows distribution of each independent variable for *Class* 0 data. It can be noticed that most of the features approximate a normal distribution (except the feature *Time* in the first figure, which records the seconds elapsed between each transaction and the first transaction). As such, we treat the data outside 3-standard deviation of a normal distribution as the outliers and select features *V1, V2, V3, V4, V5* and *Amount* to remove outliers on.

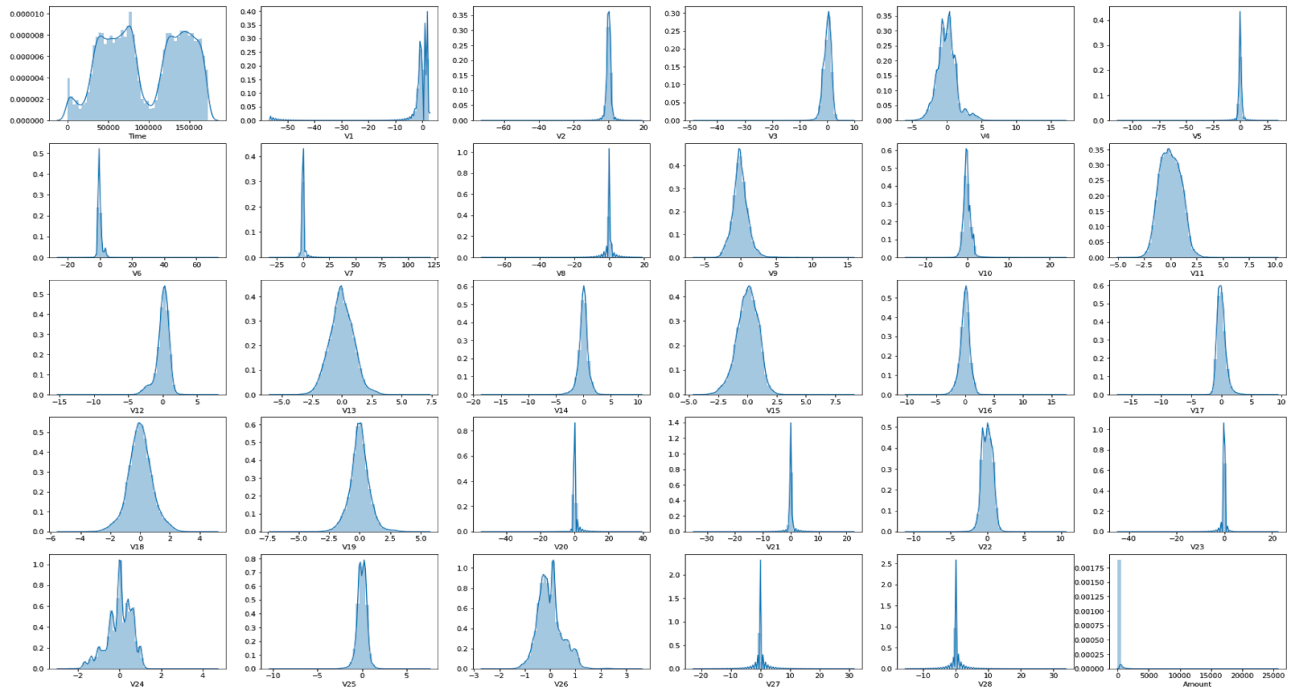

Figure 4.1.2 Distribution of each feature for Class 0

In total, 3346 records of *Class* 0 are removed from the original dataset.

4.1.3 Feature Selection

To reduce the complexity of classification models, we performed feature selection to find and remove those irrelevant features.

Originally, there are 30 input features available for predicting the *Class* label. We plotted the distribution of each feature with respect to each class, as shown in *Figure 4.1.3.a* below. From

Figure 4.1.3.a we can notice that, for certain features, the two lines are highly overlapped, which indicates that they are not much relevant to the *Class* label.

We also plotted the correlation of each feature with the *Class,* as shown in *Figure 4.1.3.b.* From *Figure 4.1.3.b*, similarly, it can be noticed that some features located in the middle have nearly 0 correlation with the *Class*.

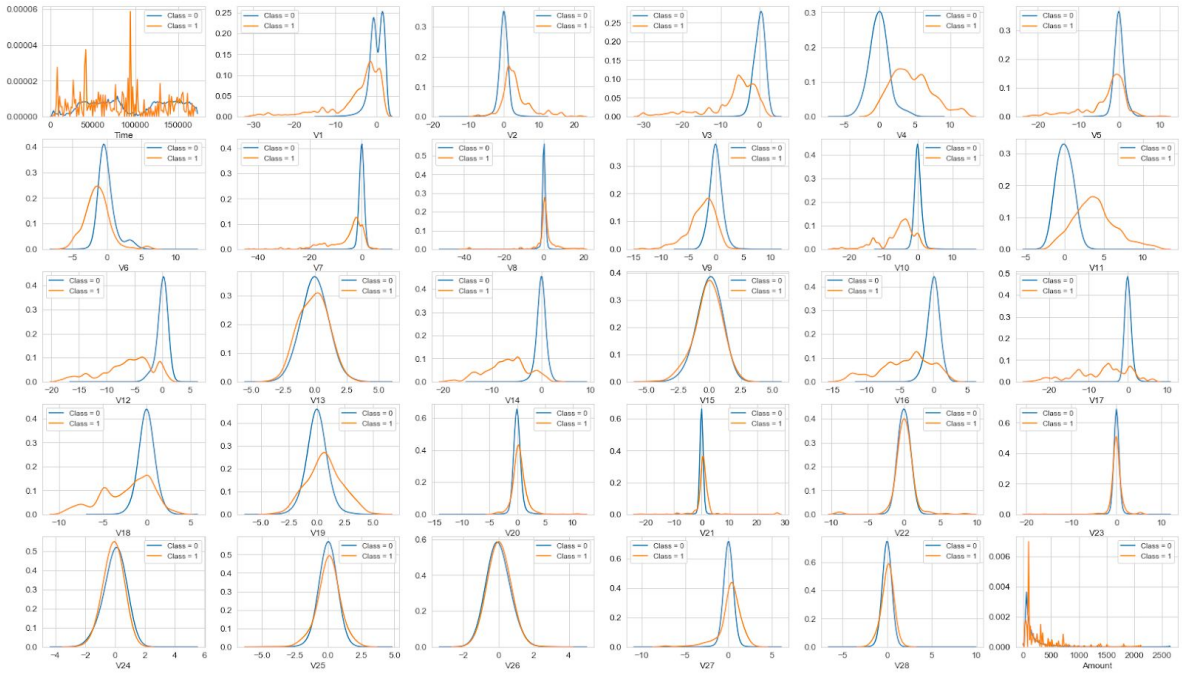

Figure 4.1.3.a Distribution of each feature for two classes

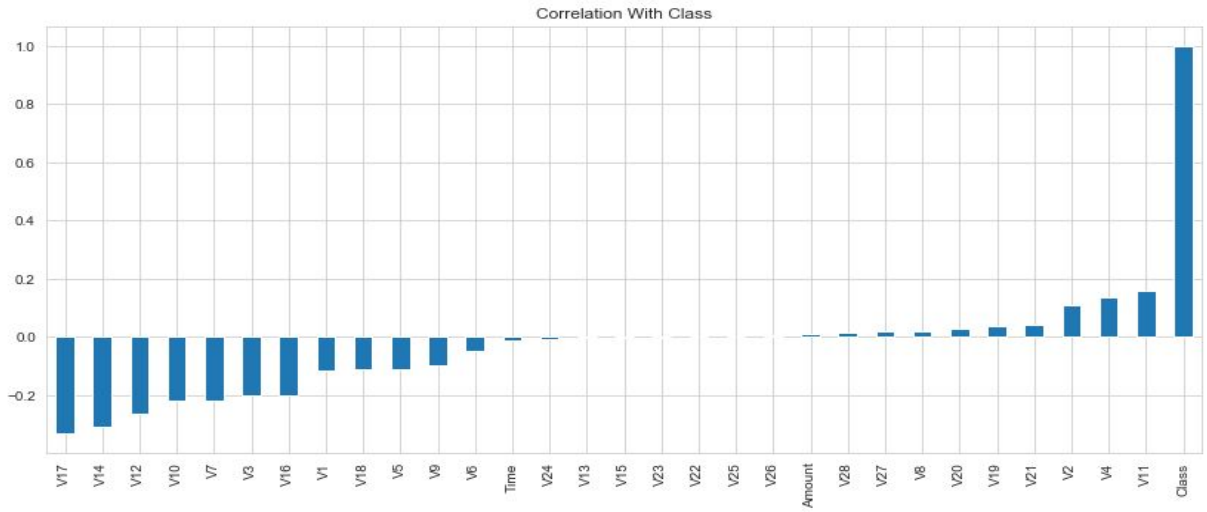

Figure 4.1.3.b Correlation of each feature with Class

Based on the observations above, we chose to remove the following 11 features from the original dataset:

Time, V8, V13, V15, V20, V21, V22, V23, V25, V26, V28.

After feature selection, we reduced the number of independent variables from 30 to 19.

4.1.4 Splitting into Training and Testing Set

We split the whole dataset into a training and testing set with a split ratio of 80:20.

Each model is trained using the training set. The testing set is used to evaluate the model's accuracy and is not seen by the model during training.

4.1.5 Data Oversampling

To deal with dataset imbalance, we performed oversampling on the training set using the method SMOTE. Fraud transactions (*Class* 1) are oversampled to 20% of valid transactions (*Class* 0).

After oversampling, the training set has the following class distribution as shown in *Figure 4.1.5*.

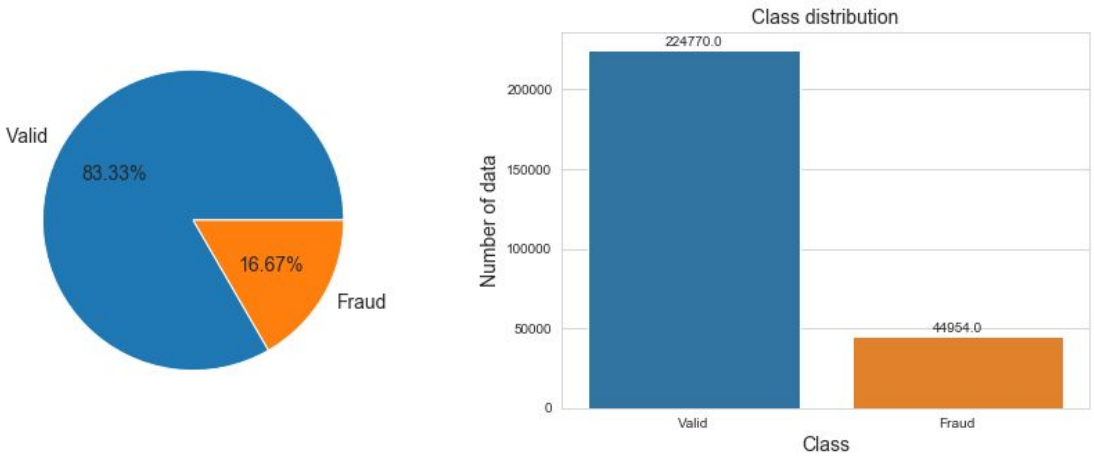

Figure 4.1.5 Class distribution in training set after oversampling

In our experiments, we will compare the results trained using the unsampled training set and the oversampled training set to see the effect of oversampling for each classifier.

4.2 Classification Models

We implemented the four classification algorithms mainly using the Scikit-Learn package. Detailed implementations for each classifier are introduced below. Experimental results and evaluation of each model will be introduced in Section 5.

4.2.1 Logistic Regression

We first normalized the values in column "*Amount*" using z-score normalization ($z = \frac{x-\bar{x}}{\sigma}$).

Logistic Regression Classifier is implemented using `LogisticRegression()` from `sklearn.linear_model`.

```
LogisticRegression(solver='lbfgs',
                   C=1,
                   penalty='l2')
```

- `solver='lbfgs'` : using L-BFGS algorithm for weight optimization.
- `C = 1` : inverse of regularization strength; optimal value found using grid search.
- `penalty = 'l2'`: using L2 regularization to regularize weights and prevent overfitting.

4.2.2 Support Vector Machine (SVM)

SVM Classifier is implemented using `SVC()` from `sklearn.svm`.

```
classifier= SVC(kernel='rbf',
                C = 10,
                gamma=0.001,
                probability=True,
                random_state=0)
```

- `kernel='rbf'` : using Radial Basis Function kernel function.
- `C = 10` : inverse of regularization strength; optimal value found using grid search.
- `gamma = 0.001` : RBF Gaussian width; optimal value found using grid search.

- `probability = True`

4.2.3 Random Forest

Random Forest Classifier is implemented using `RandomForestClassifier()` from `sklearn.ensemble`.

```
RandomForestClassifier(n_estimators=100,
                       max_depth=10,
                       min_samples_split=5,
                       random_state=0)
```

- `n_estimators=100`: number of estimators in the Random Forest model is 100.
- `max_depth=10`: maximum depth of each decision tree is 10.
- `min_samples_split=5`: minimum number of samples each node should contain in order to further split.

4.2.4 Artificial Neural Network (ANN)

We first performed data normalization on the 19 independent variables using `StandardScaler()` from `sklearn.preprocessing`.

ANN Classifier is implemented using `MLPClassifier()` from `sklearn.neural_network`.

```
MLPClassifier(solver='lbfgs',
              activation='logistic',
              alpha=1e-5,
              hidden_layer_sizes=(5, 2),
              random_state=1)
```

- `solver='lbfgs'` : using L-BFGS algorithm for weight optimization.
- `activation='logistic'` : using the logistic sigmoid function as the activation function.

- `alpha=1e-5` : L2 regularization constant is 0.00001.
- `hidden_layer_sizes=(5,2)` : two hidden layers with 5 neurons and 2 neurons respectively

5. Experimental Results and Analysis

In our experiments, we trained each classifier on the original training set and oversampled training set respectively, and we tested each model using the same testing set. We analyzed the results based on the metrics mentioned previously in Section 3.3, i.e. confusion matrix and Precision-Recall Curve. We compared the results within each classifier to see if oversampling helps to achieve a better prediction, and we also compared the performance of all classifiers with some trade-offs to decide the best one for this task. Besides, based on the Precision-Recall Curve and f1-score, we also performed threshold moving when necessary to find the optimal decision threshold for fraud transactions (*Class* 1). The default threshold is 0.5.

5.1 Logistic Regression

Before oversampling, the model achieved a precision of 0.928, a recall of 0.681, f1-score of 0.785 and AUC of 0.837, as shown in the confusion matrix and precision-recall curve below.

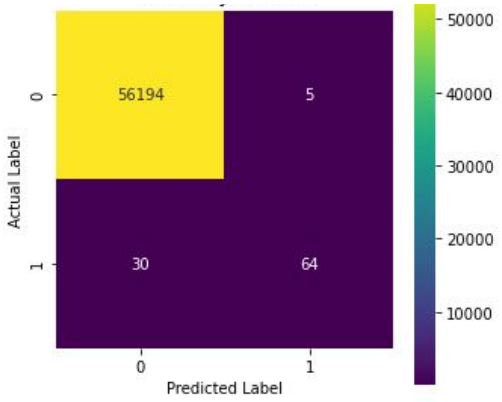

Figure 5.1.a Confusion Matrix - before oversampling

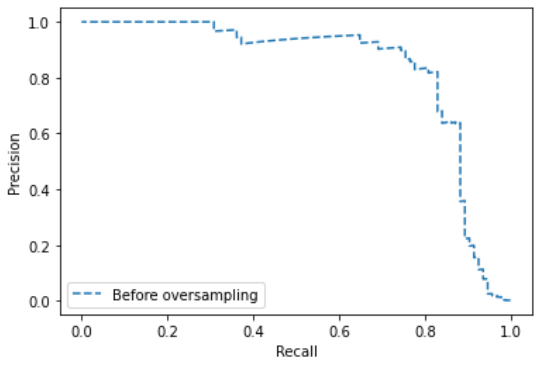

Figure 5.1.b Precision-Recall Curve - before oversampling

According to *Figure 5.1.b* , we can notice that the default threshold may not be the optimal threshold point. We performed threshold moving to find the best threshold corresponding to the highest f1-score. As a result, we found the optimal threshold of 0.109 (for *Class* 1), under which the model can achieve a precision of 0.821 and a much higher recall of 0.830, as shown in the confusion matrix below.

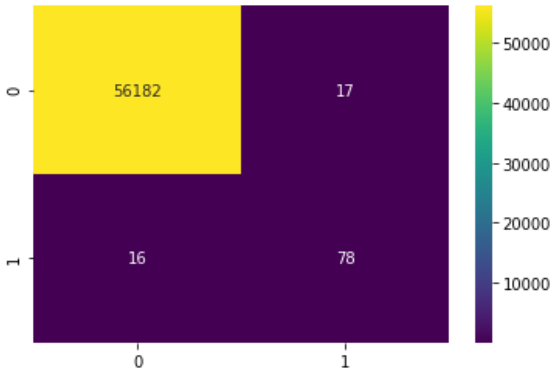

Figure 5.1.c Confusion Matrix - before oversampling - threshold = 0.109

After oversampling, the model achieved a precision of 0.191, a recall of 0.926, f1-score of 0.316 and AUC of 0.846, as shown in the confusion matrix and precision-recall curve below.

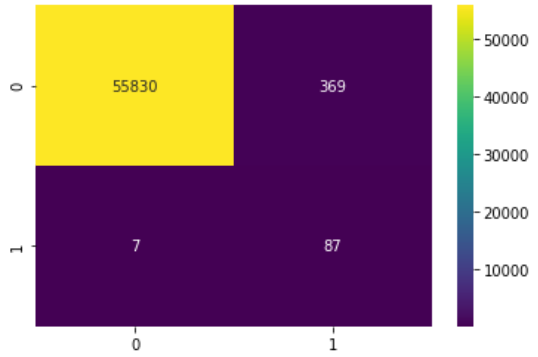

Figure 5.1.d Confusion Matrix - after oversampling

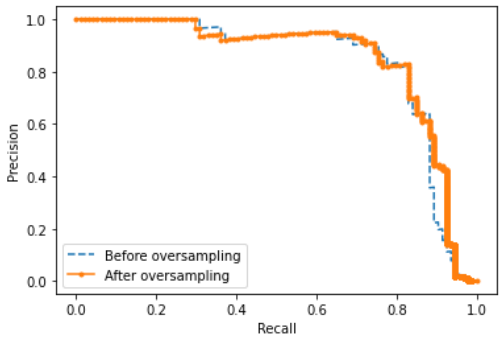

Figure 5.1.e Precision-Recall Curve - after oversampling

Similarly, we also tried to get a better prediction by moving the threshold, and we found that when the threshold is equal to 0.985 (for *Class* 1), the model can achieve the optimal result with both precision and recall of 0.830, as shown in the new confusion matrix below.

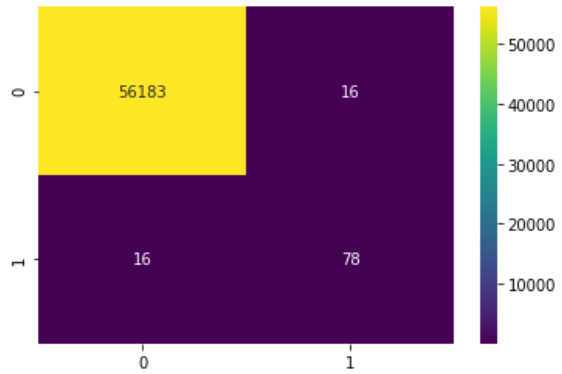

Figure 5.1.f Confusion Matrix - after oversampling - threshold = 0.985

Summarizing the results, we have the table below for Logistic Regression Classifier:

Training Set	**Precision**	**Recall**	**F1 Score**	**AUC**
Before Oversampling (Threshold = 0.5)	0.928	0.681	0.785	0.837
Before Oversampling (Threshold = 0.109)	0.821	0.830	0.825	0.837
After Oversampling (Threshold = 0.5)	0.191	0.926	0.316	0.846
After Oversampling (Threshold = 0.985)	0.830	0.830	0.830	0.846

5.2 Support Vector Machine (SVM)

Before oversampling, the model achieved a precision of 0.915, a recall of 0.798, f1-score of 0.852 and AUC of 0.854, as shown in the confusion matrix and precision-recall curve below.

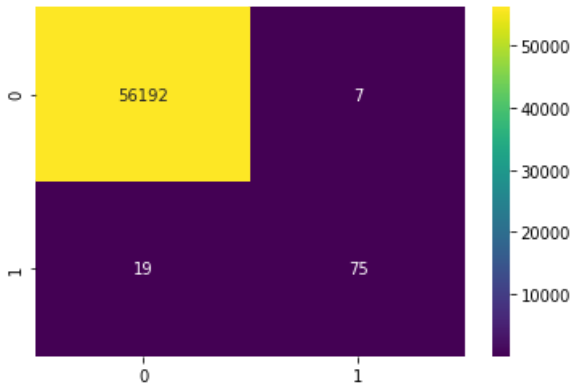

Figure 5.2.a Confusion Matrix - before oversampling

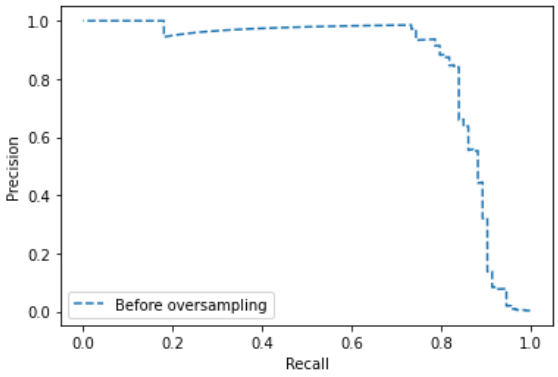

Figure 5.2.b Precision-Recall Curve - before oversampling

After oversampling, the model achieved a precision of 0.457, a recall of 0.915, f1-score of 0.610 and AUC of 0.807, as shown in the confusion matrix and precision-recall curve below. Comparing the two curves in *Figure 5.2.d* , we can notice that the performance after oversampling is not obviously improved, which can also be seen from the dropped AUC score. Thus, oversampling is not helpful for our SVM Classifier.

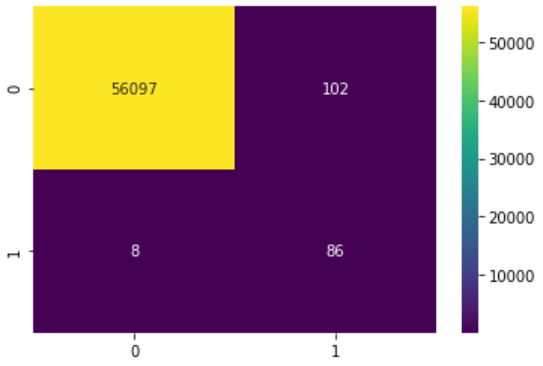

Figure 5.2.c Confusion Matrix - after oversampling

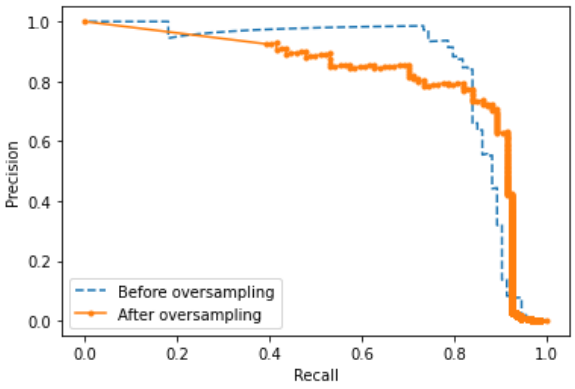

Figure 5.2.d Precision-Recall Curve - after oversampling

Summarizing the results, we have the table below for SVM Classifier:

Training Set	Precision	Recall	F1 Score	AUC
Before Oversampling (Threshold = 0.5)	0.915	0.798	0.852	0.854
After Oversampling (Threshold = 0.5)	0.457	0.915	0.610	0.807

5.3 Random Forest

Before oversampling, the model achieved a precision of 0.919, a recall of 0.840, f1-score of 0.878 and AUC of 0.899, as shown in the confusion matrix and precision-recall curve below.

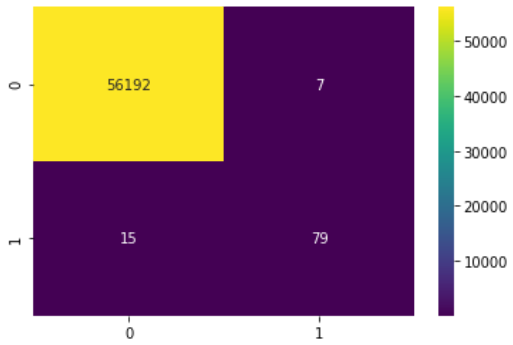

Figure 5.3.a Confusion Matrix - before oversampling

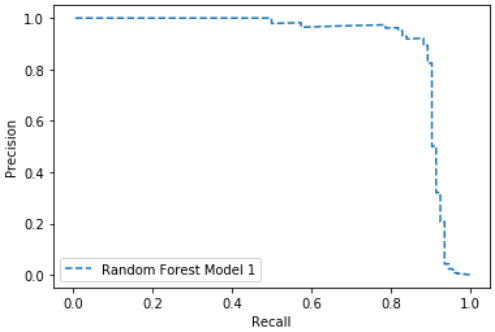

Figure 5.3.b Precision-Recall Curve - before oversampling

After oversampling, the model achieved a precision of 0.810, a recall of 0.904, f1-score of 0.854 and AUC of 0.906, as shown in the confusion matrix and precision-recall curve below.

We can notice that although the precision score decreases slightly, the recall score increases from 0.840 to 0.904, and the AUC also increases.

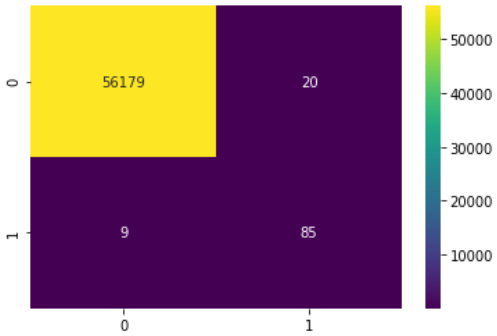

Figure 5.3.c Confusion Matrix - after oversampling

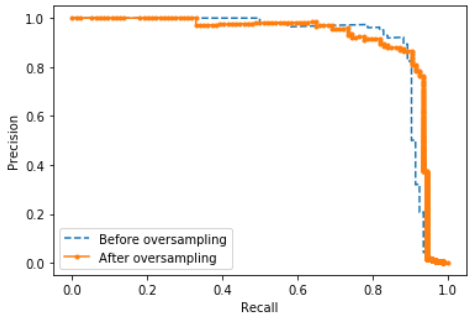

Figure 5.3.d Precision-Recall Curve - after oversampling

Based on the orange line in *Figure 5.3.d* , we can notice that the default threshold may not be the optimal threshold point. So we performed threshold moving to find the threshold corresponding to the highest f1-score. As a result, we found the optimal threshold of 0.596 (for *Class* 1), under which the model can achieve a higher precision of 0.867 and the same recall of 0.904, as shown in the confusion matrix below.

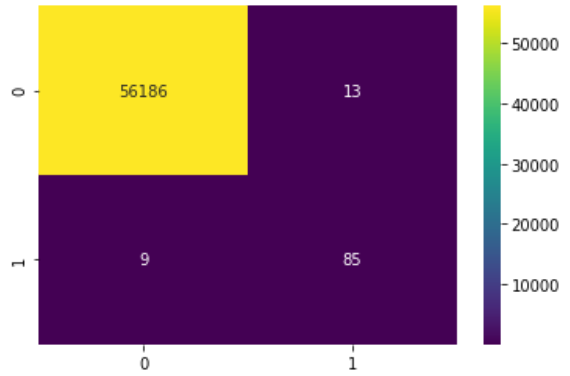

Figure 5.3.e Confusion Matrix - after oversampling - threshold = 0.596

We also checked the importance of features in building the above Random Forest model. As can be seen from *Figure 5.3.f* below, *V14, V17, V10 ,V12* are the top 4 most important features.

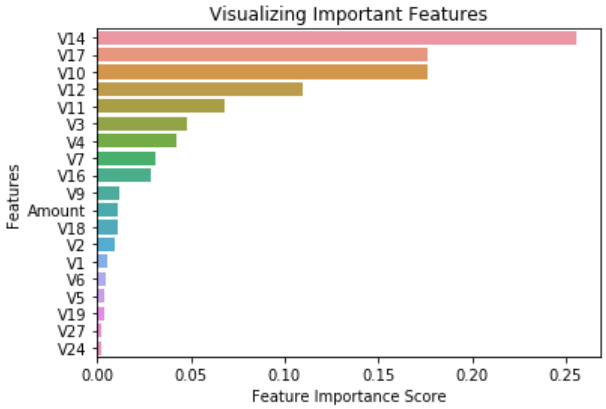

Figure 5.3.f Feature Importance in Random Forest

Summarizing the results, we have the table below for Random Forest Classifier:

Training Set	Precision	Recall	F1 Score	AUC
Before Oversampling (Threshold = 0.5)	0.919	0.840	0.878	0.899
After Oversampling (Threshold = 0.5)	0.810	0.904	0.854	0.906
After Oversampling (Threshold = 0.596)	0.867	0.904	0.885	0.906

5.4 Artificial Neural Network (ANN)

Before oversampling, the model achieved a precision of 0.878, a recall of 0.840, f1-score of 0.859 and AUC of 0.869, as shown in the confusion matrix and precision-recall curve below.

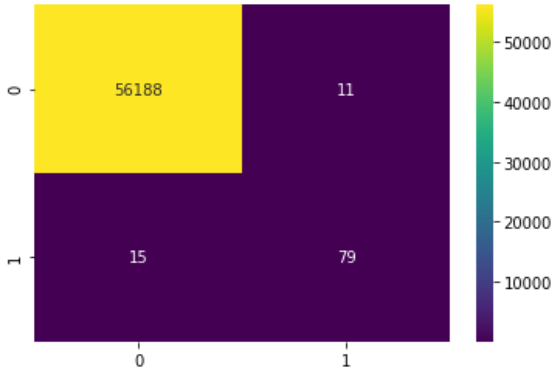

Figure 5.4.a Confusion Matrix - before oversampling

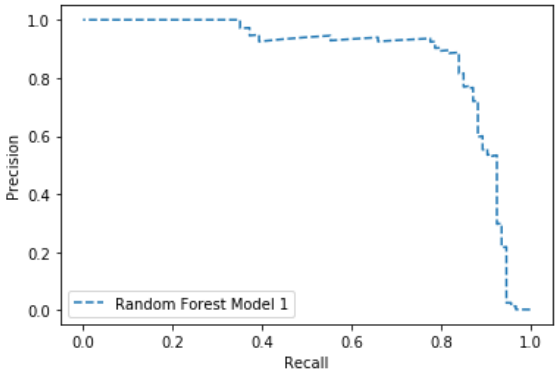

Figure 5.4.b Precision-Recall Curve - before oversampling

After oversampling, the model achieved a precision of 0.160, a recall of 0.926, f1-score of 0.273 and AUC of 0.869, as shown in the confusion matrix and precision-recall curve below. We can notice that the two curves are nearly the same.

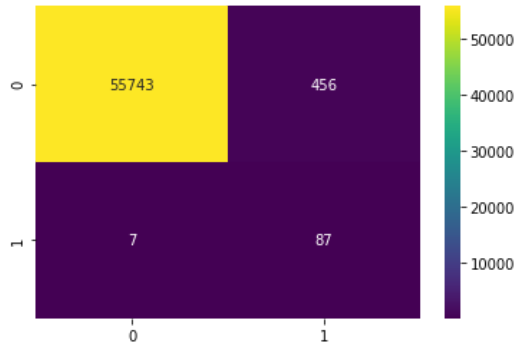

Figure 5.4.c Confusion Matrix - after oversampling

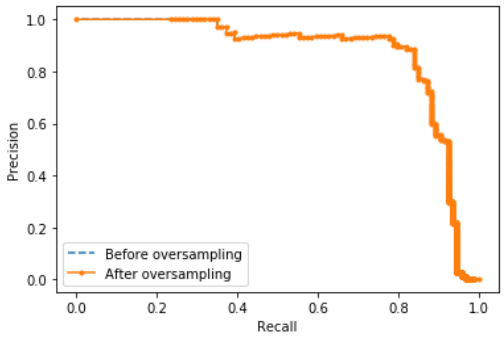

Figure 5.4.d Precision-Recall Curve - after oversampling

By threshold moving, we found an optimal threshold of 0.780 (for *Class* 1) corresponding to the maximum f1-score, under which the model can achieve a higher precision of 0.888 and a recall of 0.840. The confusion matrix with 0.780 threshold is shown below.

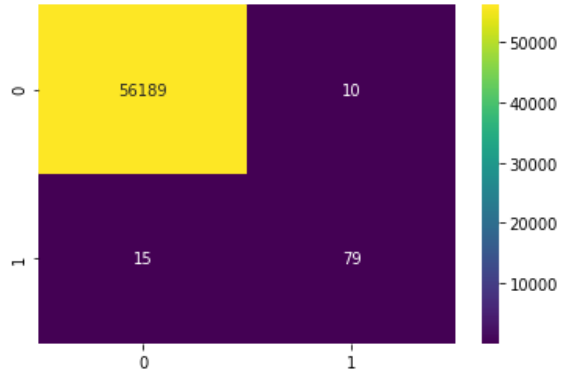

Figure 5.4.e Confusion Matrix - after oversampling - threshold = 0.780

Summarizing the results, we have the table below for ANN Classifier:

Training Set	**Precision**	**Recall**	**F1 Score**	**AUC**
Before Oversampling (Threshold = 0.5)	0.878	0.840	0.859	0.869
After Oversampling (Threshold = 0.5)	0.160	0.926	0.273	0.869
After Oversampling (Threshold = 0.780)	0.888	0.840	0.863	0.869

6. Discussion

We summarized the best model from each classifier as shown in the table below.

Classifier	Model	Precision	Recall	F1-score	AUC - Precision-Recall Curve
Logistic Regression	After Oversampling; Threshold = 0.985	0.830	0.830	0.830	0.846
Support Vector Machine	Before Oversampling; Threshold = 0.5	**0.915**	0.798	0.852	0.854
Random Forest	After Oversampling; Threshold = 0.596	0.867	**0.904**	**0.885**	**0.906**
Artificial Neural Network	After Oversampling; Threshold = 0.780	0.888	0.840	0.863	0.869

It can be noticed that among all classifiers, considering the trade-off between precision and recall, random forest provides the best prediction result with precision of 0.867 and recall of 0.904. This result is achieved using oversampled data, which proves that oversampling can help to improve the model performance in the case of an imbalanced dataset.

7. Conclusion

7.1 Summary of project achievements

In this project, we have explored and implemented some popular data mining techniques, including data preprocessing, different classification algorithms and appropriate evaluation metrics. We used the oversampling technique to deal with the imbalance of the original dataset, which proved to be useful in improving the model's performance in most cases. We built classification models using various machine learning algorithms and discovered patterns in the data for predicting whether a transaction is fraudulent. In the end, we have achieved the best recall of 0.904 and precision of 0.867 with random forest classifier, which satisfies the goal of this project.

7.2 Directions for improvements

Future improvements may involve building an ensemble classifier to combine the decisions of individual classifiers using voting. This may further improve the classification accuracy and make the results more robust.

8. References

[1] Credit Card Fraud Detection. (n.d.) Retrieved September 29, 2020, from https://www.kaggle.com/mlg-ulb/creditcardfraud

[2] Cvetanka Eftimoska. (April 30, 2020). Data preprocessing for Machine Learning in Python. *Towards data science*. Retrieved October 9, 2020, from https://towardsdatascience.com/data-preprocessing-for-machine-learning-in-python-2d465f83f18c

[3] Jason Brownlee. (February 10, 2020). A Gentle Introduction to Threshold-Moving for Imbalanced Classification. *Machine Learning Mastery*. Retrieved October 25, 2020, from https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/

[4] Scikit-learn Machine Learning in Python. (n.d.) Retrieved October 18, 2020, from https://scikit-learn.org/stable/

9. Appendix

Scripts/Code

There are five scripts implemented in this project which can be found under the `code` folder.

`data_preprocessing.ipynb`

`logistic_regression.ipynb`

`svm.ipynb`

`random_forest.ipynb`

`ann.ipynb`

Implementation Guideline

For running the scripts to reproduce the results, please refer to `README.txt` in the `code` folder.