Credit Card Fraud Detection utilizing Machine Learning Algorithm

Author: Ying (Flora) Dong

Abstract

Credit card fraud has become an important issue for the financial service industry today. Detecting fraudulent transactions is thus growing to be a major topic for data mining and machine learning research, given the large volume of real-time transactions and the speed and accuracy requirement of in-time detection of abnormal cases. There are two major challenges facing fraud detection today. First, large amount of transaction data makes it hard to build a fast and efficient algorithm, and secondly, the highly imbalanced nature of the dataset, due to the lack of fraudulent cases as compared to normal cases, makes it difficult to build a high accuracy model. It thus crucial to apply statistical sampling techniques to build a more robust data sample. As a result, the performance of any fraud detection algorithm is greatly affected by different sampling approaches and selection of modeling technique(s) applied. In this study, I investigate the performance of k-nearest neighbor, logistic regression, Naive bayes, XG boost and other popular machine learning algorithms on a highly skewed credit card fraud data. In addition to test those machine learning algorithms, different sampling techniques are experimented, and finally a synthetic over-sampling approach is carried out on the skewed data to create a more balanced training dataset. The work is implemented in Python. The performance of different ML techniques is evaluated based on accuracy, precision, recall, and confusion matrix, among which recall is the most important metric to focus on, given the nature of this problem, as the cost of misclassification of a fraud case is much higher than the cost of normal case. The comparative results show that among all testing methodologies, k-nearest neighbor achieves the highest precision and recall on test data, outperforming other techniques. At the end, we further tune k-nearest neighbor with re-sample data using cross validation to achieve optimal model performance.

1. Problem Statement

A typical organization loses an estimated of 5% per year of its revenue due to fraud. Fraud activities can come in various types, within different organizations. For example, insurance claim fraud, credit card transaction fraud, money laundering, etc. Among those fraudulent activities, credit card transaction fraud is the most commonly seen one that resulted in large losses for both credit card companies and customers who possess the cards.

To tackle this problem, for each credit card transaction, an automatic system needs to be built to identify whether a transaction is normal or fraudulent, by applying efficient computational algorithms. Thus, it requires training a machine learning algorithm to identify abnormal (fraud) observations from normal observations.

Credit card transactions involves large amount of data, in terms of sample size and number of features. Also, for the historical transaction data, most of the transactions (around 99.8%) are not fraudulent, the highly imbalanced nature of this dataset makes it hard to detect fraudulent ones, thus, some re-sample techniques are required. Further, with enormous real-time

transaction data processed every day, the detection model built needs to fast and simple enough to respond to scam in time. Lastly, the detection algorithm needs to have high accuracy, so as to be able to identify the abnormal activities correctly.

2. Data Source

I download credit card transaction level data from Kaggle. This dataset contains labels of fraud versus non-fraud as well as features. By analyzing basic descriptive statistics of the dataset, below is a summary:

- This dataset has 284,807 observations
- It contains 28 anonymized features after principal component analysis and two real-valued features which are time and amount of the transaction
- The last columns named "Class" contains labels for the dataset. Thus, given a sample, we would know whether it belongs to fraud or non-fraud category

3. Preliminary Data Analysis

3.1 Dataset Summary

The original shape of the fraud dataset contains 284,807 rows and 31 columns. The first 30 columns are feature vectors. The last column contains only 0/1 values are labels, where 1 corresponds to fraud cases and 0 corresponds to non-fraud cases. Columns v1-v29 are original features after PCA transformation due to confidentiality issues. Column Time and Amount are real-valued features that have not been PCA transformed. Time indicates the number of seconds elapsed between this transaction and the first transaction in this dataset. Amount indicates transaction amount. The last column Class are labels with 0 and 1, where 1 indicates fraud versus 0 indicates non-fraud cases. Out of 284,807 transactions, only 0.17% of them are fraud, indicating the fraud cases take very little percentage of the total samples.

```
Total number of cases are 284807
Number of Non-fraud cases are 284315
Number of Non-fraud cases are 492
Percentage of fraud cases is 0.17
```

3.2 Descriptive Data Analysis

Table below shows the summary statistics for a few columns in the dataset.

Table 1, Summary statistics of data

	Time	V1	V2	V3	V4
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e−15	5.688174e−16	−8.769071e−15	2.782312e−15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00
min	0.000000	−5.640751e+01	−7.271573e+01	−4.832559e+01	−5.683171e+00
25%	54201.500000	−9.203734e−01	−5.985499e−01	−8.903648e−01	−8.486401e−01
50%	84692.000000	1.810880e−02	6.548556e−02	1.798463e−01	−1.984653e−02
75%	139320.500000	1.315642e+00	8.037239e−01	1.027196e+00	7.433413e−01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01

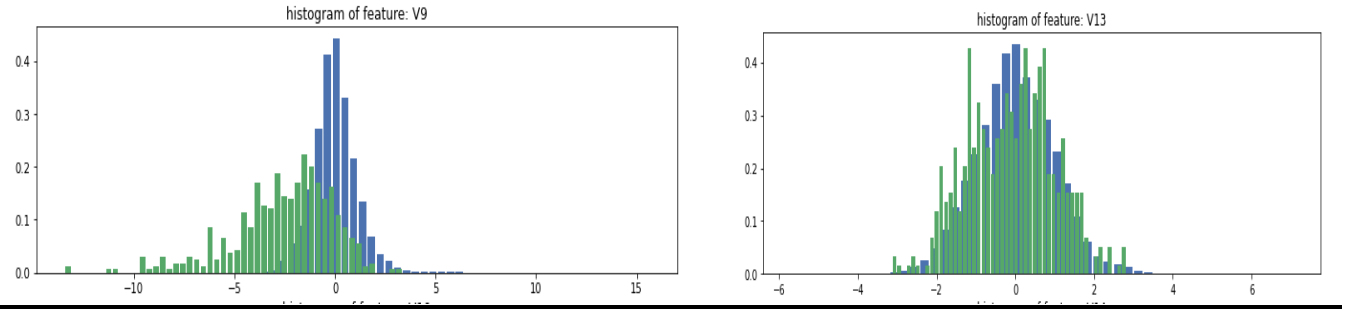
	V25	V26	V27	V28	Amount
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000
mean	1.453003e−15	1.699104e−15	−3.660161e−16	−1.206049e−16	88.349619
std	5.212781e−01	4.822270e−01	4.036325e−01	3.300833e−01	250.120109
min	−1.029540e+01	−2.604551e+00	−2.256568e+01	−1.543008e+01	0.000000
25%	−3.171451e−01	−3.269839e−01	−7.083953e−02	−5.295979e−02	5.600000
50%	1.659350e−02	−5.213911e−02	1.342146e−03	1.124383e−02	22.000000
75%	3.507156e−01	2.409522e−01	9.104512e−02	7.827995e−02	77.165000
max	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000

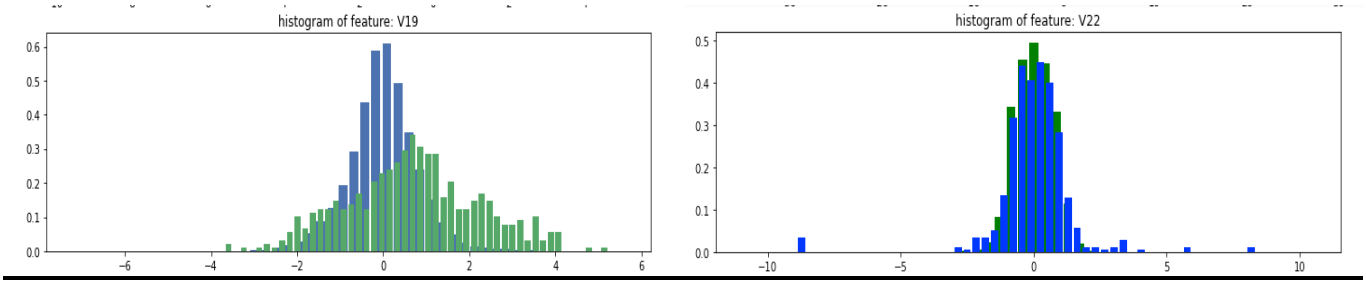
Based on summary statistics from the data, we can tell that for features of v1-v28, they have mean close to zero and standard deviation less than 2 in most cases. I would assume that all those features are already standardized and normally distributed.

3.3 Feature Univariate Analysis

To further prove my assumptions that features of v1-v28 are standardized and normally distributed, I plot the histogram of each feature separately for the fraud and non-fraud class. Figure 1 shows sample histograms for three features. Green represent non-fraud cases and blue represents fraud cases. By visually inspect the univariate plot for each feature, I could tell that they are pretty much close to normal distribution. Thus, no additional standardization needs to be done.

Figure 1, Feature Histogram





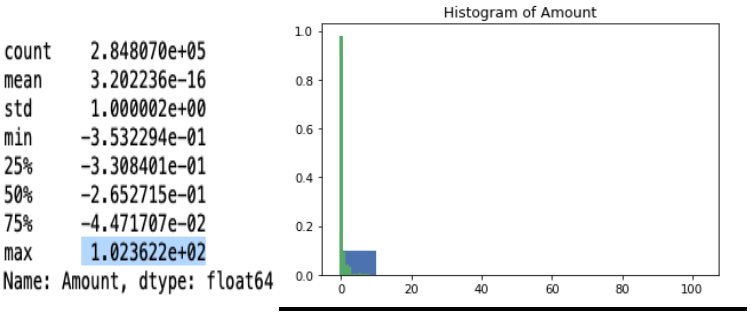
For the real-valued features time and amount, I would drop Time column since the feature itself does not make sense to predict fraud. For the Amount column, first, I examined the summary statistics it separately for fraud and non-fraud cases. By analyzing table below. I found out that for fraud cases, the median transaction amount is less than non-fraud cases, however, the 75% percentile amount is more than that of fraud case, though the max amount of fraud cases is only 10% of that of non-fraud cases. Thus, we can conclude that half of fraud transactions have amounts less than non-fraud transactions.

```
Amount detail for the fraud transaction
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
----------------------------------------
Amount detail for the non-fraud transaction
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

to be consistent with other features, I would standardize it first and then plot the above histogram to visually inspect the distribution.

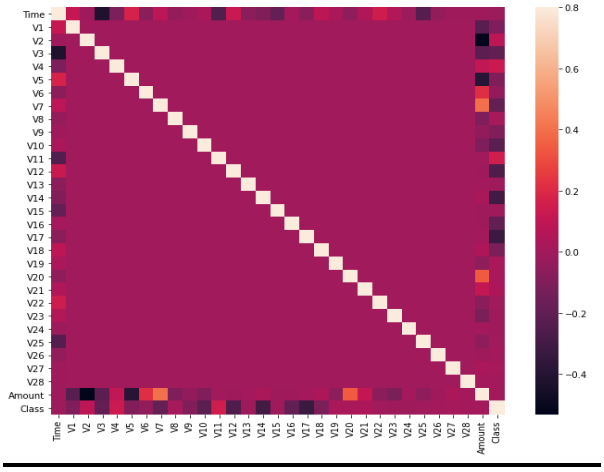
By further examine the summary statics and histogram of Amount column after applying standardization, I found out that this column of data is pretty much skewed. As shown from the summary statistics, 75% value is -0.04471, while max value is 102.36. Also shown by the histogram, we cannot clearly observe a normally distributed shape, as the distribution has been skewed by a few outliers. Thus, due to the non-normality of this column, I would drop this feature during later modeling stage.

Figure 2, Feature Histogram



In addition to the histogram, I also visualize correlation among all 28 features with a heatmap. Correlation is small among all variables.

Figure 3, Features Correlation Matrix



2. Methodology

2.1 Supervised Learning Algorithm

To test different supervised machine learning algorithms, I have built seven machine learning models, by utilizing seven different algorithms and test model performance. To avoid model overfitting, I split the data into 80% training and 20% testing. To avoid randomness, I use the same random seed in all splitting to ensure the same train and test data is applied in different models.

For model evaluation metrics, I would analyze their precision, recall, F1-score, accuracy, confusion matrix, ROC curve and r square.

2.11 K Nearest Neighbors

First, I tried KNN classifier, by using five neighbors. This classifier assigns the new test point a label by taking a majority vote over Kth training points the test point is closet to. This is a non-parametric non-linear method that performs well on different types of data shape.

Table 2, KNN Performance Statistics

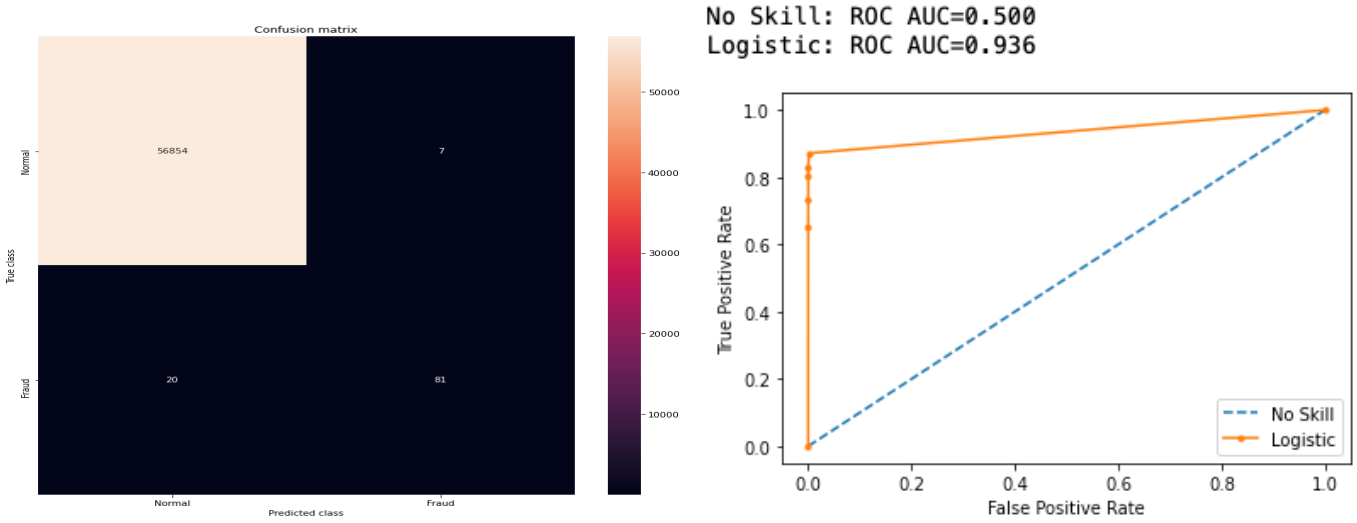
```
the Model used is KNN
The accuracy is  0.9995259997893332
The precision is 0.9204545454545454
The recall is 0.801980198019802
The F1-Score is 0.8571428571428572
```

```
r square 0.7321984251677836
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.92      0.80      0.86       101

    accuracy                           1.00     56962
   macro avg       0.96      0.90      0.93     56962
weighted avg       1.00      1.00      1.00     56962
```

Figure 4, KNN Confusion Matrix and ROC Curve



The overall model accuracy with KNN is pretty good, at around 99.95%, showing that almost 99.5% of time, the model is correct in distinguishing between fraud and non-fraud transactions. However, given the highly imbalanced nature of this dataset, i.e., more than 95% of transactions are fraud, we cannot simply use accuracy, as the majority class will dominate the calculation, which will bias results. As a result, to analyze model result, we should pay more attention to metrics such as precision and recall, as the calculation of those two will be more balanced for the minority class. In the case of KNN, Precision is lower than accuracy, at around 92.05%, while recall is even lower, at around 80.2%. Precision is the percentage of true positive among all predicted positive, while recall is the percentage of true positive among all actual positive. In the case of fraud detection, predicting a fraud transaction as a non-fraud has more

losses than predicting a non-fraud cases as fraud, therefore, we prefer to have a higher recall than precision, and pay more attention to minimize the model's predicted false negative while maximizing the model's predicted true positive.

Looking at the confusion matrix, we can tell that out of 56,861 non-fraud cases, the model predicts 56,854 as non-fraud and only makes 7 mistakes of predicting non-fraud as fraud. Out of 101 non-fraud cases, the model predicts 81 of them as fraud and makes 20 mistakes of predicting fraud as non-fraud. So clearly, we can see here that the model makes more mistakes to predict fraud cases as compared to non-fraud cases. Further analyzing the ROC curve, we can see that model has an AUC of 0.94, again which is a high number, showing that 94% of the time, the model is able to differentiate fraud and non-fraud cases.

Further analyzing the precision and recall separately for each class, we can observe that the precision for class 1 is at 92%, while recall is at 80%. Both are above 80%, at an acceptable level. The 92% precision shows that among all predicted fraud classes, 92% of them are correct; while the 80% recall shows that among are actual fraud cases, the model predicts they are fraud 80% of the time. Here, we notice that recall is slightly lower than precision for class 1, therefore, we would focus on analyzing recall for different algorithms to eventually improve the model's predictive power.

2.12 Logistic Regression

Second, I tried logistic regression classifier, by using max iterations of 200 and "liblinear" as solver. This classifier assigns the new test point a label based on thresholding probability output from the logistic model. This is a parametric non-linear method that is commonly applied and relatively robust to model mismatch.

The overall model accuracy with logistic regression is also pretty good, at around 99.92%, showing that almost 99.2% of time, the model is correct in distinguishing between fraud and non-fraud transactions. However, given the highly imbalanced nature of this dataset we cannot simply use accuracy to evaluate the model results, as the majority class will dominate the calculation, which will bias results. When compared with KNN, the overall accuracy is 0.35% lower.

Further analyzing the overall precision and recall for logistic regression, Precision is 4% lower than KNN, at around 88%, while recall is 17% lower than KNN, at around 63%. Here in the case of fraud detection, recall is a more accurate representation of model's predictive power, as it measures the model's ability to generate well on minority classes. Here, the model's recall is only at 63%, showing that among all fraud cases, model can only predict them as fraud 63% of time, which is below the 80% acceptable level. When further analyzing the precision and recall just for the class 1 (fraud class), the precision and recall is at 88% and 63%, 4% and 17% lower than that of KNN.

Looking at the confusion matrix, we can tell that out of 56,861 non-fraud cases, the model predicts 56,852 as non-fraud and only makes 9 mistakes of predicting non-fraud as fraud. This is comparable to KNN's prediction. However, out of 101 fraud cases, the model predicts 64 of

them as fraud and makes 37 mistakes of predicting fraud as non-fraud. When compared with KNN, the model's predictive power on the minority class is even worse. It makes 17 less correct fraud predictions while making 17 more mistakes of predicting fraud as non-fraud cases, which we call false negative in statistical term. In the case of fraud detection, it causes more to predict fraud as non-fraud, compared to predicting non-fraud as fraud, therefore, we should focus more on minimizing false negative.

So clearly, we can see here that logistic regression makes more mistakes when predicting fraud as compared to KNN, and when predicting non-fraud cases, the two models are at similar level of accuracy.

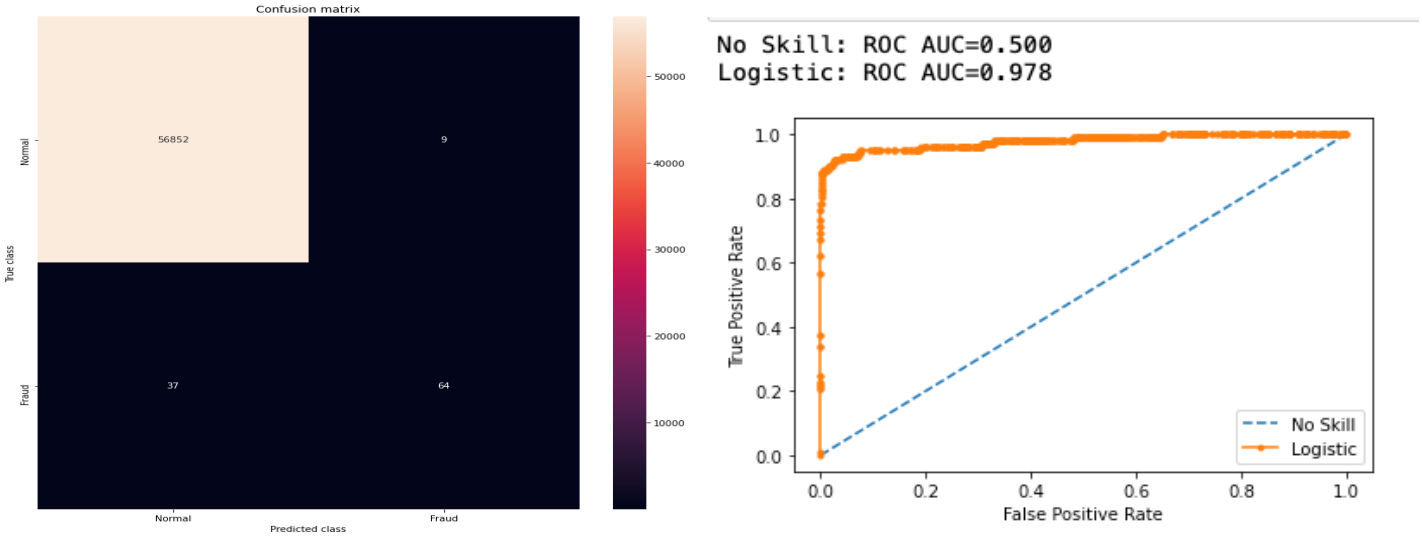
Table 3, Logistic Regression Performance Statistics

```
the Model used is LogisticRegression
The accuracy is  0.9992099996488887
The precision is 0.8888888888888888
The recall is 0.6336633663366337
The F1-Score is 0.7398843930635838
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.88	0.63	0.74	101
accuracy			1.00	56962
macro avg	0.94	0.82	0.87	56962
weighted avg	1.00	1.00	1.00	56962

r square 0.5437454651006683

Figure 5, Logistic Regression Confusion Matrix and ROC Curve



2.13 Linear SVM

Third, I tried SVM classifier, by applying linear kernel. This classifier assigns the new test point a label based a hyperplane shaped decision boundary calculated from the model. This is a non-parametric non-linear method that is commonly applied and relatively robust to model mismatch and generates non-noisy prediction as compared to classifiers such as KNN and logistic regression.

The overall model accuracy with linear SVM is comparable to KNN, at around 99.94%, showing that almost 99.94% of time, the model is correct in distinguishing between fraud and non-fraud transactions

Further analyzing the overall precision and recall for linear SVM, both precision and recall is at 82%. When compared with KNN, precision is 10% lower, however, recall is 2% higher. Since in the case of fraud detection, recall is more important than precision, thus, in the case of predicting fraud, linear SVM is slightly better than KNN.

Further analyzing the precision and recall just for the class 1 (fraud class), we notice there that the precision and recall are both at 82%, thought precision is lower 4% than KNN, recall is 2% higher.

Looking at the confusion matrix, we can tell that out of 56,861 non-fraud cases, the model predicts 56,843 as non-fraud and makes 18 mistakes of predicting non-fraud as fraud. This is comparable to KNN's prediction. More importantly, out of 101 fraud cases, the model predicts 83 of them as fraud and makes 18 mistakes of predicting fraud as non-fraud. When compared with KNN, the model's predictive power on the minority class is slightly. It makes 2 more correct fraud predictions while making 2 less mistakes of predicting fraud as non-fraud cases. When compared the number of true positive and false negative with that of KNN, we can see that linear SVM has higher number of true positive and lower number of false negatives. So clearly, we can generate the conclusion that in the case of fraud detection, linear SVM outperforms KNN.

Table 4, Linear SVM Performance Statistics

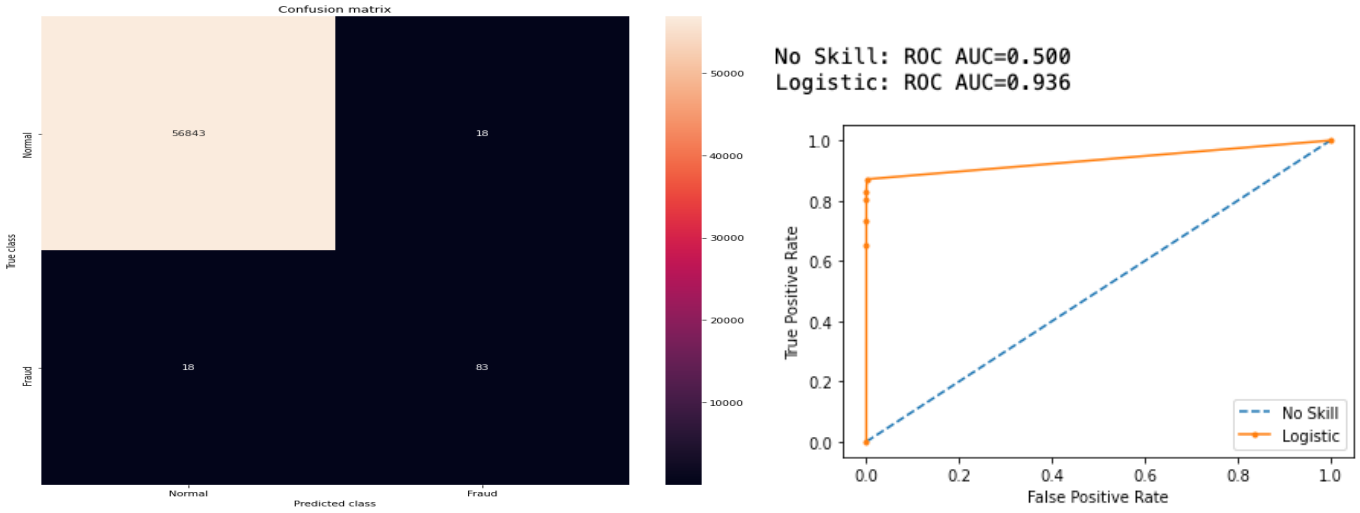
```
the Model used is Linear SVM
The accuracy is  0.9993679997191109
The precision is 0.8217821782178217
The recall is 0.8217821782178217
The F1-Score is 0.8217821782178217
```

```
r square 0.6429312335570447
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.82      0.82      0.82       101

    accuracy                           1.00     56962
   macro avg       0.91      0.91      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

Figure 6, Linear SVM Confusion Matrix and ROC Curve



2.14 Kernelized SVM

To compare with linear SVM classifier, I applied non-linear kernelized SVM, using radial basis kernel and gamma set by using the "median trick" equation taught in class. This classifier assigns the new test point a label based a hyperplane shaped decision boundary calculated from the model, by applying non-linear transformation to the data so that they are linearly separable. The kernelized SVM tends to perform well when the data is not linearly separable. This is also a non-parametric non-linear method that is commonly applied and relatively robust to model mismatch and generates non-noisy prediction as compared to classifiers such as KNN and logistic regression.

The overall model accuracy with linear SVM is comparable to kernelized SVM, at around 99.93%, showing that almost 99.94% of time, the model is correct in distinguishing between fraud and non-fraud transactions

Further analyzing the overall precision and recall for kernelized SVM, precision is at 92% while recall is at 67%. When compared with linear SVM, kernelized SVM achieves 10% higher precision, however, 15% lower recall. Further analyzing the precision and recall just for the class 1 (fraud class), we notice there that the precision and recall are at the same magnitude as the overall. Even though precision is higher, in the case of fraud detection, our objective is to minimize false negatives, therefore, we should focus on improving model's recall.

This is further proved by the confusion matrix, where out of 101 fraud cases, the model predicts 68 of them correct and 33 of them as non-fraud. This number of correct fraud predictions are much lower as compared to 83 in linear SVM. Thus, even though kernelized SVM makes less mistakes of predicting non-fraud as fraud, it predicts far less fraud for the fraud cases. Therefore, we conclude that linear SVM outperforms kernelized SVM for the fraud detection problem.

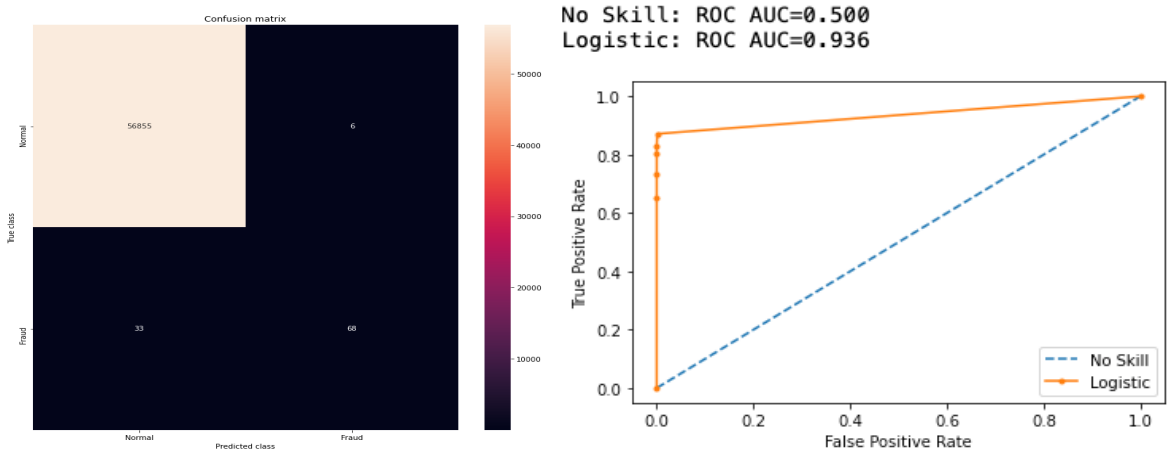
Table 5, Kernelized SVM Performance Statistics

```
the Model used is Kernal SVM
The accuracy is  0.9993153330290369
The precision is 0.918918918918919
The recall is 0.6732673267326733
The F1-Score is 0.7771428571428572
r square 0.6131755030201318
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.92      0.67      0.78       101

    accuracy                           1.00     56962
   macro avg       0.96      0.84      0.89     56962
weighted avg       1.00      1.00      1.00     56962
```

Figure 7, Kernelized SVM Confusion Matrix and ROC Curve



2.15 Naïve Bayes

Naïve Bayes is a type of probabilistic model that classifies a new data point based on conditional probability estimation such as kernel density estimation and given threshold. Here I applied gaussian multivariate normal Naïve Bayes estimations, where the density estimation for each class is based on multivariate normal gaussian distributions. There is no parameter to tune in this setting, however, strong assumptions, such as features are independent and normally distributed, need to be satisfied in order to apply the model.

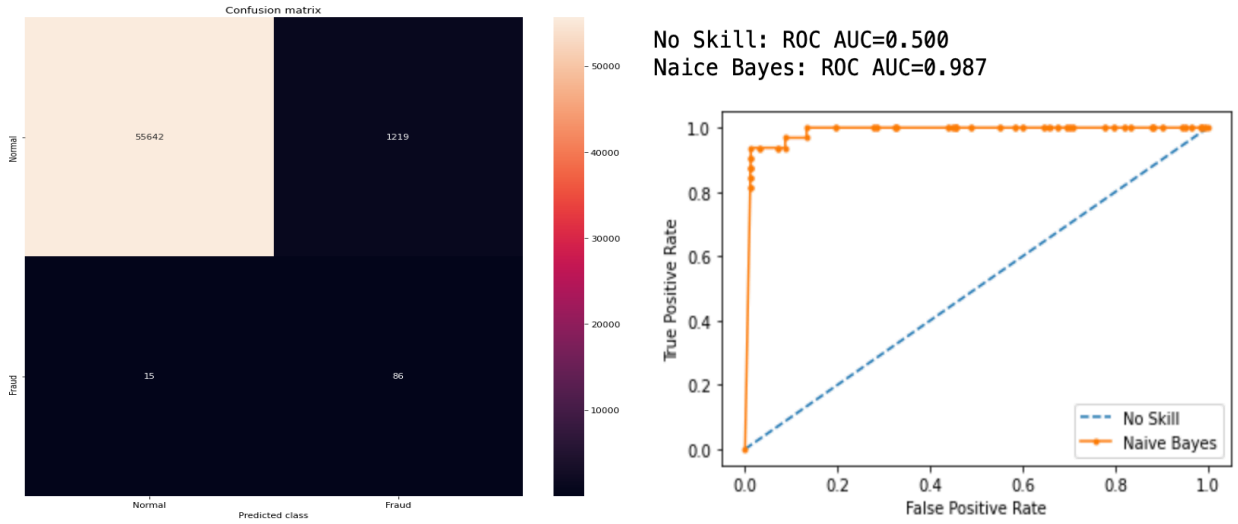
To compare with KNN (the best performing model so far), the overall accuracy is at 98%, comparable with KNN. Further analyzing the overall precision and recall for naïve bayes, here we notice that precision is very low, only at around 6%, by further analyzing confusion matrix, the low precision number is caused by the high number of false positives, where the algorithm predicts 1,219 non-fraud cases as fraud. This number is so far the highest among all algorithms. In addition, we notice that out of 101 fraud cases, the model predicts 86 of them correct and 15 of them incorrect. The correct predicted fraud cases are also highest among all algorithms so far, this leads to the highest recall overall.

Table 6, Naïve Bayes Performance Statistics

```
the Model used is Naive Bays
The accuracy is  0.9783364348161933
The precision is 0.06590038314176246
The recall is 0.8514851485148515
The F1-Score is 0.12233285917496445
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	56861
1	0.07	0.85	0.12	101
accuracy			0.98	56962
macro avg	0.53	0.92	0.56	56962
weighted avg	1.00	0.98	0.99	56962

Figure 8, Naïve Bayes Confusion Matrix and ROC Curve



Thus, we can conclude that Naïve Bayes classifier gives the highest accuracy when predicting fraud, however, meanwhile, it also assigns many non-fraud cases as fraud. The result of this algorithm shows that the probability density estimation works well on minority (fraud) cases, while it tends to predict more non-fraud cases as fraud. Considering the trade-off between precision and recall, though the recall is 3% higher than KNN, the precision is much lower. Given this, we still prefer KNN over Naïve Bayes as KNN has achieved a better balance between precision and recall while maintaining a high recall.

2.16 Decision Tree

Decision tree is a tree-based algorithm where a new test point is labeled along the tree structure. The tree is created by working top-down, choosing a variable at each step that best splits the set of items, based on some statistical measure, such as Gini impurity. The created tree structure is visualized below. I choose max_depth=3 and min_sample_leaves=100 as conventional values.

To compare with KNN (the best performing model so far), the overall accuracy is at similar level. Further analyzing the overall precision and recall for decision tree, the level of both precision and recall are lower than KNN, at 76% and 75% separately. Compared with Naïve Bayes, the precision level is much higher.

Further analyzing the confusion matrix, we notice that the number of false negatives is much lower than Naïve Bayes, this is the reason why precision is much higher compared with Naïve Bayes.

Given the complexity of the tree-based model and lower recall, KNN so far is still the best model to predict fraud.

Table 7, Decision Tree Performance Statistics

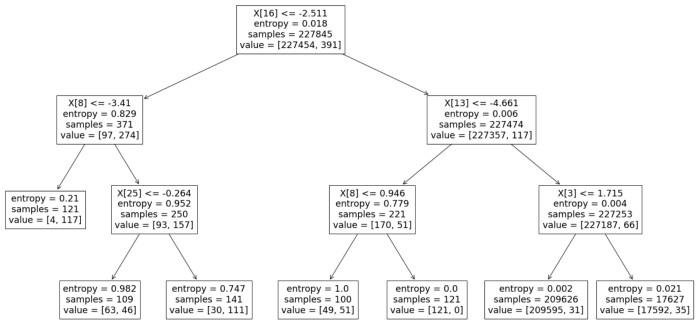
```
the Model used is Decision Trees
The accuracy is  0.9991397773954567
The precision is 0.76
The recall is 0.7524752475247525
The F1-Score is 0.7562189054726369
```

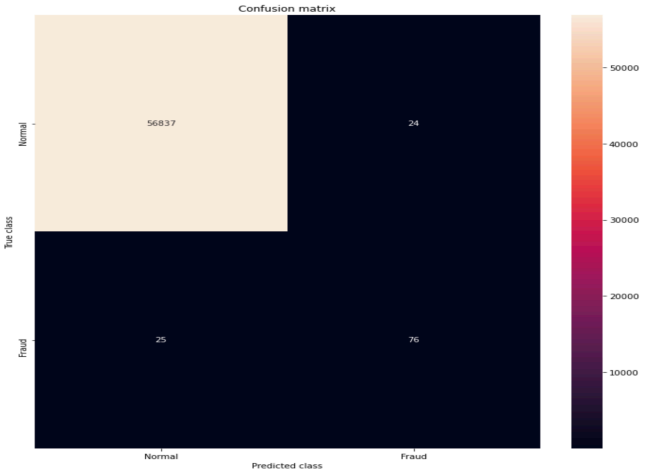
```
r square 0.5139897345637554
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.76      0.75      0.76       101

    accuracy                           1.00     56962
   macro avg       0.88      0.88      0.88     56962
weighted avg       1.00      1.00      1.00     56962
```

Figure 9, Decision Tree Confusion Matrix and Tree Plot





2.17 Random Forest

After decision tree, I applied random forest, which is an ensemble of decision trees, by randomly selecting features and samples to build multiple decision trees with output labels based on modes from the multiple trees. This is a non-parametric based model where we cannot visualize how decisions are made.

Table 8, Random Forest Performance Statistics

```
the Model used is Random Forest
The accuracy is  0.9990344440153085
The precision is 0.859375
The recall is 0.5445544554455446
The F1-Score is 0.6666666666666666
```

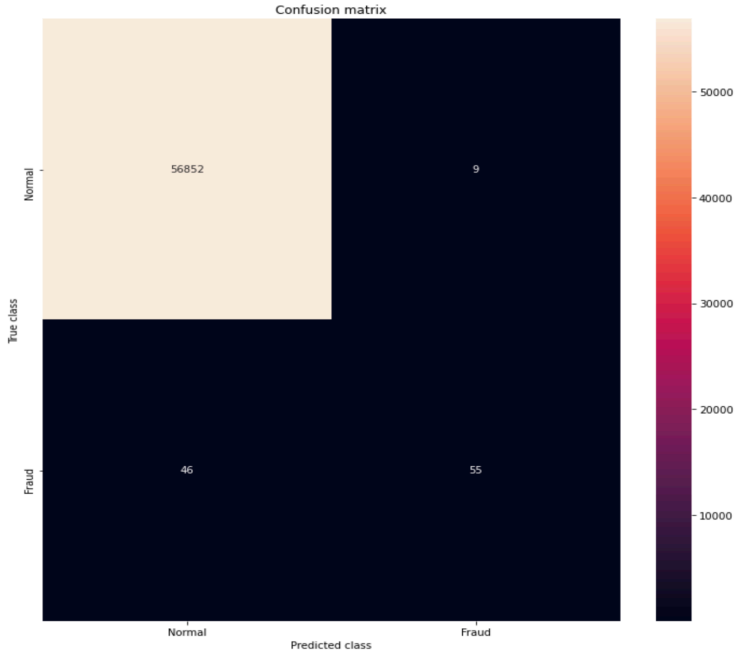
```
r square 0.45447827348992953
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.86      0.54      0.67       101

    accuracy                           1.00     56962
   macro avg       0.93      0.77      0.83     56962
weighted avg       1.00      1.00      1.00     56962
```

The overall model accuracy and precision levels are comparable with KNN (the highest performing model so far); however, the recall is lower, meaning the model makes more mistakes of predicting actual fraud as non-fraud cases. Given the complexity of this model and lower recall, we still think KNN is the best performing model on the fraud detection case.

Figure 10, Random Forest Confusion Matrix



2.18 Neural Network

Given the popularity of neural network, also applied it on the fraud dataset, by using a simple structure neural network with 2 layers, and each layer has 10 and 9 hidden unites, with “Relu” as the activation function.

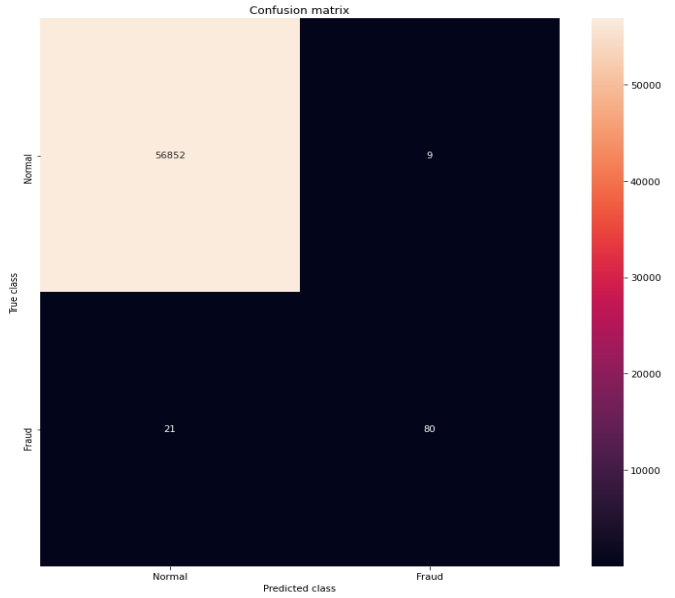
Table 9, Neural Network Performance Statistics

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.91	0.79	0.85	101
accuracy			1.00	56962
macro avg	0.95	0.90	0.92	56962
weighted avg	1.00	1.00	1.00	56962

the Model used is Neural Network
The accuracy is 0.9994733330992591
The precision is 0.898876404494382
The recall is 0.7920792079207921
The F1-Score is 0.8421052631578948

The classification result from neural network is comparable to that of KNN, in terms of accuracy, precision and recall, with a slightly lower recall number. Given the complexity of this model structure, we would still prefer KNN as the champion model as it predicts the highest number of precision, recall while requires low computer memory and has relatively faster speed of training.

Figure 11, Neural Network Confusion Matrix and ROC Curve



2.19 XG Boost

The XG Boost model is another popular machine learning algorithm that builds on weak learners and gradually adds weights to weak learners in each sequential iteration and eventually builds the most accurate model based on average of previous results.

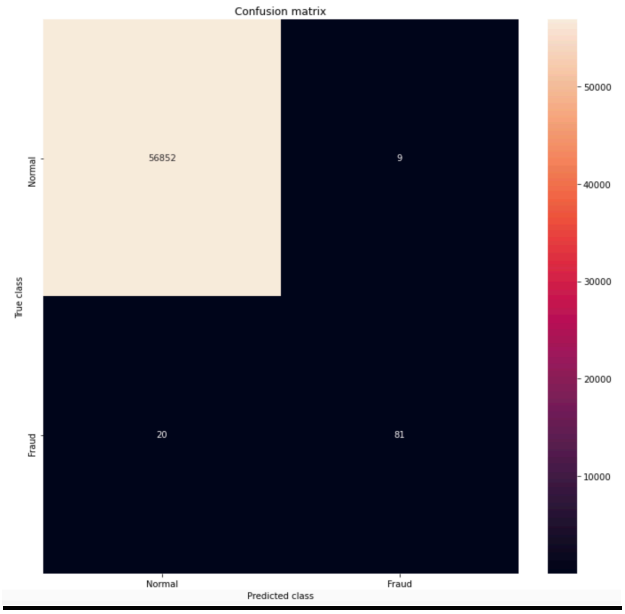
Table 10, XG Boost Performance Statistics

r square 0.7123612714765083

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.90	0.80	0.85	101
accuracy			1.00	56962
macro avg	0.95	0.90	0.92	56962
weighted avg	1.00	1.00	1.00	56962

the Model used is XG Boosting
The accuracy is 0.9994908886626171
The precision is 0.9
The recall is 0.801980198019802
The F1-Score is 0.8481675392670157

Figure 12, XG Boost Confusion Matrix and ROC Curve



The overall accuracy of the XG boost model is comparable to that of KNN, in addition, the precision and recall values are all comparable with that of KNN. Given the more complex structure of this model, we would end up selecting KNN as the champion model for this problem.

In summary, among the 9 popular machine learning algorithms, KNN performs the best on the raw data, achieving a precision of 92% and recall of 80%. It shows that it can predict fraud as well as non-fraud cases well, especially, for predicting fraud cases, it can identify fraud 80% of time among the actual fraud, which is a high number. Considering the simplicity of this algorithm compared to other more complex ones, we would choose KNN as the final fraud detection algorithm in the later study.

2.2 Resample

Resample is an important technique that can boost model performance, especially for imbalanced data, like the fraud dataset. There are two sampling techniques: Up sample for minority cases and down sample for majority cases.

First, I would try up-sample the minority cases. given that the minority cases take up only 0.17% of total sample size, I choose to up-sample the minority cases to create more fraud data points, to see if it can boost model performance. To up-sample minority cases, I would create more samples by two algorithms: random up-sample and synthetic minority oversampling (SMOT).

Then I would test on down-sample the majority cases to create a balanced sample to compare the model performance with the up-sampling strategies. I would apply clustering methods to find the cluster centroids for the majority classes as down-sample data points. For all those newly created samples, I would apply logistic regression (since it is a fast-running algorithm) on those newly created samples, to see which sampling techniques lead to the best model performance.

2.2.1 Random Resample-Up Sample Minority Cases

The first sampling technique I try is random up sample, in which I would create more samples for the minority classes randomly, by taking random draws from the fraud cases and copying those observations to increase the amount of fraud samples, and eventually balance out the fraud and non-fraud cases.

Before applying resampling, first I split the original dataset into 80% of training and 20% for testing to ensure the results I get are truly out of sample evaluation metrics. Then I only apply the re-sampling techniques on the model training data and use the original 20% of raw data to test model performance. After random up-sampling, as indicated by the table below, now we have equal number of fraud and non-fraud cases in model training sample, both are at the count of 227,454.

Figure 13, Resample Scatter Plot and Confusion Matrix

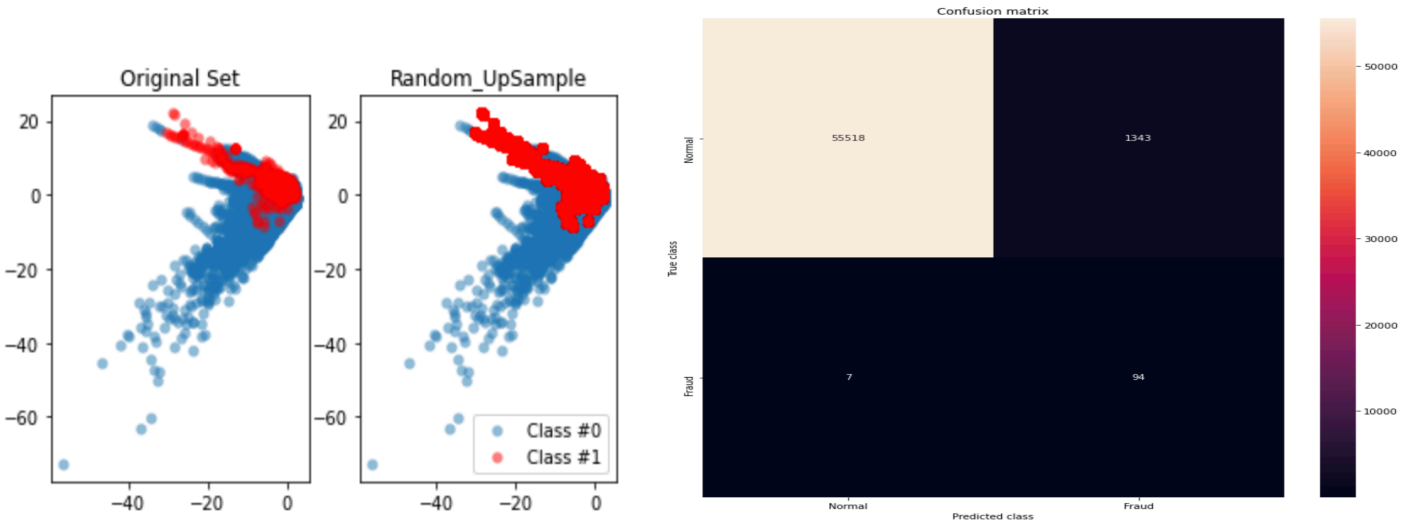


Table 11, Random Resample Model Performance

count of fraud cases 227454
count of non-fraud cases 227454

	precision	recall	f1-score	support
0	1.00	0.98	0.99	56861
1	0.07	0.93	0.12	101
accuracy			0.98	56962
macro avg	0.53	0.95	0.56	56962
weighted avg	1.00	0.98	0.99	56962

Figures above show the scatter plot comparison for the two datasets before and after the random up-sample, and the model evaluation results applying logistic regression on the re-sampled data. When comparing the scatter plots of the two samples of data, we can observe that the general shapes are similar after random up-sample, though the minority classes after up-sampling showed a denser distribution at the original locations. Model performance with random up-sample is relatively good, with recall as high as 93%. Here we notice that precision is

low, at only 7%. This is expected as out of +50,000 test data points, only 101 are actual fraud cases, whereas normal cases 99% are normal cases and it is expected that model makes more mistakes to predict non-fraud as fraud as compared to predict fraud as non-fraud. And in our study, we care more about the fraud predictive power of this model, thus, we focus on recall rather than precision.

2.2.2 Resample with SMOT- Up Sample Minority Cases

Then, I try another up-sampling technique: Synthetic minority Oversampling Technique (SMOT). SMOTE uses characteristics of nearest neighbors of fraud cases to create new synthetic fraud cases to avoid duplicating observations. However, this method only works well if the minority case features are similar. Note that we only use resampling methods on the training set, not on the test set.

Figure 14, Resample Scatter Plot and Confusion Matrix with SMOT

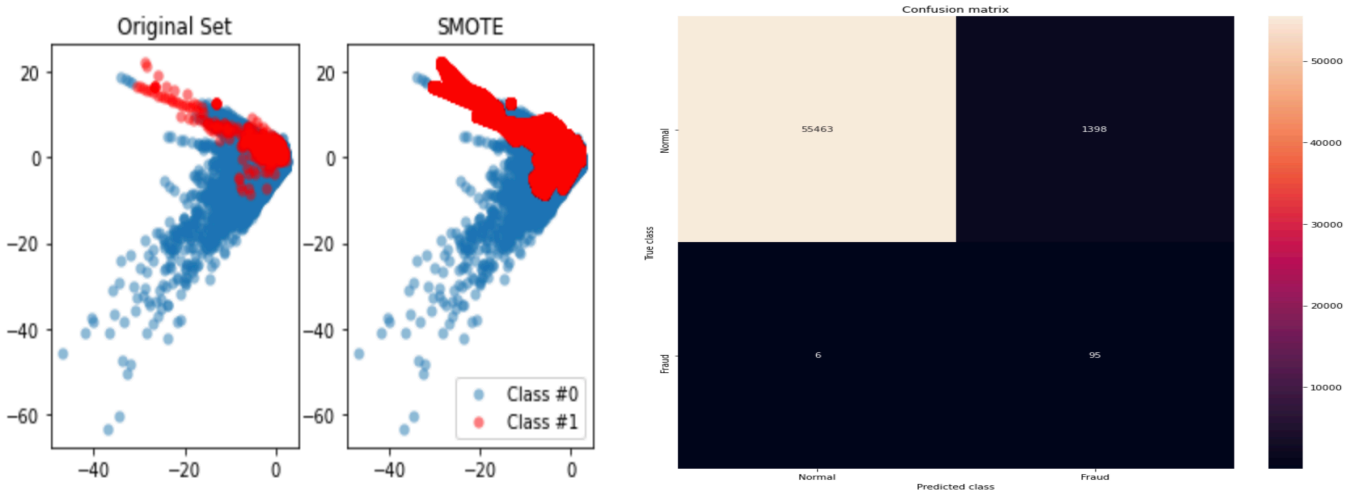


Table 12, Random Resample Model Performance

count of fraud cases 227454
count of non-fraud cases 227454

	precision	recall	f1-score	support
0	1.00	0.98	0.99	56861
1	0.06	0.94	0.12	101
accuracy			0.98	56962
macro avg	0.53	0.96	0.55	56962
weighted avg	1.00	0.98	0.99	56962

Figures above show the scatter plot comparison for the two datasets before and after SMOT up-sample, and model evaluation results applying logistic regression on the same test data as before. When comparing the scatter plots of the raw versus re-sampled data with SMOT, we can observe that the shape of the distribution is similar to that of random up-sample, though the distribution of re-sampled fraud cases is denser than random up-sample at original distributed

locations. SMOT calculates nearest neighbors of fraud cases to create synthetic samples, thus, leads to more data points close to original data points. Model performance with SMOT is slightly higher than that of random up-sample, with recall of 94%, 1% higher than before. Other evaluation metrics are at similar level. Thus, we conclude that sampling with SMOT yields better performance than random up-sample, this further proves that the features of fraud cases have similar shape and distribution.

2.2.2 Down Sample the Majority Class

Third, instead of up-sample minority cases, I try to down-sample the majority cases to balance out fraud and non-fraud cases. I applied cluster centroids down-sample, which is a technique to that uses K-means algorithm to cluster samples of the majority cases by using centers for each cluster to eventually reduce majority cases. After applying down-sample, the total sample size becomes 782, with 391 fraud cases and 391 normal cases.

Figure 15, Resample Scatter Plot and Confusion Matrix

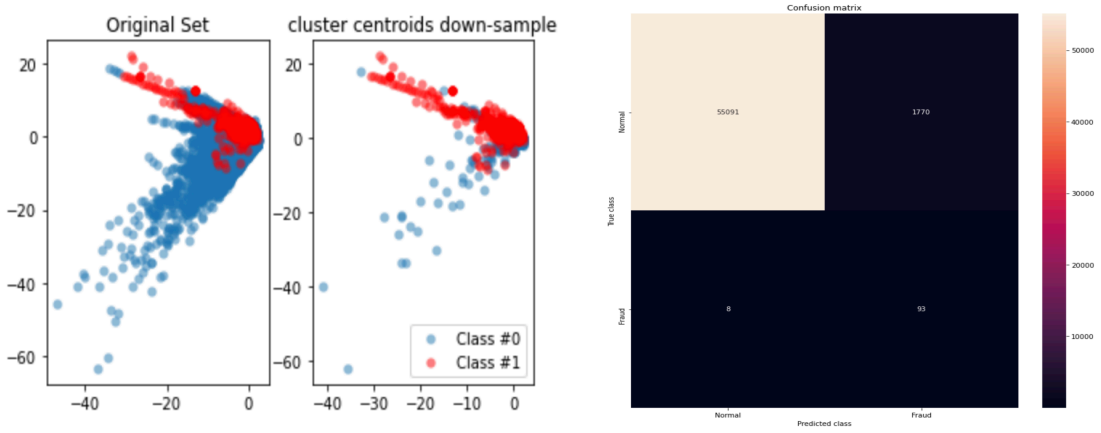


Table 13, Cluster Centroids Down-sample Model Performance

```
count of fraud cases 391
count of non-fraud cases 391
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     56861
           1       0.05      0.92      0.09       101

    accuracy                           0.97     56962
   macro avg       0.52      0.94      0.54     56962
weighted avg       1.00      0.97      0.98     56962
```

Figures above show the scatter plot comparison for the two datasets before and after cluster-centroids down-sample, and model evaluation results applying logistic regression on the same test data as before. When comparing the scatter plots of the raw versus down-sampled data, we can observe that the shape of the distribution for the minority cases are the same, however, since there are much smaller number of majority cases, its distribution is much less dense, but

the general shape is similar to the raw data. Model performance with down-sample shows better predictive power for fraud cases, achieving recall of 92%, 2% lower than SMOT. The precision also suffered at a level of 5%, 1% lower than SMOT. This is as expected, as we significantly lower the number of normal cases in our training data, the model would perform worse on this class consequently.

2.2.3 Mixture of Up-Sample and Down-Sample

Lastly, I try the mixture of up-sample and down-sample. I apply "SMOTEENN" package in Python. It applies over-sampling using SMOTE and cleaning the outliers in majority class and Edited Nearest Neighbors. The count of fraud versus non-fraud cases after re-sampling are 227,454 and 227,118 respectively.

Figure 16, Resample Scatter Plot and Confusion Matrix

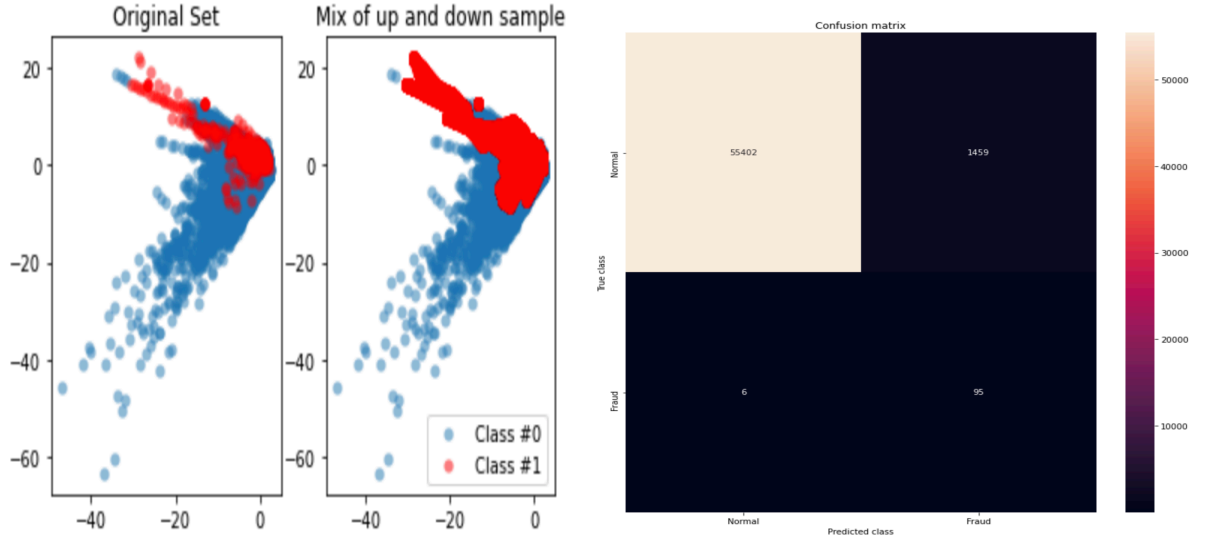


Table 14, Mixture Sampling Model Performance

```
count of fraud cases 227454
count of non-fraud cases 227118
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56861
           1       0.06      0.94      0.11       101

    accuracy                           0.97     56962
   macro avg       0.53      0.96      0.55     56962
weighted avg       1.00      0.97      0.99     56962
```

Figures above show the scatter plot comparison and model evaluation results applying logistic regression on the same test data as before. When comparing the scatter plots, we can observe that the shape of the distribution is similar to SMOT, model performance is also the same as SMOT, with precision and recall at 6% and 94% respectively.

In summary, after experimenting four techniques for resampling, we conclude that up-sample with SMOT yields the best model performance on both fraud and normal cases, and the re-

sampled data created with SMOT has similar shape of distribution as raw data, showing that this algorithm works best to capture the geometry of this fraud dataset.

2.4 Final Model Tuning with Re-Sampled Data

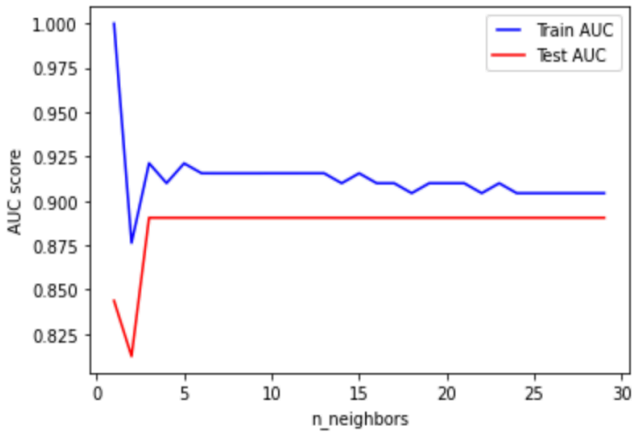
Based on the above re-sampled results, we see that up-sample the minority cases with SMOT leads to the best model performance. In this section, I would explore KNN and XG Boost performance optimization by tuning important model parameters with the re-sampled data and demonstrate final model results I built with those two algorithms. To get faster running time, I choose to run the models with 20% of the data selected from original dataset randomly.

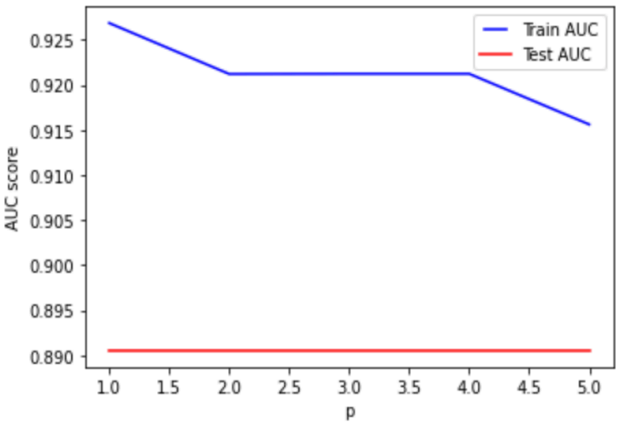
2.4.1 Tuning K Nearest Neighbors

For K nearest neighbor approach, the most important model parameter is the number of neighbors to use when making a prediction. Too high number of neighbors will lower model performance and significantly lengthen the run time while too low number of neighbors will lead to overfitting of model performance. To tune the number of neighbors, I calculated model AUC on both training and testing data and plot the versus number of neighbors. As we can see from the left chart, as the neighbors reach above 5, there is no change in both train and test AUC. And when number of neighbors=5, the test AUC is highest. Therefore, I select five neighbors to run final KNN model.

Another important parameter in KNN is the distance metrics to use. Where p in x axis represents different distance metrics. When p=1, this is equivalent to using Manhattan distance(l1), and Euclidean distance(l2) for p=2. For arbitrary p, minkowski distance (lp) is used. As we can see from the chart on the right, there is no change on test AUC with respect to different metrics. Thus, the model is indifferent on distance metrics.

Figure 17, KNN Model Tuning Cross Validation Performance





2.4.2 Final Model Results with K Nearest Neighbors

Table and figure below show final model performance with KNN, by applying synthetic minority over-sampling on model training dataset. The model evaluation is done by using 20% of the raw data for testing purposes. The model achieves high level of both precision and recall, with recall as high as 84%, indicating that among all fraud cases, the model can predict 84% of them as fraud, which is a high number. The 71% precision is also high, showing that among all predicted fraud cases, more than 71% of them are actual fraud. With an overall accuracy of 86%, we conclude that the model built with KNN using five neighbors can well predict fraud and differentiate it from normal cases.

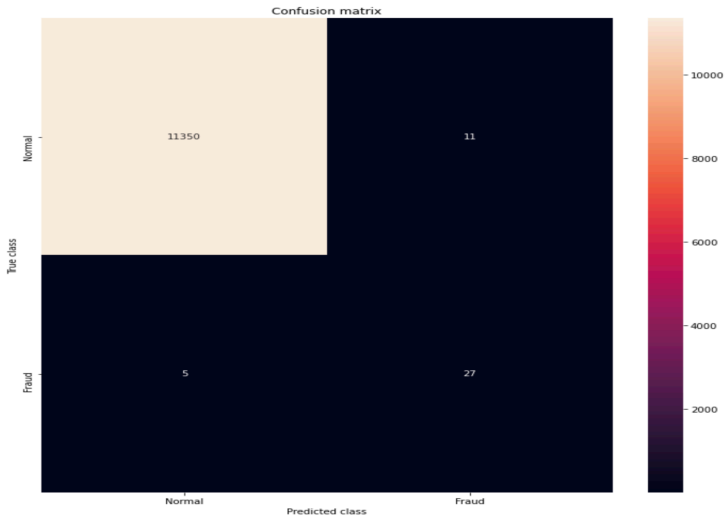
Table 15, KNN Performance Statistics with Mixture Sampling

```
count of fraud cases 45479
count of non-fraud cases 45479
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11361
           1       0.71      0.84      0.77        32

    accuracy                           1.00     11393
   macro avg       0.86      0.92      0.89     11393
weighted avg       1.00      1.00      1.00     11393
```

Figure 18, KNN Final Model with Mixture Sample


Confusion matrix
11350
11
5
27
Normal
Fraud
True class
Normal
Fraud
Predicted class
10000
8000
6000
4000
2000