Capstone Project – Credit Card Transactions Fraud Detection – Cost Benefit Analysis

Definition

Project Overview

Credit card fraud is any dishonest act or behavior to obtain information without the proper authorization of the account holder for financial gain. Among the different ways of committing fraud, skimming is the most common one. Skimming is a method used for duplicating information located on the magnetic stripe of the card.

This project aims to combine some of these common ways of detecting frauds in one single model and verify whether the combination of techniques help increase the final performance.

Problem Statement

The problem statement chosen for this project is to predict fraudulent credit card transactions with the help of machine learning models.

In this project, the data set contains credit card transactions of around 1,000 cardholders with a pool of 800 merchants from 1 Jan 2019 to 31 Dec 2020. It contains a total of 18,52,394 transactions, out of which 9,651 are fraudulent transactions. The data set is highly imbalanced, with the positive class (frauds) accounting for 0.52% of the total transactions. Now, since the data set is highly imbalanced, it needs to be handled before model building. The feature 'amt' represents the transaction amount. The feature 'is fraud' represents class labelling and takes the value 1 the transaction is a fraudulent transaction and 0, otherwise. This is a simulated data set taken from the Kaggle website and contains both legitimate and fraudulent transactions.

AIM

To build a model with the appropriate metrics and demonstrate its potential benefits by performing a cost-benefit analysis which can then be presented to the relevant business stakeholders.
To perform this analysis, you need to compare the costs incurred before and after the model is deployed. Earlier, the bank paid the entire transaction amount to the customer for every fraudulent transaction which accounted for a heavy loss to the bank.

Cost-Benefit Analysis

Let us take a look at what you need to do in order to perform the cost-benefit analysis step by step.

Part I: Analyze the dataset and find the following figures:

- Average number of transactions per month
- Average number of fraudulent transactions per month
- Average amount per fraudulent transaction

Part II: Compare the cost incurred per month by the bank before and after the model deployment:

- Cost incurred per month before the model was deployed = Average amount per fraudulent transaction * Average number of fraudulent transactions per month
- Cost incurred per month after the model is built and deployed: <Use the test metric from the model evaluation part and the calculations performed in Part I to compute the values given below>

Let TF be the average number of transactions per month detected as fraudulent by the model and let the cost of providing customer executive support per fraudulent transaction detected by the model = $1.5

- Total cost of providing customer support per month for fraudulent transactions detected by the model = 1.5 * TF.

Let FN be the average number of transactions per month that are fraudulent but not detected by the model

- Cost incurred due to these fraudulent transactions left undetected by the model = Average amount per fraudulent transaction * FN
- Therefore, the cost incurred per month after the model is built and deployed = 1.5*TF + Average amount per fraudulent transaction * FN
- Final savings = Cost incurred before - Cost incurred after.

Note that you're not including the model deployment cost since it will only be a one-time expenditure and you're trying to gauge the long-term benefits of putting this model into practice.

Analysis

Data Exploration

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days in which there were 492 frauds out of 284,807 transactions. The dataset is highly imbalanced and the positive class (frauds) account for 0.172% of all transactions, as seen on Figure 1.

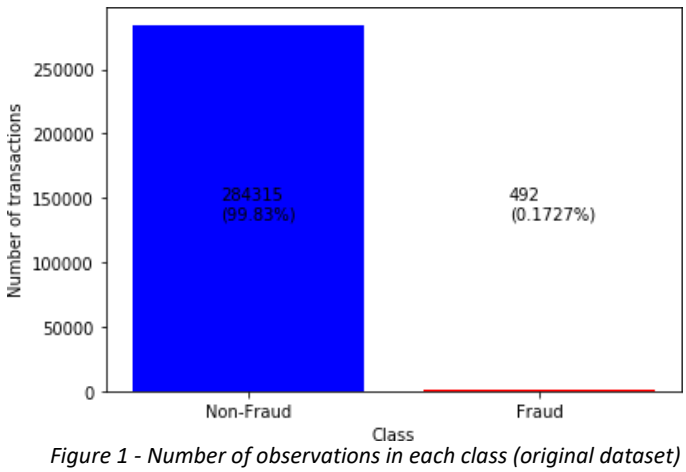


Figure 1 - Number of observations in each class (original dataset)

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, it is not possible to provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' represents the class labelling, it takes value 1 in case of a fraud and 0 otherwise. In Figure 2, it is possible to see how the dataset looks like.

```
#Display the first 5 rows of the dataset
table.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

Figure 2 - Features' overview

Figure 3 shows the basic statistics for the data.

```
#generate descriptive statistics of a data
table.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	..
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15	..
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	..
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	..
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	..
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	..
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	..
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	..

8 rows × 31 columns

Figure 3 - Basic statistics of the data

Exploratory Visualization

The interquartile range method found 31904 outliers, which represents 11.2% of the observations, as seen in Figure 4. Removing them from the dataset would be a bad idea due to the loss of a large amount of information for the machine learning models.

```
Number of outliers below the lower bound:  0  (0.0%)
Number of outliers above the upper bound:  31904  (11.2%)
```

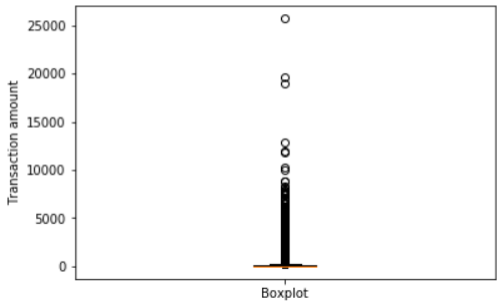


Figure 4 - Box-plot of the whole dataset

Figure 5 shows that fraudulent transactions are highly concentrated at smaller values when compared to non-fraudulent transactions.

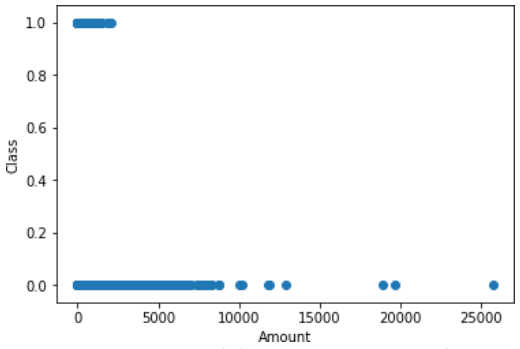


Figure 5 - Fraudulent transactions analysis

We can see on the heatmap in Figure 6 that all features have very low correlation coefficients among each other, and especially low correlation with the 'Class' feature. This was already expected since the data was processed using PCA.

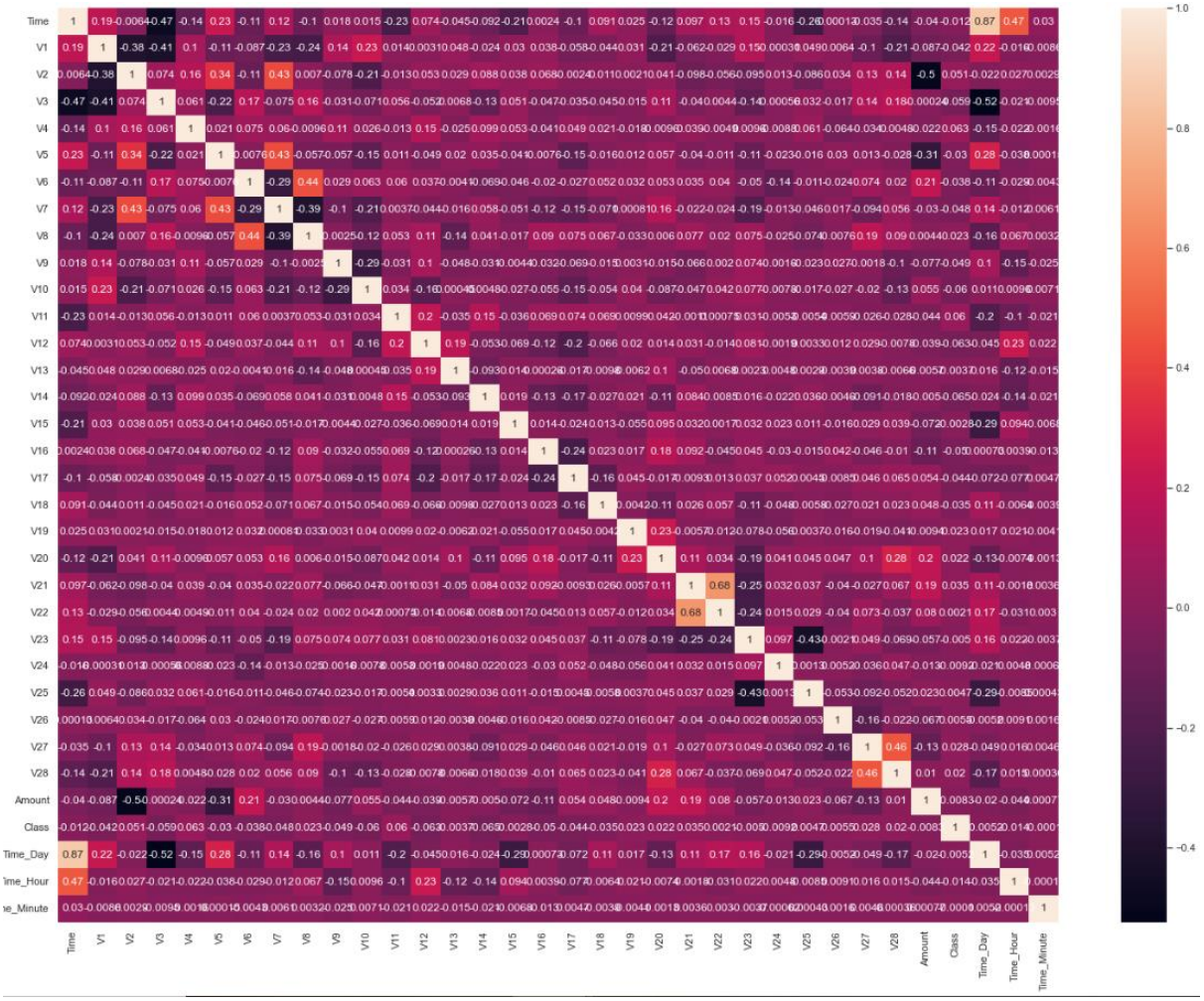


Figure 6 - Heatmap for the dataset.

Algorithms and Techniques

The first issue to be solved was the high imbalance of classes present in the dataset as mentioned before. But before balancing the classes, we need to split the observations into a training set and a testing set. This is extremely important! We can only balance the classes after we set some observations aside to be used as a test set. Otherwise, the models might use part of the test data during the training, which will lead to overfitting. I chose to use 30% of the dataset as a test set and I made a split keeping the original rate of frauds to non-frauds observations in each set.

After that, I chose three different resampling techniques to balance the training dataset: random under-sampling, random over-sampling and synthetic Minority Over-sampling. Figure 7 depicts how under sampling and oversampling works and it is very intuitive to understand. SMOTE is like oversampling but instead of copying the same original points randomly, the algorithm creates new points close to the original ones.

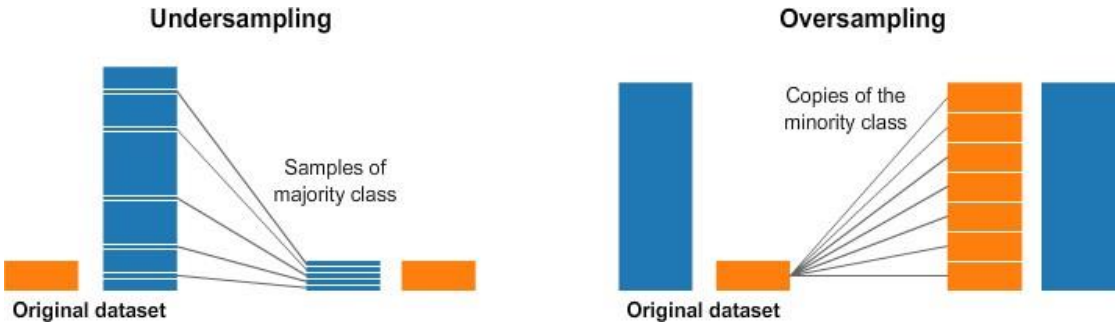


Figure 7 - Resampling techniques

A quick explanation of how SMOTE works: it consists of synthesizing elements for the minority class using the existing ones. It randomly chooses a point from the minority class and computes the k-nearest neighbors (default = 5) for this point. The synthetic points are added between the chosen point and its neighbors by choosing a factor between 0 and 1 to multiply the distance. This process can be seen below in Figure 8.

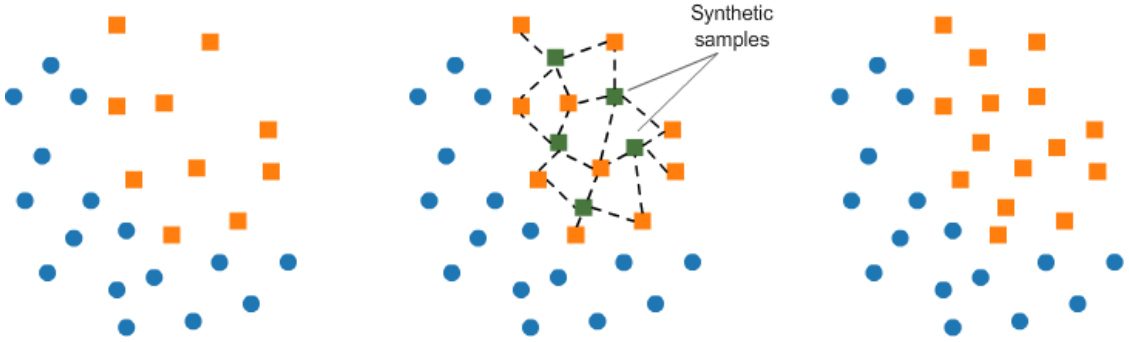


Figure 8 - How SMOTE works

After the data were balanced, the next step was to create the simple/common models and the ensemble model consisting of a soft voting classifier with the same weights to every simple model previously created. It was already proven in many competitions that classifiers that combine predictions from more than one single classifier can achieve a better performance.

The common models chosen were logistic regression, Naïve Bayes K-Nearest Neighbors, Decision Tree and Random Forest.

Benchmark

The benchmark to be used is the performance of best model among the simple models used. Usually, most people just try to find one unique model that has the best prediction and stick to this model. The objective is to get a better performance using an ensemble model created using all the simple models together.

Methodology

Data Preprocessing

The credit card data provided is 100% accurate and no bad entries are present in the dataset. Therefore, there is no need to worry about mislabeled observations or bad entries. The only preprocessing step needed was to apply feature scaling due to the large range of two features in comparison to the other 28 features.

Implementation

Each classifier should be appended to a list inside a tuple with its name as the first element and the classifier definition as the second element. This list will serve as an input for a function that plots the ROC curve, calculates the AUC and shows the confusion matrix for each classifier. The ensemble model must stay separated from this list and should be assessed separately, calling the same function mentioned before.

```
classifiers = []

classifiers.append(('Logistic Regression', LogisticRegression(random_state=42)))
classifiers.append(('Naive Bayes', GaussianNB()))
classifiers.append(('KNN', KNeighborsClassifier()))
#classifiers.append(('SVM', SVC(random_state=42, probability=True)))
classifiers.append(('Decision Tree', DecisionTreeClassifier(random_state=42)))
classifiers.append(('Random Forest', RandomForestClassifier(random_state=42)))
classifiers.append(('XGBoost', XGBClassifier(random_state=42)))

#Ensemble classifier - All classifiers have the same weight
eclf = VotingClassifier(estimators=classifiers, voting='soft', weights=np.ones(len(classifiers)))
```

The function plot_confusion_matrix was created to show the result from the confusion matrix provided by the scikit-learn package in an easier way to understand, like the example given below in Figure 9.

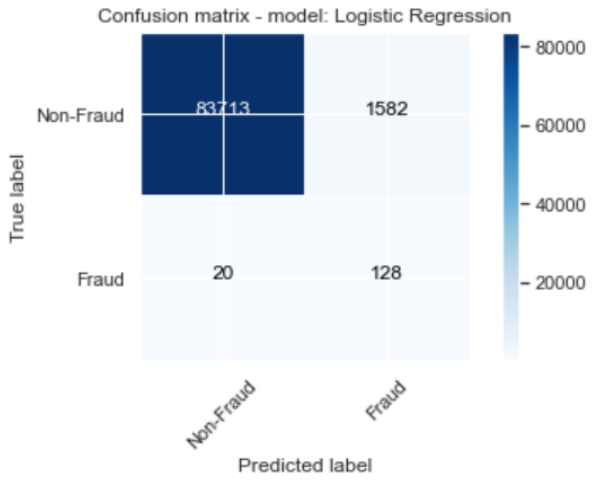


Figure 9 - Confusion matrix example

The ensemble model used here is a soft voting classifier which puts the same weight to all individual classifiers' class probabilities. It was already proven in many competitions that classifiers that combine predictions from more than one single classifier can achieve a better performance.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    #if normalize:
    #    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    #    print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

The function plot_CM_and_ROC_curve is the main function. It plots the ROC curve and calculates the area under the curve for one classifier. It also calls the plot_confusion_matrix function to create the detailed image for the confusion matrix, and calculates the accuracy, precision and recall metrics.

```
from sklearn import svm
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from scipy import interp

def plot_CM_and_ROC_curve(classifier, X_train, y_train, X_test, y_test):
    '''Plots the ROC curve and the confusion matrix, and calculates AUC, recall and precision.'''
```

```
    name = classifier[0]
    classifier = classifier[1]

    mean_fpr = np.linspace(0, 1, 100)
    class_names = ['Non-Fraud', 'Fraud']
    confusion_matrix_total = [[0, 0], [0, 0]]

    #Obtain probabilities for each class
    probas_ = classifier.fit(X_train, y_train).predict_proba(X_test)

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=1, alpha=1, color='b', label='ROC (AUC = %0.7f)' % (roc_auc))
    plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
             label='Chance', alpha=.8)
    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve - model: ' + name)
    plt.legend(loc="lower right")
    plt.show()

    #Store the confusion matrix result to plot a table later
    y_pred=classifier.predict(X_test)
    cnf_matrix = confusion_matrix(y_test, y_pred)
    confusion_matrix_total += cnf_matrix

    #Print precision and recall
    tn, fp = confusion_matrix_total.tolist()[0]
    fn, tp = confusion_matrix_total.tolist()[1]
    accuracy = (tp+tn)/(tp+tn+fp+fn)
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    print('Accuracy = {:2.2f}%'.format(accuracy*100))
    print('Precision = {:2.2f}%'.format(precision*100))
    print('Recall = {:2.2f}%'.format(recall*100))

    # Plot confusion matrix
    plt.figure()
    plot_confusion_matrix(confusion_matrix_total, classes=class_names, title='Confusion matrix -
model: ' + name)
    plt.show()
```

Refinement

This project is using commonly used models with default configurations to assess whether an ensemble model that combines all these common models is capable of outperform the best result among the common models. Therefore, there isn't a refinement step.

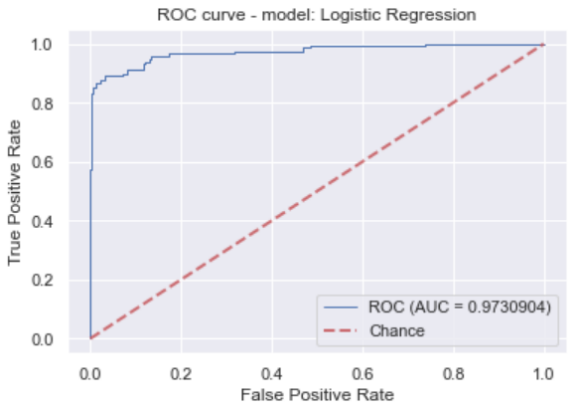
Results

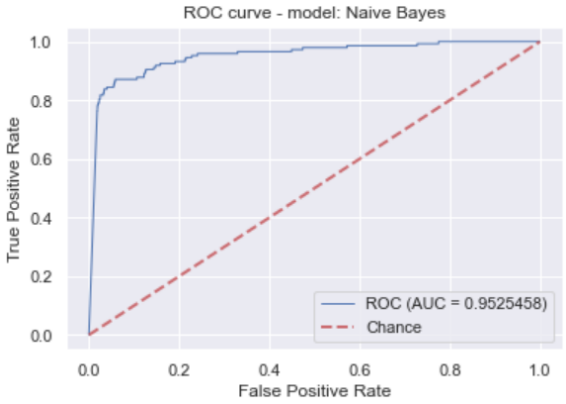
Model Evaluation and Validation

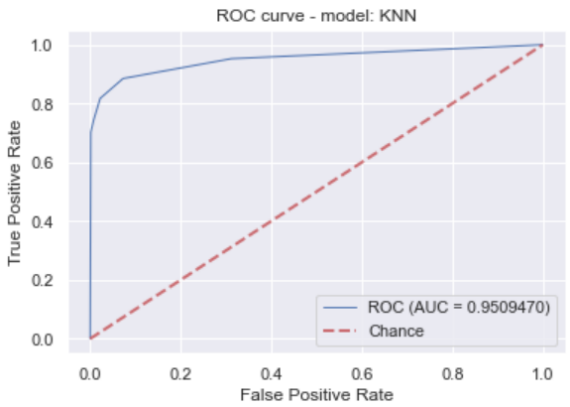
Results using random undersampling

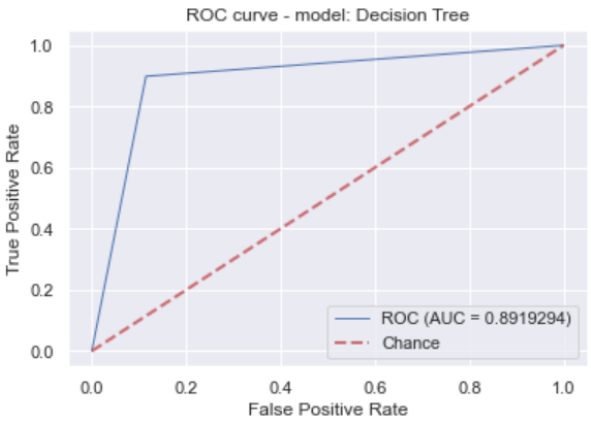
Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9730	7.49%	86.49%	98.13%
Naive Bayes	0.9525	4.07%	83.78%	96.55%
KNN	0.9509	6.01%	81.76%	97.76%
Decision Tree	0.8919	1.34%	89.86%	88.52%
Random Forest	0.9718	7.79%	86.49%	98.20%
XGBoost	0.9758	3.92%	91.22%	96.11%
Ensemble	**0.9695**	**7.83%**	**85.14%**	**98.24%**

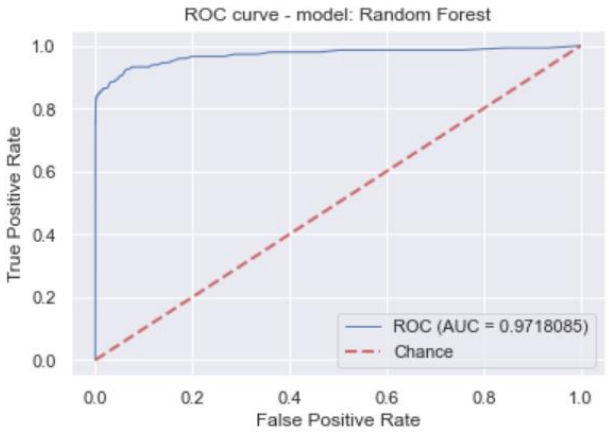
Table 1 - Performance of classifiers using random undersampling

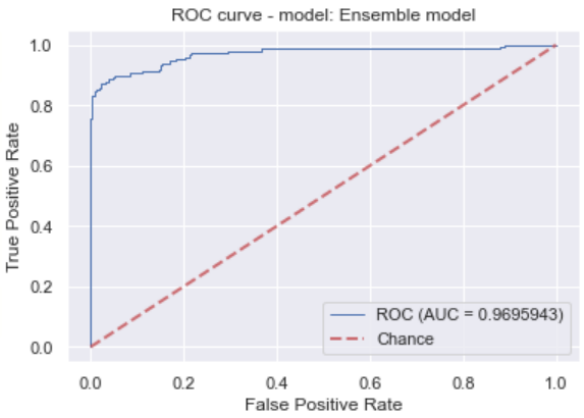


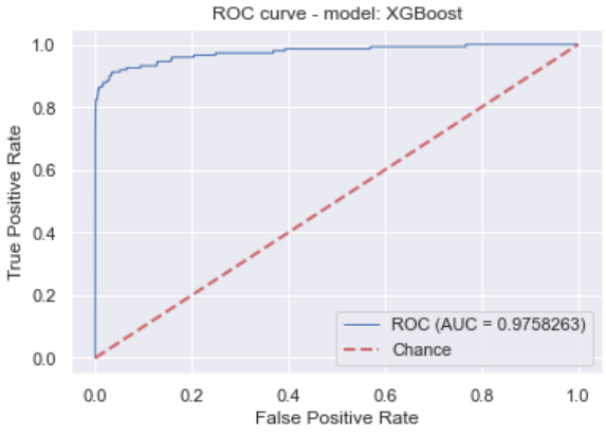






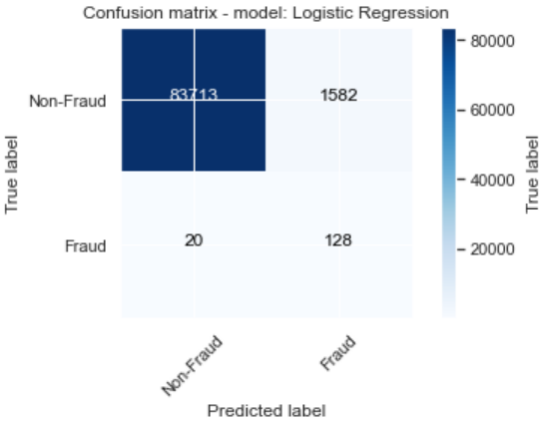


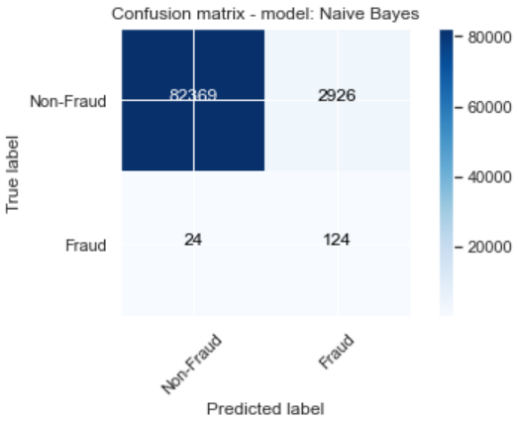


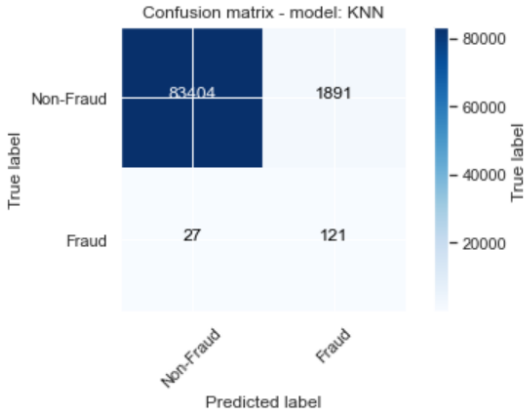


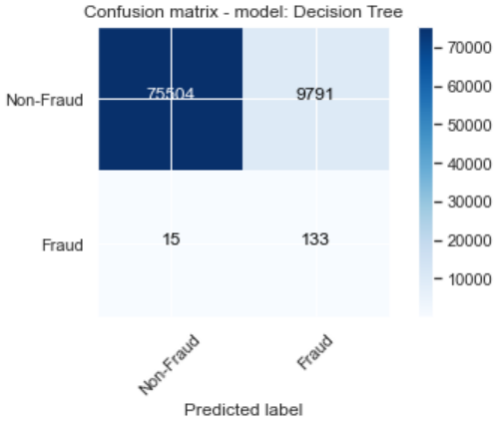
Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	83713	20	128	1582
Naive Bayes	82369	24	124	2926
KNN	83404	27	121	1891
Decision Tree	75504	15	133	9791
Random Forest	83780	20	128	1515
XGBoost	81985	13	135	3310
Ensemble	83812	22	126	1483

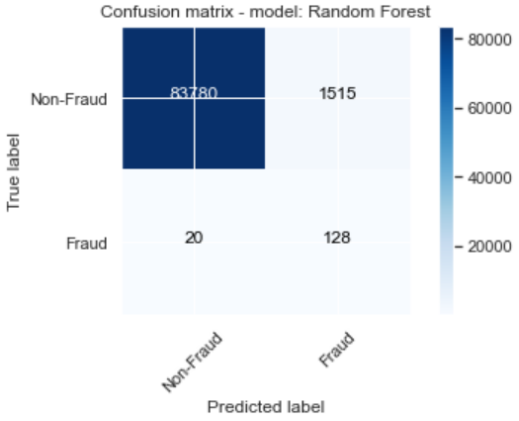
Table 2 – Confusion matrix of classifiers using random undersampling

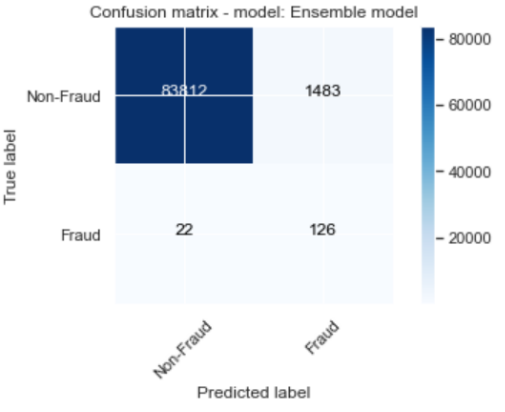

Confusion matrix - model: Logistic Regression
Non-Fraud
Fraud
True label
83713
1582
20
128
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

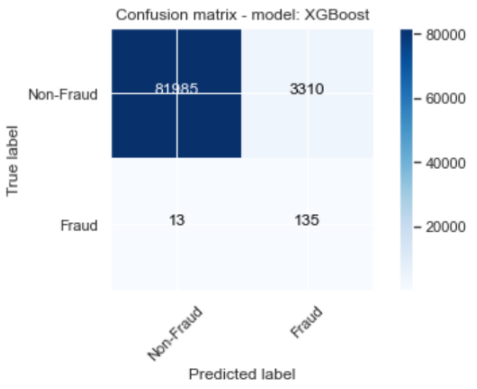

Confusion matrix - model: Naive Bayes
Non-Fraud
Fraud
True label
82369
2926
24
124
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000


Confusion matrix - model: KNN
Non-Fraud
Fraud
True label
83404
1891
27
121
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000


Confusion matrix - model: Decision Tree
Non-Fraud
Fraud
True label
75504
9791
15
133
Non-Fraud
Fraud
Predicted label
70000
60000
50000
40000
30000
20000
10000


Confusion matrix - model: Random Forest
Non-Fraud
Fraud
True label
83780
1515
20
128
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

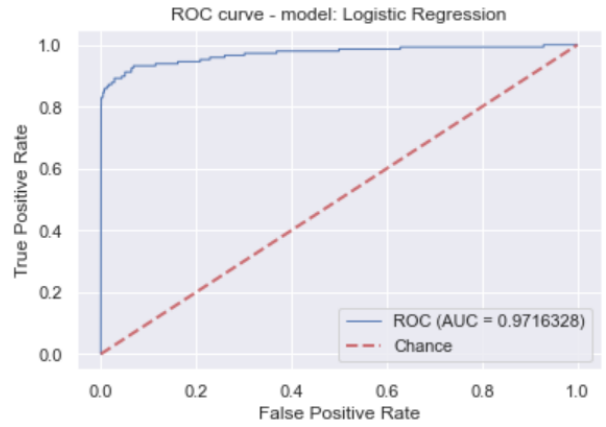

Confusion matrix - model: Ensemble model
Non-Fraud
Fraud
True label
83812
1483
22
126
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

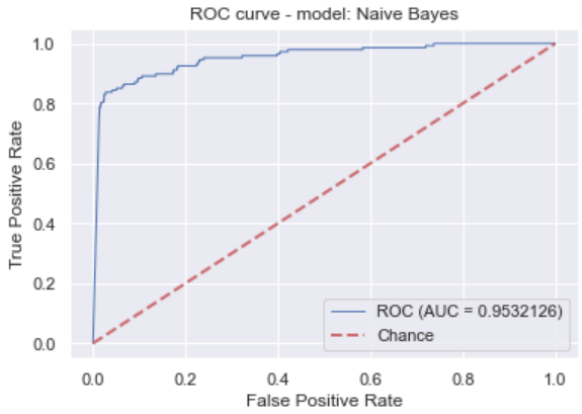

Confusion matrix - model: XGBoost
Non-Fraud
Fraud
True label
81985
3310
13
135
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

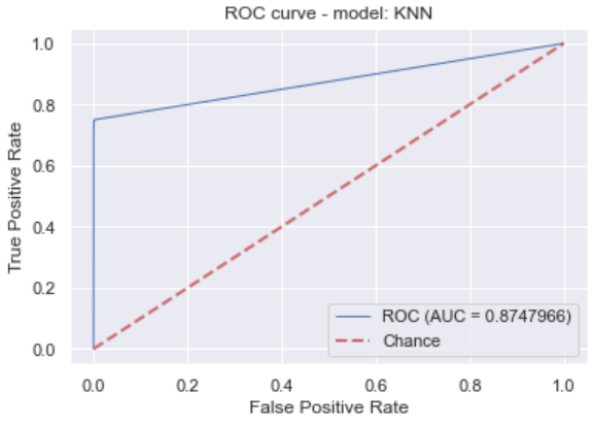
Results using random oversampling

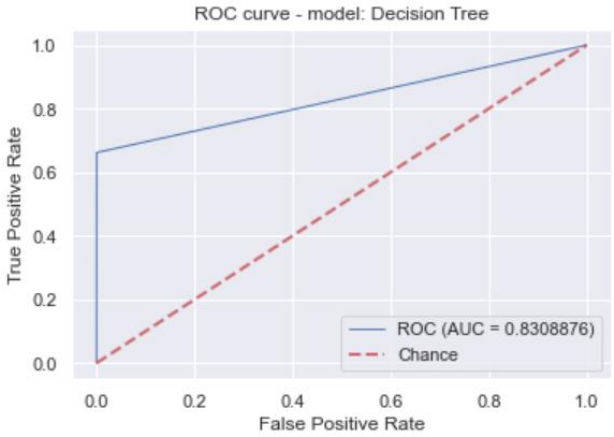
Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9716	6.47%	87.16%	97.80%
Naive Bayes	0.9532	5.32%	83.11%	97.41%
KNN	0.8747	69.48%	72.30%	99.90%
Decision Tree	0.8308	74.81%	66.22%	99.90%
Random Forest	0.9338	96.46%	73.65%	99.95%
XGBoost	0.9711	92.80%	78.38%	99.95%
Ensemble	0.9690	85.19%	77.70%	99.94%

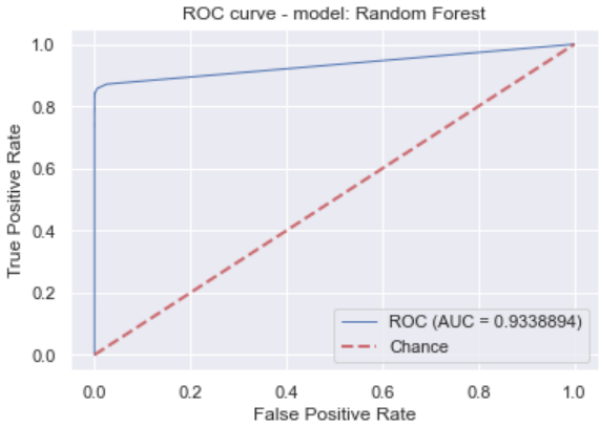
Table 3 - Performance of classifiers using random oversampling

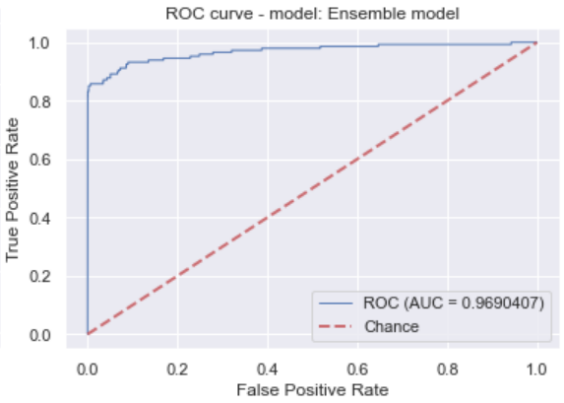


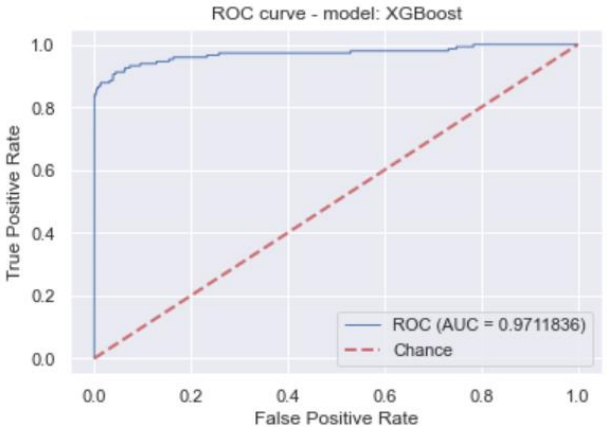






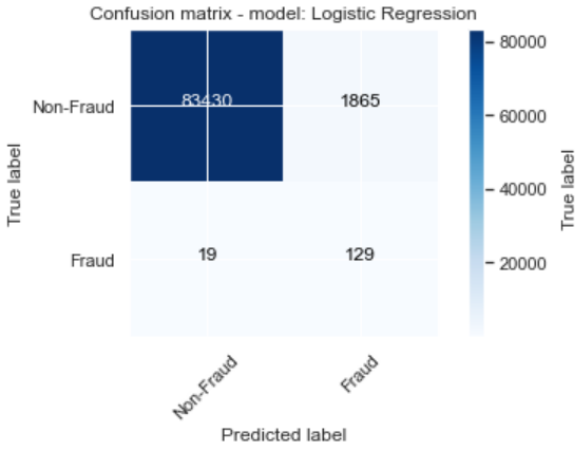


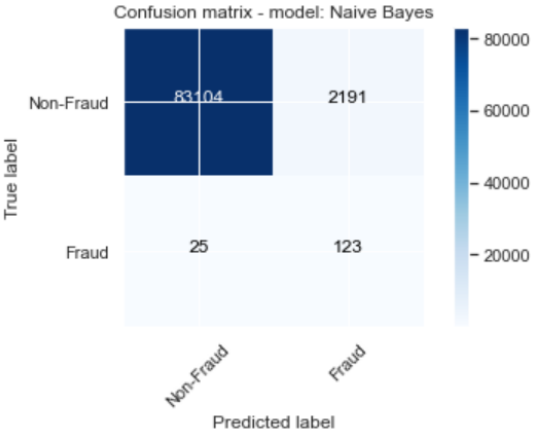


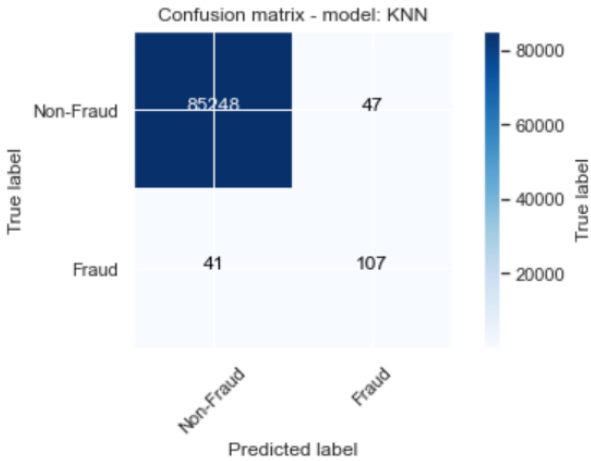


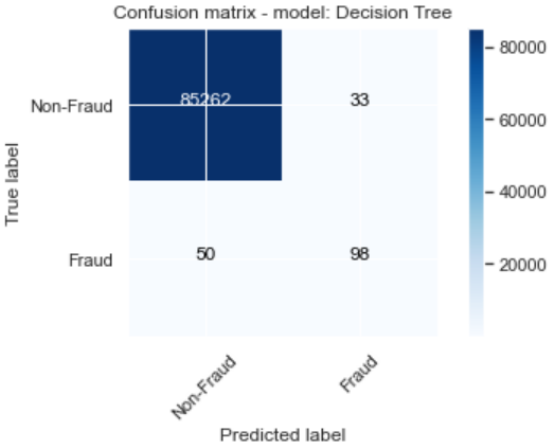
Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	83430	19	129	1865
Naive Bayes	83104	25	123	2191
KNN	85248	41	107	47
Decision Tree	85262	50	96	33
Random Forest	85291	39	109	4
XGBoost	85286	32	116	9
Ensemble	85275	33	115	20

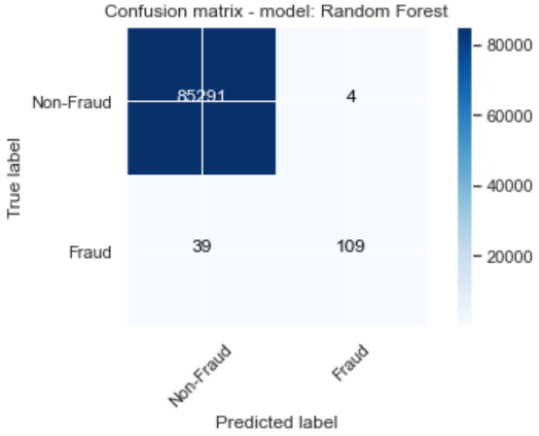
Table 4 – Confusion matrix of classifiers using random oversampling

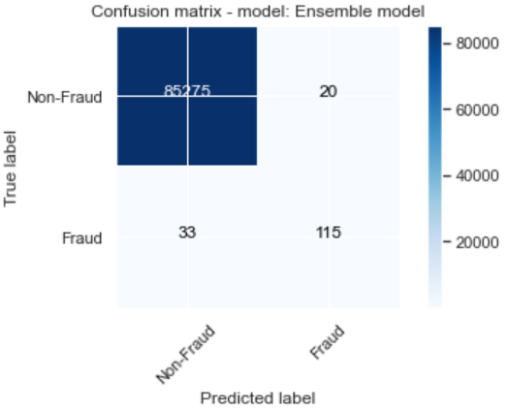

Confusion matrix - model: Logistic Regression
Non-Fraud
Fraud
True label
83430
1865
19
129
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

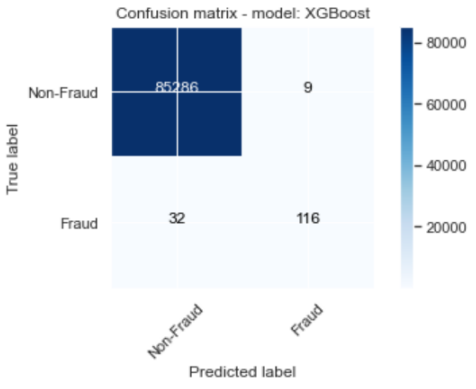

Confusion matrix - model: Naive Bayes
Non-Fraud
Fraud
True label
83104
2191
25
123
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000


Confusion matrix - model: KNN
Non-Fraud
Fraud
True label
85248
47
41
107
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000


Confusion matrix - model: Decision Tree
Non-Fraud
Fraud
True label
85262
33
50
98
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000


Confusion matrix - model: Random Forest
Non-Fraud
Fraud
True label
85291
4
39
109
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

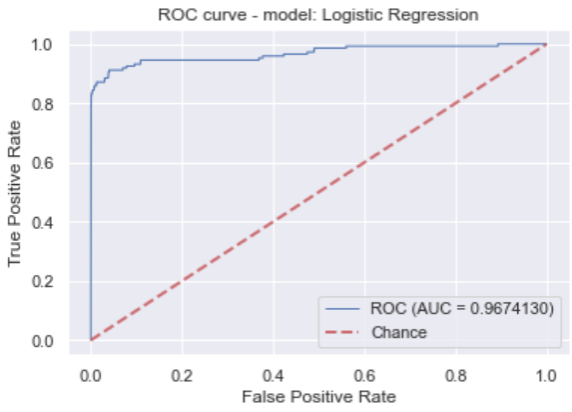

Confusion matrix - model: Ensemble model
Non-Fraud
Fraud
True label
85275
20
33
115
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

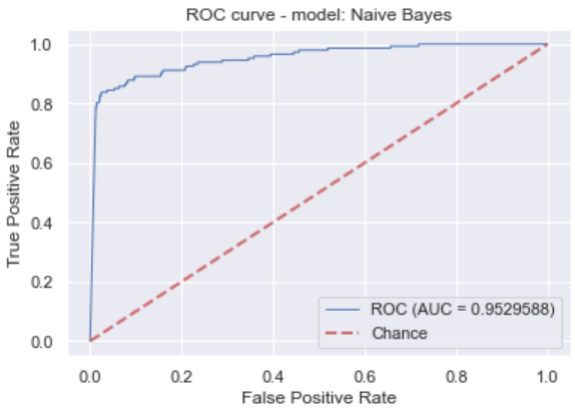

Confusion matrix - model: XGBoost
Non-Fraud
Fraud
True label
85286
9
32
116
Non-Fraud
Fraud
Predicted label
80000
60000
40000
20000

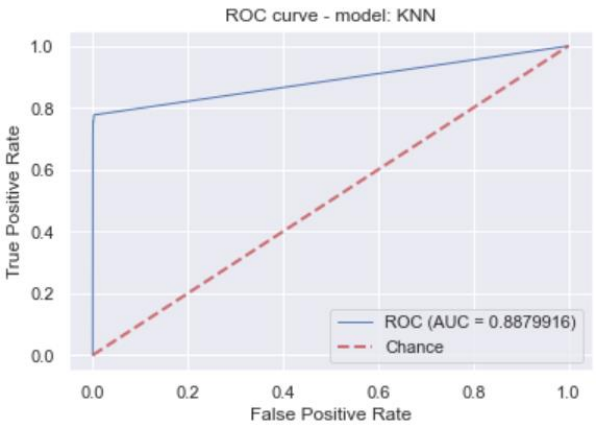
Results using SMOTE

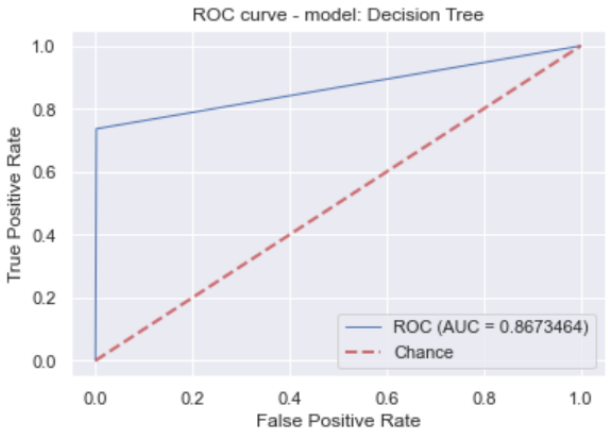
Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9674	15.99%	85.81%	99.19%
Naive Bayes	0.9529	5.93%	82.43%	97.70%
KNN	0.8879	46.86%	75.68%	99.81%
Decision Tree	0.8673	41.60%	73.65%	99.78%
Random Forest	0.9608	84.03%	79.05%	99.94%
XGBoost	0.9718	81.76%	84.62%	99.94%
Ensemble	0.9704	73.17%	81.08%	99.92%

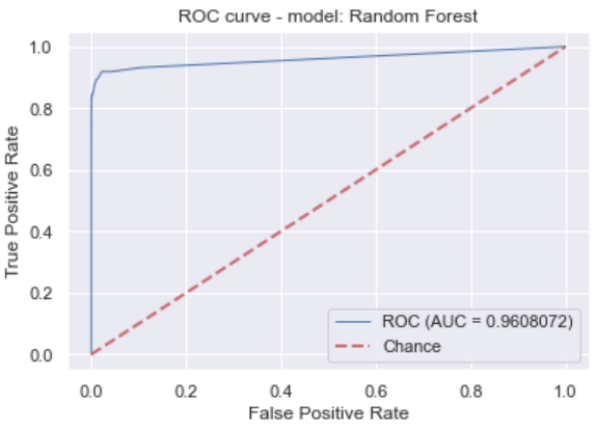
Table 5 - Performance of classifiers using SMOTE

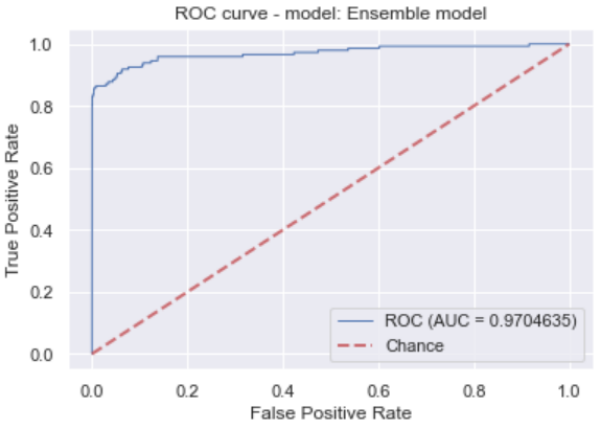


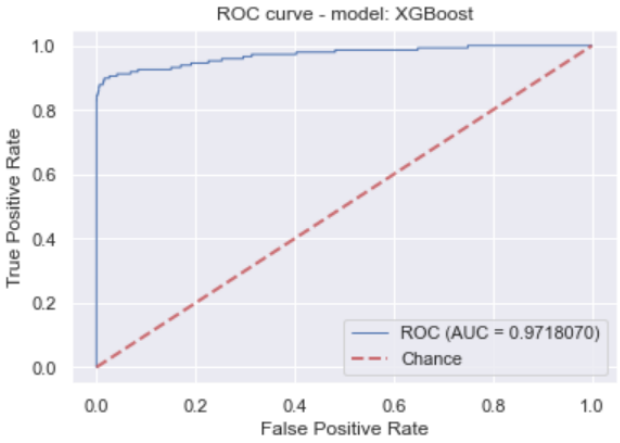






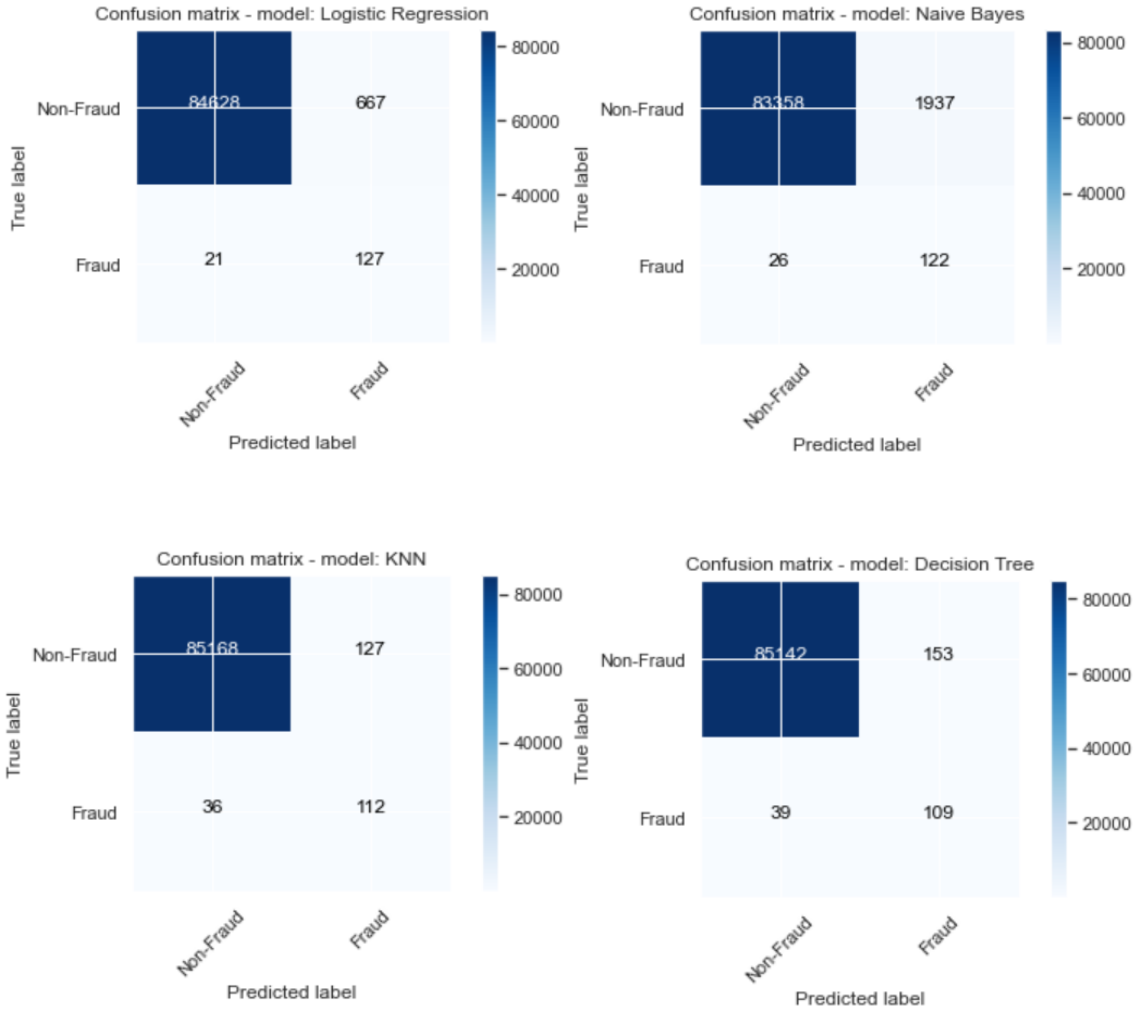

ROC curve - model: Random Forest
True Positive Rate
False Positive Rate
ROC (AUC = 0.9608072)
Chance
0.0
0.2
0.4
0.6
0.8
1.0

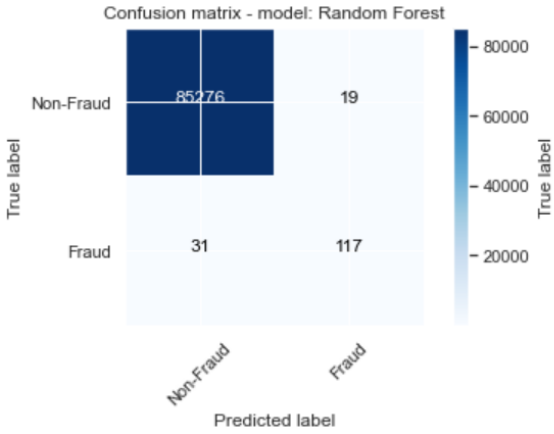

ROC curve - model: Ensemble model
True Positive Rate
False Positive Rate
ROC (AUC = 0.9704635)
Chance
0.0
0.2
0.4
0.6
0.8
1.0

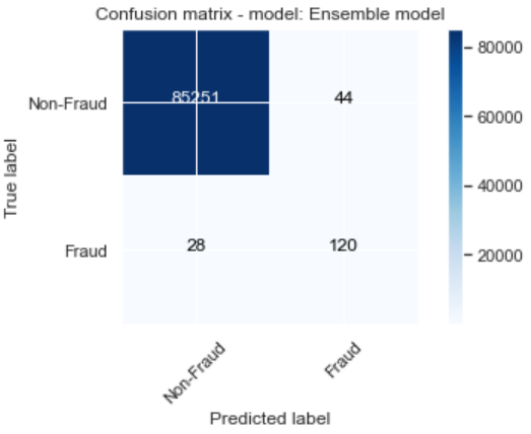

ROC curve - model: XGBoost
True Positive Rate
False Positive Rate
ROC (AUC = 0.9718070)
Chance
0.0
0.2
0.4
0.6
0.8
1.0

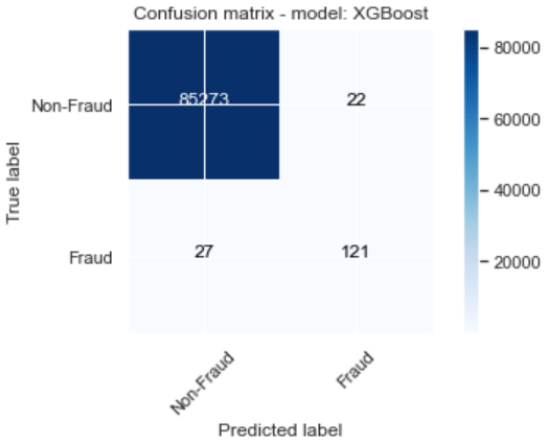
Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	84628	21	127	667
Naive Bayes	83358	26	122	1937
KNN	85168	36	112	127
Decision Tree	85142	39	109	153
Random Forest	85276	31	117	19
XGBoost	85273	27	121	22
Ensemble	85251	28	120	44

Table 6 – Confusion matrix of classifiers using SMOTE




Confusion matrix - model: Random Forest
Non-Fraud
Fraud
True label
85276
19
31
117
80000
60000
40000
20000
Non-Fraud
Fraud
Predicted label


Confusion matrix - model: Ensemble model
Non-Fraud
Fraud
True label
85251
44
28
120
80000
60000
40000
20000
Non-Fraud
Fraud
Predicted label


Confusion matrix - model: XGBoost
Non-Fraud
Fraud
True label
85273
22
27
121
80000
60000
40000
20000
Non-Fraud
Fraud
Predicted label

Justification

By using random undersampling, the best model was a simple logistic regression with AUC = 0.9730, recall = 86.49% and precision = 7.49%. The number of false negatives, the worst possible classification regarding frauds, for the logistic regression was 20 out of 148 cases present in the test set. However, we can notice that the ensemble model (a soft voting combination of all simple models with the same weights) was the second best in this case with AUC = 0.9695 and 22 false negative cases, only 2 more than the logistic regression. It is also possible to notice that in all models the number of false positives was high. The best result was with random forest (1515 false positive cases). It's good to remember that when we use undersampling method, a lot of information is lost due to discarding part of the observations. Then, it was already expected a not very good result.

With the random oversampling method, the logistic regression could get a high AUC (0.9716), precision (6.47%) and high recall (87.16%). Looking at the confusion matrix we can see the cases for false negative and false positive as 33 and 20, respectively. The logistic regression was the model with the lowest number of false negatives (only 19) and the random forest was the model with the lowest number of false positives (an astonishing result of only 4). So, if we try to tweak the weights for each model in the ensemble model, we can achieve an even better result.

The analysis for SMOTE method is very similar to the random oversampling method. Again, we can see that the ensemble model could get the best parts of all simple models, resulting in a good performance. Once again, tweaking the weights for each simple model might be a way to enhance the result.

Conclusion

Free-Form Visualization

An interesting result is that in all cases of resampling, the highest AUC score among the simple models was using logistic regression. In my opinion, binary classification problems should have at least this model as a benchmark when the objective is to maximize the AUC.

Reflection

The initial challenge was the high imbalance present in the dataset. There are many different algorithms to resample data in different ways. So, I decided to evaluate the three most common methods I could find in classification problems: random undersampling, random oversampling and SMOTE.

Regarding the classifiers chosen, I opted for commonly used ones that have the option to return the class probability instead of the class prediction on scikit-learn package because it is a parameter used to calculate the ROC curve.

The ensemble classifier proved to be useful when random oversampling or SMOTE methods are used. It had a great performance in all the metrics, which could not be obtained using only one of the common classifiers. For the random undersampling, the result was not so good due to the loss of information that occurs when we discard most of the observations from the majority class.

It is important to remember that the number of false negative and false positive cases can change by tweaking the weights for each common classifier when using an ensemble classifier. So, it is up to the credit card institution to decide how to tweak the weights in order to avoid more frauds or to avoid bothering its clients with false alarms.

Improvement

The idea of using an ensemble classifier that combines all the results equally weighted from commonly used models proved to further enhance the performance. However, this project was made using default argument values for all classifiers. An obvious improvement to this work could be optimizing each of the simple models by finding the best parameters and checking whether the ensemble model can outperform them. Another great technique worth of trying is using the cross-validation on each model.

Cost Benefit Analysis

Data Understanding

This is a simulated data set taken from the Kaggle website and contains both legitimate and fraudulent transactions. This is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants. This was generated using Sparkov Data Generation | Github tool created by Brandon Harris. This simulation was run for the duration - 1 Jan 2019 to 31 Dec 2020. The files were combined and converted into a standard format.

Project Pipeline

The project pipeline can be briefly summarised in the following steps:

1. Understanding Data: In this step, you need to load the data and understand the features present in it. This will help you choose the features that you need for your final model.
2. Exploratory data analytics (EDA): Normally, in this step, you need to perform univariate and bivariate analyses of the data, followed by feature transformations, if necessary. You can also check whether or not there is any skewness in the data and try to mitigate it, as skewed data can cause problems during the model-building phase.
3. Train/Test Data Splitting: In this step, you need to split the data set into training data and testing data in order to check the performance of your models with unseen data. You can use the stratified k-fold cross-validation method at this stage. For this, you need to choose an appropriate k value such that the minority class is correctly represented in the test folds.
4. Model Building or Hyperparameter Tuning: This is the final step, at which you can try different models and fine-tune their hyperparameters until you get the desired level of performance out of the model on the given data set. Ensure that you start with a baseline linear model before going towards ensembles. You should check if you can get a better performance out of the model by using various sampling techniques.
5. Model Evaluation: Evaluate the performance of the models using appropriate evaluation metrics. Note that since the data is imbalanced, it is important to identify which transactions are fraudulent transactions more accurately than identifying non-fraudulent transactions. Choose an appropriate evaluation metric that reflects this business goal.
6. Business Impact: After the model has been built and evaluated with the appropriate metrics, you need to demonstrate its potential benefits by performing a cost-benefit analysis which can then be presented to the relevant business stakeholders.

Cost-Benefit Analysis Procedure

Let us take a look at what you need to do in order to perform the cost-benefit analysis step by step.
Part I: Analyse the dataset and find the following figures:

- Average number of transactions per month
- Average number of fraudulent transactions per month
- Average amount per fraudulent transaction

Part II: Compare the cost incurred per month by the bank before and after the model deployment:

- Cost incurred per month before the model was deployed = Average amount per fraudulent transaction * Average number of fraudulent transactions per month
- Cost incurred per month after the model is built and deployed: <Use the test metric from the model evaluation part and the calculations performed in Part I to compute the values given below>

Let TF be the average number of transactions per month detected as fraudulent by the model and let the cost of providing customer executive support per fraudulent transaction detected by the model = $1.5

- Total cost of providing customer support per month for fraudulent transactions detected by the model = 1.5 * TF.

Let FN be the average number of transactions per month that are fraudulent but not detected by the model

- Cost incurred due to these fraudulent transactions left undetected by the model = Average amount per fraudulent transaction * FN
- Therefore, the cost incurred per month after the model is built and deployed = 1.5*TF + Average amount per fraudulent transaction * FN
- Final savings = Cost incurred before - Cost incurred after.

To perform this analysis, you need to compare the costs incurred before and after the model is deployed. Earlier, the bank paid the entire transaction amount to the customer for every fraudulent transaction which accounted for a heavy loss to the bank.