Machine Learning Engineer Nanodegree

Capstone Project – Credit Card Fraud Prediction

Alexandre Brandão Veras Daltro

March 6th, 2019

I. Definition

Project Overview

Credit card fraud is a type of crime that is on the rise (link) and this is a risk to everyone who uses this payment method. I surely know how annoying is to find that your credit card was frauded and used by criminals to buy anything they want with your hard-earned money. It already happened to me 3 times even though I am not an easy target for scams. I have always taken all the recommended measures for a regular citizen to avoid frauds and yet it wasn't enough to protect me against this crime.

Despite several papers on this theme were written in the last years, banks still struggle to correctly classify normal and fraudulent transactions because swindlers are always coming up with new strategies to disguise fake transactions. Delamaire, Abdou & Pointon (2009) list ways of detecting fraudulent transactions like pair-wise matching, decision trees, clustering techniques, neural networks, and genetic algorithms.

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. For this study I used a dataset containing transactions made by credit cards in September 2013 by European cardholders.

Problem Statement

The problem chosen for this project is to predict fraudulent credit card transactions by using machine learning models. The models are going to be trained using supervised learning techniques. A dataset containing thousands of individual transactions and their respective labels was obtained from Kaggle website (link).

Metrics

Usually accuracy is the first metric that comes to mind when someone is assessing a model performance. However, we must be careful. The data in this case is highly unbalanced, so accuracy is not a good metric at all. If we created a model that always classifies a transaction as non-fraudulent, we would have an astonishing accuracy of 99.83%! So, what is the solution? We should use other metrics to consider a model as good or bad.

The metrics to be used will be the Area Under the ROC curve (also called AUC), and the recall and precision scores obtained from the confusion matrix. With these 3 metrics the we can tell whether the model performance is good or poor.

The ROC curve is a plot with the true positive rate on the y-axis and false positive rate on the x-axis. The true positive rate answers the question "When the actual classification is positive, how often does the classifier predict positive?" and the false positive rate answers the question "When the actual classification is negative, how often does the classifier incorrectly predict positive?"

The AUC shows how good the classifier is in separating the classes. The closer to 1, the better is the classifier.

Precision answers the question "what proportion of positive identifications was actually correct?" and recall answers "what proportion of actual positives was identified correctly?" These metrics can give us an idea of false positive and false negative results returned by the model, which are undesirable misclassifications.

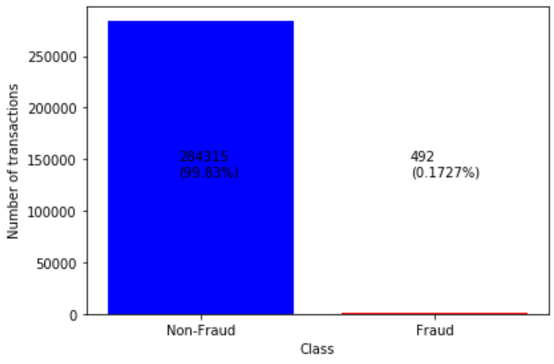
False negatives are the most dangerous misclassification for credit cards frauds because it is possible to lose a lot of money in one single fraudulent transaction that was predicted as non-fraudulent. On the other hand, if the model has a high rate of false positives, the institution will incur in high costs assessing all the cases and will eventually block clients’ credit cards providing a bad consumer experience.

II. Analysis

Data Exploration

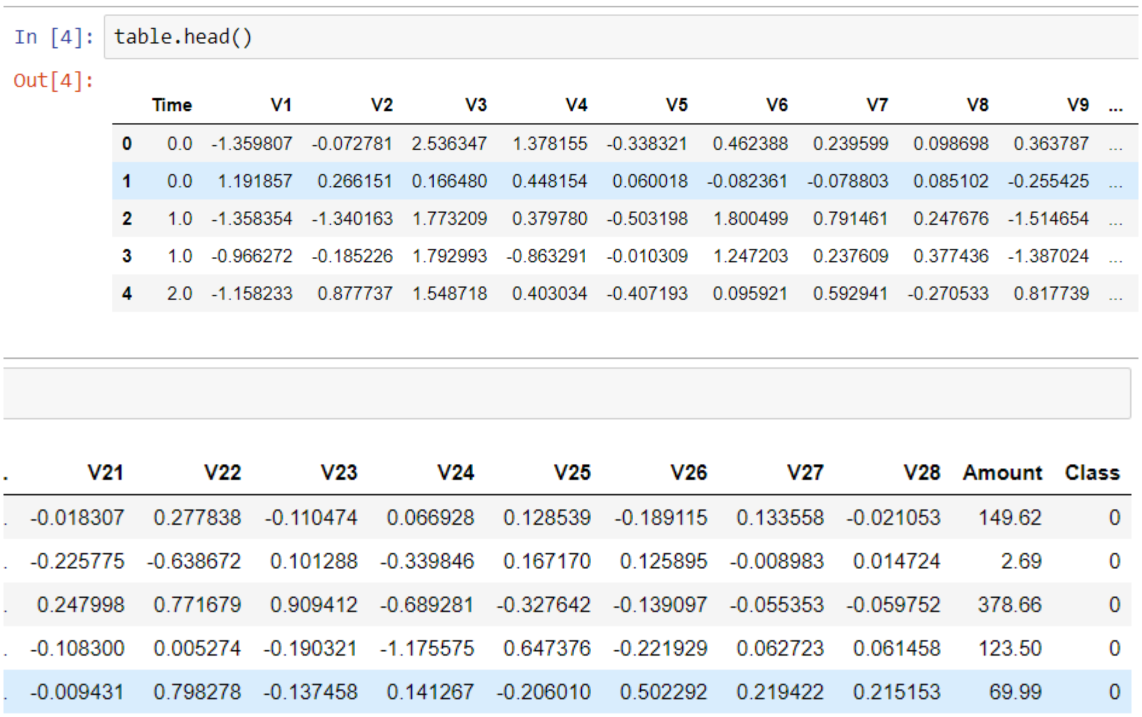
The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days in which there were 492 frauds out of 284,807 transactions. The dataset is highly imbalanced and the positive class (frauds) account for 0.172%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

Figure 2 - Features' overview



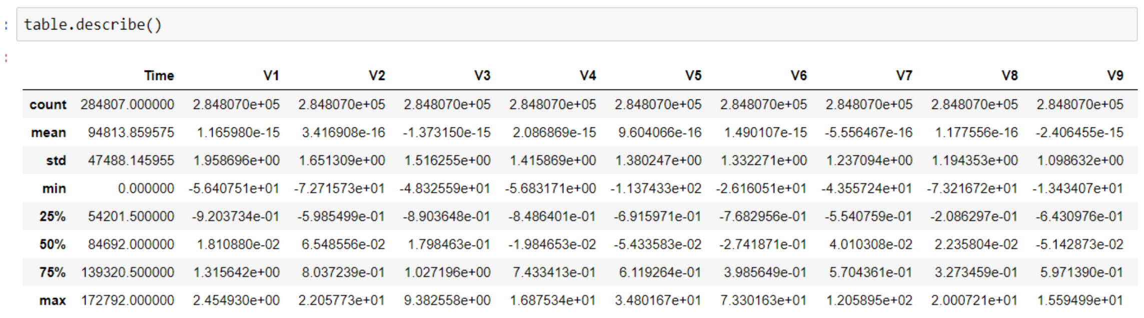
```
In [4]: table.head()
Out[4]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

.	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
.	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
.	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
.	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
.	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
.	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

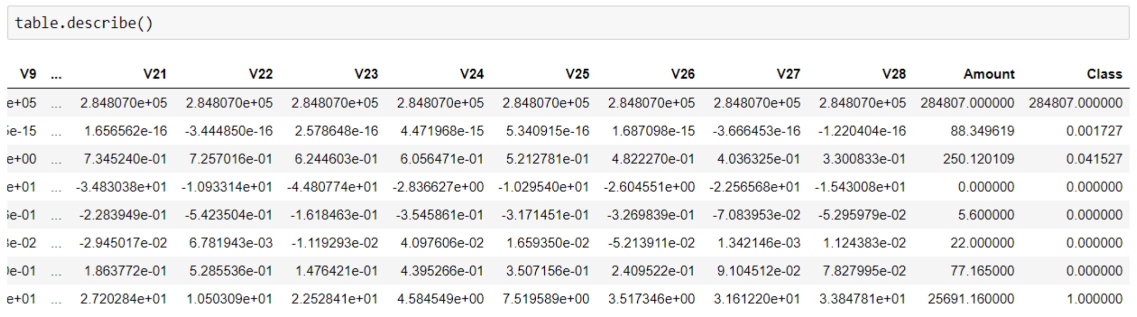
Figure 3 shows the basic statistics for the data.

Figure 3 - Basic statistics of the 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	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-16	1.490107e-15	-5.556467e-16	1.177556e-16	-2.406455e-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



```
table.describe()
```

V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
e+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	284807.000000	284807.000000
e-15	...	1.656562e-16	-3.444850e-16	2.578648e-16	4.471968e-15	5.340915e-16	1.687098e-15	-3.666453e-16	-1.220404e-16	88.349619	0.001727
e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

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.

Figure 4 - Box-plot of the whole dataset

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

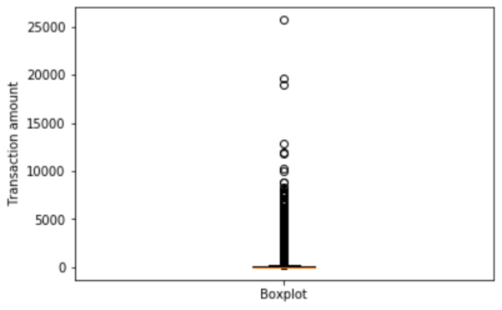
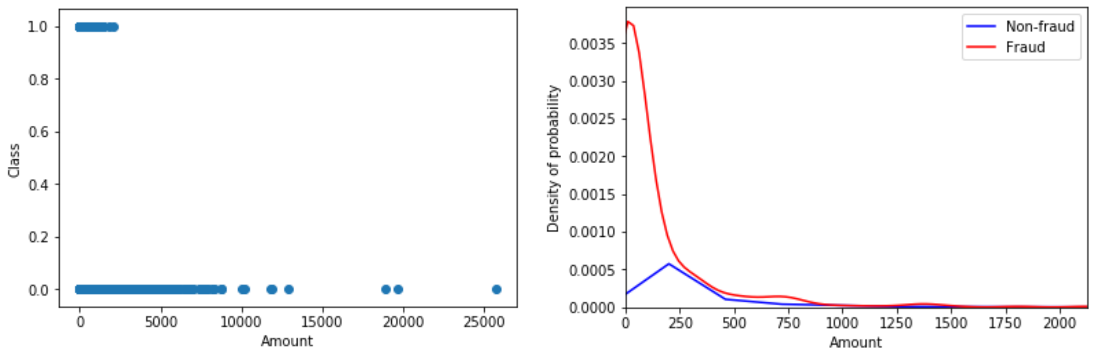


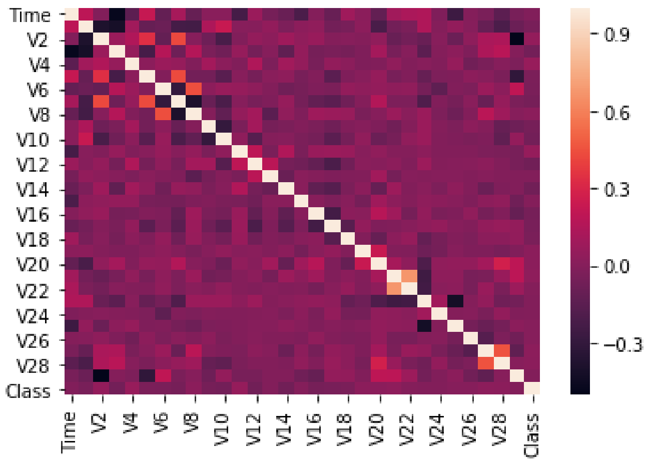
Figure 5 shows that fraudulent transactions are highly concentrated at smaller values when compared to non-fraudulent transactions.

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.

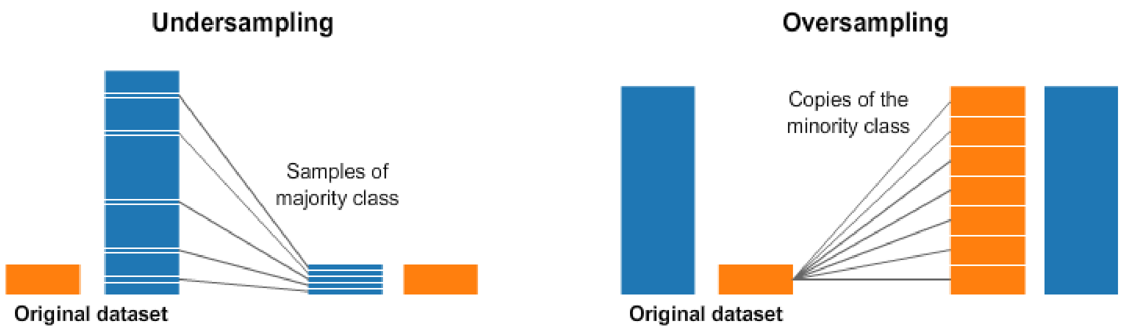


Algorithms and Techniques

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.

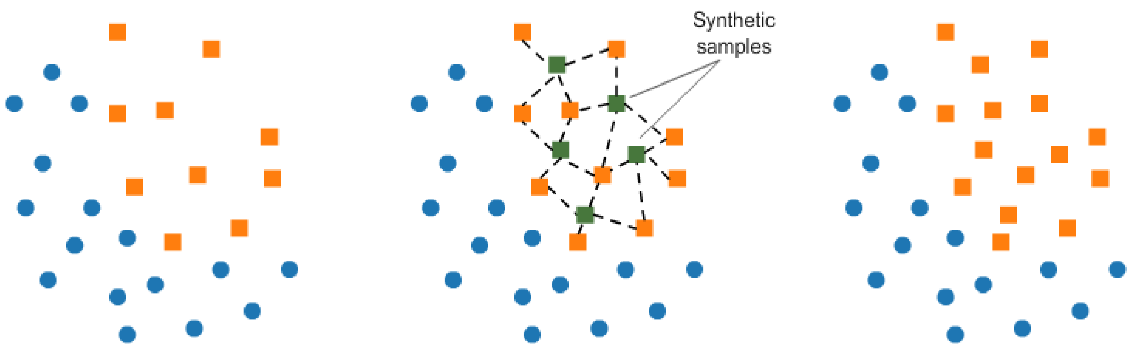
After that, I chose three different resampling techniques to balance the training dataset: random under-sampling, random over-sampling and synthetic Minority Over-sampling. Figure 7 depicts how undersampling and oversampling works and it is very intuitive to understand. SMOTE is like oversampling but 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.

III. 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.

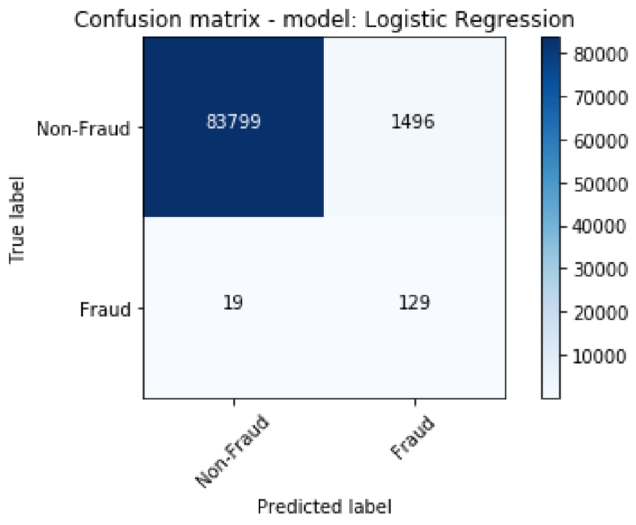
Implementation

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(('Decision Tree', DecisionTreeClassifier(random_state=42)))

classifiers.append(('Random Forest', RandomForestClassifier(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



```
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)
    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.

```
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.

IV. Results

Model Evaluation and Validation

Results using random undersampling

Table 1 - Performance of classifiers using random undersampling

Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9722	7.94%	87.16%	98.23%
Naive Bayes	0.9537	4.07%	83.78%	96.55%
KNN	0.9480	4.90%	83.78%	97.16%
Decision Tree	0.8893	1.35%	89.19%	88.67%
Random Forest	0.9646	6.07%	87.16%	97.64%
Ensemble	**0.9671**	**7.36%**	**86.49%**	**98.09%**

Table 2 – Confusion matrix of classifiers using random undersampling

Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	83799	19	129	1496
Naive Bayes	82372	24	124	2923
KNN	82890	24	124	2405
Decision Tree	75628	16	132	9667
Random Forest	83298	19	129	1997
Ensemble	**83684**	**20**	**128**	**1611**

Results using random oversampling

Table 3 - Performance of classifiers using random oversampling

Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9681	6.65%	87.84%	97.84%
Naive Bayes	0.9550	5.32%	83.11%	97.41%

KNN	0.8849	68.55%	73.65%	99.90%
Decision Tree	0.8410	75.37%	68.24%	99.91%
Random Forest	0.9151	97.25%	71.62%	99.95%
Ensemble	**0.9670**	**85.19%**	**77.70%**	**99.94%**

Table 4 – Confusion matrix of classifiers using random oversampling

Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	83471	18	130	1824
Naive Bayes	83105	25	123	2190
KNN	85245	39	109	50
Decision Tree	85262	47	101	33
Random Forest	85292	42	106	3
Ensemble	**85275**	**33**	**115**	**20**

Results using SMOTE

Table 5 - Performance of classifiers using SMOTE

Classifier	AUC	Precision	Recall	Accuracy
Logistic Regression	0.9667	14.95%	85.14%	99.14%
Naive Bayes	0.9546	5.91%	82.43%	97.69%
KNN	0.8982	48.13%	78.38%	99.82%
Decision Tree	0.8673	41.60%	73.65%	99.78%
Random Forest	0.9309	84.06%	78.38%	99.94%
Ensemble	**0.9685**	**72.02%**	**81.76%**	**99.91%**

Table 6 – Confusion matrix of classifiers using SMOTE

Classifier	True Negative	False Negative	True Positive	False Positive
Logistic Regression	84578	22	126	717
Naive Bayes	83351	26	122	1944
KNN	85170	32	116	125
Decision Tree	85142	39	109	153
Random Forest	85273	32	116	22
Ensemble	**85248**	**27**	**121**	**47**

Justification

By using random undersampling, the best model was a simple logistic regression with AUC = 0.9722, recall = 87.16% and precision = 7.94%. The number of false negatives, the worst possible classification regarding frauds, for the logistic regression was 19 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 = 0.9671 and 20 false negative cases, only 1 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 logistic regression (1496 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, we can verify the true power of ensemble models. The ensemble model combined the best part of the simple models and could get a very high AUC (0.967), high precision (85.19%) and high recall (77.7%). 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 18) and the random forest was the model with the lowest number of false positives (an astonishing result of only 3!). So, if we try to tweak the weights for each model in the ensemble model, we can achieve an even better result.

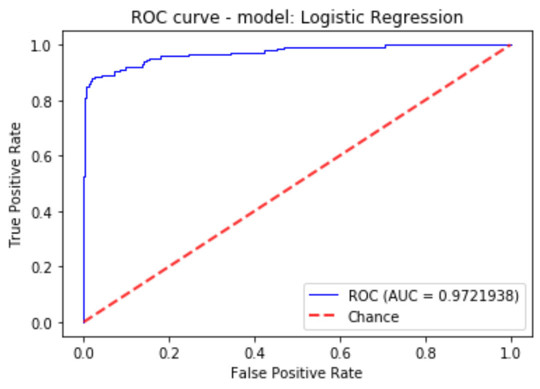
The analysis for SMOTE method is very similar to the random oversampling method. Again, we can see that the ensemble model could get the best parts of all simple models, resulting in a good performance. Once again, tweaking the weights for each simple model might be a way to enhance the result.

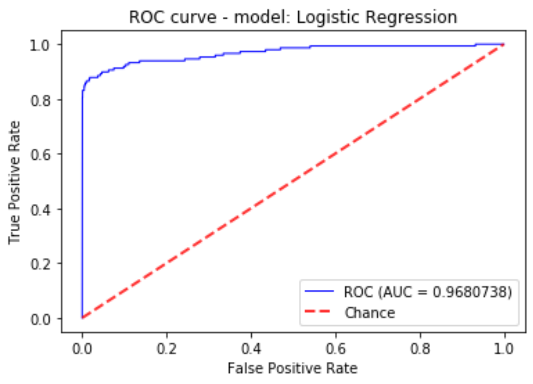
V. Conclusion

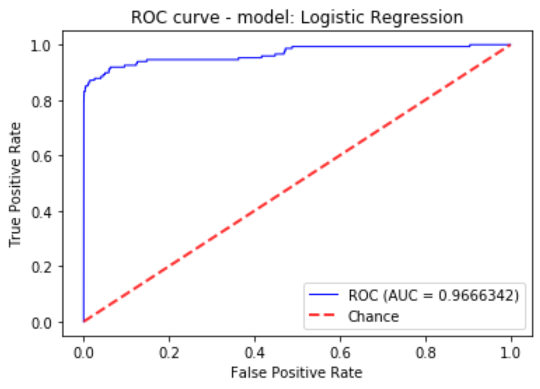
Free-Form Visualization

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.

Figure 10 - Logistic regression ROC curves for all 3 resampling techniques. From left to right and from top to bottom: random undersampling, random oversampling and SMOTE







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. I tried implementing it, but my hardware can’t run the code in an acceptable time for this project. So, I decided to not use a cross-validation.