CDA Lab 1: Recognizing Credit Card Fraud

Cas Bilstra 4381084
Roy Graafmans 4299442

https://github.com/Roy-Graafmans/CDA_CS4035

May 10, 2019

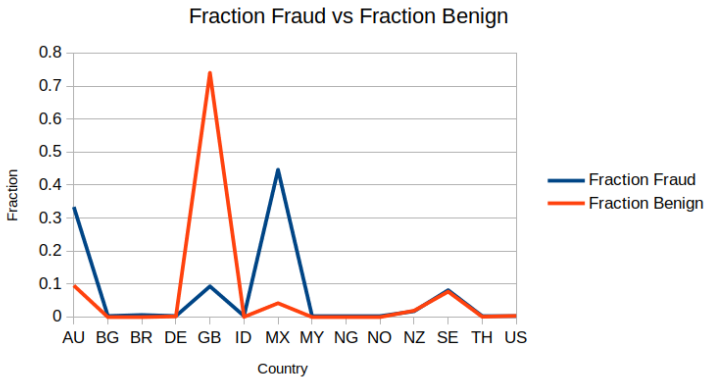
Installation instructions and instructions on how to run the program can be found in the readme.md file contained in the repository. For clarification of our choices in the code, please have a look at the applicable section in this report. Most steps will be explained in section 3.

1 Visualisation

Interesting when comparing the fraudulent and benign transactions with each other is that on average a fraudulent transation is worth 162.81 USD, whilst a benign transaction is worth 87.06 USD on average. Fraudulent transactions are thus on average worth more than benign transactions.

Also, when comparing the percentage of benign transactions which was performed in the country from which the card was issued, for benign data 2.9% if the transactions was not performed in the 'home country', whilst 4 % from the fraudulent transactions was not performed from the 'home country'.

Our last interesting finding is that when looking at the fraction of transactions from a certain country, there is a big difference between the benign and fraud data set. In the fraudulent transactions, Mexico and Australia have a much higher share than expected from looking at the benign data set. Great Britain in contrary, has a lot less fraudulent cases than expected. For the benign data, only the contries which were in the fraud data set were plotted (there are much more countries present in the data set).



To show all data points in one figure: the fraction of the total number of transactions per day for the fraudulent and benign transactions. It shows that when there are more benign transactions, there are not necessarily also more fraudulent transactions! In task1_2.ipynb the code can be found with which these results were generated.

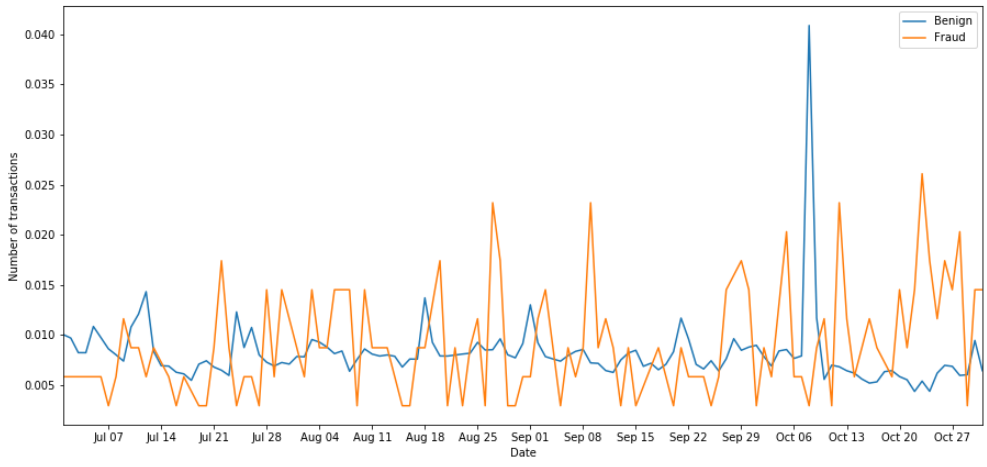
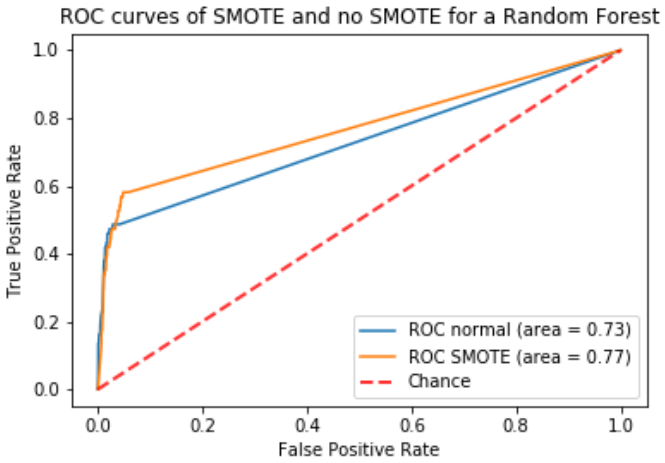
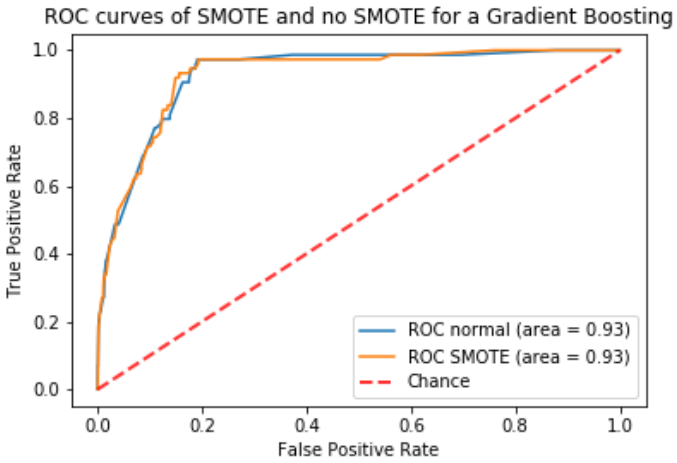


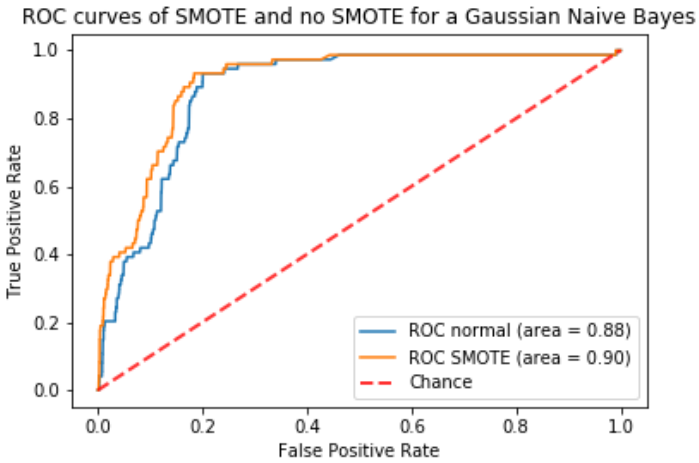
Figure 1: Fraction of total amount of transactions per day

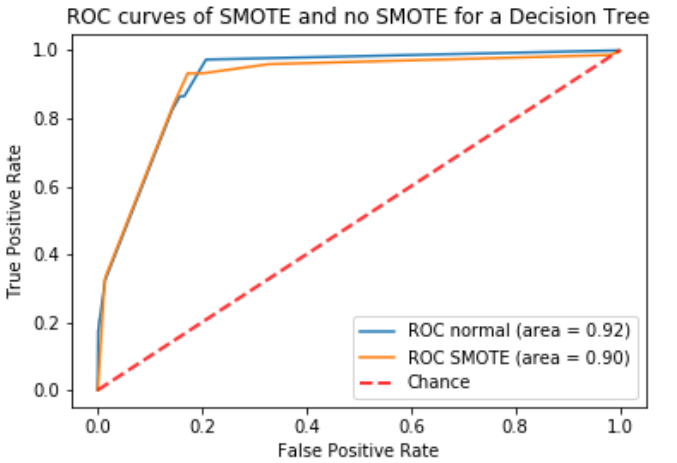
2 SMOTE

In the figures below, the obtained ROC curves for a Random forest Classifier, a Gradient Boosting classifier, a Gaussian Naive Bayes Classifier and a Decision Tree classifier are shown. 20% of the data was used as the test set, while 80 % was used to train the classifiers. The features used can be found in the jupyter code in the try_sampling method. The Area Under Curve (AUC) can be used as a metric to measure how well a certain classifier works. In the figures this AUC can be read in the legend. As can be seen, sometimes SMOTE has a bigger AUC, which means that on average, at a certain False Positive Rate, the True Positive Rate is higher, this is the case for Random Forest and Gaussian Naive Bayes classifiers. Sometimes the AUC is the same (Gradient Boosting), and sometimes the AUC is worse (Decision tree). Other metrics exist as well however, and they give some interesting insights. For the decision tree for example, no positives are found (no transaction is labeled as fraudulent) without SMOTE. This rates can be seen in the table below (TP = True Positive, FP = False Positive, FN = False Negative, TN = True Negative).The AUC might become worse, but with SMOTE allmost all positives are found (69 out of 74) at a big cost of 8142 false positives. It depends if False Positives are 'expensive' or not to be able to say if SMOTE is a good idea for Decision trees. In general, using SMOTE means that more True Positives are found at the cost of a higher False Positive rate. The Gradient Boosting and Decision Tree classifiers do not produce any TP, so we recommend using SMOTE to be able to get some Positives at least. For the other classifiers, it depends on the desired Precision (TP/(TP+FP)) if SMOTE should be used or not.









	SMOTE				No SMOTE			
Classifier	TN	FN	TP	FP	TN	FN	TP	FP
Decision Tree	39124	5	69	8142	47266	74	0	0
Random Forest	46281	43	31	985	47250	68	6	16
Gradient Boosting	40821	12	62	6445	47261	74	0	5
Gaussian Naive Bayes	38657	7	67	8609	46801	63	11	465

3 Classification

The data was processed in various stages, which will be described below. The applicable code can be found in task3.ipynb, so please have a look and run that code!

3.1 Parsing

The provided data was first read from the csv into a pandas Dataframe. The bookingdate column was removed, because using it would be cheating, as it is only present for the fraudulent cases. The transactions with label 'Refused' were removed as well, because it is unknown if these cases were fraudulent or not. Furthermore, rows containing 'NA' in the issuer id or mail id were discarded, because they would brake features we calculate or use. Also, the transaction ID is not a feature, so that column was dropped as well.

3.2 Pre-Processing

In this step, our custom features were added and calculated. They are described in section 4. Because NTI and NTC can only be taken over the training data to not polute the test data, these are added after the folding process, in the calculate_features method. Furthermore, the categorical data is processed into integers, with which the Classifiers can work. Different types of under/over sampling were tried, which can be seen in task1_2.ipynb in the last section. For our final classifiers, a Decision Tree Classifier (DTC) as our White-Box algorithm and a Random Forest Classifier (RFC) as our black-box algorithm, SMOTE was clearly the best choice. Random Under Sampling (RUS) and SMOTE both gave many False Positives (around 8000-9000, as can be seen in task1_2.ipynb) for the DTC. Near-Miss (NM) undersampling performs very bad, with the test set mostly being labeled as False Positives. The performance of RUS fluctuated a lot, so it was decided to use SMOTE instead for DTC. For RFC, SMOTE clearly outperformed RUS and NM, being the only method with acceptable performance (RUS and NM producing mostly False Positives). Ofcourse, SMOTE is only performed on the training set after folding for our final classifiers. The sampling is executed in the KFold_PRC function of task3.ipynb.

3.3 Learning algorithms

Our final classifiers can be seen in task3.ipynb.

3.3.1 White Box Algorithm

For the whitebox classifier we used a Decision tree classifier. This type of classifier gives a nice insight of the inner workings when exported to a graph. For the example in figure 3 we set the max_depth to 5 to make sure the image was small enough to show here. In our final classifier the max depth was not limited, resulting in a much more complex tree. The concept is the same though, and one can follow the tree exactly to show how the algorithm comes at a certain decision. Going a branch to the left means the decision rule of the node evaluated to True, whilst going to the right means the decision rule of the node evaluated to False.

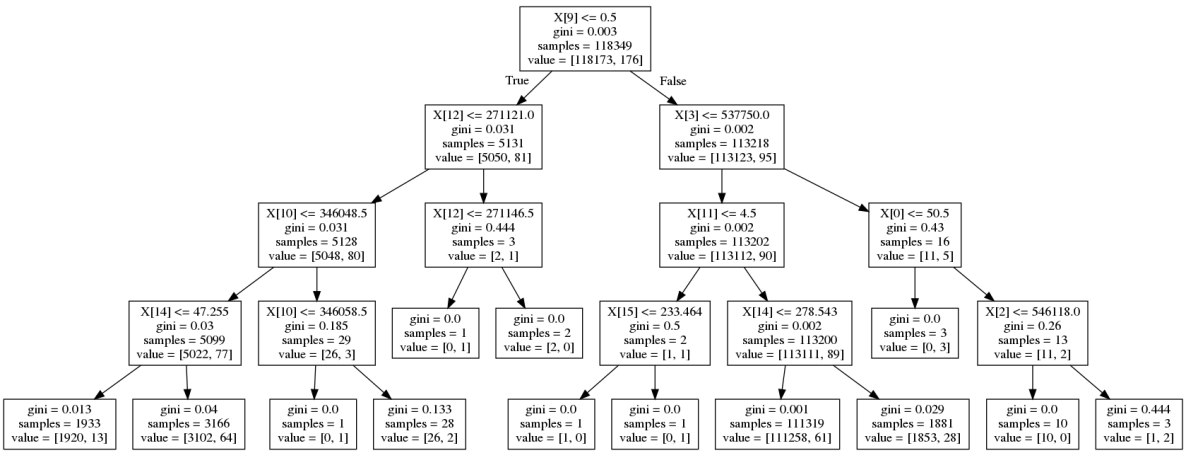


Figure 3: Inner working of the decision tree classifier with a max depth of 5

3.3.2 Black Box Algorithm

As Black Box algorithm, a Random Forest Classifier (RFC) was used. It was found that using 10 estimators produced the best results. With more estimators, the training took a lot longer, resulting in overfitting the dataset. Less estimators yielded worse results, with a lot more FP and almost the same amount of TP. A RFC is black-box in the sense that the used estimators

(Decision Trees) are combined with weights. It can then not be explained easily how much each value of a certain feature vector (row) contributed to the outcome of the classifier. This is the case because each path through a decision tree (which can be explained) is now only partly considered, and combined with other paths. It can then even be the case that two conditions lead to contrary outcomes (e.g. USD Amount > 90 leads to label 'Fraud' and in another tree USD Amount > 90 leads to label 'Benign'). It might thus be very difficult, if not impossible, to explain how the algorithm came to a certain result.

3.3.3 Comparison

When comparing the white-box model to the blackbox model it is quite hard not to overfit the Decision Tree and therefore the results are a slightly worse. It is harder to overfit a random forest since those are all different estimators and they vote together to make a decision. Although the TP are almost equal there are almost 10 times as many FP for the whitebox.

3.4 Post-Processing

We did not do any post-processing of the data, other than visualizing PR curves.

4 Bonus

Custom features were implemented, in order to use knowledge from relationships in the data. These features were used when comparing classifiers, drawing the ROC/PR curves, etc. They can be seen implemented in the preprocess part of task1 as well as the in-process data aggregation method in task3.

4.1 Home country

A new feature was added (1 - True or 0 - false) which states if a transaction happened in 'home country'. A transaction happens in home country if the 'shoppercountry' feature has the same value as the 'issuercountry' feature, which means a customer is shopping in the country where the card issuer is located as well.

4.2 Number of transactions in this country for this card (NTC)

Another feature was added, which takes all data in the training set and calculates how many transactions were made with a certain card in a certain country. This feature is added for each transaction. In the test set, these values are appended before testing as well.

4.3 Number of transactions for this interaction type for this card (NTI)

Similar to NTC, the number of transactions per interaction category for a card was calculated based on the training data as well.

4.4 USD amount

All amounts were converted into their $ amount, to be able to compare them.

4.5 Avarage amount per card

For every card_id in the training set, the avarage expense was calculated and added to the training set as well as the test set. If the card id in the test set was not in the training set, the overall avarage expense was added to this card id.

4.6 Difference between the amount and the avarage amount

For every transaction in the data set the difference between the transaction amount and avarage amount is calculated.