Machine Learning Engineer Nanodegree

Capstone Project

Carina Thobe

04.06.2017

1. Definition

1.1 Project Overview

Fraud Detection is a very important research area for businesses and banks since it has a direct impact on their profits. There are many industries that have an interest in detecting fraudulent behaviour as early as possible: Banks need to identify criminal transactions that are drawn illegally from their customers' accounts, telecommunication companies need to know if the customer is trustworthy for a device payable by instalment and insurance companies desire to reveal insurance fraud.

The European Central Bank estimated that in 2012 the total value of credit card fraud was about to be 1.33 € billion in 2012, which represents an increase of almost 15% compared to the previous year (European Central Bank, 2014). This example illustrates that fraud is a very important issue that companies and banks are highly interested in fixing. Therefore different data mining and machine learning methods are in use, such as classification, regression, clustering or prediction techniques in order to detect fraudulent cases (Dal Palozzo, 2015).

For me the topic is interesting because I find it very fascination to apply machine learning methods that are able to fight crime such as image recognition in video surveillance, text mining in chat messages or fraud concerning online banking/finance services/credit card.

The data for the analysis are credit card transactions generated by European credit cardholders. The dataset contains 0.172% cases of fraud. It was released by a research collaboration of Worldline and the Machine Learning Group of the University of Bruxelles (ULB) (Caelen, Dal Pozzolo, Johnson, & Bontempi, 2015). The datset is downloadable from kaggle under https://www.kaggle.com/dalpozz/creditcardfraud.

The aim is to classify whether a transaction is fraud or not. Therefore I am going to apply different supervised learning methods and compare the results to each other in order to find out which one performs best.

1.2 Problem Statement

A bank institute is interested in identifying credit card fraud as early as possible in order to avert damage from its customers and from themselves. The aim is to predict for a transaction whether it is fraud or not. Therefore they can apply machine learning and data mining methods that use past data which is already classified in fraud and non-fraud cases. This data should have one dependent variable (1=fraud/0=no fraud) that is going to be predicted and several independent variables such as amount of transaction, timestamp of transaction, target destination/country, country of transaction origin, frequency of transactions of card holder, ratio of foreign transations of card holder and so on. The quality of the model depends also on the quality of the predicting variables.

With using supervised learning methods such as regression, decision trees, neural networks, SVM or Bayes Learning it is possible to train a model that will predict whether a case is fraud or non-fraud when given new, previously unseen data. After examination whether the prediction was right or not, those cases can be used to update the algorithm and to make it better (this is the machine learning part).

The objective of the project is to build a model that is able to predict whether a transaction is fraud or not. I am going to implement different supervised learning methods:

- Logistic Regression (this will be the benchmark model)
- Support Vector Classifier
- Decision Tree
- Random Forest Classifier

1.3 Metrics

There are classic accuracy measurements such as

- Accuracy = (TP+TN)/Total
- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)

True Class ↓Predicted Class→	Fraud (1)	Non Fraud (0)
Fraud (1)	TP = true positive	FN = False negative
Non Fraud (0)	FP = false positive	TN = True negative

In our case it is important to maximize the number of true positive labelled cases and to minimize the false negative labelled cases. The false positive cases are not as bad for the business as the false negative cases, because the latter will cause tremendous harm to the business. The false positive cases will probably result in extra work for manually checking whether it is really fraud or not. This makes *Recall* an important metric to examine. Accuracy is not the measurement of choice because the data is highly imbalanced and precision does not take the worst case of false negatives in account (Descoins, 2013).

The authors of the dataset already point out the difficulty due to the imbalance of fraud and non-fraud cases and therefore recommend using the *Area Under the Precision-Recall Curve* (Caelen, Dal Pozzolo, Johnson, & Bontempi, 2015; Dal Palozzo, 2015). The Precision-Recall Curve is calculated as plotting a set of pairs of precision and recall for a number of thresholds and connections those dots with a line. Then the area under the curve can be calculated as definite integral (Richardson & Domingos, 2006).

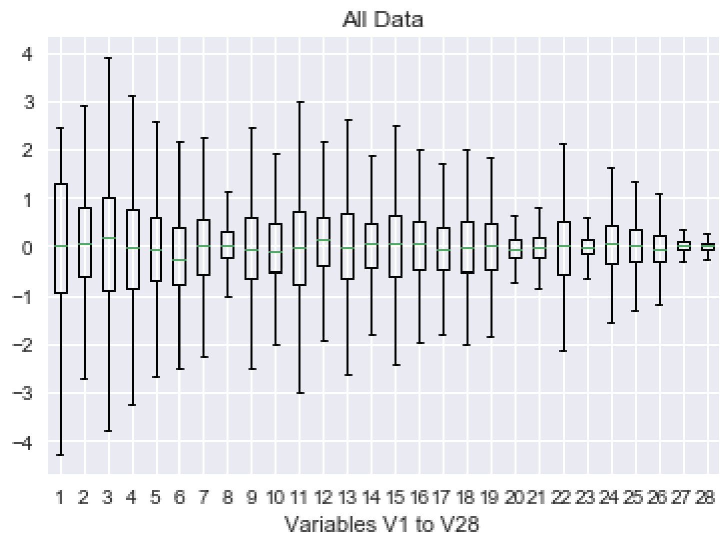
I am going to consider both recall and area under the precision recall curve.

2. Analysis

2.1 Data Exploration and Visualization

The dataset used for my analysis contains 284,807 credit card transactions of European credit cardholders whereof in 492 cases fraud was detected, which is only 0.172%. The variables of the dataset are all transformed by a PCA transformation and unfortunately there is no meta data that explains the meaning of the variables (due to confidentiality). There are 2 variables that are not transformed: “time” and “amount”, whereas the latter is the amount of the transaction and the first is the time between each transaction and the first transaction in seconds. The variable “Class” is the response variable (1=fraud, 0=no fraud). The dataset is real world data that was published by a research collaboration of Worldline and the Machine Learning Group of the University of Bruxelles (ULB) (Caelen, Dal Pozzolo, Johnson, & Bontempi, 2015).

To get a graphical overview I plotted all 28 variables as boxplots into one graph:



Even though the data is already transformed by a PCA there is still a lot of variance between the variables. For example, variables 27 and 28 have a small variance and variable 3 has a big variance, which can be seen when looking at the size of the box and the range of the whiskers. For variable 27 and 28 the box is very short, the upper and lower quartiles are quite close to each other and the whiskers are not too far apart. For variable 3 the box is much taller, which means that upper and lower quartiles (which include 50% of the data) are far apart as well as the whiskers. The median (the middle line of the box) is for all variables very close to zero.

The following table shows the statistics for the data:

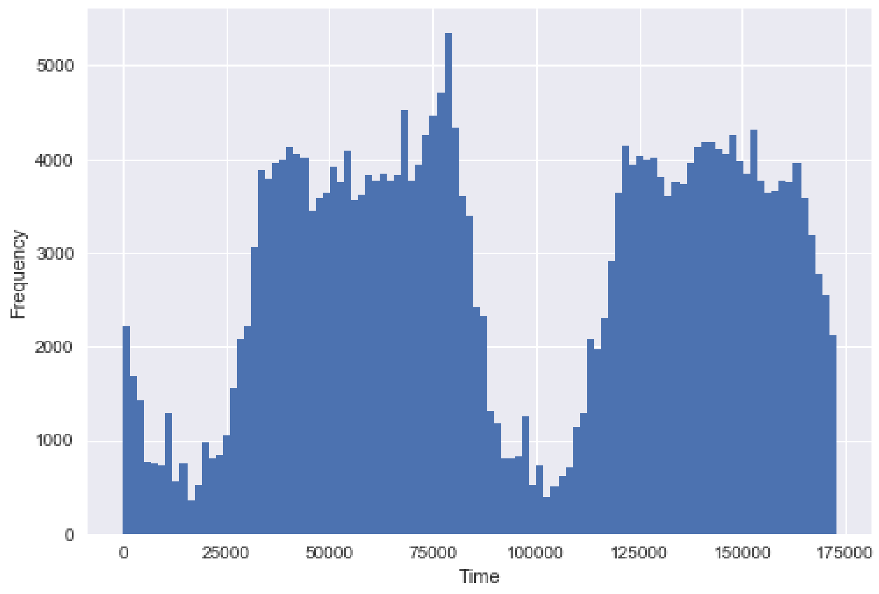
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807	284807	284807	284807	284807	284807	284807	284807	284807	284807
mean	94813.86	0	0	0	0	0	0	0	0	0
std	47488.15	1.959	1.651	1.516	1.416	1.38	1.332	1.237	1.194	1.099
min	0	-56.41	-72.72	-48.33	-5.683	-113.74	-26.16	-43.557	-73.22	-13.434
25%	54201.5	-0.92	-0.599	-0.89	-0.849	-0.692	-0.768	-0.554	-0.209	-0.643
50%	84692	0.018	0.065	0.18	-0.02	-0.054	-0.274	0.04	0.022	-0.051
75%	139320.5	1.316	0.804	1.027	0.743	0.612	0.399	0.57	0.327	0.597
99%	170560.9	2.237	3.802	2.728	4.248	3.425	4.2	2.696	2.076	2.987
max	172792	2.455	22.058	9.383	16.875	34.802	73.302	120.59	20.007	15.595

	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19
count	284807	284807	284807	284807	284807	284807	284807	284807	284807	284807
mean	0	0	0	0	0	0	0	0	0	0
std	1.089	1.021	0.999	0.995	0.959	0.915	0.876	0.849	0.838	0.814
min	-24.588	-4.797	-18.68	-5.792	-19.21	-4.499	-14.13	-25.163	-9.499	-7.214
25%	-0.535	-0.762	-0.406	-0.649	-0.426	-0.583	-0.468	-0.484	-0.499	-0.456
50%	-0.093	-0.033	0.14	-0.014	0.051	0.048	0.066	-0.066	-0.004	0.004
75%	0.454	0.74	0.618	0.663	0.493	0.649	0.523	0.4	0.501	0.459
99%	3.254	2.291	1.699	2.514	2.15	1.926	1.875	2.29	2.069	2.263
max	23.745	12.019	7.848	7.127	10.527	8.878	17.315	9.254	5.041	5.592

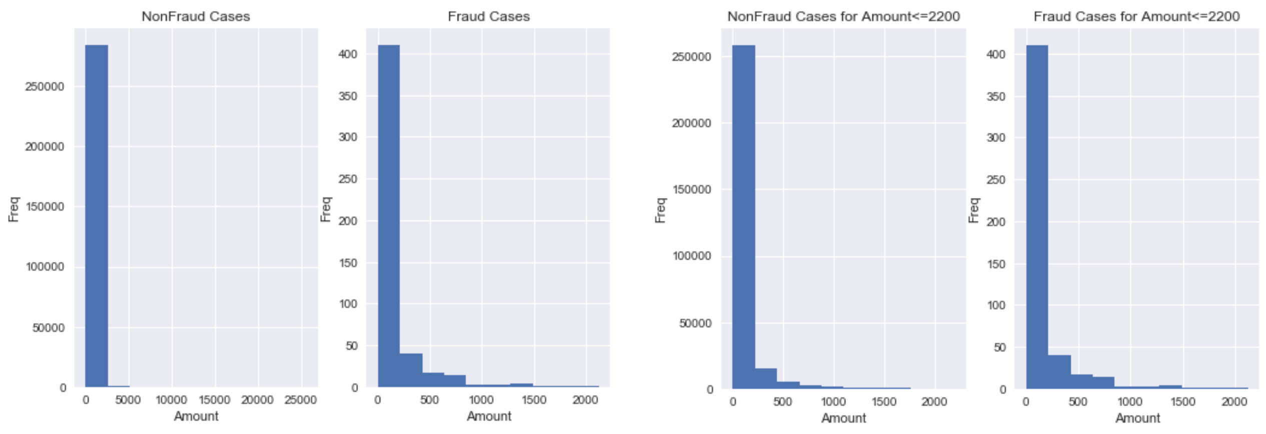
	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
count	284807	284807	284807	284807	284807	284807	284807	284807	284807	284807
mean	0	0	0	0	0	0	0	0	0	88.35
std	0.771	0.735	0.726	0.624	0.606	0.521	0.482	0.404	0.33	250.12
min	-54.498	-34.83	-10.93	-44.81	-2.837	-10.295	-2.605	-22.566	-15.43	0
25%	-0.212	-0.228	-0.542	-0.162	-0.355	-0.317	-0.327	-0.071	-0.053	5.6
50%	-0.062	-0.029	0.007	-0.011	0.041	0.017	-0.052	0.001	0.011	22
75%	0.133	0.186	0.529	0.148	0.44	0.351	0.241	0.091	0.078	77.165
99%	2.412	1.932	1.53	1.509	1.064	1.204	1.159	0.931	0.541	1017.97
max	39.421	27.203	10.503	22.528	4.585	7.52	3.517	31.612	33.848	25691.2

All variables have 284,807 values, which is also the number of records and means that there are no missing values. All variables are numerical, except the target variable “Class” which is categorical with 0 meaning non-fraud and 1 meaning fraud. Variables V1 to V28 are already transformed which can be seen by looking at mean, which is 0 for all variables, and standard deviation, which is close to 1. As already seen in the boxplots there is nevertheless some variance since the maximum and the 99% quartile have a huge range, e.g. for variable 10. Therefore I am going to set all values above the 99% quartile to the value of the 99% quartile.

The variables Time and Amount are not standardized. Since 'time' represents the seconds between the first transaction in the dataset and each transaction it has not much analytical value and can therefore be eliminated.



The variable “Amount” has a mean of 88 and the maximum is for the fraud data different than for the nonfraud data: Fraud data has a maximum amount of 2,125 and normal data has a maximum amount of 25,691. The first two graphs show the amount for non-fraud and for fraud cases. The third and fourth graphs are zoomed in for amount less than 2,200. This shows that the distribution is similar for both. Therefore I am going to normalize the amount variable since this has not been done before and not scaling would result in overweighting the variable with the larger scale.



2.2 Algorithms and Techniques

In the following I am giving a brief overview of the functionalities of the algorithms I decided to use, discuss some advantages and areas of application.

As benchmark model I am going to use a logistic regression. Logistic regression models are very common in real world industries for use cases such as Credit Scoring. They are used to predict whether an individual customer is likely to commit fraud and not to be able to pay the monthly amount. It can be used when the dependent variable is binary (which is the case in this analysis) and when the aim to predict the probability that a person is going is going to do something.

The logistic regression is a non-linear transformation of a linear model by using the logistic distribution which adjusts the prediction to lie between 0 and 1, which are the minimum and maximum of the dependent variable. It can be described as:

$$logit(p) = \ln(\frac{p}{1-p}) \;=\; \beta_0 + \beta_1 x_1 + \cdots + \beta_i x_i$$

whereas x_i represents the independent variables, β_i the regression coefficient and p is the probability y=1 is going to happen (Whitehead).

The regression coefficients β_i cannot directly be interpreted as in linear regression. Therefore the expression can be exponentiated, which gives an expression called the odds:

$$F\left(\frac{p}{1-p}\right) = e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i} = odds$$

This makes it possible to interpret e^{β_i} as the effect x_i on the odds ratio, which is the probability of y=1 divided by the probability for y=0. If $e^{\beta_i} < 1$ then x_i has a negative impact; if $e^{\beta_i} > 1$ it has a positive impact (Whitehead).

Logistic regression has some advantages: The assumptions of linear regressions, e.g. normal distribution of residuals, can be violated. The only necessary assumption is that there exists a smooth linear decision boundary and after transformation it can even handle non-linear decision boundaries. It is quite robust against outliers. It is easier to interpret especially when you have a lot of independent variables. With a lot of independent variables running time is still quick and the estimation is still reliable. It gives insights into the impact of the predictors and its significance (Logodds resp. odds ratio) (Rae, Krishnan, Kolluru, & et al.).

For the challenging models I am going to implement a SVM , a decision tree and a Random Forest Classifier.

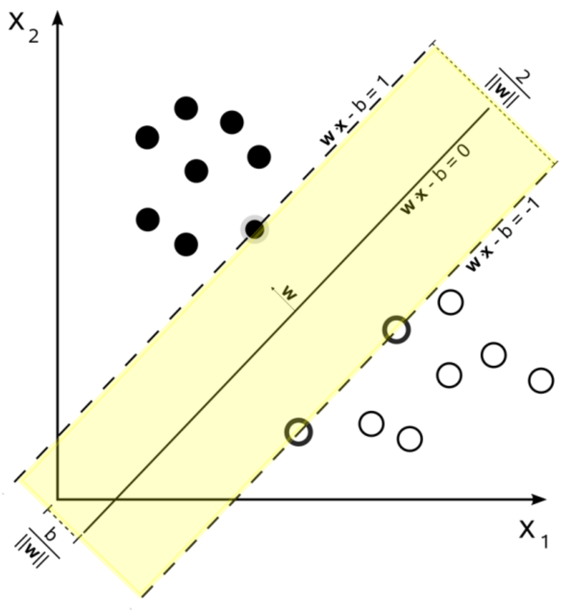
SVM is a supervised learning algorithm that is suitable for classification as it is the case in this analysis. It needs training data that is already labelled as input:

$$\{(\vec{x}_i y_i) |\ i = 1 \dots n;\ y_i \in \{-1,1\}\}$$

with y_i representing the label of the class of $\vec{x}_i$ and each $\vec{x}_i$ is a p-dimensional vector (Support Vector Machine).

Then it tries to separate the data points by putting a hyperplane between the points belonging to one class and those belonging to the other class. By doing so it tries to maximize the margin to any of the nearest points and therefore find the optimal separating hyperplane (Kowalczyk, 2014).

This is also displayed in the graphic that follows:



Source: (Support Vector Machine)

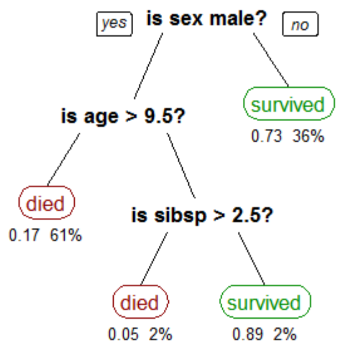
In the graphic the hyperplane is written as: $\vec{w} * \vec{x} - b = 0$ with $\vec{w}$ being the normal vector to the hyperplane. There are many possible ways that the hyperplane can be placed within the linearly separable data points and the SVM has some options to choose. Therefore I am going to explain only one, the linear SVM. As it can be seen in the graphic, the distance between the maximum and minimum boundary is $\frac{2}{||\vec{w}||}$ and to maximize this distance it is necessary to minimize $\vec{w}$. In order to avoid that points are below or above of the maximum/minimum boundary (the yellow marked area in the graphic) it is possible to formulate a side condition that needs to be satisfied:

$$y_i(\vec{x_i} * \vec{w} - b) \geq 1,\ for\ all\ 1 \leq i \leq \text{n}$$

Minimizing $\vec{w}$ while satisfying the side condition above leads to the classifier: $\vec{x} \rightarrow sgn(\vec{w} \overline{*\vec{x}} - b)$. The name of the method derives from the fact that the hyperplane is calculated by the vectors that lie the closest to it and those are called support vectors (Support Vector Machine).

SVM avoids overfitting, since not all data points are relevant for the decision boundary. It is possible to set a penalty parameter for dealing with data that has a location violating the decision boundary. By using the kernel trick you can use SVM also for non-linearly separable data. Also with using the kernel trick you can take advantage of domain knowledge. It performs well when there are lots of input variables. But with having more observations and more variables running time can get quite large. Further, it is not intuitive and not easy to interpret. It is necessary to do data preparation beforehand and to tune the parameter by an expert (OracleHelpCenter, Support Vector Machines; Support Vector Machine).

Similar to logistic regression, decision trees can be applied in marketing in order to predict a customer's probability to buy/to churn, or for Credit Scoring in order to predict fraud. A decision tree splits the dataset into a subset and splits this again into a subset and so on. This so-called *"recursive partitioning"* is continued until the subset of a node contains only data that has the same value in terms of the target variable or when the prediction cannot be improved any more (Decision tree learning). The graphic illustrates it for the Titanic Dataset as an example:



Source: (Decision tree learning)

This is called *"top-down induction of decision trees*" by a "*greedy algorithm"* (Decision tree learning). The tree consists of decision nodes and leaf nodes. A leaf node is the final classification of the samples and has no more branches, similar to a leaf of a real tree and a decision node has two or more branches. The very first node is the root node. In each node the data gets splitted into a subset based on a metric. There are different metrics such as information gain or gini impurity as a measurement of homogeneity when splitting the data. I am going to explain only gini impurity as this is the one I am going to use later on. Gini impurity measures how often a data point from a node would be categorized false if it was done randomly corresponding to the distribution of classes in the node. It can be calculated by taking the sum of the probability that a data point with a given label is chosen times the probability for misclassification. The aim is to get nodes that contain homogeneous data and therefore it reaches its minimum (zero) when a node contains only data points with the same label (Decision tree learning).

Decision Trees are simple to understand and easy to interpret. Variables that are numerical or categorial can be handled. Also extensive data preparation is not necessary. They are robust and quick in computing time even with large datasets. It is possible to validate decision trees. All of this plus the fact that they are using a white box model it makes it easy to use. On the other hand they tend to overfit and get too complex – especially when you have a lot of independent variables. In this case they do not generalize will - but you can fix this by pruning earlier. It is possible that you get a local optimum instead of the global optimum. Further, in specific settings it can happen that trees are not robust, which means that a small change in training data causes a huge change in the tree. There are also problems that are not well represented by trees, such as XOR. If you have categorial variables with many levels it might happen that the tree is biased and prefers those variables with many levels (Decision tree learning).

Since SVM avoids overfitting and since decision trees have the risk to overfit this is a nice contrast to make sure that there is no overfitting.

The third challenging model will be a Random Forest Classifier, which computes many decision trees and gives the mode of the classes back. The algorithm proceeds as follows:

There are N data points in the training set and then N cases are drawn randomly with replacement from the original data. From M independent variables $m \ll M$ variables are taken randomly on each node in order to split the data into subset. Finally each tree is grown as big as possible without pruning (Breimann & Cutler).

The single decision tree itself is computed as explained above based on the gini impurity metric. Also, as pointed out before, a single decision tree is sensible to overfitting - but a random forest is not. Further it does not need cross-validation because each tree is using another bootstrap from the whole dataset which is also the reason why it does not overfit. Random Forest is efficient and quick even on large dataset that makes it perfect for this analysis (Breimann & Cutler).

As shown above all chosen algorithms might perform great on the dataset and are a good choice for the present classification problem.

2.4 Benchmark

A very simply heuristic model could be predicting “non-fraud” in 100% of test cases and that would lead to an accuracy of 99.83% which is already quite high. This is due to fact that the dataset has only 492 cases of fraud and is thereby imbalanced. With regard to the business background this is not meaningful since just one fraud case can already cause a lot of damage to the business. Therefore a bank has a very strong interest in detecting every single fraud case.

This makes it clear that a heuristic model is not adequate. The benchmark model for this problem is a logistic regression which can in most cases deliver robust predictions and is well-known and well in use in many companies.

As explained in the following chapter, I split the data into 70% training / 30% testing set and then do undersampling in order to get a balanced dataset and. This results in 690 training samples with 50% frauds and 85,443 testing samples with the original 0.713% rate of fraud.

I implemented a logistic regression model using *LogisticRegression* from *sklearn.linear_model.* It has the following performance:

```
Performance of Prediction by Logistic Regression
Confusion Matrix
[[83030  2266]
 [   15   132]]

False negative: 15
False positive: 2266

Area under the precision recall curve: 0.477
Recall: 0.898
```

So there are 15 false negatives of 147 fraud cases in the testing set. The model was trained on the sampled data and now it was tested on the unsampled data, which contains the real percentage of fraud and thereby too many false positives were generated. Recall is 0.898 and the area under the precision recall curve is 0.477. This is going to be the benchmark for the following models.

3. Methodology

3.1 Data Preprocessing

After having examined the statistics of the dataset and the graphs, there is to conclude that not too much preprocessing is needed because the data was already transformed by a PCA. A PCA is also a method to reduce dimensions.

First I will transform the variable "amount" using *StandardScaler* from sklearn preprocessing package, which gives a mean of 0 and a standard deviation of 1.

Then I am going to take care of outliers and will set the values that are above the 99.9% percentile to the value of the corresponding 99.9% percentile. This is necessary because some variables have extreme values, e.g. V7 has a maximum of 120,589 while its mean is 0, its standard deviation is 1.332 and the 99.9% percentile is 7.97.

Next I will divide my sample into a training and test sample using *train_test_split* from *sklearn.cross_validation* and splitting into 70% training data/ 30% testing data. I will not draw a validation set because the number of fraud cases is already very low. For example drawing 10% will result in only 49 cases of fraud or drawing 1% would only result in 5 cases, which is very few.

Then I am going to do undersampling of my sample which means that I'm going to take all the fraud data and draw randomly the same amount of non-fraud cases. I will then have a balanced sample of 50% fraud and 50% non-fraud. Therefor I am going to use the package *RandomUnderSampler* from *imblearn.under_sampling*. This is the result:

Training data Undersampling	*Test data*
690 cases	85,443
345 fraud cases = 50%	147 fraud cases = 0.173%

Similar to undersampling the data I try oversampling which does replicate synthetically the minority class and will result in a balanced dataset as well. I am going to use the package *smote* from *imblearn.oversampling.*

Training data Oversampling	*Test data*
398,038 cases	85,443
199,019 fraud cases = 50%	147 fraud cases = 0.173%

After fitting the model and testing with the test data, I will also test against the whole data set. The whole dataset contains also the data that was used for fitting, which could be a problem for overfitting.

3.2 Implementation

To compare the models I implemented a function that calculates the area under the precision recall curve and the recall. It is called *performance_metric* and I used the library *metrics* from *sklearn* for the **confusion matrix** and the **area under the precision recall curve** as well as **recall**. Then I implemented each model by using the relevant library. First I initialize the model, second I fit it and then I make the prediction. The prediction can then be evaluated using the function *performance_metric*. This makes it possible to compare the performance of all models.

First I tried Support Vector Classifier using *sklearn.svm.SVC* and got the following results:

```
Performance of Prediction by SVM
Confusion Matrix
[[81347  3949]
 [   11   136]]

False negative: 11
False positive: 3949

Area under the precision recall curve: 0.479
Recall: 0.925
```

The SVC reaches almost the same value for the area under the precision recall curve, which is very low with 0.479 and the number of false positives is even higher than before. Recall is very good with 0.925 and also the number of false negatives decreased.

Second I tried a Decision Tree using *sklearn.tree.DecisionTreeClassifier:*

```
Performance of Prediction by Decision Tree
Confusion Matrix
[[78017  7279]
 [   17   130]]

False negative: 17
False positive: 7279

Area under the precision recall curve: 0.451
Recall: 0.884
```

The Decision tree has more false negatives than the two models before (17). Also the area under the precision recall curve and recall are worse than for the benchmark model.

Third model was the Random Forest Classifier (*sklearn.ensemble.RandomForestClassifier*):

```
Performance of Prediction by Random Forest Classifier
Confusion Matrix
[[83245  2051]
 [   14   133]]

False negative: 14
False positive: 2051

Area under the precision recall curve: 0.483
Recall: 0.905
```

The random forest classifier is slightly better than the benchmark model in all metrics.

After checking with all the data the area under the precision recall curve improves slightly as well as recall but the number of false positives increases dramatically:

```
Performance of Random Forest Classifier when tested with whole (unbalanced) dataset
Confusion Matrix
[[277314   7001]
 [    17    475]]

False negative: 17
False positive: 7001

Area under the precision recall curve: 0.515
Recall: 0.965
```

Summary of the Undersampled data results:

Undersampled Data	Area under the precision recall curve	Recall	False negative / false positives
Logistic Regression Benchmark	0.447	0.898	15 / 2266
SVM	0.479	**0.925**	11 / 3949
Decision Tree	0.451	0.884	17 / 7279
Random Forest	**0.483**	0.905	**14 / 2051**

In the next step I tried oversampling of data, which replicates the minority class and provides bigger samples than the undersampling algorithm and obtained the following results, presented as summary:

Oversampled Data	Area under the precision recall curve	Recall	False negative / false positives
Logistic Regression Benchmark	0.487	**0.917**	47 / 7438
SVM	Too long running time		
Decision Tree	0.592	0.782	32 / 171
Random Forest	**0.863**	0.816	**27 / 12**

With oversampled data I was able to improve the area under the precision recall curve for the Random Forest Classifier. This one had a good recall and a good value for the area under the precision recall curve. Also it had the best portion of false negatives and false positives. The number of false negatives is very important, because that are not identified fraud cases which are very harmful in real life. But also the number of false positives should not increase too much, because every single case has to be checked by an employee within a short amount of time. If that increases up to 7000 cases (as it does by the Logistic Regression) then the model is not practicable for real life applications. Also the SVM had a very long running time which would not be suitable for that as well. Therefore I consider the Random Forest Classifier as the best model compared to the other ones.

4. Results

In order to validate the chosen model, I let it run with the whole data. This is the result:

```
Confusion Matrix
[[284299     16]
 [    28    464]]

False negative: 28
False positive: 16

Area under the precision recall curve: 0.955
Recall: 0.943
```

On the whole dataset, which contains 0.173% or 492 cases of fraud, the model was able to identify 464 correctly. It missed on 28 observations and it misclassified 16 samples. Both metrics were very good with recall at 0.943 and the area under the precision recall curve at 0.955.

The final model was chosen after comparing it to the other ones implemented, which performed worse on the metrics as discussed in the previous chapter.

It is reasonable and aligning with the expectations beforehand: It classifies the majority of fraud correctly into fraud. Nevertheless there is always a percentage of misclassifications – if there would not be any misclassifications the model would be overfitted. In this case the number of false negatives is only 5.7% of the fraud cases and therefore reasonably low enough. The number of false positives is with 3.2% of the fraud cases also realistic – those cases result in manually checking by a human.

Feature Importances

V1	0.012	**V11**	**0.272**	**V21**	0.041
V2	0.009	**V12**	**0.154**	**V22**	0.005
V3	0.020	**V13**	0.009	**V23**	0.004
V4	**0.119**	**V14**	**0.125**	**V24**	0.004
V5	0.006	**V15**	0.005	**V25**	0.006
V6	0.005	**V16**	0.010	**V26**	0.006
V7	0.012	**V17**	0.032	**V27**	0.005
V8	0.012	**V18**	0.017	**V28**	0.007
V9	0.055	**V19**	0.009	**Amount_Tranfs**	0.008
V10	0.023	**V20**	0.009		

By looking at the feature importances it is possible to find out which variables had the most impact. In this case variables V4, V11, V12 and V14 were the most important and it would be very interesting to know the meaning and interpret the impact.

These results can be trusted since all metrics have a good value. Nevertheless it would be a good decision to train the model with more data since the percentage of fraud cases is very low and had to be synthetically manipulated in order to calculate a trustworthy model. This could also lead to overestimating the characteristics of fraud cases of the training data.

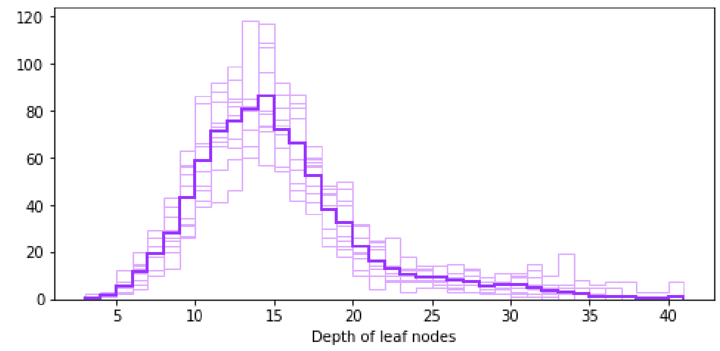
Finally I can conclude that the model is robust for the data that was given as input. Credit Card Fraud detection is a machine learning topic where the learning part has to take place continuously. When credit card fraud methods are discovered too quickly, criminals will find new ways to commit fraud. Those new ways can be seen in the data but the model has to be trained on new data as well. This makes it perfect for machine learning algorithms that learn and improve continuously.

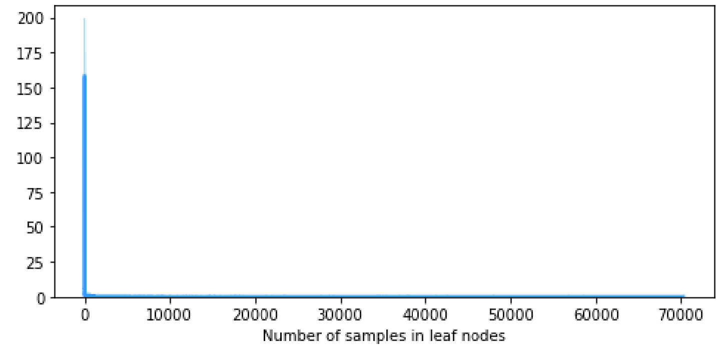
5. Conclusion and Reflection

The challenge of this analysis was to deal with the unbalanced data set. Therefore I tried two approaches: Undersampling and Oversampling. I used the rebalanced dataset only for training and the unbalanced dataset for testing. This resulted in a high number of false positives, since in the training set the fraud ratio was much higher than in the testing set. Therefore I tried a number of different models to find the one that could deal with it the best. Random Forest performed best compared to the logistic regression, SVM and Decision Tree.

A Random Forest Classifier combines several single Decision Trees. Therefore it is not useful to plot just one of those trees, as it would not represent the ensemble of trees. Also this could get very complex and huge since I used 29 variables.

Therefore I decided to plot just the depth of leaf nodes and the number of samples in a leaf for the ensemble of trees in the random forest classifier using the python implementation of Aysen Tatarinov (Tatarinov, 2015):





The graphic shows that there are trees that have a maximum depth of over 40 whereas most trees have a maximum depth of around 15 nodes. There are some trees that have leaf nodes with many samples, over 70.000. But most leaf nodes contain less than 1000 samples. Since the random forest is an ensemble classifier which is much like a black box it is really difficult to produce meaningful plots.

For me the fact that the dataset was so unbalanced was very challenging. I tried different models and I tried different combinations of test_train_split and the over/ undersampling but the models performed quite badly sometimes. In the first attempt I tried doing over/ undersampling first and then the test_train_split which resulted in really good models. But after validating on the whole dataset I found out that I was doing it wrong and that I should do it the other way round – first train_test and second over/ undersampling. Also I was shocked about the high numbers of false positives and I expected the logistic regression to perform way better. But it was only after I looked up on other models that were more robust on overfitting, that I tried the random forest and was very happy with the results.

The next thing to do would be to challenge the model with new, previously unseen data from the following bank working days. One problem of the model could be that the final validation was done on the same data as the training and testing of the model beforehand. Therefore the model could be improved by having more data available. Also it is crucial to implement a machine learning pipeline that recalculates the model with new data on a regular basis since criminals are quickly finding new ways to commit fraud.

Literature

Breimann, L., & Cutler, A. (n.d.). *Random Forests*. Retrieved 05 23, 2017, from https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Caelen, O., Dal Pozzolo, A., Johnson, R. A., & Bontempi, G. (2015). Calibrating Probability with Undersampling for Unbalanced Classification. *Symposium on Computational Intelligence and Data Mining (CIDM), IEEE*.

Dal Palozzo, A. (2015, 12). *Adaptive Machine Learning for.* Retrieved 04 30, 2017, from http://www.ulb.ac.be/di/map/adalpozz/pdf/Dalpozzolo2015PhD.pdf

Decision tree learning. (n.d.). Retrieved 05 20, 2017, from Wikipedia: https://en.wikipedia.org/wiki/Decision_tree_learning

Descoins, A. (2013, 03 25). *Tryolabs*. Retrieved 04 30, 2017, from https://tryolabs.com/blog/2013/03/25/why-accuracy-alone-bad-measure-classification-tasks-and-what-we-can-do-about-it/

European Central Bank. (2014, 02 25). *European Central Bank*. Retrieved 04 30, 2017, from https://www.ecb.europa.eu/press/pr/date/2014/html/pr140225.en.html

Kowalczyk, A. (2014, 11 02). *SVM - Understanding the math - Part 1 - The margin*. Retrieved 06 04, 2017, from SVM Tutorial: https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/

OracleHelpCenter. (n.d.). *Data Mining Concepts*. Retrieved 05 20, 2017, from Support Vector Machines: https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_svm.htm#CHDDJFDJ

OracleHelpCenter. (n.d.). *Support Vector Machines*. Retrieved 05 20, 2017, from Data Mining Concepts: https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_svm.htm#CHDDJFDJ

Rae, J., Krishnan, V., Kolluru, M., & et al. (n.d.). *Quora*. Retrieved 05 20, 2017, from What are the advantages of logistic regression over decision trees?: https://www.quora.com/What-are-the-advantages-of-logistic-regression-over-decision-trees

Richardson, M., & Domingos, P. (2006, 01 26). *Markov Logic Networks: Online Appendix*. Retrieved 04 30, 2017, from http://aiweb.cs.washington.edu/ai/mln/auc.html

Support Vector Machine. (n.d.). Retrieved 05 20, 2017, from Wikipedia: https://en.wikipedia.org/wiki/Support_vector_machine

Tatarinov, A. (2015, 11 08). *Visualization of Leaf Nodes in Random Forests*. Retrieved 06 04, 2017, from https://aysent.github.io/2015/11/08/random-forest-leaf-visualization.html

Whitehead, J. (n.d.). *An Introduction to Logistic Regression*. Retrieved 06 04, 2017, from Appalachian State University: http://www.appstate.edu/~whiteheadjc/service/logit/intro.htm