**Credit Card Fraud Detection**

**Introduction**

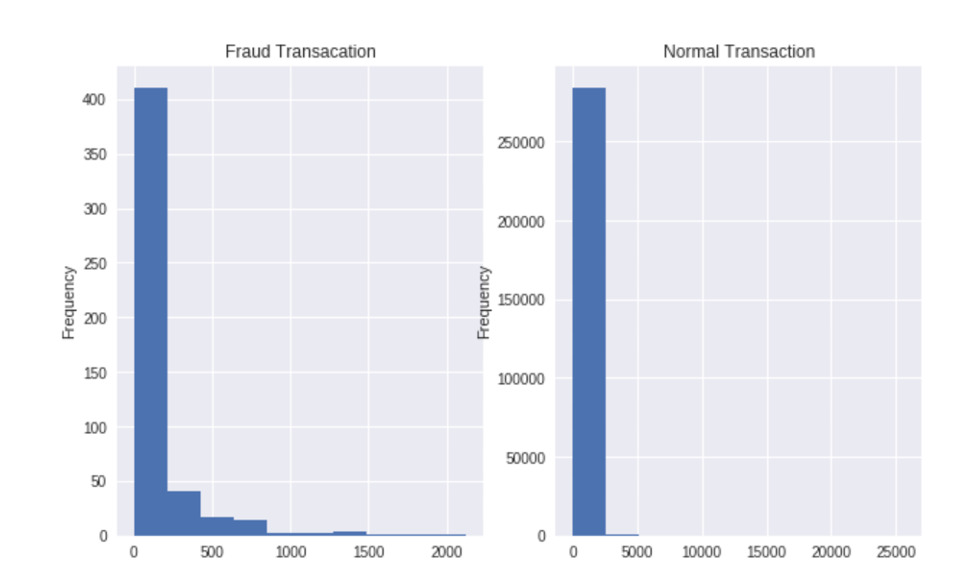
Credit card fraud is a significant issue and has considerable costs for banks and card issuer companies. Considering this massive problem, banks take credit card fraud very seriously, and have very advanced security systems to monitor transactions and detect the frauds as quickly as possible once they have occured. This measure is important in order to prevent customers from being charged for items they did not purchase.

In order to commit credit card fraud, hackers (frauders) are looking for sensitive information such as credit card numbers, bank accounts and social security numbers.

In case of offline payments, in order to perform fraudulent transactions, an attacker has to steal the credit card physically, while in the case of online payments (which occurs over the internet or phone), the fraudsters have to steal the owner's identity.

Fraud detection has become a crucial activity in order to decrease the impact of fraudulent transactions on service delivery, costs, and company reputation.

**Description of the data set**

Our data set “creditcard.csv” is a CSV file containing 31 features and over 284 807 samples. The data set consists of transactions made with credit cards in September 2013 by European cardholders. This data set was recorded over a period of two days, and 492 frauds have been detected out of 284 807 transactions. The data set is unbalanced, the positive class (frauds) accounts for 0.172% of all transactions.

Our data set contains only numeric input variables which are the results of a PCA transformation.

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

PCA is mostly used as a tool in exploratory data analysis.

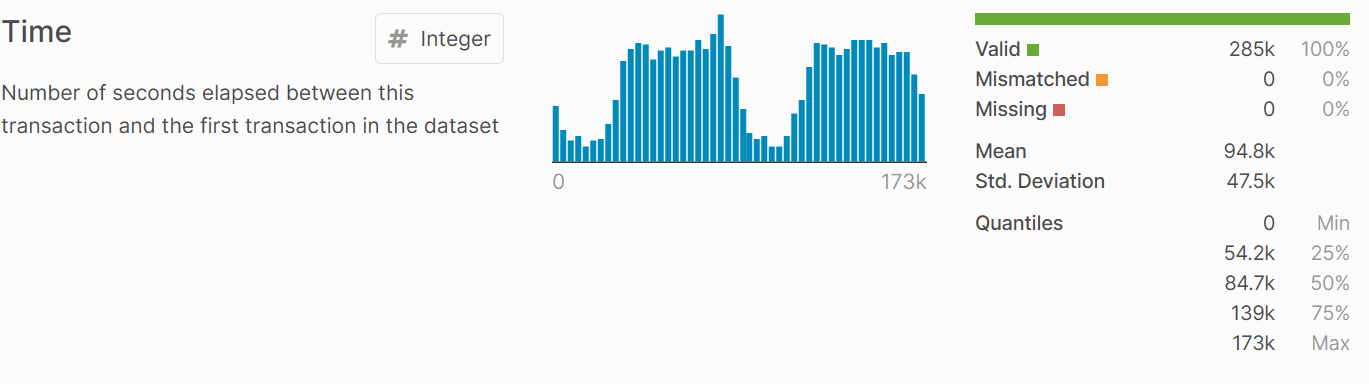
The results of a PCA are usually discussed in terms of component scores, sometimes called factor scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score).

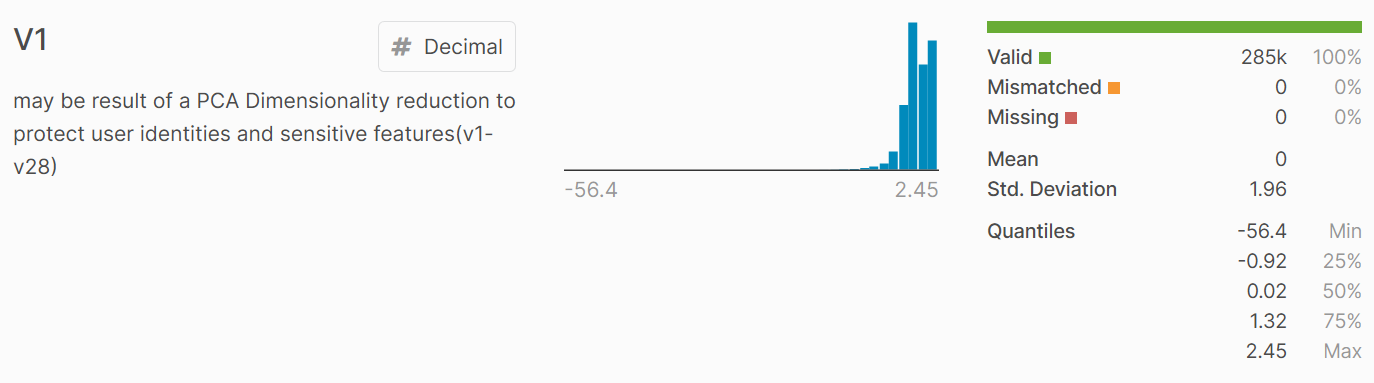
If component scores are standardized to unit variance, loadings must contain the data variance in them (and that is the magnitude of eigenvalues). If component scores are not standardized (therefore they contain the data variance) then loadings must be unit-scaled, ("normalized") and these weights are called eigenvectors; they are the cosines of orthogonal rotation of variables into principal components or back.

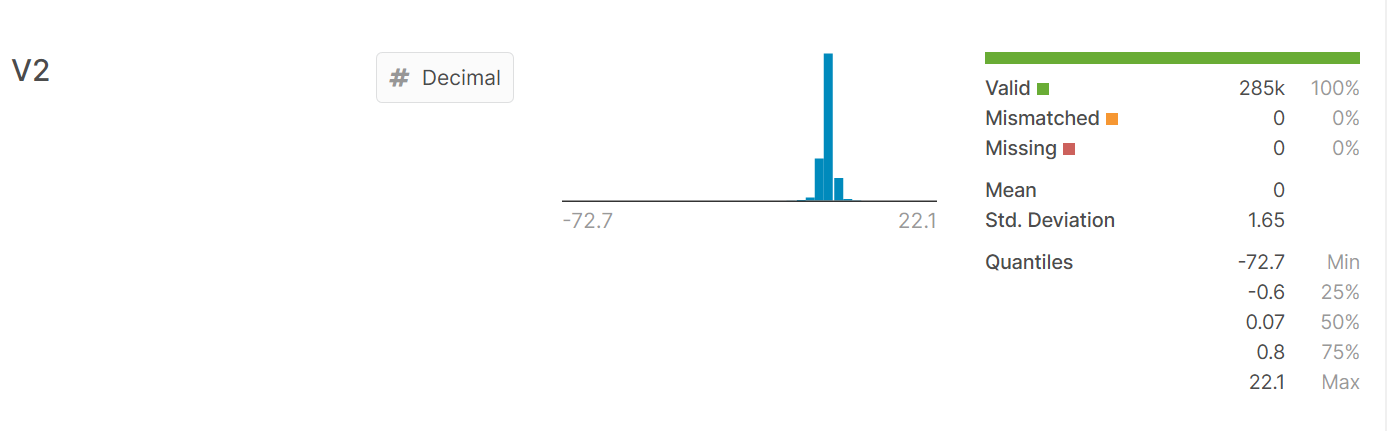
Features V1, V2, …, V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are “Time” and “Amount”.

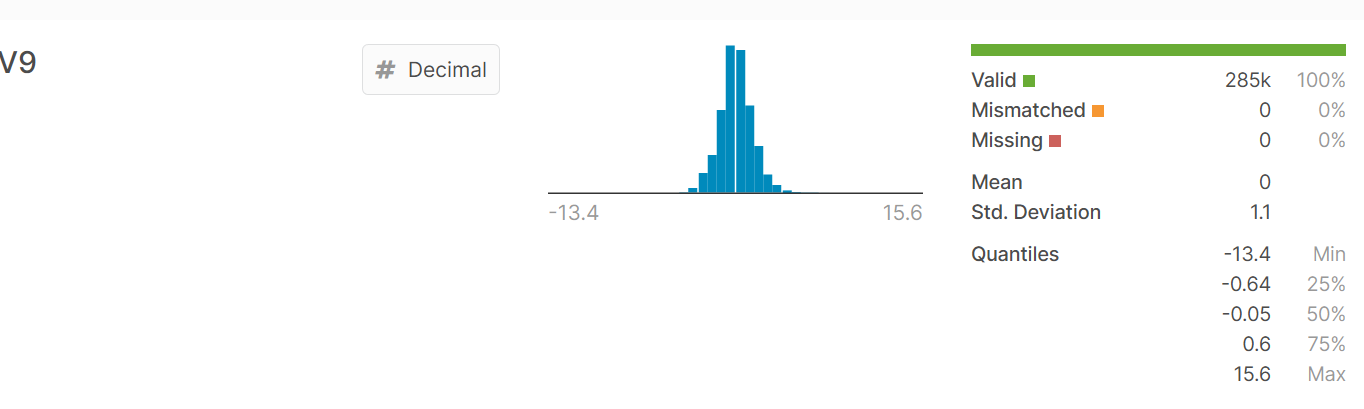
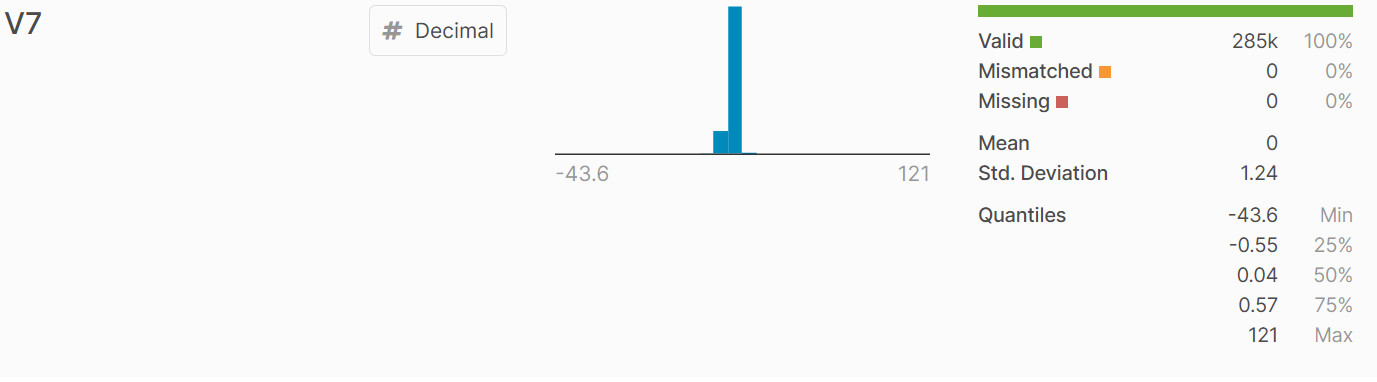
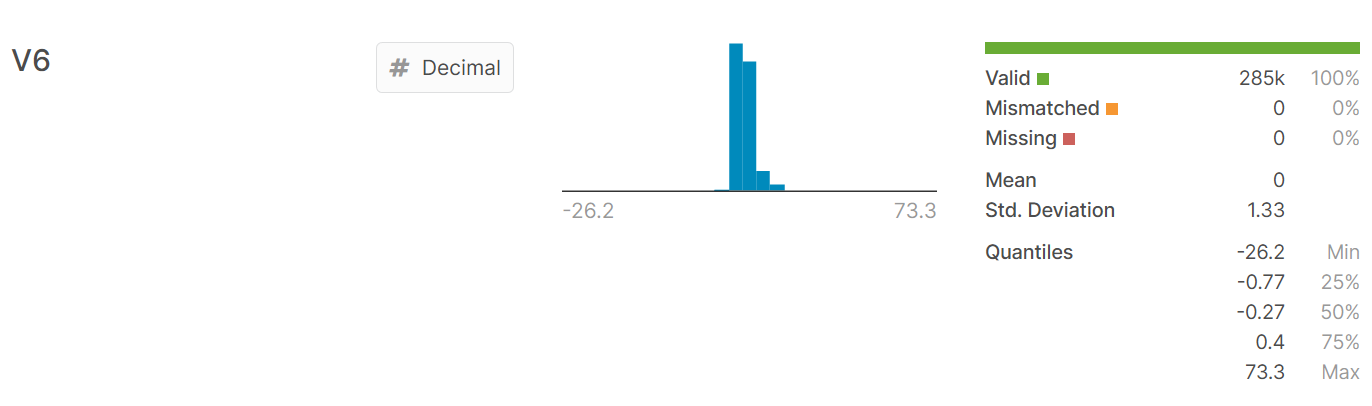
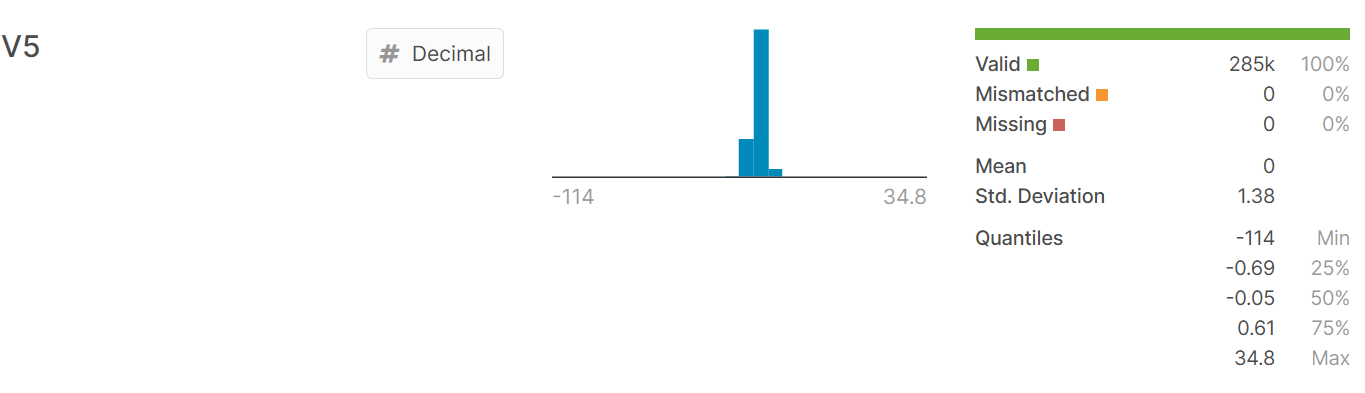
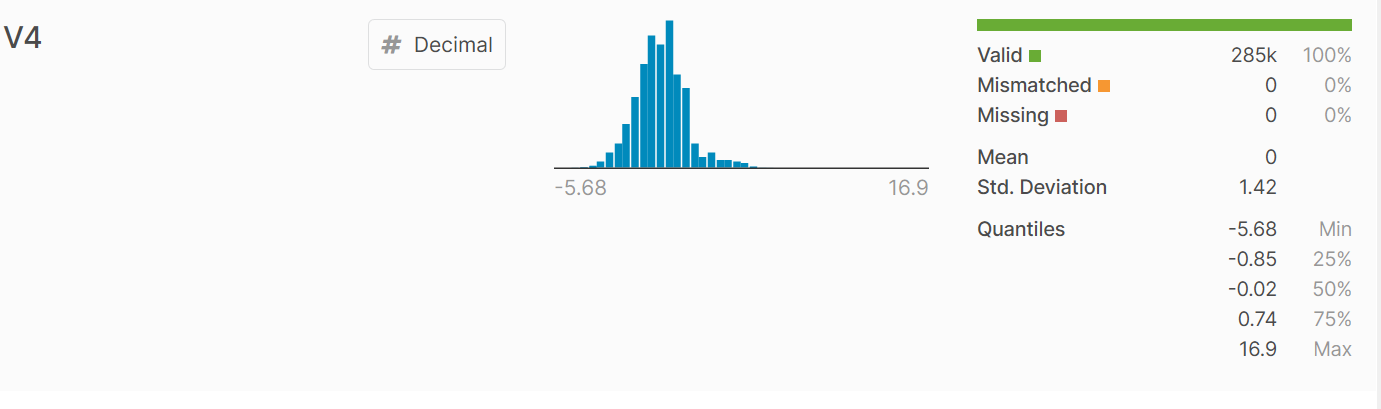
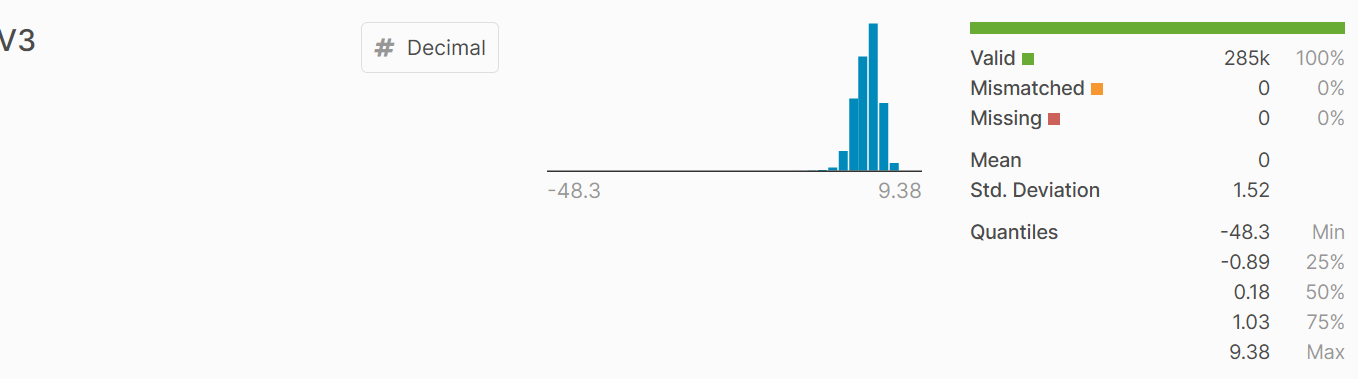
The “Time” feature contains the seconds elapsed between each transaction and the first transaction in the data set, and the “Amount” feature represents the transaction quantity, this feature can be used for example-dependant cost-sensitive learning. The “Class” feature is the response variable and it takes value 1 in case of fraud and 0 otherwise.

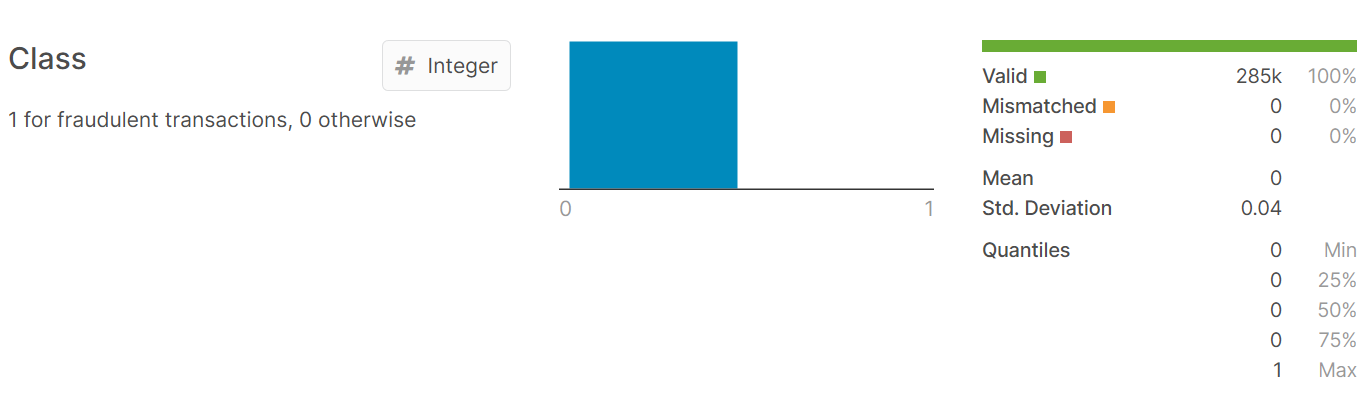
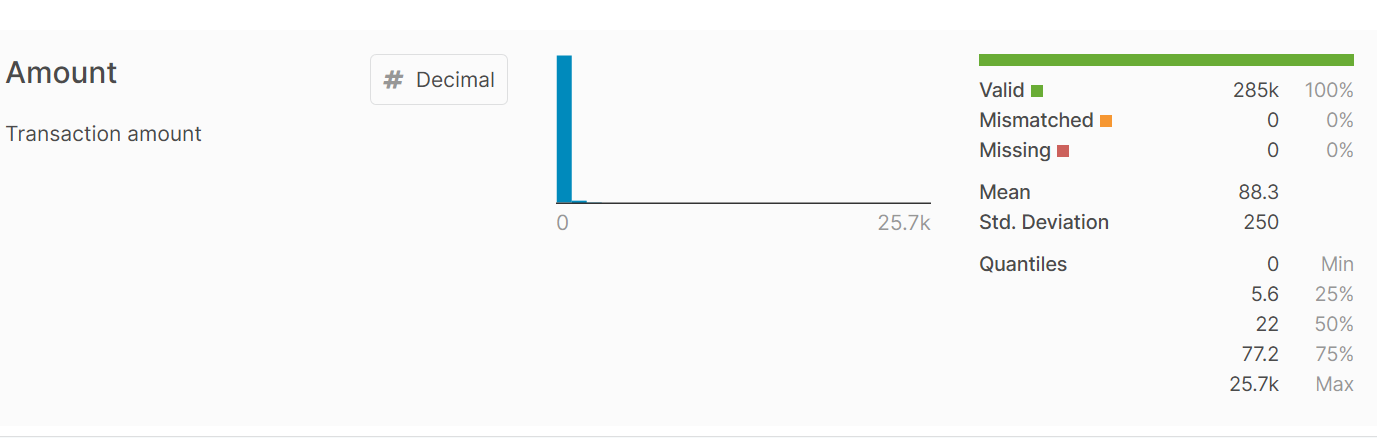
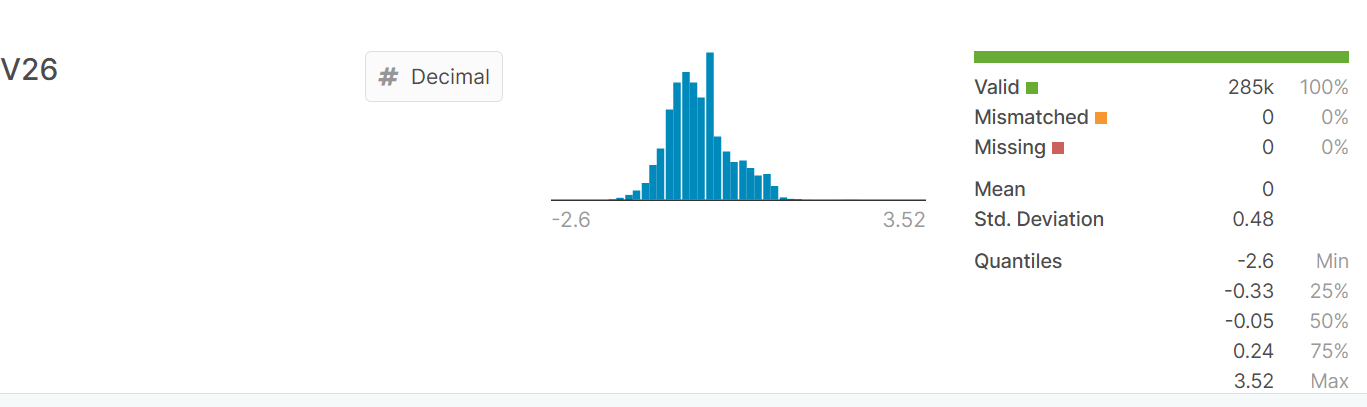
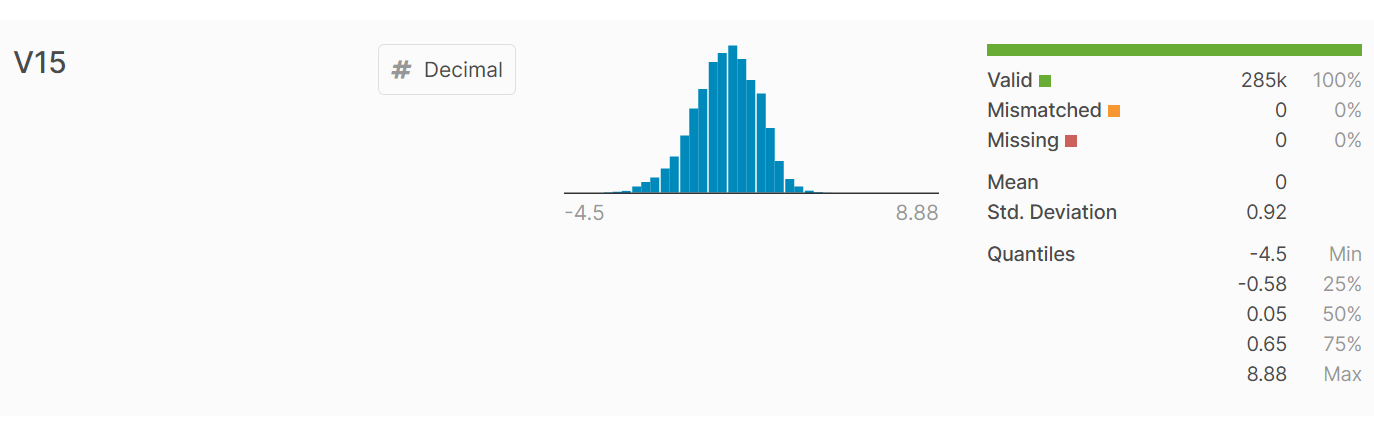
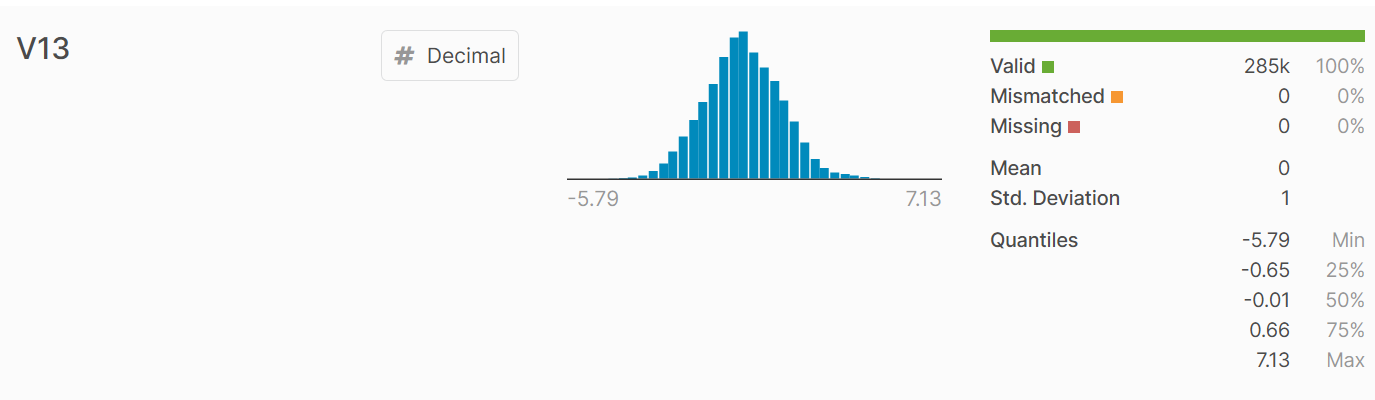
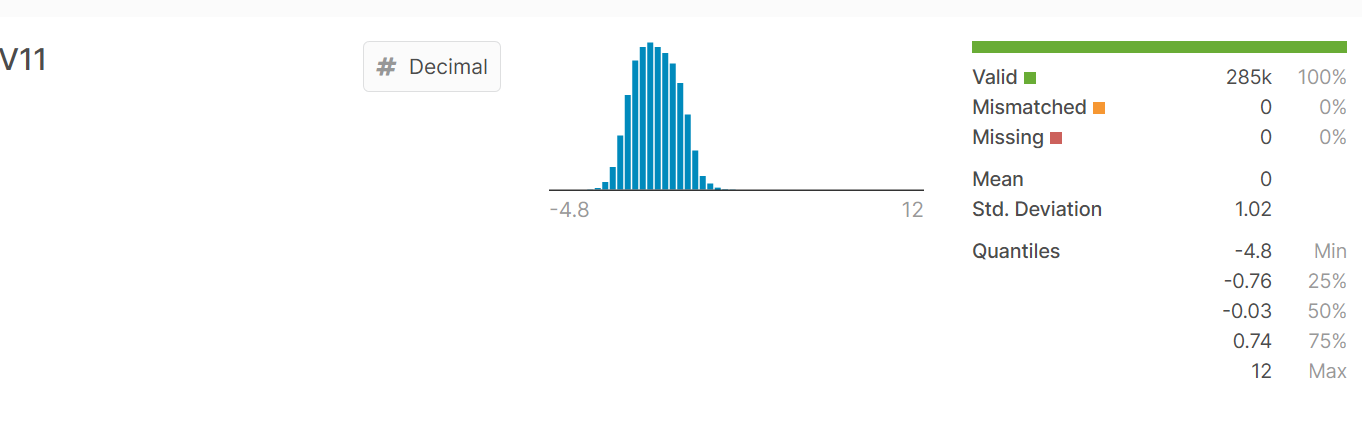
Some of the features can be seen below in a graphical way.











The data set can be found [here](https://www.kaggle.com/mlg-ulb/creditcardfraud). A very good tutorial regarding data set filtering can be found [here](https://sparkbyexamples.com/spark/working-with-spark-dataframe-filter/).

**Data set analysis**

The first steps of analysing the data set can be found [here](https://www.kaggle.com/mpanfil/credit-card-fraud-detection-first-steps). Before prediction we need to [clean the data](https://www.kaggle.com/abdilatifssou/analyse-and-cleaning-the-data-before-we-predict), after which, we [remove the duplicate rows](https://sparkbyexamples.com/spark/spark-remove-duplicate-rows/) and then deal with the imbalanced data following [this](https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets)tutorial.

**Summary:**

* The transaction amount is relatively small. The mean of all the transactions’ amounts made is approximately 88 USD.
* There are no "Null" values, so we don't have to replace any values.
* Most of the transactions were marked as Non-Fraud (99.83%), while the amount of transactions marked as Fraud was relatively small (0.17%).

**Feature Technicalities:**

* **PCA Transformation:** The description of the data says that almost all the features went through a PCA transformation (Dimensionality Reduction technique), except for *time* and *amount*.
* **Scaling:** Keep in mind that in order to implement a PCA transformation, the features need to be previously scaled. In this case, all the V features have been scaled, or at least, that is what we are assuming the people who developed the data set did.

**Scaling and Distributing**

In this phase, we will first scale the **Time** and **Amount** columns. They should be scaled like the other columns. We also need to create a subsample of the data frame in order to have an equal amount of Fraud and Non-Fraud cases, helping our algorithms better understand the patterns that determine where a transaction is a fraud or not.

**What is a subsample?**

In this scenario, our subsample will be a data frame with a 50-50 ratio of Fraud and Non-Fraud transactions. Meaning our subsample will have the same amount of Fraud and Non-Fraud transactions.

**Why do we create a subsample?**

In the beginning, we saw that the data frame was heavily imbalanced. Using the original data frame will cause the following issues:

* **Overfitting:** Our classification models will assume that in most cases there are no frauds! What we want for our model is to be certain when a fraud occurs.
* **Wrong Correlations:** Although we don't know what the "V" features stand for, it will be useful to understand how each of these features influence the result (Fraud or Non-Fraud), by having an imbalanced data frame we are not able to see the true correlations between the class and features.

**Summary:**

* **Scaled amount** and **scaled time** are the columns with scaled values.
* There are **492 cases** of fraud in our data set so we can randomly get 492 cases of non-fraud to create our new sub dataframe.
* We concat the 492 cases of fraud and non fraud, **creating a new sub-sample.**

**Splitting the data (Original data frame)**

Before proceeding with the **Random UnderSampling** technique, we have to separate the original data frame. Why? For testing purposes, remember, although we are splitting the data when implementing the **Random UnderSampling** or **OverSampling** techniques, we want to test our models on the original testing set, not on the testing set created by either of these two techniques. The main goal is to fit the model either with the data frames that were *under sampled*or *over sampled* (in order for our models to detect the patterns), and test it on the original testing set.

**Random UnderSampling**

Using this technique consists of removing data in order to have a more **balanced** data set, and thus avoiding *overfitting* our models.

**Steps**:

* The first thing we have to do is to determine how **imbalanced** is our class (use “value\_counts()” on the class column to determine the amount for each label)
* Once we determined how many instances are considered **Fraud** transactions (Fraud = “1”), we should bring the **Non-Fraud** transactions to the same amount as the **Fraud** transactions (assuming we want a 50-50 ratio), this will be equivalent to 492 cases of **Fraud** and 492 cases of **Non-Fraud** transactions.
* After implementing this technique, we have a subsample of our data frame with a 50-50 ratio regarding our classes. Then, the next step is to **shuffle** the data to see if our models can maintain a certain accuracy every time we run our script.

**Note:** The main issue with **Random UnderSampling** is that we run the risk that our classification models will not perform as accurate as we would like to, since there is a great deal of **information loss** (bringing 492 Non-Fraud transactions from 284,315 Non-Fraud transactions).

If we need to filter out non-null values we can do so by following the steps [here](https://stackoverflow.com/questions/39727742/how-to-filter-out-a-null-value-from-spark-dataframe).

**Linear Model or Linear Regression**

Linear regression is used to predict the value of an outcome variable Y based on one or more input predictor variables X. In this data set, as it can be seen above, we have one outcome, which is the Class that represents whether or not the transaction was a fraud, and thirty predictors which are amount, time, v1, v2, …, v28. The aim is to establish a linear relationship that can be written as a mathematical formula between the predictor variable, or in our case variables, and the response variable, so that we can use this formula to estimate the value of the response Y, in our case, the class, when only the predictor values are known.

As it was mentioned above, the relation can be written in a mathematical equation that can be generalized as follows:

Y =  β 0 + β 1 X1 + β 2 X2  +...+ β p Xp  + ϵ

where, Y and ϵ are random variables, X1, X2 ... Xp are not.

In this equation, Y is the response variable or the prediction, X variables are the exploratory variables (the predictors) and ϵ is the error term, residual.

In other words, a linear model draws a line in one or more hyperplanes, and if we plot a point on that graph, with the help of that line, we can predict all variables.

Scatter smoothing was used to check if any linear relationships exist between the dependent (response) variable and independent (predictor) variables. Because in this data set we have more dependent variables, we used scatter.smooth for every one of them to see the relationship between the *Class* variable and v1, v2, v3, v4, …, v28, *time* and *amount*. After that, we used the *cor* function to see the correlation between the *Class* variable and the others. As it was expected, there isn’t any high correlation between the *Class* and any variable.

After that, we built the linear model, but before using the regression model, we printed the summary statistics for the linear model.

In summary, we can see a lot of values:

* *Residuals*: The section summarizes the residuals, the error between the prediction of the model and the actual results. Smaller residuals are better.

Our model obtained the following residuals:

Min: -1.01593; 1Q=-0.00135; Median = - 0.00005; 3Q = 0.00065, Max = 1.00037

* *Coefficients*: For each variable and the intercept, a weight is produced and that weight has other attributes like the standard error, a t-test value and significance.
* *Estimate*: This is the weight given to the variable.
* *Standard Error*: Tells you how precise the estimate measured was. It’s really only useful for calculating the t-value.
* *t-value and Pr(>[t])*: The t-value is calculated by taking the coefficient divided by the Std. Error. It is then used to test whether or not the coefficient is significantly different from zero. If it isn’t significant, then the coefficient really isn’t adding anything to the model and could be dropped or investigated further. Pr(>|t|) is the significance level.

Performance Measures: Three sets of measurements are provided.

* *Residual Standard Error*: This is the standard deviation of the Residuals. Smaller is always better because we want the error to be as small as possible. It obtained 0.02697 in 85412 degrees of freedom.
* *Multiple / Adjusted R-Square*: R-squared shows the amount of variance explained by the model. Adjusted R-Square takes into account the number of variables. The value obtained is 0.506
* *F-Statistic*: The F-test checks if at least one variable’s weight is significantly different than zero. This is a global test to help assess a model. If the p-value is not significant (e.g. greater than 0.05) than the model is essentially not doing anything. It obtained 2917 on 30 and 85412 DF.

**AIC and BIC**

The Akaike’s information criterion - AIC and the Bayesian information criterion - BIC are measures of the goodness of fit of an estimated statistical model and can also be used for model selection. Both criteria depend on the maximized value of the likelihood function L for the estimated model.

The AIC is defined as:

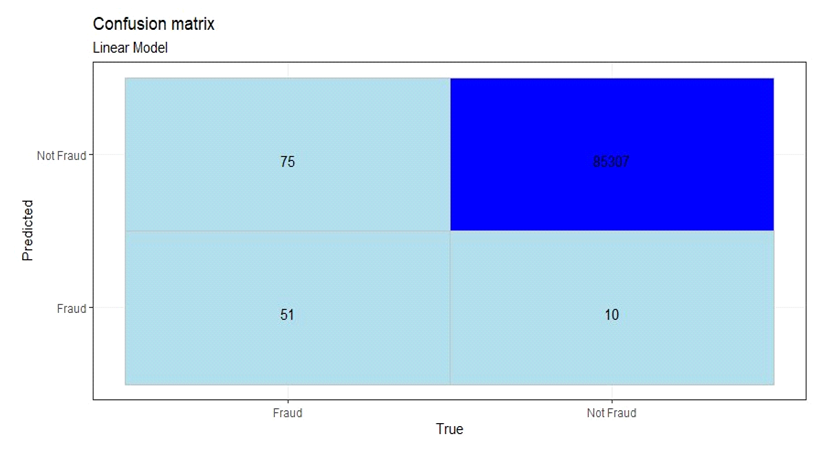
*AIC* = (−2) × *ln*(*L*) + (2×*k*)

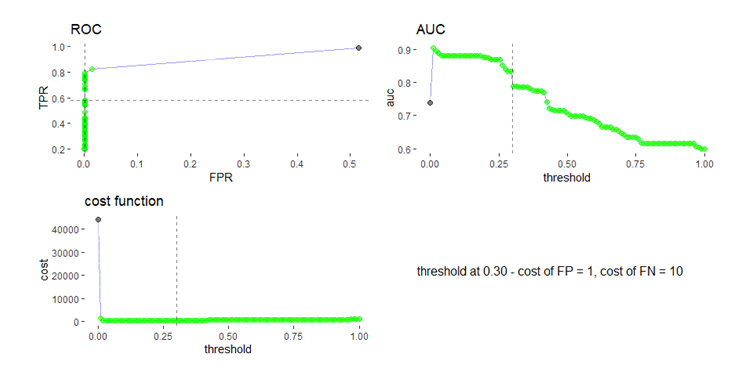
where, k is the number of model parameters, and the BIC is defined as:

*BIC* = (−2) × *ln*(*L*) + *k* × *ln*(*n*)

where n is the sample size. For model comparison, the model with the lowest AIC and BIC score is preferred. Our model scored -374878.4 with AIC and -374579 with BIC.

After analysing our data and after the prediction, we obtained the following confusion matrix:

****

As we already know, the confusion matrix is not that relevant in this data set, but the AUC value is. The maximum AUC value obtained by the linear model is 0.90, which is decent, but it will lose compared to the other models presented.

Those are the *Roc*, *Auc* and *cost function* plotted in order to better analyse our linear model.

**Decisional Tree**

Decision Tree Analysis is a general, predictive modelling tool that has applications spanning a number of different areas. In general, decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions.It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

There are two types of decision trees: *Regression trees* and *Classification trees*.

*Main idea*:segmenting the predictor space into a number of sample regions.

Prediction = ‘mean’ of the training observations corresponding to one region.

When talking about classification trees, the main difference in contrast with *Regression trees*: we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

We may use the recursive binary splitting to build classification trees, but we cannot use RSS as a performance criteria.

**Implementing a decision tree**

We are using the creditcard.csv data set to build a decision tree classifier.

The data set contains information about two classes of credit card data with the following attributes:

1. 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'.
2. The 'Time' feature contains the seconds elapsed between each transaction and the first transaction in the data set.
3. The 'Amount' feature is the transaction’s amount, this feature can be used for example-dependent cost-sensitive learning.

The 'Class' feature is the response variable and it takes value 1 in case of fraud and 0 otherwise.

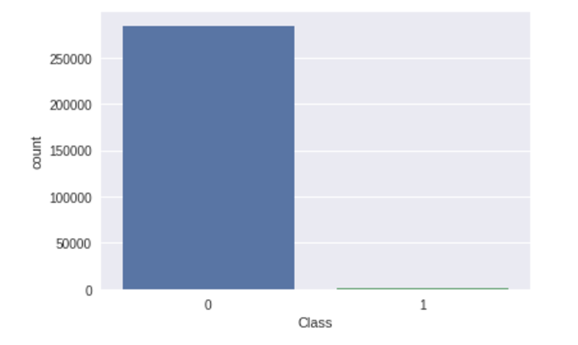
The task is to predict the class of credit card based on the attributes.



• Hence we can see there are 284 807 rows and 31 columns which is a huge data set

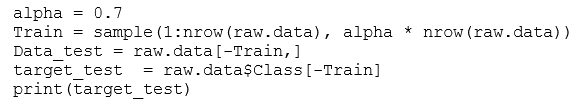
• Time is also a float here meaning it can only be seconds starting from a particular time

Let’s check the class distribution:

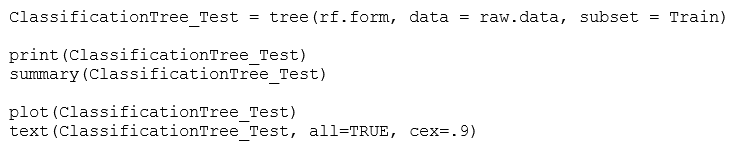


1. Only 0.17% are fraud transactions while 99.83% are valid transactions.
2. Preparing data for training and testing as we are going to use different data.

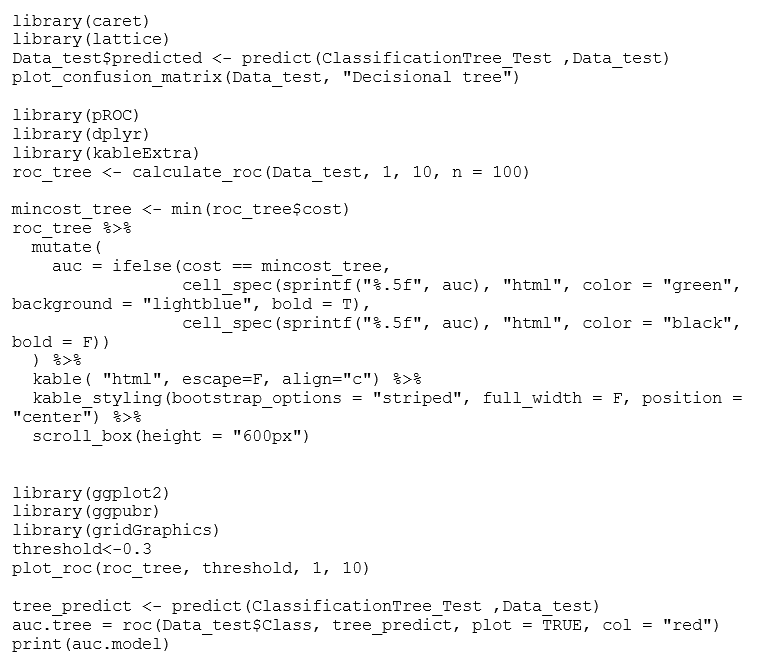
Now that we have extracted the data attributes and corresponding labels, we will split them to form train and test data sets.



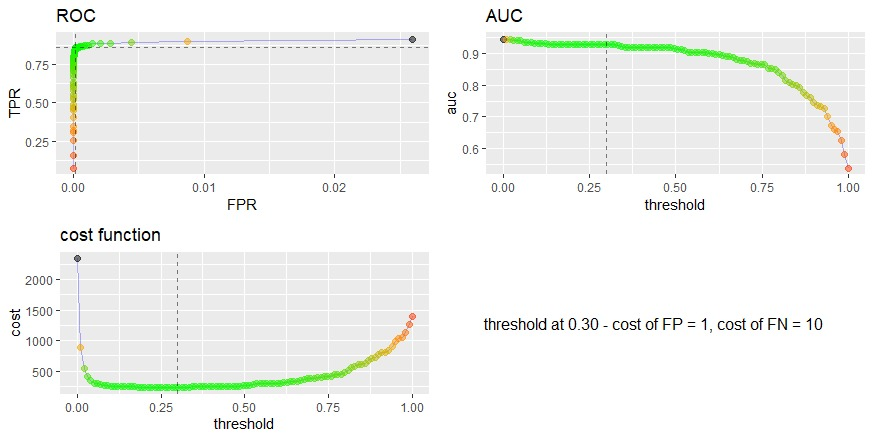
Next, we will fit the classifier on the train attributes and labels.



We will make a model function for modeling with a confusion matrix.



We concluded that our model has an accuracy of 0.91 which is better than the linear model.



**Issues in decision trees**

* *Avoiding overfitting*

It is prone to creating decision trees that overfit by performing really well on the training data at the expense of accuracy with respect to the entire distribution of data.

There are, in general, two approaches to avoid this in decision trees:

* Allow the tree to grow until it overfits and then prune it.
* Prevent the tree from growing too deep by stopping it before it perfectly classifies the training data

A decision tree’s growth is specified in terms of the number of layers, or depth, it’s allowed to have. The data available to train the decision tree is split into training and testing data and then trees of various sizes are created with the help of the training data and tested on the test data. Cross-validation can also be used as part of this approach. Pruning the tree, on the other hand, involves testing the original tree against pruned versions of it. Leaf nodes are removed from the tree as long as the pruned tree performs better on the test data than the larger tree.

* *Incorporating continuous valued attributes*

One way to make this algorithm more useful with continuous variables is to turn them, in a way, into discrete variables.

* *Alternative measures for selecting attributes*

One way to avoid this is to use some other measure to find the best attribute instead of information gain. An alternative measure to information gain is *gain ratio* (Quinlan 1986).

*Gain ratio* tries to correct the information gain’s bias towards attributes with many possible values by adding a denominator to *information gain* called *split information*. *Split Information* tries to measure how broadly and uniformly the attribute splits the data.

**Advantages and Disadvantages**

Following are the advantages of decision trees:

* Easy to use and understand.
* Can handle both categorical and numerical data.
* Resistant to outliers, hence require little data preprocessing.
* New features can be easily added.
* Can be used to build larger classifiers by using ensemble methods.

Following are the disadvantages of decision trees:

* Prone to overfitting.
* Require some kind of measurement as to how well they are doing.
* Need to be careful with parameter tuning.
* Can create biased learned trees if some classes dominate.

**Random Forest**

Random forests or random decision forests are an ensemble of learning methods for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct the decision trees' habit of overfitting to their training set.

Random Forests grow many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (from all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

In the original paper on random forests, it was shown that the forest error rate depends on two things:

1. The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
2. The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob(out of bag) error rate a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests are somewhat sensitive.

Features of Random Forests:

* It is unexcelled in accuracy among current algorithms.
* It runs efficiently on large data bases.
* It can handle thousands of input variables without variable deletion.
* It gives estimates of what variables are important in the classification.
* It generates an internal unbiased estimate of the generalization error as the forest building progresses.
* It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data is missing.
* It has methods for balancing errors in class population unbalanced data sets.
* Generated forests can be saved for future use on other data.
* Prototypes that give information about the relation between the variables and the classification are computed.
* It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
* The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
* It offers an experimental method for detecting variable interactions.

Remarks:

Random forests do not overfit. You can run as many trees as you want. It is fast. Running on a data set with 50,000 cases and 100 variables, it produced 100 trees in 11 minutes on an 800Mhz machine. For large data sets the major memory requirement is the storage of the data itself, and three integer arrays with the same dimensions as the data. If proximities are calculated, storage requirements grow as the number of cases times the number of trees.

**Decision Tree**

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

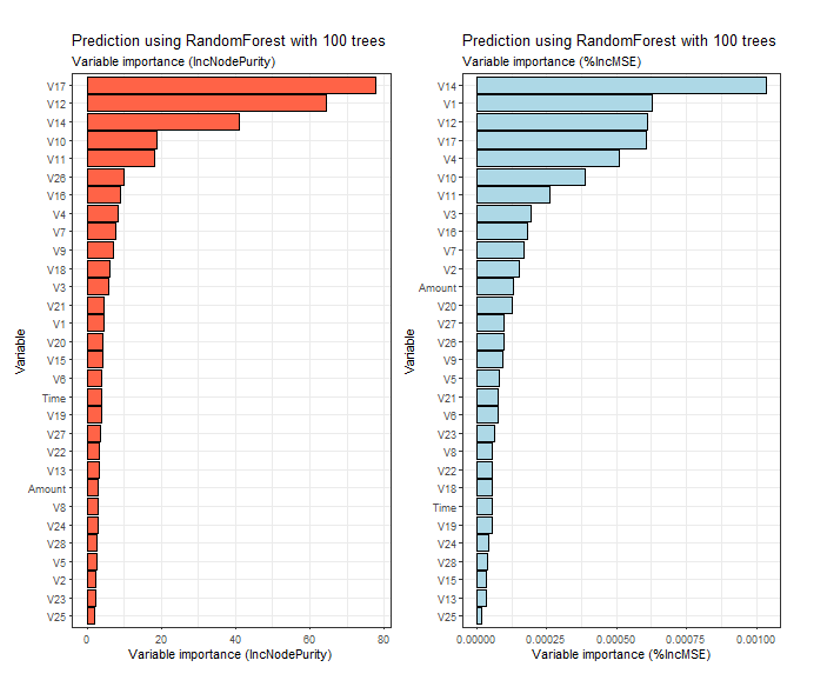
Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction (see figure below).

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science, the reason that the random forest model works so well is:

* A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.
* The low correlation between models is the key. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don’t constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

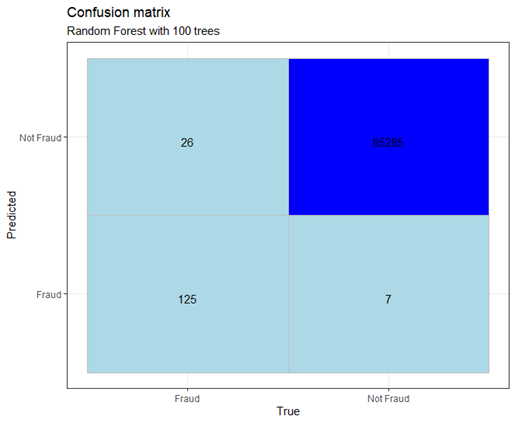
The way we used random forests in our data set was pretty straight forward. There are totally 31 columns in the data. One column, Class is the target value, it is a binary value, can have either 0 (not fraud) or 1 (fraud) value. Another two columns have a clear meaning: Amount is the amount of the transaction; Time is the time of the transaction. The rest of the features (28), anonymized, are named from V1 to V28. The data is highly unbalanced with respect to Class variable values. There are only 0.1727486% of the rows with value Class = 1. Typically, in such cases, we can either choose to preserve the data unbalancing or use an oversampling (of the data with minority value of target variable) or undersampling (of the data with the majority value of the target variable). Here we will just preserve the unbalancing of the data. In terms of validating the result, we will see that the usual matrix, using a confusion matrix or accuracy, is not the most relevant and we will prefer an alternative solution using AUC.

After we split the data in a training and test set, we create the RF model using the training set. We trained the model and plotted the importance of predictors based on MSE and Node Impurity.

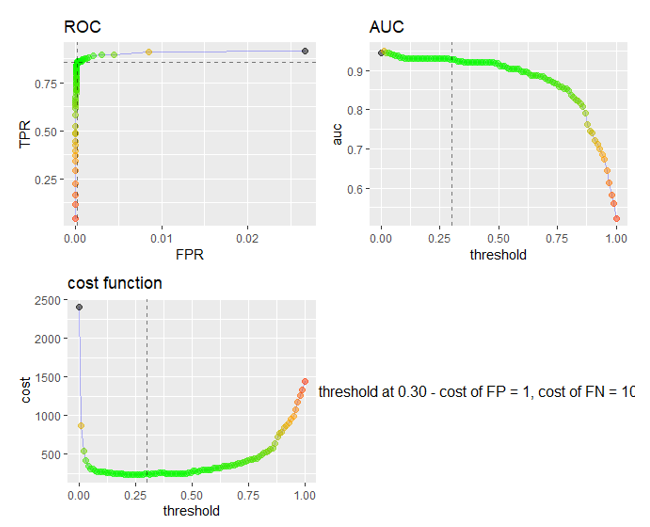


We can observe the difference between the two ways of calculating variable importance and also that the graphs are pretty similar even if the values may not be the same. Also we decided to use all the variables from the data set since they’re not that many and we obtained a good score while using all of them.

After this, we set the threshold at 0.5 and predict for the test case. We got this confusion matrix.



Let’s calculate the TP, FP, TN, FN, ROC, AUC and cost for the threshold with values between 0 and 1 (100 values equally distributed) and cost 1 for TN and 10 for FN. After this observation and also a look at the ROC graph we picked the value of threshold as 0.3.



The value obtained (0.93) is relatively good, considering that we did not perform any tuning, working with default RF algorithm parameters.

**Boosting Algorithm**

In this project we implemented GBM (Gradient Boosting Machine) in order to compare it with the other algorithms such as: Random Forest, Decisional Tree and Linear Model.

The Boosting algorithm learns slowly because it improves its prediction results by creating and training a number of weak models, learning from mistakes that every bad model has made. In other words, we fit a tree using the current residual values, not with the results. Then we add a new decision tree into the fitting function so we can update the residual values and obtain a better result in the end.

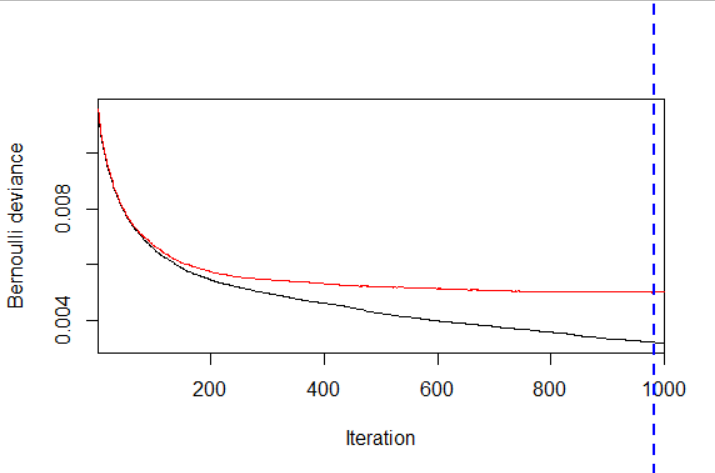
We considered a number of B generated bootstrapped training data sets from our Credit Card Fraud data set, where B is the number of trees we used. We computed B models as , , …, and then we made an “average” in order to obtain a single low variance machine learning model, that is:



In comparison with Random Forest, Boosting can overfit if the number of trees is too large.

GBM trains as many models as the Boosting Algorithm, but in a gradual, additive and sequential manner. The loss function indicates how good our models are. For example, we are trying to predict if a credit card transaction is fraudulent by using a regression, then the loss function would consist of an error between the true class and the predicted one. One of the biggest advantages of working with GBM is that it allows optimization on cost function, instead of a loss function. This gives more control and establishes a better correspondence with real world applications.

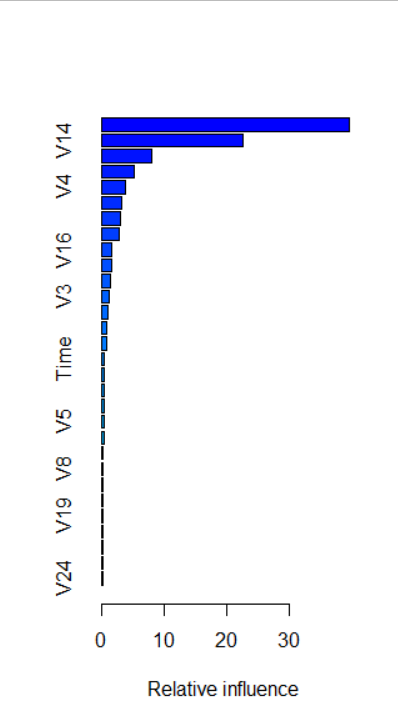
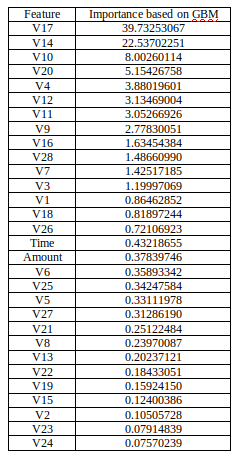
In order to train a GBM model in R, we installed the “gbm” library and used it. Then we started tuning the parameters. We used the “Bernoulli” distribution because we have to deal with a logistic regression for 0-1 outcome. We specified the data set and the number of trees (1000). In the next step, we specified the interaction depth which represents the maximum depth of each tree (6) and the learning rate known as shrinkage (0.02). By doing so, we reduced the size of incremental steps and it penalizes each consecutive iteration.



As it can be seen from the plot above, based on the test data, the best iteration is the 1000th. We also observed that some features are almost irrelevant based on this model. These results can be seen in the table below.

With a model that had 500 trees and a depth of 3, there were around six features that had an importance equaled with zero. In order to use all the features, we developed this model.

The plot below shows all the features’ influence in creating the model. The most influential ones are V17 and V14, and the less important ones are V23 and V24.

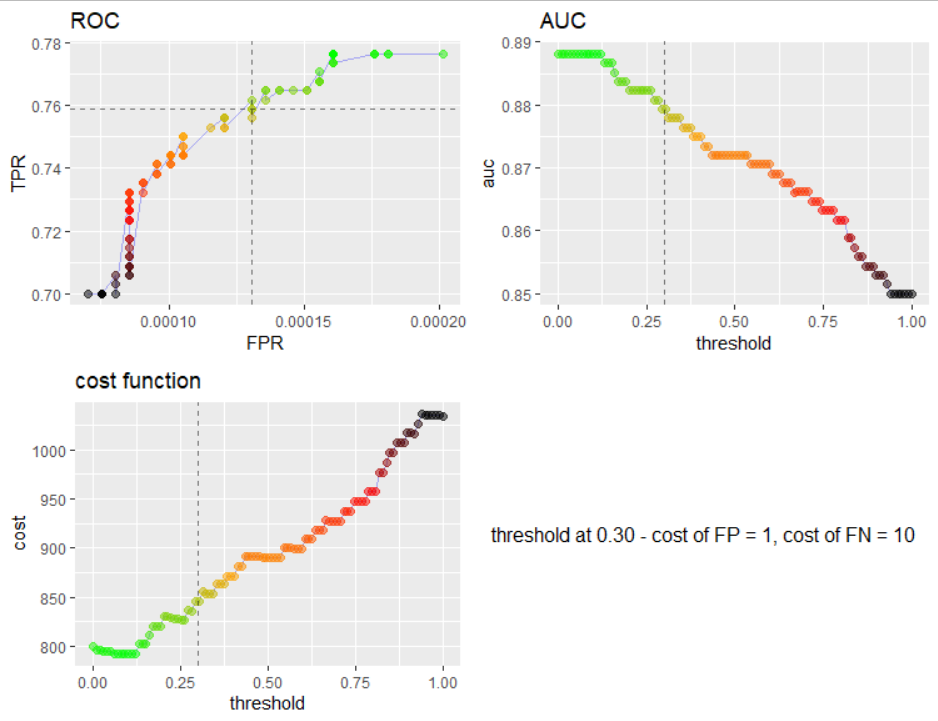


We also created a model with 500 trees and over 1000, but the results weren’t as good as these are. With around 3000 trees the training time was around 3 hours and the accuracy was a bit lower (91) and also the ROC curve had many ups and downs. We concluded that at such a big number, our model overfitted and the results aren’t concluded. This method is predisposed at overfitting.

We tested a model with a large number of trees and a little depth (under 3) and it resulted in bad performance and it overfitted. We had the same bad result with a model with a small number of trees (300) and a large depth (10) and it also overfitted.

With 500 trees and a depth of 3, we obtained a better accuracy (89), but also the ROC curve wasn’t ok. Also, there were around six features that had an importance equaled with zero. In order to use all the features, we developed this model.

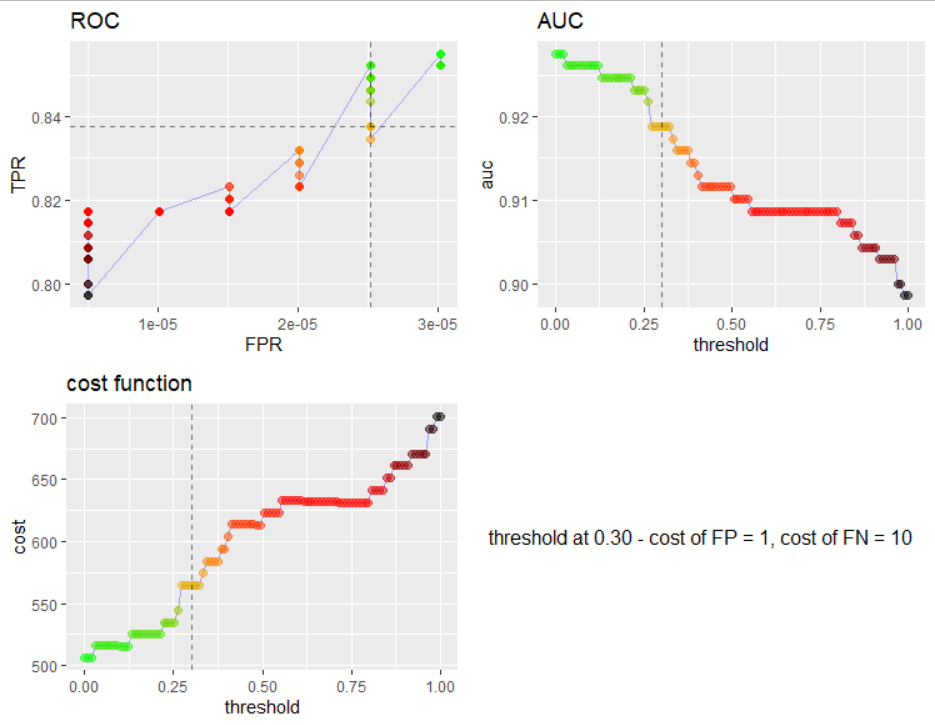
In the pictures below we can see the difference between a model trained on 500 trees and a depth of 3 and the one trained on 1000 trees and a depth of 6 in order to compare and understand them much better.



Picture of a model trained on 500 trees and a depth of 3 (ROC, AUC, cost function)

The accuracy obtained using the first model (500 trees and a depth of 3) was 89, but the one resulted from the best model trained by us (1000 trees and a depth of 6) was 94.

Talking about time, we needed around a half an hour to create the final model and around five minutes to make the first model because of the large number of trees and the bigger depth.



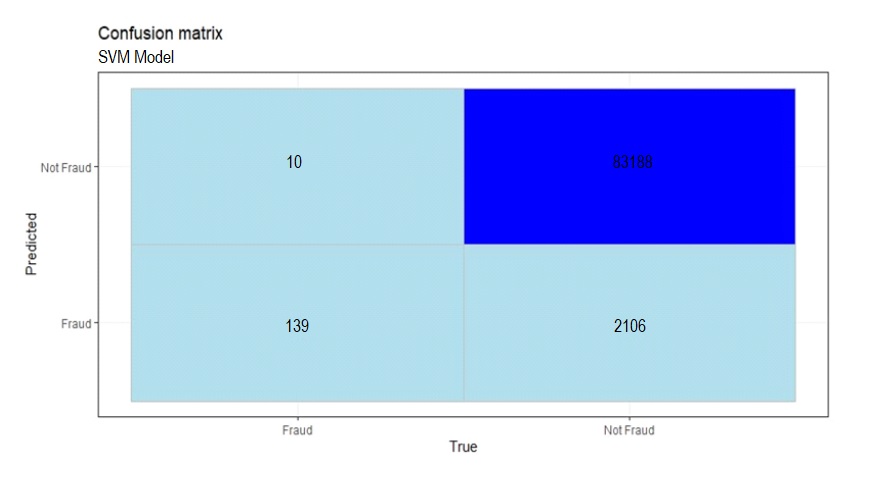
Picture of a model trained on 1000 trees and a depth of 6 (ROC, AUC, cost function)

**SVM**

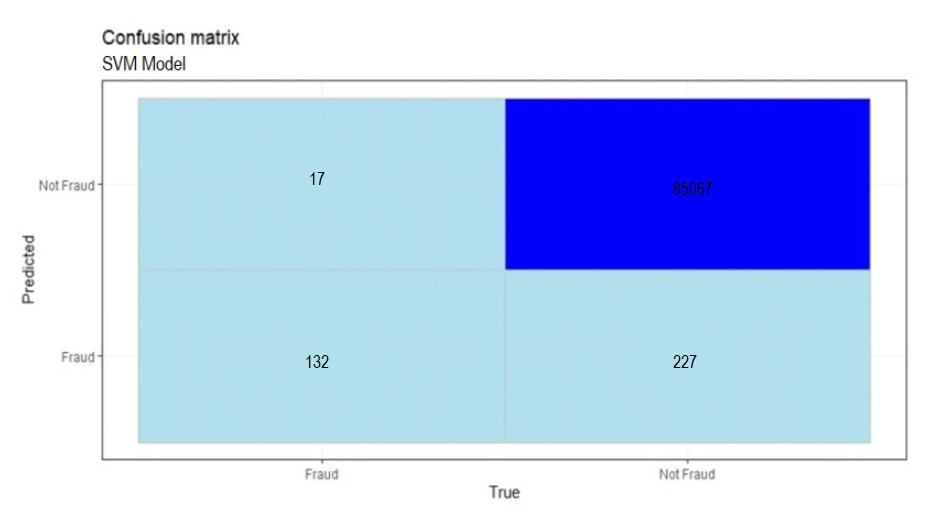
SVM is a supervised learning method based on a specific kernel chosen between: linear, polynomial and RBF (Radial Basis Function). This method is memory efficient because it uses a subset of training points in the decision function.

In our case, we needed to use class weights because we have unbalanced data. First, we trained a SVM model with a linear kernel. We obtained an impressive AUC value of 0.954 and the next confusion matrix. It can be easily observed that despite this good accuracy score, our model makes too many mistakes in detecting false positives and false negatives.

In order to obtain this data, we needed to use weighted classes. These have been calculated by and and resulted in 0.0003517226 for fraud class and 0.2032520325 for not-fraud class.



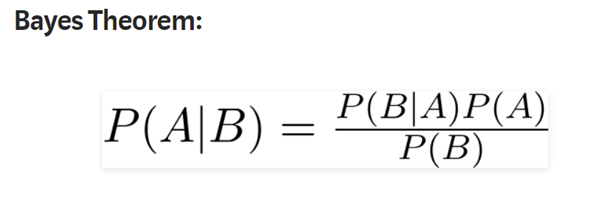
We also trained the SVM model with a polynomial kernel and we obtained the next results. The AUC value for this model is 0.942.



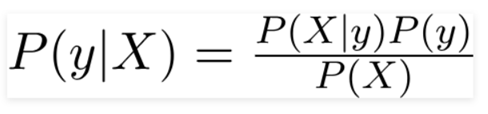
**Naive Bayes**

Naive Bayes is a Supervised Machine Learning algorithm based on the Bayes Theorem that is used to solve classification problems by following a probabilistic approach. It is based on the idea that the predictor variables in a Machine Learning model are independent of each other. Meaning that the outcome of a model depends on a set of independent variables that have nothing to do with each other. In real-world problems, predictor variables aren’t always independent of each other, there are always some correlations between them. Since Naive Bayes considers each predictor variable to be independent of any other variable in the model, it is called ‘Naive’.

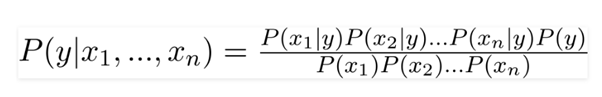
The *Bayes Theorem* is:



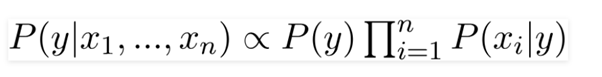
we can rewrite the formula as:



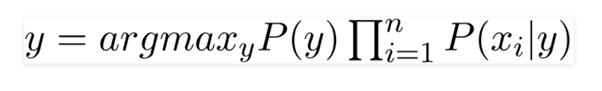
The variable y is the class variable, which represents if it is a fraud or not given the conditions. Variable X represents the parameters/features. Here x\_1, x\_2…., x\_n represent the features. By substituting for X and expanding using the chain rule we get:



Now, we can obtain the values for each by looking at the data set and substituting them into the equation. For all entries in the data set, the denominator does not change, it remains static. Therefore, the denominator can be removed and a proportionality can be introduced.

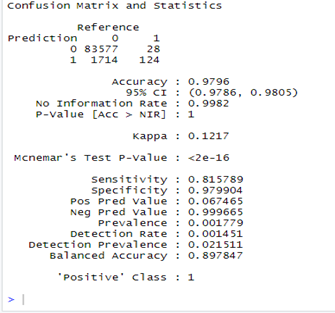


In our case, the class variable y has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class variable y with maximum probability.



Using the above function, we can obtain the class, given the predictors.

**Laplace Smoothing**

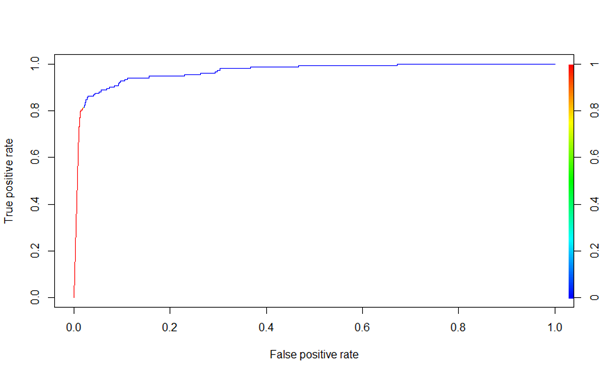
In statistics, Laplace Smoothing is a technique to smooth categorical data. Laplace Smoothing is introduced to solve the problem of zero probability. We’ve added Laplace smoothing to our algorithm.

**Naive Bayes** **in our project**

We splitted the set in a 70/30 ratio for training and testing. On the left, we can see the Confusion matrix resulted after training and predicting outcomes.

Since our data set is so unbalanced, the algorithm had a high accuracy from the beginning. So we calculated the ROC and AUC scores to check if our algorithm works well.

Here we have the ROC plot, which starts to curve around 0.8, but we can see that it becomes flat around 0.95. The AUC score was 0.96 which is great

.****

**Neural network**

After we tested all the models, the project didn’t feel complete without also implementing a neural network. What can have a better accuracy than a neural network? Deep learning should always perform better than any other model, shouldn’t it? That’s what we wanted to test. Did the neural network stand up to the expectations or did it fail?

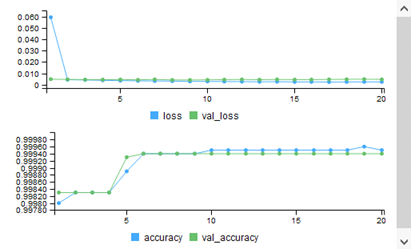
As a brief introduction to the neural network, a neural network is a supervised learning technique that copies the human brain, it is made of connected perceptrons. We have an input layer with a cell for all the features of the data set, some hidden layers with perceptrons where the thinking process takes place and an output layer where the final result is found. The basic rules were applied in order to build the best architecture, and by basic rules, we’re talking about how we chose:

* the number of hidden layers;
* the number of neurons on each layer;
* the activation and optimization functions.

In the first implementation, the time variable was removed because we didn’t get any meaningful values from the time data.

In the implementation, we used the Keras library because we considered it to be the best way to implement a neural network alongside TensorFlow. The neural network has 2 hidden layers, with a dropout layer for both of them in order to prevent overfitting. The best activation functions were *ReLU* for the first two layers and *Sigmoid* for the output layer because we have a binary classification.

The data was split as follows, 0.7 for training and 0.3 for validation, with a batch size of 100, and with just 20 epochs. The training results can be seen in the image below:



We see that it had an accuracy of ~99.995%, but as we already know, this is not important in this data set. We are comparing our models by the AUC value, and this neural network scored 0.991, which is the best value out of all the models we used.

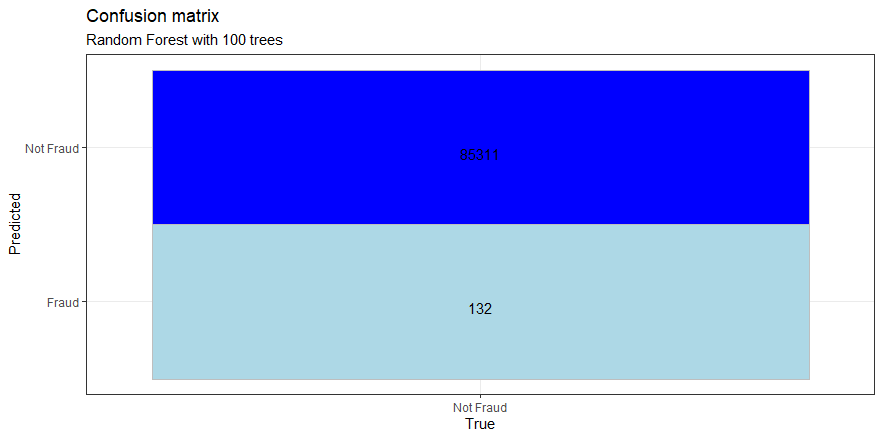
**Conclusion**

So, we used seven methods to predict whether or not a bank transaction is a fraud. Are these methods useful for something, or the results aren’t relevant at all?

In order to answer that question, we simply considered all bank transactions as legit. We changed the Class value of the data set to be 0 at all rows.

The new question was, if one can say that all transactions are legit, which mean that there are no frauds at all, does this prediction score better than our algorithms? If the answer is yes, then our algorithms are pretty much useless at predicting.

We made a confusion matrix for this prediction, and after we saw the results, we discovered that our algorithms scored better than this simple prediction. So, we can conclude that our algorithms can be useful in predicting frauds in those kinds of transactions.

****

Another thing that we concluded is that, with those types of data sets where the data we are trying to predict is very unbalanced, the accuracy is not that relevant. For such a problem, where the number of TP is very small in comparison with the number of TN, the Confusion Matrix is less useful, since it is important to use a metric that includes evaluation of FP and FN as well. It is important to minimize as much as possible the number of FN (Predicted: Not Fraud and True: Fraud) since their cost could be very large. Typically, AUC is used for such cases.

The calculated accuracy is not very relevant in the conditions where there is a very large unbalance between the number of fraud and non-fraud events in the data set. In such cases, we can see a very large accuracy. More relevant is the value of ROC-AUC. After we saw the value of ROC-AUC on all the models, we can conclude that using a neural network yields the best score, followed by Naive Bayes, SVM, Boosting, Random forest then Decisional tree and Linear model in this order.

**Bibliography**

[1] <https://en.wikipedia.org/wiki/Principal_component_analysis>

[2] <https://www.kaggle.com/mlg-ulb/creditcardfraud>

[3] <https://sparkbyexamples.com/spark/working-with-spark-dataframe-filter/>

[4] <https://www.kaggle.com/mpanfil/credit-card-fraud-detection-first-steps>

[5] <https://www.kaggle.com/abdilatifssou/analyse-and-cleaning-the-data-before-we-predict>

[6] <https://sparkbyexamples.com/spark/spark-remove-duplicate-rows/>

[7] <https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets>

[8] <https://stackoverflow.com/questions/39727742/how-to-filter-out-a-null-value-from-spark-dataframe>

[9] *“Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions”,* Hamparsum Bozdogan, sept 1987

[10] <http://r-statistics.co/Linear-Regression.html>

[11] *“Introduction of Decision Trees”*, J. R. Quinlan, 1986 Kluwer Academic Publishers, Boston

[12] <https://en.wikipedia.org/wiki/Random_forest>

[13] <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

[14] “*Gradient Boosting Machines, a tutorial*”, Alexey Natekin, december 2013, Munich Germany