TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Igor Anohhin 144117IAPM

DATA MINING AND MACHINE LEARNING FOR FRAUD DETECTION

Master's thesis

Supervisor: Leo Võhandu
Professor Emeritus

Tallinn 2017

Author’s declaration of originality

I at this moment certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Igor Anohhin

18.01.1982

Abstract

The problem of Fraud has reached an alarming scale nowadays. Losses due to the fraud are reaching billions of dollars every year. It is impossible to fight against fraud alone hand because the huge amount of data is hard to be analysed by one person. To reduce the number of losses decision systems that use efficient fraud detection algorithms should be invented. With the support of modern technologies, these systems are able to manage to analyse the information and to create a prediction feature model. However, the invention of these systems is not a trivial matter but a quite challenging task due to the huge amount of different and unbalanced data. Moreover, it is not clear which Machine Learning Algorithm should be implemented. The present Thesis aims to give some answers by focusing on the following issues: 1) which of the Machine Learning Algorithms: Logistic Regression, Decision Tree or Self-Organized Map fits better to deal with the problem of Fraud Detection, 2) why and where it is best to implement Supervised or Unsupervised methods of Machine Learning in scope of Fraud Detection, 3) the way to deal with unbalanced data and use of such data for Machine Learning. A program prototype that can examine the provided data and make a decision according to the test data will be presented at the end of the Thesis.

This thesis is written in English and is 41 pages long, including 5 chapters, 33 figures, and 18 tables.

Annotatsioon

Masinaõpe ning andmete otsimine pettuste avastamiseks.

Tänapäeval on pettuste probleem jõudnud murettekitava ulatuseni. Kahjumit tekib pettuse pärast miljardeid dollareid aastas. Inimesed ise ei saa võidelda pettuste vastu, suure hulga andmete pärast, inimesed lihtsalt ei suuda analüüsida kõiki andmeid. Kahjude vähendamiseks saame luua õppivad otsustussüsteemid, mis kasutavad tõhusaid pettuste avastamise algoritme. Need süsteemid peavad suutma analüüsida andmeid ning luua mudeli, mida saaks kasutada pettuse leidmiseks. Aga ehitada sellist süsteemi ei ole triviaalne ülesanne, see on üsna keeruline, erinevate ning alati mitte tasakaalus olevate andmete pärast. Samuti ei ole selge, millist masinõppe algoritmi me peaks kasutama. Autori eesmärgiks on pakkuda mõned vastused, keskendudes sellistele küsimustele nagu: 1) milline algoritm valikust: Logistic Regression, Decision Tree või Self Organised Maps, on paremini sobiv pettuste avastamise probleemi lahendamiseks, 2) miks ja kus on parem kasutada järelvalvega või järelevalveta meetodeid pettuste avastamiseks, 3) kuidas me saame tegeleda tasakaalustamata andmete hulkadega, kuidas kasutada selliseid andmeid masinõppes. Üheks eesmärgiks on teha prototüüp, mis võimaldab mudelit õpetada ning kasutada seda pettuse leidmiseks. Samuti, töö lõpus, autor pakub erinevad võimalused mis tasub proovida tulevikus, õpetava mudeli tulemuse paranemise jaoks. Autori poolt oli valitud see teema, kuna ta arvad, et see on väga huvitav, perspektiivne ning aitab autorile tulevikus, enda töös. Printsiibid, mis autor kasutas lõputöös, saab ka kasutada teiste probleemide lahendamiseks, näiteks klassifitseerida andmed, leida andmete hulgas mustrid, ehitada otsuste puud jne.

Selles töös, author kasutab avaliku andmete hulk. Andmete hulk koosneb kredit kaartide transaktsioonidest, andmed on väga tasakaalustamata ning kokku on 284807 rida. Andmed on leitav *http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata*.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 41 leheküljel, 5 peatükki, 33 joonist, 18 tabelit.

List of abbreviations and terms

AUC	Area Under Curve
PCA	Principal Component Analysis
SMOTE	Synthetic Minority Over-sampling Technique
APACS	Association for Payment Clearing Services
FP	False Positive (genuine)
TP	True Positive (fraudulent)
FN	False Negative (fraudulent)
TN	True Negative (genuine)
ROC	Receiver operating characteristic
TPR	True Positive Rate
TNR	True Negative Rate
FPR	False Positive Rate
FNR	False Negative Rate
NA	Not a number
DT	Decision Tree
CHAID	Chi-square Automatic Interaction Detector
SOM	Self-organizing map
BMU	Best matching unit
WSS	Within groups sum of squares
LR	Logistic Regression
MSE	Mean square error
BER	Balanced Error Rate
ERR	Error
KNN	K-Nearest Neighbour

Table of contents

List of figures

List of tables

1 Introduction

1.1 The problem of Fraud

Fraud investigators classified Fraud as "Wrongful deception with the intent to gain personally or financially and intentional deception to persuade another person to part with something of value" [1]. The fraud problem is growing and can be estimated as difficult and challenging. The issue of fraud is as ancient as the history of humanity itself; it takes different forms: spam, payment fraud, account takeover, phishing, financial fraud and much more. Its costs are not always transparent. In addition to revenue costs, the indirect losses may arise as well, and one of those may be a reputation loss. The company can lose its good standing and thus clients since nobody wants to use a product with tarnished brand image.

The Nilson Report [2] serving as the source of news and analysis in the global payment (mobile and card) industries informs that in 2015 the amount of global worldwide fraud losses has reached $21.84 billion and the $31.67 billion loss was projected in huge losses in 2020 [2]. Moreover, according to APACS, the credit card losses in the United Kingdom have been growing rapidly from £122 million in 1997 to £440.3 million in 2010 that shows the increase of credit card fraud [3]. In addition to these statements, the Annual Fraud Indicator has revealed that the UK cost of fraud was £50 billion in 2013 and could be as high as £193 billion per year [4].

1.2 Machine learning for Fraud detection

Fraud can be smart, and it is continuously evolving. Without unlimited resources, it is very hard, better say, impossible to detect and stop fraudsters. Machine learning is an excellent candidate to pursue fraud detection in a scalable manner with small expense and effort. Also, this approach can help to find hidden patterns that are not directly apparent to a person. Moreover, machine learning can self-adapt to new events. What we can do is to try machine learning algorithms to examine the data and make decisions based on new events. The output of approach comes from the available information from the system where we aim to prevent the fraudulence. However, some data may be vast, unfinished, unbalanced and not trustworthy.

1.2.1 Machine learning for credit card Fraud detection

Credit card fraud is a wide-ranging term for theft and fraud committed by utilising or using a payment card, such as a credit card or debit card, as a fraudulent source of assets in transactions. The goal may be to obtain goods without payment or to draw unauthorised cash from a user's account. Credit card fraud is also an adjunct to identity data stealing. The frequency of credit card fraud is limited to about 0.1% of all card transactions, that has resulted in massive financial losses as fraudulent transactions were large-scale operations. It is hard to generate the fraud model due to the limited amount of fraudulent transactions, so, the credit card transactions do not represent the positive examples very well. However, credit card fraud is a major problem that may result in many distresses if not dealt with efficiently.

There are some techniques that data analysts can use for credit card fraud detection.

- The decision tree, it is easy to implement, understand and, which is important, display.
- Peer group analysis (clustering the data) allows the fraud investigators to identify accounts that are behaving differently from their usual pattern at the specified period.
- Neural networks, the fraud investigators have to cluster all data according to the type of account it belongs to.
- K-Nearest Neighbour, the classifying samples supervised technique. KNN calculates the mean distances between various points on the input objects (vectors) and after that assigns the unlabelled point to the class of its nearest neighbours.
- Bayesian Network, according to the David Heckerman [5], is a "graphical model that encodes probabilistic relationships among variables of interest. When used in conjunction with statistical techniques, the graphical model has several advantages for data analysis; it handles missed data entries, learns causal relationships, handles overfitting".
- K-means, the unsupervised clustering algorithm, divides the data into k clusters, and guarantee that the data in the same cluster are similar.

2 Foundations

The machine learning plays a significant role in fraud detection. This Chapter is an Introduction to the domain of machine learning before the author is moving to the Contribution Chapter. Firstly, the target of the main topic is to clarify supervised and unsupervised learning methods. Secondly, it will disclose the cross-validation and bias vs. variance problems. The Chapter 2.4 will cover training, testing, validation and model selection. The Chapter 2.5 will introduce the issue of unbalanced data and a method to balance it.

2.1 Supervised and Unsupervised Learning

2.1.1 Supervised learning

Supervised learning is the most frequent type of machine learning problem. Data analysts classify supervised learning problems as "regression" and "classification" problems. According to the [6] general acceptance, the supervised machine learning is the task of inferring a function from labelled training data. Supervised machine learning method means that learning algorithm uses supervised and labelled data in which every example consists of the input object and the output value. Typically the input object is a vector, and the output value is a signal, for instance, binary {0; 1}. The researcher gives the dataset to the algorithm in which the correct answers algorithm provides as an output.

The researcher needs to gather training set to use this method, and training set should be distinctive in the actual use of the function. With gathered input objects, the corresponding outputs are also collected, either from human experts or measurements. The accuracy of the learned function strongly depends on how data represent the input object. However, the input data should contain enough information to predict the output accurately. To solve supervised machine learning problem the researcher also has to evaluate the accuracy of the learned function, so after the estimate function measures are made the performance of the resulting function should be measured on the test set separately from the training set.

If the investigator provides a set of N training examples of the form $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ such that x_i is the feature vector of the *i-th* example and also its label, a learning algorithm seeks a function $g : X \to Y$, where X is the input space and, Y is the output space. The function g is an element of some space of possible functions G, usually called the hypothesis space.

2.1.2 Unsupervised learning

We define unsupervised learning as problems where the researcher provides the data without the desired output. These algorithms are used to organise data clusters. Many companies, online shops and others have a lot of data in their databases, such as customer information for instance. We can look at this set of customer information and automatically detect the market segments. Unsupervised learning allows analysts to approach problems with little or no idea of what to expect as the resulting outcome. Analysts can derive structure from data where they do not necessarily know the effect of the variables [7]. With unsupervised learning, we can describe hidden structure from unlabelled data. Because the data is unlabelled there is no real estimation of the accuracy of the structure that is output by the appropriate algorithm — that is one way of distinguishing the unsupervised learning from supervised. For unsupervised methods that are used in different spheres, one good example is the voice recognition. Other examples of unsupervised learning are clustering and dimensionality reduction and classification.

The researcher's community describes unsupervised learning as learning of a probabilistic data model. Even if the researcher issues the data without supervision or rewards, it might make sense for the function to estimate a model that represents the probability distribution for a new input x_n given previous inputs $x_1, \ldots, x_n - 1$ (consider the obviously useful examples of stock prices, or the weather).

Techniques employed in anomaly detection are often combine profiling and outliers detection methods. Profiling and outliers model a baseline distribution that represent normal behaviour and then seek to detect information that shows the greatest dissimilarity with the typical response [3].

2.2 Cross validation

Evaluation and cross-validation are standard ways to measure the performance of a model. They both generate evaluation metrics that can be inspected or compared with the performance of other models.

Let us consider that we aim to train a model, but the training data set and the testing data set are limited. A certain amount of information needs to be reserved for testing, and the remaining data will be used for training. The sample used for training or testing may not be representative, and it is important to check whether the data is representative or not. A statistical technique called cross-validation could be implicated. In cross-validation method, a fixed number of partitions should be determined. For example, an analyst might choose to use 5; then data set should be split into five approximately equal partitions. The researcher must use each partition for testing and the rest of training. That means that the researcher uses one section for testing and four sections for learning and repeats this procedure five times, so the algorithm uses each partition for testing only once. After all, the researcher needs to calculate the average error and accuracy. Cross-validation helps to determine how well a model would generalise new data sets.

2.3 Bias vs. Variance

In this Chapter, the author will identify the relationship between the degree of the polynomial d and the underfitting or overfitting of the hypothesis. For a better understanding of how to improve the data fitting process resulting in more accurate models, it is important to know how different sources of errors can lead to bias and variance. High bias is under fitting, and high variance is over-fitting. Ideally, the golden mean should be found between these two.

2.3.1 Bias (under fit)

Suppose, the investigator has a classification problem, and we want to train a model. After training the model, we can plot the hypothesis:

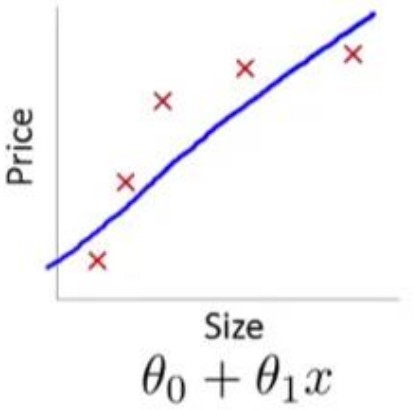

Figure 1. High bias. The author takes the picture from www.coursera.org/learn/machine-learning.

Underfitting occurs when a hypothesis cannot find and capture the underlying trend of the dataset. It might happen when a linear model is fit to non-linear data or when there is not enough training data. Such function will have poor predictive performance.

2.3.2 Variance (over fit)

Where an algorithm overfits the model (we use the large degree), is summarised in the figure below:

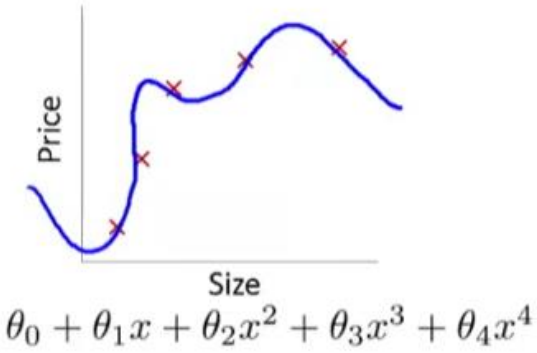

Figure 2. High variance. The author takes the picture from www.coursera.org/learn/machine-learning

In over-fitting, a statistical model describes random error or noise instead of the base ratio. Overfitting can occur when a model is excessively complex, such as having too many parameters about the number of observations. A predictive model with overfitting has poor predictive efficiency, as it overreacts to minor fluctuations in the training data [8].

2.3.3 Diagnosing Bias vs. Variance

We want to determine if bias vs. variance is the problem that leads to bad predictions. The increase in the number of features (polynomial degree *d*) will tend to bring the training error to decrease. However, at the same time, the cross-validation error might turn to decrease with the increase of *d* up and then it will increase as *d* increased, forming a convex curve [9]. The author illustrated it in the picture below:

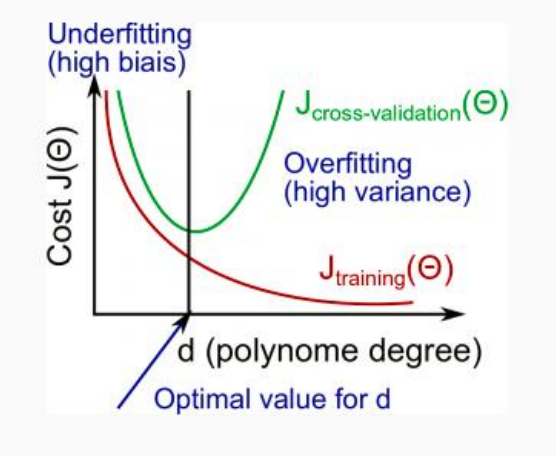

Figure 3. Plot the diagnosing of Bias-Variance trade-off. The author takes the picture from www.coursera.org/learn/machine-learning.

- Both errors should be nearly equal.

2.4 Selection of model and datasets

After data analyst trains the model, he may want to know whether the model fits well on training set or not. However, if learning algorithm fits well on the training set, it does not necessarily mean that it is a good model. For instance, it could over-fit the hypothesis and result in a poor prediction. It is especially dangerous that the error in this assumption, measured in the training data set, will be lower than any other data set.

If a system analyst tries many models with a different number of features, he also can use the polynomial degree as a feature; he can use the systematic way to find and choose the better model. An analyst can test each model and look at the error in the result.

For training and testing models, the researcher needs to split the data set. One common way to do this is by dividing the data set into three pieces: a training set 60%, cross-validation (more about cross validation on page 15) set 20%, and test set 20%. The researcher needs a separate testing set because only then he can guarantee that the

experimental data is not used in the training process. After the researcher divides the data set into three pieces, the engineer can calculate three errors separately for three data sets using the method given below.

Firstly, the engineer needs to optimise theta θ for each training dataset. Secondly, he is using least error using the cross validation method on page 15 and after that estimates the error using the test dataset.

2.5 Unbalanced data

In this Section, the author will discuss unbalanced datasets and approaches that researchers can utilise with such data interactions as well as the data set that is to be used for these experiments.

2.5.1 Dataset

The dataset is containing transactions made by credit cards in September 2013 by European cardholders, are taken from the public resource and are available at *http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata*. The data scientists have already used this dataset in a number of works, one of those is [3]. The dataset is very unbalanced, in total there are 284807 rows (transactions) and only 492 are positive (fraud) examples (about 0.172% of all data). Data contains 30 numerical input variables and one binary output variable. Only three variables from the dataset are understandable; they are Time, Amount and Class. All the others variables, from V1 till V28, are transformed using principal components and cannot be revealed. *Time* variable means the seconds between each transaction and the first transaction in the dataset. The *amount* is transaction amount. The *class* is output variable, the label of the operation; it can be one (1) positive or zero (0) negative. In the case of fraud, the transaction is marked as positive, and when the operation is genuine, it is labelled as negative.

2.5.2 Problem of unbalanced data

As the author mentions above, the dataset that is being used in experiments is very unbalanced. Such data, according to this resource [3], is hard to use in learning algorithm for model training. That is because most of the learning methods are not suitable to manage a large difference between the numbers of cases belonging to different classes, when the total number of positive examples is far less than the total number of negative

examples, as in the case of this Thesis. The learned classifier would try to classify fraudulent transactions as genuine transactions. For example, there is a dataset where the acceptable case is about 99%, and if classifier classifies all 100% cases as genuine, then the accuracy of 99% can be seen. However, this is not the satisfactory result as data scientist can not use this model in the real world because the model did not find any case of fraudulent transactions. In the case when our class labels are mostly negative or mostly positive, a classifier that always outputs 0 or 1 will achieve seemingly high accuracy. Because the function classified all of the positive examples as negative, this case is called false positive, and that is what the data scientist aim to avoid.

2.5.3 How to deal with unbalanced data

There are several methods how the analyst can deal with unbalanced data [3] and methods that distinguish data and level algorithms. In data level methods the analyst interacts with data as pre-processor aims to modify dataset, rebalance the unbalanced data and remove noise between two classes before the analyst uses the data in the algorithm. An analyst can generate synthetic positive examples (oversampling) or remove negative examples (undersampling), or he can use both methods. At the algorithmic level special algorithms that are adjusted to deal with the minority, class are used.

The author, first of all, suggests using data level methods for such unbalanced dataset and those methods that can be simply realised by researchers. The method that the author suggests using is SMOTE. SMOTE is the oversampling process that generates positive (less present) examples in the neighbourhood of observed ones. The image below illustrates SMOTE oversampling.

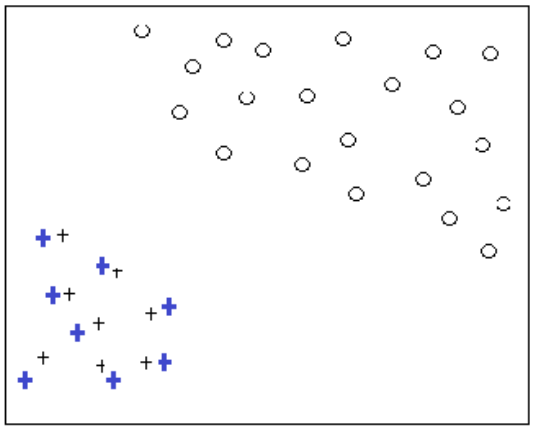

Figure 4. SMOT produced blue positive instances in the neighbourhood of observed ones.

2.6 Estimation

Evaluation model expects a scored data set as input (or 2 or even more if the analyst would like to compare the performance of different models). The model needs to be trained using the training data set and makes predictions on test dataset before it can evaluate the results. The estimation is based on the scored labels/probabilities along with the positive labels. The evaluation of learned model is crucial because only that analyst can decide if the hypothesis is well suited. In this Chapter, the author will disclose some methods that are used in the Thesis to estimate the model.

2.6.1 Area under the curve

AUC is a metric for binary classification problem. The area under the curve considers all possible thresholds that can be used to compute the accuracy from probabilities. Different thresholds result in different TP/FP rates. If an analyst decreases the threshold, the model is found that can predict more positive examples and at the same time it can increase FP classification rate.

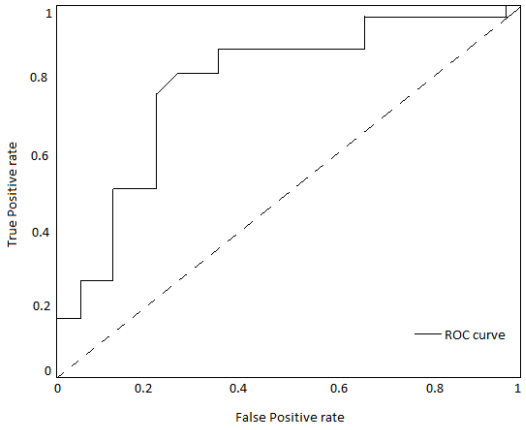

Figure 5. Relation between threshold and FP/TP rates

Dashed line on the plot shows what can be achieved with random classifier where it is expected as many true positive examples as false positive ones. The area under the curve for the case is 0.5. Perfect prediction is 1. ROC curve is used to visualise the performance of a binary classifier and helps to understand the impact of that choice visually. AUC is a way to summarise the performance in series. It is important to note that the AUC is the highest when the two curves are farthest with little overlap.

According to [10] AUC is useful metrics event if classes are highly unbalanced.

2.6.2 Evaluation of Classification

Classification will help to understand better which model is more suitable to predict that this metric will count the number of mistakes made. The binary class labels in the training set can take on only two possible values that mostly refer to as positive or negative. The positive and negative instances that a classifier predicts correctly are called true positives TP and true negatives TN. The incorrectly classified instances are called false positives FP and false negative FN. Based on that True Positive Rate, True Negative Rate, False Positive Rate, False Negative Rate, Precision, Accuracy, Balanced Error Rate and Error Rate concepts will occur.

$$TPR = \frac{TP}{TP+FN} \quad (1)$$

$$TNR = \frac{TN}{FP+TN} \quad (2)$$

$$FPR = \frac{FP}{FP+TN} \quad (3)$$

$$FNR = \frac{FN}{TP+FN} \quad (4)$$

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (6)$$

$$BER = 0.5 * \left(\frac{FP}{TN+FP} + \frac{FN}{FN+TP}\right) \quad (7)$$

$$ERR = \frac{FP+FN}{P+N} \quad (8)$$

For example:

Table 1. The confusion matrix of a binary classification estimation. The confusion matrix is simply a table showing the number of instances that fall under each of 4 categories (TP, TN, FP and FN).

Model		**Predicted Class**	
		Positive	Negative
Actual Class	Positive	5 TP	9 FN
	Negative	10 FP	100 TN

Consider equation (1) it is found that TPR = 5 / (5 + 9) = 0.36. For precision calculation using equation (5) Precision = 5 / (5 + 10) = 0.33. For accuracy calculation using equation

(6) Accuracy = 105 / 124 = 0.847. However, there can be situations when the accuracy is not a right measure of performance particularly in unbalanced classification problem. In unbalanced classification problem one class is much more frequent than the other.

The balanced error rate (BER) is the mean of the errors on each class; BER = 0.5 * (0.091 + 0.643) = 0.367. However, BER may not be suitable because of the different price of misclassification FN and FP rates. Precision and TP have completely different behaviour, having high Precision leads to bad TP and vice versa.

3 Data mining

In this Chapter, the author will investigate the dataset that is used in experiments. Data mining is the process of analysing data from different aspects and summarising it into valuable information - information that can be used to increase the revenue or to cut costs, or for both. Several techniques that researchers can use for data discovery and optimisation will be discussed.

The “R” programming language and “R” Tools for Visual Studio (for more information on “R” language, please refer to https://www.r-project.org) were used in experiments.

3.1 Dataset

The dataset dimension is 284807 rows and 31 columns, the last column is a label that shows if the transaction is fraudulent or genuine. As it is mentioned on page 2.5.118, the dataset is completely unbalanced.

Table 2. Dataset class statistic.

Class	**Rows**	**Percentage**
Negative (genuine)	284315	99,827
Positive (fraudulent)	492	0,1727

Table 3. Credit card dataset summary. Part 1. The last column shows if data are normally distributed or not.

Var.	**Min.**	**1st Qu.**	**Median**	**Mean**	**3rd Qu**	**Max.**	**Norm.**
Time	0	54200	84690	94810	139300	172800	No
V1	-56,41000	-0,920	0,018	0,000	1,316	2,455	No
V2	-72,72000	-0,598	0,065	0,000	0,804	22,060	No
V3	-48,3300	-0,890	0,180	0,000	1,027	9,383	No
V4	-5,68300	-0,849	-0,019	0,000	0,743	16,880	No
V5	-113,70000	-0,692	-0,054	0,000	0,612	34,800	No
V6	-26,1600	-0,768	-0,274	0,000	0,399	73,300	No
V7	-43,5600	-0,554	0,040	0,000	0,570	120,600	No
V8	-73,22000	-0,209	0,022	0,000	0,327	20,010	No
V9	-13,43000	-0,643	-0,051	0,000	0,597	15,590	No

Table 4. Credit card dataset summary. Part 2. The last column shows if data are normally distributed or not.

Var.	Min.	1st Qu.	Median	Mean	3rd Qu	Max.	Norm.
V10	-24,590	-0,535	-0,093	0,000	0,454	23,750	No
V11	-4,797	-0,762	-0,033	0,000	0,740	12,020	Yes
V12	-18,680	-0,406	0,140	0,000	0,620	7,848	No
V13	-5,792	-0,648	-0,014	0,000	0,662	7,127	Yes
V14	-19,210	-0,426	0,051	0,000	0,493	10,530	No
V15	-4,499	-0,583	0,048	0,000	0,649	8,878	Yes
V16	-14,130	-0,468	0,066	0,000	0,523	17,320	No
V17	-25,160	-0,484	-0,066	0,000	0,399	9,254	No
V18	-9,4990	-0,499	-0,004	0,000	0,501	5,041	Yes
V19	-7,214	-0,456	0,004	0,000	0,459	5,592	Yes
V20	-54,500	-0,212	-0,062	0,000	0,133	39,420	No
V21	-34,830	-0,228	-0,029	0,000	0,186	27,200	No
V22	-10,930	-0,542	0,007	0,000	0,529	10,500	Yes
V23	-44,810	-0,162	-0,011	0,000	0,148	22,530	No
V24	-2,837	-0,355	0,041	0,000	0,439	4,585	No
V25	-10,300	-0,317	0,017	0,000	0,351	7,520	No
V26	-2,605	-0,327	-0,052	0,000	0,241	3,517	Yes
V27	-22,570	-0,071	0,001	0,000	0,091	31,610	No
V28	-15,430	-0,053	0,011	0,000	0,078	33,850	No
Amount	0,000	5,600	22,000	88,350	77,160	25690	No

The tables above show that most of the data was not distributed normally. If the researcher does not scale the data, it can influence the accuracy of predictions, and the model will predict poorly. At the same time, the data was very similar except "*Time*" and "*Amount*" but with different precision. Also, the researcher can scale the data for better performance. The histogram is used to see how data distributions look. Some of the histograms are presented below. Though not all of the histograms are shown here because that will take much space, all pictures are included in the final package.

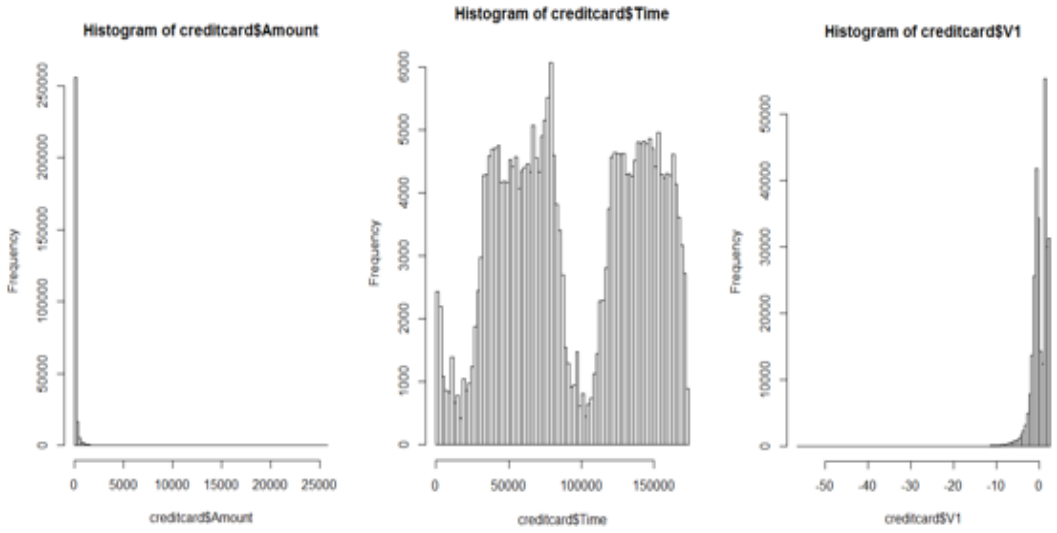

Figure 6. Badly distributed “Amount,” “Time” and “V1” variables.

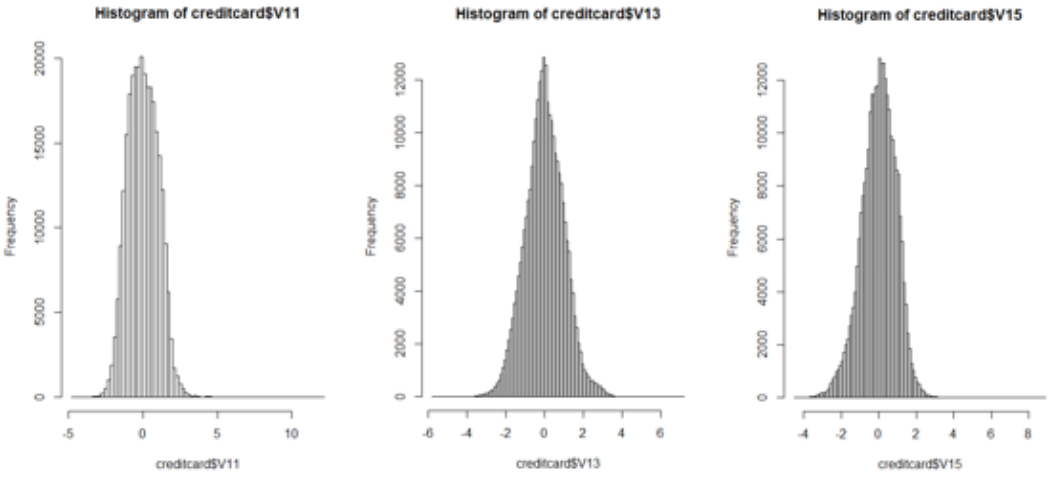

Figure 7. Normally distributed “V11”, “V13” and “V15” variables.

Before the analyst starts to use the data in machine learning algorithm, it is useful to see if the data set includes not numbers, that information can be removed. In the case of the Thesis, there is no NA data. In “R” we can simply use this line of code:

```
sapply(creditcard, function(x) sum(is.na(x)))
```

Figure 8. Returns sum of not a numbers in each variable.

Also, we can verify the data for uniqueness. In “R” we simply can use this line of code:

```
sapply(creditcard, function(x) length(unique(x)))
```

Figure 9. Returns number of unique rows for each variable.

Table 5. The variables uniqueness.

Time	**V1 till V28**	**Amount**	**Class**
124592	275663	32767	2

These results show that variables from *V1* till *V28* are mostly unique. On the other side, *Amount* is not so different as *V1* till *V28*; there are many transactions with the same quantity of money. *Time* denotes the seconds elapsed between each transaction and the first transaction in the dataset. Same time indicates that someone had made different transactions at the same time.

It is always useful to plot the data. Due to the unbalanced dataset where positive examples are represented less than negative examples data visualisation gives the opportunity to compare the variables. That shows how positive and negative examples depend on some variables.

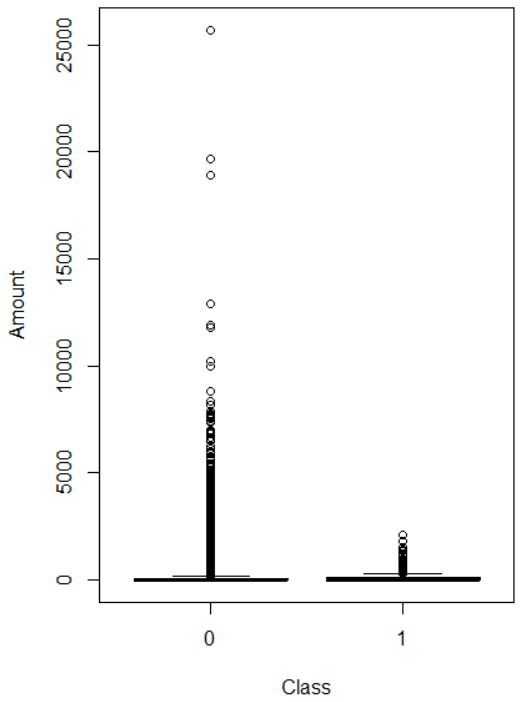

Figure 10. The amount of credit card transactions.

Figure 10 shows that fraudster made all positive transitions with the lower amount of money. That indicates that fraudsters had tried to make unnoticeable transactions. Feature *Amount* can be used in example-dependent cost-sensitive learning [11]. Next, the author plots the time series classification.

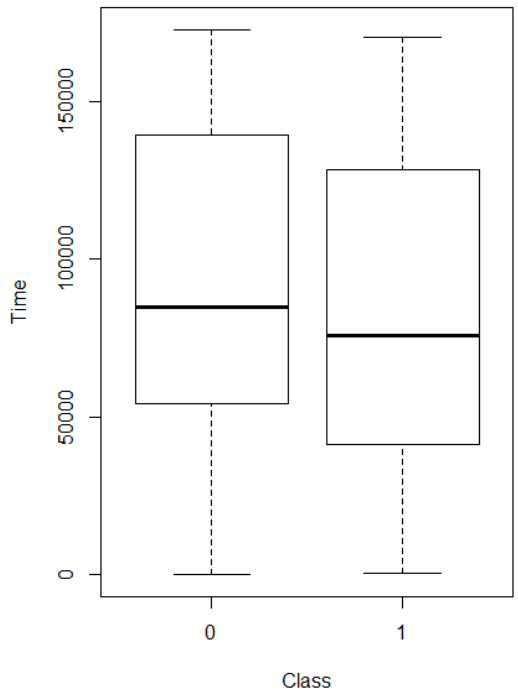

Figure 11. Time classification summary. Negative and positive.

Feature *Time* denotes the seconds elapsed between each transaction and the first transaction in the data set on page 18. Fraudulent transactions take fewer seconds than individual transactions. Other features from *V1* till *V28* have been transformed using principal components.

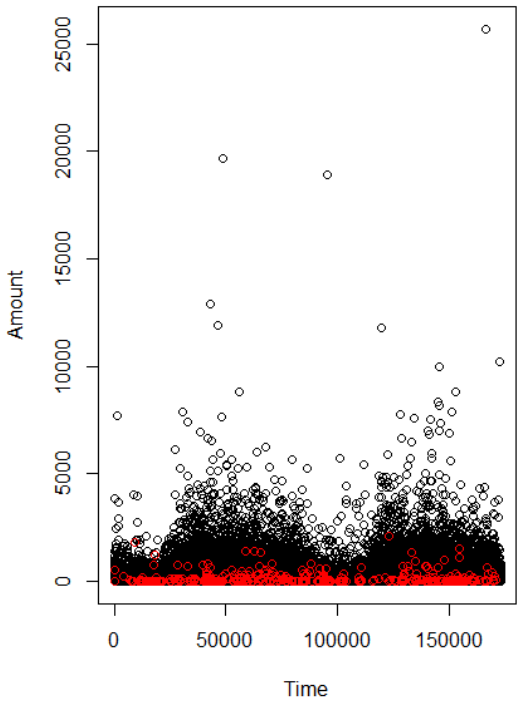

Figure 12. Features Time and Amount with positive and negative examples.

Figure 12, shows how positive examples depend on variables Time and Amount. It is hard to distinguish positive from negative samples due to data overlapping.

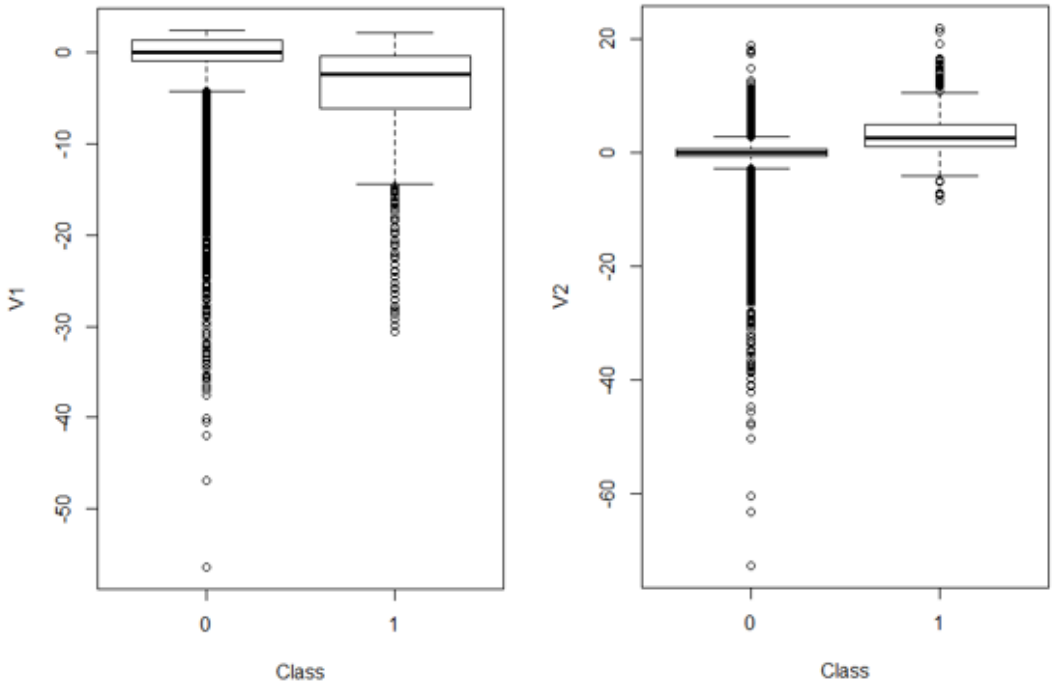

Figure 13. Credit card V1 and V2 classification summary. Negative and positive.

Figures 13 show that positive examples are less than negative examples. Next picture 14 show *V1* and *V2* together with positive examples.

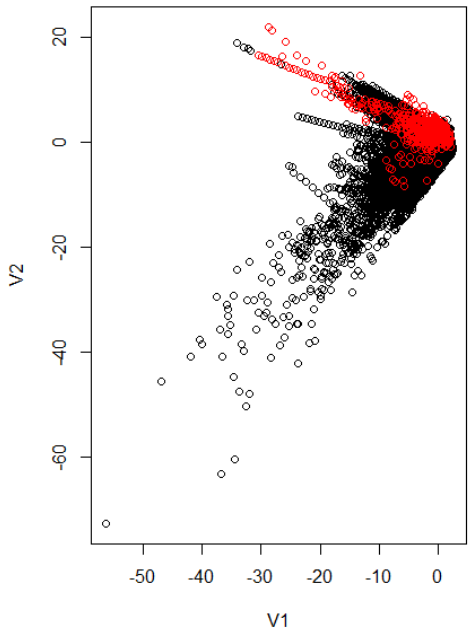

Figure 14. Plot the V1 vs. V2 features with negative and positive examples.

The author will use PCA technique to convert variables to the components that explain most of the variance, and then, the author will plot first two elements, the components that are better to describe the data.

Table 6. PCA components. The proportion of Variance indicates that first two elements are most explain the difference. From CP3 till PC29 variance values are the same.

Component	**Standard deviation**	**Proportion of Variance**	**Cumulative Proportion**
PC1	1,399	0,065	0,065
PC2	1,297	0,056	0,121
PC3	1,000	0,033	0,155
PC28	1,000	0,033	0,988
PC29	0,562	0,010	0,999
PC30	0,205	0,001	1,000

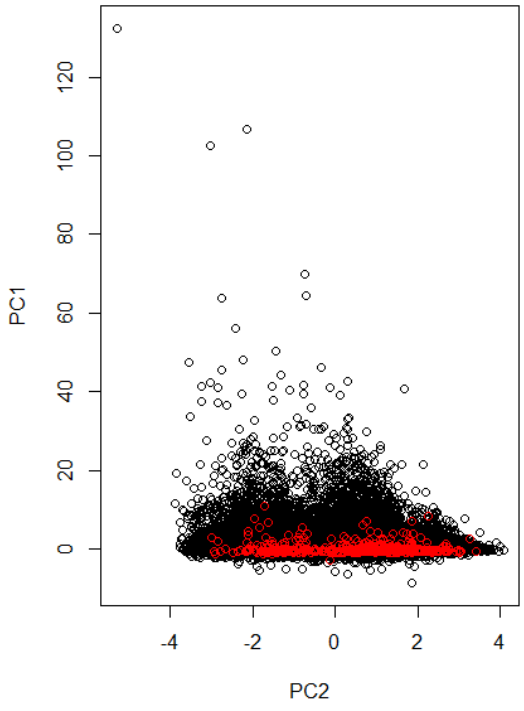

Figure 15. Two principal components, which are most explain the difference, with highlighted positive examples.

3.2 Conclusion

To conclude, data mining examinations in this chapter show that the researcher did not preprocess the data set. Negative and positive examples overlap and such information was difficult to distinguish. However, data screenshots indicate that positive examples are under-sampled and that makes it difficult to learn the model. That can affect the precision of predictions, but at the same time, the accuracy might be precise.

When using a dataset with a broad range of features, it is reasonable to use PCA method to reduce the dimension of the data. Often PCA is used as an instrument in data analysing process and in order to create models for predictions. However, PCA can be useful for analysts who want to research a lower-dimensional picture when PCA can supply the analyst with the whole picture with a projection of the object when viewed from its most clear point of view. The researcher can create a low-dimensional image by using only the first two or three principal components so that the dimensionality of the data is ultimately reduced.

PCA is not optimised for class distinguishability. Also, PCA result depends on the scaling of the data (variables). However, it is suitable for pattern recognition. Moreover, it can be used in the distance quantification between classes by calculating in the principal component area the mean and showing the Euclidean distance between the classes.

4 Experiments with learning algorithms

In this Chapter the author will implement three different algorithms: Logistic Regression, Decision Tree and Self-Organised Map and the author will apply these algorithms on the same unbalanced dataset. In the end, the author will compare the results to make the conclusion. The author uses Visual Studio 2015 and R Tools for Visual Studio 1.0.30213.1900 RC1 for the experiments.

4.1 Logistic Regression

In this Chapter, a logistic regression model (also called the logit model) shall be set to predict whether a credit card transaction is a fraud or genuine. Moreover, the author will introduce to the readers the notion of classification, the cost function for logistic regression, sigmoid function and gradient.

4.1.1 Preliminaries

Logistic regression is a method for classifying data into discrete outcomes, for example, $\{0; 1\}$. In the logistic regression model, the log odds of the outcome is modelled as a linear combination of the predictor variables. The logistic regression hypothesis is defined as:

$$\mathrm{h}_0(x) = g(\theta^T x) \tag{9}$$

Function g denotes the sigmoid function. The sigmoid function is defined as:

$$g(z) = \frac{1}{1+e^{-z}} \tag{10}$$

Where z is:

$$z = \theta^T x \tag{11}$$

For a matrix, hypothesis function should perform the sigmoid function on every element.

The cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(h_0(x^{(i)})) - (1 - y^{(i)})\log(1 - h_0(x^{(i)}))] \tag{12}$$

The gradient of the cost function is a vector of the same length as vector θ where the j^{th} element is defined as:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_0\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} where\ j\ \geq 0 \quad (13)$$

Cost function and gradient the investigator should call at the same time. This gradient looks identical to the linear regression gradient, but the formula differs because linear and logistic regressions have different definitions of $h_0(x)$.

4.1.2 Experiments

As it was mentioned on page 31 R programming language binomial logit is being used for the experiments; the code can be found in Appendix 1 – Logistic Regression code.

There is a credit card data set with 284807 rows and 31 columns, and it was divided into two pieces. First is the training data set with 199295 rows and second is the test data set with 85412 rows. The author separates the testing set from the training dataset because only that can guarantee that the trained model was not used as a testing dataset.

Table 7. Training dataset negative/positive class statistics.

Class	**Rows**	**Percentage**
Negative (genuine)	198960	99,832
Positive (fraudulent)	335	0,168

Table 8. Test dataset negative/positive class statistics.

Class	**Rows**	**Percentage**
Negative (genuine)	85258	99,820
Positive (fraudulent)	154	0,180

So, it is obvious that in both data sets there are a lot of negative examples and less positive.

The next step is to split training dataset into five CV data sets, as described on page 15, and to train the model. Cross-validation helps in determining how well the model would generalise to new information sets. Moreover, CV is the way of measuring the performance of a trained model. By analysing the accuracy results for each subset, the analyst can interpret the quality of the data set and figure out whether the model is receptive to variety in the data.

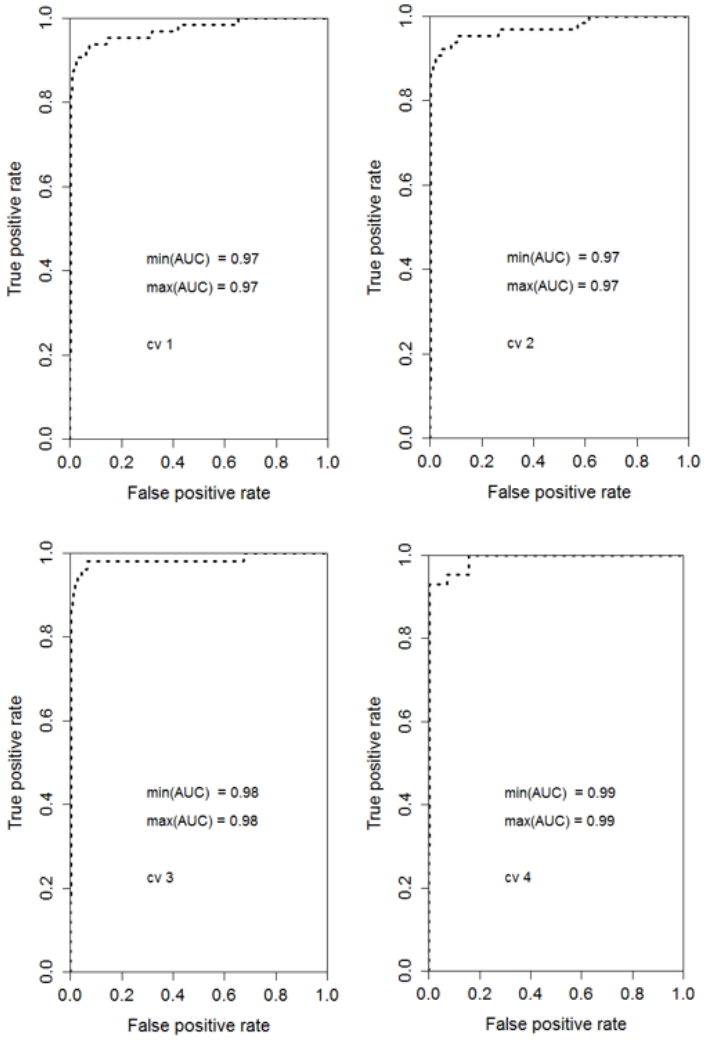

Figure 16. Logistic regression CV AUC and ROC results. AUC of the fifth CV is 0.96.

Table 9. Cross Validation results of a Logistic Regression model.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
CV1	0,562	0,999	0,0001	0,437	0,923	0,999	0,219	0,001	10,061	0,973
CV2	0,469	0,999	0,0002	0,531	0,811	0,999	0,266	0,001	10,010	0,973
CV3	0,596	0,999	0,0001	0,403	0,861	0,999	0,201	0,0005	9,801	0,984
CV4	0,744	0,999	0,0001	0,256	0,865	0,999	0,128	0,0005	9,733	0,991
CV5	0,530	0,999	0,0002	0,471	0,771	0,999	0,235	0,001	10,107	0,958
Mean	0,580	0,999	0,0001	0,420	0,846	0,999	0,210	0,001	9,943	0,976

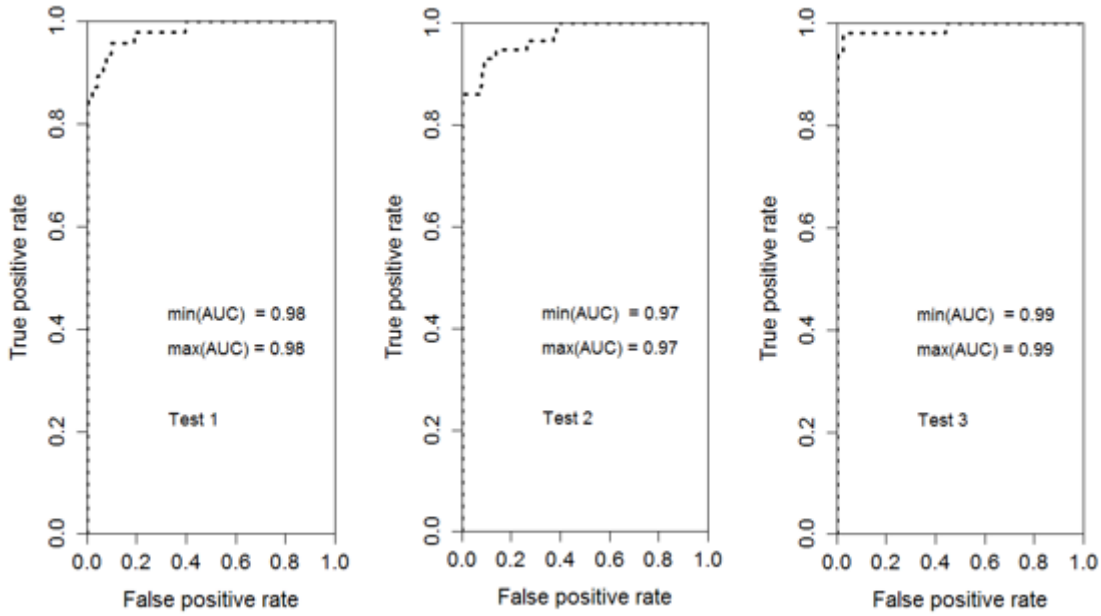

Figure 17. AUC and ROC of Logistic regression tests.

AUC for trained model is the same as for model trained with Cross Validation method.

Table 10. Confusion matrix of a Logistic Regression. The threshold is 0.5.

Num.	**TP**	**FP**	**TN**	**FN**
T1	33	6	28417	14
T2	37	6	28407	20
T3	43	5	28415	7

Table 11. Test results of Logistic Regression. The threshold is 0.5.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
T1	0,702	0,999	0,0002	0,298	0,298	0,999	0,149	0,0007	1,000	0,980
T2	0,649	0,999	0,0002	0,350	0,860	0,999	0,175	0,0009	1,000	0,973
T3	0,860	0,999	0,0001	0,140	0,895	0,999	0,070	0,0004	0,999	0,989
Mean	0,737	0,999	0,0002	0,263	0,867	0,999	0,131	0,0007	1,000	0,981

4.1.3 Conclusion

The author examines the Logistic Regression model on provided dataset. From AUC results the conclusion can be made that classes are well separated and can be well distinguished. First of all the cross-validation method was tried to be used on the dataset

and next all data sets were utilised for the model training. Trained model used three different test data sets for testing, and all the results for both methods have been added to the tables above. The result tables show the mean true positive rate for CV indicates the smaller result and at the same time the false positive rate is also lower. If the analyst wants to decrease the false positive rate for the trained model the threshold must increase. However, this will reduce true positive rate as well. In all cases the accuracy was excellent, but this is because of unbalanced data with a lot of negative and less positive examples.

In conclusion, the ROC curve and AUC will help to understand the impact of a chosen classification threshold visually. ROC will show what false positive rate should be expected according to the true positive rate. The result of the comparison of the accuracy results and error rate for each cross-validation subsets can be explained by the quality of the data set that was used and shows that the model is susceptible to the difference in the data. Furthermore, by comparing the accuracy results and error rates in the independent tests results the performance of the model reader can be interpreted as a good.

4.2 Decision Tree

In this Chapter, the author will create a decision tree model to predict whether a credit card transaction is fraudulent or genuine. The author will introduce to the reader the advantages and disadvantages of decision tree classification and the metrics.

4.2.1 Preliminaries

The decision tree used in supervised learning is a method for both classification and regression problems. The decision tree is a graph that uses a branching method to show every possible output of a decision. The decision tree creates the model that predicts the output of a target variable by learning decision rules derived from the dataset.

$$(x, Y) = (x_1, x_2, x_3, \dots, x_n, Y) \quad (14)$$

Y is the target value that engineers want to classify, and *x* is a vector of features that the function uses for predictions.

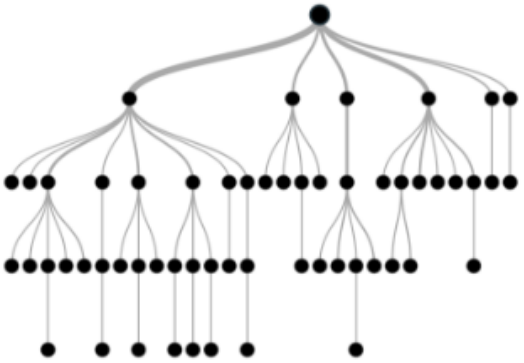

Figure 18. Decision tree. The author takes the figure from [12].

A decision tree consists of:

- Root node
- Splitting
- Decision node
- Branch/Sub-tree
- Terminal node

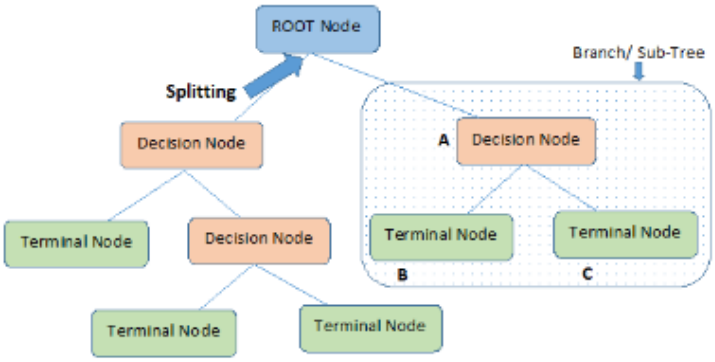

Figure 19. Decision tree elements. The author takes the figure from [12].

Some highlighted advantages of the decision tree:

- The investigator can visualise Decision trees; it is simple to understand.
- Can handle both numerical and categorical data.
- Can handle multi-output problems.
- Decision tree uses a white box model; results are straightforward to interpret.

Some highlighted disadvantages of the decision tree:

- A decision tree learners can create biased trees if some classes dominate. The dataset should be balanced.
- Over-complex trees (overfitting).

For decision tree as a metrics Gini index (in classification problem) was used. The algorithm applied these metrics to each subset candidate, and the resulting values are averaged to provide a measure of the quality of the split. The algorithm calculates Gini index for each node on according split. The approach is a summary of the square of probability for success and failure. Gini index performs binary splits, and the higher value denotes higher sameness than lower.

The algorithm which finds the statistical significance of the differences between sub-nodes and parent node is called Chi-Square. It is measured by the summa of squares of standardised differences between observed and expected frequencies of the target variable. The Chi-square algorithm works with categorical target variables, can perform two or more splits, the higher value denotes higher statistical differences between sub node and parent node. The formula for Chi-square:

$$X^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{15}$$

O is the observed value, and *E* denotes expected value. Two steps should be performed for Chi-square. First, calculate for an individual node by calculating the deviation for success and failure. Second, calculate for each node of the split using summa of all Chi-squares of success and failure.

A decision tree is built top-down from a root node and involves partitioning of the data into subsets that contain instances with similar values. The algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous, the entropy is zero and if the sample is an equally divided that means that the entropy is one.

$$Entropy = -p * log_2 * p - q * log_2 * q \tag{16}$$

Two types of entropy have to be calculated in order to build the decision tree.

First entropy is using the frequency of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i * \ log_2 * \ p_i \tag{17}$$

Second entropy is using the frequency of two attributes:

$$E(T, X) = \sum_{c \epsilon X} P(c) * E(c) \quad (18)$$

Also, we should gain the information based on the decrease in entropy after the algorithm splits the dataset by attributes. A decision tree is about finding an attribute that returns the most homogeneous branches (highest information gain). The equation of target calculation entropy:

$$Gain(T, X) = E(T) - E(T, X) \quad (19)$$

Attribute with the largest information gain should be used as the decision node. Divide the dataset by its branches and repeat the same process on every branch. Branch with null entropy will represent a leaf node and branch with entropy greater than null needs further splitting. The algorithm runs process recursively until all data is classified.

4.2.2 Experiments

As in the previous experiment in this examination, the author will use R programming language. The *rpart* library will be utilised for the decision tree; the code can be found in Appendix 2 – Decision Tree code.

The author will use the same dataset that is described on page 4.1.232, the dataset divided into two pieces. In both datasets, there are a lot of negative examples and less positive. First of all, the author will train the model with cross-validation approach. Cross-validation helps in determining how well a model would generalise to new datasets. Moreover, CV is a way of measuring the performance of a trained model. By analysing the accuracy results for each subset, the analyst can interpret the quality of the dataset and figure out whether the model is receptive to variety in the data or not.

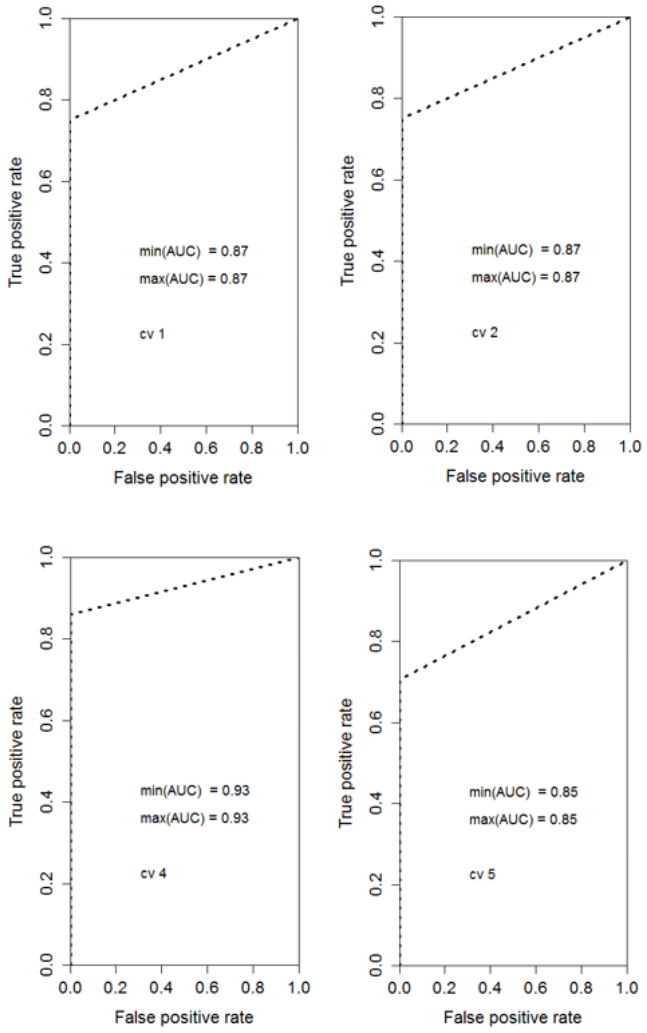

Figure 20. Decision Tree CV AUC and ROC results.

The author does not show the third Cross-validation set here because the result is the same as in the first and the second CV folds. From CV sets the fourth fold shows a better result. That can occur because of balanced data used in that training data set or the data that just better represents the model.

Table 12. Cross Validation results of a Decision Tree model.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
CV1	0,750	0,999	0,0001	0,250	0,923	0,999	0,125	0,0006	0,024	0,874
CV2	0,750	0,999	0,0001	0,250	0,980	0,999	0,125	0,0005	0,022	0,874
CV3	0,730	0,999	0,0001	0,270	0,883	0,999	0,135	0,0005	0,024	0,865
CV4	0,860	0,999	0,0002	0,139	0,840	0,999	0,070	0,0003	0,019	0,930
CV5	0,706	0,999	0.0002	0,294	0,837	0,999	0,147	0,0006	0,025	0,852
Mean	0,759	0,999	0,0001	0,240	0,892	0,999	0,120	0,0005	0,023	0,880

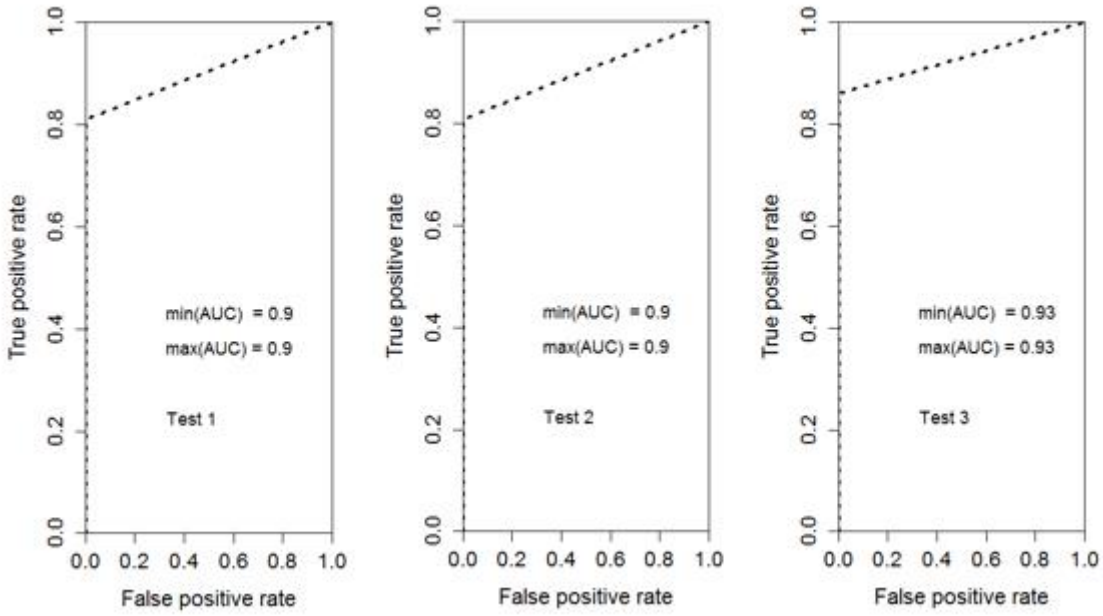

Figure 21. AUC and ROC of Decision Tree tests.

Table 13. Confusion matrix of a Decision Tree.

Num.	**TP**	**FP**	**TN**	**FN**
T1	38	4	28419	9
T2	46	6	28407	11
T3	43	4	28416	7

Table 14. Test results of Decision Tree.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
T1	0,808	0,999	0,0001	0,191	0,905	0,999	0,096	0,0005	0,021	0,904
T2	0,807	0,999	0,0002	0,192	0,884	0,999	0,097	0,0006	0,024	0,903
T3	0,860	0,999	0,0001	0,140	0,915	0,999	0,070	0,0003	0,020	0,929
Mean	0,821	0,999	0,0002	0,175	0,901	0,999	0,087	0,0005	0,022	0,912

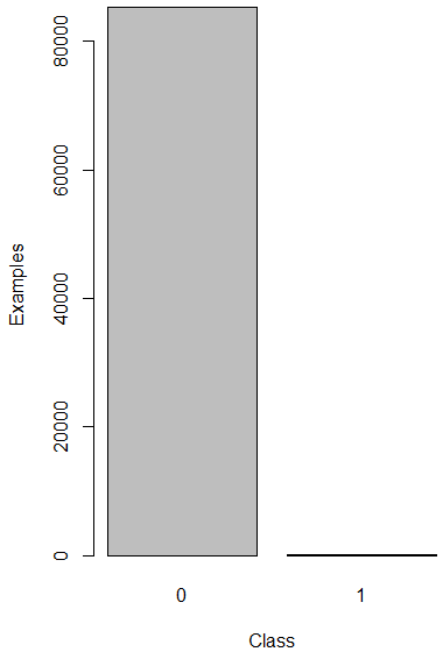

Figure 22. Decision Tree predictions.

From the forecasts, it is obvious that positive predictions count much less than negative predictions. Next figure shows the visualised decision tree.

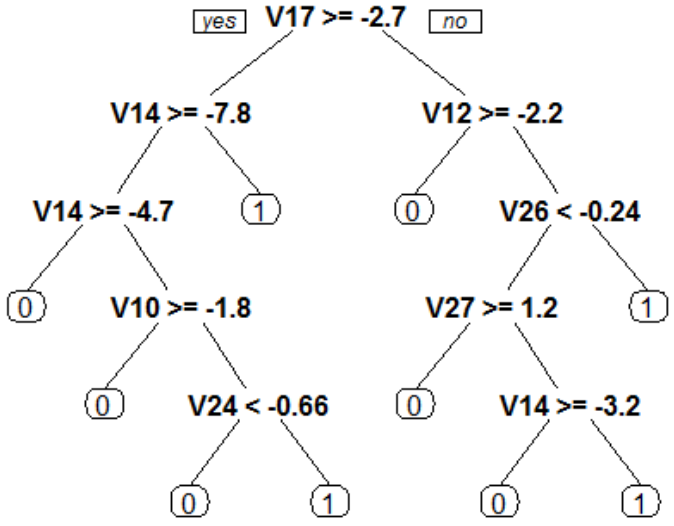

Figure 23. Visualised classification tree of the segmentation data from the credit card dataset.

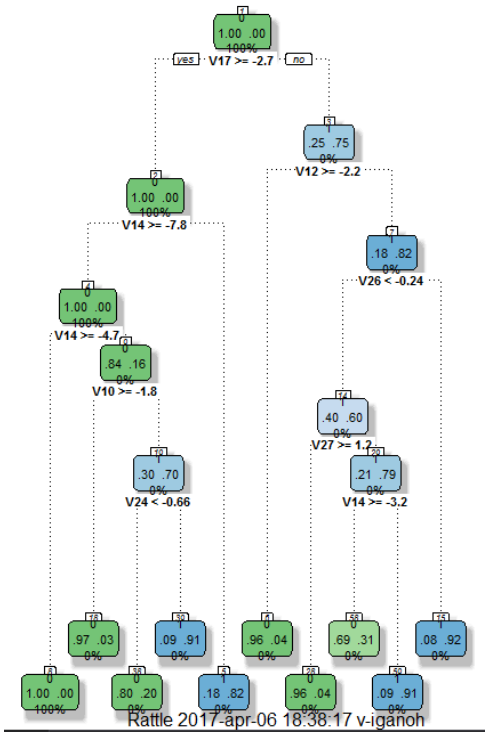

Figure 24. Visualised classification tree. Used rattle package.

Variables used in the tree construction: V10, V12, V14, V17, V24, V26, V27. Other variables filtered by the algorithm. In figure 24, each node box displays the classification, the probability of each class at that node and the percentage of observations used at that node. The dotted lines indicate to emphasise the nodes and not the tree itself, and the bottom level of leaves lining up helps to guess that the percentages in the node boxes indicate the percentage of observations that arrived at each node.

4.2.3 Conclusion

The author has examined the Decision Tree classification model on provided dataset. It is important to highlight that the dataset was not scaled and was used without any changes and not re-balanced. The AUC results indicate that classes are well separated and can be distinguished. First of all the cross validation method was examined on the dataset and then the entire dataset for the model training had to be used. The trained model uses three different test datasets for testing and all results, for both methods, have been presented in the tables above. From the result tables mean true positive rate, for CV, shows the smaller result, when at the same time the false positive rate is also lower. If the analyst wants to decrease false positive rate for the trained model, the threshold needs to be increased. However, this will reduce the true positive rate as well. In all cases the accuracy was

excellent, but this is due to the unbalanced data, where there are a lot of negative and less positive examples.

In conclusion, the ROC curve and AUC will help to understand the impact of a chosen classification threshold visually. ROC will show what false positive rate should be expected according to true positive rate. By comparing the accuracy results and error rate for each cross-validation subsets, an analyst can explain the quality of the dataset and come to the conclusion that the model is susceptible to the difference in the data. Furthermore, by comparing the accuracy results and error rates in the independent tests results the performance of the model can be interpreted as good.

4.3 Self-organizing map

In this Chapter, the author will create a Self-Organizing map model to predict whether a credit card transaction is a fraudulent or not. The author will introduce to the readers the algorithm of Self-Organizing map classification.

4.3.1 Preliminaries

SOM, or Kohonen Self Organising Feature Maps, were invented by Teuvo Kohonen, Professor of the Academy of Finland. Teuvo Kohonen had provided a way of representing multidimensional data in much lower dimensional spaces, it can be any dimension, but usually, it is one or two dimensions, a sampled representation of the input space of the training samples which is called a map. In general, the Kohonen approach creates a network that stores information in a way that the algorithm maintains any topological relationships within the training set.

The main difference from two previous algorithms, presented in this Thesis, is that this approach is a type of artificial neural network that is trained using the unsupervised learning. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backwards propagation with gradient descent) and in the sense that they use a neighbourhood function to preserve the topological properties of the input space [13]. SOM operates in two-way - training and mapping. Training process builds the map using input examples, vector quantization. Mapping automatically classifies a new input vector. SOM combines the components called nodes also known as neurones, where each neurone associated

weight vector of the same size as the input data vectors and a position in the map space. The proper arrangement of neurones is a conventional two-dimensional layout in a hexagonal or rectangular grid. The SOM describes a mapping from a higher-dimensional input to a lower-dimensional map. The procedure for putting a vector from data onto the map is to find the node with the smallest distance metric vector to the data space vector.

In the training process, training examples are put into the network; the algorithm computes its Euclidean distance (20) to all weight vectors. The neurone with most similar weight vector to the input is called the BMU. The approach adjusts the weights of the BMU and closest neurones in the SOM grid towards the vector of entry. The weight of the change decreases with time and with distance from the BMU.

$$Euclidean\ distance = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \quad (20)$$

SOM algorithm equation for an update for a neurone v with weight W_v:

$$W_v(s+1) = W_v(s) + \theta(u, v, s) * \alpha(s) * (D(t) - W_v(s)) \quad (21)$$

Where s is the step index, t an index of the training sample, u is the index of the BMU for $D(t)$, $\alpha(s)$ is a monotonically decreasing learning coefficient and $D(t)$ is the input vector; $\theta(u, v, s)$ is the neighbourhood function which gives the distance between the neuron u and the neuron v in step s. Depending on the implementations, t can scan the training dataset systematically (t is 0, 1, 2...T-1, then repeat, T being the training sample's size), can be randomly drawn from the dataset (bootstrap sampling), or implement some other sampling method [12].

The algorithm itself looks like:

- Weight vectors randomised for the nodes of the map.
- Capture an input vector.
- Go over each node (neurone) on the map and find the similarity between the vector of entry and the maps weight vector. Locate the node with the best same unit.
- Use formula (21) for updating the nodes in the neighbourhood of the BMU by pulling them closer to the input vector.
- Increase s and repeat process from step 2 until s lower then iteration limit.

A well-known problem for SOM is a selection of good initial approximation, using the random initiation of SOM weights for the approach. Due to the exact reproducibility of the results most important component initialization where the method chooses the first map weights from the space of the first principal components, it has become a popular approach.

4.3.2 Experiments

As in the previous experiment, in this examination, the author will use R programming language. For SOM the author will use *kohonen* library, the reader can find the code in Appendix 3 – Self-organized map.

The author will utilise the same data set, which was utilised by the author on page 4.1.232, the dataset is divided into the two pieces. In both pieces, there are a lot of negative examples and much less positive.

The original SOM used 15 x 15-dimensional grid with hexagonal topology.

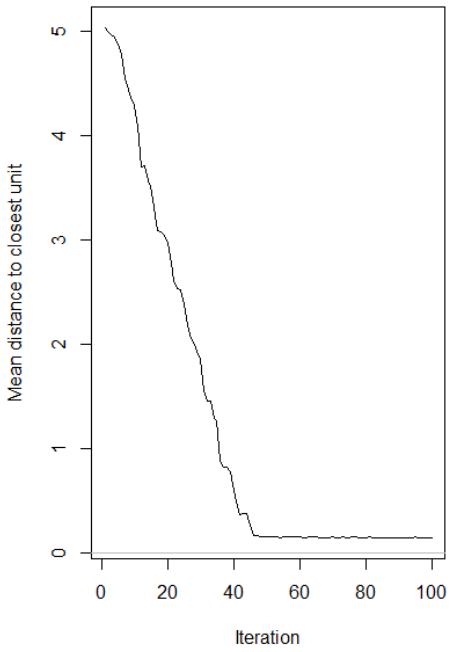

Figure 25. The plot of the training progress. This graph shows the variation between the weights of the nodes and the cases presented to it.

It shows how many iterations the approach required for the mean distance minimization. If the size of the SOM is too small, it may be hard to have convergence to a minimum. So this plot can be used to figure out the optimal size of the SOM.

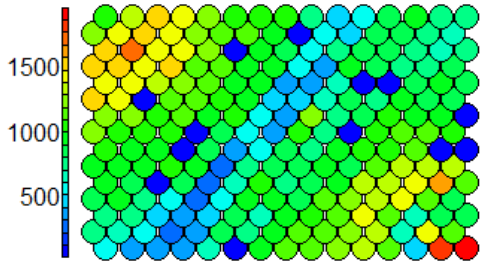

Figure 26. Count plot. Shows how many cases the approach mapped to each node on the map.

It can be used as a measure of map quality. It is good if the sample distribution is relatively uniform. Large numbers in SOM grid denote that larger map would be beneficial, but if the larger map does not change those numbers, then it can suggest a large cluster of cases.

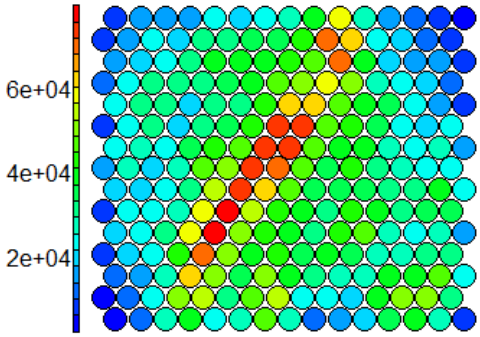

Figure 27. Plot distance between each node and it neighbours.

Next is a neighbour distance plot that shows the distance between each node and its neighbours, also known as U-Matrix. Nodes that are similar indicate small areas of neighbour distance and areas with bigger distance indicate nodes that are dissimilar. This matrix can be used to separate the clusters within the SOM.

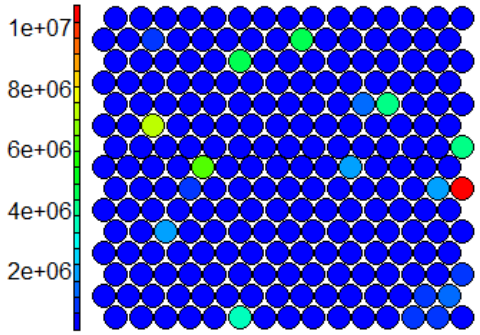

Figure 28. Plot the quality of object representation in codebook vectors.

The smaller distance shows that the codebook vectors represent objects well.

It is useful to separate data into clusters, so similar looking or behaving data points can be grouped together. Clustering helps to group related data together while these groups are different from each other.

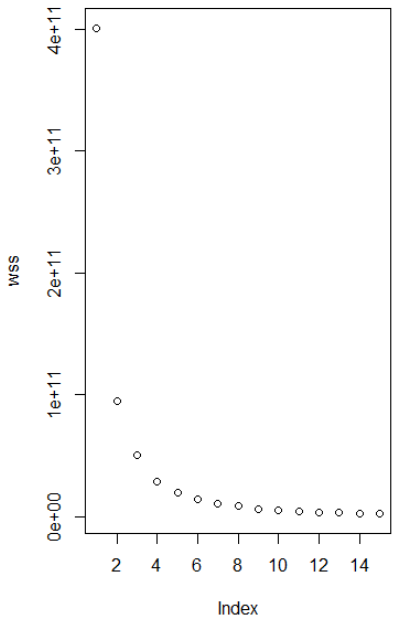

Figure 29. WSS with some potential clusters (Index).

This plot shows 14 potential groups, and two of them are significantly different from each other, the first and the second. These clusters should be able to demonstrate good homogeneity within.

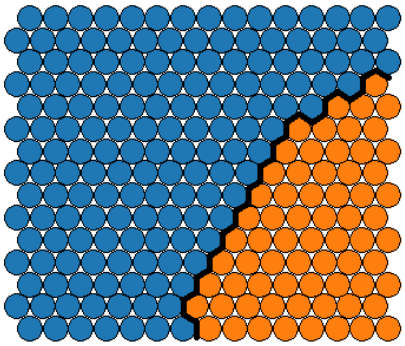

Figure 30. Two groups which is much more different from each other.

It is established that the training data consists of a lot of negative and less positive examples, so it was expected that one data cluster would be much better represented than another when being more separated. Because the SOM approach requires clean data the problem can occur when different variables have different units or data variables are posed in the various distance. It can be assumed that SOM does not perform well in this case because the data was not pre-processed, cleaned or rebalanced.

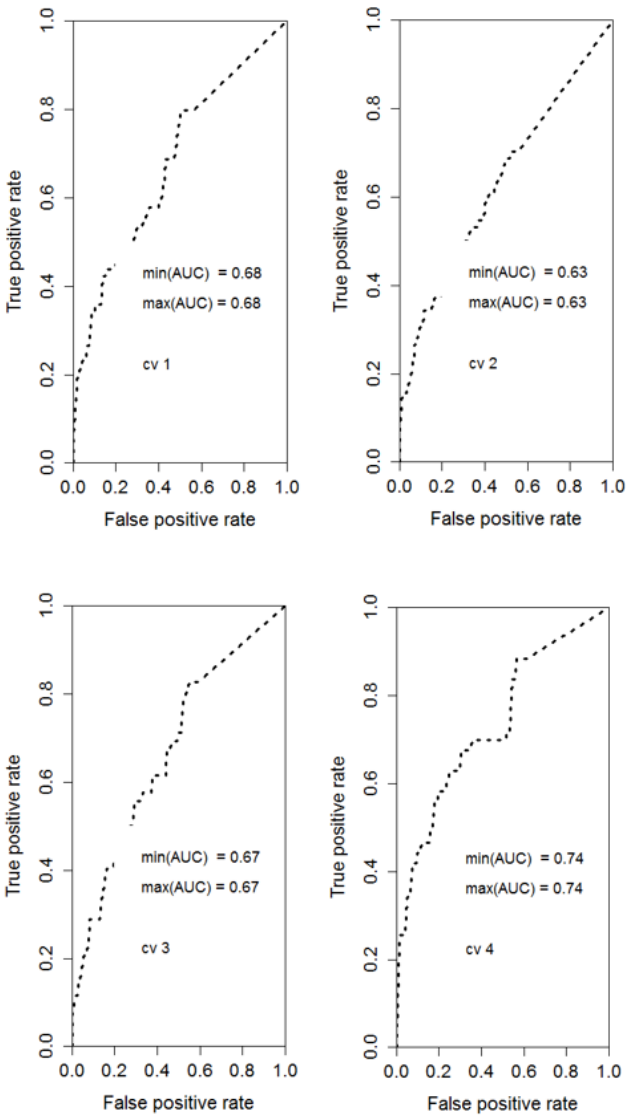

Figure 31. Self-organized map CV AUC and ROC results.

Table 15. Cross Validation results of a Self-organized map model.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
CV1	0,062	0,997	0,003	0,937	0,038	0,995	0,470	0,004	1,001	0,677
CV2	0,078	0,997	0,003	0,922	0,044	0,994	0,462	0,005	1,001	0,630
CV3	0,077	0,996	0,004	0,923	0,033	0,995	0,463	0,004	1,000	0,671
CV4	0,069	0,996	0,003	0,930	0,026	0,995	0,467	0,004	1,000	0,735
CV5	0,118	0,991	0,009	0,882	0,019	0,989	0,446	0,010	1,000	0,679
Mean	0,080	0,996	0,004	0,919	0,032	0,994	0,461	0,006	1,000	0,679

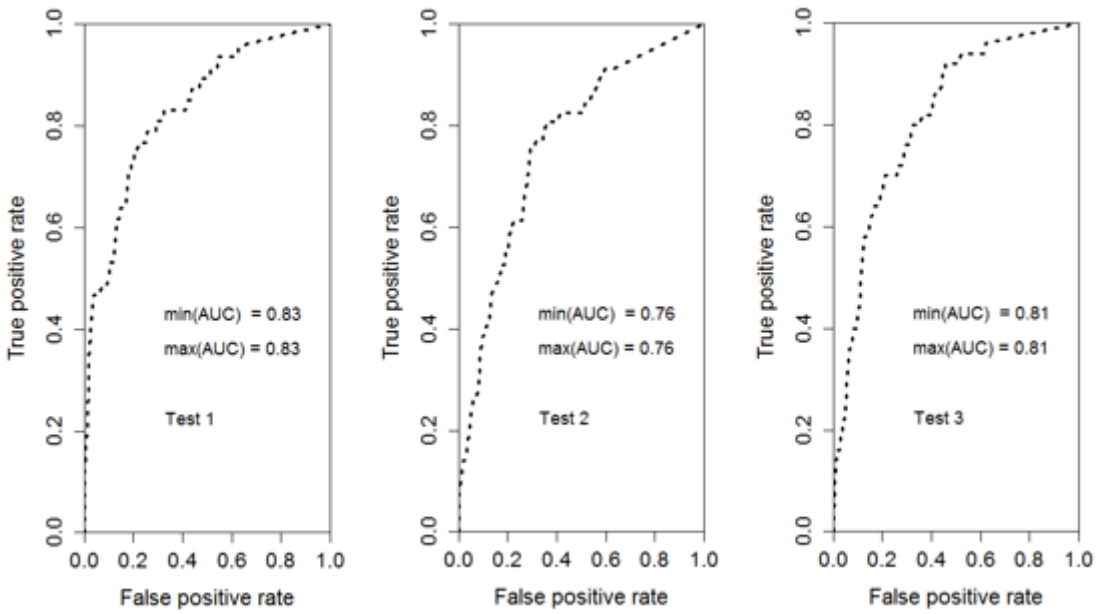

Figure 32. AUC and ROC of the Self-organized map evidence.

Table 16. Confusion matrix of a Self-organized map.

Num.	**TP**	**FP**	**TN**	**FN**
T1	7	122	28301	40
T2	5	111	28302	52
T3	3	122	28298	47

Table 17. Test results of the Self-organized map.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
T1	0,149	0,996	0,004	0,851	0,054	0,994	0,428	0,006	1,000	0,832
T2	0,088	0,996	0,004	0,912	0,043	0,994	0,458	0,006	1,000	0,762
T3	0,060	0,996	0,004	0,940	0,024	0,994	0,472	0,006	1,001	0,810
Mean	0,099	0,996	0,004	0,901	0,040	0,994	0,452	0,006	1,001	0,800

4.3.3 Conclusion

Previously the author examines the Self-Organized map classification model on provided dataset. The dataset was not scaled, without any pre-processing and highly unbalanced. First of all, the aim was to analyse training dataset; the R Kohonen library was used. Furthermore, the SOM model results were plotted. Training progress Figure 25, without

CV, shows that the mean distance was minimised after ~50 iterations. Also, the plot indicated that SOM was neither too small nor too big and it was a convergence to a minimum. Counts Figure 26 show that SOM model is relatively uniform, which means that map quality is not too bad. Next Figure 27 shows the distance between nodes; the researcher can use this plot to clarify the clusters. Four groups can be highlighted from this plot. Figure 28 indicates that better is represented only one group that has the smallest distance. Next, the map was divided into groups, and Figure 29 show 14 potential clusters and only two of them were used due to being highly different from each other. That was plotted in Figure 30.

Secondly, the author had plotted ROC with AUC of cross-validation. Thirdly, a trained model was created in order to use three different test data sets for testing. All results for both methods have been presented in the tables above. The result tables indicate the mean true positive rate, for CV, that shows the smaller effect, however, the false positive rate is similar in both cases. From AUC results the conclusion can be made that classes were not very well separated.

Results from the Table 15 show that CV1 shows the worst TPR than the other tests. FNR for CV4 is bigger than in the others cross-validation folds. in others cross-validation folds. In all cases, the accuracy was good but not enough.

In conclusion, the ROC curve and AUC help to understand the impact of a chosen classification threshold visually. ROC will indicate what false positive rate should be expected according to the true positive rate. By comparing the accuracy results and error rate for each cross-validation subsets, it is possible to explain the quality of the data set being used and understand that the model is susceptible to the difference in the data. Furthermore, by comparing the accuracy results and error rates in the independent tests results the performance of the model can be interpreted as a good.

5 Summary

This Thesis shows the research of different machine learning algorithms, the way how researchers could use algorithms in the classification problem. As well as how the data analysts or engineers can deal with unbalanced data. Three different machine learning algorithms, namely, Logistic Regression, Decision Tree, and Self-Organized map were used in order to train a model for the classification task. The trained model should be capable of classifying the transactions as fraudulent or genuine. Also, the result of this Thesis is a code of a program that can learn from provided dataset in three different manners. The code of this program can be found at the end of the Thesis in the Appendix Section. In this Chapter, the summary of main results of the Thesis will be given, and the future research directions will be presented.

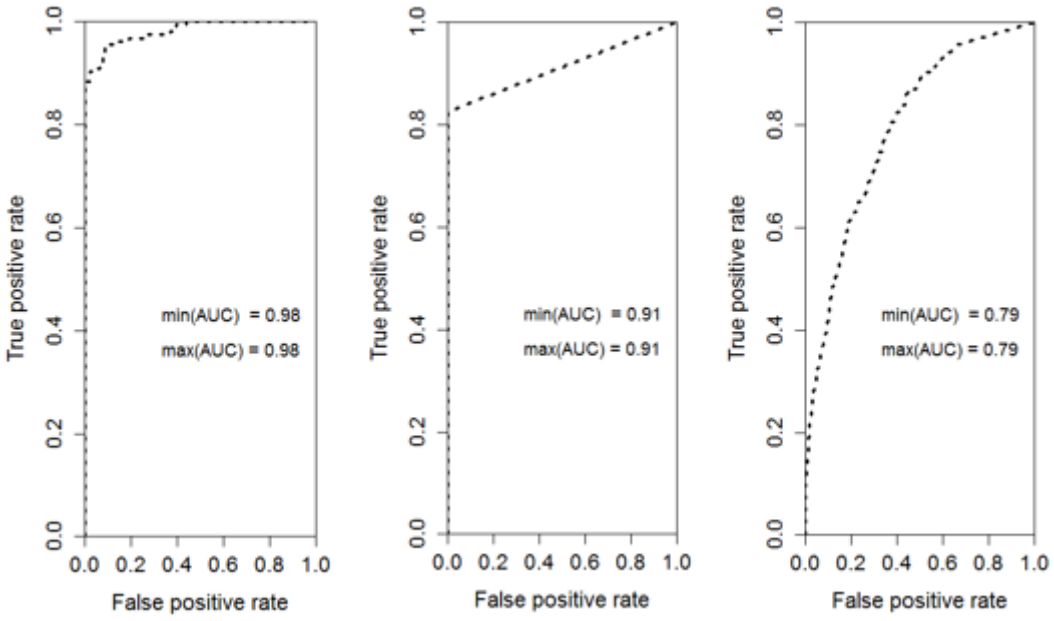

Figure 33. Models mean AUC and ROC results. First Logistic Regression, second Decision Tree and the last one is a Self-organized map.

Table 18. Mean classification results from each model.

Num.	**TPR**	**TNR**	**FPR**	**FNR**	**Prec.**	**Acc.**	**BER**	**ERR**	**MSE**	**AUC**
Logit	0,737	0,999	0,0002	0,263	0,867	0,999	0,131	0,0007	1,000	0,981
DT	0,821	0,999	0,0002	0,175	0,901	0,999	0,087	0,0005	0,022	0,912
SOM	0,099	0,996	0,004	0,901	0,040	0,994	0,452	0,006	1,001	0,800

Therefore, according to the table above the conclusion can be drawn that the DT model performs better with unbalanced data without any pre-processing of that data on the fraudulent transactions classification problem. However, ROC shows that it could be reasonable for Logistic Regression to raise the TPR for getting a better result with a minimum increase in FPR. For example, with TPR 0.903 the FPR of 0.034 was gained. That is more of a business decision, whether it would rather minimise False Positive Rate or maximise True Positive Rate.

One of the standard solutions to deal with a classification problem with unbalanced class distribution is to rebalance the classes before training a model. A popular rebalancing approach among the machine learning community is SMOTE oversampling, described on page 19.

In the current research, it was established that in this case, the supervised machine learning method did work well. In general, the generated fraud alerts the individual researchers to check and annotate alerted transactions as negative (genuine) or positive (fraudulent). This kind of feedback from individual researchers provides recently supervised samples that may be very instructive so the data analysts can use those examples in supervised algorithms.

To conclude, the tests results show good accuracy on trained models. However, it is important to consider that these results are good because the unbalanced data that has a lot of negative examples was used. In the future work the author will try to rebalance the data set with SMOTE method and also will try to rearrange the data and after that will use the pre-processed information in the same learning algorithms that were examined in this Thesis. It would be reasonable to combine two different, supervised and unsupervised, methods for the fraud classification problem. The author also suggests the implementation of the Abraham Wald sequential sampling test for probability ratio that will help to understand a quality of predictions on separate examples. As another great approach for data analysis, the author suggests trying the PRIDIT method. PRIDIT method is using principal components/factor analysis for data clustering.

References

[1] LegalDictionary.net, "Legal Dictionary," 2017. [Online]. Available: https://legaldictionary.net/fraud/.

[2] N. Report, "The Nilson Report," The Nilson Report, 01 01 2010. [Online]. Available: www.nilsonreport.com. [Accessed 01 02 2017].

[3] A. D. Pozzolo, "Adaptive Machine Learning for," 2015.

[4] D. M. Button, "City of London Police," 2016. [Online]. Available: https://www.cityoflondon.police.uk/news-and-appeals/Pages/Academic-report-indicates-cost-of-Fraud-to-the-UK-is-%C2%A3193bn-a-year.aspx.

[5] D. Heckerman, "A tutorial on Learning with Bayesian," Technical report, MSR-TR-95-06. Microsoft research, Redmond, WA 98052, 1995.

[6] t. f. e. Wikipedia, "Supervised learning," [Online]. Available: https://en.wikipedia.org/wiki/Supervised_learning.

[7] A. Ng, "Unsupervised Learning," [Online]. Available: https://www.coursera.org/learn/machine-learning/supplement/1O0Bk/unsupervised-learning.

[8] Wikipedia, "Overfitting," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Overfitting.

[9] A. Ng, "Diagnosing Bias vs. Variance," [Online]. Available: https://www.coursera.org/learn/machine-learning/supplement/81vp0/diagnosing-bias-vs-variance.

[10] K. Markham, "ROC curves and Area Under the Curve explained," Data school, 19 11 2014. [Online]. Available: http://www.dataschool.io/roc-curves-and-auc-explained/. [Accessed 03 25 2017].

[11] Alejandro Correa Bahnsen, Djamila Aouada and Bjorn Ottersten, "Example-Dependent Cost-Sensitive Logistic," in *13th International Conference on Machine Learning and Applications*, Luxembourg, 2014.

[12] A. Vidhya, "A Complete Tutorial on Tree Based Modelling from Scratch (in R & Python)," Analytics Vidhya, 2013. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#one. [Accessed 2017].

[13] Wikipedia, "Self-organizing map," Wikipedia, 31 March 2017. [Online]. Available: https://en.wikipedia.org/wiki/Self-organizing_map. [Accessed 31 March 2017].

[14] N. Report, "Nilson Report," Nilson Report, 01 01 2010. [Online]. Available: https://www.nilsonreport.com/index.php. [Accessed 01 02 2017].

Appendix 1 – Logistic Regression code

Calculating Logistic Regression model and plot the ROC fro each Cross Validation result. In the end, mean error and AUC are shown.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
calculateModelAucAndShowRoc(train_data, "log")
```

Next code denotes Logistic Regression calculation without CV with three independent tests. Also, error and AUC numbers are shown and plot the ROC and the predictions.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
test_data <- readRDS(paste(project_path, "/creditcard_test.Rdata", sep
= ""))

logistic <- performLogisticRegression(train_data, test_data)
model <- logistic@model
predictions <- logistic@predictions

test1 <- test_data[1:28470,]
test2 <- test_data[28471:56940,]
test3 <- test_data[56941:85410,]
testAndPlot(model, train_data, list(test1, test2, test3))

showErrorAndAuc(test_data, predictions)
plotRoc(predictions, test_data[, 'Class'], "")
table(predicted = predictions > .5, actual = test_data[, 'Class'])
```

Appendix 2 – Decision Tree code

Calculating Decision Tree model and plot the ROC fro each Cross Validation result. At the end, mean error and AUC are shown.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
calculateModelAucAndShowRoc(train_data, "dt")
```

Next code denotes Decision Tree calculation without CV with three independent tests. Also, error and AUC numbers are shown and plot the ROC and visualized DT in a different manner.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
test_data <- readRDS(paste(project_path, "/creditcard_test.Rdata", sep
= ""))

dt <- performDecisionTree(train_data, test_data)
model <- dt@model
predictions <- dt@predictions

test1 <- test_data[1:28470,]
test2 <- test_data[28471:56940,]
test3 <- test_data[56941:85410,]
testAndPlot(model, train_data, list(test1, test2, test3))

printcp(model)
plotcp(model)
summary(model)

confusionMatrix(predictions, test_data$Class)
plot(predictions, xlab = "Class", ylab = "Examples")

predictions <- as.numeric(predictions)
showErrorAndAuc(test_data, predictions)
plotRoc(predictions, test_data[, 'Class'], "")

prp(model, varlen = 5)
fancyRpartPlot(model)

plot(model, uniform = TRUE)
text(model, use.n = TRUE, all = TRUE, cex = .8)
```

Appendix 3 – Self-organized map code

Calculating Self-organized map model and plot the ROC fro each Cross Validation result. At the end, mean error and AUC are shown.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
calculateModelAucAndShowRoc(train_data, "som")
```

Next code denotes Self-organised map calculation without CV with three different tests. Also, error and AUC numbers are shown and plot the ROC. Plated different results to the SOM model. Prediction results are shown at the end of the script.

```
train_data <- readRDS(paste(project_path, "/creditcard_train.Rdata",
sep = ""))
test_data <- readRDS(paste(project_path, "/creditcard_test.Rdata", sep
= ""))

som <- performSom(train_data, test_data, 15, 15)
model <- som@model
predictions <- som@predictions

test1 <- test_data[1:28470,]
test2 <- test_data[28471:56940,]
test3 <- test_data[56941:85410,]
testAndPlot(model, train_data, list(test1, test2, test3))

plot(model, type = "changes", main = "")
plot(model, type = "count", main = "", palette.name = blueRed)
plot(model, type = "quality", palette.name = blueRed, main = "")
plot(model, type = "dist.neighbours", main = "", palette.name =
blueRed)

cluster(model, 15, 2)

showErrorAndAuc(test_data, predictions)
plotRoc(predictions, test_data[, 'Class'], "")

table(test_data$Class, classmat2classvec(predictions))
table(predicted = classmat2classvec(predictions), actual = test_data[,
'Class'])
threshold <- max(predictions) / 2
table(predicted = predictions > threshold, actual = test_data[,
'Class'])
```

Appendix 4 – Generic code

Plot the ROC function.

```
plotRoc <- function(predictions, data, txt) {

    pred <- prediction(predictions, data)
    perf <- performance(pred, "tpr", "fpr")

    par(mar = c(5, 5, 2, 2), xaxs = "i", yaxs = "i", cex.axis = 1.3,
cex.lab = 1.4)
    plot(perf, col = "black", lty = 3, lwd = 3)

    auc <- performance(pred, "auc")
    auc <- unlist(slot(auc, "y.values"))

    minauc <- min(round(auc, digits = 2))
    maxauc <- max(round(auc, digits = 2))
    minauct <- paste(c("min(AUC)  = "), minauc, sep = "")
    maxauct <- paste(c("max(AUC) = "), maxauc, sep = "")

    legend(0.2, 0.5, c(minauct, maxauct, "\n", txt), border = "white",
cex = 1.1, box.col = "white")
}
```

Show error and AUC value.

```
showErrorAndAuc <- function(dataTest, predictions) {
    yLabel <- c('Class')

    err <- rmse(as.numeric(dataTest[, yLabel]), predictions)
    auc <- auc(dataTest[, yLabel], predictions)

    print(paste('MSE:', err))
    print(paste('AUC:', auc))
}
```

Next is the main function which is used for model creation.

```
calculateModelAucAndShowRoc <- function(data, algorithm = "log") {
    require(xgboost)
    require(Metrics)

    CVs <- 5
    cvDivider <- floor(nrow(data) / (CVs + 1))
    indexCount <- 1
    yLabel <- c('Class')
    predictors <- names(data)[!names(data) %in% yLabel]
    lsErr <- c()
    lsAUC <- c()

    for (cv in seq(1:CVs)) {
        cvTxt <- paste('cv', cv)
        print(cvTxt)
        dataTestIndex <- c((cv * cvDivider):(cv * cvDivider +
cvDivider))
        dataTest <- data[dataTestIndex,]
        dataTrain <- data[-dataTestIndex,]

        if (algorithm == 'log') {
            model <- glm(dataTrain[, yLabel] ~ ., family =
binomial(logit), data = data.frame(dataTrain[, predictors]))
            predictions <- predict(model, data.frame(dataTest[,
predictors]), outputmargin = TRUE)

            print(summary(dataTest$Class))
            print(table(predicted = predictions > .5, actual =
dataTest[, yLabel]))
        }
        else if (algorithm == 'dt') {
            require(rpart)
            require(caret)

            model <- rpart(dataTrain[, yLabel] ~ ., data =
data.frame(dataTrain[, predictors]), method = "class")
            predictions = predict(model, dataTest, type = "class")
            print(confusionMatrix(predictions, dataTest[, yLabel]))
            predictions <- as.numeric(predictions)
        }
        else if (algorithm == 'som') {
            require(kohonen)

            x <- dataTrain[, predictors]
            xMatrix <- as.matrix(x)

            grid <- somgrid(xdim = 15, ydim = 15, topo = "hexagonal")
            model <- som(xMatrix,
                    grid = grid,
                    rlen = 100,
                    alpha = c(0.05, 0.01),
```

```
                    keep.data = TRUE,
                    n.hood = "circular")

            cluster(model, 15, 2)

            predictions <- predict(
                model,
                newdata = as.matrix(dataTest[, predictors]),
                trainX = model$data,
                trainY = as.numeric(as.vector(dataTrain[, yLabel])))

            predictions <- predictions$prediction

            threshold <- max(predictions) / 2
            print(table(predicted = predictions > threshold, actual =
dataTest[, yLabel]))
        }

        err <- rmse(as.numeric(dataTest[, yLabel]), predictions)
        print(paste('MSE: ', err))

        auc <- auc(dataTest[, yLabel], predictions)
        print(paste('AUC: ', auc))

        plotRoc(predictions, dataTest[, yLabel], cvTxt)
        readkey()

        lsErr <- c(lsErr, err)
        lsAUC <- c(lsAUC, auc)

        gc()
    }
    print(paste('Mean MSE:', mean(lsErr)))
    print(paste('Mean AUC:', mean(lsAUC)))
}
```

Next is the test function for model.

```
testAndPlot <- function(model, trainData, testDatas) {
    yLabel <- c('Class')
    predictors <- names(trainData)[!names(trainData) %in% yLabel]
    cls <- class(model)

    for (i in 1:length(testDatas)) {
        txt <- paste('Test', i)
        print(txt)

        if (cls[1] == "glm") {
            predictions <- predict(model, data.frame(testDatas[[i]][,
predictors]), outputmargin = TRUE, type = "response")

            print(summary(testDatas[[i]][yLabel]))
            print(table(predicted = predictions > .5, actual =
testDatas[[i]][, yLabel]))
        }
        else if (cls == 'rpart') {
            predictions = predict(model, testDatas[[i]], type =
"class")
            print(confusionMatrix(predictions, testDatas[[i]][,
yLabel]))

            predictions <- as.numeric(predictions)
        }
        else if (cls == 'kohonen') {
            predictions <- predict(
                model,
                newdata = as.matrix(testDatas[[i]][, predictors]),
                trainX = model$data,
                trainY = as.numeric(as.vector(trainData[, yLabel])))

            predictions <- predictions$prediction

            threshold <- max(predictions) / 2
            print(table(predicted = predictions > threshold, actual =
testDatas[[i]][, yLabel]))
        }

        showErrorAndAuc(testDatas[[i]], predictions)
        plotRoc(predictions, testDatas[[i]][, yLabel], txt)
        readkey()
    }
}
```

Next is the function for models tests.

```
makeTestsOnModels <- function(testData, modelList, testCount = 1,
trainData = NULL) {
    yLabel <- c('Class')
    predictors <- names(testData)[!names(testData) %in% yLabel]

    results <- as.data.frame(testData[, 31:31])
    columns <- yLabel
    modelsCount <- length(modelList)

    for (i in 1:modelsCount) {
        model <- modelList[[i]]
        cls <- class(model)

        for (j in 1:testCount) {

            if (cls[1] == 'glm') {
                column <- paste(cls[1], i, 'test', j)
                predictions <- predict(model, data.frame(testData[,
predictors]), outputmargin = TRUE, type = "response")
            }
            else if (cls == 'rpart') {
                column <- paste(cls, i, 'test', j)
                predictions = predict(model, testData, type = "class")
            }
            else if (cls == 'kohonen') {
                column <- paste(cls, i, 'test', j)
                predictions <- predict(
                    model,
                    newdata = as.matrix(testData[, predictors]),
                    trainX = model$data,
                    trainY = as.numeric(as.vector(trainData[,
yLabel])))

                predictions <- predictions$prediction
            }
            columns <- c(columns, column)
            results <- cbind(results, column = predictions)
        }
    }
    names(results) <- columns
    return (results)
}
```