Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Project

Fraud Detection in Unlabeled Payment Card Transactions

Josef Vonášek

Supervisor: Štěpán Kopřiva, MSc.

Study Program: Open Informatics

Branch of Study: Computer and Information Science

May 25, 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Vonášek Josef**	Personal ID number:	**456932**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Fraud Detection in Unlabeled Payment Card Transactions

Bachelor’s thesis title in Czech:

Detekce podvodů v platbách kreditní kartou

Guidelines:

1) Study the existing principles of automated fraud detection on transactions, mainly on credit card transactions.
2) Study anomaly detection techniques for unlabeled data sets (for non-time series datasets) which might be used to solve the fraud detection in transactions problem.
3) Design an algorithm for detecting the fraudulent transactions in provided real data set of card transactions with low number of fraudulent transactions in the dataset (this is one of the typical signs in the fraud detection tasks).
4) Implement the algorithm from 3).
5) Evaluate the solution quality by comparing your solution to other existing algorithms.

Bibliography / sources:

[1] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi - Calibrating Probability with Undersampling for Unbalanced Classification - In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015
[2] Varun Chandola, Arindam Banerjee and Vipin Kumar - Anomaly Detection: A Survey - ACM Computing surveys, 2009
[3] Manoranjan Pradhan, Sateesh Kumar Pradhan, Sudhir Kumar Sahu - Anomaly Detection Using Artificial Neural Network - International Journal of Engineering Sciences & Emerging Technologies, 2012

Name and workplace of bachelor’s thesis supervisor:

Štěpán Kopřiva, MSc., Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **12.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Štěpán Kopřiva, MSc.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student’s signature

Aknowledgements

I am very grateful to my supervisor MSc. Štěpán Kopřiva. I highly value his guidance that led me in the right direction. I would also like to thank Tomáš Witzany and Mikuláš Krupička for their valuable help. Finally I would also like to express my gratitude to my family and friends.

Declaration

Abstract

The developments in the Information Technology slowly shift a portion of our every day life into an electronical form. The global nature of Internet along with its anonymity make it an ideal instrument for committing a fraud, resulting in huge financial losses every year. Although prevention is the best way to reduce fraud, with enough time, fraudsters will usually find ways to circumvent these measures. Methodologies for the detection of fraud are therefore essential if we are to catch fraudsters once prevention has failed.

In this work we present an unsupervised method for credit card fraud detection, based on projecting the dataset into low dimensional representation and consecutive reconstruction. The fraud detection system consists of two components, namely an autoencoder neural network and a feature scaling technique. Autoencoder is used to compute the reconstruction error matrix, which is then further utilized in the feature scaling and selection component. An updated reconstruction error is thereafter used to detect fraudulent transactions.

The experimental results of proposed architecture are presented and discussed.

Abstrakt

Vývoj informačních technologií pomalu posouvá část našeho každodenního života do elektronické podoby. Globální povaha internetu spolu s jeho anonymitou z něj činí ideální nástroj pro páchání podvodů, což každoročně vede k obrovským finančním ztrátám. Přestože prevence je nejlepším způsobem, jak podvodům zabránit, s dostatkem času útočníci obvykle najdou způsoby, jak obejít tato opatření. Metody pro odhalení podvodu jsou tedy nezbytné, pokud se chceme, poté co selže prevence, s podvodníky vypořádat.

V této práci předkládáme *unsupervised* metodu pro detekci podvodů kreditní kartou, založenou na zobrazení dat do prostoru o nízké dimenzi. Klasifikátor se skládá ze dvou komponent, jmenovitě autoenkodéru (neuronové sítě) a metody pro selekci atributů. Autoenkodér je použit k vypočtení rekonstrukční chyby ve formě matice, která se dále použije pro selekci atributů. Aktualizovaná rekonstruční chyba je poté použita k odhalení podvodných transakcí.

Experimentální výsledky tétoarchitektury jsou prezentovány a diskutovány.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the arrival of digitalization, internet sales became one of the essential business methods for most companies. Because of that, the usage of credit card payments has dramatically increased during the last two decades. Although there are many ways to commit a fraud, online attacks trough the Internet are one of the most popular. Because of its global nature, it is easy for the user to become completely anonymous, hiding its identity and location, which makes it ideal media for fraudsters. As a result, credit card industry is a very attractive target for these individuals and their behavior causes huge financial losses each year.

Credit card fraud detection is a heavily studied branch of anomaly detection. The goal is to detect the highest number of fraudulent transactions possible, with reasonable number of false positives (misclassified normal transactions). If we know the cost of verifying, whether a transaction is fraudulent, there is also very easy way to directly quantify the optimal performance. Because of the confidentiallity issues, one of the most limiting factor for research on financial fraud detection is the lack of publicly available datasets. The dataset used in this thesis is no exception, since its features have been transformed by PCA in order to anonymize all card transactions. Thus common techniques like feature aggregation cannot be used.

We present a fraud detection method based on autoencoder neural network. Our method is completely unsupervised and includes our own feature selection and scaling algorithm. We examine the existing techniques focused on anomaly detection and provide a mathematical definition for the problem of credit card fraud detection. We discuss our proposed solution and examine its performance on a real credit card transactions dataset from an European bank. The expiremntal results show that the performance of our solution is comparable to existing unsupervised algorithms with supervised feature selection.

1.1 Goals of thesis

We can summarize our goals as follows:

1. **Study the existing principles of automated fraud detection on transactions, mainly on credit card transactions.**

In order to fully understand the fundamentals of automated fraud detection, we need to closely examine the concepts behind the detection methods used in the existing work, with the focus on credit card transactions.

2. **Study anomaly detection techniques for unlabeled data sets (for non-time series datasets) which might be used to solve the fraud detection in transactions problem.**

   To design an effective detection algorithm, we need to get familiar with the existing methods used for anomaly detection. Knowing their advantages and disadvantages will help us choose the right algorithm for the task given.

3. **Design an algorithm for detecting the fraudulent transactions in provided real data set of card transactions with low number of fraudulent transactions in the dataset (this is one of the typical signs in the fraud detection task).**

   We will design an unsupervised fraud detection algorithm based on the knowledge we attained (goals 1 and 2). To examine whether the algorithm could be used in real world scenario, we will use a provided real data set of card transactions.

4. **Implement the algorithm from** *3*.

   We need to implement our algorithm to examine its performance.

5. **Evaluate the solution quality by comparing your solution to existing algorithms**.

   To verify the efficiency and quality of our solution, we will evaluate its performance by using a labeled dataset and several performance metrics. By comparing the solution to existing algorithms, we will get a better understing of the effectivity of our solution in relation to other methods.

1.2 Chapter description

The thesis has the following structure:

- **Chapter 2** provides an overview of related work on anomaly detection. It describes both supervised and unsupervised techniques. This chapter addresses the first two goals of our thesis.
- **Chapter 3** provides a technical background on the spectral technique we will use in our detection system.
- **Chapter 4** introduces and describes the card transactions dataset.
- **Chapter 5** is concerned with the mathematical formulation of our task - detection of fraudulent behavior in credit card transactions dataset.
- **Chapter 6** presents our solution to the problem. An unsupervised fraud detection system based on autoencoder neural network along with pretraining and feauture selection. This chapter addresses the third goal of our thesis.

- **Chapter 7** describes the system implementation in detail, justyfing some of the choices for hyperparameters we have made. Pretraining and feature selection are thoroughly explained. This chapter addresses the fourth goal of our thesis.
- **Chapter 8** is devoted to the analysis of performance of our credit card fraud detection system. We address the last goal of our thesis in this chapter.

Chapter 2

Related work

In this chapter, we present an overview of work related to our topic. In order to provide a better picture we also include work from closely related domains like *insurance fraud detection*, *intrusion detection* and other areas that deal with anomaly detection. Although our work explores the area of unsupervised fraud detection, the majority of research in credit card fraud detection deals with supervised methods. Hosseinpour et al. [19] even proposed a system, which incorporates methods from both categories.

2.1 Supervised techniques

This section contains an overview of techniques that are used mostly in supervised settings (labeled dataset is required in order to train the classifier). But some of them are robust enough to be also applied as semisupervised method (training dataset contains only non-anomalous data points) or even (with small alternation) as an unsupervised method in the case of support vector machines (SVM).

2.1.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are inspired by the constitution of the animal brain where each neuron is modeled as a nonlinear function applied to a weighted sum of the neuron inputs. Despite the ANN popularity, comparative studies suggest that simpler techniques like Bayesian Belief Networks (BBN) often perform better. This might in part explained by the difficullty of finding the right paramaters (which are plenty choose from) and also by their tendency to overfit.

In year 1994 Ghosh and Reilly [15] made a study for the Mellon Bank. They compared the bank's rule based detection procedure which was checked against all portfolio activy. They concluded that ANN could reduce their losses caused by credit card fraud by 20% - 40%. Syeda et al. [30] developed a fuzzy parallel granular neural network (GNN) in order to speed up data mining and knowledge discovery process. Dorronsoro et al. [10] developed a system named *Minerva* for a company which provides a card payment scheme in Spain. They have used non-linear discriminant analysis neural models with the usual multilayer perceptron (MLP) architecture. Aleskerov et al. [6] created CARDWATCH, a neural network based data mining system for credit card fraud detection.

2.1.2 Decision Trees

Decision trees are tree structured rule based techniques where each inner node performs a test on one attribute and leafs perform the final classification. Their advantage is trivial explanation, fast classification and the possibility to work with discrete attributes.

Sahin and Duman [27] have made one of the first comparative study of Decision Trees and SVM on a nonsythethic dataset containing credit card transactions. Their result demonstrated that although both methods have similar detection accuracy, decision tree methods (particularly C&RT implementation) caught up to 33% more frauds.

2.1.3 Bayesian Belief Networks

The Bayesian belief network (BBN) represents a set of random variables and their conditional independencies using a directed acyclic graph (DAG), in which nodes represent random variables and missing edges encode conditional independencies between the variables.

Maes et al. [23] have designed an automated credit card fraud detection system comparing BBN to ANN. They concluded that although ANN was faster in classification, BBN outperformed ANN during the training phase and gave overall better results.

Viaene et al. [32] used AdaBosted naive bayes (special version of BBN where features are independent) for insurance fraud detection. Compared to unboosted and boosted naive Bayes, the framework showed slightly better accuracy and AUC but clearly improved on the cross entropy and Brier scores.

2.1.4 Artificial Immune System

The artificial immune system (AIS) comprises promising techniques inspired by an animal immune system, which has the ability to distinguish internal cells and molecules of the body from foreign pathogens, so called self and non-self respectively. This behavior is very similar to the one which is expected from fraud detection system where attacker has a malicious intent.

Gadi et al. [14] made a comparative study of five classification methods (Decision Trees, Naive Bayes, Bayesian Networks, Neural Network and Artificial Immune System) in a credit card fraud detection domain. Their results showed that NN surprisingly scored the worst while AIS produced the best classifiers of all the five methods.

Hosseinpour et al. [19] explored the area of unsupervised intrusion detection with a combination of AIS and unsupervised method. The output of DBSCAN clustering is used to feed the training data for the adaptive immune system as online and real-time training data. The system is tested in centralized and distributed mode and is able to provide a fast response to previously unknown anomalies.

2.1.5 Support Vector Machines

Support Vector Machines (SVM) seek such a hyperplane that maximizes its distance between classified data points. Its training speed grows rapidly with the number of training samples and therefore it is not suitable for large amount of data. Since this technique is robust to outliers, it can be also used for unsupervised anomaly detection.

Hu et al. [20] used robust SVM in the area of intrusion detection. They showed that RSVM are robust to overfitting caused by noisy data and also produces significantly less support vectors, therefore improving classification performance.

Amer at al. used an enhanced one-class SVM [7] for unsupervised anomaly detection. They have examined two approaches which are insensitive outlier - *robust* and *eta* one-class SVM. They conclude that the eta one-class SVM has shown the greatest potential. It was capable of maintaining the sparsity of the SVM solution, while performing best on most datasets.

2.2 Unsupervised techniques

This section contains an overview of unsupervised techniques. Thus the dataset does not have to be labeled, but it assumed that anomalous data instances will have very different behavior compared to normal ones. They might be for example located in sparse regions, harder to project into lower dimensional space or they might form their own special clusters. It is also usually required for the dataset to contain only minority of anomalous instances.

2.2.1 Nearest Neighbor techniques

Nearest neighbor (K-NN) work with the assumption that anomalies are rare - they occur in less dense clusters than normal instances. A metric which compares the similarity between individual data is therefore needed, which is also the biggest weakness of this technique as well as slow classification, which is quadratic in case of the simplest implementation. Pruning is often utilized to accelerate the calculation, because a large part of the normal instances is easily recognizable.

It has been applied by Viane et al. [31] in Automobile Insurance Claim Detection. They report that although K-NN attained one of the best performance, with increasing number of data instances the computational overhead and memory requirements for storing the distance matrix would become a burden.

2.2.2 Clustering techniques

Clustering methods are unsupervised by their nature and since their primary goal is not anomaly detection, they can be divided into three distinct groups based on the assumptions they make about anomaly behavior:

- Anomalies do not belong to any cluster
- Anomalies occur at the edges of clusters
- Anomalies form sparse and / or small clumps

There are several clustering algorithms that do not force every data instance to belong to a cluster which could be used, such as DBSCAN (Ester et al. [12]), ROCK (Guha et al. [17], and SNN clustering (Ertoz et al. [11]). These methods are advantageous, because they do not make any assumptions about anomaly behavior themselves. Therefore the anomalies can be either part of cluster. In which case the technique is robust to noise in data. Or they do not have to belong to any cluster, in which case anything outside of cluster is treated as an anomaly.

Smith et al. [29] studied Self-Organizing Maps (SOM), K-means and Expectation Maximization (EM) clustering methods. They worked with the second assumption, that anomalies occur at the edges of clusters, and they calculated the suspicion score as the distance from its cluster centroid.

It is not uncommon to use clustering in combination with other methods. We have already mentioned Hosseinpour et al. [19] who used pipeline of DBSCAN and Artificial Immune System. Dheepa et al. [9] have used a clustering method along with gaussian mixture model and bayesian networks for credit card fraud detection.

2.2.3 Isolation based techniques

Although isolation based techniques are not yet widely used, they definitely deserve a mention, because of their special features. As usual they build on the assumption that anomalies are few and have unusual attribute values. In comparison to density based techniques, they do not require a distance metric. In fact the only requirement is the possibility to split the data into multiple groups, thus nonnumerical attributes can be present in the dataset.

iBAT is an example of an isolation based technique developed by Zhang et al. [33] for detection of adverse and malicious events based on recorded taxi driver GPS trajectories. Liu et al. [22] explored a technique called Isolation Forest (iForest) where anomalies are instances with short average path lengths on the iTrees, which randomly isolate data instances from each other. The characteristic of iForest enables it to exploit subsampling and the detection phase is linear in time completexity and has minimal memory requirements [22].

2.2.4 Spectral techniques

These techniques try to project the dataset into a space with less dimensions than the original has. The suspicion score is then usually calculated as the reconstruction error with the assumption that anomalies are harder to compress. All techniques from this category require all attributes to be numeric and some distance metric is usually needed to measure the reconstruction error, although not necessarily [8].

Principal component analysis (PCA) is very common dimension reduction method and it has been therefore used by several authors. Shyu et al. [28] in the domain of intrusion detection applied PCA with custom distance measure similar to the Mahalanobis one. Although Fowler et al. [13] detected anomalies with random projections, they have also used PCA along with multitask Bayesian compressive sensing (MT-BCS) for the dataset reconstruction.

Because PCA cannot pick up on nonlinear trends in the dataset, autoencoders (type of neural networks) have been used in their place. Nicolau et al. [25] used a hybrid of autoencoder of density estimation model to detect network anomaly behavior. They utilized the compressed feature representation from hidden layer as an input to density based anomaly detection technique (they have chosen a single Gaussian and full kernel density estimation).

Variational Autoencoders (VAE) are another type of autoencoders which use reconstruction probability instead of reconstruction error during their training. They have been used for anomaly detection by An et al. [8] with positive results. The advantage of this technique is that, as they state, the reconstruction probability, being probability measure, is a much more objective and principled anomaly score than the reconstruction error.

Chapter 3

Technical Background

We have presented the most commonly used techniques in chapter 2. Because obtaining a labeled dataset in real situation is very difficult, we will focus on unsupervised techniques. After careful consideration we have decided to use autoencoder based solution. The reasoning behind this decision is described in chapter 6.

In this chapter we are going to describe the autoencoder neural network and weight initialization technique named pretraining.

3.1 Autoencoder

An autoencoder is a neural network which attempts to replicate the input as closely as possible at the output. Output layer therefore matches the input layer in number of neurons, while the hidden layer contains less neurons as depicted in Figure 3.1, to force the network into reducing the dimensionality of incoming data, for example by noise reduction.

Let $x \in \mathbb{R}^{\mathrm{N}}$ be an input. Then in the simplest case the autoencoder has only one encoding layer and its hidden representation $h(x) \in \mathbb{R}^{\mathrm{M}}$ is a function:

$$h(x) = \sigma_1(W_1 x + b_1)$$

Where σ_1 is a element-wise non-linear activation function such as rectified linear unit (relu) [4]. $W_1 \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}}$ is a weight matrix, $b_1 \in \mathbb{R}^{\mathrm{M}}$ is a bias vector. The latent representation $h(x)$ (latent representation is a representation of the input data in the hidden layer) is then reconstructed into $\hat{x} \in \mathbb{R}^{\mathrm{N}}$ in the output layer:

$$\hat{x} = \sigma_2(W_2 h(x) + b_2)$$

Where σ_2 is again an activation function such as sigmoid or softmax. $W_2 \in \mathbb{R}^{\mathrm{M}\times\mathrm{N}}$ and $b \in \mathbb{R}^{\mathrm{N}}$ are the weight matrix and bias vector of the output layer.

Autoencoder is trained to optimize the parameters $\{W_1, b_1, W_2, b_2\}$ in order to minimize the average reconstruction error, such as squared error:

$$RE(x, \hat{x}) = ||x - \hat{x}||^2 = ||x - \sigma_2(W_2(\sigma_1(W_1\hat{x} + b_1)) + b_2)||^2$$

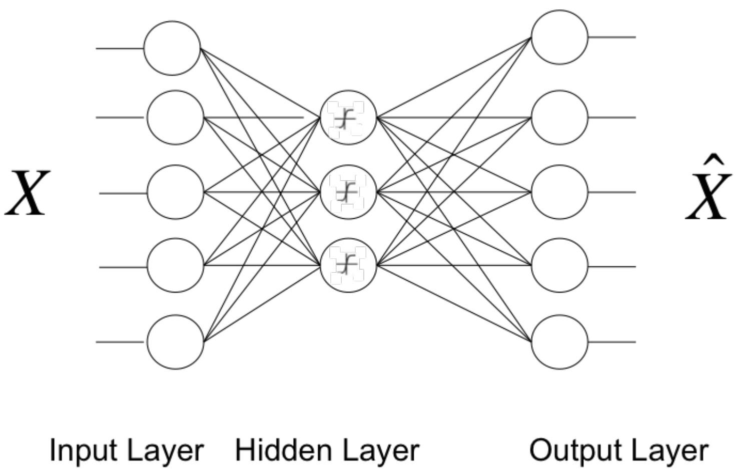

Figure 3.1: Simple autoencoder with one hidden layer.

Stochastic gradient descent (SGD) or other variants of backpropagation are commonly used to train the network. When used for anomaly detection, outliers will be harder to reproduce than normal data and will have therefore high RE values. Thus the anomaly score of data point is simply its reconstruction error and it will be treated as an anomaly / outlier, if its score exceeds certain threshold.

This defenition of autoencoder neural network is easily extensible for arbitrary even number of layers. The latent representation is simply used as an input to another autoencoder with one hidden layer. By stacking multiple autoencoders like this, we obtain new symmetrical autoencoder of arbitrary depth. As you will see, will utilize this fact shortly in the Section 3.2.

3.2 Pretraining

Pretraining is a form of weight initialization method introduced by Hinton et al. [18]. Good weight initialization of multi layered autoencoder is important. With large initial weights, autoencoders typically find poor local minima. With small initial weights, the gradients in the early layers will be tiny, making it infeasible to train autoencoders with many hidden layers. Although Hinton et al. [18] used pretraining with a stack of restricted Boltzmann machines (RBM), their technique can be applied to any symmetrical deep autoencoder.

As mentioned in 3.1, a deep autoencoder with multiple layers can be naturally built from simpler ones. We can utilize this fact by building the deep autoencoder from already trained two layer deep autoencoders presented in 3.1.

As depicted in Figure 3.2, during pretraining a stack of autoencoders is sequentially trained. While first autoencoder learns on the dataset $X \in \mathbb{R}^M$, all following encoders take the latent representation from the hidden layer of the previous autoencoder as an input.

After the training is finished, the stack of autoencoders is unrolled to form a deep autoencoder as is shown in Figure 3.3. By doing this, we can reuse the weights of the trained

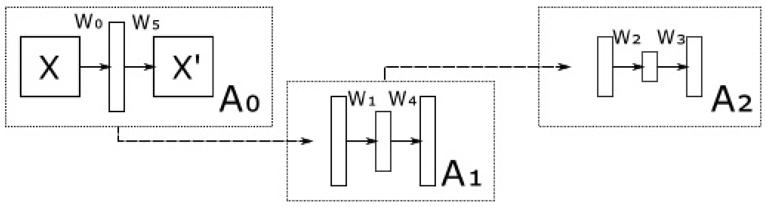

Figure 3.2: Three autoencoders $\{A_0, A_1, A_2\}$, each having only one hidden layer, are trained with an a dataset. The learned latent representations are used as the data for the next autoencoder in the set.

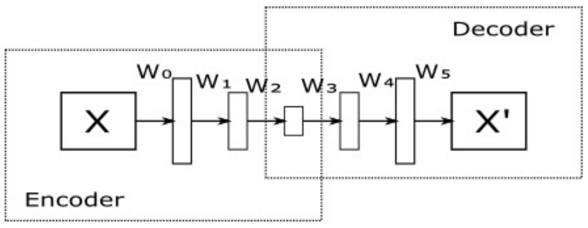

Figure 3.3: The deep autoencoder obtained by unfolding the three autoencoders from Figure 3.2.

autoencoders. As Hinton et al. [18] shows, such initialized weights greatly improve the performance of the final autoencoder.

Chapter 4

Data Description

The dataset contains 284,807 card transactions made by European card holders during the year 2013, publicly avaialable at kaggle [16]. The number of fraudulent transactions accounts only for 0.172% of all transactions.

Due to confidentiallity issues, the data has been anonymised: There are 28 numerical features named *V1 ... V28* which are the result of a Principal Component Analysis (PCA) transformation. Additionally there also are two special features - *Time* and *Amount*. *Time* contains the seconds elapsed between each transaction and the first transaction in the dataset. *Amount* contains the amount of money in an unknown currency.

As one can see from the Table 4.1, the features *V1 ... V28* have effectively mean of zero. On the other hand standard deviation (std) as well as minimum and maximum cover wider range of values.

It is also useful to look at the histogram representation of at least some of the features. It will give us a basic idea of the data distribitution. Because fraudulent data are so rare and probably also very sparse, using linear scale for the y axis is not optimal, since the outliers would most likely not be visible. Log scale is therefore used instead.

First lets look at the only two features not transformed by PCA: *Time* and *Amount*.

The *Time* histogram (Figure 4.1) shows a very dense distributioin with cyclical behavior. Because all the data points are so close to each other (even in the least dense areas), we

	V1	V2	...	V27	V28	Amount
mean	3.919560e-15	5.688174e-16	...	-3.660161e-16	-1.206049e-16	88.34
std	1.958696e+00	1.651309e+00	...	4.036325e-01	3.300833e-01	250.12
min	-5.640751e+01	-7.271573e+01	...	-2.256568e+01	-1.543008e+01	0.00
25%	-9.203734e-01	-5.985499e-01	...	-7.083953e-02	-5.295979e-02	5.60
50%	1.810880e-02	6.548556e-02	...	1.342146e-03	1.124383e-02	22.00
75%	1.315642e+00	8.037239e-01	...	9.104512e-02	7.827995e-02	77.16
max	12.454930e+00	2.205773e+01	...	3.161220e+01	3.384781e+01	25691.16

Table 4.1: A basic statistical description of features in our dataset

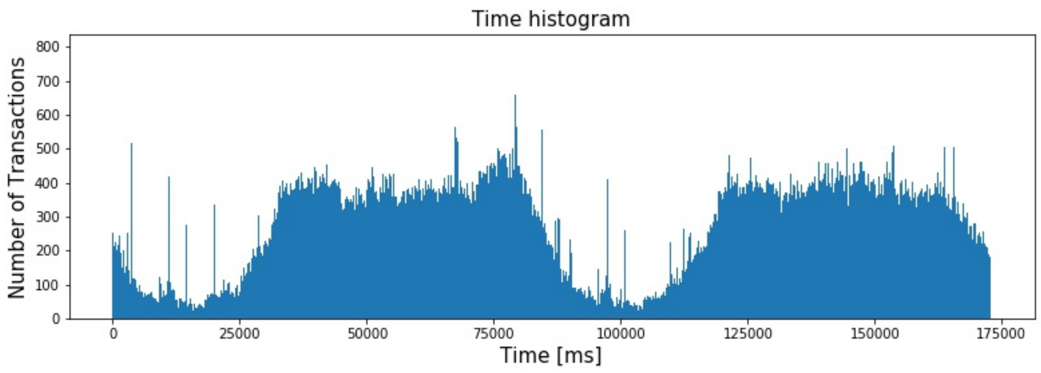

Figure 4.1: The x axis represents time in milliseconds. They y axis represents the number of transaction during each time interval. There is an obvious cyclical behavior but overall the distribution is too dense to reveal any outliers.

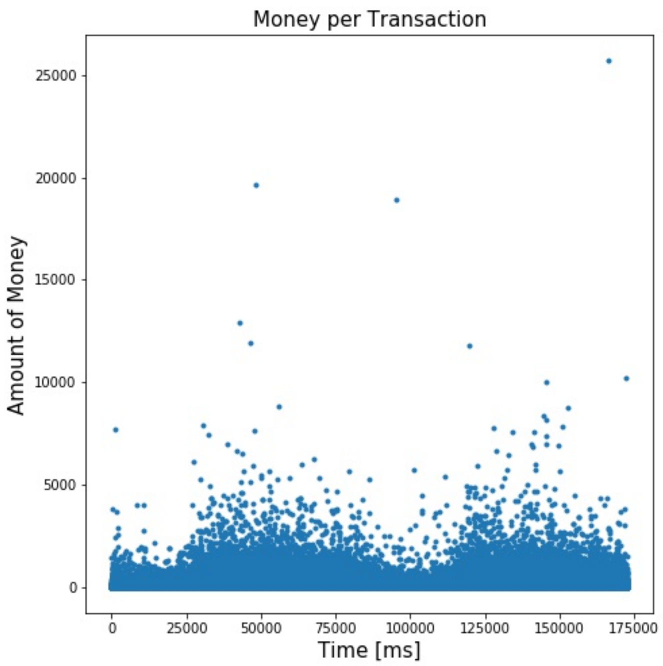

Figure 4.2: The x axis represents time in milliseconds. The y axis represents the amount of money transfered. The only outliers here are the transactions which transfer a lot of money.

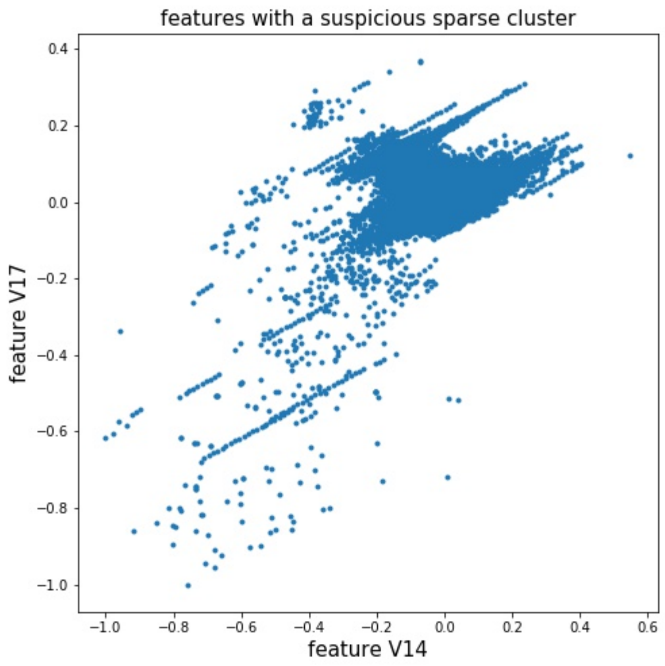

Figure 4.3: The most promising type of features. They contain data instances which are highly unlikely to be part of the major cluster because of their different distribution. The x axis represents the feature V14. They y axis represents the feature V17.

can safely remove this feature from the dataset, since it would not be useful for the dection system.

The *Amount* feature on the other hand has some obvious outliers as can be seen from Figure 4.2. The problem is, those outliers are simply transactions which transfer a lot of money - something that usually draws a lot of attention, which fraudsters try to avoid. Our domain knowledge therefore suggest that fraudsters will transfer only small amounts of money, very frequently, making them undetectable (at least based on the amount of money transfered). Because of that we were highly tempted to also remove this feature from the dataset, but that would mean fraudsters could take an advantage of this fact (if they reverse engineered our detection system). Therefore we ultimately decided to leave it in the dataset.

We will now pick some interesting examples from the rest of the features. The most important thing we can take from visualization is the number of clusters, some basic idea of anomalous behavior and an estimation of how which features will be actually useful for the detection process.

As can bee seen from both figures 4.4 and 4.3, data instances form usually only one dense cluster. Figure 4.3 also shows quite a lot data points with a very different distribution and position than the rest, which means they are most like the anomalies we are looking for. They still look like they form a cluster of some sort. This means that we could teorethically use clustering techniques based both on the assumptions that anomalies form a cluster and that anomalies do not belong to any cluster.

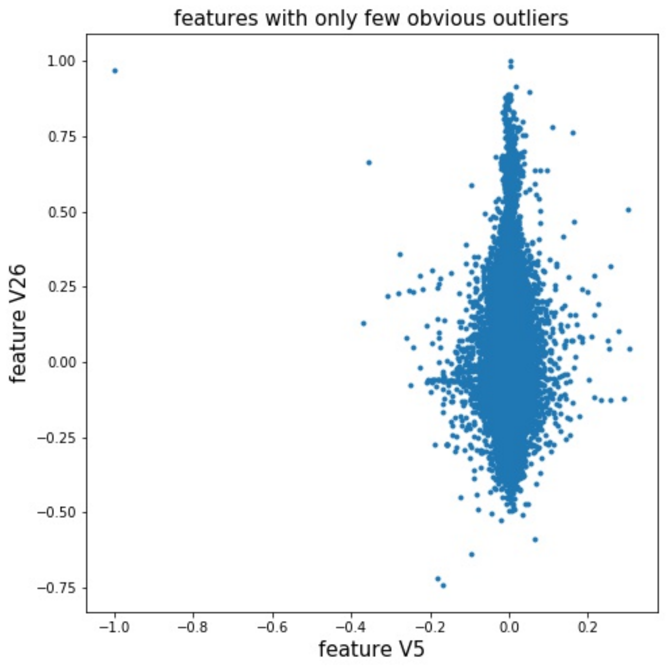

Figure 4.4: There are only few obvious outliers. Almost all datapoints look like to be generated by the same distribution. The x axis represents the feature V5. They y axis represents the feature V26.

Finally most of the features (see Figure 4.4) have only few outliers. Most of their data seems to be generated from one distribution. Anomalies will be therefore somewhere inside of those clusters, not possible to detect. It would be therefore beneficial to have some form of feature selection, to get rid of these useless features.

Chapter 5

Problem definition

We present a mathematical definition of anomaly detection in banking industry. Our task is finding fraudulent behavior in a given credit card transaction dataset.

Obtaining a labeled dataset in real world scenario is, with few exceptions, very complicated or even impossible (especially in the domain of anomaly detection). Therefore, even though we have access to the labels of all transactions, we will focus on unsupervised techniques, to reflect the most common situation in real world, where the dataset is often unlabeled.

Let T be the set of all credit card transactions. Every transaction $t_t \in T$ consists of

- The time $time_t$ of its execution in milliseconds.
- The amount $amount_t$ of money transfered.
- The set F_t of 28 anonymized, numerical features.

Furthermore for the purpose of evaluation we also have access to the set L which contains label $l_t \in L$ of each transaction.

Let $model : \theta_m \times T_{train} \rightarrow classifier_m$ be our model and $classifier_m : \theta_c \times t \rightarrow label$ be our classifier such that

- θ_m is the set of model hyperparameters closely related to detection method used.
- $T_{train} \subseteq T$ is a ransaction subset used to train the classifier.
- θ_c is the detection rate sensitivity parameter.
- The term $t \rightarrow l$ assigns a label $l \in \{Fraud, Normal\}$ to any transaction $t \in T$. As we will see, these transactions will be obtained from the subset $\{T_{test} \subseteq T \| T_{test} \cap T_{train} = \emptyset\}$.

We are not going to use the labels during training, because we use unsupervised techniques. But we can use the labels to examine the performance of our unsupervised classifier. Therefore our task is to find the best performing *model* and its hyperparameters θ_m:

$$\arg \max_{model,\theta_m} performance(model, T_{train}, \theta_m, T_{test})$$

Where the peformance is measured by function $performance : model, T_{train} \times \theta_m \times T_{test} \to \mathbb{R}$.

Chapter 6

Proposed solution

In this chapter we describe our proposed solution based on an autoencoder neural network, which is a member of the spectral based anomaly detection techniques. We have deliberately decided to use this technique for the following reasons:

1. The only requirement we have on the dataset is to be numerical (we do not need distance metric).
2. The method makes very few assumptions about anomaly behavior (it does not matter if anomalies form their own clusters or not).
3. Neural networks work well with highly dimensional data.
4. Neural networks do not have a problem with large number of samples (our dataset has around 285 000 samples).

We first describe the architecture of our solution in section 6.1. The autoencoder neural network is discussed in section 6.4 along with its wide pool of hyperparameters. We explain our choices for all of them. Then we briefly visit the weight initialization method with pretraining as described in 3.2. Finally we present our feature selection and scaling method in section 6.6 to avoid and penalize badly reconstructed features.

6.1 Architecture

As depicted in the Figure 6.1, the heart of our classifier is a four layered symmetrical neural network with two encoding and two decoding layers as described in 6.4. It uses a condition for an early training phase exit if a detection threshold is given.

The second most important component is feature scaling and selection, which is used to penalize badly reconstructed features as described in 6.6. A detection threshold is again required, otherwise the component is not utilized.

The output of our detection system is either vector of labels or the anomaly (suspicion) scores.

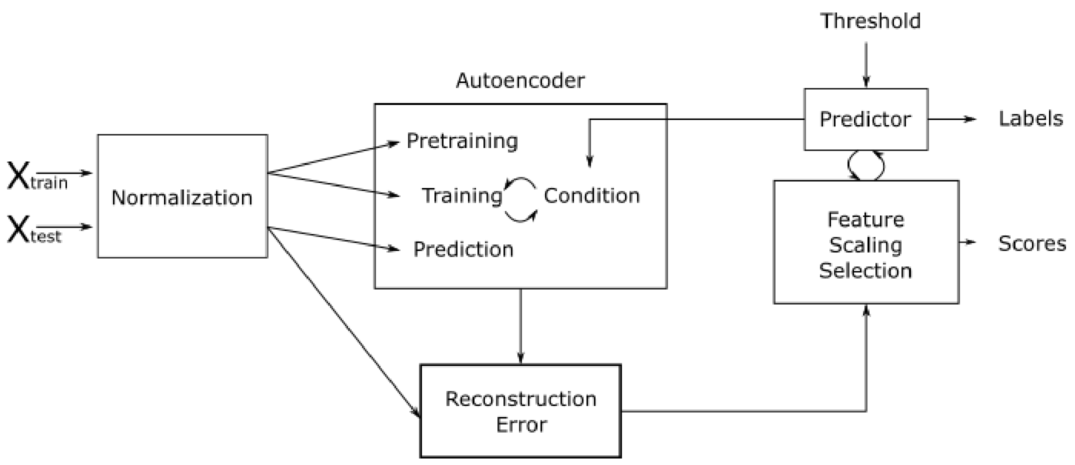

Figure 6.1: The workflow of our proposed solution. $X_{train} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}}$ and $X_{test} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}}$ are the transaction datasets used for training and testing the classifier. $Threshold \in [0,1]$ is a real number that signifies how many frauds should be detected. $Labels \in \{Fraud, Normal\}^N$ is a vector of predicted labels based on the given $Threshold$. $Scores \in \mathbb{R}^{\mathrm{N}}$ are the suspicion scores of each transaction.

6.2 Mean Squared Feature Recall

Even though we do not have access to labels during training, we would like to at least have some estimate of the performance of our solution. It turned out that the mean squared feature recall, as shown in Pseudocode 1, is a good enough measure for the actual performance of our classifier.

Algorithm 1 Compute mean of squares of each feature recall.

```
1: procedure MEAN SQUARED FEATURE RECALL($RE, t$)    ▷ RE is the reconstruction error matrix, t is a detection sensitivity
2:     $scores = predictScores(RE)$
3:     $labels = predictLabels(RE, t)$
4:     $recalls = emptyList$
5:     for $feature \leftarrow RE$ do
6:         $recalls.append(recall(labels, predictLabels(feature, t))^2)$
7:     end for
8:     return $mean(recalls)$
9: end procedure
```

The idea is that untrained autoencoder will produce a terrible reconstruction of the original dataset, because every feature will predict the anomalies differently (reconstruction error is random at the beginning). In other words the mean of their prediction recalls will be low, but steadily getting better over time.

We used this metric during experimentation with the various autoencoder hyperparameters and it is also used as an condition for an early stop of training phase: once the mean

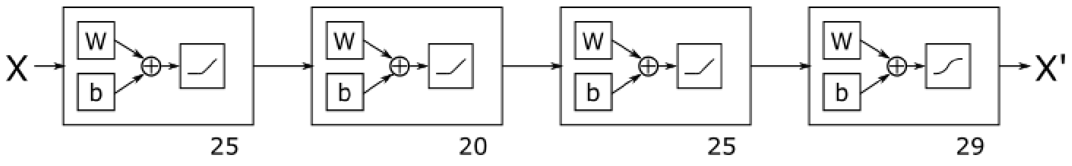

Figure 6.2: The proposed autoencoder we use to compute the reconstruction error. The network consists of four layers with 25, 20, 25 ad 29 neurons in this order. First three hidden layers use the exponential linear unit as the activation function, while the output layer is using logistic sigmoid.

squared feature recall stops improving, we halt the training. The little disadvantage of this method is the need for prediction threshold, which means we need to have some reasonable approximation of the ratio of anomalies in the dataset. For example although only 0.173% transactions are fraudulent in our dataset, the prediction threshold of 0.002 would give almost the same results as the exact threshold of 0.00173.

6.3 Normalization

Naturally, the transactions dataset must be first normalized, before it can be used for training or detecting. Because our output layer of autoencoder is sigmoid, we normalize the transactions features by their mean and limit them to the interval $[0, 1]$. The normalization is pretty straightforward as shown in 2.

Algorithm 2 normalize procedure

```
1: procedure NORMALIZE($X$)                ▷ X is a dataset, rows are data points
2:     for $column \leftarrow X$ do
3:         $column = column - mean(column)$
4:         $column = column * 0.5/max(abs(min(column)), abs(max(column)))$
5:         $column = column + 0.5$
6:     end for
7:     return T
8: end procedure
```

6.4 Autoencoder hyperparameters

One of the problems of neural networks is their large number of hyperparameters, resulting in vast space of possible configurations while only a small subpart actually gives good results. Therefore finding the meaningful ones is not a trivial task and it usually takes more effort compared to other methods. Because of that we will take the effort to describe all of the hyperparameters we have chosen and considered, when building the neural network. We will also explain what lead us to choose those particular hyperarameters.

The graph of our neural netwok is presented in Figure 6.2.

6.4.1 Activation functions

Activation function is one of the key parts of neural network, which gives them the power to approximate non-linear functions (obviously only if the activation function itself is nonlinear).

We have considered three activation functions for our hidden layers: logistic sigmoid, rectified linear unit (relu) and exponential linear unit (elu). Logistic sigmoid has been very widely used in the past, but it suffers heavily from the vanishing gradient problems and is not very computionaly efficient. The obvious next choice was the most commonly used activation function used in deep networks - rectified linear unit - which addresses both of these problems [4]. In the end we have decided to use exponential linear unit which speeds up the learning by making the mean activations closer to zero. It also addresses the dying relu problem [4].

Formula of exponential linear unit:

$$\sigma_h(\alpha, x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

The final layer should limit the output to certain bounds. We have examined two commonly used function: tanh and logistic sigmoid. Tanh should in theory perform better, because its output is symmetrical around zero. In reality we found out logistic sigmoid performed better, by giving outliers much higher suspicion scores.

Formula of logistic sigmoid:

$$\sigma_o(\alpha, x) = \frac{1}{1 + e^{-\alpha x}}$$

6.4.2 Optimization algorithm

The purpose of optimization algorithm is to minimize the loss function, based on the recontruction error and weight matrices of each layer. The best optimization algorithm should give us the best training speed.

We have tried several avaialable optimizers and measured their loss convergence over 50 epochs. Out of 7 different variants - including the SGD, Adadelta and RMSProp - Adamax showed to have the fastest convergence.

6.4.3 Loss function

Loss function is used for evaluation and minimization of the difference between expected and actual output. Since we are interesting in finding outliers in the dataset, the loss function should be insensitive to them.

We have examined the popular *mean squared error (mse)* and also not so commonly used *mean absolute error (mae)* and *logarithm of cosh (LC).*

$$MSE(x, \hat{x}) = \frac{\sum_{i=1}^{N} ||x_i - \hat{x}_i||^2}{N}$$

$$MAE(x, \hat{x}) = \frac{\sum_{i=1}^{N} |x_i - \hat{x}_i|}{N}$$

$$LC(x, \hat{x}) = \frac{\sum_{i=1}^{N} log(cosh(x_i - \hat{x}_i))}{N}$$

As it turned out, both *mse* and even *logcosh* to a point, were too sensitive to outliers, because of their exponential growth. So in the end we went with *mean absolute error*, which was more tolerant to them.

6.4.4 Layers

The number of layers and neurons generally depends on the complexity of the relations in the dataset. In other words, the more complex is the regression function we try to model, the more neurons and layers are required. With too many layers there is also the danger of overfitting the data, leading to poor general solution. In outlier detection this behavior is even worse, because overfitting leads to very low suspicion scores.

We have examined several configurations and given the dataset does not seem to be too complex (there is only one cluster of normal data), increasing number of neurons in a layer did not yield noticeably better results. Therefore our number of neurons decreases by 5 in each encoding layer. The number of layers then closely corresponds to the dimension of our latent representation.

In the end we have chosen an implementation with two encoding and two decoding layers ($29 \rightarrow 25 \rightarrow 20 \rightarrow 25 \rightarrow 29$) as shown in 6.2, because it produced very high suspicion scores during training.

In chapter 8.1 we compare different layer configurations and show how very deep autoencoder tended to overfit the outliers.

6.4.5 Batch size

Batch size indicates how many samples of data will be propagated in the neural network. Similar to optimization algorithm, batch size has a direct impact on the performance of our training phase. Using small batches instead of whole dataset during training allows us to escape bad local optima (saddle points) by introducing noise into the gradient. It also improves the convergence, because gradient evaluated over a small batch acts as an estimation (especially) of the gradient of the whole dataset.

We have examined multiple batch sizes, ranging from 32 (along with 64, this batch size is commonly used for deep neural networks and the exact size typically depends on the underlying GPU) to few thousands. Our autoencoder achieved the best convergence with the batch size of 64.

6.5 Pretraining

As described in 3.2 pretraining is weight initialization technique, particularly effective for deep autoncoder networks. It improves the training speed and helps to avoid poor local optima. Although the autoencoder we use in our solution has only four layers, we have got better results wih pretraining turned on. It was also very useful during the initial phase, when we experemintED with the number of layers in combination with other hyperparameters.

Naturally we also use *adamax* optimizer and *mean absolute loss* loss function for pretraining as for the the actual autoencoder. The only parameter which is different, is the much lower number of epochs, because the convergence is very fast with only one hidden layer.

6.6 Feature selection and scaling

Our solution relies on the fact that the data poins with highest reconstruction error will be anomalies. The reasoning behind it lies in their special distribution, which differs from the majority of data points. But there are two instances, where this approach fails:

1. If the anomalies are located inside the normal cluster and the normal cluster is hard to replicate.
2. If the normal class also has, for any reason, similar reconstruction error to anomalies.

In the first case, the anomalies will be simply undetectable. Instead, the normal instances at the edge of cluster will have high suspicion score. In the second case, we will get very high number of false positives, because the ratio of normal to anomalous instances is highly unbalanced in favor of nonanomalous data.

It makes perfect sense, that we would like to avoid all features with any of those behaviors, because they will have highly negative impact on the calculation of the total reconstruction error. The problem is that we first need to run the algorithm, before being able to select the right features, since if we want to address the first problem, we need to know which data are anomalous, and if we want to address the second problem, we need to know, what the reconstruction error is.

Our solution therefore involves selecting relevant features after the training phase is finished, based on their reconstruction error. But unlike the widely used wrapper and embedded feature selection methods, we are not going to retrain the autoencoder with the new subset of features.

Another technique that will help us to mitigate both of the mentioned problems is feature scaling. By feature scaling we just mean multiplying each feature by some scalar $x \in IR$. Compared to feature selection, there is no threshold that decides, which features are relevant and which are not. This makes the method more robust and easier to use. In fact, our features selection is just a special form of feature scaling, where we multiply the the selected feature reconstruction error by zero.

Our feature selection and scaling algorithm is described in 3. We first normalize all features with their quantile (which should be very close to the maximum loss of normal transaction), thus all anomalies will have their loss greater or equal to 1. Then we penalize them based on their recall. Finally we remove features with recall lower than 0.5, since they give us less than 50% chance to detect an anomaly. We iterate this algorithm few times to get a better result.

Algorithm 3 Feature scaling and selection algorithm. It selects the best features based on their recall.

```
1: procedure FeatureScalingAndSelection(RE,t)    ▷ RE is a reconstruction error matrix, t is a detection threshold (sensitivity)
2:     for $featureRE \leftarrow RE$ do    ▷ normalize features
3:         $featureRE = featureRE/quantile((1 - t) * 0.99, featureRE)$
4:     end for
5:     $recalls = emptyList$
6:     for $featureRE \leftarrow RE$ do    ▷ intialize recalls
7:         $recalls.push(1)$
8:     end for
9:     for $i \leftarrow [1, 2, 3]$ do    ▷ do 3x
10:        $labels = predict(ScaleAndSelectHelper(RE.copy(), recalls), t)$
11:        $recalls = emptyList$
12:        for $featureRE \leftarrow RE$ do
13:            $recalls.append(getRecall(labels, predict(featureRE, t)))$
14:        end for
15:        return $ScaleAndSelectHelper(RE.copy(), recalls)$
16:    end for
17: end procedure
18: procedure quantile(n, vector)
19:    vector.sort()
20:    index = length(vector)*n
21:    return (vector[floor(index)] + vector[ceil(index)]) / 2
22: end procedure
23: procedure ScaleAndSelectHelper(RE, recalls)
24:    for $featureRE, recall \leftarrow RE, recalls$ do
25:        if $recall > 0.5$ then    ▷ penalize features with low recall
26:            $featureRE = featureRE * recall$
27:        else    ▷ if recall is too low, remove the feature
28:            $RE.dropFeature(featureRE)$
29:        end if
30:    end for
31:    return $RE$
32: end procedure
```

Chapter 7

Implementation

In this chapter we describe the implementation of our proposed solution, which allowed us to experiment with various autoencoder paramaters and evaluate the performance of our algorithm. We discuss the programming language and libraries we have used in section 7.1. The specifics of implementation are described in section 7.2.

7.1 Technology

We have used the python language, because of its popularity in data science. It has high quality data science libraries with good documentation and convenient development environment - jupyter notebooks - which is great for easy and fast visualization.

For implementing the autoencoder, we have used the high level neural network library **keras** [4]. It uses the same API for several popular neural network backends called TensorFlow [5], CNTK [24] and Theano [3]. Therefore it allows us to switch the underlying neural network implementation without changing our code. In our implementation we use the TensorFlow backend, but there should be no problem using one of the other two instead.

For manipulating the matrices effectively, we use the **numpy** and **pandas** [2] libraries. **Numpy** allows us to use much less memory for matrices, than the default python lists, and it also makes the matrix operations much more efficient. **Pandas** is built on top of **numpy** and it provides higher level interface for manipulating datasets with named rows and columns (the input datasets are required to be **pandas** dataframes).

To measure the performance of our detection system, we have used some helper functions from the library **sklearn** [26].

Finally, we have used **matplotlib** [1] to create some of the figures shown throughout the thesis. Mostly to visualize the dataset along certain dimensions.

7.2 Code structure

Because python is an object oriented programming language (OOP), we have used objects (classes) to wrap the functionality of the components described in the proposed algorithm

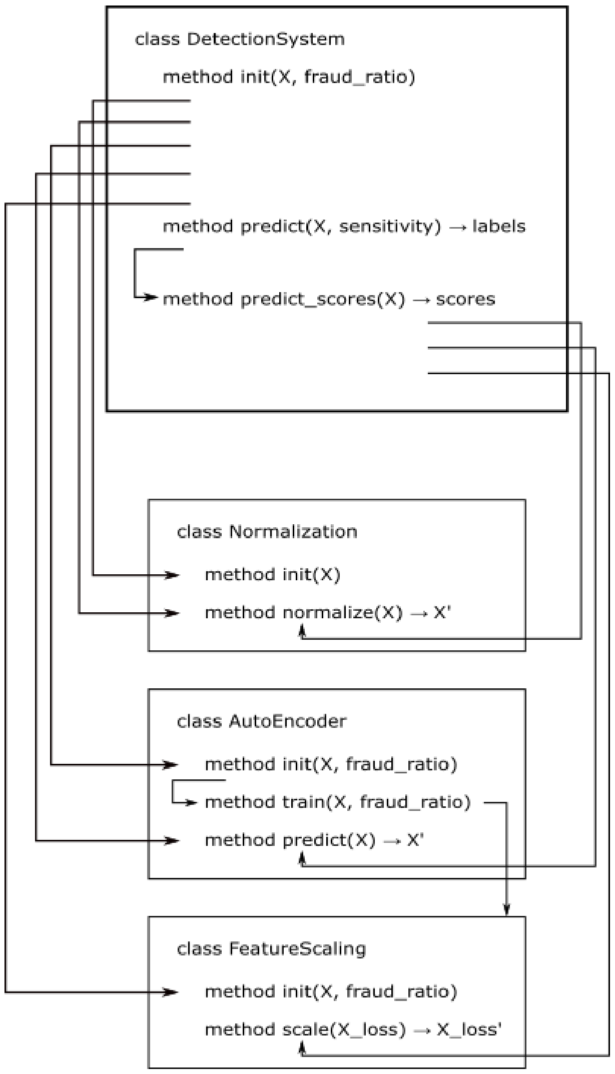
class DetectionSystem
method init(X, fraud_ratio)
method predict(X, sensitivity) → labels
method predict_scores(X) → scores
class Normalization
method init(X)
method normalize(X) → X'
class AutoEncoder
method init(X, fraud_ratio)
method train(X, fraud_ratio)
method predict(X) → X'
class FeatureScaling
method init(X, fraud_ratio)
method scale(X_loss) → X_loss'

Figure 7.1: Scheme of the implementation. After calling the DetectionSystem method init, the three other classes are instantiated and saved, so that they can be later used in the method predict. The methods are called in the order as the arrows suggest. The parameter X is an input dataset, X_loss is the reconstruction error matrix, sensitivity is the the detection threshold used for labeling transactions.

architecture 6.1. We have depicted the interaction between the components in the Figure 7.1.

As can be seen, we use four different classes:

1. DetectionSystem
2. Normalization
3. AutoEncoder
4. FeatureScaling

The DetectionSystem is the only class which is exposed to the world, using all of the other components mentioned. Therefore if we are about to classify anomalies in some dataset, the DetectionSystem will be the only class we will interact with.

It exposes these three methods:

1. $init(X \in \mathbb{R}^{\mathrm{M \times N}}, \mathrm{fraud\_ratio} \in \mathbb{R})$

   This method takes the training dataset X and an approximate ratio of frauds in the dataset $fraud\_ratio$. It initializes all the other objects (Normalization, AutoEncoder, FeatureScaling) and trains the autoencoder on the dataset.

2. $predict\_scores(X \in \mathbb{R}^{\mathrm{M \times N}}) \rightarrow \mathrm{scores} \in \mathbb{R}^{\mathrm{M}}$

   This method takes a test dataset X and returns anomaly scores for each transaction, which is computed from the reconstructed dataset by autoencoder.

3. $predict(X \in \mathbb{R}^{\mathrm{M \times N}}, \mathrm{sensitivity} \in \mathbb{R}) \rightarrow \mathrm{labels} \in \{\mathrm{Fraud}, \mathrm{Normal}\}^{\mathrm{M}}$

   This method takes a test dataset X and detection *sensitivity* and returns the *labels* for each transaction. The result of *predict_scores* with a cutoff based on the detection sensitivity is utilized to compute the vector of *labels*.

The Normalization object is there to preprocess the dataset in order to be able to compute the reconstruction error matrix. It has only two methods:

1. $init(X \in \mathbb{R}^{\mathrm{M \times N}})$

   This method takes a dataset X and initializes the object by saving the dataset features like means and maximum and minimum.

2. $normalize(X \in \mathbb{R}^{\mathrm{M \times N}}) \rightarrow \mathrm{X}' \in \mathbb{R}^{\mathrm{M \times N}}$

   This method takes a dataset X and returns a normalized dataset X'.

The AutoEncoder object contains an autoencoder neural network which is used to create a reconstruction of the input dataset. It has the following three methods:

1. *init*$(X \in \mathrm{I\!R}^{\mathrm{M \times N}}, \mathrm{fraud\_ratio} \in \mathrm{I\!R})$

   This method takes a training dataset X and initializes the object by building the autoencoder neural network and initializating its weights with pretraining. The $fraud\_ratio$ is required in order to call the *train* method.

2. *train*$(X \in \mathrm{I\!R}^{\mathrm{M \times N}}, \mathrm{fraud\_ratio} \in \mathrm{I\!R})$

   This method takes a training dataset X and an approximate ratio of frauds in the dataset $fraud\_ratio$. It trains the autoencoder neural network with the dataset X. The $fraud\_ratio$ as well an FeatureScaling object is required for an early stop of training.

3. *predict*$(X \in \mathrm{I\!R}^{\mathrm{M \times N}}) \rightarrow \mathrm{X}' \in \mathrm{I\!R}^{\mathrm{M \times N}}$

   This method takes a test dataset X and returns a reconstruction of the dataset X', which is simply the output of autoencoder neural network.

The FeatureScaling object is used to rescale and remove features, based on their utility as described in 6.6. It has exactly two methods:

1. *init*$(X \in \mathrm{I\!R}^{\mathrm{M \times N}}, \mathrm{fraud\_ratio} \in \mathrm{I\!R})$

   This is the initialization method, which takes a reconstruction error matrix X_loss an approximate ratio of frauds in the dataset $fraud\_ratio$. It computes and saves the weights of each feature.

2. *scale*$(X\_loss \in \mathrm{I\!R}^{\mathrm{M \times N}}) \rightarrow \mathrm{X\_loss}' \in \mathrm{I\!R}^{\mathrm{M \times Q}}$

   This method takes a reconstruction error matrix X_loss and returns an updated reconstruction error matrix X_loss', by rescaling the features with the saved weights and removing the features with zero weights.

Chapter 8

Experiments

In the chapter we present the experiments which show the quality of our detection system. We discuss the results of experiments with various autoencoder hyperparameters and show the impact of pretraining and feature scaling on the detection performance. For that we use the *mean squared feature recall* as presented in chapter 6.

Because we have access to the labels of the dataset, we are also going to present several performance measures to discuss the efficiency of our detection system in section 8.3 along with comparison to existing work.

8.1 Number of layers

Although the number layers generally depends on the complexity of function we want to model with the neural network, picking the right number is not trivial. As described in chapter 6, we have examined several configurations and decided for one of them based on the *mean squared feature recall* described in chapter 6.

To illustrate that, we show three figures with different layer configurations, in each the number of neurons between layers decreases (and increases) by 5. To generate these curves, we have used the detection threshold of 0.001727. The f1 score was calculated against the same dataset and same threshold by using the labels we have. Figure 8.2 (left) shows a training phase of autoencoder with four layers (thus the latent representation has 20 dimensions). Figures 8.1 (left) and 8.1 (right) show autoencoders which consist of six and ten layers respectively (thus the latent representations have 15 dimensions and 5 dimensions). We would favor the configuration on left, based on the shape of the curve and based on the maximum value of the mean squared feature recall.

As can be seen from the the figure 8.2, there is a close relation between feature recall and performance of our classifier. Both of those measures usually tend to increase and decrease together. Since the the four layered autoencoder produced a curve with highest maximum value and the most promising shape, we have decided to use it in our autoencoder.

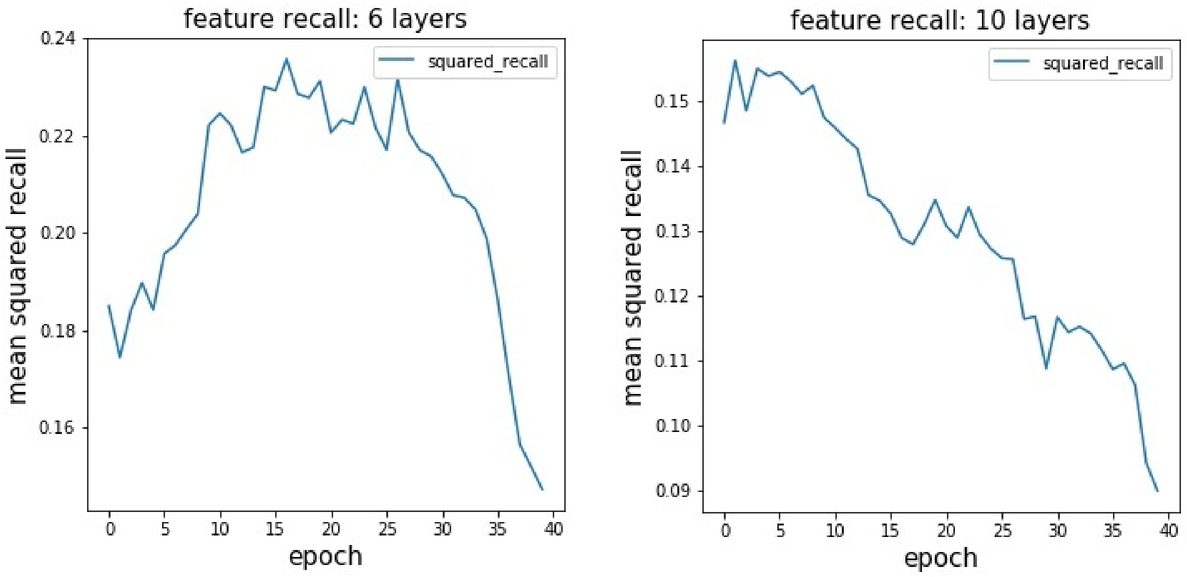

Figure 8.1: (Left) The mean squared feature recall of autoencoder with 6 layers during training. (Right) The mean squared feature recall of autoencoder with 10 layers during training. The autoencoder on the left should outperform the one on the right, because of its overall higher mean squared feature recall and because of the shape of the curve.

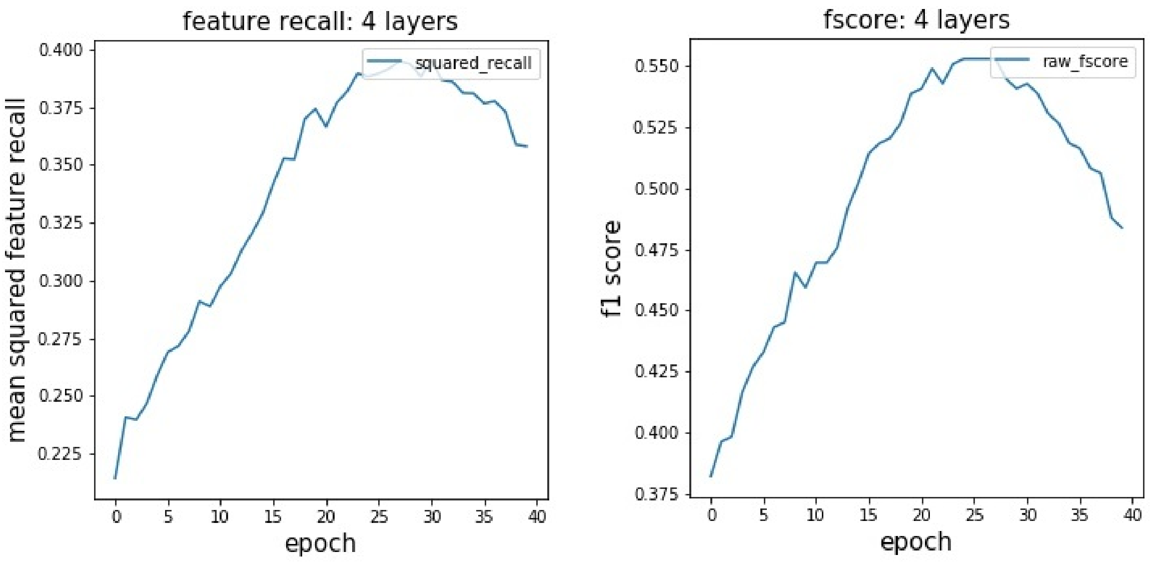

Figure 8.2: (Left) The mean squared feature recall of autoencoder with 4 layers during training. (Right) Raw f1 score (without feature scaling) of the same autoencoder during training.

feature selection	threshold θ	TP	FN	FP	TN
with feature scaling	0.000172	49	443	0	284315
	0.00069	166	326	31	284284
	0.001209	284	208	60	284255
	0.001727	388	104	104	284211
	0.002591	397	95	341	283974
	0.005009	406	86	1021	283294
	0.010019	409	83	2444	281871
	0.015029	411	81	3869	280446
without feature scaling	0.000172	47	445	2	284313
	0.00069	144	348	53	284262
	0.001209	220	272	124	284191
	0.001727	273	219	219	284096
	0.002591	353	139	385	283930
	0.005009	396	96	1031	283284
	0.010019	398	94	2455	281860
	0.015029	402	90	3878	280437

Table 8.1: Evaluation of detection performance with feature scaling turned on and off. The second column contains detection threshold. The third, fourth, fifth and sixth column contain True Positives, False Negatives, False Positives and True Negatives respectively. The default threshold of 0.001727 is in bold and it is equal to the ratio of frauds in the dataset.

8.2 Feature selection

Feature scaling as presented in 6.6 is the key component in our detection system. We are going to show its impact on performance and discuss the weights we obtained.

8.2.1 Performance improvement

In table 8.1 we compare the performance of our detection system with feature scaling turned on and off. To achieve this result, we have trained the autoencoder with whole transactions dataset with the training detection threshold of 0.001727 (the ratio of fraudulent transactions in our dataset), the detection threshold in the table was then used to predict the labels for all transactions in the dataset.

As can be seen, with increasing threshold, the number correctly classified fraudulent transactions (True Positives) increases, but without feature scaling we need much higher detection threshold to get the same number of True Positives.

8.2.2 Learned feature weights

The feature weights are used to scale down the reconstruction error of certain features as described in 6.6. For training we use the whole dataset and detection threshold of 0.001727. The weights are in fact the feature recalls and they are presented in Table 8.2. The larger

removed features	feauture	weight	feature	weight
V28	V24	0.554878	V13	0.906504
V2	V5	0.648374	V9	0.908537
V20	V1	0.658537	V17	0.926829
V27	V4	0.731707	V18	0.932927
V23	V22	0.739837	V12	0.959350
V21	V7	0.762195	V14	0.959350
Amount	V15	0.764228	V16	0.961382
V8	V11	0.794715		
V6	V10	0.843496		
V19	V3	0.845528		
V26	V25	0.898374		

Table 8.2: The very left columns contains all the features which do not influence the final detection score (their weight would be zero), because they are removed from the dataset. The rest of the features is listed with their corresponding weight, the higher, the better.

the weight (recall) is, the more important the reconstruction error of that feature is. We could say that the feature provides more information about fraudulent behavior.

For illustration the two features with largest weights *V14* and *V16* are depicted in the Figure 8.3 and the two features with the lowest weights (if we don't count the removed features with zero weights) *V24* and *V5* are depicted in the Figure 8.4. It can be seen the features in the second graph gives us almost no useful information which could be used to classify fraudulent transactions.

8.3 Classification performance

In this section we discuss the performance of our unsupervised detection system by using the labels from the dataset. Although we have already showed some performance measure in the previous sections, all of them used the whole dataset both for training and classification. This is not ideal, because we might overfit on the data (e.g. by choosing certain parameters) and our solution might not generalize well. We would also like to classify incoming transactions immediately (so called online classification), but retraining the whole autoencoder every time would be too slow.

Because of these three reasons we used cross validation to simulate the situation of receiving an unseen dataset. We use the stratified kfold method where $k = 5$, which means we split the dataset randomly into a training one, containing 80% of the original transactions, and testing one, containing 20% of the original transactions. Because the split is stratified, the ratio of fradulent transactions remains always the same.

This is done 5 times to obtain 5 different training and testing sets, which are used as an input to the detection system (with the detection threshold of 0.1727). We average the results and obtain a single confusion matrix which is depicted in 8.3.

There is also a very straightforward way to quantify the cost of various detection thresholds. If we assume that undetected fraud costs the bank as much money as was stolen, we

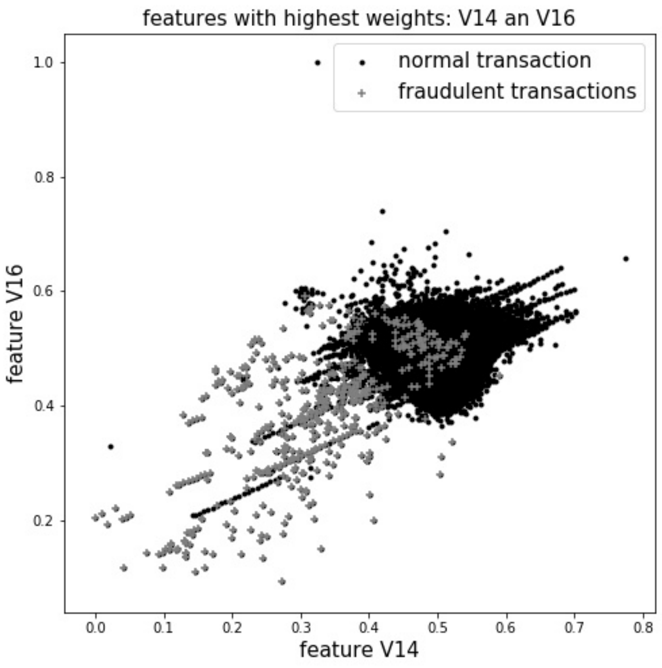

Figure 8.3: Two features with the highest weights *V14* is on the X axis. *V16* is on the Y axis. Fraudulent transactions are highlighted by using the + marker.

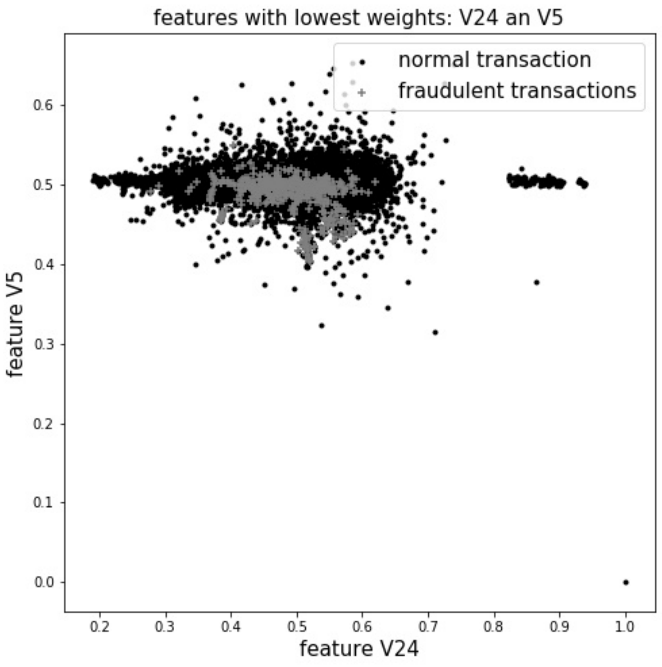

Figure 8.4: Two features with the lowest weights (excluding the removed features with zero weights). *V24* is on the X axis. *V5* is on the Y axis. Fraudulent transactions are highlighted by using the + marker.

	positive (actual)	negative (actual)
positive (predicted)	73	25.4
negative (predicted)	25	56838

Table 8.3: A confusion matrix obtained from the stratified 5-fold cross validation. It is an average of 5 different confusion matrices.

currency	verification cost	threshold	net gain
EUR	0.85 (EUR)	0.0015	37938 (EUR)
		0.0017	44665 (EUR)
		0.0020	45391 (EUR)
GBP	0.74 (GBP)	0.0015	37985 (GBP)
		0.0017	44718 (GBP)
		0.0020	45454 (GBP)
CZK	22 (CZK)	0.0015	28907 (CZK)
		0.0017	34428 (CZK)
		0.0020	33335 (CZK)

Table 8.4: The amount of money spared by our detection system is depicted in the last column. Second and third column show the verification cost and detection threshold for each of the three European currencies.

can calculate the net gain from using our detection system based on the incurred expenses to verify the transaction authenticity. We will use the same average cost of 1$ per verification as Gadi et al. [14]. Note that we don't know the currency of transactions in our dataset, so we use multiple European currencies (because the dataset is from European bank). The net gain of our solution for various currencies and thresholds are shown in the Table 8.4.

8.3.1 Comparison to existing work

Kremnyov et al. [21] have examined the performance of several clustering unsupervised techniques. We show the best performing techniques and their performance in Table 8.5 along with the performance of our detection system.

Since they do not describe in the article, how they have chosen the training and testing dataset, we have used whole transactions dataset for training and evaluating the performance (shown in the above Table 8.5) of our detection system along with the detection threshold of 0.0065.

Although our detection system performs slightly worse than their examined methods, the authors of the article used a *supervised* feature selection, based on point-biserial correlation (a measure of association between a feature and transaction label), removing up to 23 features. They have also selected the model hyperarameters by doing an exhaustive search, picking

algorithm	recall (%)	FPR (%)
our method (threshold = 0.0065)	82.72	0.51
multi variate outlier detection	84.55	0.41
BACON outlier detection	84.55	0.70
k,l means	84.95	0.55
DBSCAN	84.35	0.64

Table 8.5: Outlier techniques are listed in the first column. The performance is measured in recall (middle column) and false postive rate (last column). Our algorithm performance for the detection threshold 0.0065 is listed in the first row.

the one with best recall and false positive rate.

The reason we have picked this article, even though their approach is not unsupervised from ground up, is that it still gives us as an anchor on what should we expect from ideally set up unsupervised technique (with the right hyperarameters, applied to the most significant features). As can be seen from the Table 8.5, our solution performance is very close to it, which is very promising.

We have also tried to look for an article using the same card transactions dataset detailing an unsupervised method based on autoencoders, but not a single one had a good enough performance (usually around 50% recall and 30% precision) to be mentioned in this section.

Chapter 9

Conclusion

All goals were fullfilled. We have researched related work and decided to use spectral based detection technique. We have decided to use autoencoder based detection system because of its advantages over other anomaly detection methods: fast classification, low memory consumption and.

Our unsupervised detection algorithm architecture was presented along with its implementation. We described each component of the detection system and explained the parameter choices for the autoencoder neural network.

We have presented and explained our own unsupervised feature scaling and selection algorithm. Unlike the common filter, wrapper and embedded feature selection algorithms, we apply it at the end of the detection process, to the reconstruction error matrix obtained from autoencoder. It has dramatically improved our classifier performance by more than 50% for certain detection thresholds.

Finally, we applied our detection algorithm to a real credit card transactions dataset from an unnamed European bank. Because the dataset was labeled, we could measure the performance of our algorithm by several different measures. We also compared our unsupervised approach to existing algorithms, achieving comparable performance to the best examined classifiers.

9.1 Future work

Our dataset was rather simple, containing only one cluster of normal transactions. The next logical step is to apply our solution on more complex dataset with multiple normal clusters, possibly corresponding to different normal classes, and improve our solution in case it does not generalize well.

It is also pretty unfortunate, that the dataset we got does not include any categorical features like some kind of id of the credit card holder, a place, from where tha transaction was realized and the name of the target (e.g. salesman id). Because of that we had to completely omit a very interesting part of credit card fraud detection called feature aggregation.

There is also another interesting use case for autoencoders which we peeked into during our experimentation phase. The latent representation is technically a compressed version of the input dataset, which means only redundant data (like noise) are removed. Anomalies should be therefore be present and most importantly separable even in the lower dimensional projection. We can utilize this by using the hidden representation as an input to another unsupervised technique which perhaps does not scale well to highly dimensional data.

Bibliography

[1] Matplotlib: Python plotting library. Page: <https://matplotlib.org/>.

[2] Pandas: Powerful Python data analysis toolkit. Page: <https://pandas.pydata.org/talks.html>.

[3] The Theano deep learning library. Page: <http://www.deeplearning.net/software/theano/>.

[4] Wikipedia: Rectified linear unit, May 2018. Page: <https://en.wikipedia.org/wiki/Rectifier_(neural_networks)>.

[5] ABADI, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Page: <https://www.tensorflow.org/>.

[6] ALESKEROV, E. – FREISLEBEN, B. – RAO, B. Cardwatch: A neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, s. 220–226. IEEE, 1997.

[7] AMER, M. – GOLDSTEIN, M. – ABDENNADHER, S. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, s. 8–15. ACM, 2013.

[8] AN, J. – CHO, S. Variational autoencoder based anomaly detection using reconstruction probability. *SNU Data Mining Center, Tech. Rep.* 2015.

[9] DHEEPA, V. – DHANAPAL, R. et al. Analysis of credit card fraud detection methods. *International journal of recent trends in engineering*. 2009, 2, 3, s. 126–128.

[10] DORRONSORO, J. R. et al. Neural fraud detection in credit card operations. *IEEE transactions on neural networks*. 1997, 8, 4, s. 827–834.

[11] ERTÖZ, L. – STEINBACH, M. – KUMAR, V. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, s. 47–58. SIAM, 2003.

[12] ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, 96, s. 226–231, 1996.

[13] FOWLER, J. E. – DU, Q. Anomaly detection and reconstruction from random projections. *IEEE Transactions on Image Processing*. 2012, 21, 1, s. 184–195.

[14] GADI, M. F. A. – WANG, X. – LAGO, A. P. Credit card fraud detection with artificial immune system. In *International Conference on Artificial Immune Systems*, s. 119–131. Springer, 2008.

[15] GHOSH, S. – REILLY, D. L. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, 3, s. 621–630. IEEE, 1994.

[16] GROUP, M. L. Credit Card Fraud Detection Dataset | Kaggle, 2015. Page: <https://www.kaggle.com/mlg-ulb/creditcardfraud>.

[17] GUHA, S. – RASTOGI, R. – SHIM, K. ROCK: A robust clustering algorithm for categorical attributes. *Information systems*. 2000, 25, 5, s. 345–366.

[18] HINTON, G. E. – SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *science*. 2006, 313, 5786, s. 504–507.

[19] HOSSEINPOUR, F. et al. Artificial immune system based intrusion detection: innate immunity using an unsupervised learning approach. *International Journal of Digital Content Technology and its Applications*. 2014, 8, 5, s. 1.

[20] HU, W. – LIAO, Y. – VEMURI, V. R. Robust Support Vector Machines for Anomaly Detection in Computer Security. In *ICMLA*, s. 168–174, 2003.

[21] KREMNYOV OLEG, A. M. Dealing with Outliers, Feb 2018. Page: <https://www.codeproject.com/Articles/1231386/Dealing-with-Outliers>.

[22] LIU, F. T. – TING, K. M. – ZHOU, Z.-H. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*. 2012, 6, 1, s. 3.

[23] MAES, S. m. et al. Credit card fraud detection using Bayesian and neural networks. In *Proceedings of the 1st international naiso congress on neuro fuzzy technologies*, s. 261–270, 2002.

[24] MICROSOFT. The Microsfot CNTK neural network library. Page: <https://github.com/Microsoft/CNTK>.

[25] NICOLAU, M. – MCDERMOTT, J. et al. A hybrid autoencoder and density estimation model for anomaly detection. In *International Conference on Parallel Problem Solving from Nature*, s. 717–726. Springer, 2016.

[26] PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, 12, s. 2825–2830.

[27] ŞAHIN, Y. G. – DUMAN, E. Detecting credit card fraud by decision trees and support vector machines. 2011.

[28] SHYU, M.-L. et al. A novel anomaly detection scheme based on principal component classifier. Technical report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[29] SMITH, R. et al. Clustering approaches for anomaly based intrusion detection. *Proceedings of intelligent engineering systems through artificial neural networks.* 2002, s. 579–584.

[30] SYEDA, M. – ZHANG, Y.-Q. – PAN, Y. Parallel granular neural networks for fast credit card fraud detection. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, 1, s. 572–577. IEEE, 2002.

[31] VIAENE, S. et al. A Comparison of State-of-the-Art Classification Techniques for Expert Automobile Insurance Claim Fraud Detection. *Journal of Risk and Insurance.* 2002, 69, 3, s. 373–421.

[32] VIAENE, S. – DERRIG, R. A. – DEDENE, G. A case study of applying boosting Naive Bayes to claim fraud diagnosis. *IEEE Transactions on Knowledge and Data Engineering.* 2004, 16, 5, s. 612–620.

[33] ZHANG, D. et al. iBAT: detecting anomalous taxi trajectories from GPS traces. In *Proceedings of the 13th international conference on Ubiquitous computing*, s. 99–108. ACM, 2011.

Appendix A

Content of the CD

The attached CD contains the eletronic version of our thesis, the source code of our algorithm and saved weights of the autoencoder that we ran the benchmarks with.

The algorithm can be run by installing python programming language (e.g. with anaconda) and the required libraries (sklearn, numpy, pandas, keras, tensorflow) and executed from the command line as described in Readme.txt.

Folder / File Name	Content
source	Folder containing the python source code of our algorithm along with the card transactions dataset
text	Folder which contains the PDF version of thesis
weights	Folder containing the weights we have used in our performance analysis
README.txt	File that explains how to run our algorithm

Table A.1: Content of the CD.