Tampereen yliopisto

Teemu Soininen

SUSPICIOUS TRANSACTION DETECTION WITH STATISTICAL LEARNING METHODS

Master of science thesis

Faculty of engineering and natural sciences

ABSTRACT

TEEMU SOININEN: Suspicious transaction detection with statistical learning methods
Tampere University
Master of science thesis, 54 pages
May 2019
Faculty of science and engineering
Major: Theoretical Computer Science
Examiners: prof. Tapio Elomaa and prof. Ari Visa

Keywords: outlier detection, novelty detection, unsupervised learning, transactions

In this thesis the aim was to find a more efficient way for detecting suspicious transactions from banking data. The chosen approach was to utilize outlier detection methods.

The methods were first chosen based on a theoretical review but then narrowed down to those that have stable implementations in Python. The banking transaction data was then preprocessed and fed to the methods. For clustering methods we reviewed the running time and the CH index and for outlier detection the comparison was made from running time and visual exploration of the results.

Finally it was found that GMM and iForest were the best performing methods. They were able to perform outlier detection on the large datasets in just minutes and should scale to a dataset of any size. They also have existing implementations in SKlearn and could be implemented as a part of a detection system.

TIIVISTELMÄ

TEEMU SOININEN: Epäilyttävien pankkitapahtumien tunnistaminen oppivien tilastollisten menetelmien avulla
Tampereen yliopisto
Diplomityö, 54 sivua
Toukokuu 2019
Teknis-luonnontieteellinen koulutusohjelma
Pääaine: Teoreettinen ohjelmistotiede
Tarkastajat: prof. Tapio Elomaa ja prof. Ari Visa

Avainsanat: Poikkeamien tunnistus, ohjaamaton oppiminen, pankkitapahtumat

Tämän diplomityön tavoitteena oli löytää tehokkaampi tapa tunnistaa poikkeavia pankkitapahtumia. Läthökohdaksi valittiin poikkeamien tunnistusmenetelmät.

Alustavat menetelmät valittiin teoreettisen tarkastelun pohjalta, mutta näistä karsittiin pois ne, joilla ei ollut vakaata toteutusta Pythonissa. Pankkitapahtumatiedot esikäsiteltiin ja syötettiin metodeille, jonka jälkeen eri metodien tuloksia tarkasteltiin. Ryhmittely menetelmien osalta tarkasteltiin ajoaikaa ja CH-indeksiä. Poikkeamien tunnistuksessa tarkasteltiin ajoaikaa ja tuloksia käytiin läpi visualisointien avulla.

Työssä löydettiin kaksi hyvin toimivaa menetelmää: GMM ja iForest. Ne pystyivät suorittamaan poikkeamien tunnistusta suurille tietomäärille vain minuuteissa ja niiden pitäisi skaalautua minkä kokoiseen tietomäärään vain. Niistä on myöskin olemassa olevat toteutukset SKlearnissa, joten ne voitaisiin toteuttaa osaksi tunnistusjärjestelmää.

ALKUSANAT

Tämä diplomityö on toteutettu yhteistyössä pohjoismaisen pankin kanssa, joka on toimittanut tarvittavat ympäristöt ja pääsyn tarvittaviin historiatietoihin. Ilman tätä yhteistyötä ei tämä työ olisi ollut mahdollinen.

Kiitos myös kaikille, jotka auttoivat aiheen löytämisessä. Lisäksi myös kiitos pankin henkilöille, jotka auttoivat aineiston käsittelyssä ja sen ymmärtämisessä.

Viimeisenä tahdon kiittää tarkastajia Tapio Elomaata ja Ari Visaa arvokkaista kommenteista ja parannusehdotuksista.

Tampereella, 2.4.2019

Teemu Soininen

CONTENTS

LIST OF FIGURES

ACRONYMS

API Application Programming Interface. 24, 27, 28, 41

AUC Area Under Curve. 11, 13, 14, 22

AWS Amazon Web Services. 23–25, 29, 34, 35, 46

BST Binary Search Tree. 22

CH Calinski-Harabasz. 10, 11, 46, 47

CNN Convolutional Neural Network. 12

DPGMM Dirichlet Process Gaussian Mixture Model. 18, 19, 46, 47, 51

DPMM Dirichlet Process Mixture Model. 18

DR Dimensionality Reduction. 14, 15, 36

DW Data Warehouse. 24

EC2 Elastic Compute Cloud. 23

EM Expectation-Maximization. 18

FCM Fuzzy C-means. 7, 8

FN False Negatives. 12

FP False Positives. 12

FPR False Positive Rate. 13

GMM Gaussian Mixture Model. 17–19, 46–48, 50, 51

GWR Grow-When-Required network. 20, 22, 51

IAM Identity and Access Management. 24, 25

iForest Isolation Forest. 21, 22, 42, 44, 47, 48, 50, 51

IG Information Gain. 14

iTree Isolation Tree. 22

k-nn k-nearest neighbour. 12, 19

LDA Linear Discrimant Analysis. 14, 15

LOF Local outlier factor. 19, 47, 51

MLE Maximum Likelihood Estimation. 18

NN Neural Network. 12

OSVM One-Class Support Vector Machine. 21, 47, 51

PCA Principal Component Analysis. 14, 15

PDF Propability Density Function. 18

PR Precision-Recall. 14

ROC Receiver Operating Charasteristics. 13, 14, 22

S3 Simple Storage Service. 23

SDK Software Development Kit. 24, 25, 51

SOD Subspace outlier detection. 19, 21, 22, 51

SOM Self Organising Maps. 14–16, 20

SQL Structured Query Language. 24, 30, 36

SVM Support Vector Machine. 12, 21

TN True Negatives. 12

TP True Positives. 12

TPR True Positive Rate. 13

URI Uniform Resource Identifier. 25

1. INTRODUCTION

Banking institutions have an obligation to monitor the transactions that pass through their systems. Currently most used approaches apply static rule based methods to achieve this monitoring. This, however, responds poorly to changes in the data and such rules are hard to optimize. A more powerful and optimizable method would therefore be preferred.

Creating a model that can accurately predict suspicious transactions would allow banks to respond to them faster and more cost effectively. For example credit card frauds and illegitimate transfers could be identified earlier and with greater precision. Overall the model would decrease the need for manual labour and lower the chances of suspicious activities going undetected.

The goal of this thesis is to provide a preliminary filtering and implementation of available methods and approaches for creating and utilizing such a model. This filtering will be done based on a literature survey and an empirical data analysis. Finally the created models will be implemented and tested so that they could be used in a complete system.

The final model should be able to run as a part of the complete system. This means that all parts of the model building process and utilization will have to be automatic. The models themselves can always be improved and they should be built in a way that makes this possible.

There are multiple research papers in the same area as this thesis. However, none of them covers the whole process from theory to application. Some of the important ones on detecting illegitimate transfers are by Chen *et al.* (2018) and Zhang & Trubey (2018). Surveys of the outlier detection methods are more common and the most recent is by Domingues *et al.* (2018). Also the Kaggle competition on Credit Card Fraud Detection by MachineLearningGroup-ULB (2018) has entries that utilize machine learning for detecting credit card fraud.

This thesis starts by providing basic information on transactions and outlier detection. In addition it covers the basic concepts and theory regarding machine learning and provides the needed tools for understanding the outlier detection methods. After the basics are covered the thesis takes a more in depth look at some detection methods and explains some key differences between them.

After explaining how the models work the thesis switches to considering how to create and use them. This starts by going through the tools and environments needed for implementing them. Once the tools are familiar the thesis goes through the architecture plan for the system and explains the role of the model as a part of the whole.

Finally this thesis focuses on the data available and the transformations done to it. This is followed by implementation details about the model training and evaluation. After the implementation details the thesis moves on to the results. In Chapter 6 empirical results based on comparing the models are provided.

2. BACKGROUND

This chapter covers the basic concepts needed for understanding this thesis. It aspires to provide a more general view about transactions and outlier detection than the subsequent chapters that are more focused on the specific task at hand.

The first concept being explained is transactions. Specifically it is explained how transactions are usually modelled in banking systems and how they should be treated in statistical approaches. This section also covers the peculiarities in the type of data a transaction can hold.

The following section deals with outlier detection. It provides a definition for what makes a data item an outlier as well as describes the more general solutions for detecting them.

Finally we have two sections on different methods suitable for solving parts of the outlier detection problem. These sections form a basis for comparing different models. This is then used to explain why the specific methods discussed in Chapter 3 have been chosen.

2.1 Transactions

A financial transaction as such is defined as an agreement between two parties to exchange goods, services or financial instruments (Investopedia 2018). In this thesis we focus only on the money transfer part of this.

One problem with dealing with transactions in real banking systems is the amount of data. A transaction is generated for each transfer of money and this means that the amount of daily transactions can easily be in the millions. This causes restrictions on the applicable methods.

Each transaction has a set of attributes attached to it, among these are of course the timestamp, sender, receiver, and amount transferred. This, single transaction, is the basic building block and it is possible to deduce additional information from it. We can, for example, try to model chains of transactions or calculate frequent receivers for a certain sender (Chen *et al.* 2018). Overall it is useful to gather aggregate information regarding the senders and receivers. It is often also possible to enrich the information about senders and receivers from outside sources.

All clients have their own behaviour that can be considered normal. This leads to the fact that labelling suspicious transactions is strongly dependent on the context. For example someone could send a monthly bank transfer of 700€ and have multiple 0-50€ card purchases weekly. Determining a suspicious transaction could be as simple as denoting all card purchases over 50€ as suspicious. However, if you then have an actual person perform a check based on that it gets expensive really fast. Also it would completely miss cases where a possible thief would use the card for a large amount of small purchases in a very short time frame. Such a limit should also be chosen for each client individually.

With suspicious transactions we wish to catch most, if not all, true positives while keeping the amount of false positives as low as possible. False positives both cost money as well as take resources from investigating the actual cases. This is an important feature to keep in mind when choosing the algorithm.

2.2 Outlier detection

Outlier detection is a problem that has been called by many different names, including anomaly and novelty detection. There seems to be no formal definition for what an outlier is. This is probably because what we wish to consider abnormal is dependent on the problem we are trying to solve. However, the problem and proposed solutions are similar in each of these variations.

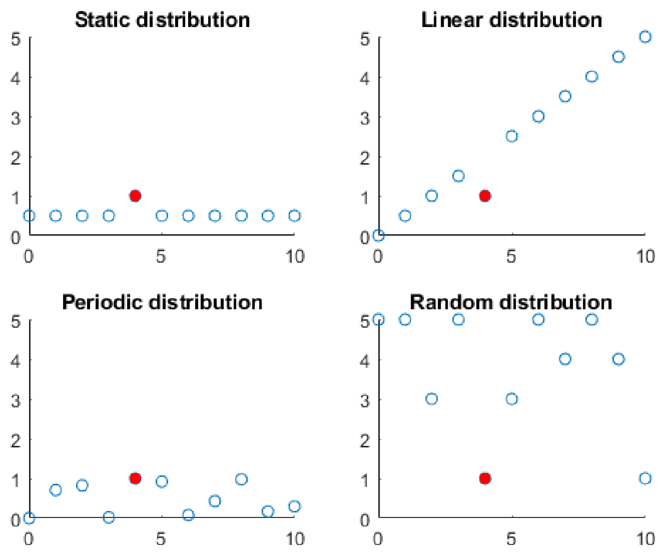

Figure 1. *Examples of outliers in 1D-distributions. Static:* $y = 0.5$, *linear:* $y = x/2$, *periodic:* $x = sin^2(x)$, *random:* $0 \leq x \leq 5$. *Artificially modified points painted with red.*

Some definitions have been given and for example one is: "An outlying observation, or "outlier," is one that appears to deviate markedly from other members of the sample in which it occurs." - (Grubbs 1969)

Very simple examples of outliers in 1D are shown in Figure 1. In cases 1-3 the distributions are based on an underlying model. The easiest way to spot outliers in these is to just fit the data to a model and look for points that differ from the predicted value of the model. The model for both static and linear distributions can be solved by fitting a line with the least squares method. If we used the same linear model for the periodic distribution we could not distinguish the outlying point. In that case we need to find the right periodic model. In case 4 the point is in fact indistinguishable. In real world situations the process of finding the right model is not so simple and different types of solutions are discussed next.

Outlier detection problems have a variety of starting conditions, but the solutions are somewhat similar and can in general be divided into three categories (Hodge & Austin 2004).

The first option is attempt to solve the problem with no prior knowledge of the data. It can be attempted with unsupervised clustering, which can be considered a good starting point for analysing existing data but is computationally expensive. With rapidly changing and large data the luxury of running the analysis for extended periods of time is not there. However the knowledge gained from such clustering can be utilized in training future models.

The second option is to solve the problem using data that has been labelled as normal or abnormal. For this approach one can use supervised classification methods. These methods tend to do well if one has enough samples of both normal and abnormal data. However when looking for outliers it is common to have an uneven distribution of the samples and this can lead to decline in the classification accuracy.

The third option is to have an algorithm model the normal data only. The suitable algorithms for such a problem utilize semi-supervised learning, which is especially useful when one has a large amount of unlabelled data compared to labelled. One way to apply semi-supervised learning is to attempt to achieve low-density separation. (Chapelle *et al.* 2006)

The methods for outlier detection will be discussed in detail in Chapter 3. These methods utilize various machine learning techniques, which will be discussed next.

One should note that machine learning can be divided into three categories. Those categories are very similar to those described above and are as follows: *unsupervised learning*, *supervised learning* and *reinforcement learning*. In the next sections we will take a closer look at unsupervised and supervised learning.

2.3 Cluster analysis

The basic idea of clustering is to group similar data points together. These groups of points are then called clusters. This grouping is not definitive and multiple valid solutions can exist. However, by doing this we gain insight on the data we are analysing. Clustering methods are utilized in several distance based outlier detection methods discussed in Section 3.2.

In general clustering algorithms can be divided into two categories: *partitional* and *hierarchical clustering*. Both of these approaches provide a grouping but the representation is fundamentally different. These categories will be further discussed in the following subsections.

2.3.1 Partitional clustering

The idea of partitional clustering is to, simply, partition the n-dimensional space to different sections. The section boundaries can be either hard or soft. A *hard boundary* means that a point can only belong to a single cluster at a time. If we move the point across a boundary it changes cluster. A *soft boundary* means that a point can belong to multiple clusters at once. The ownership of the point by clusters can then be described by a value. For example, ownership percentage of a cluster rises when we move the point closer to the cluster centroid and when the cluster with the highest value changes we can say that the point has changed cluster.

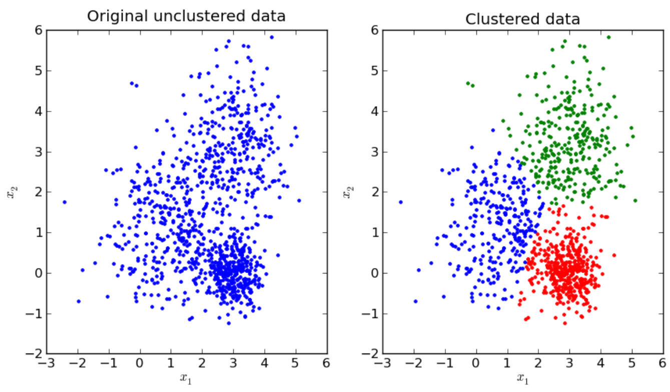

Figure 2*. Example of clustering using k-means and $k = 3$ (Landman 2016).*

An example of partitional clustering can be seen in Figure 2. The clustering was made with the k-means algorithm and choice of 3 clusters. The hard cluster boundaries are evident from this example even though they have not been drawn explicitly.

Next we will shortly consider two variants of the k-means algorithm. The first one is the classic k-means with a hard boundary and the second is Fuzzy C-means which utilizes a soft boundary.

k-means

k-means is one of the most common choices for a partitional clustering algorithm. It is a good starting point because of its simplicity. It was first introduced by MacQueen (1967) and it has many variants which allows its use in various tasks. Next we will shortly go through the basic working principle of the algorithm.

Let Γ be a set of k disjoint clusters $\{C_1, ..., C_k\}$. In k-means we wish to minimize the distance between each data point and the cluster centroids. The cluster centroids are $\beta = \{\mu_1, ..., \mu_k\}$. And they can be calculated from the cluster points with

$$\mu_c = \frac{\sum_{x_i \in C_c} x_i}{|C_c|}, \tag{1}$$

which basically just takes the average of all the points in the cluster as the centroid.

We can take the distance $||x_i - u_c||$ to be the error and then minimize the sum-of-squared error (SSE) (Greene *et al.* 2008). For SSE we then have

$$SSE(\Gamma) = \sum_{c=1}^{k} \sum_{x_i \in C_c} ||x_i - u_c||^2. \tag{2}$$

Now that the problem has been formulated we can take a look at the algorithm itself. The k-means is a greedy algorithm that makes locally optimal choices in hope of yielding a globally optimal solution. This leads to the fact that the results depend on the choice of initial cluster centroids.

One variant of k-means is the Batch k-means that is described in Algorithm 1. It is an offline version of the algorithm, which means that it needs the whole data, or batch, to be available at start time.

The description leaves room for modification in a few crucial places. First is the step $Create-Initial-Cluster-Centroids$, which can be done by choosing random centroid positions. The second step to complete is the choice of the termination condition. Termination condition is the condition that defines when the algorithm is finished. A few good choices are for example: when the cluster centroids move less than δ_d, when number of points changing cluster is less than δ_c and when the running time exceeds a threshold.

Fuzzy C-means (FCM)

The idea of FCM is very similar to that of k-means with the difference that it treats data point ownerships as probabilities. The probabilities of ownerships are stored in the matrix V. From the definition of probabilities we get $\sum_j V_{ij} = 1$. This simply means that the

Algorithm 1 Batch k-means

```
Create-Initial-Cluster-Centroids
while Termination condition is not fulfilled do
# Compute and update the closest centroid for each point
    for $x_i \in X$ do
        Find $min(||X_i - \mu_c||)$ for $1 <= c <= k$
        Update $x_i$ to to closest centroid
    end for
# Update cluster centroids
    for $c = 1 : k$ do
        $\mu_c = \frac{\sum_{x_i \in C_c} x_i}{|C_c|}$
    end for
end while
```

probability that a point x_i belongs in the clusters is 1.

The cluster centroids are then affected by both the locations and probabilities of the data points and thus we should account for both in the equation. According to Cannon *et al.* (1986) Equation 1 then becomes

$$\mu_j = \frac{\sum_{i=1}^{n} V_{ij}^{m} x_i}{\sum_{i=1}^{n} V_{ij}^{m}}, \tag{3}$$

which is similar to a probability weighted average. The effect of probabilities, or fuzziness, of the solution can be controlled by the exponent $m > 1$.

The probabilities should also affect the error function we are trying to minimize. This leads to the fuzzy criterion function (Greene *et al.* 2008):

$$F(\Gamma, V) = \sum_{i=1}^{n} \sum_{j=1}^{k} V_{ij}^{m} ||x_i - u_j||^2. \tag{4}$$

The equation for updating the ownerships is still needed to adapt the k-means algorithm. This equation is given by Cannon *et al.* (1986) and is as follows

$$V_{ij} = \frac{1}{\sum_{c=1}^{k} \left(\frac{||x_i - u_j||}{||x_i - u_c||} \right)^{\frac{2}{m-1}}}. \tag{5}$$

The actual FCM-algorithm solving this minimisation problem is described in Algorithm 2 and is very similar to the simple k-means. The main difference is just the equations used for updating cluster centroids and ownership.

Algorithm 2 Batch fuzzy c-means

```
Create-Initial-Cluster-Centroids
while Termination condition is not fulfilled do
# Compute and update the ownership of each cluster for each point
    for $v_{ij} \in V$ do
        Update $v_{ij}$ according to Equation 5
    end for
# Update cluster centroids, $k$ is the number of clusters
    for $c = 1 : k$ do
        Update $\mu_c$ according to Equation 3
    end for
end while
```

2.3.2 Hierarchical clustering

Hierarchical clustering attempts to form a tree structure of nested clusters (Greene *et al.* 2008). The idea is that some data might naturally be divided into hierarchies. Finding this structure can help in understanding the data better.

One example of data that would work well with hierarchical clustering is animals. First up one has the highest level: invertebrates, fish, birds, mammals and reptiles. The second level that one could choose is the species and under that one could have breed and finally the single animal. The actual hierarchy is larger and has been simplified for this example. A slightly more comprehensive example is shown in Figure 3.

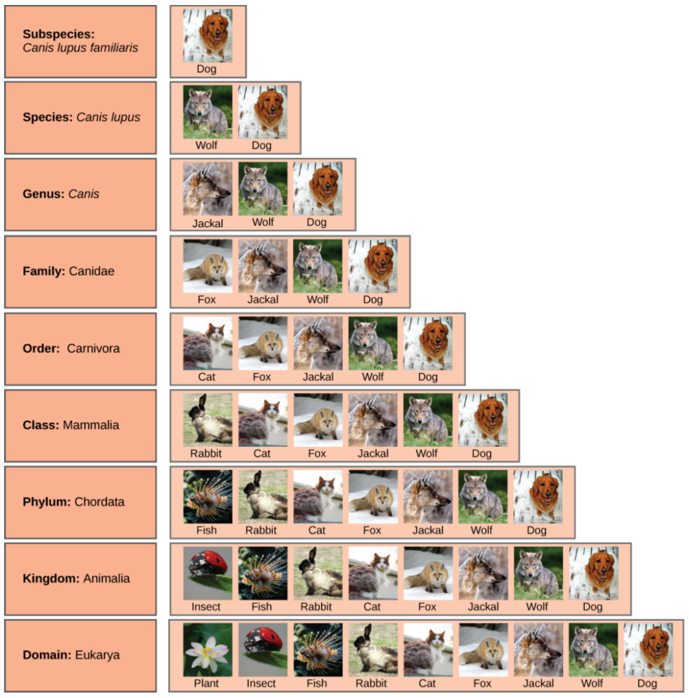

Figure 3. *An example of the hierarchical nature of animal taxinomy. (OpenStax 2012)*

The original clustering of animals has been made with years of work from biologists.

However, one does not usually have years to spend when doing data-analysis and a faster way would be preferred. One would therefore wish to do this computationally and that is what hierarchical clustering is for.

The algorithms implementing hierarchical clustering have two ways of solving the problem. First we have the *agglomerative* (bottom up) approach that starts with each data point in its own cluster and then works to combine together. The second approach is *divisive* (top down) which starts with all the points in a single cluster and works to divide the cluster to several smaller clusters.

2.3.3 Clustering metrics

With clustering it is not obvious how one should asses the performance or accuracy of the algorithms. One has to choose among different methods and use the appropriate metric for the specific problem.

The methods can be divided into three general categories (Greene *et al.* 2008). First, in internal validation the clustering methods are analysed by an evaluation function that is given just the data points and the chosen clusters. Second, in external validation the clustering methods are compared to a predefined reference result. The last category is the stability-based methods where the idea is to measure the difference in results with multiple executions of the algorithm.

Most of the evaluation functions used in internal validation use metrics that attempt to maximize intra-cluster similarity and minimize inter-cluster similarity. A simple way to start modelling this is to model these two separately.

For intra-cluster similarity we have the metric W (within-cluster sum of squares). The equation for W is as follows

$$W = \sum_{c=1}^{k} \sum_{x_i \in C_c} d(x_i, \mu_c)^2, \tag{6}$$

where d is the chosen distance metric and C_c is the c:th cluster. Then for the inter-cluster similarity we have B (between-cluster sum of squares). The equation for B is as follows

$$B = \sum_{c=1}^{k} |C_c| d(\mu_c, \hat{\mu})^2, \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{7}$$

where $\hat{\mu}$ is the centroid of the entire dataset.

As the end result one wishes to obtain a single metric modelling both of these. One way to combine them is the Calinski-Harabasz (CH) index (Calinski & Harabasz 1974) where both of the metrics are normalized and then combined. For CH we have

$$CH = \frac{B/(k-1)}{W/(n-k)}, \tag{8}$$

where a higher value corresponds to a better defined clustering.

One major advantages of the CH index is that it is relatively easy to understand – dense and well separated clusters provide better results. Other advantages include that it is computationally inexpensive.

2.4 Classification

In classification one has an input sample and wishes to assign a label to it. There are numerous methods for classification and this section only covers a small subset of these methods. Specifically methods which are related to outlier detection are preferred. This is because some methods approach outlier detection by formulating it as a one-class classification problem.

Classification methods can be divided into two main groups. These groups are linear and non-linear methods. The main difference between the groups is the form of the decision boundary. Linear methods try to find a hyperplane that separates the classes. Non-linear methods can form more complex decision boundary shapes and can be especially useful when the dataset is not linearly separable.

One important section of classification is the ability to compare results from different models to choose which one is the most suitable for our purposes. Metrics range from simple ones like accuracy to more complex ones like Area Under Curve (AUC).

2.4.1 Linear methods

Linear methods attempt to find a solution that can separate classes by a hyperplane. All linear classifiers do this, but the way in which they approximate weights for the plane can vary.

After finding the hyperplane one can calculate the distance from a point to the plane by

$$g(\mathbf{x}) = \mathbf{w}'\mathbf{x} + w_0, \tag{9}$$

where $\mathbf{x}$ is the point, $\mathbf{w}$ is the weight vector and w_0 is the bias (Duda *et al.* 2000). From this metric it is simple to form a 2-class classifier. For such a classifier we have

$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}'\mathbf{x} > T \\ 0, & otherwise \end{cases}, \tag{10}$$

where T is the threshold between the classes.

There are numerous ways to extend this to n-class problems. One of these is to train n-classifiers and choose the class according to the classifier that has the highest value using Equation 9.

Some examples of linear methods are Logistic Regression and Support Vector Machine (SVM). SVM is an interesting method as it can be made non-linear with the kernel trick.

2.4.2 Non-linear methods

Non-linear methods choose a different approach to the problem and do not need linear separability. In these methods the decision boundary can take more complex shapes and therefore, in theory, model the data more accurately. The downside is that this complex shape can lead to problems with over-fitting.

An interesting example of a non-linear method is the k-nearest neighbour (k-nn) classifier. It simply classifies data points according to k of their nearest neighbours. It is probably the simplest classifier one could use, but still the decision surface ends up being non-linear.

When discussing non-linear one should not overlook deep learning methods such as Neural Network (NN) and Convolutional Neural Network (CNN), which have lately been under a lot of attention. They are prime examples of non-linear methods and can represent almost any form of decision surface possible and learn it if enough data is available.

2.4.3 Classification metrics

Classifiers are almost never perfect and the choosing between them is not easy. Therefore one needs to understand the metrics available to make the correct choice. These metrics are also important when choosing threshold values for the classifier.

The basic building blocks are the classified True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These are shown in Figure 4. All positives is $P = TP + FN$ and all negatives is $N = TN + FP$. These are then combined to get scores like accuracy, precision, and recall. Each of these values is calculated with a set threshold value T and is representative of that special case. (Fawcett 2006)

For accuracy we have the following

$$accuracy = \frac{TP + TN}{P + N}. \tag{11}$$

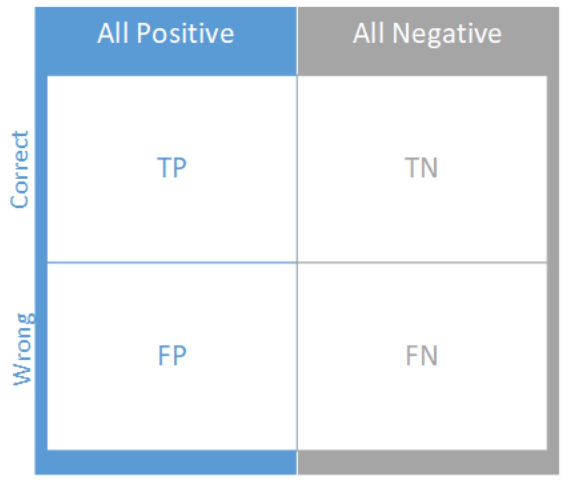

Figure 4. *The division between correctly and falsely classified positive and negative samples.*

Precision is a similar metric and the equation is as follows

$$precision = \frac{TP}{TP+FP}. \tag{12}$$

Finally the equation for recall is

$$recall = \frac{TP}{TP+FN}. \tag{13}$$

Other common metrics include True Positive Rate (TPR), which is identical to recall and False Positive Rate (FPR). For FPR we have

$$FPR = \frac{FP}{N}. \tag{14}$$

Receiver Operating Charasteristics (ROC) and AUC

When one needs to compare classifiers, or thresholds, it would be useful to have a graphical representation for the performance. One way to do this is to use the ROC curve. ROC plots TPR with respect to FPR. (Fawcett 2006)

The ROC-curve makes it possible to determine the dependency between T and $\frac{TPR}{FPR}$. In the case of suspicious transactions this allows us to find a optimal choice for T that maximizes TPR while keeping FPR as small as possible.

While ROC considers specific thresholds and classifiers it would be useful to have a single metric to describe how good a certain classifier is for a problem irrespective of the chosen T. That is where the AUC comes in. AUC is just the integral of the ROC-curve.

ROC is a very common metric for classifier quality. However, in the case of outlier detection it suffers from the problem that the dataset is very imbalanced and the amount of negatives is much larger than the positives. In this case it is advisable to use the Precision-Recall (PR) curve and PR-AUC. (Davis & Goadrich 2006)

2.5 Dimensionality Reduction

The idea of Dimensionality Reduction (DR) is to prune away undesired attributes or dimensions to acquire better accuracy or performance (Greene *et al.* 2008).

DR methods can be divided into supervised and unsupervised. Also the methods can either perform transformations on the features or simply choose an existing subset. Common approaches include Principal Component Analysis (PCA), Linear Discrimant Analysis (LDA), Self Organising Maps (SOM), and Information Gain (IG), which will be discussed further in the coming subsections.

2.5.1 Supervised dimensionality reduction

Supervised DR is similar to supervised classification in that it uses the class labels to make decisions. The difference comes from the fact that with DR one searches for the dimensions that are most important for the classification.

Perhaps the simplest DR method is the IG. It is defined as expected reduction in entropy when grouping the data in the selected dimension. A higher reduction in entropy corresponds to a more important feature. This can then be used as the basis for feature selection.

LDA uses feature transformation to achieve dimensionality reduction. The transformation performed is linear and can be expressed by the equation

$$\mathbf{x}_i' = \mathbf{W}_{k\times p}\mathbf{x}_i, \tag{15}$$

where p is the number of features in the original data $\mathbf{x}_i$ and k is the reduced number of features in $\mathbf{x}_i'$ and $\mathbf{W}$ is the linear transformation. In LDA the transformation matrix $\mathbf{W}$ is calculated by maximizing between-class separation and minimizing within-class separation. (Greene *et al.* 2008)

2.5.2 Unsupervised dimensionality reduction

Unsupervised DR is a very popular method for preprocessing datasets. It does not utilize the class labels so the methods have to find other ways to assess how important certain dimensions are. These methods are especially useful in outlier detection because class labels are often hard or expensive to obtain.

PCA can be thought of as the unsupervised equivalent of LDA. It uses linear transformation as described in Equation 15. However, unlike LDA, it uses the variance of the data to determine importance and chooses the dimensions with the largest variance. This leads to a different solution for the transformation matrix **W**. (Greene *et al.* 2008)

Self Organising Map

Another way to do unsupervised DR is SOM. It is a variation of competitive learning, which is unsupervised learning for artificial neural networks where nodes compete to represent the data. The advantage of SOM is the ability to produce a lower dimensional layout of high dimensional data. This is especially useful for visualization in the case of 2D and 3D layouts. It was introduced by Kohonen (1998).

The basic idea is that one chooses the amount of neurons, or cluster centroids and the underlying lattice structure. The neurons are then updated one data point at a time. The neuron that is the most similar to the data point is updated the most and the update is applied with a distance based damping factor to the surrounding neurons. This leads to closely related clusters being physically close to each other in the lattice.

The most similar node is selected according to

$$c(\mathbf{x}_i) = \underset{j}{\operatorname{argmin}}(d(\mathbf{x}_i, \mathbf{m}_j)), \tag{16}$$

where $\mathbf{m}_j$ is the prototype vectors of the neurons and $d()$ is the chosen distance metric, often Euclidean. Updating the neurons is done by the following equation

$$\mathbf{m}_i' = \mathbf{m}_i + \alpha h_{ic}[\mathbf{x} - \mathbf{m_i}], \tag{17}$$

where $\mathbf{m}_i$ is the node being updated, α is the learning rate and h_{ic} is the neighbourhood function.

The neighbourhood function decreases the learning applied to the nodes away from the selected node c. The decrease is calculated from the distance between the node being updated i and the selected node c in the projection space. The equation is as follows

$$h_{ci} = exp\left(\frac{||\mathbf{r}_i - \mathbf{r}_c||^2}{2\sigma}\right), \tag{18}$$

where σ is the dampening factor. It is often decreased with each iteration to obtain better fine tuned results.

The algorithm for training SOM is described in Algorithm 3, where k is the number of neurons chosen. Note that the neurons are effectively cluster centroids.

Algorithm 3 Self Organising Maps

```
Create-Initial-Lattice-And-Neurons
while Termination condition is not fulfilled do
# Update the neurons for each point in data
    for $x_k \in X$ do
        Choose the most similar neruon $c$ according to Equation 16
        for $i = 1 : k$ do
            Update $\mathbf{m}_i$ according to Equation 17
        end for
    end for
end while
```

3. OUTLIER DETECTION METHODS

This chapter provides more in depth descriptions of the candidate clustering and classification algorithms.

Outlier detection is a broad subject and there are numerous methods available for solving it. Each of these methods seems to also have an almost unlimited amount of variations. In this chapter the idea is to explore some of the more commonly used methods and method types. The methods will be discussed with regard to the specific properties of transaction data.

As the name of outlier detection is not standard so is the case with the types of methods. Different authors propose different divisions. Here we have used some of the method types proposed by Domingues *et al.* (2018) and Pimentel *et al.* (2014). These types will be discussed in the following sections.

There are many different methods but some concepts are shared between them. One of these is the idea to formulate the outlier detection problem as one-class classification. In this approach the data point $\mathbf{x}$ is not mapped to $[-1,1]$ and is instead assigned $z(\mathbf{x}) \in \mathbb{R}$. This z-score is then used as a measure of outlierness and a threshold k is assigned as the decision boundary between normal and abnormal data.

3.1 Probabilistic methods

Most probabilistic methods attempt to derive a density estimation for the normal class. This estimate can then be used to define the probability of a data point belonging to it and thus allowing us to get a value for the z-score. This can be achieved by determining the probability that the sample was drawn from the learned density.

3.1.1 Gaussian mixture model

Mixture model is a model where the samples are assumed to be drawn from a mixture of underlying densities. Gaussian Mixture Model (GMM) is a special case of mixture models where the underlying densities are assumed gaussian. The idea of using GMMs for outlier detection is not new and was already proposed by Aitkin & Wilson (1980).

Propability Density Function (PDF) for mixture models (Duda *et al.* 2000) is as follows

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{c} p(\mathbf{x}|w_j, \theta_\mathbf{j})P(w_j), \tag{19}$$

where $\theta = \{\theta_1, ..., \theta_c\}$ are the unknown parameters for the mixture density, $\mathbf{x}$ the data point, c the number of classes and w_j the state of the environment. The choice of c has large influence over the reliability of the model and should be done with care.

Now when one chooses the underlying function as multivariate gaussian the set of parameters comes known and $\theta_i = \{\mu_i, \Sigma_i\}$. For the multivariate gaussian distribution (Reynolds 2009) one has the following

$$g(\mathbf{x}|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} exp\Big(-\frac{1}{2}(\mathbf{x}-\mu_i)'\Sigma_i^{-1}(\mathbf{x}-\mu_i)\Big). \tag{20}$$

This form poses one problem. It does not have an analytical solution for Maximum Likelihood Estimation (MLE). However, it is possible to look for a good solution iteratively and this can be done with Expectation-Maximization (EM). In EM the idea is to form a base model θ_i and use that to approximate $\theta_{i+1}.\theta_{i+1}$ is then used as the base model and the process is repeated until some stopping criterion is met.

After the model is fitted it is possible to calculate z-score for a new data point x_{test} using Equation 19. Then comparing this with k we can determine whether to point is generated from the same distributions or not.

One downside of GMM is that it can only model data that is representable by Gaussian distributions, the number of clusters needs to be known in advance and it does not work with categorical data. However it is reasonably simple and the results can be understood. It also provides reasonable outlier detection accuracy even with one-hot encoded categorical variables. Other features include that it has good training/prediction time, suffers only moderately from dimensionality and the method is robust. (Domingues *et al.* 2018).

3.1.2 Dirichlet process mixture model

The Dirichlet Process Mixture Model (DPMM) is a non-parametric Bayesian mixture model. The major benefits of it being non-parametric is that the number of clusters does not have to be predefined. It utilizes the Dirichlet process, which is a generalized version of a dirichlet distribution that has infinite dimensions.

In theory a DPMM could use any underlying distribution G_0, but a Gaussian distribution is often chosen for simplicity. This variation is called the Dirichlet Process Gaussian Mixture Model (DPGMM). The non-parametric nature and infinite dimensionality make parameter estimation difficult, but Blei & Jordan (2006) propose a variational estimation for the paramaters.

When using DPGMM the end result is similar to GMM but the number of clusters does not have to be predefined. Overall the tranining/prediction time for DPGMM is low, it handles the amount of features reasonably well and is robust (Domingues *et al.* 2018).

3.2 Distance based methods

Distance based methods use the assumption that outliers should, in fact, be located away from the majority of the data. Some of the most common methods measure distances between neighbours and derive the z-score from that. Another approach is to measure distance to cluster centroids and use that to derive the z-score.

3.2.1 Nearest-neighbour distance

k-nn distance is perhaps the simplest possible way to detect outliers. The parameters that one has to choose is the amount of neighbours to use, k and the distance-metric. The simple Euclidean distance is a good choice for many situations (Pimentel *et al.* 2014). The z-score is calculated as the summed distance from $\mathbf{x}$ to k of its nearest neighbours.

Other similar methods use either the average, median or some other combination of the k-nearest neighbours. All of these methods are, however, susceptible to the problem of dimensionality and their performance drops when the dimensionality of the data is high. This problem with dimensionality can be addressed with the methods described in Section 2.5.

Some algorithms implementing nearest-neighbour distance are described next.

Local outlier factor (LOF) uses the distance between the data point and the kth closest neighbour. It is a simple method with high training/prediction time but seems to perform well on real-world datasets (Domingues *et al.* 2018).

Subspace outlier detection (SOD) is a variation of nearest-neighbour methods that finds shared points between the data point and its neighbours and utilizes subspaces to find the outlier score z. In the study by Domingues *et al.* (2018) SOD was found to be the best performing nearest-neighbour approach.

3.2.2 Clustering based distance

Clustering based distance methods utilize clustering as a sort of preprocessing step to reduce the amount of comparisons between data points. After this it is possible to consider just the distance between $\mathbf{x}$ and the cluster centroids.

3.3 Neural network based methods

Neural network based methods use the regression capability of neural networks. The network is trained so that it can reconstruct a feature from the rest. This predicted feature is then compared with the actual value and the error is used as a measure of the z-score.

3.3.1 Grow When Required network

Grow-When-Required network (GWR) is based on the SOM-network decribed in Subsection 2.5.2. GWR itself was first introduced by Marsland *et al.* (2002). It produces a mapping from high dimensional input space to a lower dimensional output space. The main benefit when compared with SOM is that the network can add new nodes when a data point does not match any existing node with high enough accuracy.

The network consists of nodes with weight vectors and edges that connect nodes together. The edges are created so that similar nodes are joined and form neighbourhoods. Both nodes and edges can be added and removed during the training phase. A node is removed if it has no edges.

New edges are formed between the node that best matches the input and the second best. Each edge is associated with a age parameter that starts from zero. When a new input is compared then the age of the edge between the best and second best nodes is set to zero and for the other edges that are connected to the best node the age is increased by one. Edges are removed when their age is over a threshold a_{max}.

New nodes are added when the activity of the best matching node is under the threshold a_t and the best matching node is not new. Deciding whether a node is new or not is done by assigning a variable counter f_c that decreases exponentially from 1 to 0 when the node is updated. This variable can then be compared to a predefined threshold h_t. The way f_c works gives the GWR a natural ability for novelty detection – if the best matching node has a high f_c then the input can be considered novel.

GWR has decent training/prediction time and resistance to dimensionality. However, it is not robust to noisy data and has one of the worst outlier detection accuracies in the study by Domingues *et al.* (2018). Despite these drawbacks one can find some use for GWR in that it helps in understanding the data.

3.4 Domain based methods

Domain based methods try to produce a model that labels certain partitions of the feature space as normal. When a data point is inside such a partition it is considered normal.

3.4.1 One-class support vector machine

One-Class Support Vector Machine (OSVM) is a special case of SVMt where the function y is trained to predict $+1$ in a region around the training points and -1 everywhere else. Like other SVM methods OSVM is a linear method. However, it is possible to map the data points $x_i \in X$ to a higher dimensional feature space $F_i = \Phi(x_i)$ using the function Φ. Calculating this function directly is not needed and the kernel trick can be used to calculate the mapping.

The equation for the function y is

$$y(x,\mathbf{w}) = \begin{cases} 1, & \mathbf{w}^T\Phi(x) - \rho \geq 0 \\ -1, & otherwise \end{cases}, \tag{21}$$

where $\mathbf{w}$ is the weight vector and ρ is the offset.

OSVM tries to find $\mathbf{w}$ and ρ that separate the training data from origin with the maximum margin. The optimization problem is as follows

$$\begin{gathered} \min_{\mathbf{w}\neq 0,\rho} = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \rho + \frac{1}{\nu N}\sum_{i=1}^{N}\varepsilon_i, \\ \mathbf{w}^T\Phi x_i \geq \rho - \varepsilon_i, \quad \varepsilon_i \geq 0 \ \forall i = 1,..,N, \end{gathered} \tag{22}$$

where ε_i are the slack variables and $\nu \in (0,1]$ is the parameter acts as the upper bound for the fraction of outliers and as the lower bound for the fraction of support vectors.
(Khan *et al.* 2014)

One of the key strengths of OSVM is that it performs well without significant tuning and works well with datasets that contain only a few outliers. It has high training/prediction time, but can cope with a large amount of features. However, it can be outperformed by SOD in many real-world data sets (Domingues *et al.* 2018).

3.5 Other methods

Outlier detection is not limited only to the previously mentioned types. Also the division of the types is constantly evolving and varies from one author to the next. This section describes those methods that do not fit any of the preceding types.

3.5.1 Isolation Forest

Isolation Forest (iForest) is a unique way of doing outlier detection that was introduced by Liu *et al.* (2008). It differs from the other methods described here in the way that it does not produce a model for the normal class. Instead it assumes that outliers are few and different. This assumption yields the property that outliers should be easier to isolate than other points.

The iForest is an ensemble of slightly different Isolation Treet (iTreet). An iTree is a Binary Search Tree (BST) structure with one test added to each internal node. The test attribute q and split point p are chosen at random and the test is simply $q < p$.

The outlier score z is proportional to the average of the path length $h(x)$ from the top of the tree to the matching leaf node in each tree. This average $E(h(x))$ is then normalized to obtain the final result. The normalization is done by using the average path length for an unsuccessful BST search and is given by

$$c(n) = 2H(n-1) - (2(n-1)/n), \tag{23}$$

where $H()$ is the harmonic number and n is the number of data points. The anomaly score is then

$$z(x,n) = 2^{-\frac{E(h(x))}{c(n)}}. \tag{24}$$

The training and prediction of iForest are both good. The training complexity is $O(t\psi log\psi)$, where t is the number of trees and ψ is the subsampling size. The prediction complexity is $O(nt \log \psi)$, where n is the size of the dataset. (Liu *et al.* 2008)

The outlier detection ability of iForest is also among the best in the study by Domingues *et al.* (2018). iForest had the best Precision-Recall AUC score and the second best ROC AUC score.

3.6 Suitability for suspicious transaction detection

When implementing suspicious transaction detection there are a few crucial metrics to consider. The algorithm should be both fast and use a small amount of memory. Having a fast and memory efficient algorithm saves a lot of trouble in the implementation phase as it can be executed without distributed computing.

Another important factor in the case of this thesis is that the methods should have tested and proven implementations, preferably, in Sklearn. This limits the use of some algorithms but is necessary to produce a stable implementation. From the methods described in this chapter this prunes away GWR and SOD.

4. TOOLS AND ENVIRONMENT

This chapter contains a description of the tools and environments used in the empirical phase. The tools are discussed in different levels of detail corresponding to their significance.

In particular the tools used for data exploration and model development, deployment and evaluation are described. In addition some tools regarding application development are also discussed. The choice of tools is limited by the environment in which the data resides and connected applications are running. Security demands limit the use of data outside protected environments, which forces certain choices about development.

In addition to the tools used this chapter also covers the environment in which the analysis is done. In this case the data source containing the transaction data as well as the final production environment both reside in Amazon Web Services (AWS). This cloud environment is a logical choice for such data as the data flows are large and the required computation power can vary. Using a cloud based environment enables scaling and or extending the services to fit our needs.

4.1 AWS

AWS is a cloud service provider that allows users to create scalable services. The possible uses include web servers, storage, analytics, large-scale processing and so on. The available services range from serverless applications to full on virtual servers. AWS is comprised of smaller services at various abstraction levels. In this thesis we limit our discussion to the components that are essential for our problem. (Amazon 2019c)

AWS has a few core components that the rest of the services utilize. The first important for this thesis is Simple Storage Service (S3). S3 is a cloud based storage service that can be used to store and access files (Amazon 2019b). The second common component is the Elastic Compute Cloud (EC2). EC2 is used to create and use cloud computing resources directly (Amazon 2019a).

4.1.1 Snowflake

Snowflake, while not an AWS service, can run on top of AWS. Snowflake is a distributed data warehouse designed for the cloud. In this thesis it serves as the data source for training data. (Snowflake 2019)

Snowflake has a built-in web console that can be used to preview the data and execute Structured Query Language (SQL) queries. This is an useful property for early data exploration.

One of the main features of the architecture of Snowflake is the separation of storage from processing units. This allows for a large amount of concurrent users and allows the choice of different processing power for different user levels.

Overall Snowflake helps in managing the huge amounts of data generated by banking transactions. The Data Warehouse (DW) it provides is essential for this thesis to be possible. All of the data from different underlying systems can be aggregated in one place and then easily accessed from anywhere.

4.1.2 Amazon Sagemaker

Amazon Sagemaker is a machine learning service provided by AWS. Its main use is to help build, train and deploy models to hosted environments. It provides some ready made algorithms that have been optimised for distributed systems. It also features a Jupyter notebook service that enables the use of an integrated Jupyter notebook without the need for additional servers. (Amazon 2019d)

Training and deploying models in python can be done using SageMaker Python SDK (Software Development Kit (SDK)). The SDK provides abstractions for Amazon SageMaker which simplifies the use of normal machine learning libraries in order to provide a simple Application Programming Interface (API) for all of them. The abstractions include estimators, models, predictors, and session. Session can be used to fetch the required AWS Identity and Access Management (IAM) execution role from the SageMaker instance to perform AWS operations. The other abstractions will be explained in the following subsections. (Amazon 2019e)

SageMaker Python SDK uses Docker containers to run the user provided python scripts. It is possible to build your own containers but often it is simpler to use the existing ones. This, however, limits the user to the following additional libraries:

- sklearn
- sagemaker
- sagemaker-containers
- numpy
- pandas
- Pillow
- Python (2.7 or 3.5)

For most applications this set of libraries is enough. Next we will look further into using SageMaker Python SDK to train and deploy Sklearn models.

Training

Estimators provide an abstraction for training. There exists an estimator for Sklearn in the SDK that simplifies training of Sklearn models. The Sklearn estimator is given as parameters a python script, an AWS EC2 instance type and an AWS IAM role. It is also possible to define hyper-parameters and other optional arguments.

After initializing the estimator one can call fit with the training and test data as parameters to start the training process. The training and test data should be provided as a dict of Uniform Resource Identifiert (URIt) to S3. The training data can be gathered to S3 separately from the training script before calling fit. Using the SDK call to fit will make the data available to the training script. After training the provided script should save the trained model as output to S3. A simplified example is shown in Listing 1.

```
import numpy as np
import sklearn
import argparse
import os

if __name__ =='__main__':
    ## Parse the arguments
    parser = argparse.ArgumentParser()
    # hyperparameters sent by the client are passed
    #  as command-line arguments to the script.
    parser.add_argument('--exampleHyperParameter', type=int, default=50)
    parser.add_argument('--modelname', type=str, default="model")
    # Data, model, and output directories
    parser.add_argument('--output-data-dir', type=str,
     default=os.environ.get('SM_OUTPUT_DATA_DIR'))

    parser.add_argument('--model-dir', type=str,
     default=os.environ.get('SM_MODEL_DIR'))

    parser.add_argument('--train', type=str,
     default=os.environ.get('SM_CHANNEL_TRAIN'))

    args, _ = parser.parse_known_args()

    modelname= args.modelname
    ## Load the data
    # Find the files
    train_file_paths = [
         os.path.join.(args.train, file) for file in os.listdir(args.trai
         ]
    # Read the files and convert to features and labels
    train_x = ... #features
    train_y = ... #labels

    ## Train the classifier
    clf = KNeighborsClassifier()
    clf.fit(train_x, train_y)

    ##Save the model
    joblib.dump(clf, os.path.join(args.model_dir,
                    "{}.joblib".format(modelname)))
```

Listing 1*. A simple training script. Modified from Amazon (2019e) examples.*

Model deployment

After training one can use deploy on the estimator to load and deploy the model to a SageMaker Scikit-learn Model Server. Two components of the server are customizable: Model loading and Model serving. (Amazon 2019f)

Model loading implementation should be provided in a python script and must contain a definition for the function $model\_fn(model\_dir)$. The function should deserialize the saved Sklearn model and return a ready-to-use one. An example implementation is provided in Listing 2.

```
from sklearn.externals import joblib
import os

def model_fn(model_dir):
    clf = joblib.load(os.path.join(model_dir, "model.joblib"))
    return clf
```

Listing 2*. Loading the model. Sample from Amazon (2019f).*

Model server

The next part is model serving. The serving is a three part process which consists of input processing, prediction and output processing. Each of these functions should be provided in the python script.

Input processing handles the input received by the SageMaker InvokeEndpoint API. The idea is to deserialize the input to a form the prediction phase can understand. The function *input_fn*(*request_body*, *request_content_type*) receives *request_body* as a byte buffer and *request_content_type* as a python string. The default implementation of *input_fn* for Scikit-learn model server can handle JSON, CSV and NPY encoded data and transform them to NumPy arrays. A simplified version for this function is described in Listing 3.

```python
import numpy as np

# Input processing, return a numpy array
def input_fn(request_body, request_content_type):
    if(request_content_type == 'application/X'):
        # Deserialize depends on the actual type
        data=deserialize(request_body)
        # Load the data
        array = np.load(data)
        return array
    else:
        # In case the request_content_type is
        # unrecognized
        pass
```

Listing 3. *Simplified version for input processing.*

Prediction phase and the prediction function *predict_fn*(*input_object*, *model*) is given as input the output of *input_fn* and the model from *model_fn*. It then uses the model to calculate the required attributes and returns them to the next phase. A simplified version for this function is described in Listing 4.

```python
import numpy as np
import sklearn

# Handle the actual prediction
# receives the array from input_fn
# and model from model_load
def predict_fn(input_object, model):
    pred = model.predict(input_object)
    return np.array(pred)
```

Listing 4. *Simplified version for predicting with Sklearn model.*

Output processing $output\_fn(prediction, content\_type)$ receives the output of $predict\_fn$ and the requested content type from the InvokeEndpoint API. It should then provide the implementation for serializing the prediction to the requested type and return the byte array. Default implementation exists for JSON,CSV and NPY. A simplified version for this function is described in Listing 5.

```
import numpy as np

# Handle serialization of prediction to the
# requested type
def output_fn(prediction, content_type):
    if(content_type == 'application/X'):
        # Seriliaze depends on the actual type
        byte_array=serialize(prediction)
        return byte_array
    else:
        # In case the content_type is
        # unrecognized
        pass
```

Listing 5. *Simplified version for output processing.*

Using the model

Sagemaker provides two main ways of using models. These are an online API endpoint and a batch transform job.

The online API endpoint can handle requests and respond in almost real time. This is useful for cases where you have for example a mobile application that sends in pictures for classification. This option is usually always waiting for new requests.

The other option is creating a batch transform job that takes a large amount of data at once, processes it, and submits the output in, for example, S3. It utilizes the API endpoints in doing this but starts and deletes them automatically.

A sample code for both of these is provided in Listing 6, this also provides the sample for creating the estimator.

```
from sagemaker.sklearn import SKLearn
# Train my estimator
sklearn_estimator = SKLearn(entry_point='train_and_deploy.py',
                            train_instance_type='ml.m4.xlarge',
                            framework_version='0.20.0')
```

```
sklearn_estimator.fit('s3://my_bucket/my_training_data/')

## 1. Create an API endpoint
# Deploy my estimator to a SageMaker Endpoint and get a Predictor
predictor = sklearn_estimator.deploy(instance_type='ml.m4.xlarge',
                                     initial_instance_count=1)

# Data should be a type supported by input_fn
# Response should be supported by output_fn
response = predictor.predict(data)

# Shutdown the endpoint container and delete the configuration
predictor.delete_endpoint()
# Delete the model
predictor.delete_model()

## 2. Create a batch transform
# Create the transformer from the estimator
# Transformer also handles creating and closing endpoints
transformer = sklearn_estimator.transformer(instance_count=1,
instance_type='ml.m4.xlarge')

# Apply the transform
transformer.transform('s3://my-bucket/batch-transform-input')
# The transform output will be saved to s3
```

***Listing 6**. Sample code for creating an API endpoint or batch transform job for an sklearn model. The sample is modified from Amazon (2019e) examples.*

4.2 Development environment

Due to security concerns about the data it can not be brought to the local development environment. However, it is possible to explore and use the data directly in the AWS environment. This is made possible by Amazon Sagemaker and Jupyter notebook.

This means that all the model development is done directly in AWS and only very limited local development is possible. Jupyter notebook can connect directly to Snowflake and fetch the necessary training data. This scripting environment is then used to visualize and analyze the data.

Jupyter notebook also contains a git integration so some common software development methods like feature branches and pull requests can be utilized.

The upside of this sort of a development environment is the relatively easy transfer to production use. The same models that have been developed in Jupyter Notebook with numpy and Sklearn can be transferred to production use with the help of Amazon SageMaker as described in Section 4.1.2.

4.3 Data processing tools

Data processing consists of a few things. One of these is data exploration. Simple previews and visualizations of the data, random samples and whatever possible to get to know the data in question. This is usually good a starting step before starting the number crunching. The number crunching is where most of the value of data-analysis comes from, but it is significantly harder without understanding the data. Both of these have their own tools and next we will go through some of them.

4.3.1 Backbone

As said before the chosen environment was Jupyter Notebook in AWS. Furthermore the choice of kernel was Anaconda and Python3. Now some justification for these choices as they determine further tools that can be used.

Python is the programming language of choice for many data scientists as it is easy to use and has comprehensive libraries available for almost anything. It can handle the dataflow process from start to finish. It can also be used to create the actual application using the models. This makes it an ideal starting candidate for data processing.

Jupyter notebook contains a set of pre installed tools. These include most of the required libraries but it is also possible to download and install additional libraries using pip. It is also easy to use and allows for simple workflow. Using Jupyter notebook in AWS is also simple because it is provided as a service by SageMaker.

4.3.2 Data exploration

In the case of this thesis there are a few different aspects of data exploration. The data can be gone through in detail using Snowflake web-console and SQL-queries. This method is great for early exploration and can give a general idea of the data structure and an idea of scope. However, as there is little to no visualization, it is almost impossible to find trends or distributions in the data.

Also the possibility to go through the data with business specialists was available. This helps give understanding about certain attributes and how they could be transformed to more suitable types, mostly from categorical to numeric. Also specialists can usually point to attributes that might be worth spending time on. This, while not a computer program, can be considered one of the most valuable tools when doing data exploration.

Finally we come to visualization. Visualization is crucial in data exploration and model development as it gives insight to the trends and distributions in the data. Visualization is also useful in evaluation as it should often be possible to find a visualization where the output of the algorithm can be seen to make sense.

One of the tools used for visualization is pandas as it is already existing in SageMaker and thus can be used to plot graphs when training and deploying the final model. Pandas has functionality beyond just visualization and the *DataFrame* it provides is an useful tool in data handling.

Other visualization tools include the matplotlib-library. Many visualization tools in python are built on top of matplotlib and aim to either expand it or make it easier and faster to use. As an example the Figure 9 is made using matplotlib.

4.3.3 Data analysis

Sklearn is perhaps the most important and most used machine learning library when one is not dealing with deep learning. All the algorithms chosen for further exploration have existing implementations in Sklearn. It is the basic building block for most machine learning applications.

Numpy is a math library for python that allows for example fast vector calculations and matrix operations. It is especially useful in the preprocessing step and possible post processing. It is the go-to toolbox when modifying data. It also provides a way to calculate statistics from the data.

Pandas is used for both visualization and data analysis. For data analysis the *DataFrame* object is used. It provides functionality to explore, modify, and transform data. It provides easy statistics from the data and allows the user to perform group and join operations on the data.

5. IMPLEMENTATION

This chapter provides an overview of the system architecture and data-flow pipeline. After these have been covered the chapter goes into more detail about the used dataset and its features. These are followed by discussion about the preprocessing and normalization steps done to the data. The training section focuses on the training of multiple models and hyper parameter search. The final section describes how the model evaluation has been implemented.

5.1 Architecture and data-flow

This section provides an overall view of the different parts and subsystems that make up the whole. The system is divided into two parts: model experimentation and model deployment. The architecture for model experimentation is relatively straightforward so most of the discussion will be on the actual model deployment.

5.1.1 Architecture of the high level system

This system, as most systems, is not meant to be operating independently and so has dependencies on other systems. In fact, what is created in this thesis is just one small component for a larger system.

The high level system is called the *main program*. A very simplified version of this *main program* is provided in Figure 5.

The part that is being created is just a single *rule* that takes in transactions and outputs whether they are suspicious or not. The *main program* runs multiple rules to produce a matrix of $k \times n$, where k is the number of transactions and n the number of rules. Some further actions are taken based on this matrix but they will be omitted in this thesis.

All the rules have to implement a single interface, here called the *transaction flagging interface*. This makes it simple to iterate over them in the *main program*. Using this interface also makes it easy to add and remove rules when necessary.

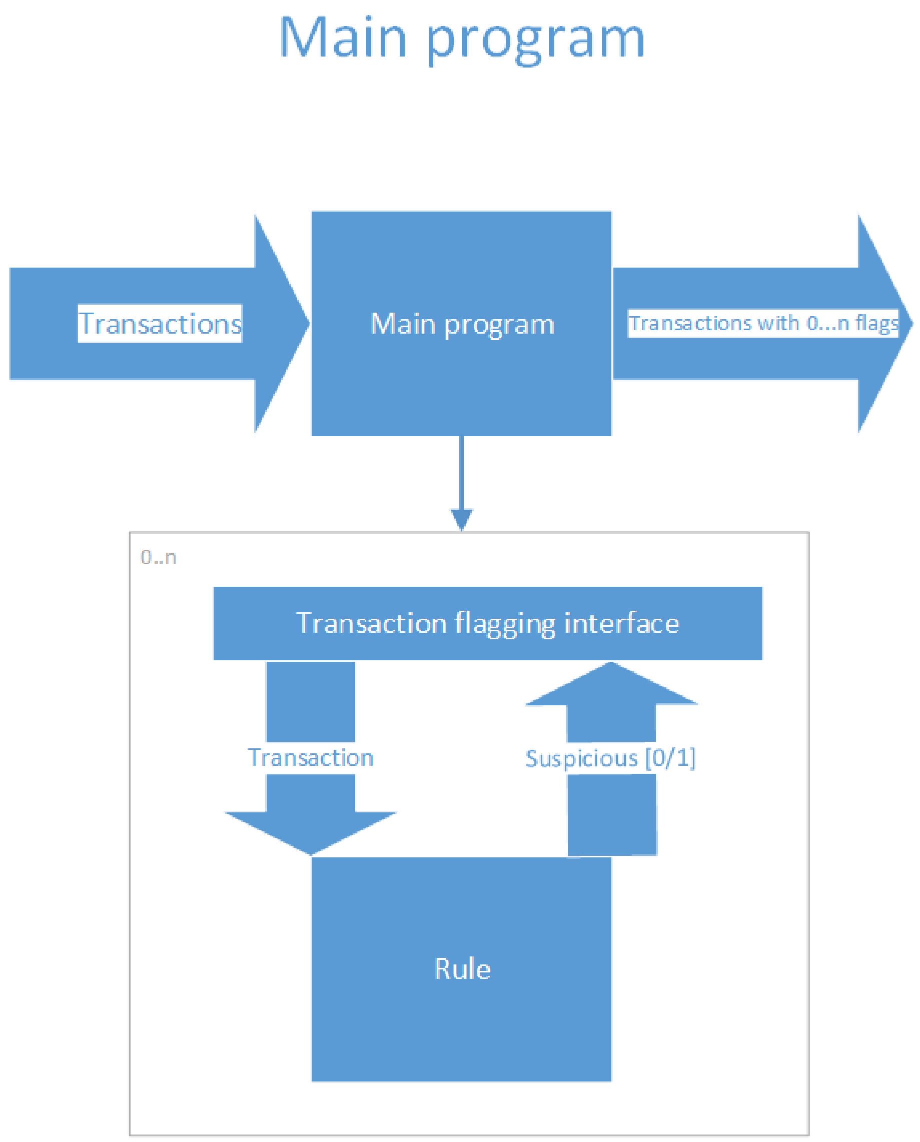

Figure 5. *An overview of the architecture of the high level system.*

5.1.2 AWS Implementation architecture

In addition to the dependency on the larger system this system will be built on AWS. This gives us the opportunity to use SageMaker for model deployment, but also slightly restricts architecture choices. The AWS architecture is described in Figure 6.

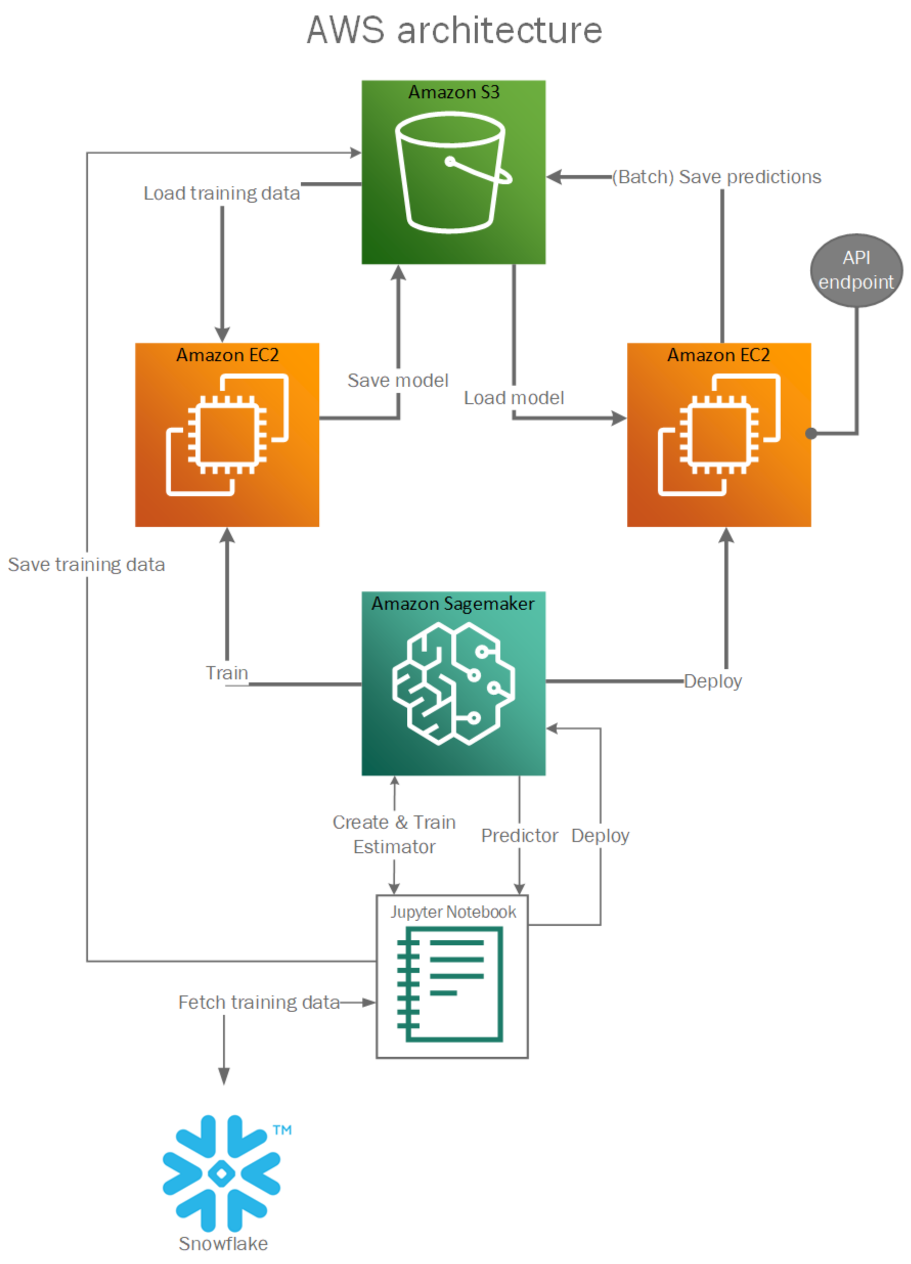

Figure 6. *An overview of the AWS architecture and SageMaker.*

Basically the process of training and deploying a model consists of three parts:

1. Fetch the data from Snowflake in Jupyter notebook and upload it to S3.
2. Create and train an estimator using SageMaker
3. Deploy the estimator and receive the predictor from SageMaker

SageMaker creates the Amazon EC2 instances and provides the parameters and data to the user provided training script. The training script is then run on the EC2 instance. The script loads the data and actually performs the necessary steps to train and save the model.

When deploying SageMaker again creates Amazon EC2 instances and loads the model. The model can then be run as a batch job, saving the results to S3, or a predictor can be created that can be accessed for online use.

5.1.3 Model training

Model training is visible in the AWS architecture description. However, as it is essential for the performance of the program it is explained in further detail here. Also the AWS subsection focuses more on the components and moving data between them.

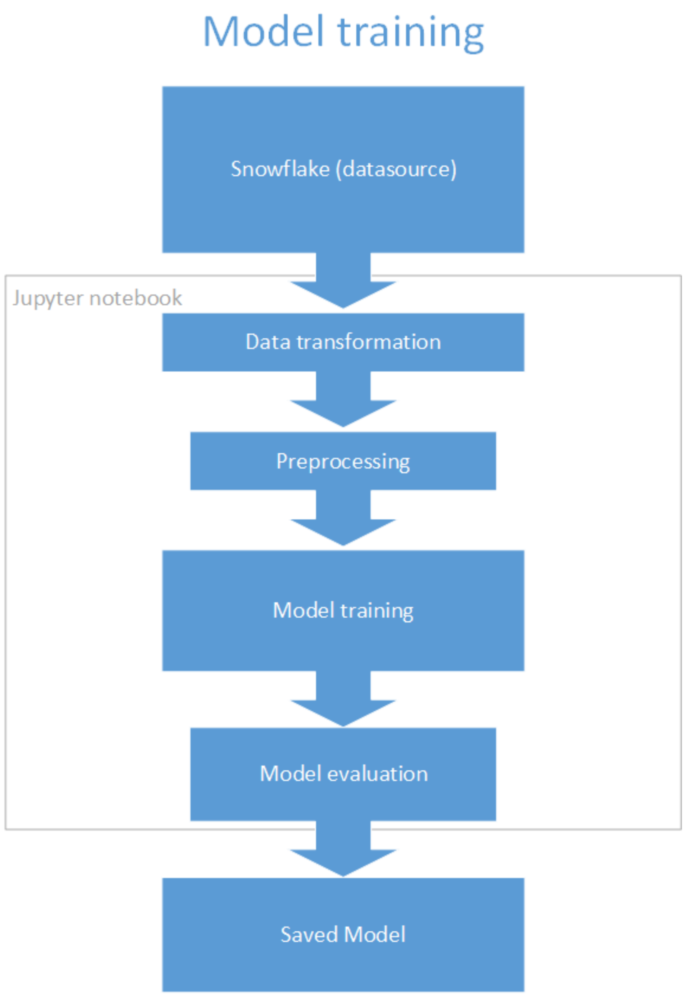

Figure 7. *An overview of the model training process*

Model training is the process of taking a machine learning model and adapting it to do inference based on a set of training data. This process is described in Figure 7.

The starting point of training is fetching the data. In this case it means fetching the data from Snowflake using an SQL query. The first part of data-processing is writing a query that returns all the useful columns as well as joins the tables containing necessary information. The data can be in either one dataset or multiple ones.

After fetching the data might not be in a format that is suitable for training. Most machine learning models do not allow for nested data and this has to be taken into account. This nesting has to be unravelled and this can be done by using additional statistics like averages on the nested values. Also the data format has to be changed to be compatible with the used models. This process is described as data transformation.

Now that the data is in a format that can be modified using conventional tools like Numpy and Sklearn we can start preprocessing the data. This process handles further improvements on the data quality. Features are normalized and depending on the model the categorized features can be one-hot encoded. Also it is possible to use DR algorithms. This step should also clean the data, if possible, so that null and empty or faulty values are handled before training.

The actual training phase is often rather simple. The chosen model is trained using the supplied training algorithm and chosen hyper-parameters. Changing between models is often simple and iteration is needed to choose the right model and parameters.

The model evaluation phase is useful when we are iterating over a number of models and wish to compare their results. We evaluate the performance of our model with separately provided data and possibly save plots and metrics from each model. It is possible to create a condition that if the trained model does not perform better than the previous best one then the model is not saved.

The last phase is saving the trained model so that it can be accessed later.

5.1.4 Outlier detection rule

The end result of this thesis should be a component that implements the *transaction flagging interface*. The flagging process should be done using machine learning approaches. The architecture for this component is described here and can be seen in Figure 8.

Most notably it is divided into three parts: scheduled model training, the saved model and the outlier detection rule. The process for model training was described in Subsection 5.1.3 and all that has been added is a scheduler to run the training at regular intervals. The saved model is saved to and loaded from S3 as described in Subsection 5.1.2 and it is the only dependency between the training and utilization.

Here the architecture for the outlier detection rule is drawn as being online. However

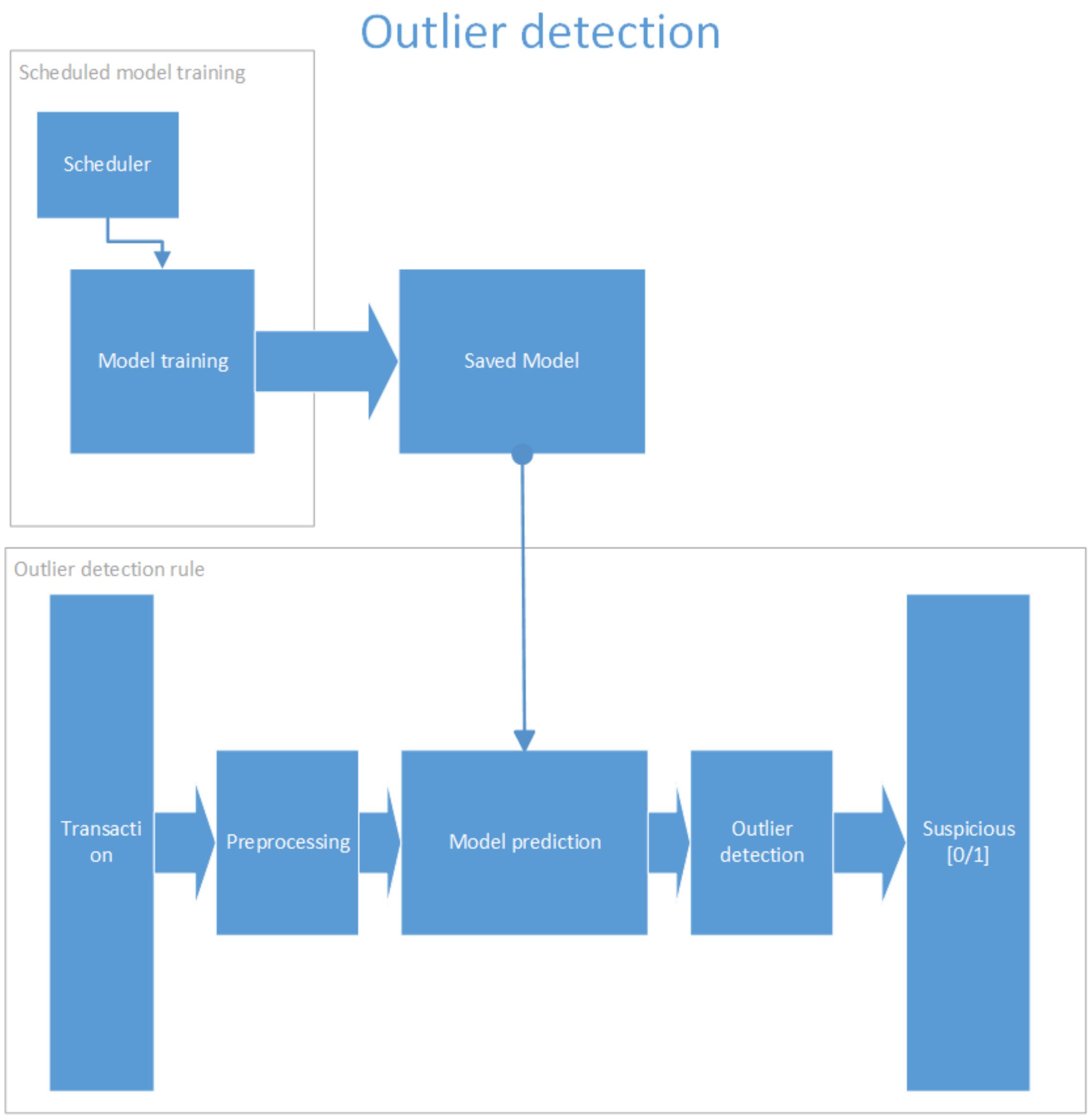

***Figure 8**. An overview of the implementation for the transaction flagging interface provided by the outlier detection rule.*

the final implementation details may differ from this. The steps needed for outlier detection are similar to model training. In this case the preprocessing means modifying the input data so that it undergoes all the same transformations as the training data. This may require saving additional models for label encoding and other transformations.

This same rule architecture is meant to be used with different models. This means that the prediction result is not always directly comparable to the outlier score. An additional step is then needed to extract the outlier score from the model, normalize it to be comparable to other models and then apply a threshold to decide whether to flag the input.

5.2 Dataset

This section covers the dataset used and the transformations and preprocessing steps made. Some details about the data and the chosen features are confidential and will not be described here.

The dataset consists of a pre-sampled set of transactions that has been gathered over a period of time. The data used is from a production environment of a Nordic bank. The amount of rows in the dataset is in the millions.

5.2.1 Available information

The set consists of two data-tables. These are the customer and transaction tables. The relationship between them is one-to-many.

The data descriptions provided here will use the following definitions for type. A link type feature is used to combine multiple tables or in grouping operations. A categorical type feature is a feature that can only take values from a discrete set of pre-defined values. The numeric type features are features that can take numeric values, continuous or discrete, and that preferably has the ability that similarity is relational to $|\mathbf{x}-\mathbf{x}'|$.

The features in customer table are described in Table 1. This table can be linked to the transactions table via the *CustomerId* field. By joining the two tables we are able to provide additional background information for the transactions.

Name	Datatype	Type
CustomerId	Varchar	Link
Nationality	Varchar	Categorical
Date	Date	Numeric
...	...	...

***Table 1**. Subset of features from customer table*

The second table is the transaction table, which is described in Table 2. This is where most of the information regarding the transactions are stored. It is quite natural to group these transactions using the $CustomerId$-field and calculate customer specific statistics to use as additional features. More detailed description of these statistics will be provided in the following subsections.

5.2.2 Feature engineering

Feature engineering is the process of deriving new features from combinations of existing ones. In some applications of machine learning it can be considered to be the most

Name	Datatype	Type
CustomerId	Varchar	Link
Country	Varchar	Categorical
Date	Date	Numeric
Amount	Float	Numeric
Type	Varchar	Categorical
...	...	...

Table 2. *Subset of features from transaction table*

important factor in the performance of the model.

One approach to doing this is using statistics. In this case one can group the transactions by the *CustomerId* and calculate for example the mean, standard deviation, and percentiles of different features. One could then use the following as a feature $DeltaAmount = Amount - \mu(Amount)_{customer}$. An example of this is shown in Figure 9.

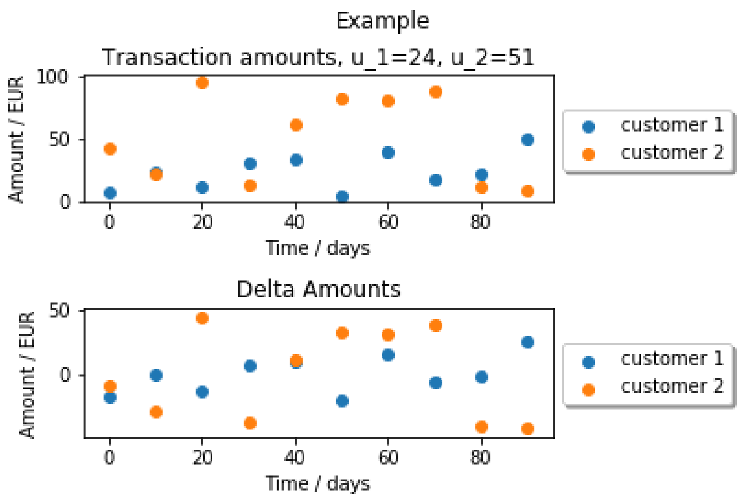

Figure 9. *Plot of transaction amounts and the corresponding DeltaAmount feature. The means are* u_1 *for customer 1 and* u_2 *for customer 2.*

Similar procedure can be used for other features. It would be possible to calculate the average frequency of transactions for a customer. This could be then compared to the time between the current transaction and the last one.

Feature engineering is not limited to using statistics and some problems and models might benefit from using different combinations of the features. In general these combinations could be of any form. However, for example one could use the equation $V_i = (\sum_{ind \in C} V_{ind})^n$, where C is the set of features being combined, V_i is the new feature, and n is chosen freely.

In suspicious transaction detection it is paramount to spend enough effort in feature engineering. The initial information known of transactions is not enough for algorithms to be able differentiate between them. It is a time consuming process of trial and error that is hard to automatize.

When doing feature engineering for suspicious transaction detection one can start with the question: "Is this transaction somehow unique with regard to other transactions of this customer, other transactions of this customer and type, other transactions of this type and all other transactions?". This question is what inspired many of the features used in this thesis.

5.2.3 Feature encoding

Most machine learning models can only work with numerical data. However, in real world data it is very common to have features that are not numerical or their numeric values should not be directly compared. In order to take advantage of the information in these features they have to be encoded to a more suitable format.

One-hot-encoding is perhaps the most commonly used way to encode categorical variables. The idea behind it is simple. The categorical variable is encoded as a $1 \times n$-vector where n is the number of values the categorical variable can take. Each point in the vector corresponds to a different value. To represent a value the vector is filled with zeros but the component corresponding to the value is assigned 1.

5.3 Creating the model

This section covers the application of the candidate models to the available dataset. The first subsection covers the common preprocessing steps made to the data. The second subsection explains how this preprocessed data is then fed to a selection of models. These models include clustering and outlier detection methods. The third subsection provides insight into evaluating the model predictions.

5.3.1 Preprocessing

To simplify the process all models are fed the same data. This allows us to write a single preprocessing script that can be run once. The processed data can then be fed to the models.

The main tools for preprocessing were pandas and numpy. These are described in more detail in Subsection 4.3.3. The preprocessing steps done are shown in Listing 7.

```python
import pandas as pd
import numpy as np
```

```python
# The data is loaded from Snowflake/S3 to a pandas dataframe
unprocessed_data=pd.DataFrame(data=raw_data)

# Only certain features are selected
data=unprocessed_data[['AMOUNT','V1',...'VN']]

# Some feature engineering can be done here
data=data.assign(AMOUNT_DELTA=0)
# Amount mean has to be calculated before this
data['AMOUNT_DELTA']=data['AMOUNT']-data['AMOUNT_MEAN']

# Use pandas to one-hot encode categoricals
data_dummies=pd.get_dummies(data)

# Fill the missing values with 0
data_dummies=data_dummies.fillna(0)

# Normalize certain columns
normalize_cols=['AMOUNT','AMOUNT_DELTA','V1',...,'VN']
for col in normalize_cols:
    col_data=data_dummies[col]
    mean_col_data=np.mean(col_data)
    max_col_data=np.max(col_data)
    min_col_data=np.min(col_data)
    data_dummies[col]=col_data.apply(
        lambda x: (x - mean_col_data) / (max_col_data - min_col_data)
        )
```

Listing 7. *Preprocessing steps done to the data using numpy and Pandas.*

5.3.2 Training

Training the models is relatively straightforward and is mostly handled by Sklearn. The model constructor is given the hyper-parameters and then method fit is called.

Sklearn provides an unified API for clustering and outlier detection models that makes it possible to iterate over models in a for-loop. While the API tries to be general there are differences between the models and not all information is available on every model.

When doing clustering with Sklearn the end result is a vector that has the index of the cluster centroid as the value for each data point. Some models also allow the extraction of the centroid locations. Clustering with k-means is shown in Listing 8.

```
from sklearn.cluster import KMeans

kmeans=KMeans(n_clusters=3,random_state=0).fit(data_dummies)
k_pred=kmeans.predict(data_dummies)
```

***Listing 8**. Using k-means with 3 cluster centroids to cluster the data.*

With outlier detection we are often interested in two values. The prediction, outlier or normal, and the outlier score z. Not all outlier detection models provide z. With Sklearn the predict call returns a vector with -1 for outliers and 1 for normal points. In Sklearn 0.20 it is possible to get the z-scores using score samples. Outlier detection with iForest and Sklearn is shown in Listing 9.

```
from sklearn.ensemble import IsolationForest

model = IsolationForest(contamination=0.01, random_state=2019)
model.fit(data_dummies)
pred = model.predict(data_dummies)
# This can be changed to model.score_samples() in Sklearn 0.20
score = model.decision_function(data_dummies)
```

***Listing 9**. Using IsolationForest with 1% of outliers to predict and score outliers.*

The models are trained with unsupervised methods and this poses another problem. It is hard to validate the results and compare the models. This makes additional training optimizations like hyper-parameter search hard to implement.

5.3.3 Evaluation

Model evaluation in unsupervised training is not simple and definitive evaluation might be impossible. However, some estimation should be possible. Also it is possible to look for cases that are certainly wrong or that should be right.

In suspicious transaction detection we have approximate knowledge of how many data points should be labelled as outliers. Also there exists some knowledge about the transactions that has been withheld from the training algorithms.

When combining this prior knowledge with visualization one can check how the points in the vicinity of data with prior knowledge behave. Also certain features in the data are more important than others and plotting the results with regard to those features can tell whether the model is on the right track.

Visualization script for clusters is provided in Listing 10.

```
import matplotlib.pyplot as plt

# Scatter plot of k—means clustering
# different clusters with different color
fig, ax = plt.subplots()
for i in range(0,3):
    if(i==0):
        color='red'
    elif(i==1):
        color='blue'
    else:
        color='green'

    inds=k_pred==i
    x=data['V_'][inds]
    y=data['V_'][inds]

    ax.scatter(x, y, c=color, label=color,
               alpha=0.3, edgecolors='none')

ax.legend()
ax.grid(True)

plt.savefig("cluster.jpg")
plt.show()
```

***Listing 10**. Plotting script for clustering with 3 cluster centers.*

The visualization results for k-means clustering on the transaction data are shown in Figure 10.

Visualization script for outlier detection is shown in Listing 11.

```
import matplotlib.pyplot as plt

# Scatter plot of the outliers vs normal points
# Outliers are displayed as red
fig, ax = plt.subplots()
for i in [—1,1]:
    if(i==—1):
            color='red'
        elif(i==1):
            color='blue'

        inds=pred==i
        x=data['V_'][inds]
        y=data['V_'][inds]

        ax.scatter(x, y, c=color, label=color,
                   alpha=0.3, edgecolors='none')

ax.legend()
ax.grid(True)

plt.savefig("outlier.jpg")
plt.show()
```

Listing 11. *Plotting script for outlier detection.*

The corresponding visualization for iForest applied to transaction data is shown in Figure 11.

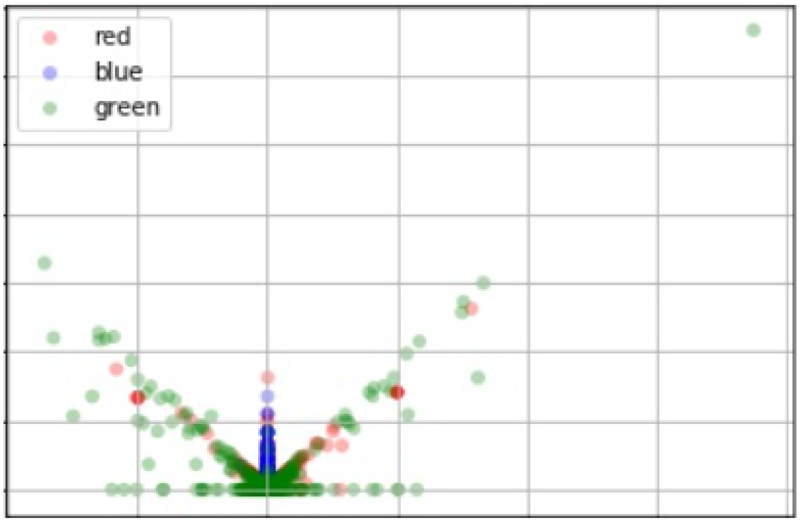

__Figure 10__. Plot of k-means clustering with 3 cluster centers.

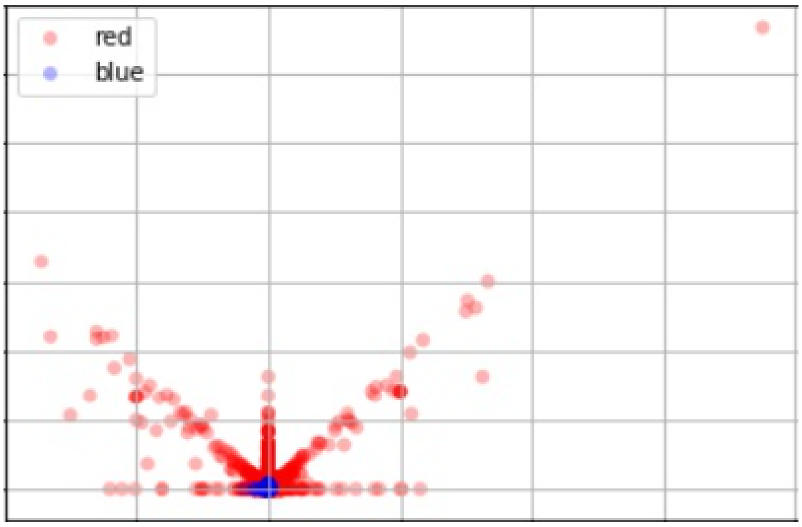

__Figure 11__. Plot of IsolationForest outlier detection results. Outliers are shown as red and normal points as blue.

6. RESULTS

This chapter covers the most important results from different methods. Clustering results are considered separately from outlier detection.

As described in Subsection 5.3.3 the evaluation of unsupervised learning methods is not straightforward. In the case of unlabelled transactions we just have to focus on the attributes that we can compare. These include running time with different amount of samples, CH-index for clustering, and visual comparison of plots.

Additional information, like amount of outliers classified per model, are available but will be withheld. Also based on the results described here some predictions will be selected for expert review and labelled. After this labelling it will be possible to produce more meaningful comparison between the algorithms. However, such a review is time consuming and the findings confidential. Furthermore it is worth noting that a globally optimal solution for this problem does not exist and the choice of the final model is a business decision.

The clustering and outlier detection methods were run on AWS hosted EC2 instance of type *ml.t2.xlarge* with 16gb ram and 4 vCPU:s. Using this instance type limits some approaches due to the algorithms running out of memory. However the dataset was also sampled to make it easier to work with. This scaling was done so that when using a larger instance then the whole dataset should compute in a decent time and have no memory problems.

6.1 Clustering

The results provided in this section have been calculated using multiple different sampling parameters to provide insight on how the methods scale.

Name	CH-Index / N	Running time
k-means	4,41	1min 3s
GMM	1,23	1min 7s
DPGMM	1,08	10min 10s

Table 3*. Clustering results for different methods at a smaller amount of samples. The amount of samples was N ≈1 000 000 and features $f = 10$.*

The results for the initial clustering with a lower amount of samples are shown in Table 3. In this case the DPGMM ran for 10 minutes and ended with the warning that it did not converge. Both k-means and GMM had no issues in performance.

Name	CH index / N	Running time
k-means	4,54	1min 47s
GMM	1,24	3min
DPGMM	0,78	23min 8s

***Table 4**. Clustering results for different methods at a moderate amount of samples. The amount of samples was N ≈2 000 000 and features $f = 10$.*

When increasing the amount of samples k-means starts to differentiate as the fastest one of the three. At this point GMM is starting to slow down, but it is still fast enough to be usable. DPGMM was still not converging at 20 minutes. These results are shown in Table 4.

k-means was the best performing method with both sampling parameters and had both the fastest time and the best CH index. Results for GMM and DPGMM could be improved with better parameter tuning. However, with these results we can conclude that it is enough to perform outlier detection only on GMM instead of both GMM and DPGMM. This is because GMM outperformed DPGMM in both running time and CH index.

6.2 Outlier detection

In the case of outlier detection without a test set of labelled samples it is difficult to provide a good estimate for the accuracy of the methods. This is why this thesis focuses on comparing the running time and only limited exploratory comparison is made. The running times for the algorithms are provided in Table 5.

Name	Running time
iForest	2min 1s
OSVM	-
LOF	4h 21min
GMM	1min 37s

***Table 5**. Outlier detection running time for different methods. The amount of samples was N ≈1 000 000 and features $f = 10$.*

From these results it is evident that even the downsampled size of N is too much for some of the algorithms. These include LOF which after parameter tuning to reduce training and prediction time ended up with a running time of over 4 hours. The other method that suffered from the large amount of samples was OSVM. With the given setup it failed to complete at all.

Both iForest and GMM provide a fast running time regardless of the data size. This makes them ideal for suspicious transaction detection. iForest has less room for parameter tuning but from exploration and visualizations the results seem good. GMM is a method that is really dependent on choosing the right parameters, but when the parameters are found it provides good and explainable results.

6.3 Credit card fraud detection

The open dataset in Kaggle by MachineLearningGroup-ULB (2018) makes it possible to further test the capabilities of the chosen algorithms with labelled credit card transaction data. This, however, is not directly comparable with the results from the different features and data used in other parts of this thesis. Despite of this shortcoming it can be used to obtain useful metrics for the models. These metrics should provide insight into the methods and also determine whether outlier detection is a suitable method for identifying fraud.

The methods chosen for this experiment were GMM and iForest based on their performance in the previous sections. Both of them are unsupervised and their thresholds were not optimized with regard to precision or PR AUC to test whether it would be possible to choose an appropriate threshold T based on the known percentage of frauds. In this dataset it was 0.172%.

The dataset was trimmed down and only two features: V14 and V17 were used. The scatter plots for the detection results are shown Figure 12 and Figure 13. When comparing these to Figure 14, which has the actual frauds, it is possible to see that the methods have found the majority of the fraudulent cases.

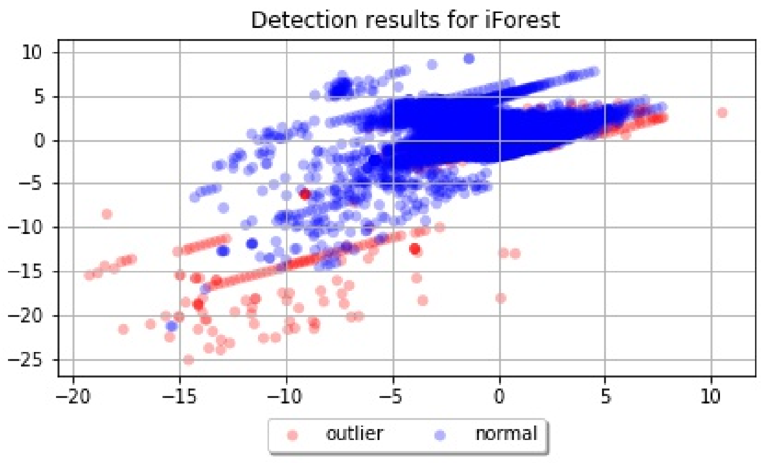

Figure 12*. A scatter plot of detection results for iForest on the credit card fraud dataset.*

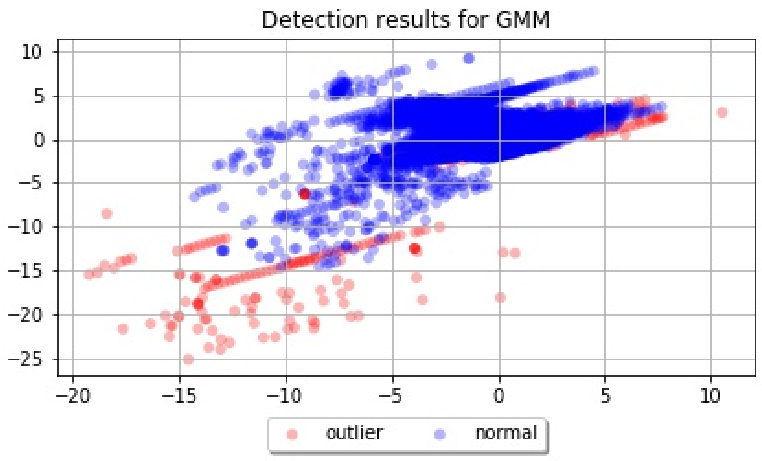

Figure 13. *A scatter plot of detection results for GMM on the credit card fraud dataset.*

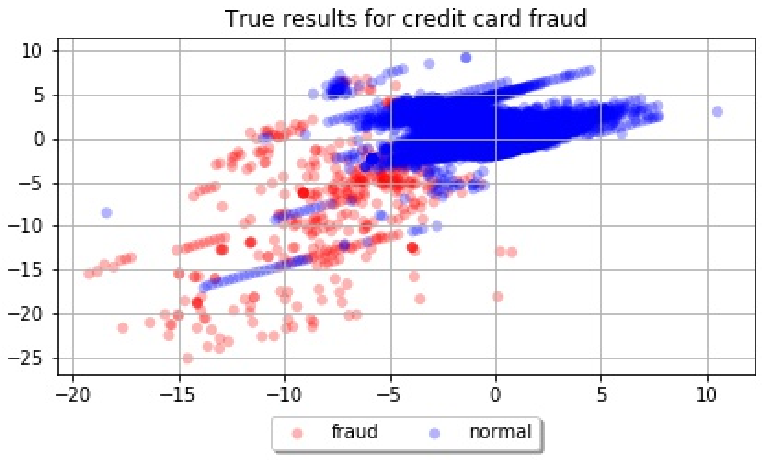

Figure 14. *A scatter plot of actual frauds in the credit card fraud dataset.*

Name	Precision (fraud)	Recall (fraud)	PR AUC
iForest	0.60	0.60	0.57
GMM	0.70	0.66	0.66

***Table 6**. Results from classifying credit card frauds with outlier detection methods.*

The labelled datasets made it possible to calculate more meaningful metrics for the methods. These results are shown in Table 6. The precision on both of the methods is promisingly high as the methods had no prior knowledge of the transactions being fraudulent. The recall is less than would be ideal, but it can be increased by adjusting the threshold values. The dependency between precision and recall is shown in Figure 15 and Figure 16. This can be used to guide the business decision of choosing the threshold.

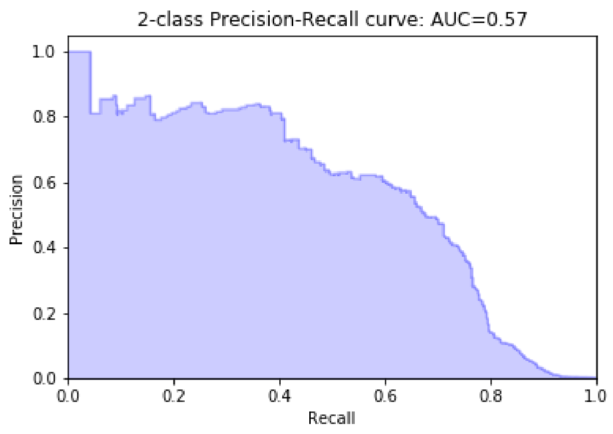

***Figure 15**. The precision-recall curve for iForest on the credit card fraud dataset.*

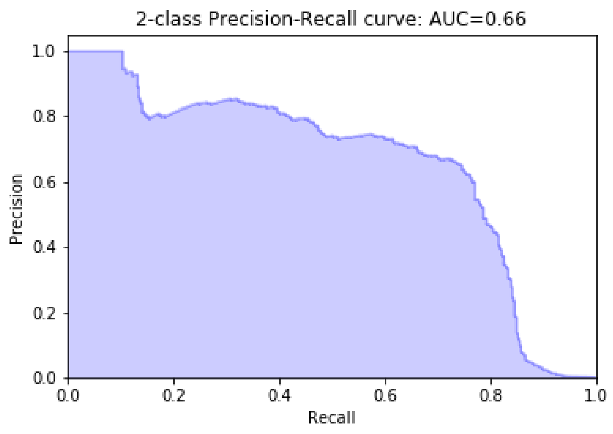

***Figure 16**. The precision-recall curve for GMM on the credit card fraud dataset.*

Overall both GMM and iForest perform well in credit card fraud detection and outlier detection can therefore be considered a good solution to this problem. The difference between the quality of the methods is small but in favour of GMM.

7. CONCLUSION

The goal of thesis was to find and test suitable outlier detection methods for suspicious transaction detection. At first the thesis approached the subject from a more theoretical viewpoint. Suitable machine learning techniques and metrics were discussed and finally a set of plausible outlier detection methods were presented.

These theoretical methods were narrowed down by the fact that we required the algorithms to have stable implementations in Python. Finally these algorithms were run on Jupyter Notebook in AWS and their results and running times could be compared.

Name	Implementation	Running time	Not redundant	PR AUC (fraud)
GWR	N	-	-	-
SOD	N	-	-	-
OSVM	Y	N	-	-
LOF	Y	N	-	-
DPGMM	Y	Y	N	-
GMM	Y	Y	Y	0.66
iForest	Y	Y	Y	0.57

Table 7*. Comparing and filtering the potential detection methods using the checks provided in the columns. N= The check did not pass, Y= The check passed, and -= The check was not run.*

The progressive filtering of the methods is shown in Table 7. The point of the column *not redundant* is to remove methods that are very similar to other possible methods but have worse performance.

The final result was that only two of the methods, GMM and iForest, were chosen as prime candidates for outlier detection on datasets of the required size. Both of them are implemented in Sklearn so they can be used with the SageMaker Python SDK to be used as a part of the complete system. Of these candidates GMM had a higher PR AUC on the credit card fraud dataset used for testing the performance and could therefore be preferred. However, the difference is small enough that changes in the features could change it the other way around.

Further research is needed to determine the optimal parameters for both of the models and to check the validity of the detection results. Also expanding the plausible algorithms outside those having stable implementations in python could bring new possibilities.

REFERENCES

Aitkin, M. & Wilson, G.T. (1980). Mixture Models, Outliers, and the EM Algorithm. Technometrics, Vol. 22(3), pp. 325–331.

Amazon (2019a). Amazon Elastic Compute Cloud. Available: https://docs.aws.amazon.com/ec2/?id=docs_gateway

Amazon (2019b). Amazon Simple Storage Service. Available: https://docs.aws.amazon.com/s3/?id=docs_gateway

Amazon (2019c). Amazon Web Services. Available: https://aws.amazon.com/

Amazon (2019d). AWS Sage Maker what is. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html

Amazon (2019e). SageMaker Python SDK. Available: https://github.com/aws/sagemaker-python-sdk

Amazon (2019f). Scikit-learn SageMaker Estimators and Models. Available: https://github.com/aws/sagemaker-python-sdk/tree/master/src/sagemaker/sklearn

Blei, D.M. & Jordan, M.I. (2006). Variational inference for Dirichlet process mixtures. Bayesian Analysis, Vol. 1(1), pp. 121–143.

Calinski, T. & Harabasz, J. (1974). A Dendrite Method for Cluster Analysis, Vol. 3.

Cannon, R.L., Dave, J.V. & Bezdek, J.C. (1986). Efficient Implementation of the Fuzzy c-Means Clustering Algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8(2), pp. 248–255.

Chapelle, O., Schölkopf, B. & Zien, A. (2006). Semi-supervised learning. MIT Press, Cambridge, Mass. Available: https://mitpress.mit.edu/books/semi-supervised-learning

Chen, Z., Khoa, L.D.V., Teoh, E.N., Nazir, A., Karuppiah, E.K. & Lam, K.S. (2018). Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review. Knowledge and Information Systems, Vol. 57(2), pp. 245–285.

Davis, J. & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. ACM, pp. 233–240.

Domingues, R., Filippone, M., Michiardi, P. & Zouaoui, J. (2018). A comparative evaluation of outlier detection algorithms: Experiments and analyses. Pattern Recognition, Vol. 74, pp. 406–421.

Duda, R.O., Hart, P.E. & Stork, D.G. (2000). Pattern Classification. John Wiley and Sons, Incorporated, Somerset. Available: http://ebookcentral.proquest.com/lib/tut/detail.action?docID=699526

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, Vol. 27(8), pp. 861–874.

Greene, D., Cunningham, P. & Mayer, R. (2008). Unsupervised Learning and Clustering. Springer Berlin Heidelberg, Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval, Berlin, Heidelberg. Available: https://doi.org/10.1007/978-3-540-75171-7_3

Grubbs, F.E. (1969). Procedures for Detecting Outlying Observations in Samples. Technometrics, Vol. 11, pp. 1–21.

Hodge, V. & Austin, J. (2004). A Survey of Outlier Detection Methodologies. Artificial Intelligence Review, Vol. 22(2), pp. 85–126.

Investopedia (2018). Transaction. Available: https://www.investopedia.com/terms/t/transaction.asp

Khan, N.M., Ksantini, R., Ahmad, I.S. & Guan, L. (2014). Covariance-guided One-Class Support Vector Machine. Available: http://www.sciencedirect.com.libproxy.tuni.fi/science/article/pii/S0031320314000077

Kohonen, T. (1998). The self-organizing map. Neurocomputing, Vol. 21(1), pp. 1–6.

Landman, Pang, W. (2016). k-means clustering. Available: https://brilliant.org/wiki/k-means-clustering/

Liu, F.T., Ting, K.M. & Zhou, Z. (2008). Isolation Forest. In: 2008 Eighth IEEE International Conference on Data Mining, 2008. , pp. 413–422.

MachineLearningGroup-ULB (2018). Credit Card Fraud Detection. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. Proc.Fifth Berkeley Symp.on Math.Statist.and Prob., (Univ.of Calif.Press), Vol. 1, pp. 281–297.

Marsland, S., Shapiro, J. & Nehmzow, U. (2002). A self-organising network that grows when required. Neural Networks, Vol. 15(8), pp. 1041–1058.

OpenStax (2012). Biology, 10.8 ed. OpenStax Cnx Biology.

Pimentel, M.A.F., Clifton, D.A., Clifton, L. & Tarassenko, L. (2014). A review of novelty detection. Signal Processing, Vol. 99, pp. 215–249.

Reynolds, D. (2009). Gaussian Mixture Models. Springer US, Encyclopedia of Biometrics, Boston, MA, pp. 659–663. Available: https://doi.org/10.1007/978-0-387-73003-5_196

Snowflake (2019). Snowflake. Available: https://www.snowflake.com/product/

Zhang, Y. & Trubey, P. (2018). Machine Learning and Sampling Scheme: An Empirical Study of Money Laundering Detection. Available: https://doi.org/10.1007/s10614-018-9864-z