UPPSALA
UNIVERSITET

UPTEC STS 19032
Examensarbete 30 hp
Juli 2019

Algorithmic trading surveillance

Identifying deviating behavior with unsupervised anomaly detection

Frans Larsson

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Algorithmic trading surveillance

Frans Larsson

The financial markets are no longer what they used to be and one reason for this is the breakthrough of algorithmic trading. Although this has had several positive effects, there have been recorded incidents where algorithms have been involved. It is therefore of interest to find effective methods to monitor algorithmic trading. The purpose of this thesis was therefore to contribute to this research area by investigating if machine learning can be used for detecting deviating behavior.

Since the real world data set used in this study lacked labels, an unsupervised anomaly detection approach was chosen. Two models, isolation forest and deep denoising autoencoder, were selected and evaluated. Because the data set lacked labels, artificial anomalies were injected into the data set to make evaluation of the models possible. These synthetic anomalies were generated by two different approaches, one based on a downsampling strategy and one based on manual construction and modification of real data.

The evaluation of the anomaly detection models shows that both isolation forest and deep denoising autoencoder outperform a trivial baseline model, and have the ability to detect deviating behavior. Furthermore, it is shown that a deep denoising autoencoder outperforms isolation forest, with respect to both area under the receiver operating characteristics curve and area under the precision-recall curve. A deep denoising autoencoder is therefore recommended for the purpose of algorithmic trading surveillance.

Handledare: Patric af Ekenstam
Ämnesgranskare: Kaj Nyström
Examinator: Elísabet Andrésdóttir
ISSN: 1650-8319, UPTEC STS19 032

Populärvetenskaplig sammanfattning

Teknikutvecklingen har de senaste decennierna bidragit till en fundamental förändring av hur de finansiella marknaderna fungerar. Från att ha varit ett område som kännetecknats av manuellt arbete, har genombrottet av algoritmhandel resulterat i bl.a. högre konkurrens och lägre transaktionskostnader.

Trots att algoritmhandel har haft flera positiva effekter, har det förekommit att algoritmer varit inblandade i incidenter på de finansiella marknaderna. Det är därför viktigt för aktörer som deltar i algoritmhandel att implementera effektiva verktyg och kontrollsystem som kan användas för att upptäcka avvikande beteenden hos handelsalgoritmer.

Eftersom den datamängd som användes i studien inte var märkt med etiketter formulerades problemet som ett oövervakat anomalidetektionsproblem. Efter en översiktlig genomgång av populära metoder inom oövervakad anomalidetektion valdes två modeller: *isolation forest* och *deep denoising autoencoder (DDAE)*. Isolation forest är en algoritm som har utvecklats specifikt för att upptäcka anomalier och bygger på antagandet att anomalier är ovanliga och annorlunda. Därför bör det vara lättare att isolera anomalier jämfört med normala observationer. DDAE är en typ av neuralt nätverk vars syfte är att komprimera en given datamängd för att sedan rekonstruera den till den ursprungliga strukturen. Eftersom anomalier per definition är ovanliga bör modellen således inte vara lika bra på att rekonstruera anomalier jämfört med normala observationer. Därför kan rekonstruktionsfelet, d.v.s. skillnaden mellan indata och rekonstruerad data, användas som ett mått på avvikelse.

I och med att datamängden inte var märkt med etiketter var en utmaning med oövervakad anomalidetektion hur utvärderingen av modellerna skulle genomföras. För att kunna jämföra modellerna med varandra konstruerades därför syntetiska anomalier som kombinerades med verklig data.

Utvärderingen av modellerna visar att både isolation forest och DDAE har förmågan att identifiera avvikande beteenden. Vidare var DDAE överlägsen isolation forest med avseende på båda prestationsmåtten som användes

i studien. Därför förefaller DDAE vara en lovande modell att använda för övervakning av algoritmhandel.

I och med att utvärderingen av modellerna enbart grundar sig på syntetiska anomalier vore det intressant för vidare forskning att undersöka om resultatet står sig när modellerna appliceras på verkliga anomalier. I och med att DDAE var överlägsen isolation forest vore det även intressant att jämföra DDAE med andra modeller som ligger i framkant.

Acknowledgements

This master's thesis is the result of the final project in the Master's Programme in Sociotechnical Systems Engineering at Uppsala University.

First of all, I would like to express my gratitude to my supervisor at SEB, Patric af Ekenstam, for constant support and encouragement throughout the project. Also, thanks to everyone at SEB that have contributed to this thesis by sharing your expertise.

I would also like to thank my subject reader Kaj Nyström, professor at the Department of Mathematics at Uppsala University, for interesting discussions and valuable input during the spring. A special thank you to Kristiaan Pelckmans at the Department of Information Technology, Division of Systems and Control at Uppsala University for sharing your expertise on anomaly detection.

Finally, I would like to thank Malin Örnberg for your patience and constant support during my university studies.

Frans Larsson
Uppsala, June 2019

Contents

List of Tables

List of Figures

Nomenclature

$\mathcal{X}$	A training set.
$\mathcal{V}_N$	A validation set without injected artificial anomalies.
$\mathcal{V}_i$	A validation set with injected artificial anomalies.
$\mathcal{T}_N$	A test set without injected artificial anomalies.
$\mathcal{T}_i$	A test set with injected artificial anomalies.
$\mathcal{A}$	A set of all anomalous records.
$\mathcal{N}$	A set of all normal records.
$A(t)$	A set of all predicted anomalies at threshold t.
$N(t)$	A set of all predicted normal records at threshold t.
$\mathcal{F}$	A set of all features in the data set.
$\mathbf{x}^{(i)}$	The i^{th} record in the data set.
$x_j^{(i)}$	The value of the feature $\mathcal{F}_j \in \mathcal{F}$ for the i^{th} record.

1 Introduction

The financial markets have changed dramatically the last decades and are no longer what they used to be. Developments in technology and new innovations have changed how financial markets operate and have contributed to more global and complex financial markets [1]. From being a labor-intensive field with high transaction costs, the breakthrough of algorithmic and high-frequency trading has resulted in both lower transaction costs and lower volatility in the financial markets [2]. Today, brokers need to use at least some kind of electronic system in order to execute an order. A part of this revolution is the introduction of algorithmic trading which has contributed to a more competitive market as well as cheaper execution [3]. The speed of execution has also become faster, ranging from 0.5 to 1 millisecond for a market order. Thus, high speed of execution is today not only limited to high frequency trading [4].

However, despite that algorithmic trading has had several positive effects on financial markets, there have been recorded incidents of algorithms running amok. One example is the Flash Crash of May 6, 2010 when the Dow Jones Industrial Average dropped close to 1000 points and recovered 600 points within 20 minutes [5]. It has been argued that the main cause was high-frequency algorithms that aggressively sold positions which resulted in a rapid decrease in prices [6]. It is therefore of interest for actors involved in algorithmic trading to implement effective control systems which can detect deviating behavior and flag rogue algorithms. Also, the increase in algorithmic participation in financial markets have attracted interest from regulators and several new regulations have been introduced to deal with the new landscape. One such major regulation is the Markets in Financial Instruments Directive II (MiFID II) which covers algorithmic trading within the EU [7]. Thus, finding effective ways to identify algorithms that do not behave or function as expected are of interest for participants in the financial markets, as well as for regulators.

1.1 Purpose

The purpose of this thesis is to investigate if machine learning can be used to monitor algorithmic trading by detecting deviating behavior. More specific, this thesis aims to evaluate and compare different machine learning models which have been chosen after a review of the machine learning literature. To limit the scope of the thesis, only algorithms that are active on equity markets are taken into consideration. The main reason for choosing equity and not any other financial instrument is that equity is the most traded asset class when it comes to algorithmic execution [2]. Furthermore, only execution algorithms[1], i.e. algorithms that are responsible for executing an order, are studied.

1.2 Contributions

The problem of how to identify behavior and strategies in algorithmic trading has attracted some interest in the machine learning literature. Previous studies have for instance found ways to identify if a trade has been executed by a high-frequency trading algorithm and to distinguish between human and algorithmic execution [8]. For example Yang et al. [8] use inverse reinforcement learning based on Gaussian processes to classify trading strategies and Hayes, Beling & Scherer [9] apply a supervised learning approach with recursive partitioning. Although there exist a few studies on how algorithmic behavior can be identified, none has yet, to the best of my knowledge studied specifically how *deviating* behavior can be detected using machine learning. This thesis will therefore focus on how deviating behavior can be detected rather than how certain behavior or strategies can be identified.

In order to see if machine learning techniques can be used to monitor algorithmic trading, an unsupervised anomaly detection approach was chosen. The problem of detecting anomalies is a difficult task, especially when it is not guaranteed that the data set used for training only consists of normal records [10]. Furthermore, every data set contains some kind of noise [11]

[1]The reader is referred to section 2.2 for a more thorough discussion regarding the definition of execution algorithms.

which also pose a challenge since noise and anomalies can overlap [12]. Since real world data sets, often to some degree, are contaminated, this study contributes to the existing machine learning literature by showing the effectiveness of the chosen algorithms *isolation forest* and *deep denoising autoencoders* when applied to real world data. Furthermore, Ahmed et al. [11], which review anomaly detection for financial fraud detection, note that obtaining real world data sets can be very difficult and that studies therefore often have to rely on synthetic data sets. A problem with this is that the performance of anomaly detection algorithms may be overestimated. Hence, it is of interest to see how anomaly detection models perform on real world data. Although the evaluation of the models in this thesis is based on artificial anomalies, the majority of the data set consists of real world data.

1.3 Structure

The structure of the thesis is as follows. Section 2 introduces the reader to some important concepts within the market microstructure literature and discusses how algorithmic trading works. Section 3 introduces the different kinds of anomalies often mentioned in the machine learning literature followed by a discussion on three different modes of anomaly detection methods: *supervised*, *semi-supervised* and *unsupervised*. Furthermore, the chosen approach, unsupervised anomaly detection will be argued for. Section 4 begins with a short description of the data set used in this thesis followed by a discussion of how the data was processed before being used in the models. In section 5, methods based on unsupervised learning are introduced briefly and the chosen methods *isolation forest* and *deep denoising autoencoder* are discussed. Also, the anomaly scoring approach used in this thesis is introduced. Section 6 discusses how the models can be evaluated when the nature of anomalies are not known beforehand, i.e. when the ground truth is missing. Finally, the thesis ends with presenting the results and conclusions in section 7 and section 8 respectively.

2 Background

The purpose of this section is to provide the reader with a fundamental understanding of financial concepts relevant for algorithmic trading. The section introduces parts of the market microstructure literature in section 2.1 followed by an overview of algorithmic trading in section 2.2.

2.1 Market microstructure

Market microstructure is a research area which focuses on trading of financial assets and the structure of markets and trading venues. Since these topics are relevant in order to understand algorithmic trading, some important concepts are summarized in this section. More specific, section 2.1.1 will give the reader a brief understanding of the limit order book and section 2.1.2 will discuss how exchanges are structured.

2.1.1 Limit order book

Before defining a limit order book, which is a central concept for algorithmic trading, let's define a limit order as in definition 2.1.

Definition 2.1. A *limit order* is an order where the price to sell or buy an asset is specified by the trader.

In other words a limit order is a way for a trader to get control over the execution price of a security. For example if a trader wants to buy asset A with a limit price of \$10, the trader will only buy the asset if the price is less than or equal to \$10. Similarly if a trader wants to sell the same asset with a limit price of \$10, the trader will only sell asset A if the price is greater than or equal to \$10.

One common way to organize exchanges today is to use limit order books in which all limit orders received by the exchange are recorded [2]. Figure 1 illustrates the structure of a hypothetical limit order book with the security price on the x-axis and the order quantity on the y-axis. All the buy orders are on the left side of the price of \$10 and all the sell orders are on the right

side of the price of \$10. The limit buy orders with the highest price are commonly known as the *best bid* and similarly the limit sell orders with the lowest price are commonly referred to as the *best ask* [2].

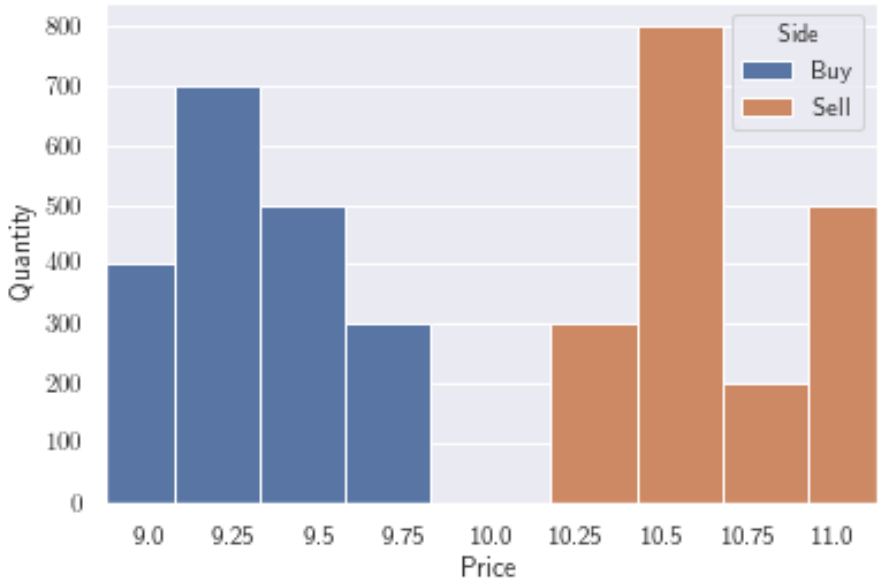


Figure 1: *Illustrative example of a limit order book.*

With an understanding of how the limit order book is structured, a market order can be defined as in definition 2.2.

Definition 2.2. A *market order* is an order to buy or sell a security at the best available price, i.e. at the best ask price or the best bid price respectively.

In the order book example in figure 1, a market buy order will be executed at a price of \$10.25, as long as the quantity of the order is less than or equal to 300. However, if the size of the market order is larger than the size of the limit orders at the best ask price, the market order will sweep through the order book at other prices as well [2]. For example if the size of the market order is 500, only the first 300 will be executed at a price of \$10.25 since that is all the liquidity available at that price. The remaining 200 will be executed at the next best ask price, i.e. at \$10.50.

2.1.2 Exchanges

The structural change in how trading operates has had large effects on the structure of exchanges and has created opportunities for new types of exchanges to emerge. New trading venues have for example entered the market and offer services that are of interest for specific needs. Speed and access are two competitive advantages that trading venues try to achieve but they also compete in what kind of order types they accept [4].

To fully understand the behavior of execution algorithms it is also necessary to make a distinction between *lit venues* and *dark pools*. Unlike lit venues, dark pools are trading venues which do not reveal their limit order book [2]. This characteristic of dark pools appeals to some investors since it allows trading a large quantity of financial assets without displaying their intentions to other market participants [1].

2.2 Algorithmic trading

There is no consensus regarding the definition of algorithmic trading [13] and some definitions are quite general. For instance, algorithmic trading can according to Treleaven et al. [14] refer to "[...] any form of trading using sophisticated algorithms [...] to automate all or some part of the trade cycle." (p. 76) [14]. However, trading algorithms can be divided into two different categories depending on their objective. This is discussed in section 2.2.1 followed by a more thorough introduction to execution algorithms in section 2.2.2.

2.2.1 Types of algorithms

The objectives of algorithmic trading can be divided into two main categories, proprietary trading and agency trading [15]. *Proprietary trading* covers trading where firms or financial institutions trade financial instruments on their own account [1]. *Agency algorithms* aims to optimize the execution of trades, often by minimizing execution costs given some kind of benchmark [15]. The focus of this study will as mentioned in the introduction be on agency trading.

Thus, proprietary trading will not be considered and trading algorithms will from now on refer to algorithms designed for agency trading, i.e. execution trading algorithms.

2.2.2 Algorithmic trade execution

When an execution algorithm with a given strategy receives an order, it has to consider several variables in order to decide how the order should be executed. One variable with significant importance is the size of the order. If the order is large it has to be sliced into smaller orders, hereafter referred to as *order slices* or *child orders*, to minimize market impact. Thus, the algorithm has to, given input from the market, decide if the order should be sliced and how large each of the child orders should be [1]. This is a very important decision, since for example Cont & Kukanov [16], find that the choice of how to slice an order has a large effect on the optimal mix between market orders and limit orders when solving an optimal execution problem.

Another aspect an algorithm has to consider is the pace of the order execution, i.e. how fast the child orders should be submitted to the market. If the order should be executed as fast as possible it may be preferable to use market orders, which are executed at the best available price at that time. However, if time is not as important the algorithm may achieve better prices by submitting limit orders and wait for them to be filled. The risk of submitting limit orders however, is that there may be price movements which leads to a worse price resulting in an unfilled order [1]. Thus, execution risk has to be considered when placing a limit order [16].

Because a security can be listed on several different markets it is not enough to focus on when to execute an order and how the order should be sliced in order to minimize market impact. One must also consider *where* an order slice should be executed. Therefore, *smart order routing* (SOR) is used to decide on which market a child order should be executed at. Thus, the purpose of smart order routing is to achieve the best possible price for a given order slice and time [17].

3 Anomaly detection

Anomaly detection is a special problem in the machine learning and data mining domain and can be defined as "[...] the problem of finding patterns in data that do not conform to expected behavior." (p. 15:1) [18]. This section is introduced with an overview of different types of anomalies in section 3.1 followed by a discussion of the different modes of learning in section 3.2.

3.1 Anomalies

Three different types of anomalies can be identified in the machine learning literature: point anomalies, contextual anomalies and collective anomalies [18]. A *point anomaly* is a record which deviates from the rest of the data. This is, according to Chandola et al. [18], the simplest type of anomalies. A *contextual anomaly* is a record that should only be defined as an anomaly in a particular context. For example a record might be considered to be an anomaly during a specific time, but should be classified as normal at a different time [18]. According to Goldstein & Uchida [19] the majority of the anomaly detection algorithms are designed for detecting point anomalies. However, these models may be able to detect contextual anomalies as well if the context is included as additional features [19]. The third type of anomalies are *collective anomalies* which can be described as sets of records which are behaving differently compared to other records and should therefore be defined as anomalies. For example a subsequence of events may be considered to be a collective anomaly, although the individual events should be considered normal when occurring in other parts of the sequence [18].

3.2 Modes of learning

Methods for anomaly detection can be divided into three different modes: supervised learning, semi-supervised learning and unsupervised learning [18]. *Supervised learning* aims to predict some kind of outcome measure given a number of input variables. A crucial underlying assumption of the supervised learning mode is the availability of labels for each class, i.e. the value of the

outcome variable is known for all the records in the training data. The supervised anomaly detection approach may at first sight be no different than an ordinary binary classification problem. However, the nature of an anomaly problem raises two issues for a supervised classifier when used for anomaly detection. First, since anomalies by definition are rare the data will be very imbalanced which in itself is a difficult challenge. Secondly, labeling data is a very manual process which requires the knowledge of a domain expert and is thus both time consuming and expensive. This is especially true for anomalies since what is considered abnormal today may change in the future, making the labeling process even more complicated. Obtaining labels which is required in order to train a supervised model is therefore often not feasible [18].

An alternative approach to supervised learning which requires less labeled data is *semi-supervised learning*. Using semi-supervised learning methods to detect anomalies does not, unlike supervised learning, require that the data includes records that have been labeled anomalous. Instead the methods assume that only records from the normal class[2] are available in the data set [18].

As noted above, both supervised learning and semi-supervised learning require labeled data, although the latter is more flexible since it only demands data from one class. This was a critical issue for this study since the available data set was unlabeled. Also, due to costs and time associated with labeling data it was not realistic to manually label each record. Therefore *unsupervised learning* was chosen which, unlike previously mentioned methods, does not require a labeled data set. Instead, unsupervised learning models often function under the assumption that anomalies are very rare in the data set [18]. Also, as mentioned above, one challenge with anomaly detection is that what is considered normal may change with time. Thus, the fact that unsupervised learning does not use any labels could be considered an advantage.

Even though unsupervised learning was the best alternative for the ap-

[2]It should be noted that there are methods which assume that the data set consists of only anomalies, although such methods are quite uncommon [18].

plication considered in this thesis, there were some challenges that had to be dealt with. When the training data set in unsupervised learning has no labels there is no clear way to measure how successful an unsupervised approach is and how its performance compares to other unsupervised methods. This is a difference compared to supervised learning for which relative straightforward approaches are available to measure how successful an algorithm is for a given problem and data set [20]. How this issue was dealt with is discussed in section 6.

4 Data and preprocessing

This section provides a general description of the data used in this thesis, followed by a discussion of how the data was processed before it was used as input to the models.

4.1 Data description

The data used in this study consisted of audit trail data on order and transaction level. Since the focus of this thesis was on algorithmic order execution the main part of the data set consisted of messages received and sent by the algorithms. These messages contained information, such as the size of a child order, type of order, price, time stamp and event type, e.g. insertion or cancellation of a child order. This information can be used to reconstruct how an order was executed. However, this data, hereafter referred to as raw data, was not appropriate to use as input for anomaly detection algorithms. Therefore, extensive preprocessing of the data, such as data transformation and feature engineering was required before the machine learning models introduced in section 5 could be trained. During the data transformation of the raw data, further discussed in section 4.2, more than ten million audit trail messages were processed and analyzed. After preprocessing, data transformation and feature engineering almost one million records were used to train the models, around $160,000$ records were used for validation and about $60,000$ records were used for the final test. In addition to the audit trail messages, the data set was enriched with market data, e.g. market prices and market volume, with the purpose of adding more context and thus making contextual anomalies possible to detect.

Based on the characteristics of features they can be categorized into two main categories: static features and sequential features. *Static features* are variables which do not change over time for a given observation while *sequential features* are variables that *do* change over time [21]. Sequential data can further be divided into two additional categories, *time series* and *event data*, with the difference that, unlike time series, event data has an uneven

time period between two events [18]. The majority of the raw data used in this study can be characterized as event data since the time between two messages is not constant. However, since some information does not change over time for an order execution, some of the data can be described as static.

4.2 Data transformation

Many of the existing methods for anomaly detection require the data to be structured as feature vectors. Sequential data must therefore be transformed into feature vectors if such methods are to be used [22]. When considering data transformations there are different approaches available that may be more or less suitable depending on the application and domain.

One approach, referred to as *time spatialization*, is to first restructure every observation with sequential features into a data matrix. Let n_d denote the number of sequential features for each observation and l_d the length of each sequence. The corresponding data matrix for each observation can then be flattened and transformed to a vector in $\mathbb{R}^{n_d \cdot l_d}$ [21].

Table 1: *Representation of sequences as feature vectors using time spatialization.*

	$Temporal$							$Static$		
	t_1			$\dots$	t_{n_T}					
	$\mathcal{F}_1^{(1)}$	$\dots$	$\mathcal{F}_{m_T}^{(1)}$	$\dots$	$\mathcal{F}_1^{(n_T)}$	$\dots$	$\mathcal{F}_{m_T}^{(n_T)}$	$\mathcal{F}_1$	$\dots$	$\mathcal{F}_{m_S}$
$\mathbf{x}^{(1)}$										
$\mathbf{x}^{(2)}$										
$\vdots$										
$\mathbf{x}^{(n)}$										

To transform the raw data using time spatialization consider the following. Let Δt be the length of each interval or *time window* and let T be the whole time period for the longest sequence in the data set. Furthermore, let n_T be the number of time windows, i.e. $n_T = \frac{T}{\Delta t}$ and denote each time window by $t_1, t_2, ..., t_{n_T}$. For each time window, m_T different features can be

extracted and the sequences can be represented as feature vectors with length equal to $n_T \cdot m_T$. Each of these features is denoted by $\mathcal{F}_j^{(t)}$ where t specifies from which time window the feature is extracted from and j is the type of feature extracted from the sequences. Because the data set also contained static features the final representation of the data resulted in $m_T \cdot n_T + m_S$ features, where m_S is the number of static features for each record. The final feature representation of the data set is illustrated in table 1. However, since the duration of an order execution can vary dramatically, the transformed data became very sparse. After experimenting with this approach it was therefore concluded that it was not suitable for the data set.

An alternative approach which does not have the data sparsity problem is to use a *window-based* technique and assign an anomaly score to each window [23]. This means that T anomaly scores can be estimated for an order execution spanning over T time windows. The idea is illustrated in table 2 where the notations are the same as in table 1 with the addition that n_i is the number of records during time window t_i. An advantage with this approach is that the data set becomes non-sparse. However, this approach has the disadvantage that the relationship between different time windows for the same order execution is lost. To make this information loss less severe, some of the features considered included historical information about the order executions.

Table 2: *Representation of sequences using a window-based technique.*

		Temporal			*Static*		
		$\mathcal{F}_1$	$\dots$	$\mathcal{F}_{m_T}$	$\mathcal{F}_{m_T+1}$	$\dots$	$\mathcal{F}_{m_T+m_S}$
t_1	$\mathbf{x}^{(1)}$						
	$\mathbf{x}^{(2)}$						
	$\vdots$						
	$\mathbf{x}^{(n_1)}$						
t_2	$\mathbf{x}^{(n_1+1)}$						
	$\mathbf{x}^{(n_1+2)}$						
	$\vdots$						
	$\mathbf{x}^{(n_1+n_2)}$						
	$\vdots$						
t_T	$\mathbf{x}^{(n-n_T+1)}$						
	$x^{(n-n_T+2)}$						
	$\vdots$						
	$\mathbf{x}^{(n)}$						

An important parameter for the window-based approach is the size of the window Δt [23]. For example if the time window is too small a model may have difficulties detecting anomalous patterns that occur over time. The window size used in this study was decided upon after consulting with domain experts to make sure that the window size was reasonable.

When using a window-based technique, an anomaly score is only assigned to a specific time window for a specific order execution. Therefore, in order to classify a whole order execution as normal or anomalous an approach for aggregating the anomaly scores over all time windows is needed [23]. This is discussed in section 5.4.

From now on, to avoid confusion, *records* will be referring to a vector with features for a specific order execution and time window denoted by $\mathbf{x}^{(i)}$. Furthermore, the set of all the records belonging to the i^{th} execution is denoted by $\mathcal{E}_i$ and the set of all records belonging to the i^{th} time window is denoted by $\mathcal{W}_i$.

4.3 Feature engineering

The feature engineering process is crucial for the performance of a machine learning model. However, when the ground truth in the data set is not known beforehand it is challenging to identify which features that have explanatory power. The features used in this thesis have been selected after consulting with domain experts and by trial and error. The choice of features has therefore been an iterative process which has evolved over time. The final set of features consisted of 16 different variables and included for example the number of messages for each event type, e.g. the number of new, canceled and traded child orders.

To make it possible for the anomaly detection models to detect contextual anomalies, the context was as suggested by Goldstein & Uchida [19] added as additional features. One straightforward way to add context to the data set was to add the time to the set of features. Another contextual feature used was the total volume of the order.

One of the drawbacks with transforming the data using a window-based technique is that the relation between time windows is lost. Therefore, in order to make this information loss less severe, some of the features included in the final data set did not only include information about the current time window but also information extracted from all preceding time windows. One such example was the cumulative traded volume up until the end of the time window divided by the total volume of the order.

4.4 Normalization

The performance of machine learning techniques are often affected by the magnitude of the features. For example if the magnitude of one feature is very large compared to the rest of the features it may dominate. It is therefore often recommended to *normalize* the features so that they have the same magnitude. Two popular approaches to perform normalization are *min-max normalization* and *z-score normalization*. A min-max normalization which maps each value to the interval $[0, 1]$ is defined in (1) where $x_j^{(i)}$ is the value for the i^{th} observation for feature $\mathcal{F}_j$ and $\tilde{x}_j^{(i)}$ is the corresponding normalized

value. Furthermore, $\min_{1\leq k\leq n} x_j^{(k)}$ and $\max_{1\leq k\leq n} x_j^{(k)}$ are the minimum and maximum values respectively among all observations for feature $\mathcal{F}_j \in \mathcal{F}$ [24].

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \min_{1\leq k\leq n} x_j^{(k)}}{\max_{1\leq k\leq n} x_j^{(k)} - \min_{1\leq k\leq n} x_j^{(k)}} \tag{1}$$

Z-score normalization is defined in (2) where μ_j is the mean of all the values for feature $\mathcal{F}_j$, $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$ and σ_j is the standard deviation for the same feature, $\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)^2$ [24].

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \tag{2}$$

Figure 2 illustrates the behavior of the two normalization methods. Since the choice of normalization often depends on the application, both of the normalization techniques have been tested throughout the study.

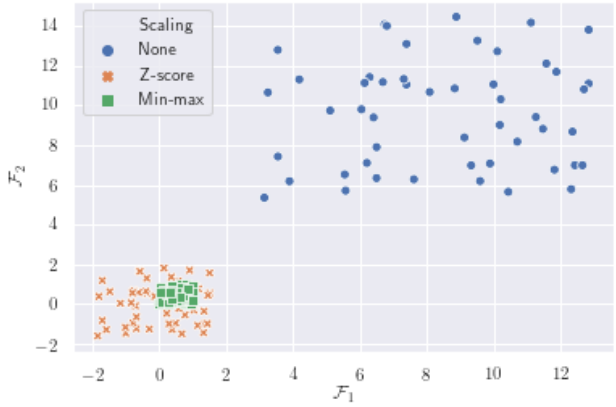


Figure 2: *Illustrates the difference between z-score normalization and min-max normalization.*

5 Modeling

This section is introduced with an overview of different categories of methods that can be used for unsupervised anomaly detection. The overview is not meant to be comprehensive, but rather meant to illustrate the focus of the literature study that was conducted at an early stage of this thesis. The chosen algorithms, *isolation forest* and *deep denoising autoencoder*, are discussed in more detail in section 5.2 and in section 5.3. Given the vectorization of the data as discussed in section 4.2, the algorithms was only able to assign an anomaly score to a specific time window for a specific order execution. Therefore, an approach for aggregating the anomaly scores for each order execution is defined in section 5.4.

5.1 Overview

There are several classes of methods that can be used for unsupervised anomaly detection. *Clustering-based* methods is one popular category of algorithms which is based on the idea that anomalies can be identified by dividing the records into clusters based on some similarity measure. The records which do not belong to any cluster, are furthest away from the center of a cluster or belong to small or sparse clusters can then be defined as anomalies. A drawback with these methods is that their main aim is to identify clusters and therefore their ability to find anomalies may not be optimal [18]. Also, it is found in [19] that cluster based techniques often don't perform as well as *nearest neighbor-based* algorithms and therefore the latter methods are recommended if computation time is not an issue. The methods based on the nearest neighbor-based approach are instead of focusing on clusters, concentrating on the local neighborhood of each record. These methods can for example either assign an anomaly score for each record by computing the distance to its k^{th} nearest neighbor or they can define a record to be anomalous if the density of its neighborhood is low. To classify a new unseen record as normal or anomalous the distances or densities for each record in the training set or test set have to be reevaluated. Thus, the computational

complexity of nearest-neighbor based algorithms is a major drawback. Due to this, nearest-neighbor based algorithms were disregarded in this study [18].

The methods discussed so far are in general trying to model normal behavior in order to identify records which deviates from a normal profile. However, some methods referred to as isolation-based, e.g. *isolation forest*, are in contrary trying to isolate anomalies rather than explicitly model normal behavior [25]. The isolation forest algorithm proposed by Liu et al. [26] has the advantage of being designed for the purpose of identifying anomalies and has been shown to perform well in terms of computational complexity for large data sets. It has also been shown in previous studies, e.g. [10] and [26], that the algorithm performs very well when compared to other state-of-the-art anomaly detection algorithms such as the density-based *local outlier factor* algorithm. Although isolation forest may perform poorly when applied to high-dimensional data [12], it was not assessed to be an issue for this study since the final data set, containing 16 features, was not considered to be high dimensional. Because of the advantages with isolation forest and that the performance of the algorithm has been shown to be very competitive, it was selected as one of the models for evaluation.

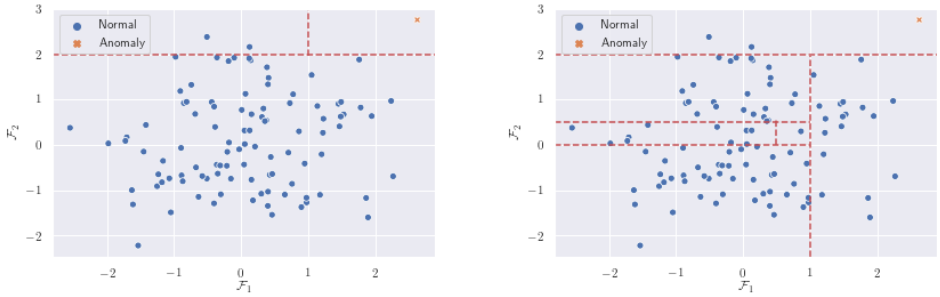
Another approach that recently has shown promising performance when applied to anomaly detection problems is to use deep learning. One such method is referred to as autoencoder. Although autoencoders have commonly been used for dimensionality reduction, they have also proved to perform well in an unsupervised anomaly detection setting, see e.g. [27], [28], [29]. Furthermore, previous studies have shown that they perform well even when the training data is contaminated [28].

5.2 Isolation forest

As mentioned in the overview in section 5.1, the isolation forest algorithm differs from many other models for anomaly detection in the sense that it is directly trying to identify anomalies rather than model normal behavior. The idea is that anomalies are easier to isolate compared to normal records since they are assumed to be both rare and different. As a consequence, fewer

partitions should be required to isolate an anomaly compared to a normal record. The path of the resulting binary tree, referred to as an *iTree* should therefore become shorter for abnormal records [25].

The idea of isolation forest is illustrated in figure 3 where 100 normal records have been sampled from a bivariate Gaussian distribution with zero mean and standard deviation equal to one and one anomaly has been sampled from a bivariate Gaussian distribution with mean equal to three and standard deviation equal to one. As can be seen in figure 3a the anomaly is isolated with only two partitions while isolating the normal record in 3b requires five partitions. Thus, isolating the anomaly requires fewer partitions compared to isolating a normal record.



(a) *Isolation of an anomaly.* **(b)** *Isolation of a normal record.*

Figure 3: *Illustration of isolation forest showing the idea that isolating an anomaly requires fewer random partitions than isolating a normal record.*

The algorithm can be divided into two stages: one *training stage* and one *evaluation stage*. During training a set of t iTrees are grown which is referred to as an *iForest*. To grow an iTree consider the following. First let $\mathcal{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)}\}$ be a training set where $\mathbf{x}^{(i)}$ is the i^{th} record and consists of $|\mathcal{F}|$ features. Furthermore, let $\mathcal{X}_S \subset \mathcal{X}$ be a set of sub-samples drawn from the training set. $\mathcal{X}_S$ is partitioned recursively by randomly selecting a feature, $\mathcal{F}_j \in \mathcal{F}$, and then performing a binary split at the value p which is randomly chosen such that $\min \mathcal{F}_j < p < \max \mathcal{F}_j$. The partitions are performed recursively until the remaining data only consists of one record, all the records in the remaining data have the same values or the height of

the resulting tree is equal to a given height limit [26].

During the training stage there are two parameters that have to be set. The sub-sampling size, denoted by ψ, defines the size of the subset drawn from the original data set when growing an iTree. According to Liu et al. [26] the sub-sampling of the data helps avoid two common problems in anomaly detection. The first problem is referred to as *swamping* and means that normal records are wrongly classified since they are too close to anomalies. The second problem referred to as *masking* means that if the anomalies are too many, they may form their own clusters and thus be difficult to isolate. Both of these problems are, according to Liu et al. [26], a result of using a too large data set. It is found in [26] that setting the sub-sampling size to 256 is a good starting point. The second parameter is how many trees the ensemble or the *iForest* should consist of, where $t = 100$ has shown to be a good value [26].

During the evaluation stage an anomaly score is estimated for each of the records $\mathbf{x}^{(i)}$, $i = 1, 2, ..., n$. Before introducing the formula for the anomaly score let's first introduce some new notations. Let $h(\mathbf{x}^{(i)})$ be the number of edges, starting from the root and ending at the terminating node for record $\mathbf{x}^{(i)}$. Furthermore, let $c(\psi)$ be an adjustment [3]. The formula for the anomaly score for a record $\mathbf{x}^{(i)}$ can with these notations in mind be defined as in (3)

$$s(\mathbf{x}^{(i)}, \psi) = 2^{-\frac{\mathbb{E}[h(\mathbf{x}^{(i)})]}{c(\psi)}} \tag{3}$$

where $\mathbb{E}[\cdot]$ is the expected value [26].

5.3 Autoencoder

The basic *autoencoder* can be described as a three-layered neural network and consists of an encoder and a decoder as illustrated in figure 4. Each layer is fully connected which means that each neuron or unit in each layer is connected with all the neurons in the next layer. Let $\mathbf{W} \in \mathbb{R}^{n \times |\mathcal{F}|}$ be

[3] For a more formal definition and discussion regarding $h(x)$ and $c(\psi)$, the reader is referred to [26].

a weight matrix and let $\mathbf{b}$ be a bias[4] vector. The encoder $h = f_\theta(\mathbf{X}) = \sigma(\mathbf{WX} + \mathbf{b})$ transforms the input data to a hidden unit while the decoder function $\hat{\mathbf{X}} = g_{\theta'}(h) = \sigma(\mathbf{W}'h + \mathbf{b}')$ aims to reconstruct the input $\mathbf{X}$ from the latent space [30].

Since an autoencoder aims to reconstruct a given input it can be used for unsupervised anomaly detection. The underlying assumption is that since anomalies are rare, the autoencoder should not be able to reconstruct these as well as normal records. Thus, the reconstruction errors can be used as anomaly scores. In this study the average reconstruction error defined as $\frac{1}{|\mathcal{F}|}||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$ was used as anomaly score for a record $\mathbf{x}^{(i)}$.

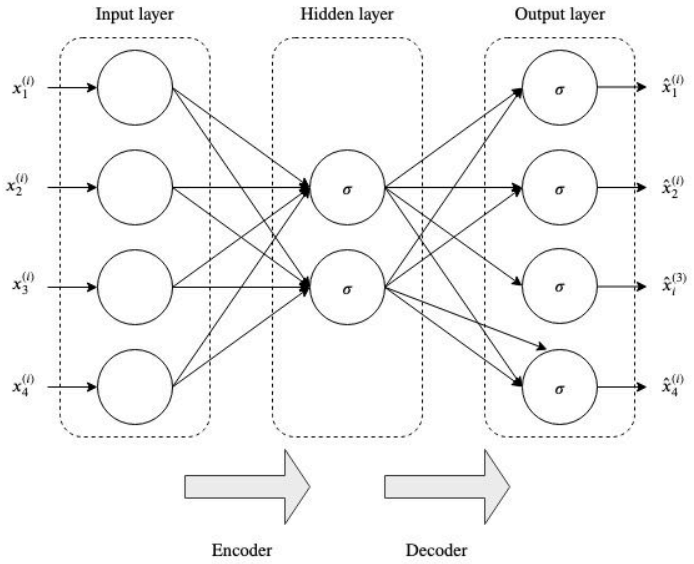


Figure 4: *An illustrative example of the structure of a three layered autoencoder with sigmoid activation functions.*

In order to learn an alternative representation and find the parameters $\theta = \{\mathbf{W}, \mathbf{b}\}$ and $\theta' = \{\mathbf{W}', \mathbf{b}'\}$ the reconstruction error is minimized in the form of a loss function as in (4) [30]. Two popular loss functions in the literature, which were considered in this study, are the squared error and the binary cross entropy. The loss function used to train the final models in this study was the

[4]Note that the bias is omitted in figure 4.

mean of the squared error for each record, i.e. $\mathcal{L}(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}) = \frac{1}{|\mathcal{F}|}||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$. This loss function can be interpreted as the average reconstruction error for each record. To optimize the expression in (4) a *stochastic gradient descent* algorithm is often used [30]. For this study the *Adam* optimization algorithm [31] was used to minimize the loss function.

$$\theta, \theta' = \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}) \tag{4}$$

Figure 4 illustrates a basic autoencoder, sometimes referred to as a *vanilla autoencoder*. However, several extensions have been used in the literature. One natural extension is to use a *deep autoencoder*[5] which means that more hidden layers are added to the encoder and the decoder. The purpose of adding more layers is that it can lead to a better feature representation [29]. Another extension, referred to as a *denoising autoencoder* [30], aims to learn the original input data from a corrupted version. These two extensions can be combined into a deep denoising autoencoder (DDAE) which has shown to perform well at detecting anomalies in event logs, even when the training data set is contaminated [28].

One way to convert a basic autoencoder into a denoising autoencoder is to randomly forcing a fraction of the neurons in the input layer to zero [30]. This approach is in [32] referred to as *masking noise*. Another approach is to add so called *salt-and-pepper noise* which means that a fraction of the units in the input layer is replaced by either the minimum or the maximum value with probability 0.5 respectively. Finally, the last approach, which after several experiments yielded the best performance in this study, is to use *additive Gaussian noise*, i.e. to add Gaussian noise directly to the input

[5]The terms *deep autoencoders* and *stacked autoencoders* are sometimes used interchangeably. However, stacked autoencoders are often referring to autoencoders which have been trained in a specific way. Therefore, to avoid confusion only the term deep autoencoder will be used in this thesis when referring to an autoencoder with more than one hidden layer, regardless of how it is trained.

data [32]. In order to define this approach more formally let $\epsilon^{(i)}$ be a vector of size $|\mathcal{F}|$ where each element is an independent and identical distributed random variable drawn from the normal distribution with zero mean and standard deviation equal to σ, i.e. $\epsilon_j^{(i)} \sim \mathcal{N}(0, \sigma^2)$. The corrupted input data used in this study can then be defined as $\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} + \epsilon^{(i)}$ for each record and the resulting loss function used during training can be defined as $\mathcal{L}(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}) = \frac{1}{|\mathcal{F}|}||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$.

In the illustration in figure 4 the sigmoid function, $\alpha_\sigma(x) = \sigma(x) = \frac{1}{1+e^{-x}}$, $\alpha_\sigma : \mathbb{R} \mapsto [0, 1]$, is used as activation function. However, there are more candidates that should be considered. Two other popular choices are the *tanh* function, $\alpha_t(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, where $\alpha_t : \mathbb{R} \mapsto [-1, 1]$ and the rectified linear unit (ReLU), $\alpha_r(x) = \max(x, 0)$, where $\alpha_r : \mathbb{R} \mapsto \mathbb{R}^+$ [33]. An advantage with ReLU is that it is a non-saturated activation function which means that it converges faster and that it can solve the vanishing gradient problem [34]. After experimentation it was found that ReLU outperformed both sigmoid and tanh when used as activation function in the hidden layers. An alternative to ReLU, called *Swish* has been proposed by Ramachandran et al. [35]. Swish is defined as $\alpha_s(x) = x\sigma(\beta x)$, where σ is the sigmoid function and β is a constant. According to the authors this activation function outperformed ReLU even though it was used in networks that were optimized for ReLU [35]. Thus, it is a quite simple procedure to try both activation functions and therefore both architectures were considered as candidates for the final model.

After experimenting with both of the normalization techniques, introduced in section 4.4, min-max normalization was chosen, simply because it consistently yielded higher performance. Since the input data was scaled using min-max transformation and thus mapped to the interval $[0, 1]$[6], a sigmoid function was initially thought to be an appropriate activation function for the output layer. However, after experiments with different architectures a linear activation function, $\alpha_l(x) = x$, was used for the output layer as well

[6]Note that since the scaling was estimated for the training set only, it is possible for the validation data to be outside the interval [0,1] if there exist a record $\mathbf{x}^{(k)}$ for which $x_j^{(k)} < \min_{1 \leq i \leq n} x_j^{(i)} \vee x_j^{(k)} > \max_{1 \leq i \leq n} x_j^{(i)}, \ x^{(k)} \notin \mathcal{X}$.

as for the bottleneck layer[7].

One approach that can be used to allow for higher learning rates during training and for making the network less sensitive to the choice of initialization is to use *batch normalization* as suggested by Ioffe & Szegedy [36]. During experiments with different structures of the neural network it was observed that by adding batch normalization the performance of the network increased.

5.4 Scoring approach

Both isolation forest and autoencoders can be characterized as scoring algorithms, i.e. neither of them are directly categorizing if a record is an anomaly but rather assigns a degree of deviating behavior. Let's denote the anomaly score for the i^{th} record by $\phi_i = \phi(\mathbf{x}^{(i)})$ where ϕ is a scoring function. As previously mentioned the scoring function ϕ depends on the anomaly algorithm. For isolation forest the scoring function was defined as $\phi(\mathbf{x}^{(i)}) = 2^{-\frac{\mathbb{E}[h(\mathbf{x}^{(i)})]}{c(\psi)}}$ and for the autoencoders the scoring function was defined as the average reconstruction error $\phi(\mathbf{x}^{(i)}) = \frac{1}{|\mathcal{F}|}||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$ [8].

To classify a record as an anomaly one has to first decide on a threshold t for which all records in $\{\mathbf{x}^{(i)} : \phi(\mathbf{x}^{(i)}) > t\}$ should be classified as anomalies. However, this approach only assigns an anomaly score to an individual record, i.e. to a specific order execution and time window. This might be of interest when looking for point anomalies which occur within the same time window but it may not be the best approach when identifying if an order execution as a whole should be considered to be normal or abnormal. One intuitive and simple approach, which was used in this study, is to aggregate the anomaly scores for each order execution. Four simple aggregation functions are the sum, the maximum, the median and the mean. After experimentation the mean of the anomaly scores for each order execution seemed to give the best

[7]The *bottleneck layer* is referring to the layer with the lowest dimensionality within the network.

[8]It should be noted that the anomaly scoring function for the DDAE used the original data as input and not the corrupted version. The corrupted version was only used during training.

performance. Let Φ be the aggregated scoring function. The anomaly score for each order execution, $\mathcal{E}_k$ can then be defined as in (5).

$$\Phi(\mathcal{E}_k) = \frac{1}{|\mathcal{E}_k|} \sum_{\mathbf{x}^{(i)} \in \mathcal{E}_k} \phi(\mathbf{x}^{(i)}) \tag{5}$$

One advantage with this approach is that the scoring function Φ is not limited to scoring order executions but can also be used, as in (6), when it is of interest to find certain time windows $\mathcal{W}_i$ that are abnormal.

$$\Phi(\mathcal{W}_k) = \frac{1}{|\mathcal{W}_k|} \sum_{\mathbf{x}^{(i)} \in \mathcal{W}_k} \phi(\mathbf{x}^{(i)}) \tag{6}$$

6 Evaluation

In order to evaluate the models some kind of performance measure is required. For supervised methods this problem is quite straightforward since labels are available, although the class imbalance can be a challenge in itself. However, for the unsupervised setting the problem becomes more complicated and challenging [12]. One solution is to generate artificial anomalies and perform the evaluation based on these. Such approaches are discussed in section 6.1 followed by a discussion of relevant metrics that can be used to evaluate and compare different models in section 6.2. The section ends with a description of the experimental design.

6.1 Generation of artificial anomalies

A challenge with unsupervised learning is how the performance of different models should be evaluated and compared to each other when the ground truth is missing. One intuitive and straightforward approach is to sample anomalous records uniformly across the domain space spanned by the features [37]. Another approach used in [38] is to add random noise to existing records and label these as anomalous. A distribution-based artificial anomaly generation is proposed by Fan et al. [37] where the idea is to generate anomalous records which are close to actual data. These types of methods have not been used in this study since it is difficult to evaluate how realistic and relevant the generated anomalies are.

Another common approach to evaluate the performance of anomaly detection algorithms when the ground truth is unavailable is to use a data set with different known classes. This approach is known as a downsampling strategy. The idea is to select a small subset of records from some of the classes and treat these as anomalies while using the remaining classes as normal data [39]. This approach was applicable in this study since data was available for several execution strategies. Let $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{N_s}\}$ be a set consisting of available strategies where $\mathcal{S}_i$ is a set containing all records belonging to the i^{th} execution strategy. The downsampling strategy could be

applied so that samples are drawn from $\mathcal{A}_S = \mathcal{S} \backslash \mathcal{S}_I$ where $\mathcal{S}_I$ is the strategy of interest. This set of samples could then act as artificial anomalies in order to evaluate how good an anomaly detection algorithm is when applied to strategy S_I. One problem with this approach is that the behavior of other strategies might be very different from the behavior of the anomalies which the model should be able to detect. This problem is illustrated in figure 5 where the orange records represent injected anomalies, while the two records at point $(1.0, 2.5)$ and $(2.5, 1.5)$ could be considered as real anomalies. In this scenario, a model that uses the red dashed line as a decision boundary can easily detect the injected artificial anomalies but will not be able to identify real anomalies since the nature of these is dramatically different. To make this problem less severe, records from other strategies have, to the extent it was possible, only been included if they shared the same order characteristics with an order execution $\mathcal{E}_k \subset \mathcal{S}_I$ belonging to the strategy of interest. The idea is that the difference between records in $\mathcal{S}_I$ and $\mathcal{A}_S$ should be limited to how each order was executed.

Figure 5: *An illustrative example that shows the potential drawbacks when using records from other strategies as artificial anomalies. Both strategy A and strategy B have been sampled from a bivariate Gaussian distribution with standard deviation equal to 0.5 but with a mean equal to zero and five respectively. The red dashed line illustrates a hypothetical decision boundary.*

In addition to the downsampling technique described above, a small set

of artificial anomalies have been manually constructed after consulting with domain experts. The advantage with this approach compared to the downsampling technique is that by manually constructing anomalies the nature of these might be more similar to the nature of the anomalies that are of interest. These manually constructed anomalies were created by manipulating real data to make sure that they were as realistic as possible. However, it might still be argued that anomalies constructed by hand may be too extreme or unrealistic and thus easier to detect.

An evaluation based on the artificial anomalies generated by a downsampling strategy and by construction will give a lower threshold of an acceptable model since an anomaly detection algorithm should *at least* be able to detect such anomalies. However, despite the precautions taken, the evaluation of a model based on the injected synthetic anomalies should be carefully interpreted.

In conclusion two types of anomalies were injected into the data set, one based on the downsampling technique denoted by $\mathcal{A}_S$ and one based on manual construction denoted by $\mathcal{A}_C$. Because the frequency of anomalies should be very low it was not appropriate to use all available observations from other strategies since this would make the data set unrealistically balanced and thus overestimating the performance. Therefore, 100 different subsets, $\mathcal{A}_{S_i}$ were sampled from the set $\mathcal{A}_S$ such that the following conditions held:

(i) $\bigcup_{i=1}^{100} \mathcal{A}_{S_i} \subset \mathcal{A}_S$

(ii) $|\mathcal{A}_{S_i}| \ll |\mathcal{A}_S| \quad i = 1, 2, ..., 100$

(iii) $|\mathcal{A}_{S_i}| = |\mathcal{A}_{S_j}| \quad i, j = 1, 2, ..., 100.$

Because the manually constructed anomalies were few in numbers all of these were combined with each subset $\mathcal{A}_{S_i}$. Let's denote the collection of these combined sets by $\mathcal{A}_A = \{\mathcal{A}_{A_1}, \mathcal{A}_{A_2}, ..., \mathcal{A}_{A_{100}}\}$ where $\mathcal{A}_{A_i} = \mathcal{A}_{S_i} \bigcup \mathcal{A}_C$ for $i = 1, 2, ..., 100$. The final ratio between the number of records in the original data set and the number of artificial anomalies was about 0.2%, making the data set extremely imbalanced.

6.2 Evaluation metrics

Before introducing the metrics that have been used for evaluation the following notations need to be introduced. Let $\mathcal{A}$ be the set of all true anomalies and let $\mathcal{N}$ be the set of all normal records in the data set. Also, let $\phi_i = \phi(\mathbf{x}^{(i)})$ be the estimated anomaly score for the i^{th} record and let $t \in [\min_{1 \leq i \leq n} \phi_i, \max_{1 \leq i \leq n} \phi_i]$ be a threshold. Furthermore, let $A(t)$ and $N(t)$ be the set of predicted anomalies and normal records at the threshold t respectively, i.e. $A(t) = \{\mathbf{x}^{(i)} : \phi(\mathbf{x}^{(i)}) > t\}$ and $N(t) = \{\mathbf{x}^{(i)} : \phi(\mathbf{x}^{(i)}) \leq t\}$.

Accuracy, as defined in (7), is an evaluation metric which is often used in the machine learning literature. This metric may be tempting to use since it is measuring the proportion of the predictions that are correct. However, in anomaly detection problems accuracy is not appropriate since anomalies are often extremely rare. Thus, a model that predicts that all the records are normal will have a very high accuracy even though it would be worthless at classifying abnormal behavior. Instead the two metrics precision and recall as defined in (8) and (9) respectively are often used [12].

$$Accuracy(t) = \frac{|\mathcal{A} \cap A(t)| + |\mathcal{N} \cap N(t)|}{|\mathcal{A}| + |\mathcal{N}|} \tag{7}$$

$$P(t) = \frac{|A(t) \cap \mathcal{A}|}{|A(t)|} \tag{8}$$

$$R(t) = \frac{|A(t) \cap \mathcal{A}|}{|\mathcal{A}|} \tag{9}$$

Precision can be interpreted as the proportion of the predicted anomalies that is classified correctly. Recall can be interpreted as the proportion of correct classified anomalies compared to the total number of abnormal records in the data set. By varying the threshold at which a record is considered to be an anomaly a *precision-recall curve* (PR curve) can be estimated [12].

Another common metric to use in anomaly detection is the so called *receiver operating characteristic* (ROC) curve [12]. The ROC curve is obtained by calculating the true positive rate versus the false positive rate, as defined in (10) and (11) respectively, for each threshold t. According to Campos et

al. [39] this measure is adjusting for the class imbalance since the false positive rate and true positive rate are normalized. Another popular measure that can be derived from the ROC curve is the ROC AUC which is defined as the area under the ROC curve and ranges from zero to one [39]. Similarly the area under the curve can also be calculated for the precision-recall curve (PR AUC) which similarly to ROC AUC can be used to express the performance of an algorithm as a number between zero and one [40].

$$TPR = \frac{|A(t) \bigcap \mathcal{A}|}{|\mathcal{A}|} \tag{10}$$

$$FPR = \frac{|A(t) \bigcap \mathcal{A}^c|}{|\mathcal{N}|} \tag{11}$$

When deciding between ROC and PR as evaluation measures it is important to take the class imbalance into consideration. ROC may not be appropriate for data sets for which the class imbalance problem is severe since the performance of the algorithms may be overestimated. Furthermore, the performance of anomaly detectors that seem to be comparable when looking at the ROC curves may be vastly different when comparing the PR curves [40].

Since the ground truth was not available for the data set, the evaluation was performed after injecting the synthetic anomalies. Thus, the evaluation metrics in this thesis are only estimates of the real metrics and should not in any way be interpreted as the true measures. To see this, consider that since the ground truth is not available it is not possible to know $\mathcal{A}$ and $\mathcal{N}$. However, the labels for the synthetic anomalies $\mathcal{A}_{A_i} \subset \mathcal{A}$ are known. Thus, if precision is estimated based on $\mathcal{A}_{A_i}$ it will only be equal to the true value if there are no anomalies in the original data set.

6.3 Experimental design

The experimental design in this study is illustrated in figure 6. Most of the different stages, represented by blocks, have already been discussed and should be familiar to the reader. However, the overall approach used to evaluate the models deserves a more detailed presentation. First, the whole

data set was split up into three parts: a training set $\mathcal{X}$, a validation set $\mathcal{V}_N$ and a test set $\mathcal{T}_N$, all without artificial anomalies. The models were trained only on the training set which did not include any of the synthetic anomalies, i.e. $\mathcal{X} \cap \mathcal{A}_A = \varnothing$. The reason for not injecting artificial anomalies into the training set was to avoid adding bias to the data. The risk with doing so would be that the resulting model would be outstanding at separating injected strategies from the true strategy, as illustrated in figure 5, but worthless at detecting any other anomalies that are of interest. Furthermore, to avoid information leakage between the training, validation and test sets the parameters for normalization were estimated using the training set only.

Each model was then evaluated using the validation set $\mathcal{V}_N$ which only contained records from the same strategy as the training set. The purpose of this was to see if the model suffered from overfitting or underfitting. The idea was that the distribution of the anomaly scores should be very similar for the training set $\mathcal{X}$ and the validation set $\mathcal{V}_N$.

In order to evaluate the models' ability to detect anomalies each of the sets $\mathcal{A}_{A_i} \in \mathcal{A}_A$ was injected into the validation set one at a time, resulting in 100 different sets for evaluation. Let's denote these sets by $\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_{100}$, where $\mathcal{V}_i = \mathcal{V}_N \cup \mathcal{A}_{A_i}$. The different models were then compared using the evaluation metrics discussed in section 6.2.

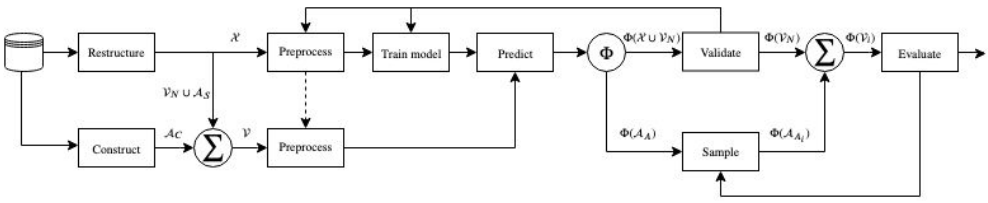


Figure 6: *Illustration of the experimental design used in the study, from the retrieval of data to the final evaluation.*

The model with highest performance was applied to a new unseen data set $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_{100}\}$, where $\mathcal{T}_i = \mathcal{T}_N \cup \mathcal{A}_{A_{100+i}}$ and $(\bigcup_{i=1}^{100} \mathcal{A}_{A_{100+i}}) \bigcap (\bigcup_{i=1}^{100} \mathcal{A}_{A_i}) = \varnothing$. The purpose of this was to analyze the performance of the models on a data set which had not in any way influenced the design choice of the models or

feature engineering. The test set, $\mathcal{T}$, consisted, similarly to the validation set, of 100 subsets of order executions from the strategy of interest and artificial anomalies.

7 Results and analysis

This section presents the results for isolation forest and deep denoising autoencoder (DDAE). Two different architectures are presented for the DDAE, one with ReLU and one with Swish activation functions. The final architectures for the models are described in section 7.1. The outcome of the validation and evaluation of the models is presented in section 7.2. All models presented in this section are models that showed the best performance for each architecture.

7.1 Model architecture

7.1.1 Isolation forest

Only two parameters were necessary to tune for isolation forest. The number of trees was initially chosen to 100, as suggested by Liu et al. [26]. It was noted that increasing the number of trees led to larger computational complexity while only having a small effect on the performance. The final number of trees was set to 200. For the sub-sampling size it was found that a considerable larger size than 256 yielded a notable increase in performance. The final sub-sampling size was set to $10,000$.

7.1.2 Deep denoising autoencoder

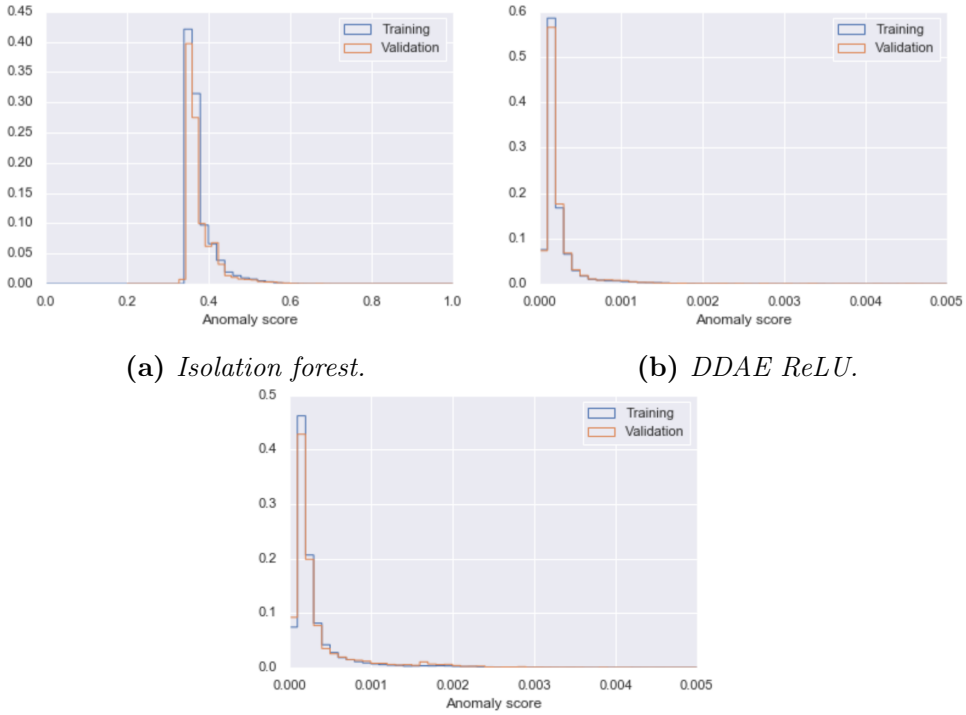
Two different deep denoising autoencoders (DDAEs) were considered for the final evaluation. The only difference between them was that one used ReLU and the other used Swish as activation functions for the hidden layers. To make notations easier, the deep denoising auto encoder with ReLU activation functions and the deep denoising autoencoder with Swish activation functions are referred to as DDAE ReLU and DDAE Swish respectively. Both autoencoders consisted of five hidden layers, including one bottleneck layer. The hidden layers had twelve, eight, four, eight and twelve neurons each. For the additive Gaussian noise the mean was set to zero and the standard deviation was set to 0.01. ReLU or Swish activation functions were used for all layers

except for the bottleneck and output layers for which linear activation functions were used. Batch normalization was implemented between all layers except for between the bottleneck layer and the first decoder layer. Following Ioffe & Szegedy [36] the batch normalization was applied before each non-linear activation function. The average reconstruction error was used as loss function and the parameters were optimized using the Adam optimizer proposed by Kingma & Ba [31]. The learning rate was initially set to 0.01, but was decreased by a factor of two if the loss function did not decrease during five consecutive epochs. Although 100 epochs were used to train the models, the training was stopped if the loss function did not decrease with at least 10^{-5} during ten consecutive epochs. The batch size was set to 32. To be able to monitor if the model started to overfit, 20% of the training data was held out during training and used to compare the training loss with the validation loss.

7.2 Performance

7.2.1 Anomaly score distribution

Figure 7 shows the anomaly score distributions for the training set $\mathcal{X}$ and the validation set without injected anomalies $\mathcal{V}_N$. The idea is that the distributions should be similar for both sets, otherwise the model might suffer from overfitting or underfitting. The distributions for isolation forest and the two DDAEs seem to be quite similar and therefore neither overfitting nor underfitting seems to be a problem for any of the models. It can be noted that the magnitude of the anomaly scores is larger for isolation forest compared to the DDAE models. However, this difference can be explained by the fact that isolation forest uses a different scoring function ϕ compared to the two DDAE models which both uses the average reconstruction error.



(a) *Isolation forest.*

(b) *DDAE ReLU.*

(c) *DDAE Swish.*

Figure 7: *The anomaly score distributions for the training set $\mathcal{X}$ and the validation set $\mathcal{V}_N$.*

7.2.2 Receiver operating characteristics

Figure 8 shows the ROC curves for the three models with an additional curve for a random classifier. All three models perform considerably better than the random classifier which can be assumed to be a minimum requirement for an anomaly detection algorithm. Both the DDAE models have considerable higher true positive rate compared to isolation forest for almost every threshold t. The only exception is at one point where the ROC curves for DDAE Swish and isolation forest overlap. However, the ROC curve for isolation forest is never above. It should be noted that the curves in figure 8 only represent one of the validation subsets, $\mathcal{V}_i$, and thus the figure does not give the whole picture. Therefore statistics for the ROC AUC values are given in table 3. The ROC AUC values confirm that both the DDAE models

outperform isolation forest on average. Furthermore, DDAE Swish performs slightly better than DDAE ReLU with an average ROC AUC value of 0.9332 compared to 0.9143.

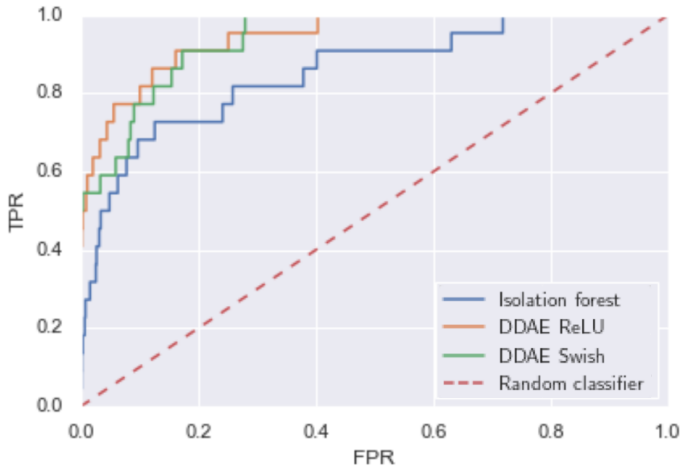


Figure 8: *ROC curves for isolation forest, two deep denoising autoencoders and a random classifier.*

Table 3: *Statistics for area under the receiver operating characteristics curves for isolation forest and two deep denoising autoencoders with ReLU and Swish activation functions.*

	Mean	Median	Min	Max	STD
Isolation forest	0.8428	0.8464	0.7837	0.8740	0.0199
DDAE ReLU	0.9143	0.9209	0.8638	0.9535	0.0240
DDAE Swish	**0.9332**	**0.9336**	**0.8990**	**0.9576**	0.0144

7.2.3 Precision-recall

As discussed in section 6.2, ROC may not be appropriate when evaluating the performance of anomaly detection models when the data set is very imbalanced. Therefore, this section evaluates the models based on precision-recall curves (PR curves) and the area under the precision recall curves (PR AUC). The PR curves for one of the validation subsets, $\mathcal{V}_i$, are presented in figure

9a. The graph clearly shows that the two DDAE models are superior to isolation forest for recall values less than 0.5. Both DDAE models can detect around 20% of the injected artificial anomalies with 100% precision. Thus, the DDAE models can detect 20% of the synthetic anomalies without making a single mistake. This can be compared to isolation forest which at most has a precision around 20% at a recall value less than 20%.

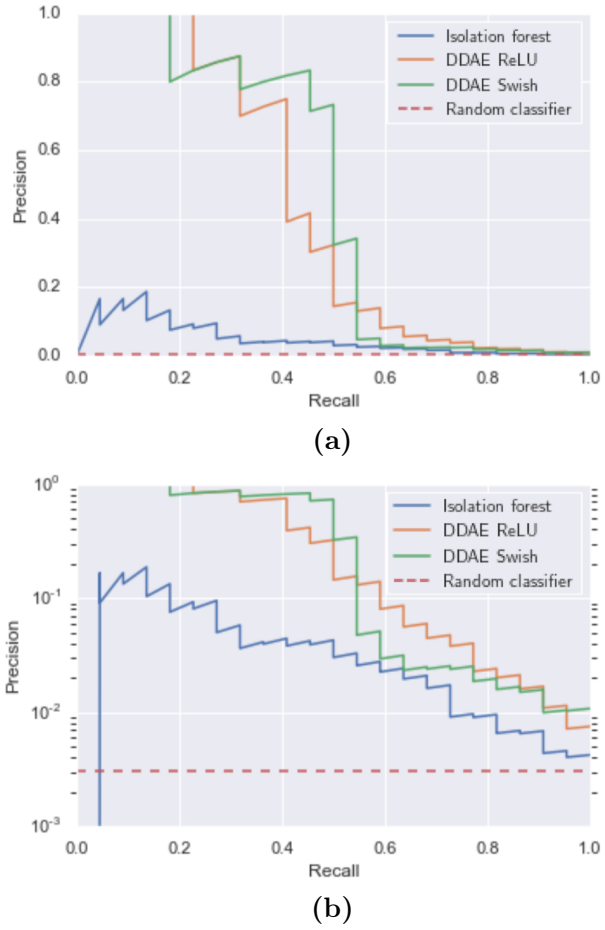


Figure 9: *Precision-recall curves for isolation forest, two deep denoising autoencoders and a random classifier. Figure 9b is identical to figure 9a except that a log scale is used for precision in figure 9b.*

To make it easier to compare the difference for higher recall values the PR curves are plotted for the methods in figure 9b where a logarithmic scale is

used for the y-axis. It can be noted that all the models outperform a random classifier, especially for lower recall levels. Although the difference between isolation forest and the DDAE models becomes smaller when recall increases, it is apparent that the DDAE models are better suited for all recall values for this validation subset.

As with the evaluation based on ROC, the curves in figure 9 are only valid for one of the validation subsets. Thus, to be able to draw any conclusion regarding the difference in performance on average, the PR AUC values are presented in table 4. These values confirm that DDAE is better suited for detecting artificial anomalies. The DDAE models have PR AUC values that, on average, are more than eight times as high compared to isolation forest. For comparison it can also be noted that a random classifier has an average PR AUC of about 0.002 and thus the PR AUC values for isolation forest and the DDAE models are about 25 respectively 200 times higher. Finally, it can be concluded that DDAE Swish performs slightly better than DDAE ReLU with an average PR AUC value of 0.4380 compared to 0.4246.

Table 4: *Statistics for area under the precision-recall curves for isolation forest and two deep denoising autoencoders with ReLU and Swish activation functions.*

	Mean	Median	Min	Max	STD
Isolation forest	0.0520	0.0431	0.0253	0.1242	0.0245
DDAE ReLU	0.4246	0.4250	0.3724	0.5010	0.0328
DDAE Swish	**0.4380**	**0.4369**	**0.3919**	**0.5105**	0.0319

7.2.4 Different anomaly types

The nature of the two types of artificial anomalies, $\mathcal{A}_S$ and $\mathcal{A}_C$, might be rather different. It is therefore interesting to see how each model performs when only applied to a data set with one type of anomalies. From the results presented in table 5 it can be concluded that the constructed anomalies are considerably easier to detect for the two DDAE models, both in terms of average ROC AUC and PR AUC. In contrast, isolation forest seems to be better at detecting artificial anomalies drawn from $\mathcal{A}_S$ compared to manually constructed anomalies. An interesting observation is that the dominance of

the DDAE models can be explained by their impressive ability to detect constructed anomalies. Although the difference in performance measured by PR AUC is much smaller when only considering anomalies drawn from $\mathcal{A}_S$, the DDAEs still perform better compared to isolation forest. Isolation forest reaches a PR AUC of 0.1031 on average compared to 0.1462 for DDAE ReLU.

Table 5: *Average ROC AUC and PR AUC for isolation forest and two deep denoising autoencoders grouped by the type of artificial anomalies.*

	$\mathcal{A}_S$		$\mathcal{A}_C$	
	ROC AUC	PR AUC	ROC AUC	PR AUC
Isolation forest	0.8650	0.1031	0.8341	0.0275
DDAE ReLU	0.8396	**0.1462**	0.9451	0.5051
DDAE Swish	**0.8643**	0.1160	**0.9595**	**0.5329**

7.2.5 Final evaluation

So far, it has been shown that both DDAE models perform better compared to isolation forest. However, it is difficult to say which of the DDAE models is the best since they seem to be better at detecting different types of anomalies. Therefore, both of the architectures were applied to the final test set in order to confirm the performance of the models.

Table 6: *ROC AUC and PR AUC for the two deep denoising autoencoders applied to unseen test data. The metrics are given for the test set with both types of artificial anomalies, $\mathcal{A}_A$ and for a test set with synthetic anomalies sampled from $\mathcal{A}_S$ only.*

	$\mathcal{A}_A$		$\mathcal{A}_S$	
	ROC AUC	PR AUC	ROC AUC	PR AUC
DDAE ReLU	0.8956	0.2864	0.8579	**0.1277**
DDAE Swish	**0.9064**	**0.3424**	**0.8781**	0.0963

In table 6 it is shown that both the ROC AUC and PR AUC values are lower for the test set compared to the validation set. However, the PR AUC values of 0.3424 for DDAE Swish and 0.2864 for DDAE ReLU in table 6 are still considerably higher than the PR AUC value for isolation forest in table 4. Comparing the models' ability to detect anomalies drawn from

$\mathcal{A}_C$ may however not be accurate since it is difficult to guarantee that the nature of the constructed anomalies is the same. Thus, it is difficult to say if the difference is due to the model or due to the construction of anomalies. Therefore, to provide a better comparison, ROC AUC and PR AUC are also given in table 5 for anomalies drawn only from $\mathcal{A}_S$. DDAE Swish performs slightly better with respect to ROC AUC and DDAE ReLU performs slightly better with respect to PR AUC. The PR AUC values have decreased with about the same amount, 0.02, for both models compared to the validation set. It should be noted that the fraction of artificial anomalies was slightly lower for the test set since it was impossible to get the same imbalance. This might have contributed to lower PR AUC values for the test set. Despite not performing as well for the test set, the DDAE models still outperform isolation forest when all artificial anomalies are injected and show promising ability to detect deviating behavior for algorithmic trading.

7.3 Reflections

First of all it can be concluded that all of the models considered in this study have shown to perform considerably better than a random classifier. Hence, in this regard, all the models can be claimed to be acceptable. However, the DDAE models showed superior ability at detecting the constructed anomalies, were slightly better at detecting artificial anomalies drawn from $\mathcal{A}_S$ and showed to perform similar when applied to the test sets as when applied to the validation sets. The results in this thesis therefore suggest that isolation forest should be disregarded in favor of DDAEs for the purpose of monitoring algorithmic trading. The choice between the two DDAE models considered is not obvious since they have been shown to be good at detecting different types of artificial anomalies. However, the difference between the models is small and further work has to be performed in order to confirm the difference between them.

The magnitude of the difference in performance between DDAE and isolation forest was surprising since isolation forest has shown to outperform other state-of-the-art models, see e.g. [25], [10]. Also, Domingues et al. [10]

show that isolation forest perform well both for large data sets and for data sets that contain noise. However, one explanation may be that the algorithm only splits the data set with respect to one feature at a time which may increase the false positives for complex data sets [41]. Thus, the complexity of the data set used in this thesis may not be well suited for isolation forest.

An interesting observation in this study was that the sub-sampling size for isolation forest had to be considerable larger compared to what is suggested by Liu et al [26] in order to achieve acceptable performance. In line with the findings in this study, Karev et al. [42] also finds that a considerable larger sub-sampling size of $8,192$ is required for the AUC to converge. Thus, as suggested in [42], more work on what affects the optimal size of the sub-samples has to be performed.

Since the evaluation of the models is based on artificial anomalies, it is not certain that the results presented would hold for real anomalies. In contrary, the performance would probably decrease for all models since real data is often more challenging. Although the results in this thesis should be carefully interpreted, they provide a guideline when choosing an anomaly detection model for monitoring algorithmic trading.

8 Conclusions and future research

8.1 Conclusions

The purpose of this thesis was to investigate if machine learning can be used to monitor algorithmic trading by detecting deviating behavior. Two models were chosen, isolation forest and deep denoising autoencoder. Due to lack of labels, artificial anomalies were generated, using two different approaches: a downsampling strategy and manual construction and manipulation of real data. The results show that a deep denoising autoencoder performs considerably better compared to isolation forest, achieving eight times higher PR AUC score on average for the validation set. Thus, a deep denoising autoencoder, either with ReLU or Swish activation functions, has been shown to be a good candidate for algorithmic trading surveillance. It should be noted that the DDAE models' ability to detect artificial anomalies were considerably higher for the constructed anomalies compared to the anomalies that was drawn from other strategies. Therefore, more work needs to focus on the feature engineering and the anomaly score aggregation in order to increase the detection rate for all types of artificial anomalies.

An important limitation of this study is that the evaluation was not based on ground truth anomalies. Even though generating artificial anomalies was considered to be the best approach given the absence of labels, the results should be carefully interpreted. However, since the DDAE showed promising performance, its ability to detect real anomalies should be investigated further.

8.2 Future research

A disadvantage with the approach in this study is that the temporal information that order executions contain is lost when using the time window approach and vectorizing the data. It would therefore be interesting to apply anomaly detection techniques that can use sequential data as input. One natural alternative to the deep denoising autoencoder used in this thesis would be to implement a long short-term memory (LSTM) autoencoder which al-

lows the input data to be of varying length.

Only two models, isolation forest and deep denoising autoencoder, have been evaluated on the data set. It would therefore be desirable to apply other state-of-the-art models in order to see if a deep denoising autoencoder outperforms those models as well.

There are a few parameters that have not been explored that would be interesting to analyze more thoroughly. The size of the time window is one such parameter. Furthermore, since the time windows used in this study do not overlap, it would be interesting to see if overlapping sliding windows can increase the performance. Also, the aggregated scoring functions considered in this study are quite simple. Therefore, it would be interesting to see if other types of aggregations can increase the performance.

Finally, only one type of execution algorithm for one type of financial asset has been considered in this study. It would therefore be desirable to investigate if the approach used in this study can be extended to other execution strategies and financial assets.

References

[1] G. Nuti, M. Mirghaemi, P. Treleaven, and C. Yingsaeree, "Algorithmic trading," *Computer*, vol. 44, no. 11, pp. 61–69, 2011.

[2] I. Aldridge, *High-frequency trading: A practical guide to algorithmic strategies and trading systems*. New York: John Wiley & Sons, Incorporated, 2 ed., 2013.

[3] K. Kim, *Electronic and algorithmic trading technology: The complete guide*. San Diego: Elsevier Science & Technology, 2007.

[4] M. O'Hara, "High frequency market microstructure," *Journal of Financial Economics*, vol. 116, no. 2, pp. 257–270, 2015.

[5] A. Madhavan, "Exchange-traded funds, market structure, and the flash crash," *Financial Analysts Journal*, vol. 68, no. 4, pp. 20–35, 2012.

[6] C. Borch, "High-frequency trading, algorithmic finance and the flash crash: Reflections on eventalization," *Economy and Society*, vol. 45, no. 3-4, pp. 350–378, 2016.

[7] W. L. Currie and J. J. M. Seddon, "The regulatory, technology and market 'dark arts trilogy' of high frequency trading: A research agenda," *Journal of Information Technology*, vol. 32, no. 2, pp. 111–126, 2017.

[8] S. Y. Yang, Q. Qiao, P. A. Beling, W. T. Scherer, and A. A. Kirilenko, "Gaussian process-based algorithmic trading strategy identification," *Quantitative Finance*, vol. 15, no. 10, pp. 1683–1703, 2015.

[9] R. L. Hayes, P. A. Beling, and W. T. Scherer, "Action-based feature representation for reverse engineering trading strategies," *Environment Systems and Decisions*, vol. 33, no. 3, pp. 413–426, 2013.

[10] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406–421, 2018.

[11] M. Ahmed, A. N. Mahmood, and M. R. Islam, "A survey of anomaly detection techniques in financial domain," *Future Generation Computer Systems*, vol. 55, pp. 278–288, 2016.

[12] C. C. Aggarwal, *Outlier Analysis*. New York: Springer, 2013.

[13] R. Cesari, M. Marzo, and P. Zagaglia, "Effective trade execution," in *Portfolio theory and management* (H. K. Baker and G. Filbeck, eds.), New York: Oxford University Press, 2013.

[14] P. Treleaven, M. Galas, and V. Lalchand, "Algorithmic trading review," *Communications of the ACM*, vol. 56, no. 11, pp. 76–85, 2013.

[15] J. Hasbrouck and G. Saar, "Low-latency trading," *Journal of Financial Markets*, vol. 16, no. 4, pp. 646–679, 2013.

[16] R. Cont and A. Kukanov, "Optimal order placement in limit order markets," *Quantitative Finance*, vol. 17, no. 1, pp. 21–39, 2017.

[17] B. Ende, P. Gomber, and M. Lutat, "Smart order routing technology in the new European equity trading landscape," *IFIP Advances in Information and Communication Technology*, vol. 305, pp. 197–209, 2009.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[19] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS One*, vol. 11, no. 4, p. e0152173, 2016.

[20] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: Data mining, inference and prediction*. New York: Springer, 2 ed., 2009.

[21] A. Leontjeva and I. Kuzovkin, "Combining static and dynamic features for multivariate sequence classification," in *2016 IEEE International*

Conference on Data Science and Advanced Analytics (DSAA), pp. 21–30, 2016.

[22] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM SIGKDD Explorations Newsletter*, vol. 12, no. 1, pp. 40–48, 2010.

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 823–839, 2012.

[24] J. Han, M. Kamber, and J. Pei, *Data mining: Concepts and techniques*. Saint Louis: Elsevier Science & Technology, 3 ed., 2011.

[25] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, pp. 1–39, 2012.

[26] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[27] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, pp. 4:4–4:11, 2014.

[28] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser", "Analyzing business process anomalies using autoencoders," *Machine Learning*, vol. 107, no. 11, pp. 1875–1893, 2018.

[29] L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang, "Anomaly-based insider threat detection using deep autoencoders," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 39–48, 2018.

[30] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1096–1103, 2008.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[34] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[35] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[37] W. Fan, M. Miller, S. Stolfo, W. Lee, and P. Chan, "Using artificial anomalies to detect unknown and known network intrusions," *Knowledge and Information Systems*, vol. 6, no. 5, pp. 507–527, 2004.

[38] E. Habler and A. Shabtai, "Using LSTM encoder-decoder algorithm for detecting anomalous ADS-B messages," *arXiv preprint arXiv:1711.10192*, 2017.

[39] G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 891 – 927, 2016.

[40] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 233–240, 2006.

[41] S. Hariri, M. C. Kind, and R. J. Brunner, "Extended isolation forest," *arXiv preprint arXiv:1811.02141*, 2018.

[42] D. Karev, C. McCubbin, and R. Vaulin, "Cyber threat hunting through the use of an isolation forest," in *Proceedings of the 18th International Conference on Computer Systems and Technologies*, CompSysTech'17, pp. 163–170, 2017.