Detecting Financial Fraud and Crimes in Capital Markets: a Study of Data-driven and Computational Approaches

POUYAN DINARVAND

PhD 2020

Detecting Financial Fraud and Crimes in Capital Markets: a Study of Data-driven and Computational Approaches

POUYAN DINARVAND

A thesis submitted in partial fulfilment of the requirements of

Manchester Metropolitan University

for the degree of Doctor of Philosophy

Department of Computing and Mathematics

Manchester Metropolitan University

2020

Declaration of Authorship

I, POUYAN DINARVAND, declare that this thesis titled, "Detecting Financial Fraud and Crimes in Capital Markets: a Study of Data-driven and Computational Approaches" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"Everything Should Be Made as Simple as Possible, But Not Simpler."

Albert Einstein

Abstract

Automatic surveillance of abnormal trading behaviours/patterns (ATPs) in capital markets is essential to protect the capital of legitimate traders from price distortion of finance assets. Detection of ATPs involves the finding of single (one trading order with large trading volume and long cancellation time, e.g. several minutes) or sequential (correlated multiple trading orders with small volume and short cancellation time, e.g. milliseconds) anomalies in trading data. However, accurate and timely identification of ATPs remains an open challenge due to high volume and high frequency data as well as unlabelled data. In this research, we have investigated anomaly detection approaches to address the challenges and filled the knowledge gap through the following four contributions:

Firstly, we have performed a literature review and conducted a thorough benchmark evaluation on existing state-of-the-art anomaly detection algorithms (i.e. Artificial Neural Network- Auto Encoder, Isolation Forest, Local Outlier Factor (LOF), Histogram-based Outlier Score (HBOS), Angle-based Outlier Detection (ABOD), Principle Component Analysis (PCA) and K-Nearest Neighbors (KNN)) using publicly available datasets from different domains such as health and finance. The experimental results show that Isolation Forest, HBOS and PCA are robust algorithms in terms of both high detection performance (Area Under the ROC Curve (AUC) = 0.95) and low computational time for large dataset.

Secondly, as one of the major contributions of this research, we have proposed a novel generic unsupervised anomaly detection model, which can be applied to anomaly detection of both financial and non-financial datasets. The essence of the proposed model consists in partitioning a bounded D-dimensional space (e.g. the unit hyper-cube I^D) by a sequence of random shapes, in which each data will be trapped either inside or outside, followed by probabilistic modelling of a pattern of falling inside or outside for a data point. Anomalous data which are rare and

different from the rest of the dataset will be assigned a higher anomaly score.

Thirdly, to investigate the robustness of the proposed anomaly detection model, we have performed a thorough sensitivity analysis under different hyper-parameters settings (i.e. the number of random shapes, shape of random shapes, etc.) and different publicly available datasets. The results show that the model performance stabilises as the number of random shapes increases. Furthermore, the shape of random shapes could affect the performance of the algorithm which needs to be optimised for a given dataset. Also, the results indicate that the algorithm's computational time increases linearly with the number of random shapes which shows the robustness of the algorithm for detecting anomalies in a timely manner.

Finally, we have applied the proposed algorithm on real Bitcoin prices as a case study and tested, evaluated and compared its performance with the benchmark algorithms such as Auto Encoder, Isolation Forest, LOF, HBOS, ABOD, PCA and KNN. The results show that the proposed algorithm achieves AUC = 0.94. Comparing to the benchmark algorithms, it also outperforms the existing algorithms by 8.5 percent increase while having low computational time.

Acknowledgements

I would like to thank my PhD supervisors (Prof. Liangxiu Han, Prof. Kevin Albertson and Dr. Yi Cao) for their considerable time and efforts supporting me during this PhD research. In addition, I would like to thank Manchester Metropolitan University (MMU) for providing funding for this research project.

Contents

List of Figures

List of Tables

List of Abbreviations

AD	**A**nomaly **D**etection
ATPs	**A**bnormal **T**rading **P**atterns
AUC	**A**rea **U**nder the ROC **C**urve
HBOS	**H**istogram-based **O**utlier **S**core
KDE	**K**ernel **D**ensity **E**stimation
KNN	**K**-**N**earest **N**eighbors
LOF	**L**ocal **O**utlier **F**actor
ML	**M**achine **L**earning
PCA	**P**rinciple **C**omponent **A**nalysis
ROC	**R**eceiver **O**perator **C**haracteristics

List of Symbols

D	Number of dimensions of a dataset
d	d-th dimension of a dataset
N	Number of data instances in a train dataset
$N^{'}$	Number of data instances in a test dataset
I	The unit interval $[0, 1]$
$\mathcal{X}$	a given train dataset
$\mathcal{X}^{'}$	a given test dataset
i	index of a train data instance
j	index of a test data instance
$\mathbf{x}_i$	i-th train data instance
$\mathbf{x}^{'}_j$	j-th test data instance
$\mathcal{S}$	a set of random shapes
$\mathbf{s}_h$	an individual random shape
v_h	Volume h-th random shape
v	Expectation volume of random shapes
$v^{'}_h$	Fraction of volume h-th random shape which falls inside the cube
$v^{'}$	Expectation volume of all random shapes which fall inside the cube
$\mathcal{V}_H$	Cumulative covered volume of random shapes after generating H shapes
$E(.)$	Expectation (mean)
$Ln(.)$	Natural logarithm
H	Number of random shapes
h	Index random shape
$\mathbf{c}_h$	Centre h-th random shape
r_h	Radius h-th random shape
Min_r	Lower bound for r_h
Max_r	Upper bound for r_h
L	Minkowski norm's parameter
$\mathcal{P}$	Posterior probability of observing a binary pattern of a data
p_h	Posterior probability of observing a data point as 1 (inside) based on the random shape $\mathbf{s}_h$
τ_h	Posterior probability of observing a data point as 1 or 0 (inside or outside of random shape $\mathbf{s}_h$)
θ_h	Number of observed data points (+1) as 1 or 0 (inside or outside) based on the random shape $\mathbf{s}_h$
α_h	Number of data instances falling inside the random shape $\mathbf{s}_h$
A	Vector to store and update all α_h

Dedicated to my parents

Chapter 1

Introduction

This chapter introduces the overview of the thesis. Section 1.1 explains the motivation and backgrounds regarding this PhD research, followed by research challenges and questions in Section 1.2. Next, aims and objective are included in Section 1.3. Contributions of this research are summarized in Section 1.4. Finally, the thesis structure is provided in Section 1.5.

1.1 Motivation and Background

Monitoring and surveillance of financial markets where traders can buy and sell financial assets (e.g. stock of a company), is an important problem. The reason is that financial markets may be manipulated intentionally by criminals who apply certain trading strategies (e.g. submitting large number of small-sized trading orders and cancelling these orders in milliseconds) in order to gain illegal profit by distorting the prices [1], [2]. As a result, legitimate traders may lose their capitals due to the unfair price change.

Since the trading data generated by criminal activities may be generally rare, different and abnormal compared with the majority normal trading data, anomaly detection algorithms can be applied for the purpose of law enforcement [3] by automatically monitoring and flagging suspicious abnormal trading behaviours/patterns (ATPs) in these markets. Automatic identification of ATPs is important as these patterns may indicate a potential fraud and crime to be further investigated rapidly by financial authorities [2], a task which can not be performed by humans since analysing billions of trading data by humans to find ATPs is impossible in a timely manner.

Regulators such as US Securities and Exchange Commission (SEC) introduced mechanisms (e.g. banning short selling for some period) to prevent market from crashing and experiencing large price movements [4].

Also, financial authorities may utilise sophisticated automatic surveillance approaches for detecting ATPs. It is important to mention that the details of such approaches are not publicly disclosed due to the security and privacy reasons in the financial domain. In other words, if the technical details of surveillance methods utilised by financial authorities are disclosed to the public, criminals may take advantage of those technical details by changing their criminal behaviours to bypass the surveillance methods.

Price manipulation as one of the most important type of Abnormal Trading Patterns (ATPs) is classified into the following types according to one of the first studies conducted in this field by Allen [5]:

- Information-based manipulation: Spreading false rumours in order to deceive investors planning to buy or sell a certain stock.
- Action-based manipulation: Actions that create an imbalance in demand and supply.
- Trade-based manipulation: Buying or selling a financial asset to create market impact.

The focus of this PhD thesis is on detecting trade-based ATPs as a case study via proposing a novel general-purpose anomaly detection algorithm. Therefore, the proposed algorithm can also be applied in domains other than finance (e.g. health domain) to detect anomalies in the data. This general-purpose feature of the proposed algorithm may enable it to be accepted and utilised by community of researchers and practitioners in different fields and industries in addition to the financial domain.

The ATPs happen as consequence of real trading (transaction of buying and selling) of a financial asset (e.g. stock of a company). The problem of ATPs detection (e.g. price manipulation detection in capital markets) can be converted into finding single or sequential anomalies in trading orders data (e.g. price, volume and time of the order) sent to the exchanges [1], [2].

The existing computational researches in this field generally applied (or modified) existing anomaly detection methods on financial trading data to identify irregularities in the data automatically.

For instance, Logistic Regression (LR [6]), Support vector Machines (SVM) and Artificial Neural Networks (ANN) were applied to daily stock prices to detect trade-based manipulations [7]. In that study, ANN and SVM outperformed the logistic regression. However, the supervised models in this study are not practical for utilisation in real-world scenarios as these require label information, which is not available due to confidentiality and security issues in the financial domain.

Zhai et al. [2] developed a more robust hybrid model than previous studies to detect single and sequential disruptive trading behaviours. First, the raw input vector [price, volume, timestamp] of each limit order are transformed into a three-dimensional feature vector to reduce the non-stationary properties of the original data. One Class SVM (OCSVM) and Hidden Markov Model (HMM, see Section 2.2.9 for more details regrading model-based anomaly detection) are utilised to detect single-order and multi-order (sequential) disruptive trading behaviours, respectively. In addition, the hybrid model uses a sliding window as an adaptive mechanism to update the model parameters. Furthermore, the authors in [2] injected artificial manipulation cases in four high liquid stocks on NASDAQ to test the performance of the model. Although the hybrid model outperformed the benchmark models, it may not be suitable for practical real-time anomaly detection due to the computational complexity of the model and the long time required for training and/or testing phases of the algorithms.

Another algorithm developed in this field is based on Hidden Markov model (HMM) is called Adaptive Hidden Markov with Anomaly states (AHMMAS) by Cao et al. [1]. This algorithm utilizes four features based on wavelet transformation and gradients of high-frequency prices. This model outperformed the existing benchmark models (K- Nearest neighbours and OCSVM), while the Area Under the Receiver Operating Characteristic (ROC) curve (AUC) [8] was used as evaluation metric. As the model does not update incrementally due to the computational complexity for modelling the probability density function (PDF), it must wait until significant changes are observed in data and then perform the updating action. This

may reduce its performance for early detection of manipulation cases. A comprehensive literature review is provided in Section 2.2.

To the best of our knowledge, the major problems of the existing methods are: 1) most of the studies focused on supervised algorithms (see Table 2.1) as a solution for this problem with an assumption that labelled datasets are available which is not true. In fact, due to the security and confidentially reasons and the time-consuming nature of hand labelling large amount of data by financial experts, real financial trading data are not labelled. Therefore, such supervised algorithms are not practically suitable for solving this problem, and 2) computational complexity of existing models (e.g. Neural Networks) requires long time for processing data and updating models' parameters, which will introduce further delays in detecting ATPs. This problem can lead to late detection of an abnormal pattern (e.g. price manipulation) in financial markets, which will consequently lead to loosing the investments of legitimate traders, if it is not detected rapidly.

High dimensionality and imbalance data, streaming processing, computational complexity of existing methods and unavailability of labelled datasets for training the algorithms are significant barriers to fully utilise anomaly detection algorithms for real-world surveillance and monitoring systems in which the existing methods fail to address. Therefore, a rapid unsupervised algorithm, which does not require the label information (e.g. which trading data is normal or abnormal) and can detect such ATPs rapidly and efficiently in real-world scenarios is needed to solve the issue.

1.2 Research Challenges and Questions

Designing an unsupervised algorithm for the task of detecting ATPs rapidly as mentioned above, is more efficient and appealing than reliance on the supervised algorithms. However, designing an unsupervised algorithm is very challenging due to:

- An unsupervised algorithm need not rely on label information to detect irregularities in datasets. However, the algorithm must rely only on the data features to learn the patterns of normality.

- Large volume and high-velocity of trading data. Analysing large volume of streaming trading data in near real-time requires a rapid algorithm.

In addition to the above items, as financial trading datasets are not usually labelled, evaluation of anomaly detection performance of an algorithm introduces another major challenge. Therefore, we are interested to find potential answers for the following research questions to address the above challenges:

- to design, develop, implement and propose a novel unsupervised anomaly detection algorithm for general purpose applications (applicable on different datasets including but not limited to financial/trading datasets).
- to determine appropriate hyper-parameters values so as to improve or maximise performance of the algorithm.
- to determine which features are required for detecting ATPs.
- to apply the proposed algorithm on financial trading data.
- to analyse financial trading data and label the dataset for evaluation purposes and comparison with existing benchmark algorithms.

1.3 Aims and Objectives

In order to address the research challenges and questions explained in Section 1.2, the aim of this PhD research will be to design, implement and evaluate a novel unsupervised anomaly detection algorithm capable of detecting anomalies in both financial and non-financial datasets (general purpose) in a timely manner. However, the main focus will be on fast detection of trade-based Abnormal Trading Patterns (ATPs) in financial markets as a case study.

The objectives will be to:

- Perform a thorough review of general anomaly detection algorithms for irregular trading patterns detection based on data-driven, computational approaches in the literature.

- Design a robust and computationally efficient unsupervised anomaly detection algorithm.
- Analyse financial data and label the dataset.
- Apply the proposed algorithm on the financial data.
- Evaluate and compare the performance of the proposed algorithm with existing algorithms.

1.4 Contributions

The contributions of this PhD research are listed below:

- A literature review and benchmark evaluation on existing anomaly detection algorithms such as Artificial Neural Network Auto Encoder, Isolation Forest, Local outlier Factor (LOF), Histogram-based outlier Score (HBOS), Angle-based Outlier Detection (ABOD), Principle Component Analysis (PCA) and K-Nearest Neighbors (KNN). These algorithms evaluated via Area Under the ROC Curve (AUC) and computational time on publicly labelled datasets from different domains such as finance and health to investigate which one has a combination of both high AUC and low computational time, thus serving as a foundation for developing a new algorithm. The results show that Isolation Forest, HBOS and PCA are robust algorithms in terms of high detection performance (Area Under the ROC Curve (AUC) = 0.95) and low computational time specially for large datasets.
- A novel unsupervised anomaly detection algorithm/model. The algorithm is intuitively based on the idea of partitioning a bounded D-dimensional space (e.g. the unit hyper-cube I^D) by a sequence of random shapes, in which each data will be trapped (isolated) either inside (encoded by 1) or outside (encoded by 0). Such a partitioning scheme, encodes each data into a binary pattern (sequence of 0s and 1s). Under the fundamental assumption for all anomaly detection algorithms that anomalous data (minority data) are rare and have different characteristics, the proposed algorithm learns the binary patterns by the

probabilistic modelling approach and can distinguish anomalous data whose binary patterns of trapping (inside or outside) based on the sequence of random shapes is significantly different from rest of the dataset (normal data or majority). Finally, the algorithm assigns an anomaly score for each data, which indicates the degree to which it is anomalous.

- Sensitivity analysis of the proposed algorithm under different hyper-parameters settings. We investigate the robustness of the proposed algorithm on publicly available datasets and show that the performance of the algorithm will stabilise as the number of random shapes increases. Furthermore, the shape of random shapes can effect the performance of the algorithm which needs to be optimised for a given dataset. Also, the results indicate that the algorithm's computational time increases linearly with the number of random shapes which shows the robustness of the algorithm for detecting anomalies in a timely manner.

- Application of the proposed anomaly detection algorithm to financial trading data. We applied the proposed algorithm on real Bitcoin prices as a case study and tested, evaluated and compared the performance of the proposed algorithm with the Auto Encoder, Isolation Forest, LOF, HBOS, ABOD, PCA and KNN. The results show that the proposed algorithm achieves AUC = 0.94. Comparing to the benchmark algorithms, it also outperforms the existing algorithms by 8.5 percent increase while having low computational time.

1.5 Thesis Structure

The organisation of this thesis is as follows:

- Chapter 2 reviews literature regarding existing computational methods for detecting Abnormal Trading Patterns (ATPs) in financial markets. Furthermore, general anomaly detection algorithms in the machine learning field are reviewed as these algorithms provide the foundation for constructing algorithms capable of detecting ATPs. The algorithms are categorised and classified based

on their underlying assumptions and compared with each other in order to identify strengths and weaknesses associated with each individual algorithm.

- In Chapter 3, publicly available datasets are utilised to compare the performance of the existing anomaly detection algorithms in terms of AUC of anomaly detection task and computational time. This chapter identifies the most robust algorithm(s) with high AUC and low computational time in order to build a theoretical foundation for proposing a new anomaly detection algorithm.

- A novel anomaly detection algorithm is proposed in Chapter 4. The theoretical foundations of the algorithm will also be described in that chapter.

- Sensitivity analysis of the hyper-parameters of the proposed algorithm is provided in Chapter 5.

- In Chapter 6, the proposed anomaly detection algorithm will be applied on real Bitcoin trading data (which are publicly available) and its performances for detecting ATPs is compared with benchmark anomaly detection algorithms. We determine that the proposed algorithm performs better than existing algorithms in terms of considering both AUC of outlier detection and computational time (measured in seconds).

- The thesis is concluded in Chapter 7.

- The Python code is provided in Appendix A.

Chapter 2

Background

This Chapter reviews the background and domain knowledge in financial trading. It is essential to understand the fundamentals of exchange markets (Section 2.1.1) and how limit order book (Section 2.1.2) works, in order to be able to design novel anomaly detection algorithms in this field. Furthermore, understanding these concepts facilitates understating how the process of manipulative trading tactics (Section 2.1.3) works.

Since the problem of detecting ATPs is closely related to anomaly detection, and the aim of this PhD (see 1.3) is to design and develop a novel general purpose unsupervised anomaly detection algorithm, which can be applied on any dataset including but not limited to financial data, we also review the general-purpose anomaly detection algorithms in Section 2.2. In that section, we also address and review how these algorithms have been modified/applied to financial data for ATPs detection.

Section 2.3 summarises and discuss the existing approaches for ATPs detection.

2.1 Abnormal Trading in Financial Markets

2.1.1 Exchange Markets

An Exchange market is a physical or virtual place, such as online platform, where demand and supply sides of the market called buyers and sellers, respectively, interact with each other in order to trade (exchange) an asset (e.g. stock of a company) [9]. There exist several markets such as stocks, Forex (Foreign Exchange), commodity, cryptocurrency and energy markets, each one, offers traders certain types of assets for trading. In addition, there exists derivative markets (e.g. options and futures

contracts), where the price of the asset is derived from an underlying spot price (current market price of a financial asset) [10].

As a result, nowadays traders have many diversified trading venues to trade and invest on portfolio of assets in different markets. In addition, with the rise of high-speed computer/networks, human traders and the whole manual trading processes are being replaced with intelligent trading algorithms trading on behalf of humans or corporations [11]. Such a combination of algorithmic trading with the ability to buy or sell an asset in the fraction of a second has opened a new era of trading called high-frequency trading (HFT) [12] which itself creates new opportunities, challenges and problems.

2.1.2 Limit Order Book

According to [2], the exchanges (where buyers and sellers exchange financial assets) have a limit order book (LOB) for each asset (e.g. stock of a company), which is a collection of the entire trading orders received by traders for that particular asset. If a trader wants to buy a financial asset at the current market price known as spot price, he/she can submit a market order. Alternatively, the trader may want to wait until the price reaches to a certain level before buying or selling the asset. In that case, he/she can submit limit orders by specifying the desired price level and amount of the asset to be bought or sold. Therefore, each limit order can be viewed as an intention of a buyer (or seller) to buy (or sell) a certain amount of a particular asset at a certain price. The exchange keeps the record of non-executed limit orders of buyers and sellers on buy (bid) and sell (ask) sides of the order book, respectively and matches the orders, accordingly. The best bid price is the maximum price among buyers (bid side of the LOB). Similarly, the best ask price is the minimum price among the sellers (ask sides of the LOB). The difference between these two values; $BestAskPrice - BestBidPrice$ is called bid/ask spread.

Many traders who believe that bid and ask sides of the LOB are true reflections of demand and supply may use the information provided in the LOB to predict the future price movement and consequently initiate a trade. However, when manipulators trade in the market, they may submit fake orders without any intention of

executing that order as a means to influence the behaviour of other normal traders [1].

2.1.3 Manipulative (Abnormal) Trading Strategies

Financial markets can be manipulated intentionally by traders who apply certain trading strategies in order to gain illegal profit. As a result, other traders may lose their capital. Furthermore, the image of investing in financial markets as a secure and well-regulated place for investing may be destroyed. Therefore, detecting and preventing the manipulators is a must for financial authorities.

Price manipulation as one of the most important type of Abnormal Trading Patterns (ATPs) is classified into the following types according to one of the first studies conducted in this field by Allen [5]:

- Information-based manipulation: Spreading false rumours in order to deceive investors planning to buy or sell a certain stock.
- Action-based manipulation: Actions that create an imbalance in demand and supply.
- Trade-based manipulation: Buying or selling a financial asset to create market impact.

The focus of this PhD thesis is on detecting trade-based ATPs via anomaly detection algorithms. Such ATPs happen as consequence of real trading (transaction of buying and selling) of a financial asset (e.g. stock of a company).

Although, there are various types of trade-based manipulation tactics, these strategies are actually the same method in which, "non-bona fide" orders (fake orders intended to deceive other investors and create false image of supply or demand, e.g. large orders away from best Bid/Ask or sequential normal sized orders inside Bid/Ask spread) are submitted to the exchange to influence the price upward or downward. When the equity price is affected, the "non-bona fide" orders are cancelled and the manipulators can gain profit due to the artificial price change caused by executing their "bona fide" order on the other side of the order book [2].

The manipulation tactics and disruptive trading strategies can be implemented via a single order or sequential (multiple) orders to delude other investors. Each of

these techniques have some abnormal characteristics. For instance, sequential spoofing orders (multiple fake low-volume trading orders being submitted within the best bid-ask price range which are then being cancelled in short time intervals to change best bid and ask prices) may create a saw shape in the Bid and Ask time series. Another important feature of sequential manipulation strategies is the cancellation time of orders in short time intervals [2].

2.2 Anomaly Detection Algorithms

The problem of abnormal trading patterns (ATPs) detection (e.g. price manipulation detection in capital markets) can be converted into finding single or sequential anomalies in trading data (e.g. price, volume and time of a trade). Therefore, it is essential to review and understand what is meant by the term "anomaly" and how an anomaly can be detected automatically. This section provides an overview of anomaly detection definitions and general purpose anomaly detection algorithms along with particular algorithms, which were modified/applied on financial data for detection of ATPs. Anomalous data are rare (minority) data that deviate from majority (normality) patterns [13], [14]. In this thesis, anomalies are defined as collection of data points in a dataset, which are: 1) few (minority) and 2) different (having distinguishable statistical characteristics) from the rest of data points. These two features may separate anomalies from the majority of data points without the requirement of labelling the data points in advance.

Anomaly detection has many applications such as credit card fraud detection, network intrusion detection, loan application processing, fault detection, medical condition monitoring, detecting abnormal images and so on [15]. The anomaly detection algorithms can be classified into the following groups according to the availability of labelled datasets [13]:

- **Supervised:** When data are labelled (normal and abnormal data are flagged), a classifier algorithm can be trained on a subset of data using these labels as accurate outputs to learn the input-to-output mappings. The learnt model is used to predict the output for a given additional data input.

- **Semi-supervised:** When partially labelled data are available or only one class of data (e.g. just normal instances) are labelled, the algorithm learns the model of normality to describe the labelled data. Any data point that deviates from the model is flagged as an anomaly.

- **Unsupervised:** When data are not labelled (the algorithm does not know which instance is normal or abnormal), the algorithm assumes that anomalous data

are few and different from the rest of data. In other words, it learns the structure of a dataset by assuming that abnormal instances are differentiable from normal ones (these two types of patterns differ in some statistically significant fashion).

Due to security and confidentiality reasons, and the time consuming task of hand-labelling data, many real-world datasets are not labelled. Therefore, unsupervised anomaly detection algorithms are more popular than the supervised and semi-supervised approaches.

A detailed review of existing anomaly detection algorithms can be found in [13]–[15]. In the following sections, the principles of these algorithms are reviewed and compared.

2.2.1 Classification-based Anomaly Detection

Classification-based anomaly detection includes supervised algorithms which require a training set of labelled data. They utilise this to learn the mapping from input to output (labels) in a training phase [6]. Once trained, the algorithm classifies data points into either normal class (0) or abnormal (1). Existing supervised machine learning algorithms such as Artificial Neural Networks (ANN) [6] and Support Vector Machines (SVM) [16] can be utilised for supervised anomaly detection. The advantage of this approach is that, once the classifier is trained, it can predict the anomaly label very fast for subsequent data. However, this type of method has several drawbacks:

- Labelled data are needed to train the algorithm [13], which may not be available in real-world applications.
- Real-world datasets are very imbalanced (the proportion of abnormal instances in the dataset is very low), which may introduce problems for existing classifiers to differentiate the patterns of normality and abnormality without bias [13].

Logistic Regression (LR [6]), SVM and ANN were applied to daily stock prices to detect trade-based manipulations in Istanbul Stock Market [7]. In that study, differences of daily price, trading volume and volatility of stocks in comparison with

a market index (non-manipulated benchmark) were used as input features for the algorithms. Although ANN and SVM outperformed the logistic regression, the research was conducted only on daily prices with known manipulation cases (supervised learning) and did not consider the limit order data. Furthermore, the supervised algorithms are not adaptive and may not be able to detect high frequency manipulative trading strategies that evolve over time in an unsupervised environment (the data in real application may not be labelled which is typical in capital markets). In addition, the assumption of that paper that deviations of market variables from a benchmark (e.g. market index) is an indicator of manipulation may not be true since the deviations may have been originated from legitimate sources such as economic events rather than being the result of price manipulations as explained in [2].

Leangarun et al. [17] applied ANN and a 2-dimensional Gaussian model (see Section 2.2.5 for details regarding density-based anomaly detection) on intraday trading data (price, volume and limit order book) of Amazon, Intel and Microsoft stocks to detect pump-and-dump (large-volume trading orders, which are submitted away from the best bid and ask prices to create a delusion of fake demand or supply in the bid and ask sides of the LOB) and spoofing (multiple fake low-volume trading orders being submitted within the best bid-ask price range which are then being cancelled in short time intervals to change best bid and ask prices) manipulations, respectively. Since the raw data were not labelled, the authors labelled the data by creating binary rules based on their own assumptions of manipulation conditions. Mean square error was utilised as a performance evaluation metric and they claim their models were able to detect manipulations. The supervised ANN trained on labelled data can only detect those specific type of manipulations determined by the authors and not unseen patterns.

2.2.2 One-Class Anomaly Detection

One-class anomaly detection includes semi-supervised algorithms. In datasets containing only one class (e.g. normal class), a One-class Support Vector Machine (OCSVM) [18] learns the boundaries (hyper plains) of normal class against the centre of origin via the application of kernel functions. Any data which falls inside the learnt boundary is classified as normal, otherwise as abnormal. Although this semi-supervised

method requires only one class to be labelled in the dataset, it still has the drawbacks mentioned in the Section 2.2.1 regarding the supervised algorithms. Therefore, these methods are less desirable than unsupervised approaches in anomaly detection tasks.

V-detector algorithm [19] is another one-class (semi-supervised) anomaly detection algorithm based on finding anomalous regions of space given labelled normal data. The algorithm divides the space into normal and abnormal sub-spaces based on the assumption that the normal subspace includes hyper-spheres with the centres located at normal training data points. Any test data point that falls inside this normal region is declared as normal, otherwise the data is identified as anomaly. This algorithm is equivalent to one-class SVM to find the boundaries of the normal samples. V-detector algorithm has the following drawbacks:

1) V-detector algorithm requires labelled normal training data to work properly. When the training data for V-detector is contaminated with abnormal data (e.g. some of the data points in the training set are anomalies but are included in the training data set thus are labelled incorrectly as normal data), the algorithm fails to correctly identify test data points as anomaly, which are close to those incorrectly labelled training data points because such test data points fall inside the normal hyper-spheres. The reason is that the algorithm assumes that the training dataset includes only normal data points and builds the normal sub-space based on this assumption. In other words, contamination of training dataset with abnormal data points may distort the true normal subspace to be learned by the algorithm, which will create bias in the performance of the anomaly detection.

2) Hyper-spheres are generated data-dependent in V-detector algorithm based on the nearest-neighbor distance of a random point to the normal samples, which is not computationally efficient for big high-dimensional data.

3) if the distribution of normal samples changes over time (which is the case for financial non-stationary time-series), hyper-spheres of V-detector algorithm need to be regenerated every time (because the boundaries of normal samples changes and previous shapes are no longer valid as anomaly detector). This is computationally expensive.

4) V-detector only declares an anomaly label (no anomaly score), if a data point

falls inside one of the detector hyper-spheres. For many real-world application, providing a label is not enough and data need to be ranked according to their anomaly score.

Basically, the V-detector algorithm is trying to achieve what a Kernel Support Vector Machine (SVM) achieves which is finding abnormal regions of space which have boundary with sub-spaces, where are populated with normal data.

Zhai et al. [2] developed a more robust hybrid model than study performed by [7] to detect single and sequential disruptive trading behaviours. First, the raw input vector [price, volume, timestamp] of each limit order are transformed into a three-dimensional feature vector to reduce the non-stationary properties of the original data. One Class SVM (OCSVM) and Hidden Markov Model (HMM, see Section 2.2.9 for more details regrading model-based anomaly detection) are utilised to detect single-order and multi-order (sequential) disruptive trading behaviours, respectively. In addition, the hybrid model uses a sliding window as an adaptive mechanism to update the model parameters. Furthermore, Zhai et al. [2] injected artificial manipulation cases in four high liquid stocks on NASDAQ to test the performance of the model. Although the hybrid model outperformed the benchmark models, it may not be suitable for practical real-time anomaly detection due to the computational complexity of the model and the long time required for training and/or testing phases of the algorithms. In addition, the fundamental structure of the hybrid model does not change over time, although it has an adaptive mechanism. Therefore, the hybrid model will not be able to detect evolutionary manipulative tactics and the performance of the algorithm may decrease over time due to the static structure of the model.

2.2.3 Distance-based Anomaly Detection

Distance-based anomaly detection algorithms are unsupervised methods based on the general assumption that abnormal data in high-dimensional spaces are far away from other data, that is, they differ significantly [14].

Mathematically, consider a D-dimensional dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ with N data instances, such that each data instance $\mathbf{x_i} = [x_{i,d}]_{d=1}^{D}$ is a D-dimensional vector.

Distance-based algorithms use a distance (similarity) measure between any two D-dimensional points $\mathbf{x_i}$ and $\mathbf{x_j}$; $i, j \in \{1, 2, \ldots, N\}$, denoted by $Distance(\mathbf{x_i}, \mathbf{x_j})$, to compute how close the points are located to each other. Different similarity measures such as Minkowski [20] (Equation 2.1), Euclidean (Equation 2.1 with $L = 2$) and Manhattan (Equation 2.1 with $L = 1$) can be utilised to measure the distance [14].

$$Distance(\mathbf{x_i}, \mathbf{x_j}) = (\sum_{d=1}^{D} |x_{i,d} - x_{j,d}|^L)^{\frac{1}{L}}. \tag{2.1}$$

Next, an anomaly score for any point $\mathbf{x_i} \in \mathcal{X}$ is computed as the sum of its distance to all the other points in the dataset as shown in

$$AnomalyScore(\mathbf{x_i}) = \sum_{\mathbf{x_j} \in \mathcal{X}} Distance(\mathbf{x_i}, \mathbf{x_j}). \tag{2.2}$$

Finally, all the points are ranked in a list from high to low anomaly scores and points in the top of the list may be considered anomalies according to a certain threshold, e.g. having an anomaly score falling in the top 5 percent. This general algorithm has different versions based on which similarity measures are utilised. In addition, instead of computing the sum of distances to all points in the dataset, which is computationally very expensive, only k-nearest neighbours (points) of $\mathbf{x_i}$ may be considered for the calculation of the anomaly score. Alternatively, the distance of k-th neighbor of a given data point can be utilised as anomaly score[21] . Moreover, the average or median of the distance of k-nearest neighbours of point $\mathbf{x_i}$ can be utilised for the computation of an anomaly score, which may improve the performance of the algorithm [14], [22].

The advantage of distance-based anomaly detection algorithms is that they do not need labelled data and are easy to implement as computer programs. However, their computational complexity increases quadratically with the size of dataset ($O(N^2)$) as the distance of any pairs of data must be computed [8], [13], [14]. This time-consuming drawback makes it impractical to utilise distance-based algorithms for large datasets due to the computational resource constraints.

As a result of curse of dimensionality [23]–[25], standard distance functions such as Euclidean distance utilised in the anomaly detection algorithms, do not work well

in high-dimensional data due to the 1) noise effect of irrelevant dimensional values [26] and 2) distance concentration (data sparsity) of data points, which means most of the pairwise distances are similar and not distinguishable.

Survey [27] reviews distance measures utilised in the literature of Network Intrusion Detection and suggests that fractional l_L distance functions are useful for anomaly detection in high-dimensional datasets Particularly, [28] shows that $0.5 \leq L \leq 1$ are useful for such settings. Aggarwarl et al. [29] proposes fractional distance metrics as special case of Minkowsi metric with $L \leq 1$. They show that these fractional metrics can improve the performance of algorithms in higher dimensions as there will be more contrast between min and max distance of uniform points from a target data point. In other words, using such metrics will results in more discrimination between pair-wise distances of data.

2.2.4 Clustering-based Anomaly Detection

Clustering algorithms are unsupervised machine learning algorithms that divide the data into k groups or clusters based on the goal that similarity of the points within a cluster should be maximized, while the similarity between points from different clusters be minimized [14]. The most famous clustering algorithm, k-means [30] initializes the centre of clusters (centroids) randomly as set $C = \{c_r\}_{r=1}^{k}$, then the following two steps run alternately until a termination criterion is reached:

1. Assignment step: the points closest to each centroid are assigned to that cluster.
2. Update step: the average of points assigned to c_r is computed as r-th updated centroid [14].

This algorithm can have variety of versions such as computing the median in the update step. This is called k-median clustering since the calculation of the mean is sensitive to outliers while determination of the median is more robust to outliers. Furthermore, this type of algorithm can be classified as hard clustering since every point in the dataset is assigned to a cluster.

The assumption in the hard clustering-based algorithms is that, points which are far away from centroids of clusters, are potential anomalies. For example, an

anomaly score can be computed by the distance from each point to its cluster's centroid (or all the clusters' centroids), or average or median distance of each point to the points within its cluster. Alternatively, points in clusters with size below a threshold are potential anomalies. Since the clustering algorithm is fast, integrating it with distance-based approaches results in improvements in runtime.

In contrast to the hard clustering, soft clustering algorithms enable points to be assigned to many clusters at the same time and it is possible that some points do not belong to any cluster. These are classified as anomalies [14]. $\Omega(\mathbf{x_i}, c_r)$, a real-number in range $0 - 1$ which quantifies the degree of membership of a point $\mathbf{x_i}$ to r-th cluster. The fuzzy k-means clustering algorithm [31] computes Ω for each point in the dataset according to Equation 2.3 and then normalises the membership values of each point to sum to one in the assignment step. Next, it updates the Ω-weighted centroid of the clusters.

$$\Omega(\mathbf{x_i}, c_r) = \frac{1}{\exp\left(Distance(\mathbf{x_i}, c_r)^2\right)}. \tag{2.3}$$

The drawback of the clustering algorithms is the sensitivity to k and the initialization of clusters, which can significantly change the performance of the anomaly detection algorithm. Furthermore, the algorithms are not deterministic [14]. Another type of clustering algorithm which does not need k to be specified in advance is agglomerative clustering [14] algorithm. With computational complexity $O(N^3)$, it is based on the idea that first, small clusters are formed in datasets and then these small clusters merge together (based on distance to centroids) to form bigger clusters until a termination criterion is reached.

2.2.5 Density-based Anomaly Detection

The assumption behind the density-based anomaly detection algorithms (which are unsupervised) is that data in less-dense areas of high-dimensional space, where the number of data points per volume of space determines density, are more anomalous than data located in denser regions [14]. Estimation of density can be performed by:

- Dividing the space into small sub-spaces (grids) and counting the number of points in each grid. A histogram in one-dimension is an example of such

density estimation. This simple and fast method (in low-dimensional spaces) struggles in high-dimensional spaces due to the large number of grids, which need to be investigated. Furthermore, this approach is sensitive to the size of the grids. In addition, the density is estimated in discretised spaces and not in continuous form.

- Assuming that data follow a specific type of distribution (e.g. Gaussian) and estimating the parameters of the distribution based on the data. However, such an assumption may not be true for the real-world datasets.

- Utilizing kernel functions [14] which are positive and symmetric functions with integral of one from $-\infty$ to $+\infty$, as similarity measures between points. The distance of point $\mathbf{x_i}$ from other points in the dataset is utilised as input of the kernel function to estimate a continuous form of density around point $\mathbf{x_i}$.

Furthermore, density can be viewed as reversely proportional to distance of points from each other. Points that are close to each other are located in denser areas while far-apart points are located on sparse regions. This idea serves the basis for a well-known algorithm called Local Outlier Factor (LOF) [32], which compares local density of point $\mathbf{x_i}$ with its k-nearest neighbours and can detect both local and global anomalies in the dataset.

Leangarun et al. [17] applied a 2-dimensional Gaussian model on intraday trading data (price, volume and limit order book) of Amazon, Intel and Microsoft stocks to detect pump-and-dump (large-volume trading orders, which are submitted away from the best bid and ask prices to create a delusion of fake demand or supply in the bid and ask sides of the LOB) and spoof-trading manipulations (multiple fake low-volume trading orders being submitted within the best bid-ask price range which are then being cancelled in short time intervals to change best bid and ask prices), respectively. Since the raw data were not labelled, the authors labelled the data by creating binary rules based on their own assumptions of manipulation conditions. Mean square error was utilised as a performance evaluation metric and they claim their models were able to detect manipulations. Matched volume and cancellation volume of orders as two dimensions used in the 2D-Gaussian model and points outside of 95 percent confidence interval are declared anomalous. The paper does

not provide any justification regarding why the Gaussian distribution is suitable for modelling the trading data.

2.2.6 Depth-based Anomaly Detection

The Depth-based unsupervised anomaly detection algorithm [33], [34] assumes data are distributed like an Onion (layer by layer) in high-dimensional space. The algorithm starts from the outer layer to determine which data are located there. Such a process is recursively repeated layer by layer until all data points in the datasets are assigned a layer number (depth from outer layer). Convex hull analysis [33] is utilised to determine the layers in the algorithm to identify data points located in the tail of the distribution of a dataset. Once such data points are removed from the dataset, this process can be repeated sequentially until there is no data point left in the dataset.

Data points which are located in the outer layers of the Onion are considered more anomalous than data which are located in the dense inner layers. Therefore, the depth can be utilised as an anomaly score for comparing the degree of anomalousness of data points.

Although, the algorithm appears very easy to implement, it will be only capable of finding anomalies located in the outer layers and sparse regions of space. These data points are considered global extreme values of the dataset. However, local anomalies, especially those which are trapped between dense clusters of data and are in the middle layers, may not be detected properly.

The main fundamental problem with this algorithm lies in its assumption that data have a single cluster that can be viewed as layer-by-layer shape. This is not valid in real-word datasets as those datasets have multiple clusters located in different regions' of space.

2.2.7 Angle-based Anomaly Detection

The Angle-based unsupervised anomaly detection algorithm proposed by Kriegel et. al. [35] discriminates outliers from normal data points based on computation of angle rather than distances. With a dataset consisting of N data points, an angle is

defined between any three data points forming a triangle. One of the vertices of the triangle will be the data point to be classified as normal or abnormal, and the other two vertices are chosen among a set of $\frac{(N-1)(N-2)}{2}$ combinations corresponding to choosing two points among the rest of the $N-1$ data points [33].

The assumption behind this algorithm is that, the standard deviations of the angles computed for each data point will be higher if such data is normal, while for an abnormal data point, the standard deviation of the angles will be lower. Therefore, the standard deviation of angles will be utilised as a measure for computing anomaly score for each data point.

Such an algorithm can detect global extreme value anomalous data, which are located in the tail of the data distribution (outer bound). However, it will have difficulty detecting abnormal data points which are local and located hidden between and close to dense normal clusters since the standard deviation of angles will be similar to the normal data points.

Another drawback of this algorithm is its computational cost which is cubic ($O(N^3)$). Therefore, it is not practical to utilise such an algorithm for processing of large datasets.

2.2.8 Isolation-based Anomaly Detection

The Isolation-based unsupervised anomaly detection algorithm proposed by Lui et. al. [36], [37] utilises a completely different approach in comparison with previously mentioned anomaly detection algorithms.

In this algorithm, no distance metric or any density estimation is utilised. Instead, a concept called "Isolation" is proposed by the authors which is a computational mechanism for separating (isolating) normal data points from abnormal ones. Such a mechanism involves construction of a random tree structure of data in which each node corresponds to a random division (partitioning) of high-dimensional data into two parts (sub-spaces). Practically, this can be achieved by selecting a random axis-parallel hyper-plane. Those data which are located on the left or right of each randomly-chosen plane, correspond to the left and right division of each node in the tree. This formulation is applied on the data and the tree grows recursively, until there will be one unique sample in the final leaf nodes.

Once such a tree is constructed, the length of the path from the root node until the leaf node for each data point will be utilised as a measure for computing the anomaly score of that data which means less path length results in higher degree of anomalousness. The assumption is that outliers will be trapped (isolated) sooner and located more on the top parts of the tree, thus having shorter path length and higher anomalous score than normal data, while normal data points will be trapped in the bottom of the tree structure, thus having a longer path length and less anomaly score than the anomalous data points.

Since the proposed tree structure is a random structure, there is a possibility that some outliers are still located on the bottom of the tree leading to less accurate anomaly detection results. To overcome this problem that just one tree may not be sufficient for anomaly detection by this approach, the authors propose the construction of an ensemble (forest) of trees. Such an ensemble approach involves construction of set of random trees individually and then computing the average (mean) path length for each data point to give a more accurate result.

Although, the ensemble of trees approach may outperform an algorithm based on just one tree in terms of anomaly detection accuracy, it is more computationally expensive than construction of a tree. To address this issue, a sub-samples of data (rather than the whole dataset) during the construction of tree can be utilised to reduce the computational time. However, the number of sub-samples must be chosen carefully depending on the dataset as very few sub-samples may not capture and represent the true distribution of the data. Therefore, there might be a trade-off between accuracy and computation time.

Although, the Isolation Forest is among one of the robust anomaly detection algorithms, since it utilises axis-parallel hyper-planes and linearly partitions the space into hyper-rectangles, it will need large number of partitions for real-world non-linear datasets in order to truly capture the intrinsic of the dataset, resulting in more computational cost.

In the worst-case scenario, $2D$ number of hyper-planes (two hyper-planes for each dimension) are needed for each outlier data point to be isolated from the rest of data points. For instance, in two-dimensional and three-dimensional spaces, squares and cubes are required corresponding to four lines and eight surfaces in such spaces,

respectively. This issue may demand more memory and computational resources for the algorithm to have a reasonable accuracy as the number of dimensions grow.

2.2.9 Model-based Anomaly Detection

In the model-based approach [14], [33], a model is constructed to describe the normal data. The assumption is that data that deviate from the model's predictions are anomalous. Alternatively, if model's parameters change significantly during the training phase, it can also be considered as abnormality. These deviations are quantified by distance functions into anomaly score for a test data.

The models which can be supervised or unsupervised are classified as linear and non-linear based on assumption whether data features have linear or non-linear relationships [38]. In the linear version, a regression-based model or PCA-based model [39] is constructed, which represent the linear low-dimensional hyper-plane in which, the original data are projected. Large deviations (distances) from such hyper-plane triggers anomaly detection algorithm. If the data have non-linear properties, linear models may fail to capture the curvature of data, thus non-linear models will be useful in such cases. For instance, Auto Encoder Neural Networks [40] is a nonlinear algorithm, which can find the non-linear patterns in data. The Auto Encoder (an architecture of neural networks) applies a series of non-linear transformations via hidden neural layers to encode the data to a low dimensional space and then, decoding the low dimensional space back to the original space, in order to reconstruct the data. Anomalous points are data, which have high reconstruction error. The drawback of these algorithms is their high computation costs for training such non-linear models. Survey [41] and [42] provided comprehensive review of anomaly detection methods based on the neural networks.

The model-based approach for anomaly detection is also suitable to find anomalous patterns in time series and sequence data. The problem here is how to construct the model of an underlying data generating process. Such a generating process is usually a probabilistic model (in contrast to previous deterministic models), which describes how the data are generated or drawn from combination of underlying probability distributions [43], [44]. For example, an Auto Regressive Integrated Moving Average (ARIMA [45]) model can be constructed for continuous

time-series. However, these models struggle with evolutionary data. So, for dynamic data, window-based approaches are utilised to retrain the model on each sequence time-intervals. This increases the computational complexity of the problem. Alternatively, Markov Models can be used for modelling discrete sequences [6], [43]. These probabilistic models learn the conditional probabilities of the transitions from one state (category) to another one and a less probable transition observed indicates irregularity.

The advantage of model-based anomaly detection is its flexibility to model complex data structures. The challenge is how to build and train a model that can capture, in a timely manner, not only the evolutionary nature of data but also the correlations between features of data.

The hybrid model developed by Zhai et al. [2] detects single and sequential disruptive trading behaviours. First, the raw input vector [price, volume, timestamp] of each limit order are transformed into a three-dimensional feature vector to reduce the non-stationary properties of the original data. One Class SVM (OCSVM, see Section 2.2.2 for details regarding One-class anomaly detection) and Hidden Markov Model (HMM) are utilised to detect single-order and multi-order (sequential) disruptive trading behaviours, respectively. In addition, the hybrid model uses a sliding window as an adaptive mechanism to update the model parameters. Furthermore, the authors in [2] injected artificial manipulation cases in four high liquid stocks on NASDAQ to test the performance of the model. Although the hybrid model outperformed the benchmark models, it may not be suitable for practical real-time anomaly detection due to the computational complexity of the model and the long time required for training and/or testing phases of the algorithms. In addition, the fundamental structure of the hybrid model does not change over time, although it has an adaptive mechanism. Therefore, the hybrid model will not be able to detect evolutionary manipulative tactics and the performance of the algorithm may decrease over time due to the static structure of the model.

Another model-based algorithm developed for ATPs detection based on HMM is called Adaptive Hidden Markov with Anomaly states (AHMMAS) by Cao et al. [1]. This algorithm utilises four features based on wavelet transformation and gradients of high-frequency prices. Hidden states of the AHMMAS correspond to normal

and manipulation states which are inferred by observations of prices as observation states and a rolling window are also proposed to update the model only when the statistic of data changes significantly as an adaptive mechanism. Seven stocks on NASDAQ and London Stock Exchange along with simulated prices were selected to test the algorithm. This model outperformed the existing benchmark models (K-Nearest neighbours and OCSVM, see Sections 2.2.3 and 2.2.2 for details regarding distance-based and one-class anomaly detection, respectively) while the Area Under the Receiver Operating Characteristic (ROC) curve (AUC) [8] used as evaluation metric. As the model does not update incrementally due to the computational complexity for modelling the probability density function (PDF) of hidden states, it must wait until significant changes are observed in data and then performs the updating action. This may reduce its performance for early detection of manipulation cases.

Regression models have been developed to predict stock manipulations in Tehran stock exchange [46]. First, stock companies are labelled to 1 (manipulated) and 0 (non-manipulated) by statistical tests. Then, the models regress dependent variable (label 0 or 1) over independent variables such as the size of the company, ratio of price to earnings (P/E), information clarity index (released daily by stock exchange), and rank of stock liquidity (released daily by stock exchange). The model's accuracy is reported to be 90 percent on self-labelled data. However, the input variables are judgmental and the supervised methods are not suitable for unsupervised high-frequency detection due to the computational complexity and static nature of such regression models.

An Agent-based model (artificially simulated stock market) with a combination of genetic programming (a computational method that mimics the evolutionary processes in nature to solve optimization tasks [47]) has been utilised for price manipulation detection in the French stock market [48]. The model consists of 2000 simulated agents as traders whose trading rules (strategies) are evolved by genetic programming while receiving information about the market and trying to maximize their utility function (wealth). The dataset used in that paper includes daily prices and volumes of trades regarding CAC40 index and two major companies within the index. The author claims that deviation of forecasted prices of the model from real prices is sign of manipulation and quantifies this by a metric called Forecast

Directional Accuracy (FDA), the percentage that forecasted price aligns with real directional price change. FDA below 50 percent represents an incorrect model prediction which is a sign of manipulation. Based on this approach, the author indicates the existence of disruptive trading behaviours in the French market. However, since the data are not labelled, and also other economic factors may influence the prices, this approach only indicates signs of manipulations and not the actual manipulations. Furthermore, due to the computational complexity of genetic programming approaches, this method is not suitable for real-time detection of manipulations.

2.3 Summary of the algorithmic studies for ATPs detection

Since the focus of this PhD research will be on fast detection of trade-based Abnormal Trading Patterns (ATPs) in financial markets as a case study, Table 2.1 summarises and compares the existing algorithmic studies developed for ATP detection purpose. Each method is categorised based on the required supervision, type of anomaly detection approach and the run-time (corresponding to whole training and test phases) to perform ATP detection. The table clarifies the identified issues which motivated this study. The main identified issues are 1) high supervision and 2) high run-time requirements for existing methods which prevent those algorithms to be utilised for the propose of real-world ATP detection.

Most of the existing methods focused on supervised algorithms as an easy solution for the problem of ATPs detection with an assumption that labelled datasets are available which is not true. In fact, due to the security and confidentially reasons and time-consuming of hand labelling large amount of data by financial experts, real financial trading data are not labelled. In other words, we do not know which trade is normal or abnormal in advance and a robust method is required for identifying the abnormal trades. Therefore, such supervised algorithms are not practically suitable for solving this problem. Alternatively, an unsupervised algorithm which does not require the label information (e.g. which trading data is normal or abnormal) and can detect such ATPs rapidly and efficiently in real-world scenarios is needed to solve the issue.

In addition, large amount of trading data and its high-velocity requires fast algorithms with low run-time capable of processing the data in a timely manner, a requirement which most of the existing methods fail to address.

To date, the problem of detecting ATPs in a timely manner and unsupervised fashion has remained a challenge due to the high-volume and high-velocity of trading data. Therefore, this research attempts to fill the gap by proposing a novel general-purpose unsupervised algorithm (see Chapter 4), which can be applied on both financial and non-financial data and is capable of detecting ATPs rapidly.

TABLE 2.1: Summary and comparison of the algorithmic studies for detection of ATPs.

Approach	Supervised	Semi-supervised	Unsupervised	Run-time	References
Classification-based	**LR, ANN, SVM, KNN, Decision Trees**			High	[7], [46], [49], [50], [51], [1], [50], [52]
One-class-based		**OSVM**		High	[2], [1], [52]
Density-based			**Gaussian Model**	High	[17], [1]
Model-based	**HMM, Agent-based model, Genetic model, Regression**			High	[2], [53], [52], [17], [17], [1], [54], [46], [55], [49]

Chapter 3

Anomaly Detection Benchmark Evaluation

This chapter compares the performance of existing popular anomaly detection algorithms on public datasets in different industry sectors such as trading, finance and health. Section 3.1 explains the details regarding the datasets, feature extraction and further pre-processing steps before feeding the data to the machine learning algorithms. Section 3.2 reviews the evaluation (performance) metrics utilised in the computational experiments. Experiments setups are provided in the Section 3.3. Finally, results and discussions are included in Section 3.4.

3.1 datasets

In this study, we utilise publicly available labelled datasets such as Credit Card Fraud Detection [56], [57], [58], [59], [60], [61] and Breast Cancer [62], [63], [8]. These datasets have been labelled and feature extracted by experts in those fields and are ready to be utilised by machine learning algorithms.

The Credit Card Fraud Detection is downloaded from [57]. It is a 30-dimensional dataset (as input features) with 284,807 data samples. Each data sample is a credit card transaction which is labelled as normal or fraudulent (abnormal) transaction. 492 of the data samples are associated with fraud. For the security and anonymity reasons, the original features have been transferred by PCA algorithm so the data cannot be linked to an individual person. The reasons for choosing this dataset are:

1) its domain (fraud detection) is closely related to this PhD topic which is anomaly detection in financial markets, and 2) it has a large number of data samples.

The Breast Cancer dataset is downloaded from [62]. The original Breast Cancer Wisconsin (Diagnostic) dataset [63] from UCI Machine Learning repository, were prepared for the task of unsupervised anomaly detection by keeping only 10 anomalies in the total 367 data samples [8]. The 30 input features are computed from medical images and each data sample is labelled as malignant (abnormal class) or benign (normal class). We choose this dataset from a different domain from finance to also evaluate the algorithms on non-financial data as we are interested in developing a general-purpose algorithm applicable to both financial and non-financial datasets.

3.2 Evaluation Metrics

Characteristic Curve (ROC) [64] is a curve, which plots True Positive Rate and False Positive Rate at various thresholds of a binary classifier[6], where:

$$TruePositiveRate = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{3.1}$$

$$FalsePositiveRate = \frac{FalsePositives}{FalsePositives + TrueNegatives} \tag{3.2}$$

True Positives, False Negatives, False Positives and True Negatives are predictions of a binary classifier (a classifier, which outputs positive label 1 and negative label 0) that are correctly predicted as 1, mistakenly predicted as 0, mistakenly predicted as 1 and correctly predicted as 0, respectively.

We utilise Area Under the ROC Curve (AUC) as a metric to measure the anomaly detection capability of the algorithms. This metric is widely used in the literature to measure the performance of anomaly detection algorithms. AUC can intuitively be interpreted as the probability of assigning a higher anomaly score to a random given anomalous data sample than a randomly given normal data sample [8]. Such a metric is bounded between 0 and 1, the higher the better the performance of the anomaly detection in detecting the anomalous data samples.

In addition, we measure the computation time (in seconds) of the anomaly detection algorithms. Obviously, lower computation time is preferable.

3.3 Experiments setups

In this study, the goal will be to compare the performance of important existing unsupervised anomaly detection algorithms such as Artificial Neural Network AutoEncoder[40], Isolation Forest [36], [37], ABOD [35], KNN [21], PCA[39], LoF[32] and HBOS [65]. These benchmark algorithms are implemented by Zhao et. al. as a Python package called pyod [66] which are utilised in this experiment.

Furthermore, the Python code developed for the benchmark evaluation is provided in Appendix A. The code is written on Google Colab [67], an online platform provided by Google, where Python code can be written and run on the cloud. As a result, the code can be efficiently run on the Google computing servers without the requirement to install and run any packages on a local machine. When we run the code on Google Colab, we just use CPU computing engine. The Python 3 Google Compute Engine backend in Google Colab has 2vCPU Intel(R) Xeon(R) @ 2.2GHz, 13 GB RAM and 62 GB Disk.

First, we shuffled data with a static seed just for the random number generator for shuffling purpose. This enables the shuffling procedure to be reproducible. Then, we scaled the data to range $[0, 1]$ by min-max approach in order to have all dimensional values in the same range. Then, we utilised a 5-fold cross validation approach [68] in which each dataset is divided into 5 parts, at each time one part (fold) is the test set and the other 4 folds are training sets. Each time an algorithm runs, the average of 5-folds are computed as a performance value for that run.

When we run the above 5-fold cross validation method on deterministic algorithms (e.g. PCA), the evaluation results do not vary from one run to another run. The reason is that the dataset is fixed (due to a static seed utilised for data shuffling and cross validation) and the algorithm's hyper-parameters are deterministic and do not change from one run to another run. However, for the probabilistic algorithms (e.g. Isolation Forest) which may have different hyper-parameters values during initialisation and run time, the 5-fold cross validation can produce different results

from one run to another. To capture and quantify this source of variation which comes from the algorithm and not from the dataset, each algorithm will further be run 10 times utilising the above-mentioned approach and at the end, the computed performance metrics for 10 runs are averaged in order to have a final value of performance metric. Such an approach also enables us to measure the standard deviation of the algorithm's performance during different run times. Idealistically, low deviation is desirable.

It is important to bear in mind that since, all algorithms are unsupervised (not requiring labelled data), by training and test sets, we refer to the learning phase for the in-samples data and computing anomaly scores for out-of-samples data. In other words, the individual algorithms do not use label information and it is only utilised for computing the performances. Generally, we prefer algorithms with high AUC and low computation time.

3.4 Results and Discussions

Figure 3.1 illustrates the AUC box-plot results on the Breast Cancer dataset. The X-axis indicates the name of each algorithm and Y-axis shows the range (distribution) as a box-plot of AUC. This range includes minimum, quartiles and the maximum value of AUC for a particular algorithm. Furthermore, the mean (μ) and standard deviation σ of each range is depicted on the right side of each box-plot. The numbers are rounded up to two decimal points. Most of the benchmark algorithms have mean AUC around 0.98 except ABOD with mean AUC = 0.9. Therefore, most of the benchmark algorithms performed well on this dataset. Furthermore, it can be observed that the standard deviation of AUC for all algorithms are near zero. This shows that when algorithms are re-run (re-initialised), they tend to have a same performance over different runs which is desirable. Algorithms such as Isolation Forest and Auto Encoder show a range performance rather than a exact performance in comparison with other deterministic algorithms such as PCA.

Figure 3.2 illustrates the computational time (measured in seconds) box-plots for bench mark algorithms. The X-axis indicates the name of each algorithm and Y-axis shows the range (distribution) as a box-plot of time. This range includes minimum,

quartiles and the maximum value of run time for a particular algorithm. As the size of this dataset is small with only 366 data samples, the computational time of most of the algorithms except Auto Encoder are low. For instance, the fastest one is PCA.

The AUC box-plots results rounded to two decimal points on the large dataset in our experiment called Credit Card Fraud Detection are shown in Figure 3.3. Since the size of this dataset is larger (with 284,807 data samples) than previous dataset, many of the algorithms failed to complete the experiment due to large required memory and computational resources, thus removed from the figure. Only Isolation forest, HBOS and PCA could survive this experiment with mean AUC = 0.95. In addition, it is observed that Isolation Forest produces a range rather than a exact value due to its probabilistic nature.

As the numbers are rounded up to two decimal points, $\sigma = 0.0$ is shown on diagrams for the algorithms. This either means the underlying algorithm is deterministic and not probabilistic (e.g., PCA) thus producing the same performances during different run times, or the computed value for σ is very small insignificant number (e.g., 0.001) which by rounding up to two decimal points equals zero. As explained previously, to capture and quantify the source of variation which comes from algorithms with probabilistic nature (e.g., isolation forest) and not from the dataset, each algorithm will be run ten times in which each individual run includes 5-folds validation evaluation (average of AUC for 5-folds). Dataset and the corresponding 5-folds are fixed during each run and the only thing that may change is the underlying algorithm if it has a probabilistic nature. At the end, these ten AUC values are utilised to compute final value of performance metric. σ shows the standard deviation of these ten different runs and not 5-folds. Such an approach enables us to measure the standard deviation of the probabilistic algorithms' performance during different run time.

Figure 3.4 shows the computational time box-plot rounded to two decimal points on the Credit Card Fraud Detection dataset. As mentioned before, only Isolation Forest, HBOS and PCA survived this test and other algorithm removed from the figure due to their run time failure. It can be observed that HBOS and PCA require lower mean and standard deviation computational time than Isolation Forest. The reason for this is that Isolation Forest is an ensemble method, which consists

of different computational trees, thus requiring more time than PCA and HBOS to perform anomaly detection task.

The results of the experiments are summarised in Table 3.1. From these results, we can conclude that Isolation Forest, HBOS and PCA are robust algorithms for the task of unsupervised anomaly detection on large data.

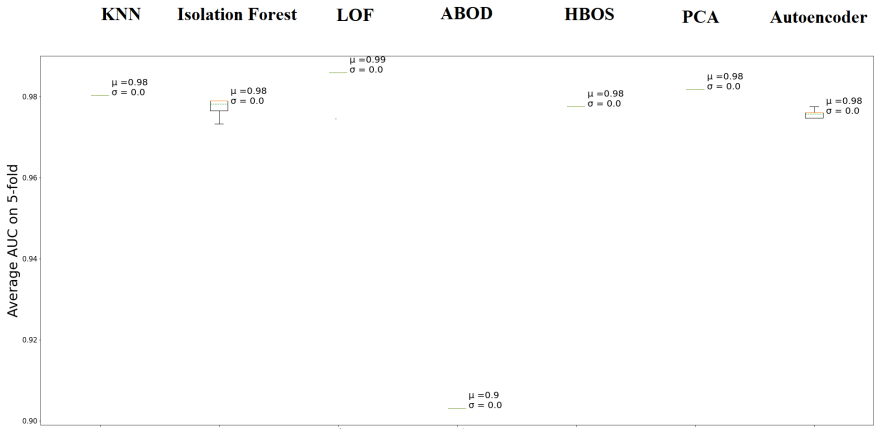

FIGURE 3.1: AUC box plot of benchmark evaluation of existing anomaly detection algorithms on the Breast Cancer dataset.

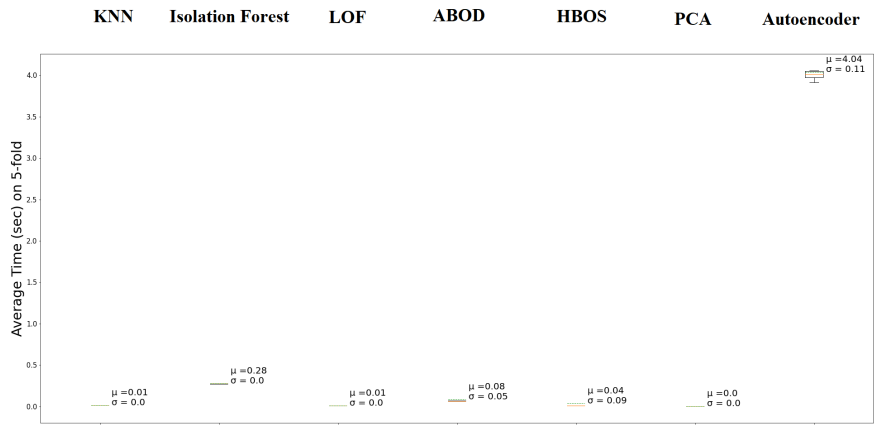

FIGURE 3.2: Computation time (measured in seconds) box plot of benchmark evaluation of existing anomaly detection algorithms on the Breast Cancer dataset.

FIGURE 3.3: AUC box plot of benchmark evaluation of existing anomaly detection algorithms on the Credit Card dataset. Those benchmark algorithms which are not shown on the figure either failed or took long time to perform thus not completed, during the experiment.

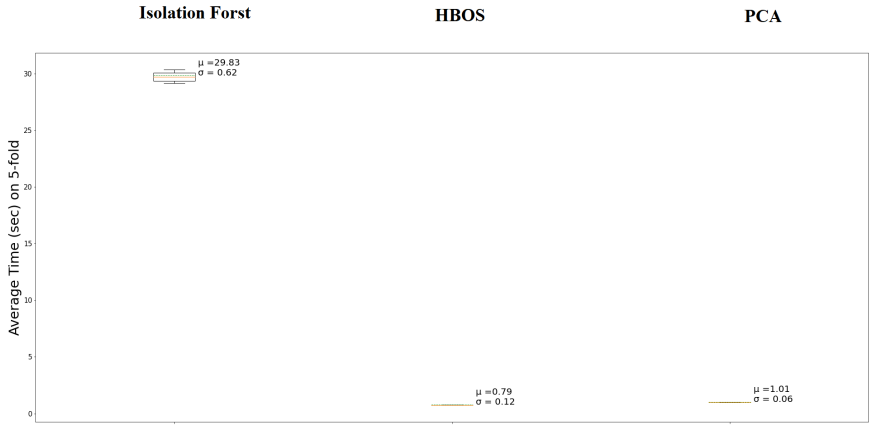

FIGURE 3.4: Computation time (measured in seconds) box plot of benchmark evaluation of existing anomaly detection algorithms on the Credit Card dataset. Those benchmark algorithms which are not shown on the figure either failed or took long time to perform thus not completed, during the experiment.

TABLE 3.1: Summary table of anomaly detection benchmark evaluation. All numbers are rounded up to two decimal points. Values with NA label (not applicable) correspond to the algorithms failure during the run time due to large computational or memory requirements.

Algorithm	Average AUC on Breast Cancer Dataset	Average Run Time on Breast Cancer Dataset	Average AUC on Credit Card Dataset	Average Run Time on Credit Card Dataset
KNN	**0.98**	**0.01**	**NA**	**NA**
Isolation Forest	**0.98**	**0.28**	**0.95**	**29.83**
LOF	**0.99**	**0.01**	**NA**	**NA**
ABOD	**0.9**	**0.08**	**NA**	**NA**
HBOS	**0.98**	**0.04**	**0.95**	**0.79**
PCA	**0.98**	**0.00**	**0.95**	**1.01**
Autoencoder	**0.98**	**4.04**	**NA**	**NA**

Chapter 4

Proposed Anomaly Detection Algorithm

Detecting anomalies in a timely manner without the requirement to label the data by humans in advance motivated proposing a novel unsupervised anomaly detection algorithm. This new algorithm which is inspired from the fact that anomalies can be isolated from the rest of data, is proposed in this chapter based on generating random shapes data independently.

Since the algorithm is unsupervised, it does not require label information to distinguish data as normal or abnormal. Furthermore, the proposed algorithm can detect anomalies rapidly which may have potential application in situations where detection of anomalies in real-time is a requirement (e.g. real-time financial fraud detection). Section 4.1 explains the general principles of the proposed algorithm. Section 4.2 includes details regarding utilisation of Minkowski distance function in the algorithm as a special case for processing of high-dimensional data. Next, the lower and upper bounds mathematical formula for the number of required random shapes in the algorithm are provided in Section 4.3, followed by some guidance regarding choosing hyper-parameters of the algorithm in Section 4.4.

4.1 Proposed Anomaly Detection Algorithm

Mathematically, consider a given D-dimensional dataset, which is split into train and test sets $\mathcal{X}$ and $\mathcal{X}^{'}$, respectively. The train set $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ includes N data instances (samples) such that, each data instance $\mathbf{x}_i = [x_{i,d}]_{d=1}^{D}$ is a D-dimensional row vector

with dimensional values $x_{i,d}$. Similarly, the test set $\mathcal{X}' = \{\mathbf{x}'_j\}_{j=1}^{N'}$ includes N' data instances (samples) such that, each data instance $\mathbf{x}'_j = [x'_{j,d}]_{d=1}^{D}$ is a D-dimensional row vector with dimensional values $x'_{j,d}$. We further assume that data are located in the unit hyper-cube I^D ($I = [0, 1]$). Such an assumption is essential for having a bounded high-dimensional space where $0 \leq x_{i,d} \leq 1$ and $0 \leq x'_{j,d} \leq 1$.

The proposed unsupervised anomaly detection algorithm (see the pseudo-code and Python code in Algorithm 1 and Appendix A.1, respectively) is intuitively based on following steps:

- Partitioning a bounded high-dimensional space (e.g. the unit hyper-cube I^D) by a sequence of random shapes, in which each data will be trapped (isolated) either inside (encoded by 1) or outside (encoded by 0). As shown in Figure 4.1 as an example, the shapes are generated completely random with different sizes and locations, independently of each other as well as data. The shapes can be expressed either by a user defined mathematical function or an algorithm (called *Draw* in the pseudo-code), which describes the boundary properties of the shapes. It should be noted that shapes must be closely bounded to divide the unit-hyper cube into inside and outside sub-spaces. More details regarding generation of random shapes are explained in Section 4.1.1.

- Such a partitioning scheme, encodes each data into a binary pattern as a sequence of 0s and 1s depending on whether the data falls outside or inside a random shape. Section 4.1.2 includes more details regarding this binary encoding.

- Next, under the fundamental assumption for all anomaly detection algorithms that anomalous data (minority data) are rare and have different characteristics, the proposed algorithm learns the binary patterns by the probabilistic modelling approach and can distinguish anomalous data whose binary patterns of trapping (inside or outside) based on the sequence of random shapes is significantly different from rest of the dataset (normal data or majority). More detail regarding how the algorithm learns from data is provided in Section 4.1.3.

- Finally, the algorithm computes the probability of observing a new test data conditioned on previously observed binary patterns. Then, this probability

is converted to an anomaly score for the data sample. This anomaly score, which is a real-valued number indicates the degree to which the data sample is anomalous (See section 4.1.4 for more details). Therefore, test data samples can be ranked in a list via their corresponding anomaly scores, where anomalies will be ranked on top of the list, while normal data samples will be assigned low anomaly scores, thus ranked lower.

In fact, normal data points which are located closer to each other, are more likely to be trapped by the same random shapes, thus having similar binary patterns, while anomalous data will be trapped by different shapes and their binary patterns will be different from the majority observed patterns. This concept is shown by an example in Figure 4.1 in which, five random shapes (in green) are generated. Normal data (in blue circles) and abnormal data (in red stars) are trapped by different random shapes, respectively.

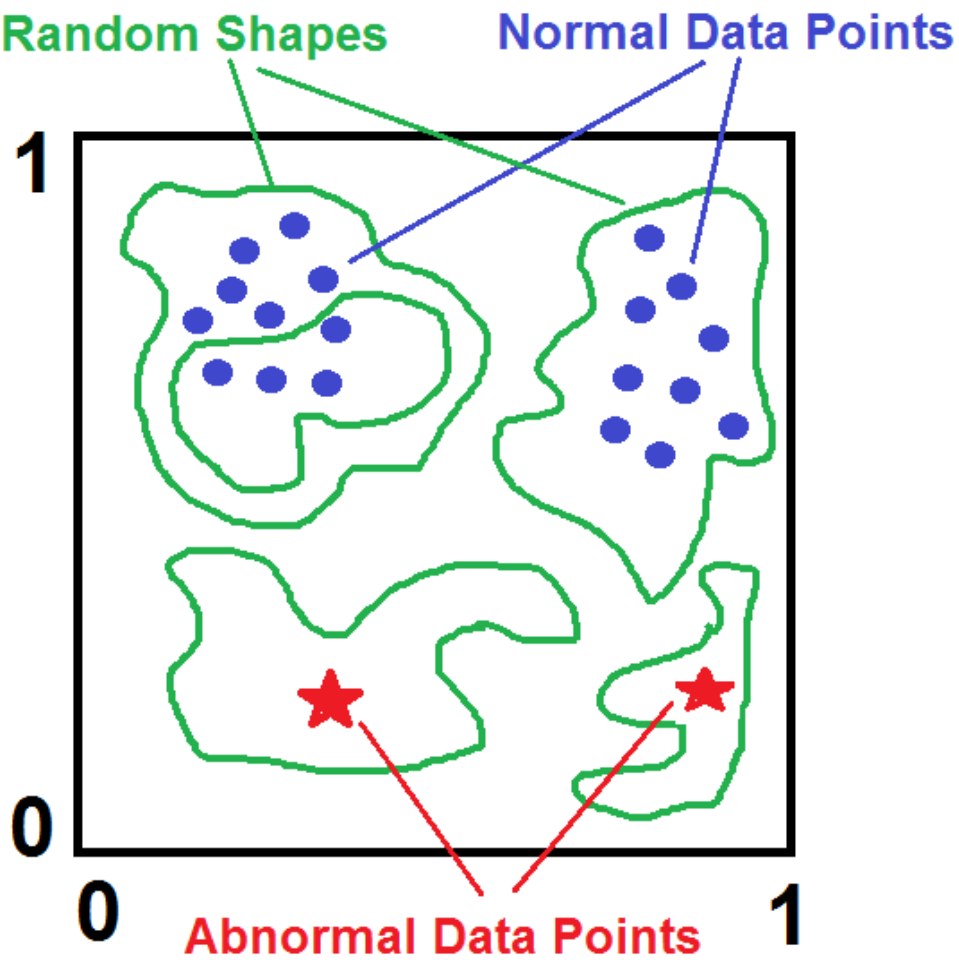

FIGURE 4.1: Illustration of five random shapes in green, normal data in blue circles and abnormal data in red stars. Data points which are closer to each other are more likely to be trapped by the same random shape.

Algorithm 1 General pseudo code of the proposed anomaly detection algorithm.

Require: Training dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$, in which, each row $\mathbf{x}_i = [x_{i,d}]_{d=1}^{D} = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ represents a training data sample in a D-dimensional unit hyper-cube such that $0 \leq x_{i,d} \leq 1$.

Require: Test dataset $\mathcal{X}^{'} = \{\mathbf{x}^{'}_j\}_{j=1}^{N^{'}}$ in which, each row $\mathbf{x}^{'}_j = [x^{'}_{j,d}]_{d=1}^{D} = (x^{'}_{j,1}, x^{'}_{j,2}, \ldots, x^{'}_{j,D})$ represents a test data sample (for which anomaly score will be computed) in a D-dimensional unit hyper-cube such that $0 \leq x^{'}_{j,d} \leq 1$.

Require: Number of Random Shapes H (user-defined)

Require: User-defined function/algorithm called *Draw* to draw (generate) a random shape (closely bounded sub-space) in a D-dimensional space, which divides the unit hyper-cube into two sub-spaces: inside and outside.

Initialise *AnomalyScoreVector* as a vector with size $N^{'}$ in which, the j-th element corresponds to the anomaly score for test sample $\mathbf{x}^{'}_j$.
Based on *Draw*, generate H random shapes with different random size and locations in the unit hyper-cube and store them in memory as set $\mathcal{S} = \{\mathbf{s}_h\}_{h=1}^{H}$.
Initialise vector $A = [\alpha_1, \alpha_2, \ldots, \alpha_H] = [\alpha_h]_{h=1}^{H}$ with size H in which, all $\alpha_h = 0$.
for $i = 1$ **to** N **do**
  for $h = 1$ **to** H **do**
    if $\mathbf{x}_i$ is inside the shape $\mathbf{s}_h$ **then**
      $\alpha_h \leftarrow 1 + \alpha_h$
    end if
  end for
end for
for $j = 1$ **to** $N^{'}$ **do**
  Initialise $\theta = 1$
  for $h = 1$ **to** H **do**
    if $\mathbf{x}^{'}_j$ is inside the shape $\mathbf{s}_h$ **then**
      $\theta \leftarrow \theta * (1 + \alpha_h)$
    else
      $\theta \leftarrow \theta * (1 + N - \alpha_h)$
    end if
  end for
  $AnomalyScoreVector[j] \leftarrow -Ln(\theta) + H * Ln(2 + N)$
end for
return *AnomalyScoreVector*

4.1.1 Random Shapes Generation

In the first step of the proposed algorithm, a set of random shapes $\mathcal{S} = \{\mathbf{s}_h\}_{h=1}^{H}$ is generated in random locations of the hyper-cube, independently of the data. H is a user-defined parameter, which determines the number of the random shapes. Each random shape $\mathbf{s}_h$ has a topology, which divides the hyper-cube into two sub-spaces: inside and outside. Figure 4.2 illustrates an example of three random shapes $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$, which are generated in the unit square (2D space) independent both of the data and the previously-generated random shapes. From a very general point of view, there are no restrictions on the shapes' forms (topological properties) and some shapes can even have shared sub-spaces with others, such as $\mathbf{s}_2$ and $\mathbf{s}_3$ in the Figure 4.2.

Once the shapes are created, these are fixed through the rest of the algorithm and will be utilised for partitioning the space rapidly, which is essential for trapping anomalies in sub-partitions of the space, where there will be less likelihood for a normal data instance to fall in the same sub-spaces, where anomalies are located. In addition, this novel approach enables the algorithm to trap anomalies in a single or few shots, which is desirable from the computational point of view as shown in the Figure 4.1

Since the proposed algorithm is for general purpose, there is no restriction on the types of the random shapes to be generated. In fact, the shapes can be symmetric (e.g. square, circle and etc), asymmetric (e.g. shapes generated by a user-defined kernel function based on a specific application) or combination of both. The random nature of shapes may enable the algorithm to partition the space efficiently and data independently. However, this advantage may come with extra computational costs for generating the random shapes.

Although, the random shapes can have any shape from a theoretical point of view, for simplicity and efficiency of further computations, we assume that $\mathbf{s}_h = (\mathbf{c}_h, r_h)$ is a hyper-sphere (see an illustrative example in the Figure 4.3) with center $\mathbf{c}_h \sim Uniform(0,1)$ and radius $r_h \sim Uniform(Min_r, Max_r)$ such that, $Uniform(a,b)$ is a uniform distribution, which generates random numbers between a and b. Min_r and Max_r are user-defined parameters, which determine the range for minimum

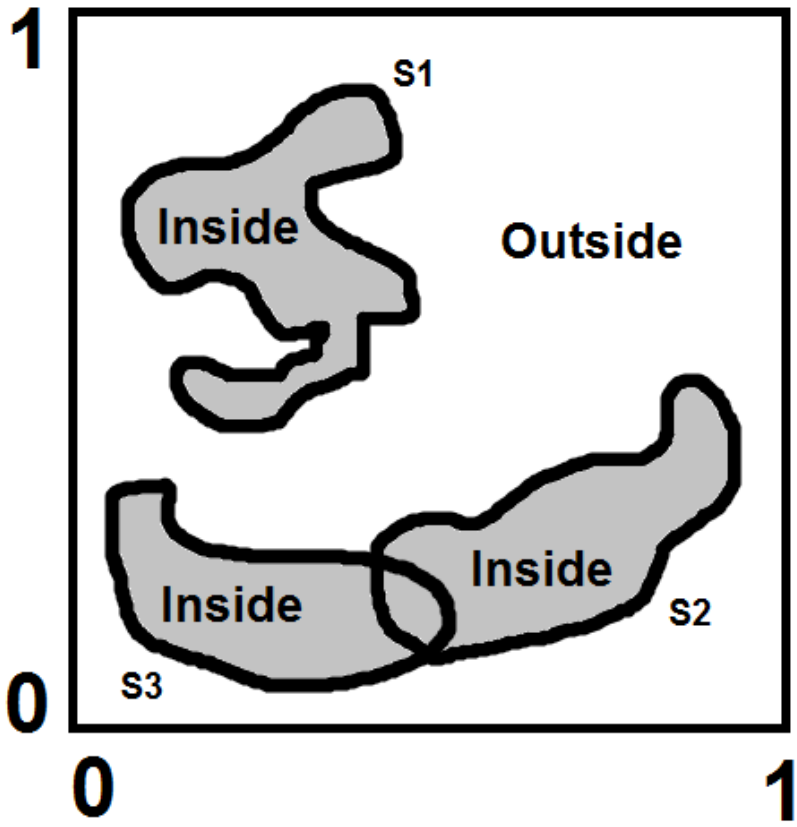

FIGURE 4.2: Three random shapes $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$ in the unit square (2D space). Each random shape (topological object) partitions the space into Inside and Outside sub-spaces.

and maximum size of the random shapes that can be generated during this process, respectively.

In Section 4.1.2, we explain the second step of the proposed algorithm on how to utilise the above partitioning scheme after generating the random shapes.

4.1.2 Binary Encoding

During the initialisation of the algorithm, H random shapes are generated and numbered sequentially and stored in memory as set $\mathcal{S} = \{\mathbf{s}_h\}_{h=1}^{H}$. The orders of random shapes are preserved during the training and testing as explained in Algorithm 1.

Next, binary encoding transforms real-value data features into sequences of 0s and 1s, which will be efficient for data modelling. As explained in Section 4.1.1, each random shape $\mathbf{s}_h$ divides the region in the hyper-cube into two sub-regions: inside and outside of the shape. In practice, whether a data point lies within a shape will be defined as a function of the distance of the datum from the centre of the shape. Mathematically, we define

$$Hash(\mathbf{x}_i, \mathbf{s}_h) = \left\{ \begin{array}{ll} 0 & ; Distance(\mathbf{x}_i, \mathbf{c}_h) \geq r_h \\ 1 & ; Distance(\mathbf{x}_i, \mathbf{c}_h) < r_h \end{array} \right\}, \tag{4.1}$$

as a binary hash function with an output 1 or 0, whether a train data instance $\mathbf{x}_i$ falls inside or outside of a random shape $\mathbf{s}_h$, respectively. In Equation 4.1, $Distance(\mathbf{x}_i, \mathbf{c}_h)$ can be any distance metric, which computes the distance between a data instance $\mathbf{x}_i$ and the associated centre $\mathbf{c}_h$ of the random shape $\mathbf{s}_h$ (see Section 4.2 for more details). Similarly, replacing $\mathbf{x}_i$ with $\mathbf{x}'_j$ in Equation 4.1 results in a new formula of the hash function $Hash(\mathbf{x}'_j, \mathbf{s}_h)$ for a test data sample $\mathbf{x}'_j$.

In the second step, the algorithm applies the above encoding hash function on each train and test data instances $\mathbf{x}_i; i \in \{1, 2, 3, \ldots, N\}$ and $\mathbf{x}'_j; j \in \{1, 2, 3, \ldots, N'\}$, respectively with respect to the set of random shapes $\mathcal{S}$ to convert the real-value features (dimensions) to binary using equation 4.1. This generates binary pattern $Pattern(\mathbf{x}_i)$ as shown in the Equation 4.2. Similarly, replacing $\mathbf{x}_i$ with $\mathbf{x}'_j$ in this equation results in a new equation of $Pattern(\mathbf{x}'_j)$ for test data sample $\mathbf{x}'_j$. Figure 4.3 demonstrates this concept for three example data points in the 2D space.

$$Pattern(\mathbf{x}_i) = [Hash(\mathbf{x}_i, \mathbf{s}_1), Hash(\mathbf{x}_i, \mathbf{s}_2), \ldots, Hash(\mathbf{x}_i, \mathbf{s}_H)]. \tag{4.2}$$

The rationale for this encoding scheme is that modeling the binary patterns of data is mathematically and computationally more efficient than directly modelling the probability distribution of original dataset in the high-dimensional space, as we

will show. Furthermore, we can benefit from Bayesian conjugate properties which facilitates modelling the data in an online fashion and has application in rapid data analysis. This will be explained in more details in Section 4.1.3.

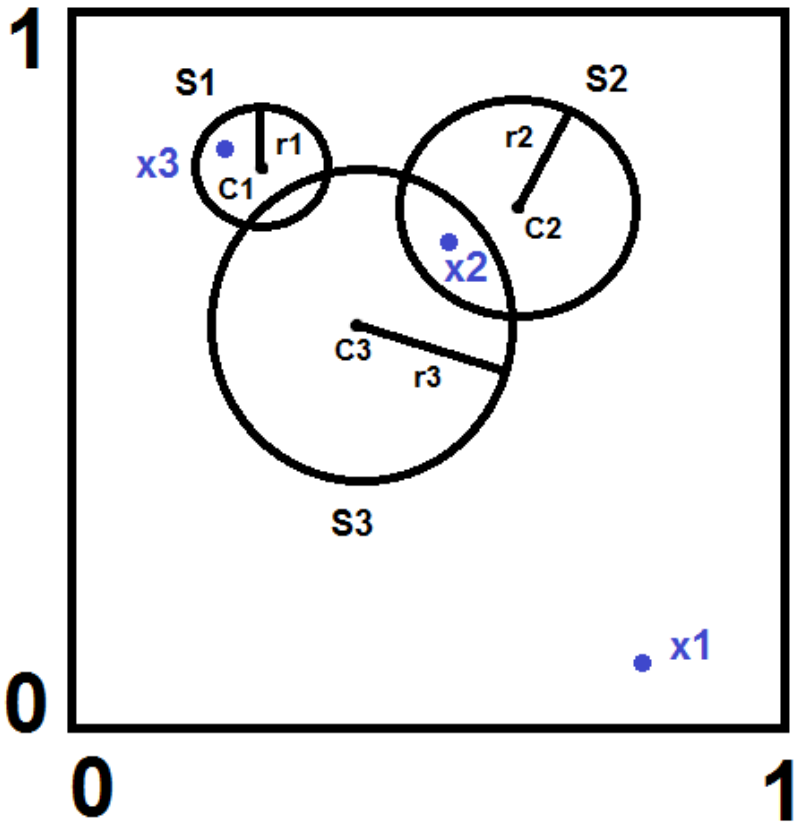

FIGURE 4.3: Three random shapes (circles) $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$ in the unit square (2D space). c and r represents the center and radius of each circle. Data points are $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ with patterns 000, 011 and 100, respectively.

4.1.3 Bayesian Incremental Updating

Bayesian Incremental Updating provides a robust mathematical framework and computationally efficient mechanism for probabilistic modelling of the binary patterns, which is suitable for modelling data rapidly. Such a model is essential to distinguish normal and abnormal patterns from each other.

In the third step, the algorithm computes the probability of observing a new pattern $Pattern(\mathbf{x}^{'}_{j})$ of a test data sample conditioned on previously observed patterns displayed by the training data. The intuition behind this is based on an assumption that normal data have frequent and similar patterns to those observed in the training set, while abnormal patterns are significantly different from normal ones. Thus, a high or low estimate of a specific pattern's probability corresponds, respectively to lower or higher degree of anomalousness of the test data instance.

In what follows, we describe how the algorithm utilises incremental Bayesian modeling approach to compute this probability efficiently.

Mathematically, the posterior probability $\mathcal{P}$ of observing a binary pattern conditioned on previously observed data and the sequence of random shapes defined in the training phase can be written as product of probabilities thus:

$$\mathcal{P}(Pattern(\mathbf{x}^{'}_{j})|previousPatterns) = \prod_{h=1}^{H} \tau_h(\mathbf{x}^{'}_{j}). \tag{4.3}$$

In Equation 4.3, $\tau_h(\mathbf{x}^{'}_{j})$ is defined as

$$\tau_h(\mathbf{x}^{'}_{j}) = \left\{ \begin{array}{cc} p_h & ; Hash(\mathbf{x}^{'}_{j}, \mathbf{s}_h) = 1 \\ 1 - p_h & ; Hash(\mathbf{x}^{'}_{j}, \mathbf{s}_h) = 0 \end{array} \right\}, \tag{4.4}$$

in which, p_h and $1 - p_h$ correspond to the posterior probability of observing 1 and 0 for the random shape $\mathbf{s}_h$, respectively. In other words, these probabilities model the binary hashes by computing the chance of a new test data instance to be trapped inside or outside of a particular random shape.

By utilising the Baye's rule, we assume a prior Beta distribution $Beta(1,1)$ for p_h and a categorical likelihood, for generating 1 and 0 to model these binary hashes. The reason for choosing the beta distribution as our prior is that it generates a random number between 0 and 1, which can be interpreted as the probability of a test

data being inside a random shape. Furthermore, such a prior distribution is conjugate with categorical likelihood. Therefore, the posterior distribution p_h will be again a Beta distribution and only the parameters needs to be updated incrementally and stored in the computer memory. This Bayesian conjugacy is beneficial from the computational point of view as shown in:

$$p_h \sim Beta(1 + \alpha_h, 1 + N - \alpha_h). \tag{4.5}$$

α_h in Equation 4.5 determines the number of the observed training data points, which fall inside the random shape $\mathbf{s}_h$. This hyper parameter will be calculated for each random shape in $\mathcal{S}$, thus the algorithm generates a hyper-parameter vector A as

$$A = [\alpha_1, \alpha_2, \ldots, \alpha_H], \tag{4.6}$$

which is updated incrementally as described above by observing training data points. This online updating mechanism enables the algorithm efficiently to perform anomaly detection rapidly.

From Equation 4.5, the expected probabilities of a test data point falling respectively inside or outside of the h-th random shape can be derived

$$E(p_h) = \frac{1 + \alpha_h}{2 + N}. \tag{4.7}$$

$$E(1 - p_h) = \frac{1 + N - \alpha_h}{2 + N}. \tag{4.8}$$

By defining

$$\theta_h(\mathbf{x}'_j) = \left\{ \begin{array}{ll} 1 + \alpha_h & ; Hash(\mathbf{x}'_j, \mathbf{s}_h) = 1 \\ 1 + N - \alpha_h & ; Hash(\mathbf{x}'_j, \mathbf{s}_h) = 0 \end{array} \right\}, \tag{4.9}$$

the expected value of the posterior probability $\mathcal{P}$ as shown in Equation 4.3 can be written

$$
\begin{aligned}
E(\mathcal{P}(Pattern(\mathbf{x}'_j)|previousPatterns)) &= E(\prod_{h=1}^{H} \tau_h(\mathbf{x}'_j)) \\
&= \prod_{h=1}^{H} E(\tau_h(\mathbf{x}'_j)) \\
&= \frac{\prod_{h=1}^{H} \theta_h(\mathbf{x}'_j)}{(2+N)^H}.
\end{aligned} \tag{4.10}
$$

Equation 4.10 is a fundamental equation in this algorithm, which calculates the expected probability of observing a particular pattern. The lower is this number, the more anomalous is the associated data point. This probability will be converted to an anomaly score, as explained in the next Section 4.1.4.

4.1.4 Computing Anomaly Score

The anomaly score is a real-valued number which demonstrates the degree of abnormality of a given data point. Such a score has application in ranking data points in a list based on their associates anomaly score. Those data which appear on top of the list are more anomalous than the others. This will ensure that relatively abnormal data points that is, those with low probability of binary patterns, have a high corresponding anomaly score. Such a scaling enables the algorithm to distinguish normal from abnormal data efficiently. Furthermore, if such a real-valued number will be greater than a user-specific threshold (depending on the application and dataset), the corresponding data point can be declared as an anomaly.

In the final step of the anomaly detection algorithm, expected probability $E(\mathcal{P})$ (see Equation 4.10) is converted to an anomaly score via Equation 4.11.

$$
\begin{aligned}
AnomalyScore(\mathbf{x}'_j) &= -Ln(\frac{\prod_{h=1}^{H} \theta_h(\mathbf{x}'_j)}{(2+N)^H}) \\
&= -Ln(\prod_{h=1}^{H} \theta_h(\mathbf{x}'_j)) + Ln((2+N)^H) \\
&= -\sum_{h=1}^{H} Ln(\theta_h(\mathbf{x}'_j)) + H.Ln(2+N),
\end{aligned} \tag{4.11}
$$

in which, Ln is the natural logarithm function. The reason for utilizing $-Ln$ in the above equation is to map probability values in range $[0, 1]$ to positive real numbers as

$$0 < AnomalyScore(\mathbf{x}^{'}_{j}) < +\infty. \tag{4.12}$$

4.2 Utilising Minkowski Distance Function in the Algorithm as a Special Case

Although, the proposed general algorithm can work with any distance function corresponding to different shapes based on the principles explained in Section 4.1, for the purpose of practical implementation of the algorithm, we choose $Distance(\mathbf{x_i}, \mathbf{c}_h)$ in Equation 4.1 to be the general family of Minkowski distance functions (l_L norms) as shown in Equation 4.13.

$$Distance(\mathbf{x_i}, \mathbf{c}_h) = (\sum_{d=1}^{D} |x_{i,d} - c_{h,d}|^L)^{\frac{1}{L}}. \tag{4.13}$$

In Equation 4.13, $L > 0$ (corresponding to different shapes) and $\mathbf{c}_h = [c_{h,d}]_{d=1}^{D}$ is a D-dimensional vector with dimensional values $c_{h,d}$. Similarly, replacing $\mathbf{x}_i$ with $\mathbf{x}^{'}_{j}$ in this equation results in a new equation of $Distance(\mathbf{x^{'}_j}, \mathbf{c}_h)$ for test data sample $\mathbf{x}^{'}_{j}$.

Choosing different L in Equation 4.13 results in changing the shapes as follows:

- $L = 1$: Manhattan distance
- $L = 2$: Euclidean distance
- $L = \infty$: Chebyshev distance

Figure 4.4 illustrates how different distance metrics make different shapes in a two-dimensional unit square. Each shape includes all the points in the space with distance equal to 1 from the origin 0. As L increases, the corresponding shape reaches towards the boundaries of the unit square. The computational efficiency of such distances can be one of the major reasons for choosing these distances.

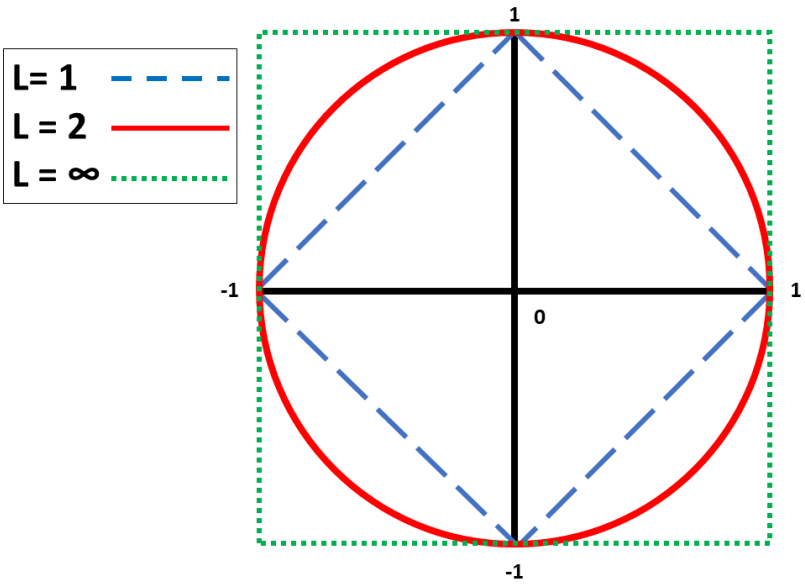

FIGURE 4.4: Different shapes corresponding to Manhattan ($L = 1$), Euclidean ($L = 2$) and Chebyshev ($L = \infty$) distance.

4.3 Lower and Upper Bounds for the Number of Required Random Shapes

This section explains the theoretical upper and lower bounds for the number of required random shapes H, an important user-defined parameter of the algorithm which is necessary in terms of algorithm efficiency and computing complexity.

We denote v_h as volume of the random shape $\mathbf{s}_h$ and define $v_h^{'}$ as portion of v_h, which falls inside the unit hyper-cube. The reason for that is some portion of a shape may fall outside the unit hype-cube, thus it should be accounted in our formulation. An example of this is shown in Figure 4.5, in which some shapes (circles) are not completely inside the unit-square thus, the fraction of their volumes inside the square are only considered.

Consider that the fractional volume of the first random shape is $v_1^{'}$. We define $\mathcal{V}_H$ as cumulative volume of the shapes, which occupy inside the unit hyper-cube (with total volume one), after generating H random shapes. Therefore, in the beginning

$$\mathcal{V}_1 = v_1^{'}. \tag{4.14}$$

When the second shape is generated,

$$\begin{aligned} \mathcal{V}_2 &= \mathcal{V}_1 + v_2^{'}(1-\mathcal{V}_1) \\ &= v_1^{'} + v_2^{'}(1-v_1^{'}), \end{aligned} \tag{4.15}$$

In which, $v_2^{'}(1-\mathcal{V}_1)$ is the contribution of the second random shape by adding a new uncovered space to the previous covered space $\mathcal{V}_1$.

Generation of the third shape results in

$$\begin{aligned} \mathcal{V}_3 &= \mathcal{V}_2 + v_3^{'}(1-\mathcal{V}_2) \\ &= v_1^{'} + v_2^{'}(1-v_1^{'}) + v_3^{'}(1-v_1^{'}-v_2^{'}(1-v_1^{'})), \end{aligned} \tag{4.16}$$

In which, $v_3^{'}(1-\mathcal{V}_2)$ is the contribution of the third random shape by adding a new uncovered space to the previous covered space $\mathcal{V}_2$.

Therefore, the general recursive formula after generating H shapes will be

$$\mathcal{V}_H = \mathcal{V}_{H-1} + v_H^{'}(1-\mathcal{V}_{H-1}). \tag{4.17}$$

In order to extract a theoretical lower bound for H based on Equation 4.17, we assume

$$v_h^{'} = v^{'} = E(v_h^{'}); h \in \{1,2,3,\ldots,H\}. \tag{4.18}$$

such a simplification means that all shapes have same volume $v^{'}$ equal to the expectation (mean) random variables $v_h^{'}$, which enables us to simplify Equation 4.17 to a geometric series as

$$\begin{aligned} \mathcal{V}_H &= v^{'}\sum_{h=1}^{H}(1-v^{'})^{h-1} \\ &= v^{'}(\frac{1-(1-v^{'})^{H}}{1-(1-v^{'})}) \\ &= 1-(1-v^{'})^{H}. \end{aligned} \tag{4.19}$$

In Equation 4.19, $0 \leq v^{'} \leq 1$, so $0 \leq 1 - v^{'} \leq 1$. In case the number of random shapes $H \rightarrow \infty$, we have

$$\lim_{H \rightarrow \infty} \mathcal{V}_H = 1, \tag{4.20}$$

which proves the upper limit of H is infinity (as shown in inequality 4.21). This is equivalent to covering the whole space, which is desirable from theoretical point of view. Note that, as $H \rightarrow \infty$, the performance of the anomaly detection algorithm will be improved. The reason is that as $H \rightarrow \infty$, the probability of having sub-spaces which are outside all random shapes decreases. This phenomenon is illustrated in Figure 4.5. The red area in the figure is the sub-space which is not covered by any random shape. The lesser is the "uncovered" space, the greater is the performance of the algorithm because data points (whether they are anomaly or normal), which fall in this uncovered region, will be encoded to the same binary pattern, specifically $000\ldots0$, thus the algorithm can not differentiate them and may confuse normal and abnormal data with each other.

Such a situation of uncovered sub-spaces must be avoided by choosing a large value for H as much as the computational resources allow. Therefore, there is no theoretical upper bound for the number of required random shapes, that

$$H < \infty. \tag{4.21}$$

However, in practice, we cannot generate an infinite number of shapes. The minimum number of the required random shapes (lower bound for H) is essential from a computational point of view as, if H is chosen mistakenly below this lower bound, there will be a high probability of having uncovered sub-spaces leading to inaccurate results. In what follows, the mathematical descriptions of this problem is provided.

As we require the shapes to cover the whole space (volume of the unit hype-cube) as much as possible thus avoiding uncovered spaces as shown in Figure 4.5, mathematically we quantify this as a user-defined parameter $0 \leq CoveredVolume \leq 1$, which is the minimum desired fraction of the unit hype-cube's volume to be covered by the shapes. Therefore, Equation 4.19 turns to

$$
\begin{aligned}
CoveredVolume &\leq 1 - (1 - v^{'})^{H} \\
\implies (1 - v^{'})^{H} &\leq 1 - CoveredVolume \\
\implies Ln((1 - v^{'})^{H}) &\leq Ln(1 - CoveredVolume) \\
\implies H.Ln(1 - v^{'}) &\leq Ln(1 - CoveredVolume) \\
\implies \frac{Ln(1 - CoveredVolume)}{Ln(1 - v^{'})} &\leq H,
\end{aligned}
\tag{4.22}
$$

since $Ln(1 - v^{'}) \leq 0$.

Equation 4.22 determines the lower bound for the number of required random shapes. From this Equation, we can infer two important facts:

1. As $CoveredVolume$ increases and becomes closer to 1, the lower bound for H increases, more shapes are required to cover the desired volume, requiring more computational resources.

2. Alternatively, as the volume of each random shape increases, the fraction of its volume which falls inside the cube represented by $v^{'}$ increases, thus quantity $1 - v^{'}$ decreases, leading to less required number of shapes. However, there are some constraints on the optimal volume of shapes which will be discussed in more detail in Section 4.4.

This section provided the upper and lower bounds for H (number of random shapes). These theoretical bounds can be utilised as a general guidance for setting the required number of random shapes.

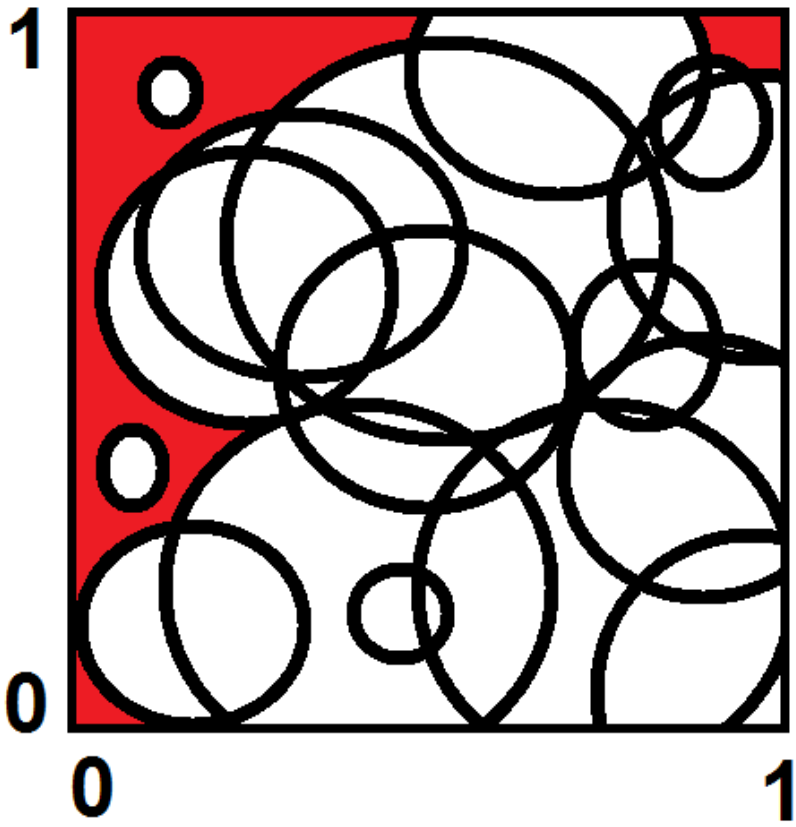

FIGURE 4.5: Illustration of the unfilled sub-spaces (in red color) as a result of insufficient number of shapes, which do not cover the whole space. All data instances which fall inside of these unfilled sub-spaces, have same binary pattern $0000\ldots0$ and may introduce bias in the anomaly detection performance.

4.4 Optimal Volume of Random Shapes

As we have seen, if the volume of random shapes v_h is too big or too small, the proposed algorithm (see Section 4.1) may not work properly, due to the fact that majority of data instances would fall inside or outside of all random shapes, respectively, thus having a same binary pattern. This will reduce the effectiveness of the algorithm in classifying data. As a result, there is a trade off between the random shapes being too large and too small random shapes. This will be discussed in more detail in this section.

Since the volume of each random shape v_h is directly affected by the radius $r_h \sim Uniform(Min_r, Max_r)$ (see Section 4.1.1 for more explanation regarding how the random shapes are generated), choosing appropriate Min_r and Max_r is crucial for the algorithm to work effectively and efficiently.

Clearly, the distance between any two data points a, b located in the D-dimensional unit hyper-cube cannot exceed the largest diagonal length corresponding to the distance between points $(0, 0, 0, \ldots, 0)$ and $(1, 1, 1, \ldots, 1)$. Since we choose L-norms for the underlying distance function, (see section 4.2) the maximum diagonal length is $D^{1/L}$ as shown in Equation 4.23.

$$Distance(a, b) \leq LengthDiagonal = D^{1/L}. \tag{4.23}$$

Therefore, the upper bound for r_h can be found by

$$r_h \leq D^{1/L}. \tag{4.24}$$

Allowing a larger radius than Equation 4.24 results in large volumes, which decrease the performance of the algorithm. Therefore, it is recommended to choose Min_r and Max_r below this upper bound. For instance, portions of this maximum length such as

$$Max_r = \frac{D^{1/L}}{2}, \tag{4.25}$$

and

$$Min_r = \frac{D^{1/L}}{50}, \tag{4.26}$$

can be suggested as a general guidance regarding choosing these hyper-parameters (the empirical justifications are provided in Chapter 5). This corresponds to a range between 1/50 and half of the maximum length. For instance, half of the maximum length can be associated with a hyper-sphere located in the center of the hyper-cube while its surface passes the vertices of the hyper-cube. 1/50 of the maximum length as a lower bound for radius of hyper-spheres may prevent an undesirable situation where some shapes have radius near to zero, thus all data will fall outside of these very small shapes, which waste computational and memory resources.

However, these general formulas and fractions are for recommendation only and these need to be adjusted for each real-world dataset and application to find the optimal hyper-parameters.

In the next chapter, we test the sensitivity of the proposed algorithm under different hyper-parameters settings to see the effect of these parameters on the performance of the algorithm.

Chapter 5

Sensitivity Analysis of the Proposed Anomaly Detection Algorithm

In this chapter, we utilise the publicly available labelled datasets such as Breast Cancer and Credit Card Fraud Detection datasets with 30 dimensions ($D = 30$) as mentioned in Chapter 3 for investigating the sensitivity of the proposed algorithm (see Chapter 4) on its hyper-parameters.

In Section 5.1, the research questions regarding the sensitivity analysis are mentioned. Since the proposed algorithm is a probabilistic algorithm, we want to test, how its performance varies, if we change the hyper-parameters. Then, Section 5.2 explains the evaluation metrics utilised for performance measurement of the algorithm. Section 5.3 describes different experiment settings/designs for sensitivity analysis purposes. Finally, the results and discussions of the sensitivity analysis are provided in Section 5.4.

5.1 Research Questions

Particularly, we are interested to investigate the following research questions:

- How does the shape of random shapes (corresponding to utilising different distance functions, e.g. different l_L norms) affect the performance of the algorithm?

- What is the empirical optimal number of random shapes (H)? In other words, how many random shapes are required to be generated in order to have a stable performance?
- How do the upper and lower bounds for the size of random shapes (corresponding to Max_r and Min_r, respectively) affect the performance of the algorithm?

5.2 Evaluation Metrics

We utilise the same performance metrics as discussed in Section 3.2. Generally, higher mean AUC, lower standard deviation of AUC, lower mean and standard deviation of computation time are desirable.

5.3 Experiments setups

In order to answer the above questions, we design the following sensitivity analysis experiments. The aim of each experiment is to understand the relationship between the sensitivity of the algorithm's performances and shape, number and size of the random shapes.

We utilise the same settings mentioned in Section 3.3 which is scaling the data in range $[0, 1]$ and using 5-fold cross validation method for splitting the data into train and test sets. In addition, in the following experiments, we run the proposed algorithm with different setups corresponding to varying one hyper-parameter and fixing the rest hyper-parameters to see the effect of the varying parameter on the algorithm's performance.

5.3.1 Experiment 1: Investigation of Shape of Random Shapes

The purpose of this experiment is to investigate how the shape of random shapes determined by utilising different distance measures, affects the performance of the proposed algorithm. As mentioned in Section 4, the shape of random shape is directly related to the underlying distance function utilised in the algorithm. In other

words, changing the distance function corresponds to changing the shape (topology) of random shapes, thus changing the detection ability of the algorithm. Although, there are many distance functions, here we focus on Minkowski family l_L with $L = \{1(Manhattan), 2(Euclidean), \infty(Chebyshev)\}$ to show the concept, as these distances are computationally very efficient. Chebyshev distance is a type of Minkowski distance function when $L = \infty$ [69].

In addition, the minimum radius of random shapes (Min_r) and maximum radius of random shapes (Max_r) control the size of the shapes.

Therefore, we choose the combination settings from following sets and compute the performance metrics for each set.

For Manhattan and Euclidean distances, we choose the hyper-parameters from the combination of the following sets (since the range of distance values for any given two points in the unit hyper-cube is in range $[0, D^{1/L}]$, we adjust Min_r and Max_r as portions of this range):

1. H= $\{25, 100\}$
2. Min_r= $\{0, \frac{D^{1/L}}{50}, \frac{D^{1/L}}{10}\}$
3. Max_r= $\{\frac{D^{1/L}}{5}, \frac{D^{1/L}}{2}, \frac{D^{1/L}}{1}\}$
4. L= $\{1(Manhattan), 2(Euclidean)\}$

For Chebyshev distance, we choose the hyper-parameters from the combination of the following sets (since the range of distance values for any given two points in the unit hyper-cube is in range $[0, 1]$, we adjust Min_r and Max_r as portions of this range):

1. H= $\{25, 100\}$
2. Min_r= $\{0, 0.25, 0.5\}$
3. Max_r= $\{0.5, 0.75, 1\}$
4. L= $\{\infty(Chebyshev)\}$

5.3.2 Experiment 2: Investigation of Number of Random Shapes

In the second sensitivity analysis experiment, we are interested in exploring the effect of number of random shapes (generated by utilising different distance metrics) on the performance of the algorithm. Therefore, we fix the $Min_r = \frac{D^{1/L}}{50}$ and $Max_r = \frac{D^{1/L}}{2}$ (rounding the numbers to 2 decimal points) for different $L = \{1, 2\}$. For $L = \infty$, we fix $Min_r = 0.5$ and $Max_r = 1$ and increase number of the shapes from 5 to 5000 ($H = \{5, 10, 25, 50, 100, 500, 1000, 2500, 5000\}$) to see how the H affects the performance and computational time of the proposed algorithm.

5.4 Results and Discussions

The results and discussions of the above sensitivity analysis experiments are explained individually below:

5.4.1 Results and Discussions Experiment 1:

In the first sensitivity analysis experiment, we changed the shape of random shapes by utilising different distance metrics to see how it affects the performance of the algorithm.

Figure 5.1 illustrates the AUC box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $H = 25, L = 1$ and changing Max_r and Min_r). The Y-axis shows the range (distribution) as a box-plot of AUC. This range includes minimum, quartiles and the maximum value of AUC for each particular parameter setting. Furthermore, the mean (μ) and standard deviation σ of each range is depicted on the right side of each box-plot. It can be observed that $Max_r = 15 = \frac{30^1}{2}$ corresponding to half of the maximum length diagonal in the 30-dimensional unit hyper-cube, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.2 shows the AUC box plots for parameters settings ($H = 100, L = 1$ and changing Max_r and Min_r) on the Breast Cancer dataset. It can be observed than the mean of AUC is increased, while standard deviation is decreased

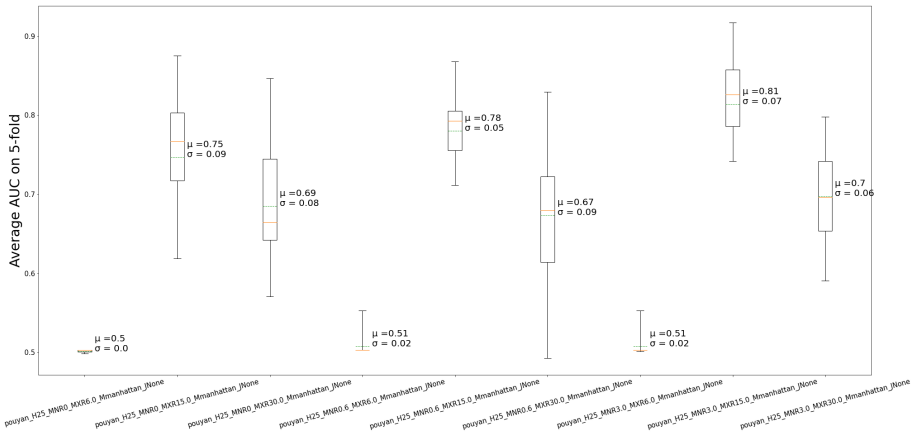

FIGURE 5.1: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = 1$) on the Breast Cancer dataset.

in comparison with Figure 5.1. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

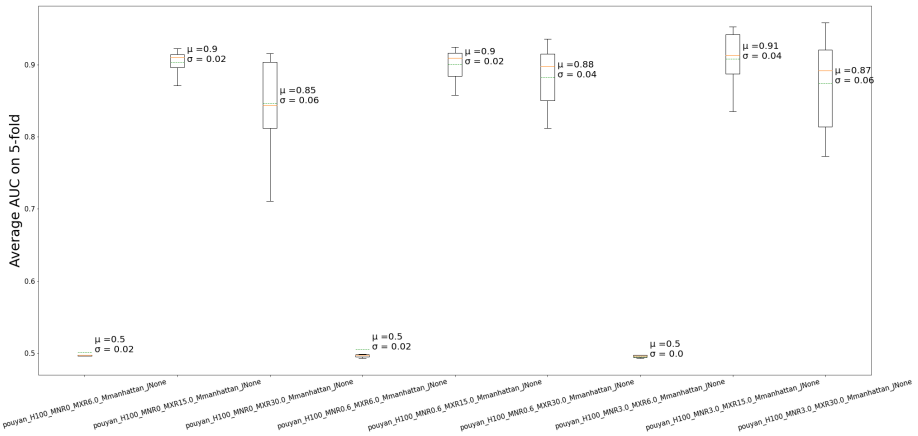

FIGURE 5.2: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = 1$) on the Breast Cancer dataset.

Figure 5.3 illustrates the AUC box-plot results on the Breast Cancer dataset, fixing $H = 25, L = 2$ and changing Max_r and Min_r. It can be observed that $Max_r = 2.74$ corresponding to half of the maximum length diagonal in the 30-dimensional unit hyper-cube, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.4 shows the AUC box plots for parameters settings ($H = 100, L = 2$ and changing Max_r and Min_r) on the Breast Cancer dataset. It can be observed that the mean of AUC is increased, while standard deviation is decreased

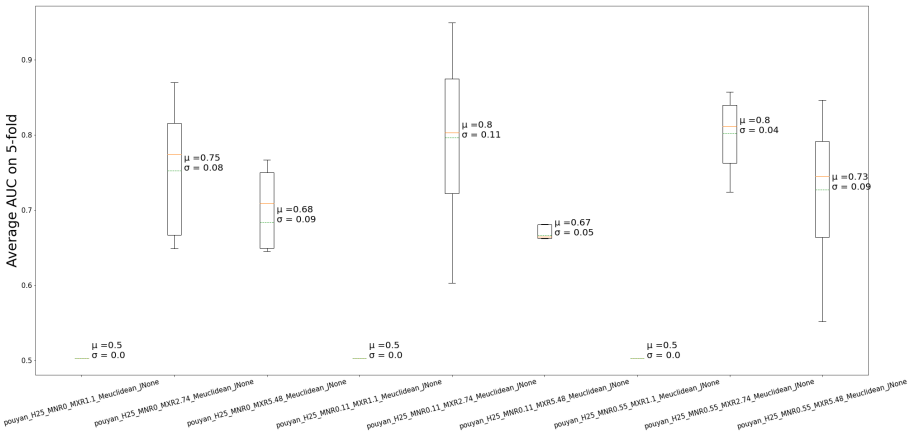

FIGURE 5.3: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = 2$) on the Breast Cancer dataset.

in comparison with Figure 5.3. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

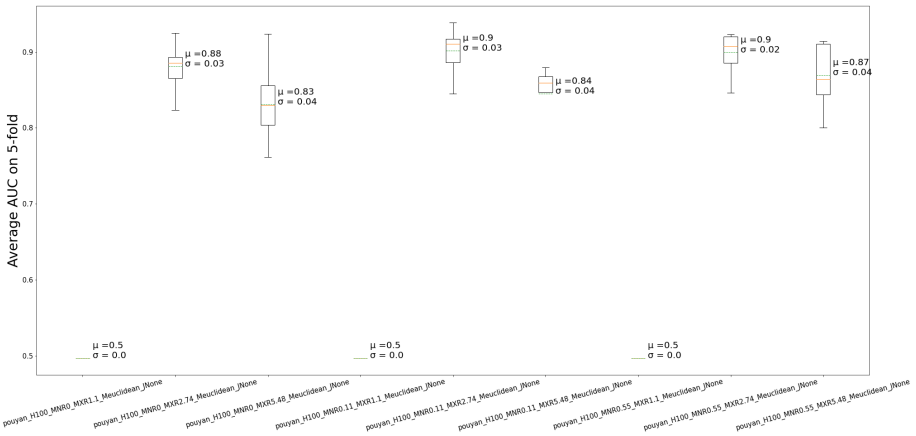

FIGURE 5.4: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = 2$) on the Breast Cancer dataset.

Figure 5.5 illustrates the AUC box-plot results on the Breast Cancer dataset, fixing $H = 25, L = \infty$ and changing Max_r and Min_r. It can be observed that $Max_r = 1$ corresponding to the maximum length diagonal in the 30-dimensional unit hypercube, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.6 shows the AUC box plots for parameters settings ($H = 100, L = \infty$ and changing Max_r and Min_r) on the Breast Cancer dataset. It can be observed that the mean of AUC is increased, while standard deviation is decreased

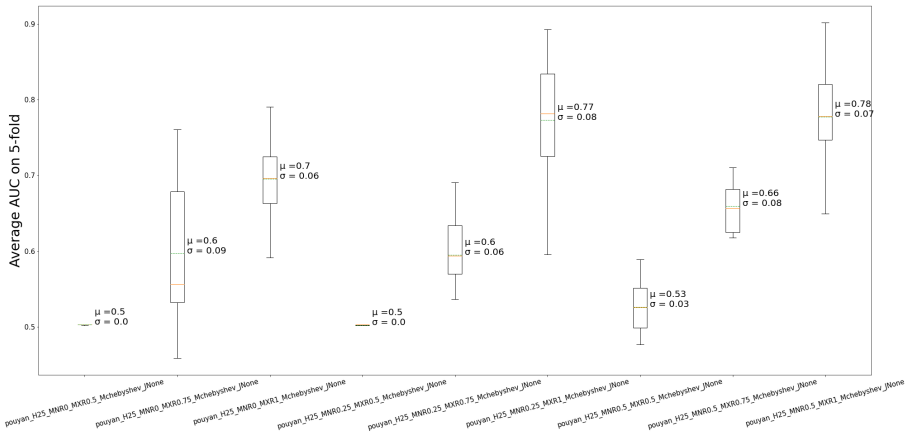

FIGURE 5.5: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = \infty$) on the Breast Cancer dataset.

in comparison with Figure 5.5. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

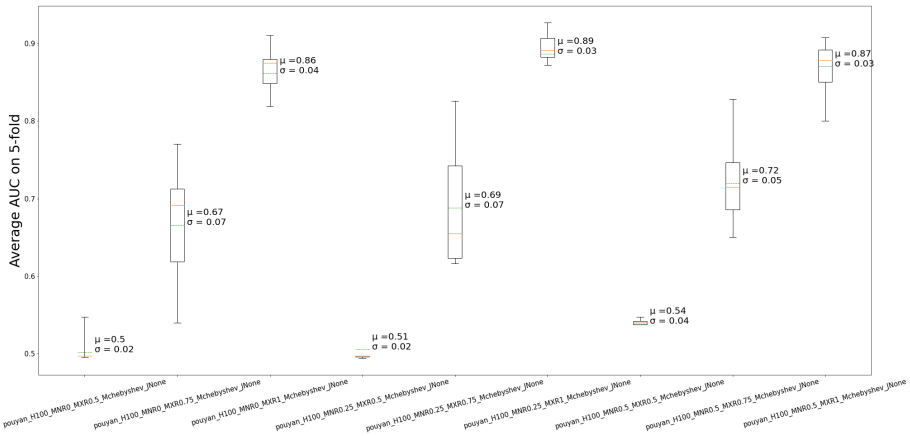

FIGURE 5.6: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = \infty$) on the Breast Cancer dataset.

Figure 5.7 illustrates the AUC box-plot results on the Credit Card Fraud Detection dataset, fixing $H = 25, L = 1$ and changing Max_r and Min_r. It can be observed that $Max_r = 15 = \frac{30^1}{2}$ corresponding to half of the maximum length diagonal in the 30-dimensional unit hyper-cube, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.8 shows the AUC box plots for parameters settings ($H = 100, L = 1$ and changing Max_r and Min_r) on the Credit Card Fraud Detection dataset. It can be observed than the mean of AUC is increased, while standard deviation is

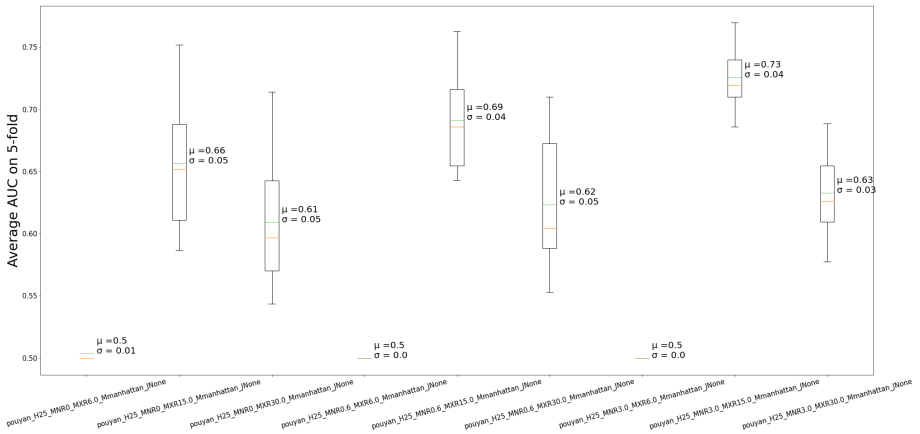

FIGURE 5.7: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = 1$) on the Credit Card Fraud Detection dataset.

decreased in comparison with Figure 5.7. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

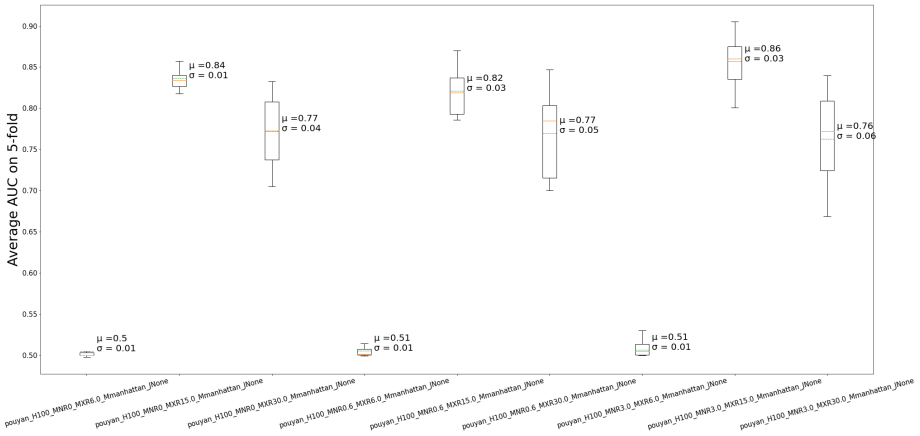

FIGURE 5.8: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = 1$) on the Credit Card Fraud Detection dataset.

Figure 5.9 illustrates the AUC box-plot results on the Credit Card Fraud Detection dataset, fixing $H = 25, L = 2$ and changing Max_r and Min_r. It can be observed that $Max_r = 2.74$ corresponding to half of the maximum length diagonal in the 30-dimensional unit hyper-cube, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.10 shows the AUC box plots for parameters settings ($H = 100, L = 2$ and changing Max_r and Min_r) on the Credit Card Fraud Detection dataset.

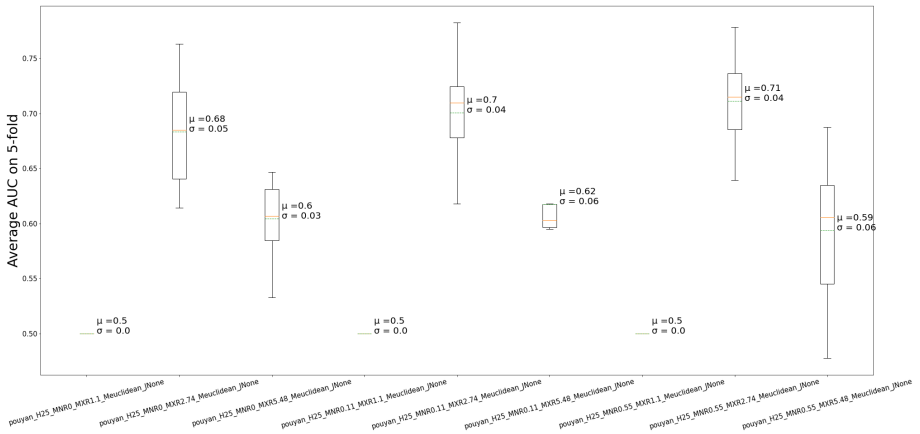

FIGURE 5.9: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = 2$) on the Credit Card Fraud Detection dataset.

It can be observed than the mean of AUC is increased, while standard deviation is decreased in comparison with Figure 5.9. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

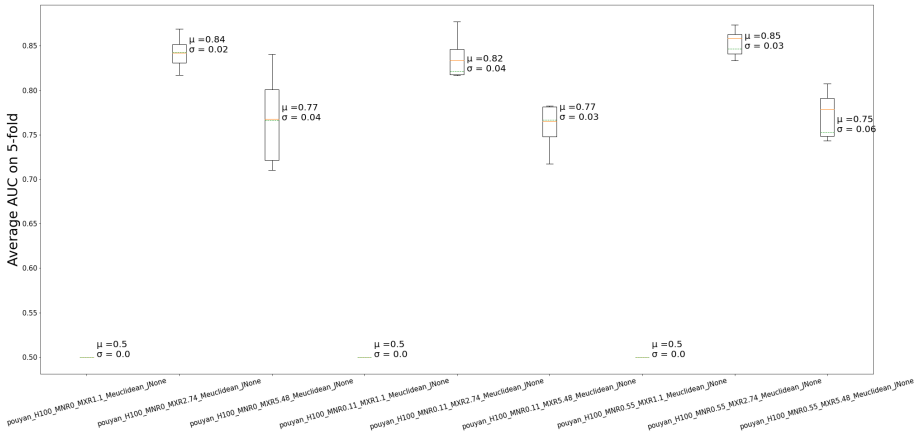

FIGURE 5.10: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = 2$) on the Credit Card Fraud Detection dataset.

Figure 5.11 illustrates the AUC box-plot results on the Credit Card Fraud Detection set, fixing $H = 25, L = \infty$ and changing Max_r and Min_r. It can be observed that $Max_r = 0.75$, tends to have higher AUC than other parameter settings. Furthermore, the performance of the algorithm is not sensitive on Min_r.

Similarly, Figure 5.12 shows the AUC box plots for parameters settings ($H =$

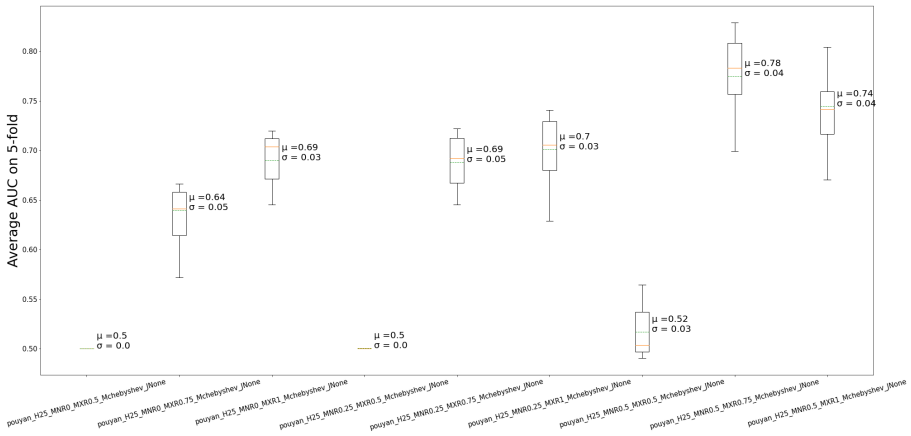

FIGURE 5.11: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 25, L = \infty$) on the Credit Card Fraud Detection dataset.

$100, L = \infty$ and changing Max_r and Min_r) on the Credit Card Fraud Detection dataset. It can be observed that the mean of AUC is increased, while standard deviation is decreased in comparison with Figure 5.11. This observation points out that more number of random shapes can increase the detection performance of the proposed algorithm.

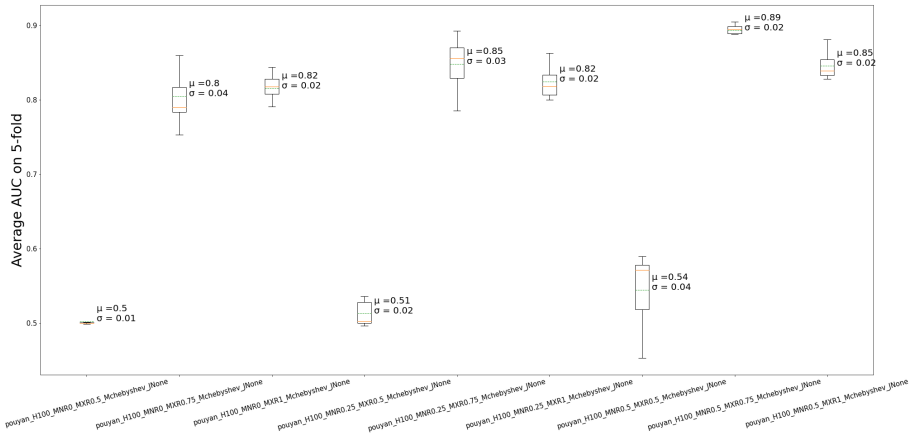

FIGURE 5.12: AUC box plot for sensitivity analysis of the proposed algorithm ($H = 100, L = \infty$) on the Credit Card Fraud Detection dataset.

Tables 5.1 and 5.2 summarise the results of Experiment 1. In summary, we can observe and conclude the followings from the first experiment:

- Generally, Manhattan distance ($L = 1$) tends to have a robust performance in this experiment.

- The improvement of performance of the algorithm when lower L such as $L = 1$ is chosen rather than the standard Euclidean distance $L = 2$ is due to the higher separability of the underlying distance functions in the high-dimensional space than $L = 2$.

- Generally, the algorithm is more sensitive to varying Max_r than Min_r. The best optimal performance is always observed for large portions of maximum length diagonal of the unit hyper-cube. For example $Max_r = \frac{D^{1/L}}{2}$ corresponding to the half of the maximum length of the diagonal of the D-dimensional hyper-cube if Manhattan or Euclidean distances are chosen. If Chebyshev is chosen as distance function, generally Max_r near 0.75 tends to perform well on the datasets. Choosing Max_r above these numbers results in the volume of the shapes, encompasses a large portion of the space, thus all the data points will land inside and have the same binary pattern, which results in less capability of the algorithm to distinguish normal data from abnormal ones. In addition, low values for Max_r lead to small shapes, which do not have any data points inside, thus again the same problem of having similar binary patterns for most of the data and less separability of them occurs, which results in poor performance. Therefore, the optimal half of the length of the diagonal of the hyper cube as Max_r creates a balance between these two extreme situations and improves the performance of the algorithm.

- H is independent of geometrical parameters such as L, Min_r and Max_r.

- Min_r parameter is less sensitive than the Max_r. It can also be chosen as 0, however shapes with very small volume do not bring any separability power to the algorithm.

- Finally, it should be noted that each dataset requires an specific geometrical parameters (e.g. underlying distance function and minimum and maximum upper bounds for the size of random shapes) to be capable of extracting patterns which needs optimisation for that particular dataset. Therefore, the above mentioned items only provide a general (default) prescriptions on how to choose a good starting value for geometric hyper-parameters and the user

is not limited to only utilise Minkowski distances and other distances may be suitable for different datasets.

TABLE 5.1: Summary table of sensitivity analysis (average AUC) for the proposed algorithm on the Breast Cancer (BC) and Credit Card (CC) Fraud Detection datasets in Experiment 1 for $H = 25$. All numbers are rounded up to two decimal points.

Proposed Algorithm	**AUC on BC;** $L = 1$	**AUC on BC;** $L = 2$	**AUC on BC;** $L = \infty$	**AUC on CC;** $L = 1$	**AUC on CC;** $L = 2$	**AUC on CC;** $L = \infty$
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.75**	**0.75**	**0.6**	**0.66**	**0.68**	**0.64**
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.69**	**0.68**	**0.7**	**0.61**	**0.6**	**0.69**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.51**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.78**	**0.8**	**0.6**	**0.69**	**0.7**	**0.69**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.67**	**0.67**	**0.77**	**0.62**	**0.62**	**0.7**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.51**	**0.5**	**0.53**	**0.5**	**0.5**	**0.52**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.81**	**0.8**	**0.66**	**0.73**	**0.71**	**0.78**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.7**	**0.73**	**0.78**	**0.63**	**0.59**	**0.74**

TABLE 5.2: Summary table of sensitivity analysis (average AUC) for the proposed algorithm on the Breast Cancer (BC) and Credit Card (CC) Fraud Detection datasets in Experiment 1 for $H = 100$. All numbers are rounded up to two decimal points.

Proposed Algorithm	**AUC on BC;** $L = 1$	**AUC on BC;** $L = 2$	**AUC on BC;** $L = \infty$	**AUC on CC;** $L = 1$	**AUC on CC;** $L = 2$	**AUC on CC;** $L = \infty$
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.9**	**0.88**	**0.67**	**0.84**	**0.84**	**0.8**
$Min_r = 0$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.85**	**0.83**	**0.86**	**0.77**	**0.77**	**0.82**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.5**	**0.5**	**0.51**	**0.51**	**0.5**	**0.51**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.9**	**0.9**	**0.69**	**0.82**	**0.82**	**0.85**
$Min_r = \begin{cases} \frac{D^{1/L}}{50} & ; L = 1,2 \\ 0.25 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.88**	**0.84**	**0.89**	**0.77**	**0.77**	**0.82**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{5} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$	**0.5**	**0.5**	**0.54**	**0.51**	**0.5**	**0.54**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{2} & ; L = 1,2 \\ 0.75 & ; L = \infty \end{cases}$	**0.91**	**0.9**	**0.72**	**0.86**	**0.85**	**0.89**
$Min_r = \begin{cases} \frac{D^{1/L}}{10} & ; L = 1,2 \\ 0.5 & ; L = \infty \end{cases}$ $Max_r = \begin{cases} \frac{D^{1/L}}{1} & ; L = 1,2 \\ 1 & ; L = \infty \end{cases}$	**0.87**	**0.87**	**0.87**	**0.76**	**0.75**	**0.85**

5.4.2 Results and Discussions Experiment 2:

The results of the second sensitivity analysis experiment in which, we increase the number of random shapes to see how it affects the performance of the algorithm, are shown and discussed below. The numbers are rounded up to two decimal points.

Figure 5.13 illustrates the AUC box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 1$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust. Also, increasing the H more than 1000 is not required as the performance is stabilised around this point. This pattern is also observed in the following diagrams.

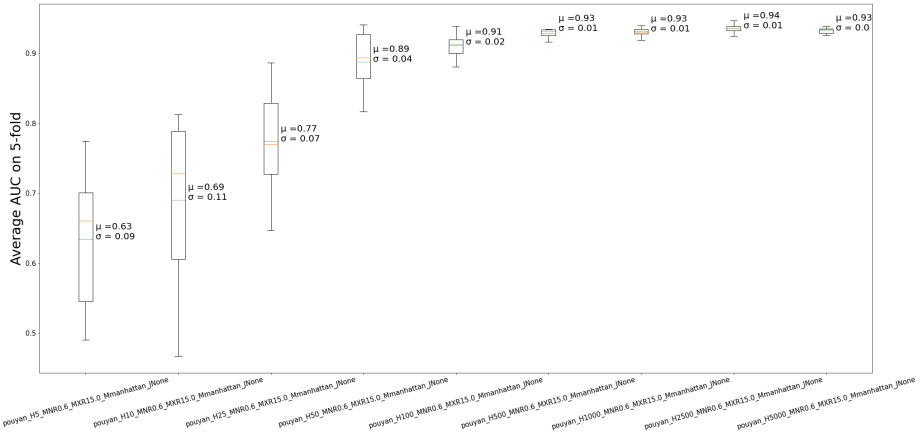

FIGURE 5.13: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 1$) on the Breast Cancer dataset.

Figure 5.14 illustrates the computational time box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 1$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean time increases nearly linearly.

Figure 5.15 illustrates the AUC box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 2$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases,

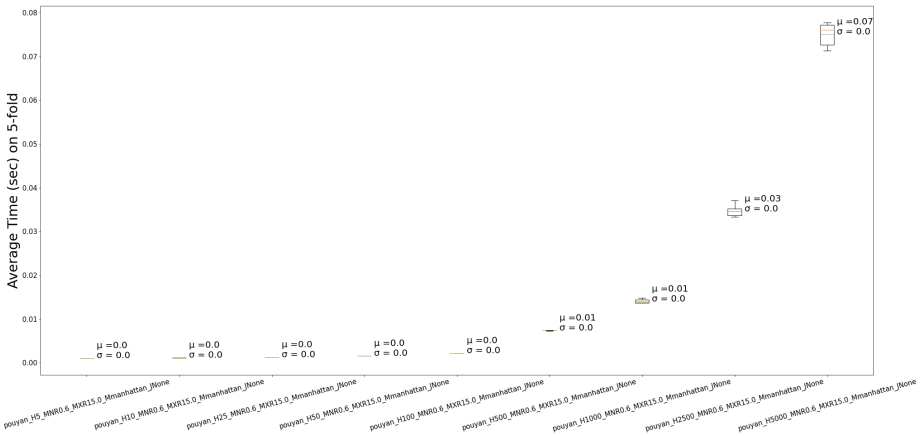

FIGURE 5.14: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 1$) on the Breast Cancer dataset.

meaning that performance of the anomaly detection becomes more stable and robust.

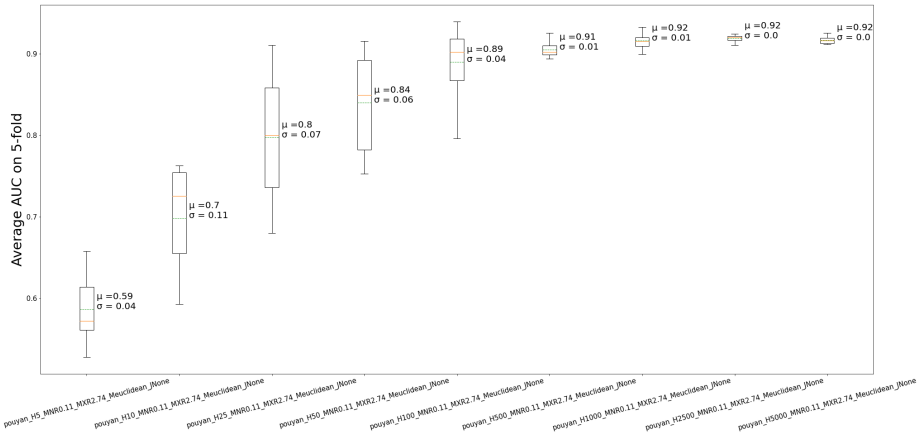

FIGURE 5.15: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 2$) on the Breast Cancer dataset.

Figure 5.16 illustrates the computational time box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 2$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean Time increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

Figure 5.17 illustrates the AUC box-plot results on the Breast Cancer dataset.

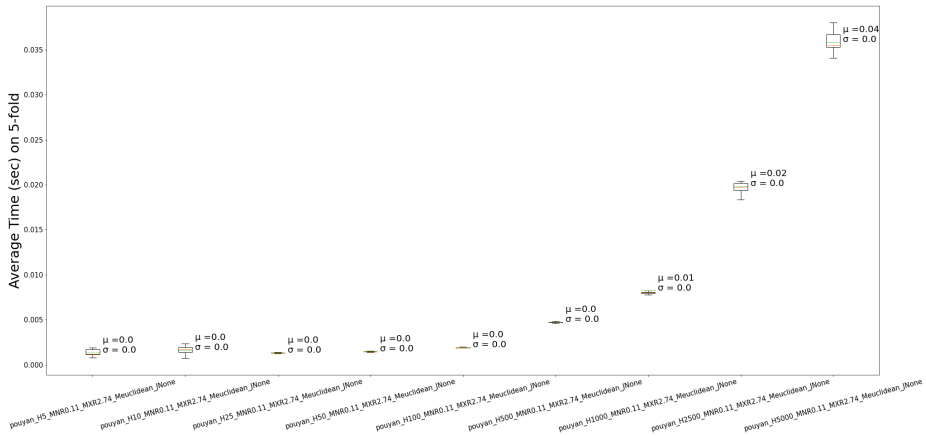

FIGURE 5.16: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 2$) on the Breast Cancer dataset.

The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = \infty$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

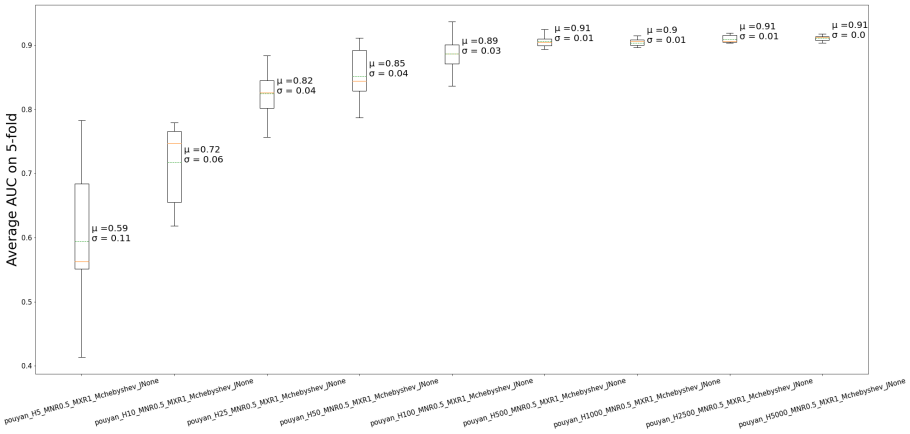

FIGURE 5.17: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = \infty$) on the Breast Cancer dataset.

Figure 5.18 illustrates the computational time box-plot results on the Breast Cancer dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = \infty$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean time increases nearly linearly.

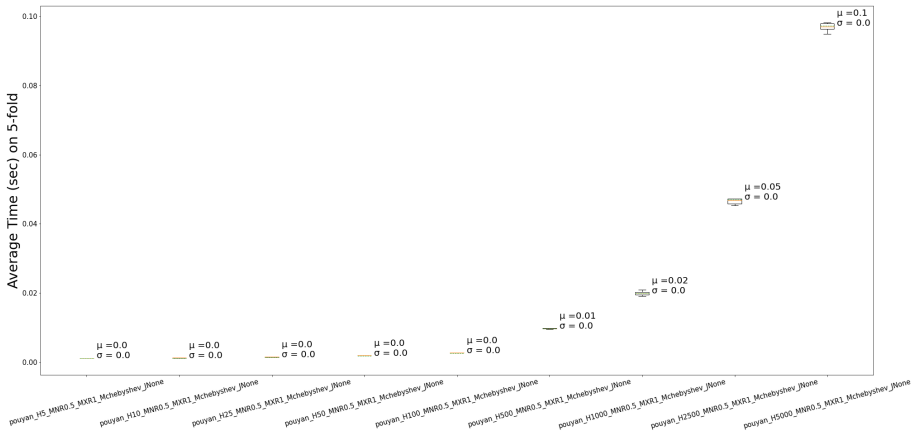

FIGURE 5.18: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = \infty$) on the Breast Cancer dataset.

Figure 5.19 illustrates the AUC box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 1$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

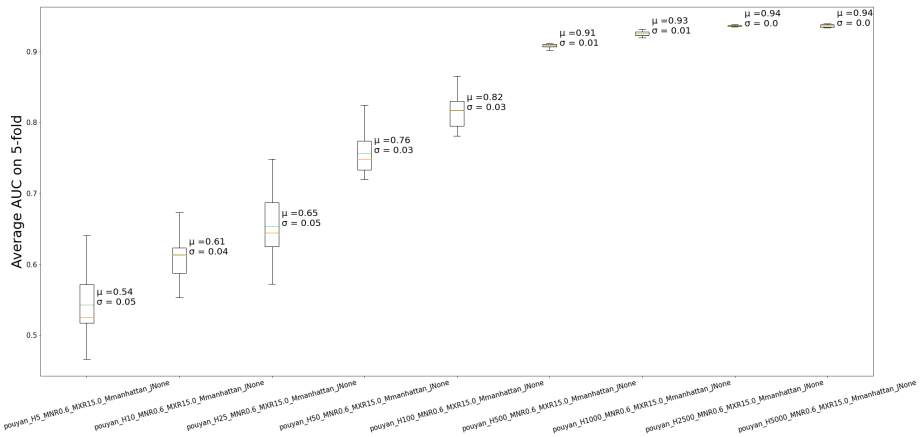

FIGURE 5.19: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 1$) on the Credit Card Fraud Detection dataset.

Figure 5.20 illustrates the computational time box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 1$ and increasing H from 5 -

5000). It can be observed that as H increases, the mean time increases nearly linearly.

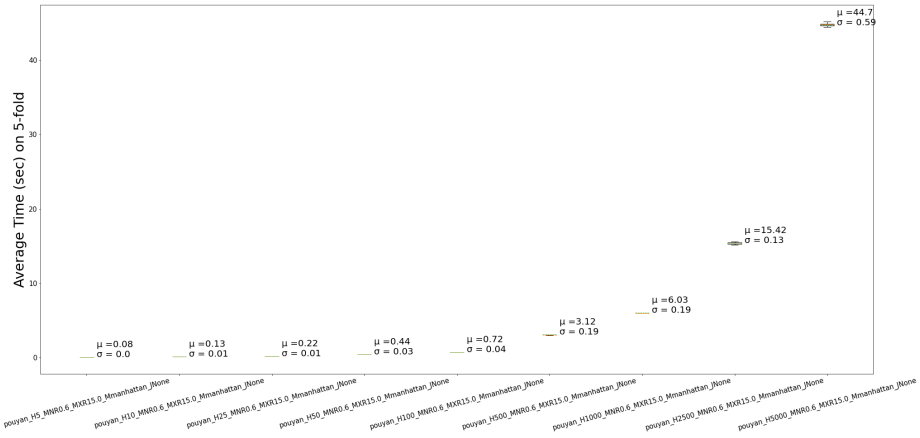

FIGURE 5.20: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 1$) on the Credit Card Fraud Detection dataset.

Figure 5.21 illustrates the AUC box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 2$ and increasing H from 5 - 5000). The Y-axis shows the range (distribution) as a box-plot of AUC. It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

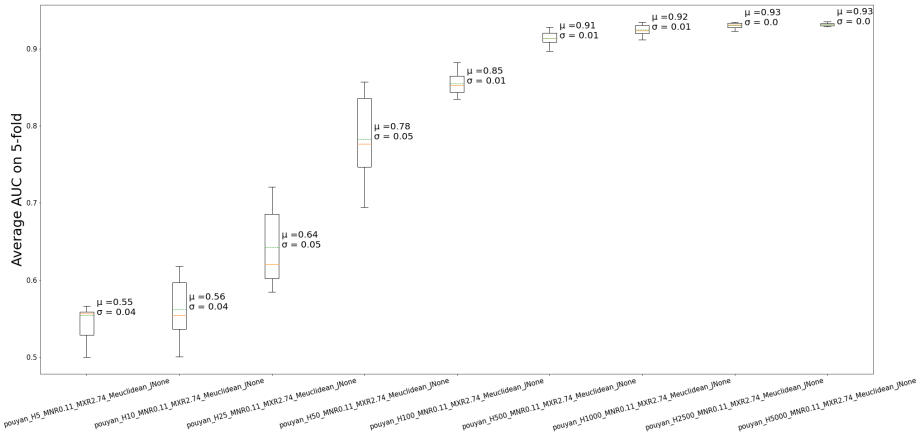

FIGURE 5.21: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 2$) on the Credit Card Fraud Detection set.

Figure 5.22 illustrates the computational time box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the

proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = 2$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean time increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

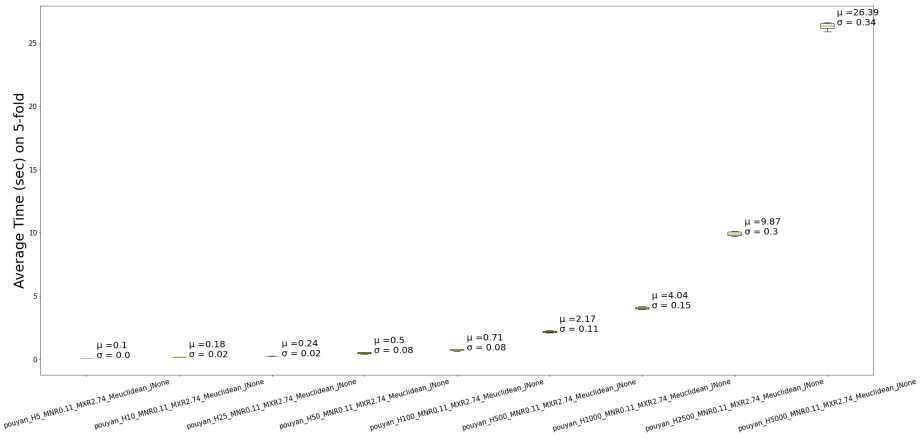

FIGURE 5.22: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = 2$) on the Credit Card Fraud Detection dataset.

Figure 5.23 illustrates the AUC box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = \infty$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean AUC increases, while its standard deviation decreases, meaning that performance of the anomaly detection becomes more stable and robust.

Figure 5.24 illustrates the computational time box-plot results on the Credit Card Fraud Detection dataset. The X-axis indicates the different parameter settings of the proposed algorithm (fixing $Min_r = 0.6, Max_r = 15, L = \infty$ and increasing H from 5 - 5000). It can be observed that as H increases, the mean time increases nearly linearly.

Tables 5.3 and 5.4 summarise the results of the sensitivity analysis on the benchmark datasets. In summary, we can observe and conclude the followings from the second experiment:

- As H increases, the performance of the algorithm (mean AUC) increases and becomes more stable and robust (standard deviation of AUC decreases) at the

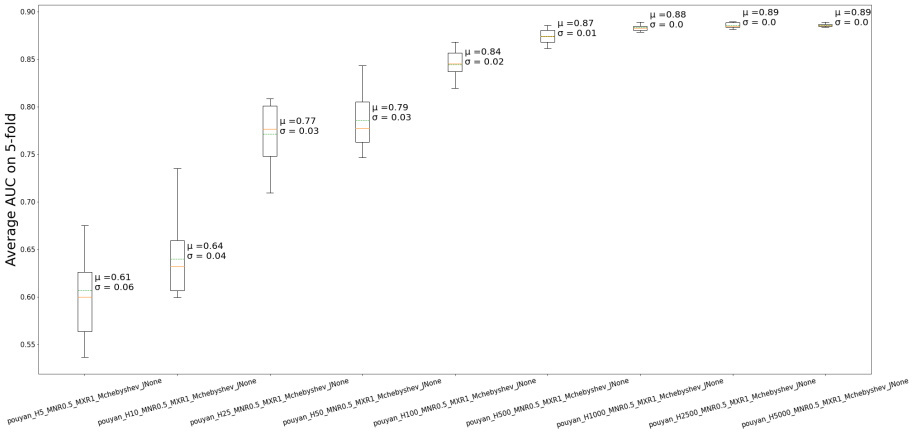

FIGURE 5.23: AUC box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = \infty$) on the Credit Card Fraud Detection dataset.

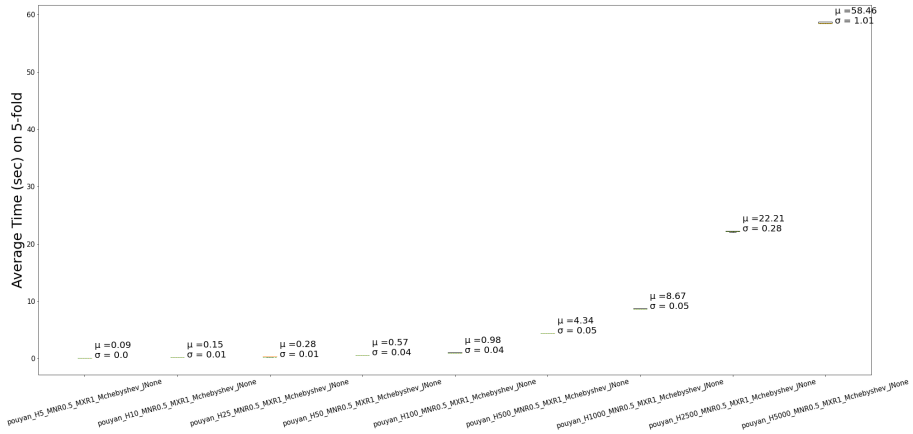

FIGURE 5.24: Computation Time box plot for sensitivity analysis of the proposed algorithm ($H = [5, 5000], L = \infty$) on the Credit Card Fraud Detection dataset.

expense of the higher computational time. This is due to the fact that the covered volume of the shapes increases and more regions of the space will be explored by the increasing number of shapes. Furthermore, the separability power of the algorithm increases.

- The optimal $H = 1000$ is recommended for best performance (high mean AUC and low standard deviation of AUC). Increasing H, above this point may improve the performance of the algorithm a little further but more computation time is needed. In other words, the performance is saturated around this number.

- Obviously, as number of random shapes increases, the computational time increases. However, this increase remains linear as a function of the number of random shapes (H). This is very beneficial for processing large datasets.

- On the Breast Cancer dataset, Manhattan distance performs better (mean AUC = 0.94) than Chebyshev distance in terms of AUC, while Euclidean distance performs well in terms of rapidity of computation.

The experiments 1 and 2 investigated the effect of independent variables called shapes' parameters and number of shapes, respectively. Generally, increasing the number of random shapes improves the detection performance of the algorithm while introducing more computational time. In addition, the shapes' parameters must be optimised for a specific dataset. Generally, the shape parameters must not be chosen too small or too large to reduce the detection performance of the algorithm.

Finally, we can conclude that the proposed unsupervised anomaly detection algorithm is generic and able to identify anomalies in both financial and non-financial-related datasets.

TABLE 5.3: Summary table of sensitivity analysis (average AUC and Average run time) for the proposed algorithm on the Breast Cancer Dataset in Experiment 2. All numbers are rounded up to two decimal points. We fix the $Min_r = \frac{D^{1/L}}{50}$ and $Max_r = \frac{D^{1/L}}{2}$ for different $L = \{1, 2\}$. For $L = \infty$, we fix $Min_r = 0.5$ and $Max_r = 1$. Number of the shapes increases from 5 to 5000 ($H = \{5, 10, 25, 50, 100, 500, 1000, 2500, 5000\}$).

Proposed Algorithm	**AUC** $L = 1$	**Run Time** $L = 1$	**AUC** $L = 2$	**Run Time** $L = 2$	**AUC** $L = \infty$	**Run Time** $L = \infty$
$H = 5$	**0.63**	**0.00**	**0.59**	**0.00**	**0.59**	**0.00**
$H = 10$	**0.69**	**0.00**	**0.7**	**0.00**	**0.72**	**0.00**
$H = 25$	**0.77**	**0.00**	**0.8**	**0.00**	**0.82**	**0.00**
$H = 50$	**0.89**	**0.00**	**0.84**	**0.00**	**0.85**	**0.00**
$H = 100$	**0.91**	**0.00**	**0.89**	**0.00**	**0.89**	**0.00**
$H = 500$	**0.93**	**0.01**	**0.91**	**0.00**	**0.91**	**0.01**
$H = 1000$	**0.93**	**0.01**	**0.92**	**0.01**	**0.9**	**0.02**
$H = 2500$	**0.94**	**0.03**	**0.92**	**0.02**	**0.91**	**0.05**
$H = 5000$	**0.93**	**0.07**	**0.92**	**0.04**	**0.91**	**0.1**

TABLE 5.4: Summary table of sensitivity analysis (average AUC and Average run time) for the proposed algorithm on the Credit Card Fraud Detection dataset in Experiment 2. All numbers are rounded up to two decimal points. We fix the $Min_r = \frac{D^{1/L}}{50}$ and $Max_r = \frac{D^{1/L}}{2}$ for different $L = \{1, 2\}$. For $L = \infty$, we fix $Min_r = 0.5$ and $Max_r = 1$. Number of the shapes increases from 5 to 5000 ($H = \{5, 10, 25, 50, 100, 500, 1000, 2500, 5000\}$).

Proposed Algorithm	**AUC** $L = 1$	**Run Time** $L = 1$	**AUC** $L = 2$	**Run Time** $L = 2$	**AUC** $L = \infty$	**Run Time** $L = \infty$
$H = 5$	**0.54**	**0.08**	**0.55**	**0.1**	**0.61**	**0.09**
$H = 10$	**0.61**	**0.13**	**0.56**	**0.18**	**0.64**	**0.15**
$H = 25$	**0.65**	**0.22**	**0.64**	**0.24**	**0.77**	**0.28**
$H = 50$	**0.76**	**0.44**	**0.78**	**0.5**	**0.79**	**0.57**
$H = 100$	**0.82**	**0.72**	**0.85**	**0.71**	**0.84**	**0.98**
$H = 500$	**0.91**	**3.12**	**0.91**	**2.17**	**0.87**	**4.34**
$H = 1000$	**0.93**	**6.03**	**0.92**	**4.04**	**0.88**	**8.67**
$H = 2500$	**0.94**	**15.42**	**0.93**	**9.87**	**0.89**	**22.21**
$H = 5000$	**0.94**	**44.7**	**0.93**	**26.39**	**0.89**	**58.46**

Chapter 6

Abnormal Trading Detection and Evaluations

In this Chapter, we apply the proposed anomaly detection algorithm (see Chapter 4) on the Bitcoin [70] trading data and compare its performance with existing algorithms. The aim of this chapter is to see how well the proposed algorithm is capable of identifying real abnormal patterns in the Bitcoin data as a case study. Since the data are not labelled and not feature extracted, we will pre-process the data, feature extract and label the data by statistical analysis, which will be explained in more detail in Section 6.1.

6.1 Methodology

6.1.1 Bitcoin dataset

The unlabelled Bitcoin data utilised in this study are Bitcoin trading data of the BitMex Exchange, one of the most liquid Bitcoin exchanges in the world with the 24-hour, 30-day and 365-day volume of 4.93 billion, 80.77 billion and 1.27 trillion US dollars, respectively [71].

The exchange has a limit order book just like a typical stock exchange which enables traders and investors to buy and sell Bitcoin and other crypto-currencies. Since the trading data (such as limit order book) of the BitMex are publicly available and can be downloaded via the APIs provided by the exchange [72], the data are very accessible and suitable for this research, in contrast to existing stock market

data, which are hard to obtain and usually are not free (especially the limit order book data).

We utilise bid/ask prices and bid/ask volumes of the Bitcoin limit order book of the BitMex exchange in our experiments. The data for a specific trading day are download from [72] automatically by our Python code, which we developed in Google Colab, a free cloud platform provided by Google for writing Python code online and performing the computation and machine learning tasks in Google computing back-end engines. The reason for doing the computation on the cloud is the large number of data instances in each trading data (over 4 million data in each day). Therefore, cloud computing enables us to efficiently obtain and pre-process the data.

The features we extract from the data are discussed in Section 6.1.2. Since the data are not labelled (we do not know in advance which data instance is normal or abnormal), the data will be labelled with a robust statistical method which is explained in more detail in Section 6.1.3.

6.1.2 Feature Extraction in Bitcoin data

Feature extraction is an important task before feeding the data into the machine learning algorithms. It computes meaningful features (dimensions) from data, enabling machine learning algorithms for further distinguishing normal and abnormal data based on the extracted features. In this research, we utilise 20 features from the data which include 4 features as original data and 16 derived features from the original features. Mathematically, consider a 20-dimensional dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ with $N = 5000$ data samples from each trading day such as each data sample $\mathbf{x}_i = [x_{i,d}]_{d=1}^{20}$ is a 20-dimensional vector with dimensional values $x_{i,d}$.

The details of the features are provided in Table 6.1. The Bid and Ask prices and sizes at different times are essential primary variables, which build the main time-series data. The rest of features compute moving average and gradient-based features derived from the original features to extract the movement and slope of main variables during specific time intervals. The reason for choosing these 20 feature is to extract variables which may aid the anomaly detection algorithms to distinguish normal data from abnormal ones. Section 6.1.3, provides more details regarding how these 20 features are utilised for labelling the data.

TABLE 6.1: Extracted features of the Bitcoin trading data. $\lambda = 10$ is chosen for computing the features.

Name Feature $x_{i,d}$	**Description**
$Time_i$	seconds passed after the first timestamp = $timestamp_i - timestamp_1$
$BidPrice_i$	i-th bid price
$BidSize_i$	i-th bid size (volume)
$AskPrice_i$	i-th ask price
$AskSize_i$	i-th ask size (volume)
$DeltaTime_i$	$Time_i - Time_{i-1}$
$DeltaBidPrice_i$	$BidPrice_i - BidPrice_{i-1}$
$DeltaAskPrice_i$	$AskPrice_i - AskPrice_{i-1}$
$DeltaBidSize_i$	$Bidsize_i - BidSize_{i-1}$
$DeltaAskSize_i$	$AskSize_i - AskSize_{i-1}$
$SpreadPrice_i$	$AskPrice_i - BidPrice_i$
$RatioAskBidSize_i$	$\frac{AskSize_i}{BidSize_i}$
$MovingAverageBidPrice_i^\lambda$	$\frac{\sum_{g=i-\lambda-1}^{i} BidPrice_g}{\lambda}$
$MovingAverageAskPrice_i^\lambda$	$\frac{\sum_{g=i-\lambda-1}^{i} AskPrice_g}{\lambda}$
$MovingAverageBidSize_i^\lambda$	$\frac{\sum_{g=i-\lambda-1}^{i} BidSize_g}{\lambda}$
$MovingAverageAskSize_i^\lambda$	$\frac{\sum_{g=i-\lambda-1}^{i} AskSize_g}{\lambda}$
$GradientBidPrice_i^\lambda$	$\frac{BidPrice_i - MovingAverageBidPrice_i^\lambda}{Time_i - Time_{i-\lambda-1}}$
$GradientBidSize_i^\lambda$	$\frac{BidSize_i - MovingAverageBidSize_i^\lambda}{Time_i - Time_{i-\lambda-1}}$
$GradientAskPrice_i^\lambda$	$\frac{AskPrice_i - MovingAverageAskPrice_i^\lambda}{Time_i - Time_{i-\lambda-1}}$
$GradientAskSize_i^\lambda$	$\frac{AskSize_i - MovingAverageAskSize_i^\lambda}{Time_i - Time_{i-\lambda-1}}$

6.1.3 Bitcoin Data Statistical Analysis and Labelling

As mentioned, BTC data are not labelled. In other words, we do not know in advance which data sample is normal or abnormal. Since labelling the dataset with only one variable may introduce bias, full density estimation of 20 features must be considered to include all the interactions between the features. However, accurate density estimation in a 20-dimensional space is not possible due to large number of features and curse of dimensionality. To bypass this issue, the dimensionality of the dataset is first reduced and then, density estimation is performed on the reduced space as a proxy method.

In order to label the data, first we reduce the dimensionality of data via PCA [73] to one dimension, then the probability density function of the encoded data can be estimated by a well known density estimation technique called Kernel Density Estimation (KDE) [74]. Anomalies are located in the low density regions of the space. The left diagrams in Figures 6.1 and 6.2 illustrate this concept.

To quantify the degree of anomalousness for each data sample, we define an anomaly score for each data sample as the fraction of maximum density observed by the KDE in the whole dataset (corresponding to the mode of the 1D probability distribution) divided by individual data density (computed by the KDE). The right diagrams in Figures 6.1 and 6.2 illustrate this concept.

Such a relative density of a data sample quantification acts as a normalisation score in order to have a measure to rank data based on their degree of anomalousness. Higher score corresponds to higher anomalousness. The distribution of such anomaly score are shown in the right-hand diagrams of the following figures.

Finally, data samples in the top 1 percentile of the anomaly score distribution, are labelled as 1 (positive class corresponding to anomaly samples) and the rest of data are labelled as 0 (negative class or corresponding normal samples)

The Figures 6.1 and 6.2 illustrate the above process of how labeling the BTC dataset works for samples from dates 2/4/2020 and 6/4/2020, respectively. As it is shown in these figures, the distribution of BTC data changes dynamically over time (see left diagram of each figure below). However, the anomaly scores distribution (see right diagram of each figure below) show a general pattern in which, most of the

data samples have low anomaly scores (corresponding to normal samples), while a few have significantly large anomaly scores. These extreme values are outliers and such data points can be separated by a percentile threshold 0.99 (corresponding to the top 1 percent of data located in the right tail of the anomaly scores distribution).

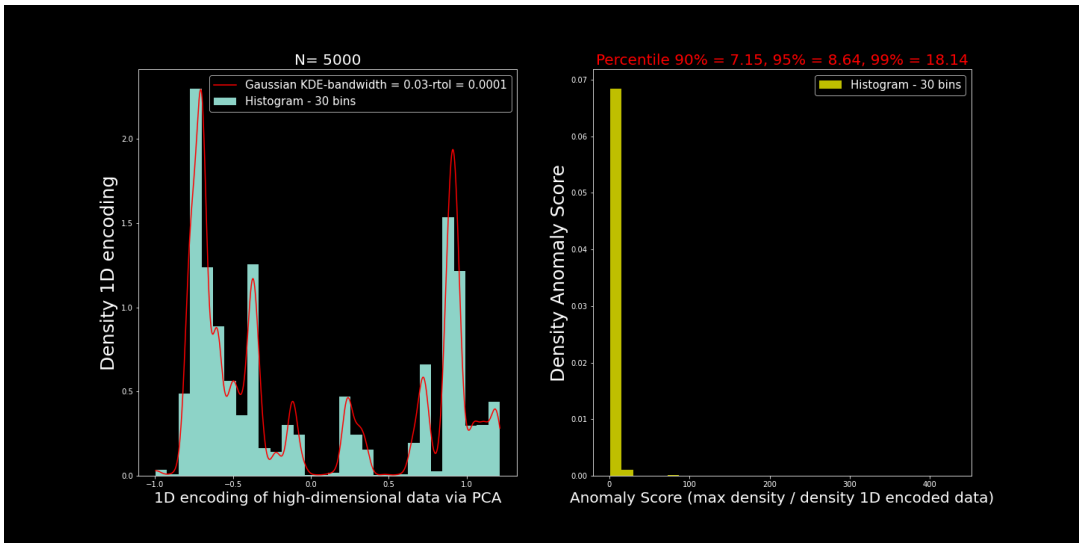

FIGURE 6.1: Left diagram: Density estimation on 1D encoded BTC data via PCA. Right diagram: Anomaly Scores distribution based on relative densities. The data are 5000 samples from 2020/04/02.

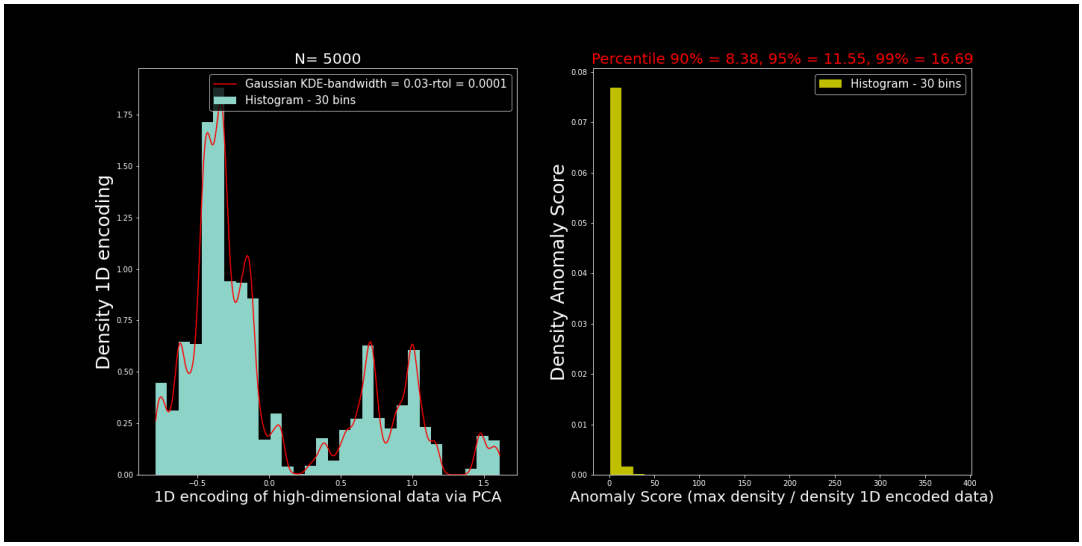

FIGURE 6.2: Left diagram: Density estimation on 1D encoded BTC data via PCA. Right diagram: Anomaly Scores distribution based on relative densities. The data are 5000 samples from 2020/04/06.

6.1.4 Evaluation Metrics

Same performance metrics as discussed in Section 3.2 are utilised in this study. Generally, higher mean AUC, lower standard deviation of AUC, lower mean and standard deviation of computation time are desirable.

6.1.5 Experiments Setups and Customisation of the proposed Algorithm

The experiment condition in this study is same as explained in Section 3.3. Please refer to that section for detailed description of the experiment. The only difference between this experiment and other experiments is the dataset, which is feature extracted and labelled as explained in Sections 6.1.2 and 6.1.3, respectively. As explained previously, the labelled dataset for each sample date contains 5000 samples of which 1 percent (50 samples) are anomaly and the rest are normal.

Furthermore, we choose the proposed anomaly detection algorithm with hyper-parameter settings $H = 1000, MinR = 0.95, MaxR = 1, Distance = Chebyshev$ and $H = 1000, MinR = 10, MaxR = 10, Distance = Canberra$ for two samples dates 02/04/2020 and 06/04/2020, respectively. As explained and investigated in Chapter 5 regarding hyper-parameters of the proposed algorithm, number of random shapes $H = 1000$ is suitable for most of the datasets since increasing this number will introduce more computational requirement. Furthermore, the shape parameters are chosen so the resulting generated random shapes are neither too small nor big to affect the performance of anomaly detection.

The Canberra distance [75], [76] is a weighted form of Manhattan distance [77] and has been utilised for detecting computer intrusion [78]. It is defined as:

$$CanberraDistance(\mathbf{x_i}, \mathbf{x_j}) = \sum_{d=1}^{D} \frac{|x_{i,d} - x_{j,d}|}{|x_{i,d}| + |x_{j,d}|}. \tag{6.1}$$

Next, we compare the proposed algorithm with the existing bench mark algorithms described in Section 3.3. The benchmark algorithms include KNN, Isolation Forest, LOF, ABOD, HBOS, PCA and Autoencoder as explained in Section 3.3. The hyper-parameters setting for these algorithms are same as previous experiments. Generally, we are interested to see which algorithm has a high AUC and low computational time.

6.2 Results and Discussions

Figures 6.3 and 6.5 show AUC box-plots of the proposed anomaly detection algorithm on 02/04/2020 and 06/04/2020, respectively. Each plot corresponds to the AUC results for a specific date. The X-axis of each plot shows the name of each algorithm and Y-axis demonstrates the AUC distribution shown via a box-plot. Since some algorithms are deterministic and some are probabilistic, this range representation of performances including minimum, quartiles and maximum values facilitates identifying robust anomaly detection algorithms. In addition, we show the mean and standard deviation of each distribution next to its corresponding box-plot. From these diagrams, we can observe that AUC of the proposed algorithm is higher than the benchmark algorithms. For instance the proposed algorithm has a AUC = 0.903, while the best bench mark algorithm on 06/04/2020 has AUC = 0.818. This significant 8.5 percent change demonstrates the robustness of the proposed algorithm in terms of anomaly detection capability. Also, AUC around 0.94 is observed for the proposed algorithm on 02/04/2020, which is higher than the performance of the benchmark algorithms.

Figures 6.4 and 6.6 show Computational time (measured in seconds) box-plots of the proposed anomaly detection algorithm on 02/04/2020 and 06/04/2020, respectively. The X-axis of each plot shows the name of each algorithm and Y-axis demonstrates the computational time distribution shown via a box-plot. As some algorithms are deterministic and some are probabilistic, this range representation of performances including minimum, quartiles and maximum values facilitates identifying robust anomaly detection algorithms. In addition, we show the mean and standard deviation of each distribution next to its corresponding box-plot. From these figures, we can observe that the computational time of the proposed algorithm is generally lower than most of the benchmark algorithms.

The computational efficiency of the proposed algorithm is due to generating random shapes data independently. Since the computational time of the algorithm increases linearly with number of generated random shapes, enough random shapes can perform the task of anomaly detection in a timely manner.

Furthermore, the underlying anomaly detection mechanism of the proposed algorithm which is based on converting data features into binary patterns of falling inside or outside of the shapes and then, computing the probability of observing such patterns, facilities the anomaly detection task.

Considering both anomaly detection performance and computational time, a requirement for detecting ATPs in real-world cases rapidly, we can conclude that the proposed unsupervised anomaly detection algorithm performs better than the existing algorithms on the BTC dataset. Thus, it can serve as a potential algorithm in future for detecting ATPs in a timely manner.

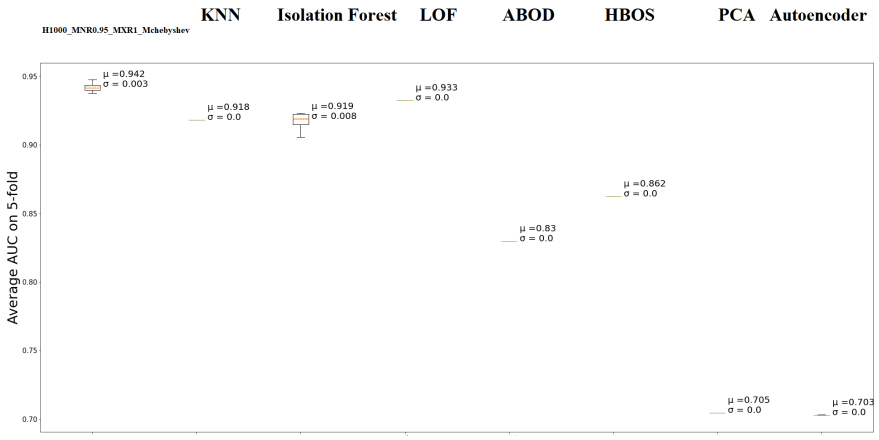

FIGURE 6.3: AUC boxplot of proposed algorithm compared with benchmark algorithms on Bitcoin data (02/04/2020).

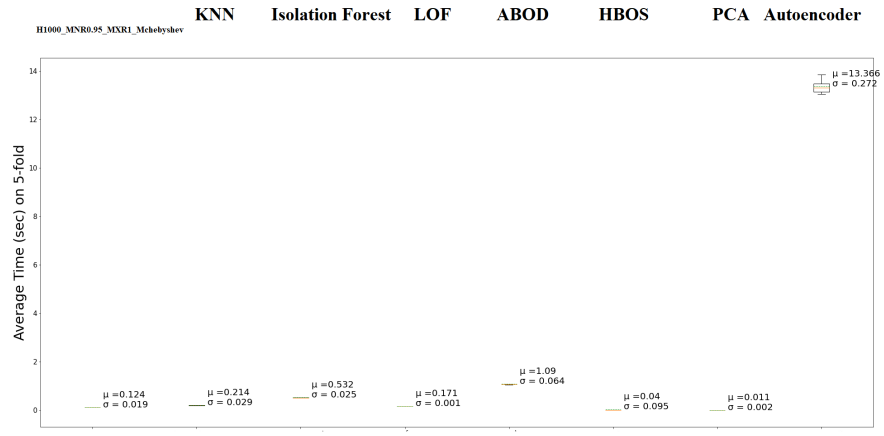

FIGURE 6.4: Computational time boxplot of proposed algorithm (Pouyan) compared with benchmark algorithms on Bitcoin data (02/04/2020).

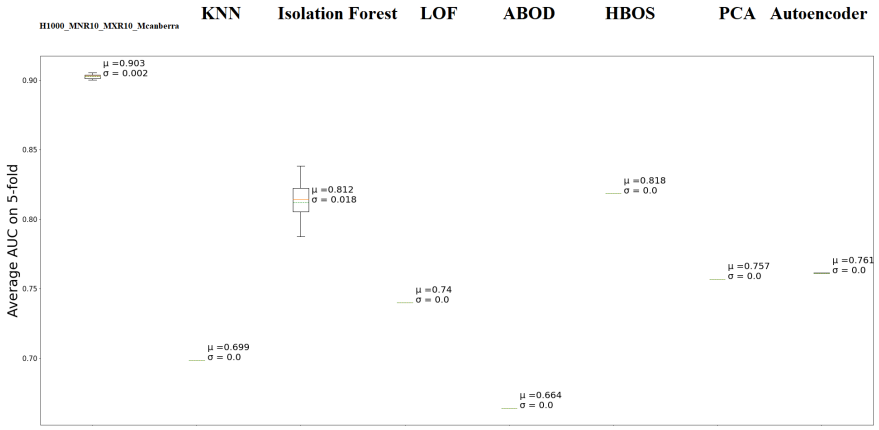

FIGURE 6.5: AUC boxplot of proposed algorithm compared with benchmark algorithms on Bitcoin data (06/04/2020).

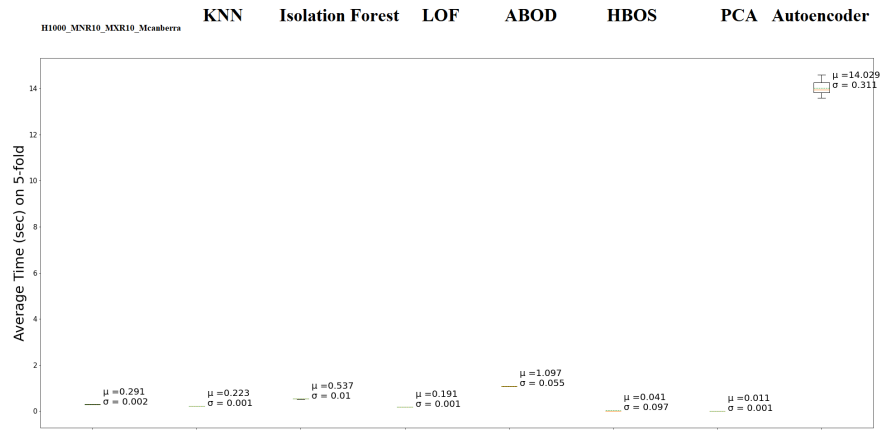

FIGURE 6.6: Computational time boxplot of proposed algorithm (Pouyan) compared with benchmark algorithms on Bitcoin data (06/04/2020).

Chapter 7

Conclusions and Future Work

7.1 Summary of the Thesis

Surveillance and monitoring of financial markets, where illegitimate high frequency trading strategies may artificially distort the price of a financial asset for gaining illegal profit, is a key and challenging task to protect the legitimate traders which can not be performed by humans due to the large size and high-velocity of trading data. Alternatively, anomaly detection algorithms can automatically detect such abnormal trading patterns (ATPs). However, designing and developing an unsupervised algorithm capable of detecting such ATPs rapidly is a gap in the body of literature, which this PhD research attempts to fill.

In this PhD thesis, a literature review and thorough benchmark evaluation on existing anomaly detection algorithms such as Artificial Neural Network Auto Encoder, Isolation Forest, Local outlier Factor (LOF), Histogram-based outlier Score (HBOS), Angle-based Outlier Detection (ABOD), Principle Component Analysis (PCA) and K-Nearest Neighbors (KNN) was performed. These algorithms evaluated via Area Under the ROC Curve (AUC) and computational time on publicly labeled datasets from different domains such as finance and health. The results show that Isolation Forest, HBOS and PCA are robust algorithms in terms of high detection performance (Area Under the ROC Curve (AUC) = 0.95) and low computational time specially for large datasets.

Next, a novel unsupervised anomaly detection algorithm/model based on probabilistic (random) partitioning of a high-dimensional space is proposed. The algorithm is intuitively based on the idea of partitioning a bounded D-dimensional

space in the unit-hyper cube by a sequence of random shapes, in which each data will be trapped (isolated) either inside (encoded by 1) or outside (encoded by 0). Such a partitioning scheme, encodes each data into a binary pattern (sequence of 0s and 1s). Under the fundamental assumption for all anomaly detection algorithms that anomalous data (minority data) are rare and have different characteristics, the proposed algorithm learns the binary patterns by the probabilistic modelling approach and can distinguish anomalous data whose binary patterns of trapping (inside or outside) based on the sequence of random shapes is significantly different from rest of the dataset (normal data or majority). Finally, the algorithm assigns an anomaly score for each data, which indicates the degree to which it is anomalous. The proposed algorithm/model is generic, fast and robust. Not only can it be applied on financial data, but also can have applications in detecting anomalies rapidly in datasets other than the financial domain.

Then, we performed sensitivity analysis of the proposed algorithm on publicly available labeled datasets and show that the performance of the algorithm will stabilise as the number of random shapes increases. Furthermore, the shape of random shapes can affect the performance of the algorithm which needs to be optimised for a given dataset. Also, the results indicate that the algorithm's computational time increases linearly with the number of random shapes which shows the robustness of the algorithm for detecting anomalies in a timely manner.

Finally, we applied the proposed algorithm on Bitcoin trading data as a case study and tested, evaluated and compared the performance of the proposed algorithm with the Auto Encoder, Isolation Forest, LOF, HBOS, ABOD, PCA and KNN. The results show that the proposed algorithm achieves AUC = 0.94. Comparing to the benchmark algorithms, it also outperforms the existing algorithms by 8.5 percent increase while having low computational time.

7.2 Future Work

Future work may include investigation and study of how ensembles of the proposed algorithm with different hyper-parameter settings can affect and possibly boost the

performance of the proposed algorithm. Utilising different shapes and even designing an algorithm to generate random shapes arbitrarily may be worth the future investigations. Specially, non-symmetrical shapes which are generated by another algorithm may occupy the high-dimensional unit hyper-cube completely random, thus exploring the space differently from symmetrical shapes.

In addition, the random shapes which are generated data independently can be modified to be generated data dependently. In other words, the algorithm can produce the shapes depending on a given dataset's properties such as the location of the data points in the high-dimensional space. This may improve the anomaly detection performance of the algorithm but at higher computational cost since the data properties may be first analysed, before generating the shapes.

For processing of large data streams in real-time, the algorithm can be modified to have a training window in which, the algorithm learns and updates itself just by the data points within the window and not the whole training dataset. Once a new test data point arrives, its corresponding anomaly score will be computed and the test data point can be included as part of the training window. The length of the window can be further adjusted to have a dynamic and adaptive algorithm which can be flexible when the underlying data distribution changes over time.

Another further research path may include applying parallel and distributed computing strategies to speed up the algorithm. For instance, the data can be divided into smaller size datasets, which can be processed individually in parallel on multiple computing engines, and later the computational results can be merged together to form the final result. This can significantly increase the speed of the algorithm and will be suitable for processing big data.

Furthermore, financial authorities may investigate how to practically utilise this algorithm for creating a real-time surveillance system capable of automatic monitoring and flagging abnormal trading patterns (e.g. price manipulations) tactics in the financial markets in order to protect the capital of legitimate traders without any delay. The proposed algorithm can be incorporated in financial exchanges for the purpose of law enforcement, by analysing billions of trading data in real-time and identifying and highlighting abnormal and suspicious activities. The flagged trades by the algorithm, can be further investigated by financial authorities. Furthermore,

the algorithm's signal can trigger other automatic surveillance systems to take action and prevent the suspicious traders from trading in the market in order to stop a potential crime.

Another application of the proposed algorithm can be detecting fraudulent bank accounts associated with criminal activities such as money laundering. Since, billions of bank transactions cannot be monitored by humans in real-time, the algorithm can be incorporated in the surveillance systems of the banks to detect and prevent suspicious accounts from transferring illegal money to other accounts.

It is worth highlighting that the algorithm is not limited to be applied on financial datasets. It can be applied on non-financial datasets as well. For instance, the medical and health related data of a patient can be monitored and analysed by the algorithm in real-time to identify an abnormal pattern in the health status of the patient rapidly.

Appendix A

Python Code

This Chapter provides the Python(3+) code developed in this PhD research:

- The Python code developed for the proposed anomaly detection algorithm is provided in Section A.1.
- Section A.2 includes the Python code developed for sensitivity analysis of the proposed algorithm and comparison of different benchmark anomaly detection algorithms with each other and proposed algorithm.

The code is written on the Google Colab [67], an online platform provided by Google, where Python code can be written and run on the cloud. As a result, the following code can be efficiently run on the Google computing servers without the requirement to install and run any packages on a local machine. In order for the code to run properly, the input datasets must be uploaded and stored on the cloud (Google Drive) and it needs to be mounted so the google computing server can have access to the input datasets online. Furthermore, path name files in the code for loading input datasets needs to a appropriately adjusted according to where the datasets are located on the Google Drive, before running the code.

The major open source Python libraries utilised in the following code include pyod [66], numpy [79], [80], pandas [81], [82], sklearn [83], [84], and matplotlib [85].

A.1 Python Code Developed for the Proposed Anomaly Detection Algorithm

The Python code developed for the proposed unsupervised algorithm in Chapter 4 are provided below as a Python function called PouyanAnomalyDetector. To find

anomalies in any given dataset, the data needs to be scaled in range $[0,1]$ and then, splitted into train and test sets. The label information is not required, since the algorithm is unsupervised, it learns from training set and then, predicts the anomaly score for each sample in the test set. Higher anomaly score represents higher degree of anomalous for a given test sample, a real-value number by which, test samples can be ranked.

```
# proposed  anomaly detection algorithm

'''
All Rights Reserved
Author/developer: POUYAN DINARVAND
'''

# imports libraries
import numpy as np
from sklearn.metrics import pairwise_distances

def PouyanAnomalyDetector(train_set, test_set, n_hash = 100,
        min_r = 0.01, max_r = 1,metric='manhattan', p = 1,
        n_jobs = None):
    '''
    - output: generates anomaly scores for test_set as dictionary
              {'anomaly_scores': [anomaly score for test data 1,
              anomaly score for test data 2, ...]}
    - Note 1: train and test data must have same dimensions and be
              in the [0,1] interval (data must be in the unit
              hyper-cube)
    - Note 2: pairwise_distances function can be written for a
              customised distance function representing a
              specific geometry of random shapes
    - inputs:
```

```
26      -   train_set, test_set = training/test datasets as numpy
27              array [[data sample 1],[data sample 2],[...],...],
28              each row corresponds to a data sample
29      -   n_hash = integer number of hashes (random shapes) to
30              obtain binary codes of each data sample
31      -   min_r , max_r = minimum radius and maximum radius
32              (float numbers) for generating a random number
33              from uniform(min_r, max_r) distribution as a
34              threshold random variable for binary hashing
35      -   metric = string name distance function e.g. 'minkowski',
36              'euclidean', 'chebyshev'; which determines
37              the geometry of random shapes
38      -   p = number for Lp norm when the metric  = 'minkowski'.
39              e.g:  => p=2 => Euclidean distance,
40              p=1 =>  Manhattan distance,
41      -   n_jobs=None : computing pair-wise distance on parallel jobs
42              (-1 uses all cores)
43      '''
44
45      n_dim = train_set.shape[1]
46      # random array with size = n_hash * n_dim  as random points uniformly
47      # generated within the unit hyper cube
48      random_points_array = np.random.uniform(0,1,size = (n_hash, n_dim))
49
50      #1D random points uniformly generated between min_r and max_r as
51      #random thresholds for hashing
52      random_r = np.random.uniform(min_r,max_r, n_hash)
53
54
55      # hash training data
56      if(metric == 'minkowski'):
```

```
57          my_hash_train_set = pairwise_distances(X = train_set,
58                      Y = random_points_array, metric=metric, p = p,
59                      n_jobs = n_jobs) <= random_r
60      else:
61          my_hash_train_set = pairwise_distances(X = train_set,
62                      Y = random_points_array, metric=metric,
63                      n_jobs = n_jobs) <= random_r
64
65      # update alpha hyper-parameters
66      #  alpha one (inside)
67      alphas_one = 1 +  my_hash_train_set.sum(axis = 0)
68
69      alphas_zero = np.copy(2 + train_set.shape[0] - alphas_one)
70
71      # compute -log alphas
72      log_alphas_one = -np.log(alphas_one)
73      log_alphas_zero  = -np.log(alphas_zero)
74
75      # clear memory
76      del my_hash_train_set, alphas_one, alphas_zero
77
78      #hash test data
79      if(metric == 'minkowski'):
80          my_hash_test_set = pairwise_distances(X = test_set,
81                      Y = random_points_array, metric=metric, p = p,
82                      n_jobs = n_jobs) <= random_r
83      else:
84          my_hash_test_set = pairwise_distances(X = test_set,
85                      Y = random_points_array, metric=metric,
86                      n_jobs = n_jobs) <= random_r
87
```

```
88      #compute anomaly scores for the test set
89      anomaly_scores = np.dot(my_hash_test_set, log_alphas_one.T) +
90                       np.dot(~my_hash_test_set, log_alphas_zero.T)
91
92      return {'anomaly_scores': anomaly_scores +
93              ( n_hash* np.log(2 + train_set.shape[0]))}
```

A.2 Python Code Developed for Evaluation of Anomaly Detection Algorithms and Sensitivity Analysis

The following Python code developed for the purpose of evaluation of anomaly detection algorithms and performing sensitivity analysis on the proposed algorithm. Please note that the python function A.1 must be loaded before running the following code. The code must be run on the Google Colab. In order for the code to run properly, the user input parameters for running the experiment and path name files must be adjusted depending on where the datasets (as csv file) are stored. The code evaluates different algorithms (or the proposed algorithm with different parameter settings) and outputs the AUC and computational time. The results are also saved as csv files and shown/plotted as png files. The path name, where the results should be saved, must also be adjusted in the code depending on where the user prefers to store the output files.

```
1
2   '''
3   - All Rights Reserved
4   - Author/developer: POUYAN DINARVAND
5   '''
6
7   # mount drive to be able to load datasets from Google drive
8   from google.colab import drive
9   drive.mount('/gdrive')
10
```

```
print('mounting completed.')

#------------ Imports ----------------

!pip install pyod
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import pairwise_distances
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import  StratifiedKFold
from pyod.models.knn import KNN
from pyod.models.iforest import IForest
from pyod.models.lof import LOF
from pyod.models.abod import ABOD
from pyod.models.hbos import HBOS
from pyod.models.pca import PCA
from pyod.models.auto_encoder import AutoEncoder
from time import time

# sensitivity analysis parameters of
#proposed anomaly detection algorithm

list_name_pouyan_algorithms = []
n_hash= [] # number of random shapes
min_r = []
max_r = []
metric = [] # name of distance function as string
n_jobs = [] # number of jobs
p = [] # L norm in the thesis
```

```
42
43 # !!! include numbers in lists [....] of For loops
44 # to set the experiment parameters ready
45 # for sensitivity analysis
46 for nh in [1000]: # number of hashes (random shapes)
47     for minr in [0.6]: # Min R
48         for maxr in  [15]: # Max R
49             # e.g. 'minkowski' ,'manhattan',
50             #'canberra', 'euclidean','chebyshev',
51             for mtr in ['manhattan']:
52                 # None for small datasets, -1 for
53                 #large datasets uses all cpu cores
54                 #(for multi #processing)
55                 for njb in [None]:
56                     for ppp in [1]:
57
58                         if(mtr == 'minkowski'):
59                             list_name_pouyan_algorithms.append(
60                             'pouyan_H'+str(nh)+'_MNR'+str(minr)+
61                             '_MXR'+str(maxr)+'_M'+str(mtr)+'_L'+
62                             str(ppp)+'_J'+str(njb))
63                             n_hash.append(nh)
64                             min_r.append(minr)
65                             max_r.append(maxr)
66                             metric.append(mtr)
67                             p.append(ppp)
68                             n_jobs.append(njb)
69
70                         else:
71                             list_name_pouyan_algorithms.append(
72                             'pouyan_H'+str(nh)+'_MNR'+str(minr)+
```

```
73                                  '_MXR'+str(maxr)+'_M'+str(mtr)+'_J'+
74                                  str(njb))
75                                  n_hash.append(nh)
76                                  min_r.append(minr)
77                                  max_r.append(maxr)
78                                  metric.append(mtr)
79                                  p.append(0)
80                                  n_jobs.append(njb)
81                                  break
82
83 #@title ------------- User Inputs -------------
84
85 #@markdown > Enter experiment details:
86
87 #@markdown Number of experiments:
88 num_experiments = 10 #@param {type:"integer"}
89
90 #@markdown n-fold:
91 n_fold = 5 #@param {type:"integer"}
92 #@markdown Random seed integer for shuffling data:
93 random_seed_integer = 123456789 #@param {type: "number"}
94
95 #@markdown Name of the input csv file:
96 name_data =
97     "BTC_date20200406_start_index_data1500000_moving_average10_max_num_
98 ␣␣␣␣fake_anomaly0_num_real_data1000000_prob_fake_abnormal0_
99 ␣␣␣␣time1590938691_NEW_Labeled_by_PCA_KDE_threshold_percentile_99_
100 ␣␣␣␣N_5000"#@param ["creditcard","breast-cancer-unsupervised-ad",
101     "BTC_date20200402_start_index_data1000000_moving_average10_max_
102 ␣␣␣␣num_fake_anomaly0_num_real_data1000000_prob_fake_abnormal0_
103 ␣␣␣␣time1590861495_NEW_Labeled_by_PCA_KDE_threshold_percentile_99_N_5000", "BT
```

```
104 ␣␣␣␣max_num_fake_anomaly0_num_real_data1000000_prob_fake_abnormal0_
105 ␣␣␣␣time1590938691_NEW_Labeled_by_PCA_KDE_threshold_percentile_99_
106 ␣␣␣␣N_5000"] {allow-input: true}
107
108
109 #@markdown Save performance results:
110 save_results_gradually = False #@param {type: "boolean"}
111 save_results_as_csv = True #@param {type: "boolean"}
112
113 #@markdown Save and show performance plots:
114 save_plots = True #@param {type: "boolean"}
115 #@markdown how many dicimal points to round numbers on plots:
116 round_my_numbers_decimals = 3 #@param {type:"integer"}
117
118 #@markdown >Show log each algorithm
119 verbose_each_algo = False  #@param {type: "boolean"}
120 ################## Pouyan Algorithm ######################
121 #@markdown > Pouyan anomaly detection algorithm parameters:
122
123 #@markdown  Run Pouyan algorithm(s):
124 pouyan_yes = True #@param {type: "boolean"}
125
126 #@markdown Manually insert list in format [.,.,.,] for
127 # parameters of Pouyan's algorithms (if False, defualt
128 # parameters lists will be used):
129 insert_parameters_sensitivity_manually = False #@param
130                                         {type: "boolean"}
131
132
133 if(insert_parameters_sensitivity_manually):
134     #@markdown List name pouyan algorithms in format
```

```
135     # 'pouyan_HXXXX_minRXXX_maxRXXX_M-XXXXXXXXX'
136     list_name_pouyan_algorithms = ['pouyan_H1000_minR-
137 ␣␣␣␣␣␣␣␣1_maxR-1_M-euclidean', 'pouyan_H1000_minR-1_
138 ␣␣␣␣␣␣␣␣maxR-1_M-manhattan', 'pouyan_H1000_minR0.01_
139 ␣␣␣␣␣␣␣␣maxR0.5_M-braycurtis'] #@param
140     #@markdown List Number of hashes:
141     n_hash = [1000,1000,1000] #@param
142
143     #@markdown List Min R for generating random threshold
144     #for hashing (if -1 is chosen, then min_r = sqrt(number
145     #of data dimensions)/50):
146     min_r =  [-1,-1,0.01]#@param
147     #@markdown List Max R for generating random threshold
148     #for hashing (if -1 is chosen, then max_r = sqrt(number
149     #of data dimensions)/2):
150     max_r =  [-1,-1,0.5]#@param
151     #@markdown List Distance metric for hashing:
152     metric=['euclidean', 'manhattan', 'braycurtis']  #@param
153     #@markdown List p-norm:
154     p=[2, 1, None]  #@param
155     #@markdown List Number of parallel cores for speeding up
156     #the algorithm (if -1 is chosen, all avaliable cores are
157     #used):
158     n_jobs = [None,None, None] #@param
159     ########################################################
160
161 #@markdown > Run and compare benchmark anomaly
162 #detection algorithms:
163 run_benchmark_algo = True  #@param {type: "boolean"}
164
165 #@markdown Choose benchmark  algorithms:
```

```
166 knn_yes = True  #@param {type: "boolean"}
167 isolation_forest_yes = True  #@param {type: "boolean"}
168 lof_yes = True  #@param {type: "boolean"}
169 abod_yes = True  #@param {type: "boolean"}
170 hbos_yes = True  #@param {type: "boolean"}
171 pca_yes = True  #@param {type: "boolean"}
172 autoencoder_yes = True  #@param {type: "boolean"}
173 #@markdown ---
174
175 list_names_anm_algos = []
176
177
178 if(pouyan_yes):
179     for item in list_name_pouyan_algorithms:
180         list_names_anm_algos.append(str(item))
181
182 if(run_benchmark_algo):
183     if(knn_yes):
184         list_names_anm_algos.append('knn')
185     if(isolation_forest_yes):
186         list_names_anm_algos.append('isolation_forest')
187     if(lof_yes):
188         list_names_anm_algos.append('lof')
189     if(abod_yes):
190         list_names_anm_algos.append('abod')
191     if(hbos_yes):
192         list_names_anm_algos.append('hbos')
193     if(pca_yes):
194         list_names_anm_algos.append('pca')
195     if(autoencoder_yes):
196         list_names_anm_algos.append('autoencoder')
```

```
197
198
199
200
201 #----------------------------------Data------------------
202
203 # Note: change path-file  name according on where you store
204 #datasets as csv files in the Google Drive
205 # Note: do not change the name of files
206 data = pd.read_csv("/gdrive/My␣Drive/python/anomaly␣detection
207 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/data/"+name_data+".csv")
208
209
210 # BTC dataset name msut started with 'BTC'
211 if(name_data[0:3] == "BTC"):
212
213     y = np.copy(data['label'].values) # labels
214     # delete irrelevant columns from dataframe
215     del data['Unnamed:␣0'], data['label']
216     X = data.values # get features
217
218     # scale X in range [0,1]
219     X = (X - X.min(axis = 0))/(X.max(axis = 0) -
220         X.min(axis = 0))
221
222
223
224 elif(name_data == "creditcard"):
225
226     y = np.copy(data['Class'].values) # labels
227     # delete irrelevant columns from dataframe
```

```
228     del data['Class']
229     X = data.values # get features
230
231     # scale X in range [0,1]
232     X = (X - X.min(axis = 0))/(X.max(axis = 0) -
233     X.min(axis = 0))
234
235
236
237 # other datasets
238 else:
239     X = data.values[:,0:-1]
240     y = data.values[:, data.shape[1] - 1]
241     # change labels 'n' and o' to 0 = normal and 1 =
242     # abnormal respectively
243     y = np.where(y == 'n', 0, 1)
244
245     # scale X in range [0,1]
246     X = (X - X.min(axis = 0))/(X.max(axis = 0) -
247     X.min(axis = 0))
248
249 print('name dataset: ', name_data)
250 print('number of all data = ' + str(X.shape[0]))
251 print('number of dimensions = ' + str(X.shape[1]))
252
253
254
255
256 #---------------- Model ------------------------
257
258 # initialize the dictionary to store performance
```

```
259 # metrics for each algorithm
260 output_dic = {'experiment_id': []}
261 for algo in list_names_anm_algos:
262     # auc score at each experiment
263     output_dic[algo+'_auc'] = []
264     # computation time at each experiment
265     output_dic[algo+'_time'] = []
266
267
268 num_dim = X.shape[1]
269
270 # max_r and min_r list should have equal lengths
271 # replace -1 by the defaults values
272 if(pouyan_yes):
273     for i in np.arange(len(max_r)):
274         if(metric[i] == 'minkowski'):
275             if(max_r[i] == -1):
276                 max_r[i] = round(np.power(num_dim,
277                                  1/p[i])/2,2)
278             if(min_r[i] == -1):
279                 min_r[i] = round(np.power(num_dim,
280                                  1/p[i])/50,2)
281
282         elif(metric[i] == 'manhattan'):
283             if(max_r[i] == -1):
284                 max_r[i] = round(float(num_dim)/2,2)
285             if(min_r[i] == -1):
286                 min_r[i] = round(float(num_dim)/50,2)
287
288
289         elif(metric[i] == 'chebyshev'):
```

```
290                 if(max_r[i] == -1):
291                     max_r[i] = 1
292                 if(min_r[i] == -1):
293                     min_r[i] = 0.5
294
295             else:  # for other metrics
296                 if(max_r[i] == -1):
297                     max_r[i] = round(np.sqrt(num_dim)/2,2)
298                 if(min_r[i] == -1):
299                     min_r[i] = round(np.sqrt(num_dim)/50,2)
300
301
302 # n-fold cross validation (Stratified version for
303 #having balanced data partitions)
304 kf = StratifiedKFold(n_splits=n_fold, shuffle=True,
305                     random_state=random_seed_integer)
306
307
308 if(len(name_data)>=20):# shorten large name string
309     name_data = name_data[0:20]
310
311 for n in range(0,num_experiments):
312
313     output_dic['experiment_id'].append(n+1)
314     print("-------------------------------------")
315     print('>>> Num Experiment:' + str(n+1))
316
317     # counter to see how many pouyan algorithms have
318     # been run in the recent experiment
319     counter_pouyan_algorithms = 0
320
```

```
321     for algo in list_names_anm_algos:
322
323         if(verbose_each_algo):
324             print('Algorithm ' + algo + ' is performing
325 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣anomaly␣detection␣task␣...␣')
326         algo_ok = True # algorithm performed ok
327         try:
328             if(algo[0:6] == 'pouyan'):
329                 # just for compatibility issues
330                 anm = 'pouyan'
331                 counter_pouyan_algorithms +=1
332             elif(algo == 'knn'):
333                 anm = KNN()
334             elif(algo == 'isolation_forest'):
335                 anm = IForest()
336             elif(algo == 'lof'):
337                 anm = LOF()
338             elif(algo == 'abod'):
339                 anm = ABOD()
340             elif(algo == 'hbos'):
341                 anm = HBOS()
342             elif(algo == 'pca'):
343                 anm = PCA()
344             elif(algo == 'autoencoder'):
345                 # min number of dimensions
346                 # should be >= 10
347                 anm = AutoEncoder(hidden_neurons =
348                     [8,4,4,8], verbose=0)
349
350         except Exception as e:
351             print('Error: ' + str(e))
```

```
352             algo_ok = False
353
354         # fit the model
355         # AUC on each n-fold test sets
356         auc_validation_list = []
357         # computaion time on each n-fold test sets
358         time_validation_list = []
359
360         for train_index, test_index in kf.split(X,y):
361
362             X_train, X_test = X[train_index],
363                               X[test_index]
364
365             y_train, y_test = y[train_index],
366                               y[test_index]
367
368             if(algo_ok):
369                 try:
370                     # compute anomaly scores for
371                     # my algorithm
372                     if(algo[0:6] == 'pouyan'):
373                         t1 = time()
374                         y_test_scores = PouyanAnomalyDetector(
375                             train_set = np.copy(X_train),
376                             test_set = np.copy(X_test),
377                             n_hash = n_hash[
378                             counter_pouyan_algorithms
379                             - 1], min_r = min_r[
380                             counter_pouyan_algorithms
381                             - 1], max_r = max_r[
382                             counter_pouyan_algorithms
```

```
383                         - 1], metric=metric[
384                         counter_pouyan_algorithms
385                         - 1], p = p[
386                         counter_pouyan_algorithms - 1],
387                         n_jobs = n_jobs[
388                         counter_pouyan_algorithms -
389                         1])['anomaly_scores'] # outlier scores
390                     t2 = time()
391
392                 # compute anomaly scores for the
393                 # rest of benchmark algorithms
394                 else:
395                     t1 = time()
396                     anm.fit(np.copy(X_train))
397                     y_test_scores =
398                     anm.decision_function(
399                         np.copy(X_test)) # outlier scores
400                     t2 = time()
401
402
403             except Exception as e:
404                 print('Error: ' + str(e))
405                 algo_ok = False
406
407
408             if(algo_ok):
409                 # auc score at each experiment
410                 auc_validation_list.append(
411                     roc_auc_score(y_test,
412                         y_test_scores))
413
```

```
414                     # computation time at each experiment
415                     time_validation_list.append(
416                         t2 - t1)
417
418         # compute performance metrics:
419         # average AUC and Computation time (train+test)
420         # on n-fold CV
421         if(algo_ok):
422             output_dic[algo+'_auc'].append(np.mean(
423                 auc_validation_list))
424             output_dic[algo+'_time'].append(np.mean(
425                 time_validation_list))
426             del anm, y_test_scores # clear memory
427         else:
428             output_dic[algo+'_auc'].append(0) # failed
429             output_dic[algo+'_time'].append(np.Inf) # failed
430
431     # save results as csv gradually in case
432     # the computation timeout
433     string_index_time = "_n_exp_" + str(num_experiments)
434         + "_folds_"+str(n_fold) +"_"+str(int(time()))
435
436     #print(output_dic)
437     if(save_results_as_csv == True):
438         if(save_results_gradually == True or n ==
439             num_experiments-1):
440             # Note: change path-name file according
441             # to where you want to save results as csv
442             pd.DataFrame(output_dic).to_csv("/gdrive/
443 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣My␣Drive/python/anomaly␣detection/data/
444 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣results/"+name_data + string_index_time
```

```
445                 + ".csv")
446
447 # save and show summary statistics output
448
449 list_name_col_auc = []
450 list_name_col_time = []
451 for algo in list_names_anm_algos:
452     list_name_col_auc.append(algo+'_auc')
453     list_name_col_time.append(algo+'_time')
454
455 df_output_stats = pd.DataFrame(output_dic)
456 df_output_stats_auc = df_output_stats[list_name_col_auc]
457 df_output_stats_time = df_output_stats[list_name_col_time]
458
459 df_output_stats_auc = df_output_stats_auc.describe()
460 df_output_stats_auc = df_output_stats_auc.T
461
462 df_output_stats_time = df_output_stats_time.describe()
463 df_output_stats_time = df_output_stats_time.T
464
465 if(save_results_as_csv == True):
466     # Note: change path-name file according to
467     # where you want to save results as csv
468     df_output_stats_auc.to_csv("/gdrive/My␣Drive/
469 ␣␣␣␣␣␣␣␣python/anomaly␣detection/data/results/"+
470         name_data + string_index_time + "_AUC_stats.csv")
471     df_output_stats_time.to_csv("/gdrive/My␣Drive/python/
472 ␣␣␣␣␣␣␣␣anomaly␣detection/data/results/"+name_data +
473         string_index_time + "_Time_stats.csv")
474
475 print('**********␣Top␣Summary␣Statistics␣AUC␣Results
```

```
476 ␣␣␣␣(sorted␣from␣max␣to␣min)␣***************')
477 print(df_output_stats_auc.sort_values(by = 'mean',
478     ascending=False)[['mean']])
479
480 print('***********␣Top␣Summary␣Statistics␣Time␣Results
481 ␣␣␣␣(sorted␣from␣min␣to␣max)␣***************')
482 print(df_output_stats_time.sort_values(by = 'mean',
483     ascending=True)[['mean']])
484
485 print('*********************************')
486
487 # --------------- show results ----------------
488 if(save_plots):
489     try:
490         #  figure line plot results AUC/computation time
491         plt.figure("Anomaly␣detection␣on␣data␣" +
492             name_data, figsize=(20,10))
493         plt.title("Anomaly␣detection␣on␣data␣"+ name_data
494             + "num_data-␣" + str(data.shape[0]) + "num_dim:"
495             + str(data.shape[1]) )
496         # AUC plot
497         plt.subplot(2,1,1)
498         for algo in list_names_anm_algos:
499             plt.plot(output_dic['experiment_id'],
500                 output_dic[algo+'_auc'], marker = 'o',
501                 label = algo + '(' +str(round(np.mean(
502                 output_dic[algo+'_auc']),
503                 round_my_numbers_decimals)) +','+ str(
504                 round(np.std(output_dic[algo+
505                 '_auc']),round_my_numbers_decimals))+
506                 ')')
```

```
507         plt.xlabel('# Experiment')
508         plt.ylabel('# Average AUC on '+ str(n_fold) +
509             '-fold')
510         plt.legend(loc = 'best')
511
512         # time plot
513         plt.subplot(2,1,2)
514         for algo in list_names_anm_algos:
515             plt.plot(output_dic['experiment_id'],
516                 output_dic[algo+'_time'], marker =
517                 'o', label = algo + '(' +str(
518                 round(np.mean(output_dic[algo+
519                 '_time']),round_my_numbers_decimals))
520                 +','+ str(round(np.std(
521                 output_dic[algo+'_time']),
522                 round_my_numbers_decimals))+   ')')
523         plt.xlabel('# Experiment')
524         plt.ylabel('# Average Time (sec) on '+
525             str(n_fold) + '-fold')
526         plt.legend(loc = 'best')
527
528         # save image as png
529         if(save_plots):
530             # Note: change path-name file according
531             #to where you want the plot to be saved
532             #as png file
533             plt.savefig("/gdrive/My Drive/python/
534 ________________anomaly detection/data/results/"+
535                 name_data + string_index_time +
536                 "_lineplot.png")
537
```

```
538         plt.show()
539         plt.close()
540         print('Line␣plot␣of␣performance␣results
541 ␣␣␣␣␣␣␣␣␣␣␣␣plotted.')
542
543     except Exception as e:
544         print('Error:␣' + str(e))
545
546     try:
547         # 1) figure box plot results AUC
548         plt.figure("AUC␣Anomaly␣detection␣on␣data␣" +
549             name_data, figsize=(33,15))
550
551         # AUC plot
552         position_box = 1
553         for algo in list_names_anm_algos:
554             bxpt = plt.boxplot(output_dic[algo+'_auc'],
555                 meanline = True, showmeans= True,vert=
556                 True, autorange= True, showfliers= False,
557                 positions= [position_box])#, labels =
558                 algo + '(' +str(round(np.mean(
559                 output_dic[algo+'_auc']),2)) +','+
560                 str(round(np.std(output_dic[algo+'_auc']),
561                 2))+  ')')
562             x_bxpt, y_bxpt =
563                 bxpt['means'][0].get_xydata()[1]
564             plt.annotate('␣\u03BC␣=' +str(
565                 round(np.mean(output_dic[algo+'_auc']),
566                 round_my_numbers_decimals)) +'\n␣\u03C3␣=
567 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣'+ str(round(np.std(output_dic[algo+'_auc']),
568                 round_my_numbers_decimals)) , xy =
```

```
569                 (x_bxpt, y_bxpt) , fontsize = 20)
570             position_box += 1
571
572         plt.xticks(np.arange(1,position_box),
573             list_names_anm_algos, rotation = 15,
574             fontsize = 15)
575         plt.yticks(fontsize = 15)
576         plt.ylabel('Average␣AUC␣on␣'+ str(n_fold) +
577             '-fold', fontsize = 30)
578
579         # save image as png
580         if(save_plots):
581             # Note: change path-name file according
582             #to where you want the plot to be saved
583             #as png file
584             plt.savefig("/gdrive/My␣Drive/python/
585 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣anomaly␣detection/data/results/"+
586                 name_data + string_index_time +
587                 "_AUC_boxplot.png")
588
589         del bxpt
590         plt.show()
591         plt.close()
592         print('Box␣plot␣of␣AUC␣results␣plotted.')
593
594     except Exception as e:
595         print('Error:␣' + str(e))
596
597     try:
598
599         # 2) figure box plot results time
```

```
600         plt.figure("Time␣Anomaly␣detection␣on␣data␣"
601             + name_data, figsize=(33,15))
602 
603         # time plot
604         position_box = 1
605         for algo in list_names_anm_algos:
606             bxpt = plt.boxplot(output_dic[algo+'_time'],
607                 meanline = True, showmeans= True,
608                 vert= True, autorange= True, showfliers=
609                 False, positions= [position_box])#, labels
610                 = algo + '(' +str(round(np.mean(
611                 output_dic[algo+'_auc']),2)) +','+
612                 str(round(np.std(output_dic[algo+
613                 '_time']),2))+  ')')
614             x_bxpt, y_bxpt =
615                 bxpt['means'][0].get_xydata()[1]
616             plt.annotate('␣\u03BC␣=' +str(round(
617                 np.mean(output_dic[algo+'_time']),
618                 round_my_numbers_decimals)) +
619                 '\n␣\u03C3␣=␣'+ str(round(np.std(
620                 output_dic[algo+'_time']),
621                 round_my_numbers_decimals)) ,
622                 xy = (x_bxpt, y_bxpt), fontsize =
623                 20) # , fontsize =
624                 int(70/len(list_names_anm_algos))
625             position_box += 1
626 
627         plt.xticks(np.arange(1,position_box),
628             list_names_anm_algos, rotation = 15,
629             fontsize = 15)
630         plt.yticks(fontsize = 15)
```

```
631         plt.ylabel('Average␣Time␣(sec)␣on␣'+
632         str(n_fold) + '-fold', fontsize = 30)
633 
634         # save image as png
635         if(save_plots):
636             # Note: change path-name file according
637             #to where you want the plot to be saved
638             #as png file
639             plt.savefig("/gdrive/My␣Drive/python/
640 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣anomaly␣detection/data/results/"+
641                 name_data + string_index_time +
642                 "_Time_boxplot.png")
643 
644         del bxpt
645         plt.show()
646         plt.close()
647         print('Box␣plot␣of␣Computation␣time␣results
648 ␣␣␣␣␣␣␣␣␣␣␣␣plotted.')
649 
650     except Exception as e:
651         print('Error:␣' + str(e))
652 
653 # ------------------------- clear memory
654 del data, X, y, X_train, X_test, y_train, y_test
```

Bibliography

[1] Yi Cao, Yuhua Li, Sonya Coleman, et al. "Adaptive Hidden Markov Model With Anomaly States for Price Manipulation Detection". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.2 (2015), pp. 318–330.

[2] Jia Zhai, Yi Cao, Yuan Yao, et al. "Computational intelligent hybrid model for detecting disruptive trading activity". In: *Decision Support Systems* 93 (2017), pp. 26–41.

[3] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2017. Chap. An Introduction to Outlier Analysis.

[4] Talis J Putnins. "Market Manipulation: A Survey". In: *Journal of Economic Surveys* 26.5 (2012), pp. 952–967.

[5] Franklin Allen. "Stock-Price Manipulation". In: *The Review of Financial Studies* 5 (1992), pp. 503–529.

[6] Christopher M. Bishop. *Pattern recognition and machine learning*. New York: Springer, 2006.

[7] Hulisi Öğüt, M. Mete Doğanay, and Ramazan Aktaş. "Detecting stock-price manipulation in an emerging market: The case of Turkey". In: *Expert Systems With Applications* 36.9 (2009), pp. 11944–11949.

[8] Markus Goldstein and Seiichi Uchida. "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data". In: *PLoS ONE* 11.4 (2016), e0152173.

[9] Stephen Valdez and Philip Molyneux. *An Introduction to Global Financial Markets*. Palgrave Macmillan, 2015.

[10] John C. Hull. *Options, Futures, and Other Derivatives*. Pearson Education, 2011.

[11] Tom Lin. "The New Investor". In: *UCLA Law Review* 60 (Feb. 2013).

[12] Irene Aldridge. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. John Wiley Sons, 2010.

[13] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Computing Suerveys* 41.3 (2009), pp. 1–58.

[14] Kishan G Mehrotra, Huaming Huang, and Chilukuri K Mohan. *Anomaly Detection Principles and Algorithms*. Springer, 2017.

[15] Victoria Hodge and Jim Austin. "A Survey of Outlier Detection Methodologies". In: *Artificial Intelligence Review* 22.2 (2004), pp. 85–126.

[16] C. CORTES and V. VAPNIK. "SUPPORT-VECTOR NETWORKS". In: *MACHINE LEARNING* 20.3 (1995), pp. 273–297.

[17] Teema Leangarun, Poj Tangamchit, and Suttipong Thajchayapong. "Stock Price Manipulation Detection Based on Mathematical Models". In: *International Journal of Trade, Economics and Finance* 7.3 (2016), pp. 81–88.

[18] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, et al. "Estimating the Support of a High-Dimensional Distribution". In: *Neural Computation* 13.7 (2001), pp. 1443–1471.

[19] Zhou Ji and Dipankar Dasgupta. "V-detector: An efficient negative selection algorithm with "probably adequate" detector coverage". In: *Information Sciences* 179 (Apr. 2009), pp. 1390–1406. DOI: 10.1016/j.ins.2008.12.015.

[20] Friedrich Riesz. "Untersuchungen über Systeme integrierbarer Funktionen". In: *Mathematische Annalen* 69 (Dec. 1910), pp. 449–497. DOI: 10.1007/BF01457637.

[21] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient Algorithms for Mining Outliers from Large Data Sets." In: vol. 29. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2000, pp. 427–438. DOI: 10.1145/335191.335437.

[22] Fabrizio Angiulli and Clara Pizzuti. "Fast Outlier Detection in High Dimensional Spaces". In: vol. 2431. Proceedings of the Sixth European Conference on the Principles of Data Mining and Knowledge Discovery, Aug. 2002, pp. 15–26. DOI: 10.1007/3-540-45681-3_2.

[23] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2017. Chap. High-Dimensional Outlier Detection: The Subspace Method.

[24] Charu C. Aggarwal and Philip S. Yu. "Outlier Detection for High Dimensional Data". In: *SIGMOD Rec.* 30.2 (May 2001), pp. 37–46. ISSN: 0163-5808. DOI: 10.1145/376284.375668. URL: https://doi.org/10.1145/376284.375668.

[25] Charu C. Aggarwal and Philip S. Yu. "Outlier Detection for High Dimensional Data". In: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. SIGMOD '01. Santa Barbara, California, USA: Association for Computing Machinery, 2001, pp. 37–46. ISBN: 1581133324. DOI: 10.1145/375663.375668. URL: https://doi.org/10.1145/375663.375668.

[26] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. "What Is the Nearest Neighbor in High Dimensional Spaces?" In: *Proceedings of the 26th International Conference on Very Large Data Bases*. VLDB '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 506–515. ISBN: 1558607153.

[27] David J. Weller-Fahy, Brett J. Borghetti, and Angela A. Sodemann. "A Survey of Distance and Similarity Measures Used Within Network Intrusion Anomaly Detection". In: *IEEE COMMUNICATION SURVEYS TUTORIALS* 17.1 (2015), pp. 70–91.

[28] Andrzej Chmielewski and Sławomir T. Wierzchoń. "On the Distance Norms for Detecting Anomalies in Multidimensional Datasets". In: *ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ* 2 (2007), pp. 39–49.

[29] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the surprising behavior of distance metrics in high dimensional space". In: vol. 1973. BERLIN: 8th International Conference of Database Theory, SPRINGER-VERLAG BERLIN, 2001, pp. 420–434.

[30] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[31] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell: Kluwer Academic Publishers, 1981.

[32] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, et al. "LOF: identifying density-based local outliers". In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 93–104.

[33] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2017.

[34] Ted Johnson and Ivy Kwok. "Fast Computation of 2-Dimensional Depth Contours". In: (Apr. 1999).

[35] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. "Angle-based outlier detection in high-dimensional data". In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 2008, pp. 444–452. DOI: 10.1145/1401890.1401946.

[36] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. "Isolation forest". In: International Conference on Data Mining, IEEE, 2008, pp. 413–422.

[37] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. "Isolation-Based Anomaly Detection". In: *ACM Transactions on Knowledge Discovery From Data - TKDD* 6 (Mar. 2012), pp. 1–39. DOI: 10.1145/2133360.2133363.

[38] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2017. Chap. Linear Models for Outlier Detection.

[39] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, et al. "A Novel Anomaly Detection Scheme Based on Principal Component Classifier". In: Proceedings of International Conference on Data Mining, Jan. 2003.

[40] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2017. Chap. Linear Models for Outlier Detection.

[41] Guansong Pang, Chunhua Shen, Longbing Cao, et al. "Deep learning for anomaly detection: A review". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–38.

[42] Xiaoxiao Ma, Jia Wu, Shan Xue, et al. "A comprehensive survey on graph anomaly detection with deep learning". In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[43] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. English. London;Cambridge, Mass; MIT Press, 2009. ISBN: 9780262013192;0262013193;

[44] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[45] James D. Hamilton. *Time Series Analysis*. Princeton: Princeton University Press, 1994.

[46] F. Rahnamay Roodposhti, M. Falah Shams, and H. Kordlouie. "Forecasting Stock Price Manipulation in Capital Market". In: *International Journal of Economics and Management Engineering* 5.8 (2011), pp. 957–967.

[47] Riccardo Poli, William B. Langdon, Freitag McPhee, et al. *A Field Guide to Genetic Programming*. Lulu, 2008.

[48] Andreea Vlad. "Financial Market Manipulation: How to identify the Mechanisms?" In: *International Journal of Economic Practices and Theories* 4.1 (2014), pp. 77–88.

[49] Murugesan Punniyamoorthy and Jose Joy Thoppan. "ANN-GA based model for stock market surveillance". In: *Journal of Financial Crime* 20.1 (2013), pp. 52–66.

[50] Aihua Li, Jiede Wu, and Zhidong Liu. "Market Manipulation Detection Based on Classification Methods". In: *Procedia Computer Science* 122 (2017), pp. 788–795.

[51] David Diaz, Babis Theodoulidis, and Pedro Sampaio. "Analysis of stock market manipulations using knowledge discovery techniques applied to intraday trade prices". In: *Expert Systems With Applications* 38.10 (2011), pp. 12757–12771.

[52] Jia Zhai, Jia Zhai, Yi Cao, et al. "Data analytic approach for manipulation detection in stock market". In: *Review of Quantitative Finance and Accounting* 50.3 (2018), pp. 897–932.

[53] Longbing Cao, Yuming Ou, Philip Yu, et al. "Detecting abnormal coupled sequences and sequence changes in group-based manipulative trading behaviors". In: ACM, 2010, pp. 85–94.

[54] Yi Cao, Yuhua Li, Sonya Coleman, et al. "A hidden markov model with abnormal states for detecting stock price manipulation". In: IEEE International Conference on Systems, Man, and Cybernetics, SMC, 2013, pp. 3014–3019.

[55] Neil Gandal, JT Hamrick, Tyler Moore, et al. "Price manipulation in the Bitcoin ecosystem". In: *Journal of Monetary Economics* 95 (2018), pp. 86–96.

[56] Andrea Dal Pozzolo. "Adaptive Machine Learning for Credit Card Fraud Detection". PhD thesis. ULB MLG, 2015.

[57] Machine Learning Group - ULB. *Credit Card Fraud Detection: Anonymized credit card transactions labeled as fraudulent or genuine*. URL: https://www.kaggle.com/mlg-ulb/creditcardfraud/home. (accessed: 2020).

[58] Andrea Dal Pozzolo, Olivier Caelen, Reid Johnson, et al. "Calibrating Probability with Undersampling for Unbalanced Classification". In: IEEE Symposium Series on Computational Intelligence (SSCI), Dec. 2015. DOI: 10.1109/SSCI.2015.33.

[59] Andrea Dal Pozzolo, Olivier Caelen, Yann-Aël Le Borgne, et al. "Learned lessons in credit card fraud detection from a practitioner perspective". In: *Expert Systems with Applications* 41 (Aug. 2014), pp. 4915–4928. DOI: 10.1016/j.eswa.2014.02.026.

[60] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, et al. "Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy". In: *IEEE Transactions on Neural Networks and Learning Systems* PP (Sept. 2017), pp. 1–14. DOI: 10.1109/TNNLS.2017.2736643.

[61] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, et al. "SCARFF : a Scalable Framework for Streaming Credit Card Fraud Detection with Spark". In: *Information Fusion* 41 (Sept. 2017). DOI: 10.1016/j.inffus.2017.09.005.

[62] Markus Goldstein. "breast-cancer-unsupervised-ad.tab". In: *Unsupervised Anomaly Detection Benchmark*. Harvard Dataverse, 2015. DOI: 10.7910/DVN/OPQMVF/MTUJ5F. URL: https://doi.org/10.7910/DVN/OPQMVF/MTUJ5F.

[63] Olvi Mangasarian, Nick Street, and William Wolberg. "Breast Cancer Diagnosis and Prognosis Via Linear Programming". In: *Operations Research* 43 (Feb. 1970), pp. 1–18. DOI: 10.1287/opre.43.4.570.

[64] Tilmann Gneiting and Peter Vogel. *Receiver Operating Characteristic (ROC) Curves*. 2018. URL: https://arxiv.org/pdf/1809.04808.pdf.

[65] Markus Goldstein and Andreas Dengel. "Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm". In: KI-2012: Poster and Demo Track, Sept. 2012, pp. 59–63.

[66] Yue Zhao, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection". In: *Journal of Machine Learning Research* 20.96 (2019), pp. 1–7. URL: http://jmlr.org/papers/v20/19-011.html.

[67] *Google Colab*. URL: https://colab.research.google.com. (accessed: 2020).

[68] M. Stone. "Cross-Validatory Choice and Assessment of Statistical Predictions". In: *J R Stat Soc Series B Stat Methodol* 36 (Jan. 1974), pp. 111–113. DOI: 10.1111/j.2517-6161.1974.tb00994.x.

[69] C. Cantrell. *Modern Mathematical Methods for Physicists and Engineers*. Oct. 2000. ISBN: 9780521591805. DOI: 10.1017/9780511811487.

[70] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[71] *BitMEX: Bitcoin Mercantile Exchange: P2P Trading*. URL: https://www.bitmex.com/. (accessed: 10.05.2020).

[72] URL: https://public.bitmex.com/?prefix=data/quote/. (accessed: 10.05.2020).

[73] Ian T. Jolliffe. *Principle Component Analysis*. Springer, 2002.

[74] M. Rosenblatt. "Remarks on Some Nonparametric Estimate of a Density Function". In: *The Annals of Mathematical Statistics* 27 (Jan. 1956), pp. 832–835.

[75] G. Lance and W. Williams. "Computer Programs for Hierarchical Polythetic Classification ("Similarity Analyses")". In: *The Computer Journal* 9 (May 1966), pp. 60–64. DOI: 10.1093/comjnl/9.1.60.

[76] G. Lance and W. Williams. "Mixed-Data Classificatory Programs I – Agglomerative Systems". In: *Australian Computer Journal* 1 (Jan. 1967), pp. 15–20.

[77] Giuseppe Jurman, Samantha Riccadonna, Roberto Visintainer, et al. "Canberra Distance on Ranked Lists". In: Advances in Ranking – NIPS 09 Workshop, Dec. 2009, pp. 22–27.

[78] Syed Emran and Nong Ye. "Robustness of Chi-square and Canberra distance metrics for computer intrusion detection". In: *Quality and Reliability Engineering International* 18 (Jan. 2002), pp. 19–28. DOI: 10.1002/qre.441.

[79] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), p. 22.

[80] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[81] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[82] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[84] Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[85] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.