UPPSALA
UNIVERSITET

UPTEC STS 22020

Examensarbete 30 hp

Juni 2022

Interpretable Outlier Detection in Financial Data

Implementation of Isolation Forest and Model-Specific Feature Importance

Kasper Knudsen
Vilhelm Söderström

Civilingenjörsprogrammet i system i teknik och samhälle

UPPSALA UNIVERSITET

Interpretable Outlier Detection in Financial Data

Kasper Knudsen
Vilhelm Söderström

Abstract

Market manipulation has increased in line with the number of active players in the financial markets. The most common methods for monitoring financial markets are rule-based systems, which are limited to previous knowledge of market manipulation. This work was carried out in collaboration with the company Scila, which provides surveillance solutions for the financial markets.

In this thesis, we will try to implement a complementary method to Scila's pre-existing rule-based systems to objectively detect outliers in all available data and present the result on suspect transactions and customer behavior to an operator. Thus, the method needs to detect outliers and show the operator why a particular market participant is considered an outlier. The outlier detection method needs to implement interpretability. This led us to the formulation of our research question as: How can an outlier detection method be implemented as a tool for a market surveillance operator to identify potential market manipulation outside Scila's rule-based systems?

Two models, an outlier detection model Isolation Forest, and a feature importance model (MI-Local-DIFFI and its subset Path Length Indicator) were chosen to fulfill the purpose of the study. The study used three datasets, two synthetic datasets, one scattered and one clustered, and one dataset from Scila.

The results show that Isolation Forest has an excellent ability to find outliers in the various data distributions we investigated. We used a feature importance model to make Isolation Forest's scoring of outliers interpretable. Our intention was that the feature importance model would specify how important different features were in the process of an observation being defined as an outlier. Our results have a relatively high degree of interpretability for the scattered dataset but worse for the clustered dataset. The Path Length Indicator achieved better performance than MI-Local-DIFFI for both datasets. We noticed that the chosen feature importance model is limited by the process of how Isolation Forest isolates an outlier.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Gustav Tano (Scila) & Isak Olsson (Scila) Ämnesgranskare: Stefan Engblom

Examinator: Elísabet Andrésdóttiír

Populärvetenskaplig sammanfattning

I och med en ökad aktivtet på de finansiella marknaderna har har också antalet av finansmarkndasbrott ökat i en liknande takt. Finansmarknadsbrott genomförs genom att en, eller flera aktörer tillsammans på olovligt sätt påverkar prisättningen av ett värdepapper till egen vinning men på andra aktörers bekostnad. För att bekämpa denna typ av brottslighet krävs det att misstänkt aktivtet upptäcks och då mängden transaktioner som genomförs inom det finansiella systemen dagligen är väldigt stor blir upptäckten av manipulation ett krävande uppdrag som kräver speciella system framtagna för denna uppgift.

Scila är ett fintech bolag som levererar en marknadsövervakningsprodukt till olika kunder världen över. Deras grundsystem bygger på inbyggda regler och tröskelvärden som baseras på historik från olika sorters fall av marknadsmanipulation. Detta innebär att systemet inte kan upptäcka nya sorters marknadsmanipulation, samt kräver regelbunda uppdateringar av regler och tröskelvärden till nya marknadsförhållanden. I denna studie undersöktes hur en alternativ metodik av upptäckande av misstänkt aktivitet kunde implementeras för att komplementera den tradionella metodiken i Scilas system. Metoderna tittar istället efter ovanliga mönster i data som i sin tur eventuellt kan tyda på olovlig aktivitet. Ett stort problem med dessa metoder, vilket även är en av huvudanledningarna till varför de inte används i stor utsträckning idag, är att tillskillnad mot de klassiska systemen inte tydligt visar varför en transaktion flaggats som misstänkt. Med detta som bakgrund undersöktes därmed hur man för vald modell kunde implementera så kallad "tolkbarhet", så att man med mänsklig granskning av datapunkten kan bestämma om den är värd att undersöka närmare eller inte.

Efter en genomgång av olika metoder inom fältet samt granskning av datan modellen ska tillämpas på, valdes metoden Isolation Forest, samt två påbyggnadsmetoder för ökad tolkbarhet, MI-Local-DIFFI och Path Length Indicator. För att på ett enkelt och tydligt sätt kunna utvärdera prestandan av modellerna skapades två syntetiska dataset. Datasetten skapades med två olika strukturer (ett glest och ett klustrat) och innehöll båda så kallad "ground truth", något som saknades i orginaldatan. Detta innebar att vi visste vilka datapunkter som var outliers samt varför dessa var outliers. Med dessa två dataset kunde vi därmed utvärdera hur väl modellerna presterade.

Våra resultat visar på att Isolation Forest har en utmärkt förmåga att hitta anomalier i de olika datastrukturerna vi har undersökt. Dessutom har resultaten en relativt hög grad av tolkbarhet för det glesa datasettet men sämre för det klustrade datasettet. Path Length Indicator presterade bättre än MI-Local-DIFFI för båda datasetten. Vi kunde även konstatera att förklaringsmodellen som metod var begränsad av hur Isolation Forest isolerar en anomali.

Contents

1 Introduction

The trend of increased activity in the securities market persists. With approximately 227 million annual transactions, 2021 was a record year for Nordic securities trading in terms of activity, and since the year 2000, the number of executed orders has increased elevenfold [37]. Unfortunately, with increased activity in the market and an increased interest in the securities market, the frequency of financial market crimes also grows [38]. This includes offenses of the categories such as *insider trading* and *market manipulation*. Insider trading means that a participant trades securities with the help of insider information, e.g. unpublished information about a listed company. Market manipulation includes trading and actions that give false or misleading supply, demand, or price signals. This study will focus on market manipulation.

The approaches to market manipulation vary and may involve manipulation through information dissemination, with *pump and dump* as a standard method. To inflate the price of a security, one or several participants together boost the sentiment surrounding the security using positive statements on social media or similar. When the public is affected to buy securities, the price rises, and the scheme operators can sell at a profit. Another type of manipulation is based on the actual trading and order placement with *momentum ignition* and *spoofing (layering)* as typical methods. The actual carry-out differs between the methods. Still, the basic principle is the same, via different types of ordering techniques in an artificial way make the price of a security rise [36]. Market manipulation, primarily the types based on the actual trading and order placement, can be detected by monitoring orders and closing lists.

Today, all stock exchanges, trading platforms, and persons who professionally carry out transactions in Sweden are subject to the EU-regulated Market Abuse Regulation (MAR). The market participants are obliged to report to Finansinspektionen (FI) if they discover orders or transactions that may be suspected to be of the type of financial market offenses, and monitoring is a requirement [18]. Monitoring is, in most cases *rule-based*, meaning that pre-programmed alarms based on expertise and historical data are triggered if certain "thresholds" are met. These systems have proven to be quite effective, but they still suffer from drawbacks, such as the risk that manipulators gain knowledge of these systems, and how to circumvent them. Rule-based alarms also pose the risk of missing activity since the rules are not based on all the customers' available data but only on selected parts [5].

A solution to this is to use an alternative way to monitor trade data more objectively by using machine learning to perform outlier detection. The basic principle for outlier detection is to differentiate between what classifies as normal in a dataset and what classifies as abnormal. This same principle is also used in rule-based monitoring, where the threshold determines whether an activity classifies as normal or not normal. The difference is in how to determine the classification itself. Market manipulation deviates from the expected behavior of a participant, and rule-based systems try to find these deviations. Rule-based systems use combinations of static thresholds to find these deviations, while machine learning classifies normal depending on the structure in the actual dataset. In contrast, the classification is dynamic and may detect new market manipulation techniques, because the classification is not based on specific parts of the data where rule-based thresholds exist, but on all available data [19].

However, a general problem for machine learning solutions, which are not found in rule-based systems, is the interpretability of the methods. The rule-based system clarifies why a transaction is classified as market manipulation since it shows which thresholds have been crossed. On the other hand, machine learning solutions often only generate a result, and it is not apparent to the user why a data point is classified as it is. The absence of interpretability is a general problem for machine learning and artificial intelligence (AI), and it results in a lack of confidence (trust) from the users as they do not understand why methods provide a particular output. Because of this issue, machine learning solutions are not used to the extent that they could be [28]. This highly topical issue has led to the emergence of a new field in machine learning and AI called *eXplainable Artificial Intelligence (XAI)*. The goal of this field is to make so-called black-box machine learning models easily understandable to humans [21].

1.1 Collaboration

This work has been carried out in collaboration with the company Scila. Scila is a Stockholm-based fintech company that provides specific solutions for market surveillance and compliance to banks and security exchange customers within the finance and energy sectors worldwide. The product range includes anti-money laundering (AML) detection systems and market surveillance solutions of all asset classes for traditional market abuse scenarios such as spoofing, layering, insider trading, and high-frequency trading. As with most market monitoring solutions, Scila's product solutions primarily comprise rule-based systems. Therefore, these warning systems are limited to fixed rules, and Scila currently has large amounts of customer data that are not actively analyzed. Although no alarms have been triggered, there may be undiscovered potential market manipulations.

1.2 Purpose

As mentioned, there are several ways for market manipulators to circumvent existing rules in a rule-based system. In this thesis, we will try to implement a complementary method to Scila's pre-existing rule-based systems to objectively detect outliers in all available data. The model should not be a part of the alarm system but instead be used as a complementary tool for an operator to get clues on suspect customer behavior. Thus, the method needs to detect outliers and show the operator why a particular market participant is considered an outlier. The outlier detection method needs to implement interpretability.

- Research question: How can an outlier detection method be implemented as a tool for a market surveillance operator to identify potential market manipulation outside Scila's rule-based systems.

In order to answer the research question more precisely, we formulate these two sub-questions:

- Sub-question 1: What type of outlier detection method is appropriate to fulfill the purpose of the study?

- Sub-question 2: To what degree can the chosen outlier detection method achieve a level of interpretability that enables interpretation through human expertise?

1.3 Structure of the Thesis

In Section 2, the reader is introduced to basic theory in outlier detection and its main concepts. The information presented then acts as basis for Section 2.4, where we, linked to the purpose, argue for our method choices. Background is followed by Section 3, here we visualize the data from Scila and describe the pre-processing, we also present two synthetic datasets created for validation purposes. In Section 4, we present the two methods, Isolation Forest and MI-Local-DIFFI, selected to achieve the purpose of the study. We also presents the structure of the experiments. In Section 5, we validate the methods on synthetic data and then visualize the model's results on the Scila dataset. Section 6, addresses the relevant results and their implications. In Section 7, we present the findings of the study by answering our initial questions with the achieved results and discuss what can be investigated further to potentially improve the model and the understanding of the model. Lastly, in Appendix A we present the full Table with outlier detection results, and in Appendix B we present our code for the implementations of Isolation Forest and MI-Local-DIFFI.

2 Background

This section will review the literature, including the basics behind outlier detection and its most common methods, which will explain the more specific models presented later in the paper. We will also present a formal definition of explainability and interpretability in machine learning, which will build a theoretical foundation for understanding *feature importance*.

2.1 Outlier Detection

The term *outlier detection*, also referred to as *anomaly detection*, refers to the identification of events or observations that deviate significantly from the majority of a dataset, so-called *outliers* or *anomalies* [12]. The discovery of these observations is of great importance in various domains. In healthcare, for example, outlier detection is used to detect anomalies that may indicate illness/diseases [50]. In cybersecurity, anomalies of traffic patterns in computer networks can indicate unauthorized access [15]. The concept of outliers is divided into *local and global anomalies*, where the two differ in terms of the scope. To be classified as a local outlier it is sufficent to be an outlier in comparison to the neighboring data points whilst a global outlier is an outlier to all data points in the dataset [19].

Outlier detection is not a new concept and has been central in statistics for hundreds of years. In statistics outliers are defined as data of low probability in the assumed distribution $P(x)$ of a phenomenon and its descriptive data $\{x_1, x_2, ..., x_n\}$. The problem with this methodology is that in real situations it is often challenging, and many times impossible, to define the probability distribution $P(x)$, describing the data. Today, the field of outlier detection has one leg in statistics and the other in machine learning, where the latter includes "algorithmic driven" methods that arose as a result of dealing with the problem that followed the statistical methods [52].

An important measure in machine learning is the degree of "supervision" used during the training stage. Depending on this degree, the method is classified as *supervised*, *unsupervised* or *semisupervised*. The concept is essential in outlier detection as well, since many outlier detection methods are machine learning methods [12].

Supervised outlier detection requires a classification of the training data as "normal" or "abnormal," so-called supervision. Based on this classification, a model generates labels for unseen data points. More formally, the concept can be described as:

$$min[f(x) - y], \tag{1}$$

where f is a predictor, based on the input data x trained to predict the correct output y.

Unsupervised outlier detection on the contrary, trains a method on data that lacks labeling and information about the desired output labeling, also called *ground truth*. *Semisupervised outlier detection* methods are a mixture of these two and assume that part of the training data is labeled, but the more significant part is not [12]. Unsupervised methods are the most commonly used methods in outlier detection. This is primarily due to "the nature of the data" meaning that in many phenomena examined, there is often a lack of

labeled data. The general lack of labeled data is because labeling by the usage of human expertise is costly in terms of work and time [10].

2.2 Methods in Outlier Detection

As mentioned in the previous section, outlier detection methods can be either of statistical or algorithmic nature and have varying degrees of supervision. Since all of these methods have varying ways of measuring the "outlierness" of observations and thus work differently with different datasets, this section aims to break down these methods further and briefly review the different primary categories to motivate our choice of method. The methods vary in terms of output. Some methods generate a score, representing the probability of being an outlier. Others generate a "binary output" in the form of a label whether the data point is an outlier or not. The advantage of the "scoring approach" is that the score can rank outliers, and if the desired result is a binary form, it can be achieved by defining a threshold on the score [26].

2.2.1 Model-Based

The model-based methods function by creating a model representing the normal data in a dataset, and observations that do not fit into the model categorizes as outliers. The previously mentioned statistical methods, and Neural Networks, belong to the model-based category.

1) *Statistical Methods:* As described in section 2.1, statistical methods aim to find a probability distribution $P(x)$ estimated by deduction of model parameters λ which describes the dataset X. Outliers are those data points with the lowest probability generated by the distribution $P(\lambda \mid x)$. This methodology is called *parametric techniques* and is the traditional way of looking at statistical methods [16]. However, as also described in section 2.1, a significant problem with this methodology is finding a representative probability distribution for the dataset, something crucial for the method's performance. There are ways to handle this by using so-called mixture models on datasets where it is not easy to find the proper distribution. Mixture models use different distributions to fit the data better. As a consequence, however, the models become more complex, and the problem remains to a certain extent [1]. Parametric techniques also struggle with high-dimensional data since it make the parameter estimation problematic [19]. There is another category of statistical methods that, unlike parametric techniques, does not use predefined parameters to define a model. These methods are called *non-parametric techniques* and the strength of these methods is that they do not require an assumption of the data as parametric techniques since they are built directly from data. Typical non-parametric techniques use histogram techniques or kernel functions. Common to these is that they both, like parametric techniques, have difficulty handling high dimensional data [19].

2) *Neural Networks: Artificial Neural Networks* are another model-based method inspired by the characteristic functions of the human brain. The similarity can be seen in the input signals to so-called neurons, which in Neural Networks consist of relative weights affecting the impact of the inputs in a fashion similar to our biological counterparts. The inputs are then summed with a bias subsequently added.

The bias intercepts a linear equation, allowing the model to fit the best input data. Lastly, an activation function is applied to determine whether a neuron should be activated or not, which can create a multi-layered network. Without it, the output would be a linear transformation of the input. Layers can be considered containers for the neurons and are divided into input, hidden, and output layers. The input layer brings the initial data into the system to be processed by multiple hidden layers, which execute the process described before. Each hidden layer can be specific and use different transformations for input data, where the result of all hidden layers is visualized by the output layer in the end [33].

2.2.2 Distance-Based

Distance-based outlier detection methods classify data points based on the distance to their neighboring points. The general idea is simple. If a data point is close to its neighbors, it is classified as a normal data point, while if it is far away from its neighbors, it is classified as an outlier. Euclidean distance is the most frequent measurement used to measure distances, with the requirement of continuous variables [32]. Distance-based methods are the most common outlier detection category, and the most well-known distance-based algorithm is the k-nearest neighbor (kNN) [26]. A significant problem for these methods is handling high-dimensional data since the distance between all pairs of data points usually must be calculated, resulting in high time complexity, $O(n^2)$ [28].

2.2.3 Density-Based

The general idea behind density-based methods is to compare the density around a point with its local neighbor points. The points whose density is similar to the density of the neighbor points classify as normal. In contrast, if the density around a point is distinctly different from its neighbors, it is classified as an outlier. Several different density-based methods are all based on this fundamental principle, but most differ in how they estimate density [26]. One of the more commonly used density-based methods is the Local Outlier Factor (LOF), which produces an outlier score based on how much the relative density of a data point differs from the density of its local neighbors [48]. However, this method has its limitations, and many extensions intend to solve some of these limitations. Since the fundamental principle in density-based methods relies on computing the density for all points, the distance must be calculated in all dimensions, which quickly becomes complex with an increasing number of dimensions since the time complexity is $O(n^2)$. Density-based models, therefore, struggle with high-dimensional data. However, for low-dimensional datasets, the time complexity may be reduced to $O(n)$ or $O(n \cdot log(n))$ [5].

2.2.4 Deviation-Based

Deviation-based methods are closely related to statistical methods. The basic idea is the same. The outliers are the points that do not fit with the general characteristics of the dataset. However, for deviation-based methods, the detection of these outlier points differs, and outliers are discovered by deleting points and then studying how the variance of the dataset changes. If the variance decreases, the point may be an outlier and vice versa. This methodology assumes that the outliers are the furthest points of the dataset,

which means that if the outliers are in the middle of a dataset, they are not detected. In addition, the method is not very efficient, and the time complexity is high, $O(2^n)$ [26].

2.2.5 Clustering-Based

Clustering methods aim to identify groupings in datasets where the data points in each cluster share more similarity than data points in other clusters [47]. Several different clustering method variants exist, with primarily three different approaches. The first method assumes that normal data points in a dataset belong to large dense clusters while outliers belong to small, sparse clusters. The second approach relies on the idea that outliers are the points that do not belong to any cluster. The third approach uses the centroids, the geometric midpoints of each cluster, to classify points. Normal points are the points closest to each cluster's centroids, and outliers are defined as points far from the centroids. The most common cluster algorithm k-means belongs to this category [19]. The general complexity of clustering methods depends on the clustering algorithm used. However, clustering-based methods, like distance and density-based methods, often are instance-based, where the distance between various points must be calculated. Nevertheless, for cluster algorithms, usually, only the distance to the cluster center has to be calculated for each data point instead of the entire dataset. This results in a low complexity limited to $O(n)$ [5].

2.2.6 Isolation-Based

Many of the presented models and most of the available outlier detection methods rely on detecting and classifying what is normal data in the dataset and then classifying the abnormal data as outliers. An alternative way is to reverse it and directly isolate outliers rather than profiling normal cases first. As a result of this approach, these methods are optimized for profiling outliers instead of inliers. Two clear advantages over the classic approach are: the method becomes more accurate due to fewer false classifications and fewer actual deviations missed, due to directly targeting outliers. In addition, the method can handle much larger datasets and higher dimensions because of the small calculation requirements. Thus, the complexity is greatly reduced as the method does not focus on all data points but directly focuses on a few abnormal points [28]. There are primarily two disciplines within isolation-based methods, tree-based and nearest neighbor-based, with the former being the most prominent and widely used. Tree-based isolation methods will be described in further detail in section 4.

Table 1: Summary of categories for outlier detection methods for n data points.

Method	Description	Example	Complexity (General)	Output
Model-based	Outliers are the points that do not fit into the model created to represent the data classified as normal.	[43]	Method dependent	Binary or score
Distance-based	By calculating the distance between all points, the point that is far away from its neighbors is classified as an outlier.	[24], [42]	$O(n^2)$	Score
Density-based	Compare the density around a data point with the density around its local neighbors. Points whose density differs from the density around its neighbors have a high probability of being outliers.	[7], [49]	$O(n^2)$	Score
Deviation-based	Outliers are detected by deleting points and then studying how the variance of the dataset changes for those points.	[2]	$O(2^n)$	Binary
Clustering-based	Outliers in clustering algorithms are the points that do not belong to the large and dense clusters in the dataset.	[22], [17]	$O(n)$	Binary or score
Isolation-based	Isolation-based methods try to directly isolate outliers by profiling what an outlier should be and then isolate those points instead of looking at which points are deviating from the normal points.	[28], [3]	$O(n)$	Score

2.3 Explainability

So far, we only discussed outlier detection methods, but in agreement with the problem presented in section 1, these methods only generate an outlier score or binary output. It does not explain why a data point classifies as an outlier, and there exists no interpretive value, except for statistic methods. We will, therefore, in this section briefly present basic concepts and relevant methods from the field of *eXplainable Artificial Intelligence* (XAI) whose goal is to make so-called *black-box* machine learning models understandable to humans [21].

In order to be able to present the subject, it is crucial to sort out and define two critical

concepts within the area of *interpretability* and *explanation*. To do so, we will use the definitions in the article [35] and their notations to explain these concepts.

Definition 2.1. Interpretability: An interpretation is the mapping of an abstract concept (e.g., a predicted class) into a domain that the human can make sense of.

Definition 2.2. Explanation: An explanation is the collection of features of the interpretable domain that have contributed for a given an example to produce a decision (e.g., classification or regression).

The result of an explanation method, which explains the result of a machine learning model, is often a relevance scores which indicates to what extent features contributes to a specific result [35]. The methods, can be *model-specific* or *model-agnostic*. These concepts explain whether the result is based on the internal structure of a specific model or whether the method can be applied to any model, by analyzing the input and output relationships of the features. Thus, the advantage of model-agnostic methods is that they are universal, and the disadvantage is that they often have high computational complexity. The high computational complexity originates from the fact that these methods have to simulate many different outcomes and operate after the outlier detection has already taken place, meaning additional computations. Another important categorization for explanation methods deals with the question of whether the explanations are local, explanations of individual predictions, or whether they are global and thus explain the entire model's behavior [34]. Another important concept related to explanation is how intelligible the machine learning algorithm is. *Intrinsic explainability* means the algorithm itself is easily understood, and thus, *post-hoc* explainability is not necessary [34].

2.4 Method of Choice

The chosen outlier detection method for our thesis should be able to fulfill some fundamental characteristics, i.e., operating successfully on the given data structure. The exact design of the data is described in more detail in section 4, but in short, it should have the ability to handle unlabeled numerical data with > 10,000 samples and > 40 dimensions. Therefore, the chosen method needs to detect outliers in unlabeled, large, high-dimensional datasets and return an outlier score for each observation. With these requirements in mind, we could directly exclude some method categories that do not meet these requirements. Methods excluded will be those not performing well with high-dimensional data, methods of supervised nature, and methods returning a binary output. Reviewing Table 1, we can directly exclude deviation-based, distance-based, and density-based methods because of their high time complexity or binary output.

The model is planned to be able to run on aggregated data on a daily basis and not in real time. Therefore the time complexity is not of primary interest since it will not need to produce instantaneous results. However, it is still advantageous to use a model with low time complexity, considering unpredictable amounts of data samples and time frames. Concluding what has been discussed, isolation-based, clustering-based, and Neural Networks should best suit our dataset. However, clustering-based methods suffer from robustness problems since generated clusters differ with every run and often require multiple runs for a relevant average outlier score. Among isolation-based methods, Isolation Forest is the

most prominent and widely used algorithm.

A study [41] compares Isolation Forest and Neural Networks, specifically the two popular Neural Network methods, Auto-encoder and Restricted Boltzmann Machine (RBM). It can be stated that Isolation Forest generally performs equally or better than Auto-encoder and RBM, especially when looking at run time and its properties of tuning hyperparameters, which is essential given the unsupervised nature of our data. Apart from this study it should be mentioned that Neural Networks have been successfully used in many studies of outlier detection. Nevertheless, one of the primary issues with Neural Networks is the difficulties with interpretation, i.e., black-box, which opposes our purpose to achieve an interpretable result following XAI [51]. A study from 2018 [14] examines several state-of-the-art outlier detection methods to find the advantages and disadvantages of the methods. The study benchmarked precision, robustness, computation time, and memory usage in 14 different unsupervised outlier detection algorithms on synthetic and real-world datasets. The results from the study showed that Isolation Forest effectively and with good precision identified outliers. At the same time, Isolation Forest had excellent scalability on large datasets where memory usage was acceptable for up to one million samples, and it had the highest total score of all the unsupervised algorithms evaluating the benchmarks. Lastly, Isolation Forest works well in the case when several features are irrelevant, which is an essential trait when dealing with unlabeled and high-dimensional data. With this in mind, we consider the characteristics of Isolation Forest as appropriate to fulfill the purpose of the study.

A drawback of the Isolation Forest algorithm is that the method is not intrinsically explainable. The lack of interpretability is mainly due to the number of trees in the Isolation Forest, so post-hoc explanations in the form of feature importance algorithms are necessary [5]. However, the tree structure allows for good efficiency of the isolation-based methods and allows for, in theory, quite simple implementation of model-specific methods. The tree structure is favorable as those methods are not as computationally demanding as model-agnostic methods. In addition to being a model-specific method, our desired method should yield results in terms of local explanations as we are looking to analyze individual predictions. With this requirement in mind, we found the extension method *Multiple Indicator Depth-based Feature Importance for the Isolation Forest (MI-Local DIFFI)* as the most suitable method. The method will be described in further detail in Section 4.

Before going deeper into our two main methods, we will now introduce the necessary basics from graph theory and tree theory because understanding the tree structure is essential to understanding the function of Isolation Forest and MI-Local-DIFFI.

2.5 Graph Theory

A *tree* is defined as an undirected graph where two nodes connect via exactly one edge. One node will be assigned as the root node in a rooted tree, where the root node has no parents. In a rooted tree, a parent to node v is the node linked to v on the path to the root, and a child to v is a node with v as parent. A node without children is called a leaf. An internal node is a node that has children, excluding the root, and an external node is a node without children. A leaf is, therefore, an external node. If a tree t_i has a finite amount of nodes n, the amount of edges for the tree will be n-1 [20]. A path in a tree is a sequence

of edges of the form (v_1, v_2), (v_2, v_3), ..., (v_{n-1}. V_n) [44].

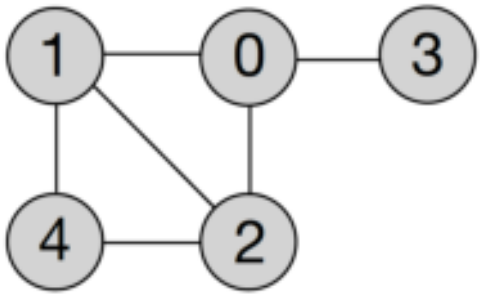


Figure 1: An undirected graph consisting of nodes V = {0,1,2,3,4} and edges E = {{0,1}, {0,2}, {0,3}, {1,2}, {1,4}, {2,4}}.

A binary search tree is a rooted tree structure where one node is assigned the root. The left sub-tree contains only keys of a lower value than the root key. The right sub-tree contains only keys of a higher value than the root key. Accordingly, the left and right sub-trees must be binary search trees. The tree is built based on the root node's value, so if the value is less than the root node, we search for an empty space in the left sub-tree and vice versa. A *full binary tree* is a tree where all non-leaf nodes have exactly two children. A leaf thus always has zero children, and therefore all internal nodes have two children, and all external nodes have zero children in a *full binary tree* [4].

2.5.1 Decision Tree

A decision tree is a sequential model that combines logical sequences based on simple tests to create a decision support tool, where each test compares a numerical attribute to a threshold value. A decision tree classifies data points by asking questions about the feature concerning the node and the data point. Every node contains a single question, and every single internal node points to a child node for every possible answer to the question. Therefore, the questions become hierarchical, which can be visualized in a graphic form of a tree. When creating a decision tree based on the training data, question nodes are added incrementally with the purpose of trying to label the classes in the training set correctly. Questions are thus recursively selected to split training items into smaller subsets combined into one tree. The best question for each node is chosen by maximizing the information gain or minimizing the impurity. The optimal scenario is to achieve as few splits as possible to create homogeneous class labels. The measure used in decision trees to measure impurity in items is usually entropy and Gini index [23]. Since decision trees require labeled data to function, it is classified as a supervised method. A data point that is to be classified follows the path from the root node, through the internal nodes, which have a "yes" and a "no" child, until the point reaches a leaf node. The data point will then be assigned to the class associated with the leaf node where it ends up. As a consequence it is easy to understand why a point is classified as it is [25], which puts decision tree in the category of intrinsic explainable algorithms, where the algorithm is easily understood itself, a frequent property among supervised algorithms.

A disadvantage of decision trees is that they are prone to overfitting because the model depends on the training data. Overfitting can be counteracted by applying techniques such

as random forest, which uses an ensemble of decision trees [19].

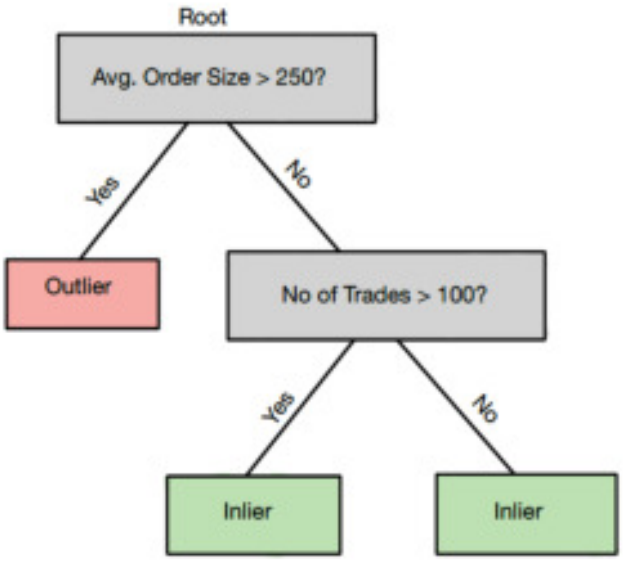


Figure 2: A simple decision tree classifies outliers and inliers by supervised learning and training on labels.

2.5.2 Random Forest

Random forest is a well-used further development of decision trees that build each tree using bootstrap samples. A random forest contains multiple decision trees where each tree has randomly sampled data with replacements from the original dataset (this technique is known as bagging) [46]. Every single decision tree in an ensemble contains a sample with replacement from the training data with bagging. Every single tree in the ensemble acts as a classifier to determine the class label of an unlabeled data point. The labeling utilizes majority voting, where each classifier votes for its own predicted class label. Then the class label with the most votes from the different classifiers is used to classify the data point [6]. The principle behind random forest is thus, to utilize the law of large numbers to address the issue of overfitting. We avoid overfitting by random forests majority voting structure which results in a strong learner by combining a large number of weak learners. Many algorithms further develop decision trees, however, the random forest algorithm has proven to be one of the better when compared to other methods such as boosting or Naive Bayes [11].

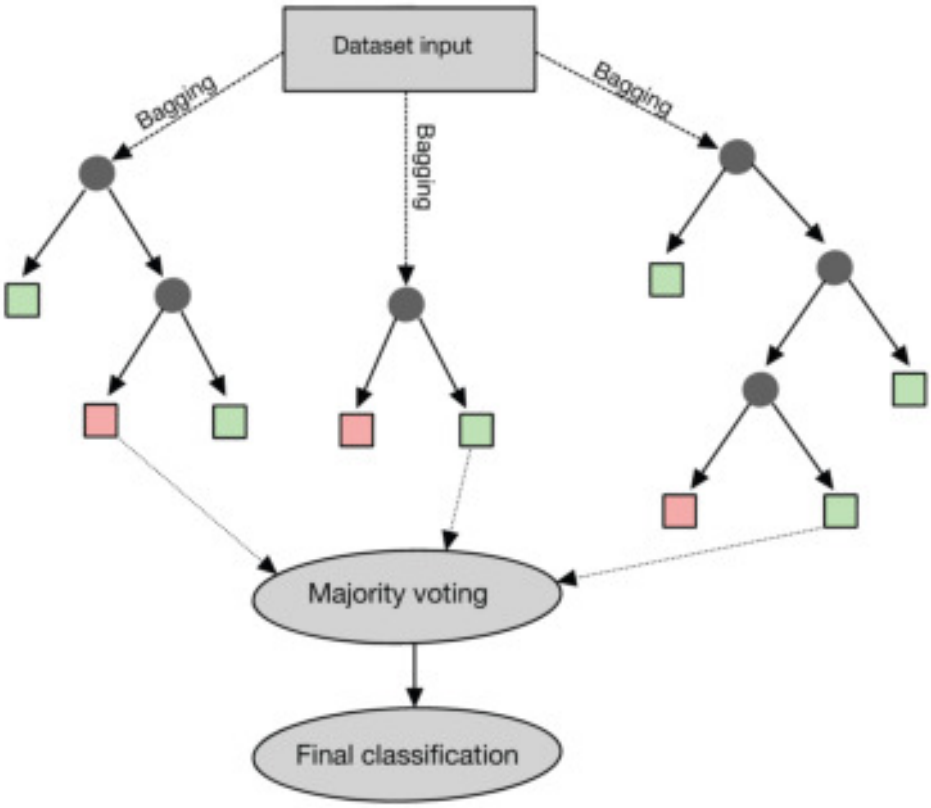


Figure 3: A random forest with three trees and a max depth of three.

2.6 Evaluation Metrics

To measure how well a model performs on different data we will be using some well known metrics to gain feedback from our experiments. The different scoring metrics will enable us to gain an understanding of the model's performance for different data distributions, and its sensitivity when varying the input parameters. In other words it helps us understanding some essential parts when creating a model within the field of XAI.

2.6.1 Confusion Matrix

The confusion matrix is a table used to evaluate the performance of classification models when the true values are known. The data is split into four elements; true negative, false positive, false negative, true positive, see Table 2 [9].

Table 2: The confusion matrix is a summary of prediction results on a classification problem.

		Predicted	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Now we will present some common measures used in connection with the confusion matrix, and use its four elements to evaluate performance in different ways.

2.6.2 Accuracy

Accuracy is simply a measure of how often the classifier makes correct predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

2.6.3 Recall

Recall, also called the true positive rate (TPR), measures how many data points of positive class are predicted as positive. Recall is good when we want to measure the classifiers ability to catch positives.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

2.6.4 Precision

The precision tells us how many predictions are actually positive out of all the positives predicted. It is useful when the rate of false positives is of higher concern than false negatives.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

2.6.5 Area Under the Receiver Operating Characteristics

AUC is abbreviation for Area Under the Curve and is based on the ROC, Receiver Operating Characteristic curve. It is the result of plotting the trade-off between True Positive Rate (TPR), also called recall, and False Positive Rate (FPR) for different thresholds, which equals the probability a classifier ranks a positive observation higher than a negative one when chosen randomly. A ROC curve begins at (0, 0) and ends at (1, 1). A random classifier would have a straight line between these two points. A good classifier will extend above this straight line, and a perfect classifier would completely follow the y-axis and the x-axis (see Figure 4) [8].

Even though the AUC is a good performance measurement of a classifier, it suffers from some drawbacks. One is its ability to evaluate imbalanced data, i.e., data where the distribution between the occurence of the classes is significant, e.g. one class only occurs in up to 10% of the data. The curve of AUC risks being swamped by the proportion of TN's, which leads to the AUC being potentially high for bad classifiers, because of the large amount of TN's, producing a misleading score. Similar to AUC, Area Under the Precision-Recall Curve (AUPRC) is a well known measure commonly used for imbalanced datasets, which are often found in outlier detection. It is based on the trade-off between precision versus recall for different thresholds [29]. AUPRC does not include TN and instead looks at how the classifier handles positives, making it more suitable for

imbalanced data.

AUPRC uses recall and precision, as mentioned above. Recall is defined by how well the model classifies positives as positives and can be found in the AUC as well. Precision is only found in the AUPRC and is defined by measuring how the classifier manages to not classify negatives as positive, used in cases when the cost of false positives is high. The plot of the precision-recall curve begins with each observation classified as negative, i.e., at the point where recall = 0 and precision = 1. Like the ROC curve, the threshold changes until every instance is classified as positive. An optimal result is a square with sides of one, and a perfect AUPRC score is thus also equal to one. With such a result, there exists a threshold where the model can find all positive instances, i.e. perfect recall, without classifying negative instances as positive, i.e. perfect precision [45].

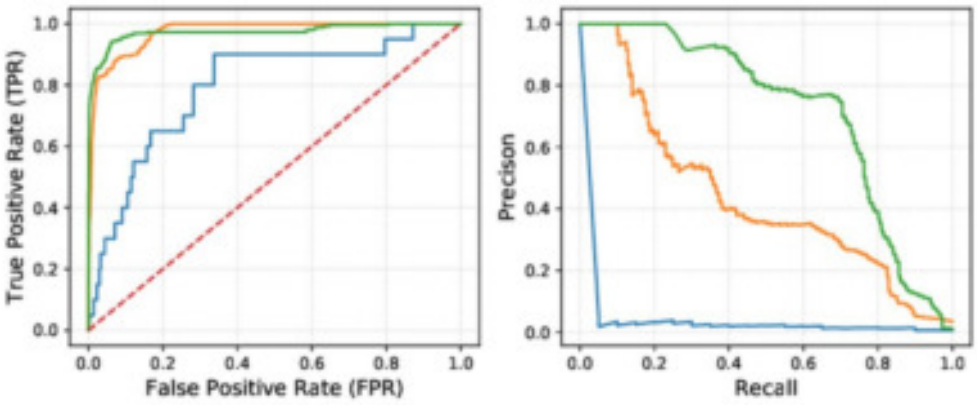


Figure 4: An illustration of the ROC curve (left) and Precion-Recall curve (right) for different classifier results.

3 Data

The goal of this section is to in detail describe the datasets used in the study: one dataset from Scila and two synthetic datasets created by us for validation purposes. We will describe the process of how we utilized t-SNE to visualize and gather knowledge about the distribution of the Scila dataset and how the result were used as basis for the synthetic datasets. We also describe the necessary means we took to process Scila's data before the models described in the previous section could be applied.

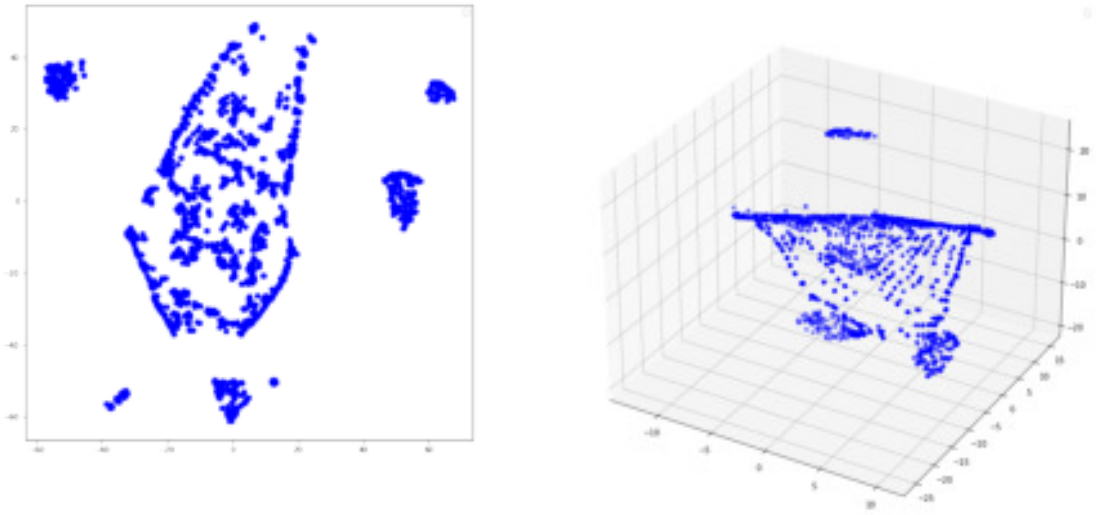
3.1 The Scila Dataset

The primary dataset in this study is from Scila and thus the type of dataset the model will operate on in production use. It contains 37 features, 4 categorical and 33 numerical. It contains 86760 observations distributed over different order books, where the proportion of observations per order book varies. The data contains daily-aggregated information at participant-level in securities such as BTC/USD and AAPL (Apple). Table 3 shows an example of this structure. For outlier detection purposes, it is only interesting to compare the participant's activity in the same security since what can be considered normal activity depends on the underlying security, and comparing different securities would yield incorrect results. Therefore, the model will run on sub-datasets, one security at a time.

Table 3: Structure of the Scila Dataset.

Date	**Trader**	**Security**	**No. of Orders**	...	**Avg. Trade Size**
2022-01-15	S**	AAPL	34	...	1245
2022-01-16	O**	AAPL	22	...	567
⋮	⋮	⋮	⋮	⋮	⋮
2022-03-12	E**	ERIC,A	49	...	40
2022-01-15	F** Holdings	IBM	27	...	1372

An essential part for us to be able to utilize the chosen methods and tuning them in the best possible way, was understanding the structure of the data from Scila. This allowed us to construct suitable synthetic datasets for validation and parameter tuning, since the Scila data lacked ground truth. Because of the data's high dimensionality, it was impossible to visualize it as it is. Therefore we utilized a dimension-reducing methodology t-Distributed Stochastic Neighbor Embedding (t-SNE) via its implementation in Python's scikit learn to visualize the data in 2D and 3D. The basic principle of t-SNE is to calculate similarities between observations through the distance between them and assign diverse observations a low probability through equal distribution and equal observations a high probability. Then the observations from the high-dimensional data are projected to a lower dimension, and a similar probability distribution across the points in the low-dimensional map is defined. The goal of the method is then to minimize the Kullback-Leibler divergence (KL divergence) between the two distributions. It is important to note that the result of the t-SNE will never give an error-free image of the data, because t-SNE reveals only select parts of the data structure. The method creates a rough map of the data, were certain structures can be captured, and not the actual data. Some structures can never be reflected in a low-dimensional map [30].



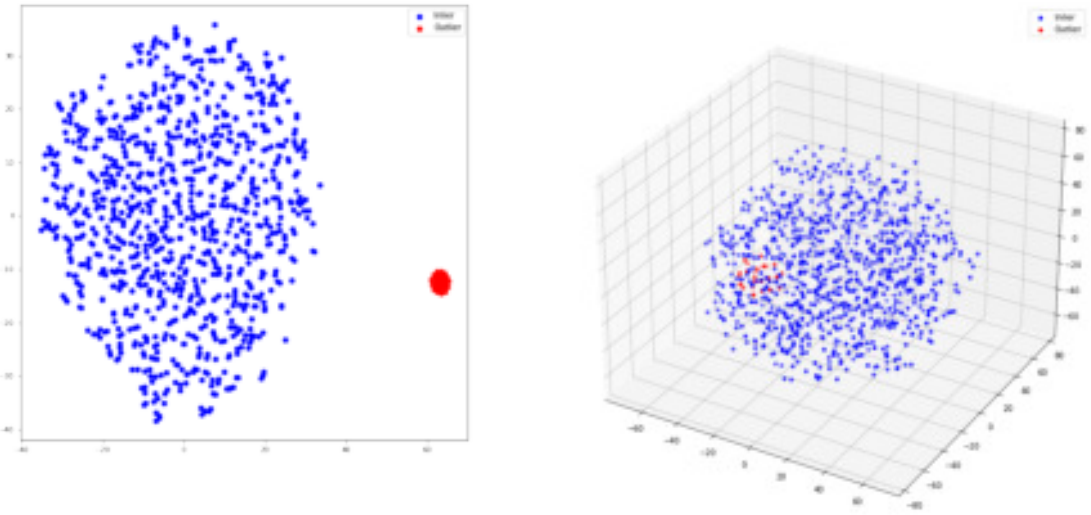
(a) BTC/USD in two dimensions. *(b) BTC/USD in three dimensions.*

Figure 5: Sample from the Scila dataset (BTC/USD) visualized with t-SNE.

Figure 5 displays the visualization of BTC/USD in two and three dimensions. From those we can observe several dense clusters of observations and the majority of the observations being located in some kind of midpoint of the data. This indicates that the dataset overall most likely has a clustered structure. These conclusions about the structure of the data will form the basis for constructing the synthetic datasets in Section 3.2.

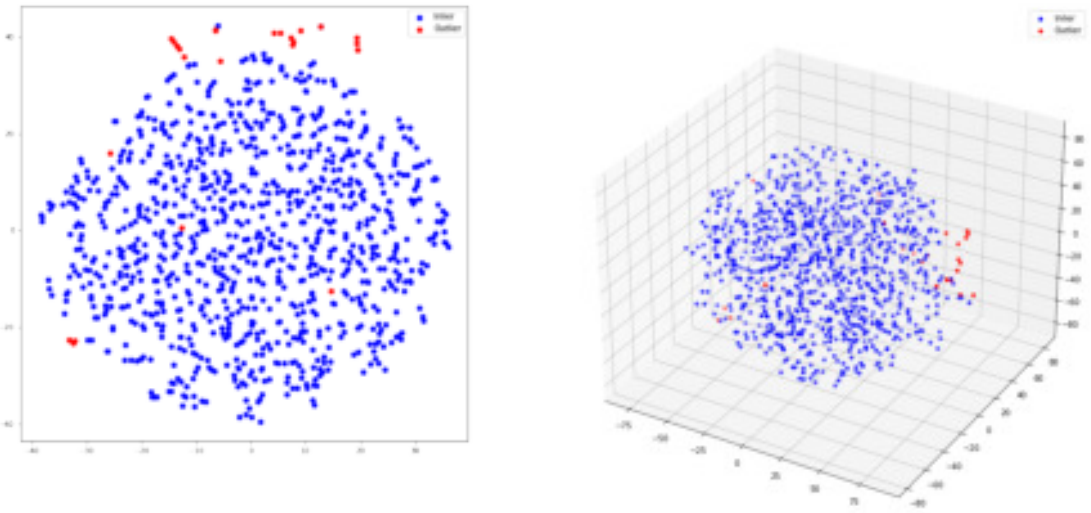
3.2 Synthetic Data

By observing the t-SNE plots of BTC/USD's distribution in lower dimensions, we concluded that it is of a clustered structure. However, as mentioned earlier, t-SNE does not give a complete picture, but it reveals parts of the distribution. Based on the data distribution, we created two synthetic datasets, The first dataset has dense clusters of outliers and a concentrated amount of normal observations in the center, see Figure 6. The second dataset contains sparsely scattered outliers and a concentrated amount of normal observations in the center, see Figure 7. The outliers are highlighted in red for both figures, while normal observations are blue. The clustered dataset should simulate USD/BTC's distribution, and the scattered dataset act as a benchmark dataset for comparison of results. As the data from Scila contains over 80 order books, the distribution may vary between datasets, which is another reason why we should not be limited testing to one distribution. The two datasets contains ten features, 1000 observations, and 2.5% injected outliers. The normal instances had features which were uniformly distributed between 0-100 for both datasets and the outliers had varying amounts of features injected as large numbers compared to their respective medians. To create the two distribution the exact injection of outliers was done in different ways. For the scattered dataset the outliers were injected for feature 1, 2, 3 and 6. Feature 3 had the largest values, followed by 2 and 6 with same uniform distribution and feature 1 had the smallest values of the outlying features. This injection of outlying features produced a clustered data structure and at the same time provided the ability to validate feature importance. The outliers in the scattered dataset were created by injecting large random values in random features, for 2.5% of the dataset.



(a) The clustered dataset in two dimensions. *(b) The clustered dataset in three dimensions.*

Figure 6: The clustered dataset, visualized with t-SNE. Outliers are highlighted in red.



(a) The scattered dataset in two dimensions. *(b) The scattered dataset in three dimensions.*

Figure 7: The scattered dataset, visualized with t-SNE. Outliers are highlighted in red.

A problem that may arise when creating synthetic datasets with ground truth labels has to do with so called trivial and non-trivial outliers. Trivial outliers include clear protruding values in features that can be easily seen by visual inspection, while non-trivial, also called hidden outliers, are harder to see. The non-trivial outliers can arise as a combination of several values in different dimensions. In single or lower dimensions, the value of the observation is not perceived as protruding, but as a composition in higher dimensions, it can become protruding. Hence, non-trivial outliers are challenging to handle when creating datasets with ground truth, as they are a composition of the "normally" classified data and thus are incorrectly classified as normal observations in the creation of the dataset. In higher dimensional datasets, this risk is amplified. Validating these methods will result in poorer performance measures than what they might actually deserve.

3.3 Pre-processing of the Data

The data from Scila is in CSV format where each row represents an observation i.e., a participant, and each column represents a feature, i.e., Avg. No of Orders/Avg. Trade Size, as seen in Table 3 on page 20. In some cases, the observations contained empty cells labeled with "Not a Number" (NaN), meaning that the observation was missing. Since Isolation Forest can not handle empty values, the NaN values were replaced by the median value for the corresponding feature. The median was used since it can be seen as the most normal value, meaning that it should not affect the scoring of outliers to a degree that matters as it will be found in a cluster of dense normal observations. In addition, both Isolation Forest and MI-Local-DIFFI have difficulties in handling scenarios where the vast majority of all values in a specific feature are of the same value. This can yield problems when isolating observations; therefore, a minimal Gaussian noise was added to all values in the entire dataset to make each value unique. The Gaussian noise was selected to 0.0005% of the cell value, thus not affecting the distribution of the data but instead improving the robustness of the model. Since our model is written in *Python*, the pre-processing was done on the Scila data using the tool Pandas, which enabled simple data-processing directly from a CSV file.

Since the the synthetic datasets were made by us, they were created in such way that pre-processing was not necessary.

4 Methodology

The following section builds on Section 2 and aims to describe the concepts of our method of choice, namely Isolation Forest with the MI-Local-DIFFI extension, and how we obtained the study results. We will present the most important definitions and functions of the algorithms, but we refer to the original papers for a reader who wants deeper knowledge [28] [5]. Afterward, the efficiency and performance of the Isolation Forest, MI-Local-DIFFI algorithms as well as a component of MI-Local-DIFFI, path length are evaluated separately, where we use the well-known measurements AUC and AUPRC for Isolation Forest and recall for MI-Local-DIFFI and Path length. Path length is used as a benchmark against MI-Local-DIFFI.

4.1 Isolation Forest

The Isolation Forest is a further development of the decision tree and random forest classification, built to detect outliers in datasets. As mentioned in Section 2.2.6, Isolation Forests takes on a different approach to outlier detection than most outlier detection methods. Instead of classifying what is normal in the data and then classifying observations which do not met the profile of a normal point, it never profiles the normal points. Instead, it directly targets the outliers resulting in an overall computation complexity significantly lower compared to most competing methods. This approach relies on the principle that anomalies are few and different and thus more susceptible to isolation than normal points.

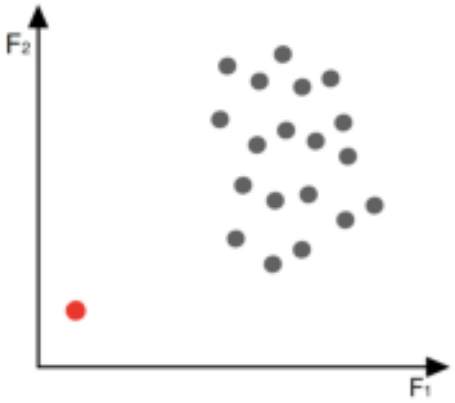
In short, Isolation Forest functions by the principle of recursively partitioning the data, by selecting a random feature and a random split value between the minimum and maximum value of the observations in the selected feature, creating so called isolation trees (iTrees). The trees are then used to test the data, with the goal of isolating each of the observations. Depending on the different splits in the trees the observations will require various number of splits to be isolated, and the path length will differ. Observations isolated closer to the root node, with short path length are thus more likely to be an outlier since they required fewer splits, indicating more extreme values than those needing many splits. See Figure 8. We base the representation of Isolation Forest in this section on the article [28] and use their notation in all parts of the section, if nothing else is stated. Also, our implementation of the code can be found in Appendix B.

The process divides into two phases: training and testing. It is in the training phase the iTrees are created and in the test phase, observations are tested in all iTrees to obtain path lengths and an outlier score for each observation as an average path length for the whole Isolation Forest (iForest).

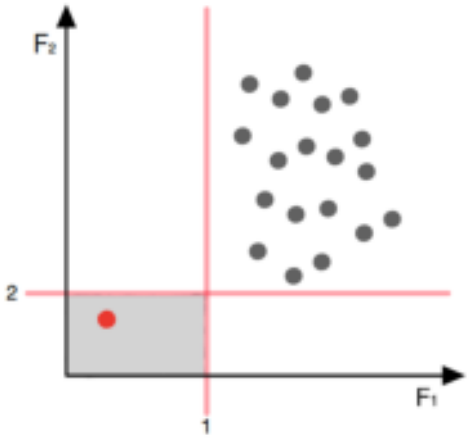
4.1.1 Training Stage

It is in the training stage, the iTrees, and thus the iForest, are constructed by recursive partitioning on the given dataset until the observation is isolated. The dataset used in the training stage is called training set, and can either be the complete dataset being examined or sub-samples (ψ) of the dataset. Unlike many classification methods, the dataset used in the training stage is therefore also used in the evaluation. The original paper suggests the use of sub-samples in the construction of the iTrees since it reduces the memory size

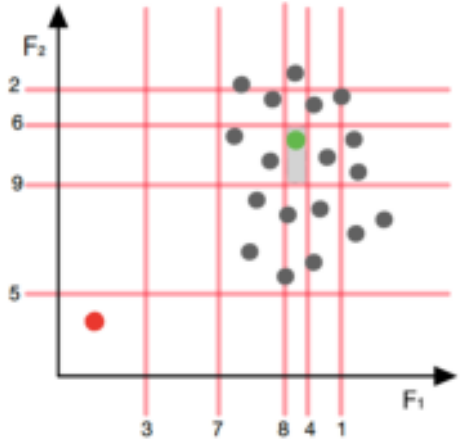
Figure 8: A two-dimensional dataset with 20 observations, 19 inliers and one outlier in red.



(a) Visualization of isolating an outlier in red.



(b) To isolate the outlier(red) two splits was required.



(c) To isolate the inliner(green) nine splits was required.

and processing time without a significant loss in performance. They also claim based on empirical studies that a sub-sample size of 256 many times provides sufficient results. The pros and cons of subsampling will be further discussed in section 4.4. To further improve performance of the model the author also included a maximum allowed tree height l, as an approximation of the average tree height. The theory behind this implementation is that, since we are looking to find outlier we are in practice only interested in trees shorter than the average path length. Therefore it makes no sense from a computational perspective to continue iterations pass this height. The average path length i calculated as $l = ceiling$ $(\log_2 \psi)$, and has it roots in the graph theory.

Isolation Forest is an ensemble methods, and due to the random elements in the construction of the isolation trees the method can not be dependent on one tree but has to build several random trees (t) and calculate the average of these. In tests made by the authors of the model, they found that path lengths tend to converge before $t = 100$, therefore 100 trees is sufficient for most datasets.

Details of the the training is to be found in Algorithm 1 and 2.

Algorithm 1 *iForest(X,t,ψ)*

Inputs: *X - input data, t - number of trees, ψ - sub-sampling size*
Output: a set of *t iTrees*
Initialize *Forest*
Set height limit $l = ceiling(\log_2 \psi)$
for i = 1 to t **do**
    $X' \leftarrow sample(X, \psi)$
    $Forest \leftarrow Forest \cup iTree(X', 0, l)$
end for
return *Forest*

Algorithm 2 *iTree(X,e,l)*

Inputs: *X - input data, e - current tree height, l - height limit*
Output: an iTree
if e $\geq l$ or $|X| \leq$ **then**
    return exNode{Size $\leftarrow |X|$*}*
else
    let Q be a list of attributes in X
    randomly select an attribute $q \in Q$
    randomly select a split point p from *max* and *min* values of attribute q in X
    $X_t \leftarrow filter(X, q < p)$
    $X_r \leftarrow filter(X, q \geq p$
    return *inNode{Left* $\leftarrow iTree(X_t, e+1, l)$*,*
                Right $\leftarrow iTree(X_r, e+1, l)$*,*
                SplitAtt $\leftarrow q$*,*
                SplitValue $\leftarrow p$*}*
end if

4.1.2 Evaluating Stage

In the evaluation step, an outlier score *s* is derived for each observation in the dataset. The outlier score is derived from the expected path length, *E(h(x))*, and not only the path length *h(x)* for each observations traversing through all iTrees in an iForest. This is due to the use of a max tree height, causing early termination for some of the observations. Because iTrees has the same structure as a binary search tree (BST), a rooted tree where all node values to the left are smaller and node values to the right are larger, than their parent nodes [4]. The estimate for early terminations, *h(x)*, is the same as for an unsuccessful search in a BST. Therefore, BST's definition helps to estimate the average path length of iTree:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{5}$$

Where *H(i)* is the harmonic number that can be estimated to $ln(i) + 0.5772156649$, where the constant is Euler's constant. Since *c(n)* is the mean of *h(x)* given n, it is used to normalize *h(x)*. The outlier score s for an observation x is thus defined as follows:

$$s(x, n) = 2^{\frac{-E(h(x))}{c(n)}} \tag{6}$$

where *E(h(x))* is the estimated value of *h(x)* from a collection of iTrees.

Details of the the evaluation are to be found in Algorithm 3.

Algorithm 3 *PathLength(x, T, e)*

1: **Inputs:** x - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
2: **Output:** path length of x
3: **if** T is an external node **then**
4: return $e + c(T.size)$ *{c(.) is defined in Equation 2}*
5: **end if**
6: $a \leftarrow T.splitAtt$
7: **if** $x_a < T.splitValue$ **then**
8: return *PathLength(x, T.left, e+1)*
9: **else**$\{x_a \geq T.splitValue\}$
10: return *PathLength(x, T.right, e+1)*
11: **end if**

4.2 Multiple-Indicator Depth-based Feature Importance for the Isolation Forest (MI-Local DIFFI)

To understand the logic behind the classifications made by Isolation Forest, we will use a model-specific feature importance method at a local level, called MI-Local-DIFFI. As mentioned in Section 2, MI-Local-DIFFI uses multiple indicators for calculating the feature importance instead of one indicator, the path length indicator, as its origin the DIFFI does [10]. MI-Local-DIFFI also consists of the path length indicator, but with the additions of split proportion indicator and split interval length indicator. However, as achieving high interpretability is essential for this study, we will be using the path length indicator individually in addition to all three indicators combined. This decision is made due to the fact that path length indicator is derived from one of the key functionalities of the Isolation Forest, while split proportion indicator and split interval length indicator are derived from less fundamental concepts of the Isolation Forest algorithm, and we fear that it may complicate the interpretability without any significant gain. The authors of the DIFFI algorithm express the importance of simplicity as following: "Along these lines, the proposed methods are consistent with the simplicity that characterizes the Isolation Forest model, thus avoiding the risk of developing an interpretability framework which is more complex than the model itself" [10]. But the addition of multiple indicators is interesting, and as mentioned we will present both the results of using the Path Length Indicator individually, and the results using all three indicators combined. To keep the study uniform and simple, we will only refer to MI-Local-DIFFI and not the Path Length Indicator unless in cases where it is necessary to refer to both, such as when presenting results or when comparing in the discussion. Path Length Indicator is, after all, a subset of MI-Local-DIFFI. Our implementation of the code can be found in Appendix B.

The primary assumption of MI-Local-DIFFI is that we can use the information in the splitting points of outlier paths to determine which features have been most important in the outlier identification process. To understand why an observation is classified as an outlier, we look at three different indicators based upon when an outlier traverses the trees in a forest. We base the representation of MI-Local-DIFFI in this section on the article [5] and use their notation in all parts of the section, if nothing else is stated.

- **Path length indicator:** The length of the outlier path in all trees.
- **Split proportion indicator:** In each split, we look at the proportions of observations that follow the path of the outlier against the observations that go to the opposite node. We assign a feature importance score to the splitting feature involved depending on this proportion.
- **Split interval length indicator:** The length of the sub-interval containing the outlier o after a split in proportion to the length of the interval before the split.

The path length indicator will as mentioned be used both individually and combined in MI-Local-DIFFI. It produces the feature importance score by observing path lengths for outliers, giving short path lengths a higher weight. Now we will discuss how all indicators are involved in measuring feature importance in detail, and introduce the corresponding mathematical expressions for the three different weights.

1) **Path length indicator.**
PL(o, i) is the path length of an outlier o leaf node in isolation tree t_i. Then we are using a standardized measure of the path lengths impact on feature importance by defining lower and upper path lengths, see Equations 7 and 8. The lower estimate corresponds to the shortest possible path length, and the upper estimate to the average path length. The path length weight is calculated according to Equation 9. Features in long outlier paths are less important than those in short paths. However the importance should also be increased if a feature provides a good split in a long path, therefore the lowest weight is 0.1 and not 0 for Equation 9.

$$PL_{lower} = 1 \tag{7}$$

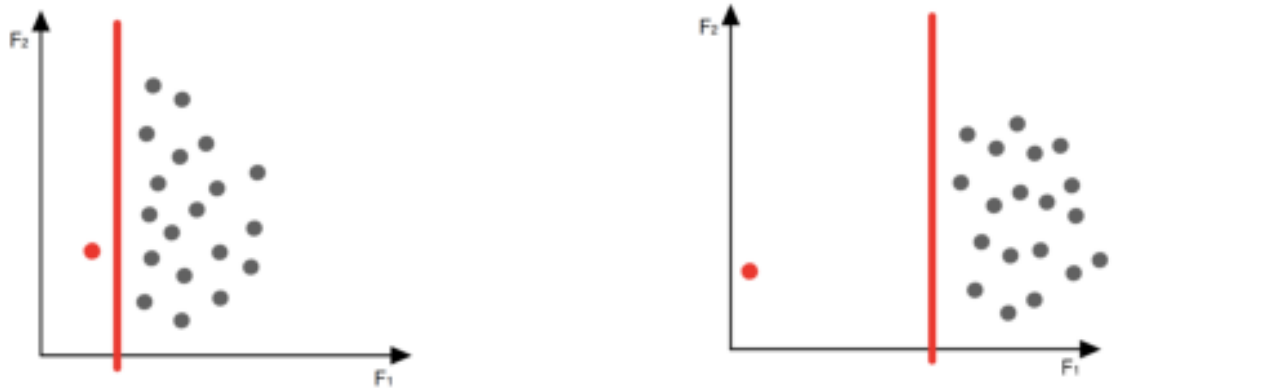
$$PL_{upper} = [2(log(\psi) + \gamma - 1)] \tag{8}$$

$$w^{PL}(o,i) = max\left\{0.1, min\left[1, 1 - \left(1 - \frac{PL(o,i) - PL_{lower}}{PL_{upper} - PL_{lower}}\right)\right]\right\} \tag{9}$$

2) **Split proportion indicator.**
Let q_{ij} be the number of observations in node v_{ij} and q_{ij}^0 the number of observations in child node of v_{ij}. Using the weighting from Equation 10, features extensively involved in isolating an outlier will receive a higher score than those that not significant in isolating an outlier. A feature split improves when the split proportion is higher. If a feature occurs multiple times in the same path, only the best split (highest w^{SP}), will be considered. This is because of Isolation Forests random partitioning, were an important feature may have a bad split, therefore those potential bad splits should not be punishing the important features.

$$w_j^{SP}(o,i) = \begin{cases} 0, & \text{if } q_{ij} = 2 \\ 1 - \frac{q_{ij}^0 - 1}{q_{ij} - 2}, & \text{otherwise} \end{cases} \tag{10}$$

3) **Split interval length indicator.** This indicator give us information about where the split was chosen. The indicator will decrease when a split value is close to the maximum or minimum of all values in the feature. The function of this indicator is to reward the split proportion indicator on good splits (when the interval is high) and punish it when the split is bad (when the interval is low). These two scenarios can be seen in Figure 10 below.



(a) Split with high proportion but small interval.

(b) Split with high proportion and large interval.

Figure 10: The figures above show that (b) is more of an outlier than (a). However, since a split value is chosen randomly, there is a risk that scenario (a) occurs. If we would not use the split interval indicator, scenario (a) would result in a feature importance weight as high as in scenario (b).

Viewing Figure 10, we see that (b) is a more evident outlier than (a), and therefore it should have increased weight based on how large the split interval is. The *outlier split interval* in Equation 11 is the interval $[a, p]$ if the outlier traverse to its left child and $[p, b]$ if the outlier traverse to its right child. a and b are the minimum and maximum values of the split feature, and p is the split value.

$$si(o, v_{ij}) = \frac{|\text{ outlier split interval of } v_{ij} \text{ with respect to o }|}{|\text{ feature split interval of } v_{ij} \text{ }|} \tag{11}$$

$$w_j^{SI}(o,i) = 1.5 - \frac{1}{si(o,v_{ij})+1} \tag{12}$$

For small intervals the weight will be close to 0.5 and for large intervals the weight will be close to 1, leaving the weight in the interval of [0.5, 1].

Let *Path(o, i)* denote the path in tree t_i that outlier o traverses. The MI-Local-DIFFI can then be presented as follows:
Consider an outlier o detected by Isolation Forest from a dataset X. To calculate the MI-Local-DIFFI feature importance score in every tree t_i, the information about feature splits is extracted. For each tree the randomly selected features receives weights depending on

Algorithm 4 *MI-Local-DIFFI*

procedure: MI-Local-DIFFI(o, IF = $\{t_1, ..., t_T\}$, PL)
  Let $G_1, ..., G_{n_F}$ be the unique features used to create Isolation Forest. We want to compute the feature importance for each G_k, $k = 1, ..., n_F$.
  Initialize feature importance vector $FI = \vec{0}$ with length equal to number of features n_F.
  Initialize feature occurence vector occurence(F) = $\vec{0}$ with length equal to number of features n_F.
for isolation tree t_i in IF **do**
    Let $\vec{F}_i = [F_{i,1}, ..., F_{i,m}]$ be the features that are used to split in each node in *Path(o,i)*.
    Calculate $w^{PL}(o, i)$ using Equation (5).
    Calculate $\vec{w^{SP}}(o, i)$ using Equation (6).
    Calculate $\vec{w^{SI}}(o, i)$ using Equation (8).
    for k in 1 to n_F **do**
      if Feature G_k occurs in $\vec{F}_i$ **then**
        Let j_k be the location of the best split of feature G_k in *Path(o,i)*.
        occurence(k) = occurence(k)+1
        $\sigma(k) = w^{PL}(o,i) \cdot w^{SP}_{jk}(o, i) \cdot w^{SI}_{jk}(o, i)$
        $FI(k) = FI(k) + \sigma(k)$
      end if
    end for
end for
$FI = FI$/occurence
return FI

the value of the three indicators. If a feature occurs more than once in a path it will be adjusted by the *occurrence* term (see Algorithm 4 above), which averages the combined score of all occurrences in a path. The three indicators are then multiplied with each other, which results in a final feature score between [0, 1]. The feature receiving the highest score is regarded to be the most important feature in regards of isolating the outlier. Notice the path length indicator will be in the interval of [0.1, 1] when used individually.

4.2.1 Visualization of the Indicators

To demonstrate how the split proportion and split interval length indicators of MI-Local-DIFFI works, we look at why o was chosen as an outlier in Figure 11 below. The Isolation Forest is created with a 3D-dataset and contains two isolation trees. Outlier o can be seen in red and its path is marked by red arrows.

(a) The first isolation tree in the forest; t_1. The red arrows highlights the path of outlier o.

(b) The second isolation tree in the forest; t_2. The red arrows highlights the path of outlier o.

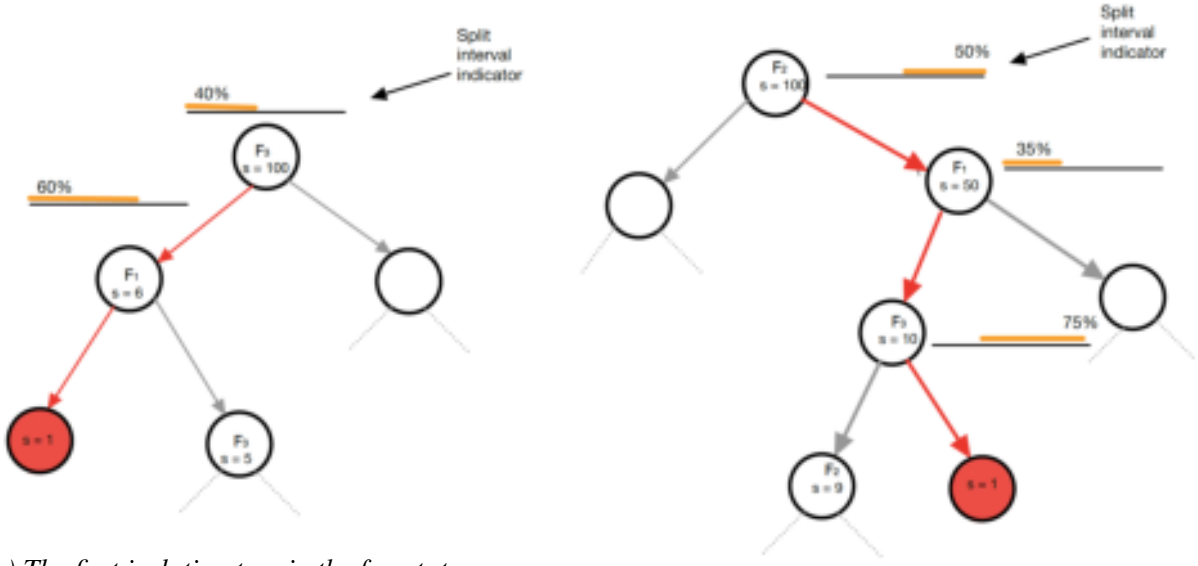
Figure 11: Isolation Forest with the two trees, t_1 and t_2.

In Figure 12, both trees start with $s = 100$, which equals the number of observations prior to the first split. For t_1, the first split is done to the left using feature 3 with a split interval indicator of 40% and a proportion of 6-94, i.e. very good in the sense of split proportion, and mediocre in the sense of split interval. Then feature 1 is chosen randomly for the next split and manages to isolate the outlier to the left with a 1-5 split, and an interval indicator of 60%. The same process can be seen in (b) but it takes one more step to isolate the outlier for this tree, and also the tree contains all three features.

By multiplying the weights w_{PL}, w_{SP} and w_{SI}, feature 3 is the most important for both trees, because of having relatively high split interval length indicator values and very high split proportion values, in both trees.

4.3 Tools

We used the programming language Python for implementing our version of Isolation Forest and MI-Local-DIFFI, following the structure of the pseudo-codes presented earlier.



(a) The first isolation tree in the forest; t_1, and its path outlier o traverses.

(b) The second isolation tree in the forest; t_2, and its path outlier o traverses.

Figure 12: The path of an outlier o for two trees t_1 and t_2, where s is the number of observations in each node.

For this thesis, we mainly used two libraries from Python, *Pandas* and *Numpy* [40] [39]. Pandas simplified data management through its data frame module, and Numpy was used to implement specific statistical methods smoothly. Together they facilitated the handling of large datasets. Otherwise, the main code is structured in a neutral way as possible to make it available for translation into other programming languages, such as Java which Scila uses. During the validation phase, specific methods from scikit learn were used, which is an efficient and straightforward tool for implementing machine learning methods in Python [31].

4.4 Validation & Hyperparameter Tuning

This section aims to describe how the performance and effectiveness of the model were measured during the tests on the scattered and clustered datasets, using different hyperparameters. Furthermore, this is linked to how the measures AUC, AUPRC, and recall are used for Isolation Forest and MI-Local-DIFFI, respectively. We will also describe how the evaluation stage is done differently between the two methods.

To perform validation tests on the models, it was necessary to perform tests on Isolation Forest and MI-Local-DIFFI separately, because of the different nature of the models. Isolation Forest is an outlier detection method which can be classified using binary terms of "0" and "1" where 0 is an inlier and 1 is an outlier. MI-Local-DIFFI, however, is not compatible with the binary classification form, due to one observation may contain multiple outlying features with different degrees of importance. Therefore, the measurements AUC and AUPRC were used for Isolation Forest, and recall was used for MI-Local-DIFFI.

4.4.1 Validation of Isolation Forest

As previously mentioned, the two configurable hyperparameters in Isolation Forest are the following; sub-sample size ψ and number of trees t. The parameters must be determined before Isolation Forest can train, and both have a major impact on its performance. Choosing the correct sub-sampling size can be crucial for the models performance this is primarily due to the problematic data distributions masking and swamping., two well-known problems in outlier detection. Swamping is the effect of miscategorizing normal observations as outliers, which usually occurs when the number of normal observations are too many or the observations are very scattered, making it harder to isolate outliers. Masking is the effect of a too concentrated amount of outliers that therefore hide their presence as outliers. Both of these effects can be counteracted by creating sub-samples of the data, since each sub-sample can build a better iTree than with the entire dataset [27]. An issue regarding sub-samples is that a low size combined with imbalanced data can lead to some trees not containing outliers as they are absent from certain sub-samples. Another parameter that needs to be specified is the threshold. It determines at which outlier score the model should start classify observation as outliers. This score depends on the structure of the dataset and thus varies between datasets. To achieve the best possible classification, it is therefore, desirable to choose the threshold with some form of perception of the structure of the data.

A common way to determine hyperparameters and parameters is with a so-called grid search, where you test the model based on different parameter choices and evaluate performance for each parameter choice [13]. The number of trees was chosen to $t = 100$ since the authors of [28] found that at $t = 100$ the path lengths usually converge, and after that, the computation burden out weights the marginal gain in performance. By systematically varying sub-sample size and thresholds, Isolation Forest's ability to distinguish outliers in the two synthetic datasets was evaluated and plotted with AUC and AUPRC. For Isolation Forest the threshold equals what classifies as an outlier in terms of the outlier score.

The plotting is done by initially setting the threshold to a value that causes Isolation Forest to classify all values as positives, i.e., the the same category. This means that both *TN* and *FN* will become one as no values are classified as negative. By varying the threshold, these fractions change, and in the end, all values have been classified as negative, which gives TN and *FN* values of zero. Each threshold-step is plotted, and the result is a curve showing the trade-off between the changes in TN and FN. The perfect result has a threshold value where there is no overlap of the classifications, and each observation is correctly classified, which generates a zero value of TPR and a value of one of FPR. The result is a curve that, together with the x- and y-axis, forms a perfect square with a side value of one. A perfect AUC is represented by a score of one [9].

4.4.2 Validation of Feature Importance

Validation of feature importance was a more difficult task than outlier detection because as mentioned in Section 4.4 it required a ground truth that was not binary. To evaluate the performance of MI-Local-DIFFI, we constructed an intuitive way to verify which of the most important features are for each outlier observation. The hyperparameters used during this evaluation were based on the ones most effective on respective dataset from

the evaluation results of Isolation Forest. As discussed in Section 3, the synthetic datasets are of two different distributions, and the ranking of feature importance was constructed in the same way for both datasets. Therefore, we measured how well MI-Local-DIFFI classified outlier features in the correct order (in terms of how protruding the values are from the median) and obtained the recall for each dataset. A feature with a value higher than 100 classified as an outlier feature, and the maximum number of outlier features per observation was four. MI-Local-DIFFI ran three times per synthetic dataset, and the recall was calculated as an average of those runs, also measuring the deviation between the runs. An example of the feature structure of the clustered dataset can be seen in Table 4 below, where the correct order would be 3, followed by 2 or 4, and lastly 1.

Table 4: Here we display 4 out of 25 outliers in synthetic dataset 1, where feature importance was determined by proportion to its median. Three was the most outlying feature, followed by two and four, followed by one. Five to ten were inliers uniformly distributed between 0-100 as well as features one to four, with the exception of outlier points 976 to 1000.

Point	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 5**	...	**Feature 10**
997	192	960	9906	966	16.0	...	4.00
998	196	973	9953	956	68.0	...	55.0
999	198	952	9971	976	55,0	...	75.0
1000	200	986	9943	983	96.0	...	32.0
Median feature	42	46	63	53	24.0	...	51

Table 5: The corresponding feature importance scores for the values in Table 4, in a range between [0, 1] where closer to 0 indicates a less important feature and closer to 1 indicates a more important feature.

Point	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 5**	...	**Feature 10**
997	0.43	0.60	0.75	0.60	0.32	...	0.32
998	0.43	0.61	0.76	0.60	0.30	...	0.29
999	0.44	0.60	0.75	0.61	0.29	...	0.31
1000	0.45	0.62	0.74	0.61	0.32	...	0.30

After evaluating Isolation Forest and MI-Local-DIFFI on our synthetic datasets, we used the hyperparameters that performed best on the synthetic data and ran the models on the Scila dataset, and presenting the visualization on the largest security (BTC/USD) creating visualizations for the five observations that received the highest outlier score and the

3, 10 and 33 most important features ranked by both MI-Local-DIFFI and Path Length Indicator.

5 Results

This section will describe the results given by the models presented in Section 4. The first part of this section will focus on the results of the synthetic data, displaying performance of Isolation Forest for different sub-sample sizes and thresholds using the AUC and AUPRC. The second part of the section will display the performance of MI-Local-DIFFI and Path Length Indicator on the synthetic data, extracting sub-sample size from the results of the Isolation Forest. The last part of this section will present the results from the models running on the largest security (BTC/USD) from the Scila dataset creating visualizations for the five observations that received the highest outlier score, with their 3, 10 and 33 most important features. All of the feature importance results in this section will be done for both MI-Local-DIFFI and Path Length Indicator.

5.1 Outlier Detection

Isolation Forest was evaluated on both of the synthetic datasets, using threshold values from 0.2 to 0.7, doing five runs per threshold, and for the sub-sample sizes of 10, 64, 128, 256 and 1000. Below we show the resulting AUC and AUPRC.

The performance of Isolation Forest for different sub-samples sizes on the scattered dataset is visualized in Figure 13, and its exact values are presented in Table 6. We see a varied performance, where small sub-samples have lower AUC and AUPRC, both measures increasing as the sub-sample size increases. For sub-sample size 1000, we see a higher AUC than for sub-sample size 256, but a slightly lower AUPRC. This is an indication of what we discussed in Section 2.6.5, where we discuss a weakness of the AUC concerning imbalanced data and its high sensitivity to large amounts of TN. Therefore, the trend change in AUPRC shows a pattern of weakness despite sub-sample size 1000 having a higher AUC than its previous value, 256.

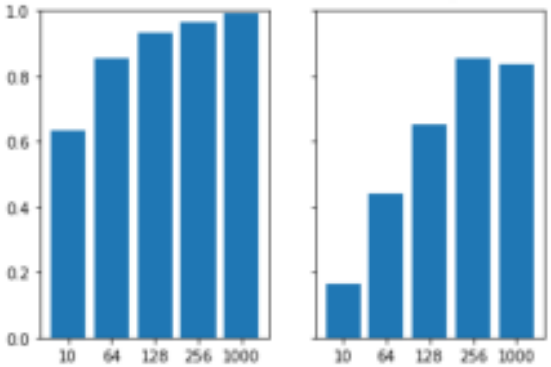


Figure 13: Resulting bar plots of AUC (left) & AUPRC (right) for each sub-sample size of scattered dataset.

The performance of Isolation Forest for different sub-sample sizes on the clustered dataset is visualized graphically in Figure 14, and its exact values are presented in Table 7. It is clear from the result that Isolation Forest performed at a near perfect level on the clustered data for all of the sub-sample sizes. The model reached an AUC score of almost 1 and an AUPRC in the higher ranges of 0.8 for all sub-sample sizes. Studying the numbers more

Table 6: AUC & AUPRC for each sub-sample size of the scattered dataset.

Sub-sample size	AUC	AUPRC
10	0.635	0.163
64	0.852	0.439
128	0.930	0.652
256	0.964	0.852
1000	0.992	0.836

closely, one can notice slightly worse performance on the sub-sample of 10 in AUC and AUPRC. Additionally, the sub-sample size of 1000 has slightly worse performance than the best scores in AUPRC, which as for the scattered dataset indicates what we discussed in Section 2.6.5.

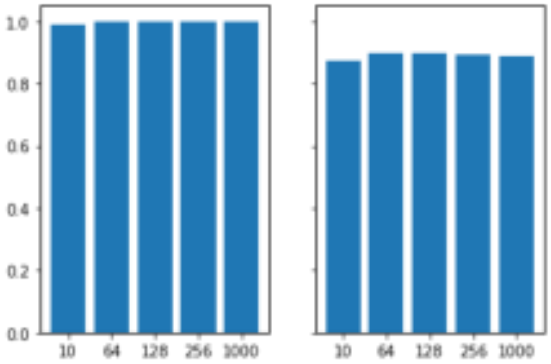


Figure 14: Resulting bar plots of AUC (left) & AUPRC (right) for each sub-sample size of the clustered dataset.

Table 7: AUC & AUPRC for each sub-sample size of the clustered dataset.

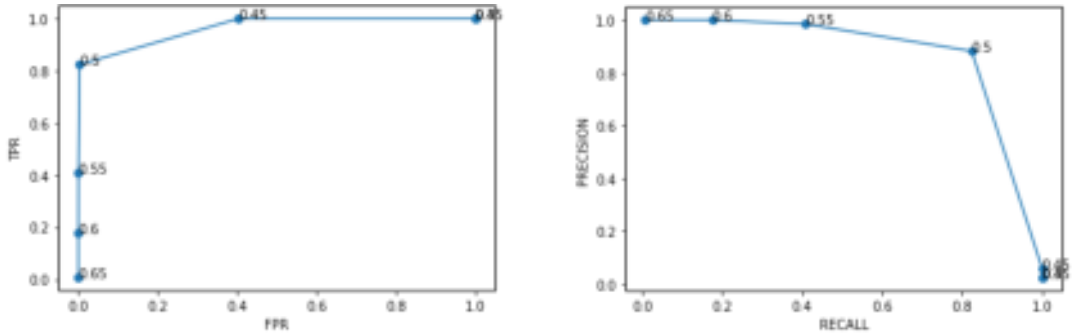
Sub-sample size	AUC	AUPRC
10	0.988	0.872
64	0.999	0.896
128	0.999	0.898
256	0.999	0.894
1000	0.998	0.886

Overall, Isolation Forest seems to perform weaker for the datasets in the lower scopes of sub-sample sizes and the largest sub-sample size. This can be concluded by looking at the AUPRC but not the AUC, which should be favorable because we are dealing with imbalanced data. The best performance is found with sub-sample size 256, which also corresponds to the recommendations of the authors in [28], who tested a variety of sub-sample sizes in different datasets. However, the differences are minor for the clustered dataset between sub-sample sizes. The AUC and AUPRC are stable with high scores, indicating that Isolation Forest performs well on this data distribution type, with clustered

outliers. We will use a sub-sample size of 256 based on those results when evaluating MI-Local-DIFFI later on in this section.

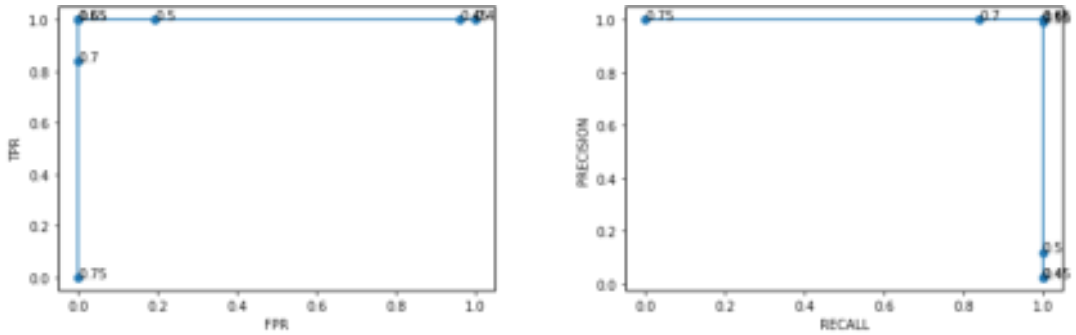
Given our selection of sub-sample size 256, Figure 15 and 17 displays the ROC and the Precison-Recall curve for this sub-sample size respectively. We are primarily interested in the Precison-Recall measurement considering the imbalanced data, as we discussed in Section 2.6, but in these cases the values of the AUPRC pretty much coincide, for different thresholds. Judging from the figures, Isolation Forest had the best performance in the clustered dataset near 0.6 and for the scattered dataset near 0.5. Hence, we will be using these thresholds for the respective datasets moving forward. The data from Scila lacks ground truth, and therefore we will use the same threshold as for the clustered dataset, 0.6, following our conclusions in Section 3.2, where we visualized the distributions.

Figure 15: ROC and Precison-Recall curve for the scattered dataset and sub-sample size of 256 observations with plotted thresholds.



(a) ROC curve *(b) Precision - Recall curve*

Figure 17: ROC and Precison-Recall curve for the clustered dataset and sub-sample size of 256 observations with plotted thresholds.



(a) ROC curve *(b) Precision - Recall curve*

5.2 Feature Importance

The evaluation of MI-Local-DIFFI and Path Length Indicator was performed as described in Section 4.4.2 with the sub-sample size 256 determined from the results presented in Section 5.1 For both datasets we measured the recall, i.e., the amount of ground truth features which were classified correctly. Two different tests were performed where one

test required correctly classified outlying features but without regard to order while the other test required correct classification and correct order. The results can be seen in Table 8 and 9.

Table 8: The average recall of MI-Local-DIFFI and Path Length Indicator on the scattered dataset, and the min/max deviations from the the experiments.

Scattered dataset	With ordering	Deviation	Without ordering	Deviation
MI-Local-DIFFI	0.66	+0.08 -0.09	0.88	+0.06 -0.06
Path Length Indicator	0.79	+0.01 -0.01	0.93	+0.01 -0.02

Table 9: The average recall of MI-Local-DIFFI and Path Length Indicator on the clustered dataset, and the min/max deviations from the the experiments.

Clustered dataset	With ordering	Deviation	Without ordering	Deviation
MI-Local-DIFFI	0.32	+0.08 -0.09	0.63	+0.12 -0.11
Path Length Indicator	0.42	+0.05 -0.05	0.84	+0.01 -0.01

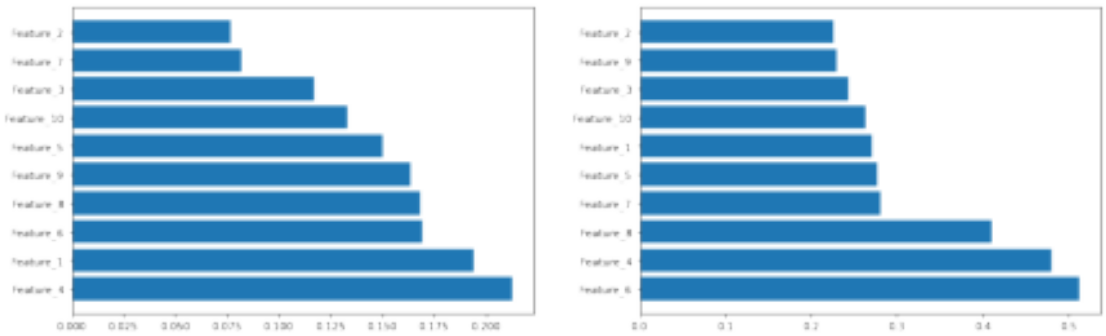
As expected, without ordering produces better results than with ordering in both dataset as well as for both MI-Local-DIFFI and Path Length Indicator. We also notice that Path Length Indicator receives a higher recall than MI-Local DIFFI on all of the four tests and looking at the deviation it also produces more stable results. For both the MI-Local-DIFFI and Path Length Indicator we notice how it is more difficult to classify the correct features in the clustered dataset, both with ordering and without ordering. Looking at the deviations, we see that the deviations from each run are larger for MI-Local-DIFFI compared to Path Length Indicator. There is also a small tendency of higher deviations in the clustered data, for both methods. As mentioned, the recall is worse on the clustered data, for both methods. However, as we look at Figure 14 we saw that Isolation Forest performed particularly well for the clustered dataset, compared to the scattered dataset. There is a discrepancy between the results of the Isolation Forest and the feature importance algorithms, especially looking at the clustered dataset.

Below vi visualize the distribution for the feature importance scores, created by both MI-Local-DIFFI and Path Length Indicator for the top outlying point in the scattered dataset and the clustered dataset respectively.

Table 10: Feature values for each feature in point 850 in the scattered dataset.

Point	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
850	75	24	80	947	24	1000	9	627	56	32

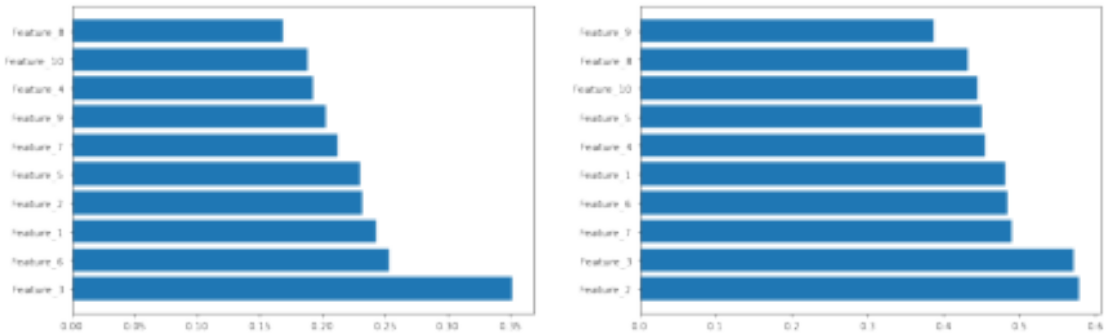
Comparing Table 10, which is showing the actual feature value for observation 850 in the scattered dataset, and the histograms in Figure 19, which shows the feature scores for both



(a) MI-Local DIFFI *(b) Path Length Indicator*

Figure 19: Representation of the most important features for the top outlier in the scattered dataset from both MI-Local-DIFFI and Path Length Indicator.

MI-Local-DIFFI and Path Length Indicator, some observations can be made. The first thing we look at is how well the methods have managed to score the correct features. From Table 10, we can see that feature 6, feature 4, feature 8 and feature 10 are the outlying features in descending order. MI-Local-DIFFI has scored the features in the following order: feature 4, feature 1, feature 6 and feature 8 as the top 4 outlying features. Feature 10 which is the 4th outlying feature ends up in 7th place according to MI-Local-DIFFI. Path Length Indicator on the other hand scores the features in the following order feature 6, feature 4, feature 8 and feature 7 as top 4 outlying features. Feature 10 is also placed on 7th place according to Path Length Indicator. We can also see that Path Length Indicator generates greater difference in the score from the highest outlying features to the rest. The "normal" features, except feature 10, also seems to get an equal and even points among themselves, as well as the magnitude of the top scores visually seems to have an reasonable magnitude and difference, comparing to the ground truth. This is not notable in the same way for MI-Local-DIFFI, although producing scores in descending order but with differences of small magnitude for each rank.



(a) MI-Local-DIFFI *(b) Path Length Indicator*

Figure 20: Representation of the most important features for the top outlier in the clustered dataset from both MI-Local-DIFFI and Path Length Indicator.

When we do the same analysis for the clustered data, we see in Table 11 that the outlying features in the ground truth are feature 3, feature 2, feature 6 and feature 1 in descending order, where feature 3 has outlier value of far greater magnitude than the rest of the outlier values. It is also notables that the difference in value between feature 6 and feature 2 is

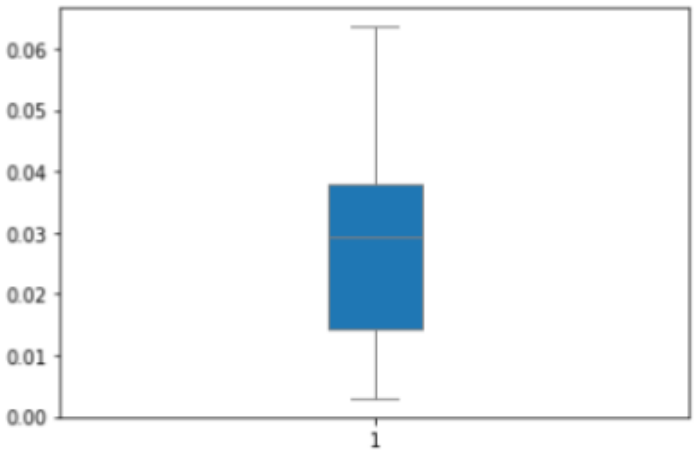
Table 11: Feature values for each feature in point 998 in the clustered dataset.

Point	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
998	199	955	9951	34	6	994	5	62	52	63

small. Analysing the result from MI-Local-DIFFI we can see that feature 3, feature 6, feature 1 and feature 2 is ranked as the top 4 outlier, meaning that the method correctly classifies the outlying features, but not completely in the correct order, since feature 1 receives a slightly higher score than feature 2. Regarding the magnitudes of the scores one can see that feature 3 stands out as excepted. As for the other features the difference is very small between the points and interestingly F5 received almost an identical score as feature 2. The Path Length Indicator scores the features in the following descending order, feature 2, feature 3, feature 7, feature 6. This means that the method correctly classifies 3 out of 4 top outliers, missing feature 6 which is placed on the 5th place. Concerning the order it perform worse than MI-Local-DIFFI giving feature 2 the highest score and placing feature 7 as third. Notable is that the difference between feature 2 and feature 3 is very small, although the real difference is large. The general difference between the feature score, can be observed as quite small.

5.3 Evaluation of the Scila Dataset

By evaluating Isolation Forest on both synthetic datasets we have chosen the sub-sample size of 256 and threshold of 0.6 for the Scila dataset. Isolation Forest ran on all of the 88 securities in the Scila dataset using the hyperparameters above, and the number of outliers for each security is presented in Table 14 in Appendix A. Due to limited space when visualizing the feature importance of the securities, we present the largest security in terms of observations. We visualize the five largest outliers from this security with its ten most important features.

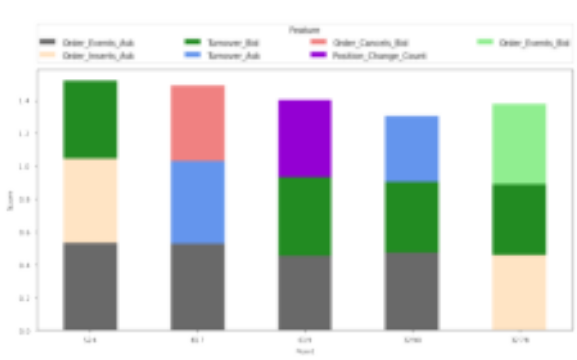


Average	Max	Min
0.028	0.063	0.003

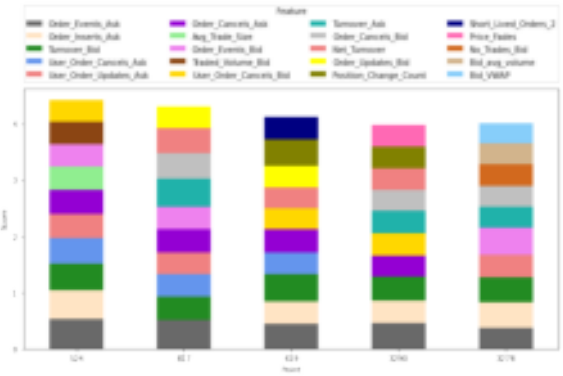
Figure 21: Box plot showing the distribution of the outlier fraction from Scila dataset (BTC/USD) visualized with t-SNE.

From Figure 21, we can see that the average fraction of outliers in the Scila dataset is 2.8%, which confirms that the datasets are imbalanced. This was an expected result, considering the prior evaluations of the t-SNE visualizations. However, the average fraction is not a representation for a specific security, so from Figure 21, we notice that 50% of the data contains about 1.5% to 4% outliers. Then there is 25% each between 0.3% to 1.5% and 4% to 6.3%. This concludes that all of the securities categorize as imbalanced datasets but also confirms that the chosen threshold was a reasonable choice.

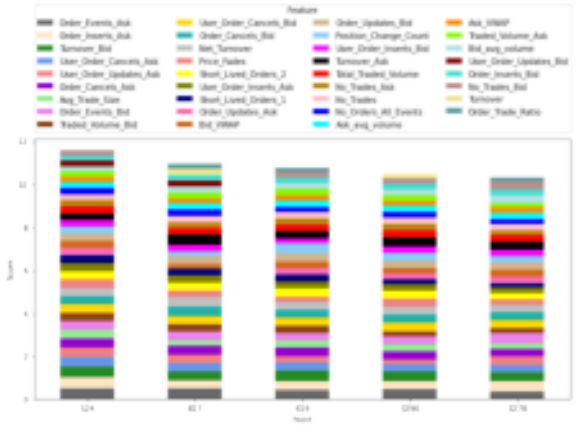
Figure 22: Top n outliers with bar plot representation of top ten most important features for the five largest securities with MI-Local-DIFFI.



(a) n=3.



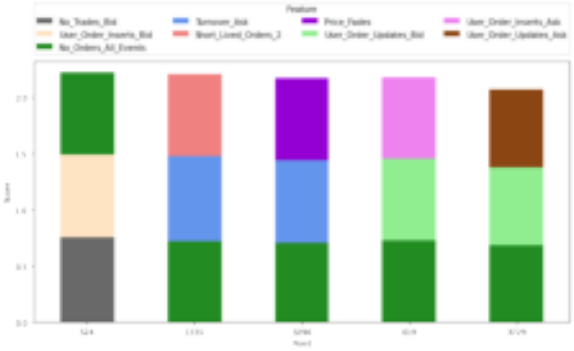
(b) n=10.



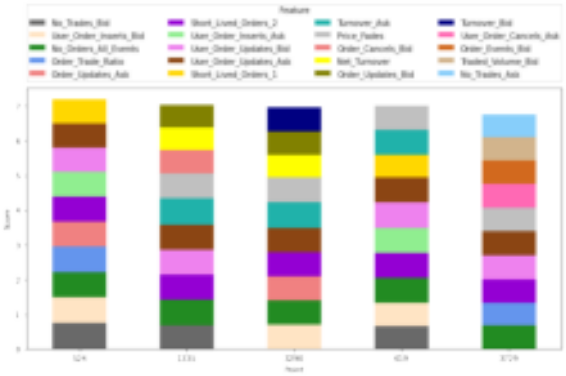
(c) n=33.

Figure 22 and 24 presents the visualization result of the MI-Local-DIFFI on the top five outliers from the five largest securities with top 3, 10 and 33 most important features. The result from Figure 22 is created with MI-Local-DIFFI and the result from figure 22 is created with Path Length Indicator. As we can see from the bar plots they look similar for MI-Local-DIFFI and Path Length Indicator. However, judging from Figure 26 we see there are differences in the score distribution between MI-Local-DIFFI and Path Length Indicator, which was hard to notice watching only the bar plots in Figures 22 and 24.

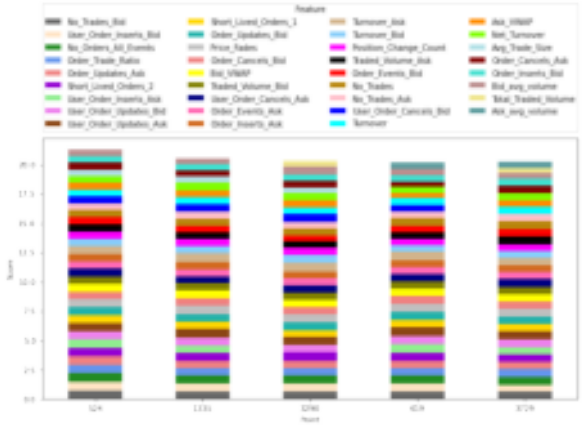
Figure 24: Top n outliers with bar plot representation of top ten most important features for the five largest securities with Path Length Indicator.



(a) n=3.

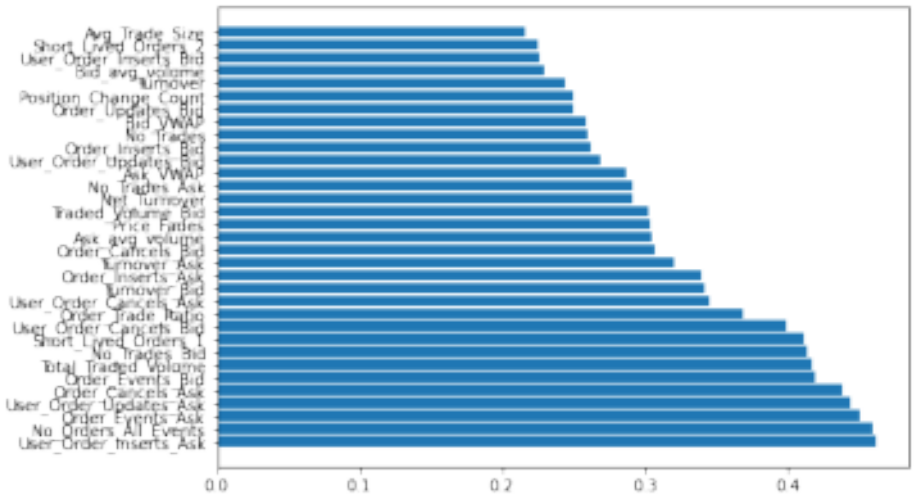


(b) n=10.

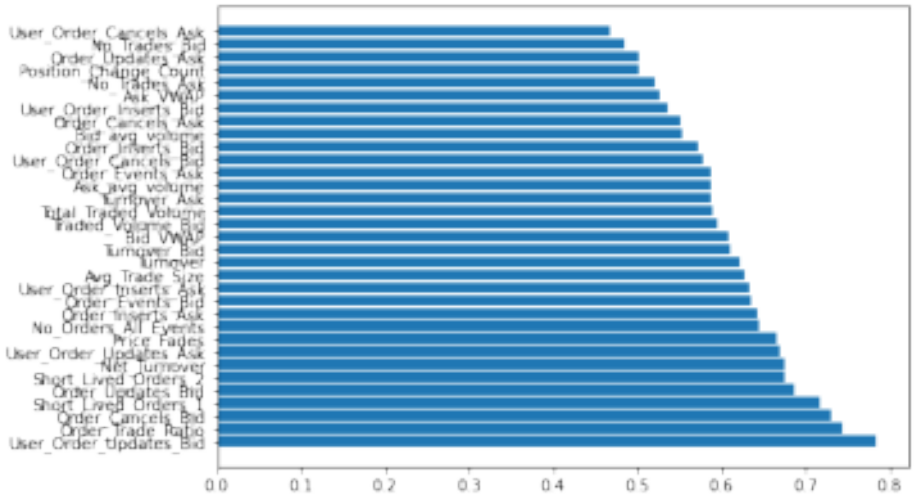


(c) n=33.

Figure 26: Representation of the most important features for the largest outlier in security BTC/USD of the Scila dataset.



(a) MI-Local-DIFFI



(b) Path Length Indicator

6 Discussion

The following section will discuss and analyze the results from Section 5. First we will discuss the overall performance of the models. Then we will talk about the performance differences for the models between the different distributions scattered and clustered data. Lastly we touch upon the visualization of the results, and talk about how to make the results of feature importance more interpretable using the right visualization.

6.1 Overall Performance of the Models

From Figure 13 and 14 we can see that Isolation Forest has a higher AUC and AUPRC for the clustered dataset compared to the scattered dataset on all sub-sample sizes. In the scattered dataset from Figure 13 the $\psi = 256$ has the second-best AUC but the best AUPRC. This is considered a reasonable result based on what we mentioned in Section, 2.6.5, where AUC is being boosted for an imbalanced dataset due to true negatives. Hence AUPRC is a necessary complement. The method receives a higher AUC with $\psi = 1000$ but a lower AUPRC, suggesting that Isolation Forest with a sub-sample size of 1000 is a slightly worse classifier than 256, for the scattered dataset. Regarding the clustered data, we can see that the values are similar. The method performs close to perfect for all sub-sample sizes, indicating that the method generally performs well in that type of data distribution. It performs well for the scattered dataset, but is more sensitive to the choice of hyperparameters.

The threshold decision for the Scila dataset was, as mentioned in Section 5.1 due to its similarities with the synthetic clustered dataset. Choosing the threshold this way is not optimal since we base the threshold selection on a visualization of Scila's data with t-SNE that does not reveal the whole picture. Setting a too low threshold may correctly classify all *true positives* but may also include many *false positives*, and a too-high threshold, on the other hand, might result in the method failing to find all true positives. Regardless, the threshold needs to be defined, and it was the obvious way to choose a somewhat valid threshold. Nevertheless, the threshold was less critical for this study since it primarily focuses on the most outlying observations, i.e., the observations with the highest outlier score. A limited number of outliers are reasonable for an operator to examine using feature importance, and naturally, the most significant outliers will be reviewed first. Therefore, finding the exact threshold is not crucial as long as the threshold is sufficient to give us the top outliers.

As mentioned in Section 4.2, both MI-Local-DIFFI and Path Length Indicator were evaluated in Section 5. The result for each run is presented in Table 8 and Table 9. It shows that Path Length Indicator receives a higher recall than MI-Local-DIFFI on both datasets, for both with and without ordering. Additionally, MI-Local-DIFFI had larger deviations between runs, indicating more uneven results. Based on these results, the Path Length Indicator, a subset of MI-Local-DIFFI, can be concluded to perform better than MI-Local-DIFFI on the synthetic data. Henceforth, we will focus on Path Length Indicator for further discussions. This is also supported by the fact that results from the Path Length Indicator should be easier to understand, as discussed in Section 4.2. Nevertheless, we will discuss the results for MI-Local-DIFFI and potential reasons why it under performed the Path Length Indicator for this study.

The result of our synthetic data for MI-Local-DIFFI was, as mentioned, worse than for Path Length Indicator. This result is not in agreement with the results of the original paper [5], which also made a test comparing the methods MI-Local-DIFFI and Path Length Indicator. Our different results could have to do with shortcomings in the implementation of the algorithm. However, we consider this unlikely since we made continuous, detailed testing throughout the development and for the finished product, showing no signs of shortcomings. The code can be examined in Appendix B. The most likely reason, according to us, has to do with the datasets used and differences in data distributions. Nevertheless, to make conclusions about the disparity in the results between MI-Local-DIFFI and Path Length Indicator, the three indicators must be examined in-depth, particularly the correlation between all three indicators. This is outside the scope of this study, but it would add an increased understanding of MI-Local-DIFFI.

6.2 Performance Difference between Data Distributions

As mentioned above, we will focus on the results of the Path Length Indicator for feature importance. If we compare the synthetic datasets in Table 8 and 9, the recall is noticeably better for the scattered dataset. Looking at the proportions between "with ordering" and "without ordering" for both datasets, we notice more significant deviations for the clustered data than for the scattered data. Moreover, we notice a discrepancy in the results of Isolation Forest and Path Length Indicator for the synthetic datasets. Isolation Forest performed better for the clustered dataset than for the scattered dataset, while Path Length Indicator performed worse for the clustered dataset and better for the scattered dataset. This is likely to be explained by the random partitioning functionality mentioned above.

When Isolation Forest classifies an outlier it produces an aggregated score for all features while Path Length Indicator looks at the features individually. When the data is dense and clustered, the Isolation Forest's random selection of features and values for splits will affect Path Length Indicator's ability to score features. More minor differences in random choices of features and especially values will lead to higher variances in the feature scores. We can also see this tendency by viewing the deviations between runs in 8 and 9, where the deviation in "with ordering" for clustered data is five times larger than for scattered data. Lastly, Isolation Forest has high performance in clustered data because it uses aggregated feature scores without regard for the correct order.

As previously mentioned, the synthetic clustered dataset was created to mimic Scila's data distribution. Figure 20 and 26 show how features are scored in the synthetic clustered data and BTC/USD respectively. We see for both datasets that normal features get relatively high scores compared to outlying features. This phenomenon can be linked to the path length's functionality in Isolation Forest, where all features per tree are randomly selected in the process of isolating an outlier, giving the same score to all features selected in that tree. This clarifies that outlying features that contribute to a fast isolation (i.e. shorter path) will increase the score for normal features selected for the same tree. The same phenomenon can be viewed for the scattered dataset from Figure 19 where its normal features (all above feature 8) have an even, relatively high score. This is a disadvantage of the Path Length Indicator and is based on the model-specific nature of the algorithm that it scores features based on how Isolation Forest isolates outliers.

Table 12: Feature values of four observations in BTC/USD.

Point	**Order Updates Ask**	**Short Lived Orders 2**	**Order Trade Ratio**	**User Order Updates Bid**	...	**Order Events Ask**
35	4177	1007	162.25	4068	...	3970
512	4038	948	209.45	3885	...	4082
1344	3918	910	186.17	3871	...	4247
453	3840	855	98.76	3983	...	3890
Median feature	8	2	4.6	7	...	19

Table 13: Feature importance scores of four observations in BTC/USD.

Point	**Order Updates Ask**	**Short Lived Orders 2**	**Order Trade Ratio**	**User Order Updates Bid**	...	**Order Events Ask**
35	0.754268	0.684484	0.647482	0.648452	...	0.603475
512	0.757334	0.658698	0.649812	0.609489	...	0.615597
1344	0.722228	0.653658	0.629210	0.606686	...	0.633741
453	0.757334	0.617650	0.540102	0.648452	...	0.600746

In Table 12 and 13 we see the instrument BTC/USD with its feature values and feature scores respectively. If we examine the features Order Updates Ask and User Order Updates Bid, we can see that feature values are the same for all observations/points, while feature scores differ. We could not make sense of these relations by applying our knowledge on how path length scores are distributed. However, we assume that these discrepancies may be differences in data distribution, which are difficult to see with only a manual comparison of the values and the median.
So far, we have discussed why the Path Length Indicator performs worse while Isolation Forest performs better on dense, clustered data distributions based on the random partitioning functionality of Isolation Forest. We have also discussed why normal features get a relatively high score, given that outlying features affect their path length and shorten it, thus boosting their scores. Finally, we commented on how data distribution potentially impacted the scoring of features. Now we shall discuss how feature importance can be visualized to make it more interpretable.

6.3 Interpretable Visualization of the Data

An essential part of the study was the ability to visualize the feature importance so that the result becomes interpretable for an operator. In Section 5.3, we presented two ways

to visualize the data with various advantages and disadvantages. A bar plot with one observation per bar allows simultaneously plotting several outliers and their respective features. However, for the Scila dataset, we saw that when 33 features were presented in the same bar, it became difficult to distinguish the proportions between the features. The optimal case would be to showcase fewer features and only the ones that have had the most effect. However, when, as in Scila's case, almost all of the features have similar scores as we discussed in Section 6.2, it becomes problematic to use bar plots even when showcasing fewer features. The horizontal histogram was the one that presented the proportions between features best but with the drawback of being limited to only showcasing one observation per chart. Therefore we can conclude that histogram results in a better overall interpretability, given it can be used to successfully display proportions between features for any distribution of data.

7 Conclusions & Future Work

In this thesis, we have tried to implement a complementary tool for an operator to get clues on suspect customer behavior, with the purpose of creating an interpretable result which can be understood with the corresponding expertise. More specifically, we chose the unsupervised machine learning model Isolation Forest and a feature importance model (MI-Local-DIFFI and its subset Path Length Indicator) to detect outliers and produce the results in an interpretable way for an operator with expertise.

Our models were used on three datasets, two synthetic datasets that we created, with different distributions (one scattered and one clustered), and one dataset from Scila. The synthetic datasets were used to evaluate the performance of our models. To what degree can the chosen outlier detection model achieve a level of interpretability that enables interpretation through human expertise?

First, it can be noted that Isolation Forest has an excellent ability to find outliers in the various data distributions we investigated. We used a feature importance model to make Isolation Forest's scoring of outliers interpretable. Our intention was that the feature importance model would specify how important different features were in the process of an observation being defined as an outlier. Our results have a relatively high degree of interpretability for the scattered dataset but worse for the clustered dataset. The Path Length Indicator achieved higher recall than MI-Local-DIFFI for both datasets. We can note that the feature importance model as a model-specific method is limited by how Isolation Forest isolates an outlier.

One problem is that features with a high score also enhance the score of features that generally have a lower score. The other problem is that Isolation Forest's efficiency in finding outliers largely depends on the randomness of the model, which affects a model-specific feature importance method's ability to indicate which features have had the most significant impact in isolating an outlier. How the data is distributed also affects the result. The feature importance model is more sensitive to a clustered data set in terms of the problems described above than a scattered data set. This correlation was reversed for Isolation Forest because it does not need to specify the correct order for features but instead use an aggregated score.

How could one then move on to make the result more interpretable? This section will focus on the structural and theoretical improvements that could be interesting to study further.

There are a number of things that could be investigated further to improve the models in this study, especially MI-Local-DIFFI. An initial step would be to clarify the correlation between the three indicators in MI-Local-DIFFI. This could lead to an increased understanding of necessary improvements of the model, and to a more interpretable understanding of the results. Another interesting subject is the connection between the feature importance model and the data distribution.

Additionally, the possibility of implementing a model-specific feature importance algorithm for the many extensions of Isolation Forest should be reviewed. Also the synthetic datasets could be created in a more systematic way, e.g. varying the distributions by

changing one or multiple parameters, thus testing the models on multiple distributions.

To conclude, we showed that Isolation Forest is a good model for finding outliers in high dimensional, imbalanced datasets, both with scattered and clustered distributions, however it performed especially well for the clustered dataset. The Path Length Indicator performed better than MI-Local-DIFFI for our datasets. Lastly, we noticed some performance flaws in Path Length Indicator's ability to rank features originating from its method-specific nature of Isolation Forest.

Bibliography

[1] Agarwal, D. "Detecting anomalies in cross-classified streams: a Bayesian approach". In: *Knowledge and information systems* 11.1 (2006), pp. 29–44.

[2] Arning, A, Agrawal, R, and Raghavan, P. "A Linear Method for Deviation Detection in Large Databases." In: *KDD*. Vol. 1141. 50. 1996, pp. 972–981.

[3] Bandaragoda, T.R. et al. "Isolation-based anomaly detection using nearest-neighbor ensembles". In: *Computational intelligence* 34.4 (2018), pp. 968–998.

[4] Bender, E.A and Williamson, G. *Lists, Decisions and Graphs*. University of California at San Diego, 2010.

[5] Bergbórsdottir, K.B. "Local Explanation Methods for Isolation Forest: Explainable Outlier Detection in Anti-Money Laundering". MA thesis. Delft University of Technology, 2021.

[6] Breiman, L. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[7] Breunig, M. et al. "LOF: identifying density-based local outliers". In: *SIGMOD record* 29.2 (2000), pp. 93–104.

[8] Calders, T and Jaroszewicz, S. "Efficient AUC Optimization for Classification". In: *PKDD*. 2007.

[9] Campos, G.O et al. "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". In: *Data mining and knowledge discovery* 30.4 (2016), pp. 891–927.

[10] Carletti, M, Terzi, M, and Susto, G.A. "Interpretable anomaly detection with diffi: Depth-based feature importance for the isolation forest". In: (2020).

[11] Caruana, R and Niculescu-Mizil, A. "An empirical comparison of supervised learning algorithms". In: Proceedings of the 23rd international conference on Machine learning, pp. 161–168.

[12] Chandola, V, Banerjee, A, and Kumar, V. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (2009).

[13] Claesen, M and De Moor, B. *Hyperparameter Search in Machine Learning*. 2015.

[14] Domingues, R et al. "A comparative evaluation of outlier detection algorithms: Experiments and analyses". In: *Pattern Recognition* 74 (2018), pp. 406–421.

[15] Ertoz, L et al. "Minds-minnesota intrusion detection system". In: *Next generation data mining* (2004), pp. 199–218.

[16] Eskin, E. "Anomaly Detection over Noisy Data using Learned Probability Distributions". In: In Proceedings of the International Conference on Machine Learning, 2000, pp. 255–262.

[17] Ester, M et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[18] Finansinspektionen. *Marknadsmissbruk*. https://www.fi.se/sv/publicerat/statistik/marknadsmissbruk/. 2022. (Accessed: 2022.03.22).

[19] Golmohammadi, S.K. “Time series contextual anomaly detection for detecting stock market manipulation”. PhD thesis. Department of Computing Science, University of Alberta, 2016.

[20] Gross, J.L, Yellen, J, and Zhang, P. *Handbook of graph theory*. Second. Boca Raton: CRC Press, Taylor Francis Group, 2014.

[21] Gunning, D and Aha, D. “Explainable artificial intelligence (XAI)”. In: *Data Mining. ICDM ’08. Eighth IEEE International Conference* (2008), pp. 413–422.

[22] Kherbache, M, Espes, D, and Amroun, K. “An Enhanced approach of the K-means clustering for Anomaly-based intrusion detection systems”. In: IEEE, 2021, pp. 78–83.

[23] Kingsford, C and Salzberg, S.L. “What are decision trees?” In: *Nature biotechnology* 26.9 (2008), pp. 1011–1013.

[24] Knorr, E.M. and Ng, R.T. “A unified approach for mining outliers”. In: Toronto, Canada: Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON), 1997.

[25] Kotsiantis, S.B. “Decision trees: a recent overview”. In: *Artificial Intelligence Review* 39.4 (2013), pp. 261–283.

[26] Kriegel, H.P, Kröger, P, and Zimek, A. “Outlier Detection Techniques”. In: P16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2010.

[27] Liu, Fei Tony, Ting, Kai Ming, and Zhou, Zhi-Hua. “Isolation-Based Anomaly Detection”. In: *ACM Trans. Knowl. Discov. Data* 6 (2012), 3:1–3:39.

[28] Liu, F.T, Ting, K.M, and Zhou, Z.H. “Isolation Forest”. In: *2008 Eighth IEEE International Conference on Data Mining* (2008), pp. 413–422.

[29] Luque, A et al. “The impact of class imbalance in classification performance metrics based on the binary confusion matrix”. In: *Pattern Recognition* 91 (2019), pp. 216–231. ISSN: 0031-3203.

[30] Maaten, L Van der and Hinton, G. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[31] *Machine learning in Python*. https://scikit-learn.org/stable/index.html. (Accessed: 2022.04.02).

[32] Mandhare, H.C and Idate, S.R. “A comparative study of cluster based outlier detection, distance based outlier detection and density based outlier detection techniques”. In: *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2017, pp. 931–935.

[33] Mehlig, B. *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.

[34] Molnar, C. *Interpretable Machine Learning: A Guide For Making Black Box Models Explainable*. Second. 2022.

[35] Monatvon, G, Samek, W, and Müller, K.-R. “Methods for interpreting and understanding deep neural networks”. In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[36] Myrdal, J and Leopoldson, J. *Aktiefusk - var går gränsen?: en bok om marknadsmanipulation*. Norstedts juridik, 2021.

[37] Nasdaq. *Yearly Nordic Statistics 2000-2021*. http://www.nasdaqomxnordic.com/news/statistics. 2021. (Accessed: 2022.03.22).

[38] Nordqvist, C. *Växande aktiefusk – en angelägenhet för både professionella och småsparare*. https://www.nj.se/nyheter/vaxande-aktiefusk-en-angelagenhet-for-bade-professionella-och-smasparare/. 2021. (Accessed: 2022.03.22).

[39] *Numpy documentation*. https://numpy.org/doc/stable/. (Accessed: 2022.04.02).

[40] *Pandas documentation*. https://pandas.pydata.org/docs/. (Accessed: 2022.04.02).

[41] Pijnenburg, M and Kowalczyk, W. “Extending an anomaly detection benchmark with auto-encoders, isolation forests, and rbms”. In: *International Conference on Information and Software Technologies*. Springer, 2019, pp. 498–515.

[42] Ramaswamy, S, Rastogi, R, and Shim, K. “Efficient algorithms for mining outliers from large data sets”. In: *SIGMOD record* 29.2 (2000), pp. 427–438.

[43] Reynolds, D. “Gaussian Mixture Models”. In: Encyclopedia of Biometrics. Boston, MA: Springer US, 2009, pp. 659–663.

[44] Safavian, S.R and Landgrebe, D. “A survey of decision tree classifier methodology”. In: *IEEE transactions on systems, man, and cybernetics* 21.3 (1991), pp. 660–674.

[45] Saito, T and Rehmsmeier, M. “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets”. In: *PloS one* 10.3 (2015).

[46] Sugiyama, M. *Introduction to statistical machine learning*. 2015.

[47] Tan, P.-N et al. *Introduction to data mining*. 2nd. Harlow: Pearson, 2020.

[48] Tang, B and He, H. “A local density-based approach for outlier detection”. In: *Neurocomputing* 241 (2017), pp. 171–180.

[49] Tang, J et al. “Enhancing Effectiveness of Outlier Detections for Low Density Patterns”. In: vol. 2336. Advances in Knowledge Discovery and Data Mining. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 535–548.

[50] Zhang, H et al. “A scale-adaptive positive selection algorithm based on B-cell immune mechanisms for anomaly detection”. In: *Engineering Applications of Artificial Intelligence* 94 (2020).

[51] Zhang, Z et al. “Opening the black box of neural networks: methods for interpreting neural network models in clinical applications”. In: *Annals of translational medicine* 6.11 (2018).

[52] Zimek, A and Filzmoser, P. “There and back again: Outlier detection between statistical reasoning and data mining algorithms”. In: *WIREs Data Mining and Knowledge Discovery* 8.6 (2018).

Appendix A

Table 14: Outlier detection results for all the instruments in the Scila dataset

Security	# of points	# of outliers
SPY X315	2132	135
SCILA SEK (Consolidated-ALL)	2226	54
CME Globex: NBP Apr 2021	2155	125
SPY L305	2100	76
GBP/USD	4484	150
BTC/USD	4673	150
ALI APR2021	4623	140
SPY J305	2130	96
GOOG J1520	2106	85
GOOG (Cl C)	3508	103
CME Globex: NBP May 2021	2096	132
USD/SEK	2141	80
EUR/USD	4469	147
CME Globex: NBP Mar 2021	4480	149
PETROLEOS MEXICANOS EURO 5.5% 24/02/25 (RFQ)	375	19
SPY X305	2106	70
CME Globex: NBP Jul 2021	2165	99
ERIC.A	1882	60
IBM	335	1
VOD GBP (Consolidated-ALL)	822	24
SPY V305	2117	70
ALI MAR2021	4606	154
SPY K300	2121	76
CME Globex: NBP Jun 2021	2150	99
USD/SEK (RFQ)	519	27
TSLA (Consolidated)	434	6
VOD (Consolidated)	297	6
ASX 90 Bank Accepted Bill JUN2021	700	20
AAPL (Consolidated)	466	8
SPY K305	2153	89
TSLA-2 (Consolidated)	356	10

ASX 90 Bank Accepted Bill MAR2021	533	8
VOD (Dark)	540	13
SCILA	1840	56
BCH/USD	546	7
TSLA (Internal)	535	6
TSLA USD (Consolidated-ALL)	973	12
GOOGL (Cl A)	379	7
BANK OF AMERICA CORPORATION 4% 22/01/25 (RFQ)	422	24
SEC.A	542	11
SPY J310	514	9
GOOGL (Cl A) (Consolidated)	476	7
SPY W305	2116	81
IBM (Consolidated)	432	3
BTC/EUR	2242	96
AAPL 2.5 04/2023	532	4
SPY	376	2
SCILA-3	543	7
IRS Libor 1Y	623	31
SCILA-2 (Consolidated)	308	4
CEGH VTP JUL20	650	9
TSLA	337	3
SPY (Consolidated)	473	4
AAPL	376	7
TSLA-2	266	6
CDX.NA.IG 5Y 31-1 (RFQ)	283	18
SEB.A (Consolidated)	343	4
SPY V300	515	7
BACORP.SNRFOR.USD.XR14.100.2023-12-20 (RFQ)	352	12

Appendix B

Isolation Forest

```
import numpy as np
import pandas as pd
import random


def select_feature(data):

  """
        Selects a random feature in the dataset.

        Parameters
        ----------
        data : The complete dataset as type pandas.dataframe

        Returns
        -------
        feature : Randomly selected feature name as type str

        """

     feature = random.choice(data.columns)

    return feature



def select_value(data, feature):

  """
        Selects a random value whitin the range of maximum feature value
        and minimum feature value.

        Parameters
        ----------
        data : The complete dataset as type pandas.dataframe
        feat : Feature from where the value should be selected as type str.

        Returns
        -------
        split_value : Randomly selected value as type float.

        """

    mini = data[feature].min()
    maxi = data[feature].max()

    split_value = (maxi-mini)*np.random.random()+mini

    return split_value



def split_data(data, split_feature, split_value):
```

```
"""
    Splits the whole data set (all features) accordding to split_value for
    split_feature in to two dataset, one with values below and one with
    values above.

    Parameters
    ----------
    data : The complete dataset as type pandas.dataframe
    split_feature : Feature that should be splitted as type str
                    (output from function select_feature(data).
    split_value : Randomly selected value within the feature range as
                  type float (output from function select_split(data, feature).


    Returns
    -------
    data_below : Resulting data points for the complete dataset below the
                 splitvalue in the feature as type pandas.dataframe
    data_above : Resulting data points for the complete dataset below
                 the splitvalue in the feature as type pandas.dataframe

    """

data_below = data[data[split_feature] <= split_value]
data_above = data[data[select_feature] >  split_value]

return data_below, data_above



def isolation_tree(data, max_depth, counter=0, random_subspace=False):

"""
    Recursively iterates through the data set with the goal of isolating the
    data points in branches. For each iteration a feature is splitted according
    to a splitvalue in order to binary divide the dataset and create branches.
    Each branch is stoped if there is only one data point in the branch or the
    tree has reached the maximum depth.

    Parameters
    ----------
    data : The complete dataset as type pandas.dataframe
    max_depth: Tells the function when a branch should stop growing if no data
     point is isolated , as type float.


    Returns
    -------
    sub_tree : Returnes nested dictionaris with split features and split values
    to mimics a tree.

    """

# End Loop if max depth or isolated
if (counter == max_depth):
# Data isolated when subset only contains of one row (one point)
    return 'Early Leaf', data.shape[0]
```

```
elif data.shape[0]<=1:
    return 'Leaf'
else:
    counter +=1 # Current tree height for the data set

    # Select feature
    split_feature = select_feature(data)

    # Select value
    split_value = select_value(data,split_feature)

    # Split data
    data_below, data_above = split_data(data,split_feature,split_value)
    # Instantiate sub-tree
    question = "{} <= {}".format(split_feature, split_value)
    # String: "split feature (eg.Total Traded Volume) <= split_value (eg.23)"
    sub_tree = {question: []}
    # Creates empty list for the question to store points fulfilling the question,
    #the subtree

    # Recursive part
    below_answer = isolation_tree(data_below, counter, max_depth)
    #Run the whole algorithm for the splitted data set "data_below"
    #set recursively. Code will not go to next line before this is done for all data.
    above_answer = isolation_tree(data_above, counter, max_depth)
    #When the "data_below" is done this function will run the whole algorithm
    #for the splitted data set recursively

    #Appends subtree to subtree
    sub_tree[question].append(below_answer)
    sub_tree[question].append(above_answer)

    return sub_tree
def pathLength(instance,iTree,path=0,nodepath=None, node_traverse="", data_list=None):

    """
    Calculates the path for a observation traversing down through a tree.

    Parameters
    ----------
    instance : One observation from dataset.
    iTree: The tree in which the observations path is being calculated.

    The other paramters are to save variables between each iteration in the recursive
    part.

    Returns
    -------
    path : Path lenght for an instance as int.
    nodepath : Traversed path for instance, with split features, split values and
    split directions, as a list of lists.
    """

 # path is a counter for each iteration down the nodes left or right
path=path+1

#First split in tree
question = list(iTree.keys())[0]
```

```
# list(iTree.keys())[0] // question = the key to the subtree, or the first split
# of the tree.
# The iTree then has a list of dictionaries and lists within itself, which can be
#reached recursively by
# Splitting apart the tree starting from the (top) key i.e question.

# Details of the first split
feature_name, comparison_operator, value = question.split()


# example[feature_name].values = data point value of the feature used in split,
# compared to split value
point_value = example[feature_name].values

if point_value[0] <= float(value):
    #subtree to the left. Out of 2 lists with dictionaries, choosing the
    #first one (left)
    answer = iTree[question][0]
    node_traverse ="left"
else:
     #subtree to the right. Out of 2 lists with dictionaries, choosing the second
     #one (right)
    answer = iTree[question][1]
    node_traverse ="right"

if nodepath is None:
    nodepath=[]
nodepath.append([feature_name,value,node_traverse])
if data_list is None:
    data_list=[]
data_list.append(point_value[0])

# Base case, checks if an instance (answer) is an instance or subclass of
# class (dictionary).
# If not leaf node is reached. I.e if the question (top) key contains no more
# lists of dictionaries.


if not isinstance(answer, dict):
    if isinstance(answer,str):
        return path,nodepath,data_list
    else:

        if answer[1] == 0:
            pass
        else:
            c_nt = c_n(answer[1])
            path = path + c_nt
        return path,nodepath,data_list

# recursive part. Function runs on subtree to previosuly subtree.
else:
    # residual_tree becomes the left or the right subtree, depending
    # on the previous if / else.
    residual_tree = answer
    return pathLength(instance, residual_tree, path=path, nodepath=nodepath,
                          node_traverse=node_traverse,data_list=data_list)
```

```
def isolation_forest(df, threshold, n_trees=100, subspace=256):


        """
        The function creates n-trees, by calling on the function isolation_tree,
        from a given subspace of the original dataset. Afterwards each datapoint
        in the original dataset is traversed down each tree. For every datapoint
        in each tree the path length is calculated by use of the function path_length.
        The average path length for each point from all the trees forms the basis of
        the outlier score. In the last part of the code the function also
        runs the result on MI-Local DIFFI to see the feature imortance for each point.

        Whitin the function a couple of support function is also run. This includes
        the c(n) function which calculates the estimated path length in nodes. It also
        runs the functions dictionary and dictionary_test. Two support functions to
        create nested dictionaries to store paths for each point update no and tree.

        The function also calls on MI_Local_DIFFI(outlier_point, forest, df, list_of_dicts,
        list_of_datavalues, subspace) for the outlying observation.

        Parameters
        ----------
        df : The origibal dataset as type pandas.dataframe
        threshold: Determines the outlier score for classifying outliers.
        n_trees: Number of trees , deafult = 100 (from original paper)
        subspace: Size of subsample, default = 256 (from original paper)

        Returns
        -------
        anomaly score: The anomaly score for each point in the dataset, as a
                       dictionary.
        final: Nested dictionary with observation number as primary key and
                features as nested keys with score as value.
        """

#Given an input data, a number of trees and a sampling size (how many data
# is fed to each tree), we fit
# as many trees as desired and return a forest

    forest = []
    anomaly_score = {}
    n = subspace
    c = c_n(n)
    list_of_dicts = np.zeros(n_trees,dtype=object)
    list_of_datavalues = np.zeros(n_trees,dtype=object)
    final = {}


    max_depth = np.ceil(np.log(subspace))

    for i in range(n_trees):
        list_of_dicts[i] = {}
        list_of_datavalues[i] = {}
        # Sample the subspace
```

```
        dataFrame = df.sample(subspace)

        # Fit tree
        tree = isolation_tree(dataFrame, max_depth=max_depth)

        # Save tree to forest
        forest.append(tree)

    for i in range(df.shape[0]):
        instance = df.iloc[[i]]
        paths = []
        node_paths = []
        list_of_path_lengths = []
        j = 0

        for tree in forest:
            paths.append(pathLength(instance,tree))
            node_paths.append(paths[j][1])
            # For each tree append path and for each point update no. for path and
            # add new nodes
            list_of_dicts[j] = dictionaries(node_paths[j],list_of_dicts[j])
            #Stores each points path length for each tree in a list
            list_of_path_lengths.append(paths[j][0])
            list_of_datavalues[j] = dictionaries_test(node_paths[j],list_of_datavalues[j]
                                    ,paths[j][2])

            j+=1


        E = np.mean(list_of_path_lengths)
        s = 2**-(E/c)

        anomaly_score[i] = s

    sorted_anomaly_score = dict(sorted(anomaly_score.items(), key=lambda x: x[1],
                            reverse=True))

    start = 0
    for value in list(sorted_anomaly_score.values()):
        if value > threshold:
            outlier_point = (list(sorted_anomaly_score.keys())[list(
                                sorted_anomaly_score.values()).index(value,start)])
            result = MI_Local_DIFFI(outlier_point, forest, df, list_of_dicts,
                                    list_of_datavalues, subspace)
            final[outlier_point] = result
        start += 1
    return anomaly_score, final



def c_n(n):

  """
        Estimates path lenght of nodes that have not grown to full height to improve
        computioanl efficency.

        Parameters
```

```
        ----------
        n : Number of observations

        Returns
        -------
        c : Estimated path length

        """


    if n == 1:
        c = 0
    elif n == 2:
        c = 1
    else:
        c = 2.0*(np.log(n-1)+np.euler_gamma) - (2.0*(n-1.)/(n*1.0))
    return c



def dictionaries(items,dikt):
  """
        Transforms lists (paths) containing several strings from items to lists
        with only one string. Each string in each list is added as key to a
        empty dictionary together with occurence as value.

        Parameters
        ----------
        items : Traversed path for instance, with split features, split values
                and split directions, as a list of lists.
        dikt : A empty dictionary from place i i list_of_dicts

        Returns
        -------
        a : Dictionary of split from path (str) as key and occurence as value (int)

        """

    a = dikt

    for group in items:
        [s, i, j] = group
        combined = '%s,%s,%s' % (s, i, j)
        if combined in a:
            a[combined] += 1

        else:
            a[combined] = 1
    return a


def dictionaries_test(items,dikt,data_values):
  """
        Transforms lists (paths) containing several strings from items to lists with
        only one string. Each string in each list is added as key to an empty
        dictionary together with data_values.

        Parameters
```

```
    ----------
    items : Traversed path for instance, with split features, split values and
            split directions, as a list of lists.
    dikt : A empty dictionary from place i i list_of_dicts.
    data_values: A list of all the observations values for each split.

    Returns
    -------
    a : Dictionary of split from path (str) as key and occurence as
        data_values (list)

    """

p = 0
a = dikt
for group in items:
    [s,i,j] = group
    combined = '%s,%s' % (s,i)
    if combined in a:
        a[combined].append(data_values[p])
        p+=1
    else:
        a[combined] = [data_values[p]]
        p+=1
return a
```

MI-Local-DIFFI

```
def path_length_indicator(path_length, subspace):
 """
        Calculates the path lentgh indicator used in MI-Local-DIFFI

        Parameters
        ----------
        path_length : The path length of an observation as float.
        subspace : Specified subspace for isolation forest.

        Returns
        -------
        p_l_I : Path lentgh indicator score as float

        """


    n = subspace
    PL_lower = 1
    PL_upper =  2.0*(np.log(n-1)+0.5772156649) - (2.0*(n-1)/n) #Average path length
    p_l_i = max(0.1, min(1, 1 - ((path_length-PL_lower)/(PL_upper-PL_lower))))

    return p_l_i


def splitPropotion(q,q0):
 """
        Calculates the split proprotion indicator used in MI-Local-DIFFI

        Parameters
        ----------
        q : Number of observation in parent node.
        q0 : Number of observation in child node.


        Returns
        -------
        sp : Split proportion indicator score as float

        """
    if q == 2:
        sp = 0
        return sp
    else:
        sp = 1-((q0-1)/(q-2))
        return sp



def split_interval_length_indicator(split_value,split_direction, max_value, min_value):
 """
        Calculates the split interval lentgh indicator used in MI-Local-DIFFI

        Parameters
        ----------
        split_value:  Split value in a split as float
        split_direction: If observation traverse right or left in the tree.
```

```
    max_value:  Max observation value for feature.
    min_value; Min observation value for feature.

    Returns
    -------
    ws : Split interval length indicator score as float.

    """
split_value = float(split_value)
interval = abs(max_value-min_value) #feature intervall
a_interval = abs(split_value-min_value)
b_interval = abs(max_value-split_value)


if split_direction == "left":
    s = a_interval/interval

else:
    s = b_interval/interval

ws = 1.5 - (1/(s+1))
return ws


def MI_Local_DIFFI(outlier, IF, df, list_of_dicts, list_of_datavalues, subspace):
"""
    Calculates MI

    Parameters
    ----------
    outlier: Outlier observation from isolation forest as int.
    IF: The complete isolation forest created with isolation_forest.
    list_of_dicts: List of dictionaries where each list item represents a
                   tree from isolation forest with a dictionary of all
                   splits from all observations.
    list_of_datavalues: A list of dictionaries where each list item represents
                        a tree from isolation forest with a dictionary of
                        all the observations values for each split.
    subspace: Specified subspace for isolation forest.

    Returns
    -------
    DIFFI_score : Dictionary of with each feature as key and a score as value.

    """


w_PL = [] # Initiation of path legnth indicator score array
w_SP = [] # Initiation of split proportion indicator score array
w_SI = [] # Initiation of split interval length indicator score array

#Let G_1, ..., G_nF be the unique features used to create the isolation forest.
#We want to compute the feature importance for each _k, k = 1, ..., nF .
features = list(df.columns.values)

#Initialise feature importance vector FI = 0 with length equal
# to number of features nF .
FI = [0] * len(features)
```

```
    #Initialise feature occurence vector occurrence(F) = 0 with length
    # equal to number of features nF.
    occurrence = [0] * len(features)

    # Values for outlierpoint.
    instance = df.iloc[[outlier]]

    paths = []
    i = 0

    for tree in IF:
        path_features = []
        dict_features_PL = {}

        #Path length and path for the a enskild tree and a enskild outlier point
        paths.append(pathLength(instance,tree))
        for split in paths[i][1]:
            # Let path_features be the features that are used to split in each node
            # in Path(o, i)
            path_features.append(split[0])

#################################################################################
        # a) Path length indicator
#################################################################################
        path_length = (paths[i][0])# The length of the outlier's path in each tree
        # Returns travaers path length in tree for outlier
        w_PL = (path_length_indicator(path_length, subspace))

        for feature_name in path_features:
            dict_features_PL [feature_name] = w_PL

#################################################################################
        # b) Split proportion indicator
#################################################################################

        traverse_path = paths[i][1]   #Returns traverse path in tree for outlier
        child_index = len(traverse_path) - 1

        w_SP = []
        dict_features_SP = {}

        for item in traverse_path:
            if child_index == 0:
                child_node = traverse_path[child_index]
                str_child_node = child_node[0] +','+ child_node[1] +','+ child_node[2]
                #If traverse_path = length 1 the parent node will contain all points
                q = len(range(df.shape[0]))
                q0 = list_of_dicts[i][str_child_node]

                w_SP.append(splitPropotion(q,q0))

                child_index -= 1

            else:
                parent_index = child_index - 1
                child_node = traverse_path[child_index]
                parent_node = traverse_path[parent_index]
```

```
                str_child_node = child_node[0] +','+ child_node[1] +','+ child_node[2]
                str_parent_node = parent_node[0] +','+ parent_node[1] +','+ parent_node[2]

                q = list_of_dicts[i][str_parent_node]
                q0 = list_of_dicts[i][str_child_node]
                w_SP.append(splitPropotion(q,q0))

                child_index -= 1
        s=0

        for feature_name in path_features:
            if feature_name in dict_features_SP.keys():
                dict_features_SP[feature_name].append(w_SP[s])
            else:
                dict_features_SP[feature_name] = [w_SP[s]]
            s+=1

#############################################################################################
        # c) Split interval indicator
#############################################################################################


        w_SI_path = []
        dict_features_SI = {}

        for item in traverse_path:

            print(item)
            split_feature = item[0]
            split_value = item[1]
            split_direction = item[2]
            feature_item = split_feature +','+ split_value
            datapoint_values = list_of_datavalues[i][feature_item]
            max_val = max(datapoint_values)
            min_val = min(datapoint_values)


            w_SI_path.append(split_interval_length_indicator(split_value, split_direction,
                                                             max_val, min_val))

        s=0

        for feature_name in path_features:
            if feature_name in dict_features_SI.keys():
                dict_features_SI[feature_name].append(w_SI_path[s])
            else:
                dict_features_SI[feature_name] = [w_SI_path[s]]
            s+=1


#############################################################################################
        # Calculate sigma(k) feature importance score for each feature in the
        # tree and point
#############################################################################################

        sigma = [0] * len(features)

        for k in range(len(features)):
```

```
        if features[k] in list(path_features):
            w_LP_k = dict_features_PL[feature_name]

            #Let  be the location of the best split of feature  in (, ).
            w_SP_best_k = max(dict_features_SP[features[k]]) #(w_SP)
            w_SI_best_k = max(dict_features_SI[features[k]]) #(w_SI)

            occurrences_k = list(path_features).count(features[k])

            occurrence[k] = occurrence[k] + occurrences_k

            #Comment out for just path length
            #sigma[k] = w_LP_k * w_SI_best_k * w_SP_best_k

            #Comment out for MI-Local-DIFFI
            sigma[k] = w_LP_k


            FI[k] = FI[k] + sigma[k]
    i+=1

FI = np.array(FI)

occurrence = np.array(occurrence)
FI = FI/occurrence
DIFFI_score = dict(zip(features, FI))

return DIFFI_score
```