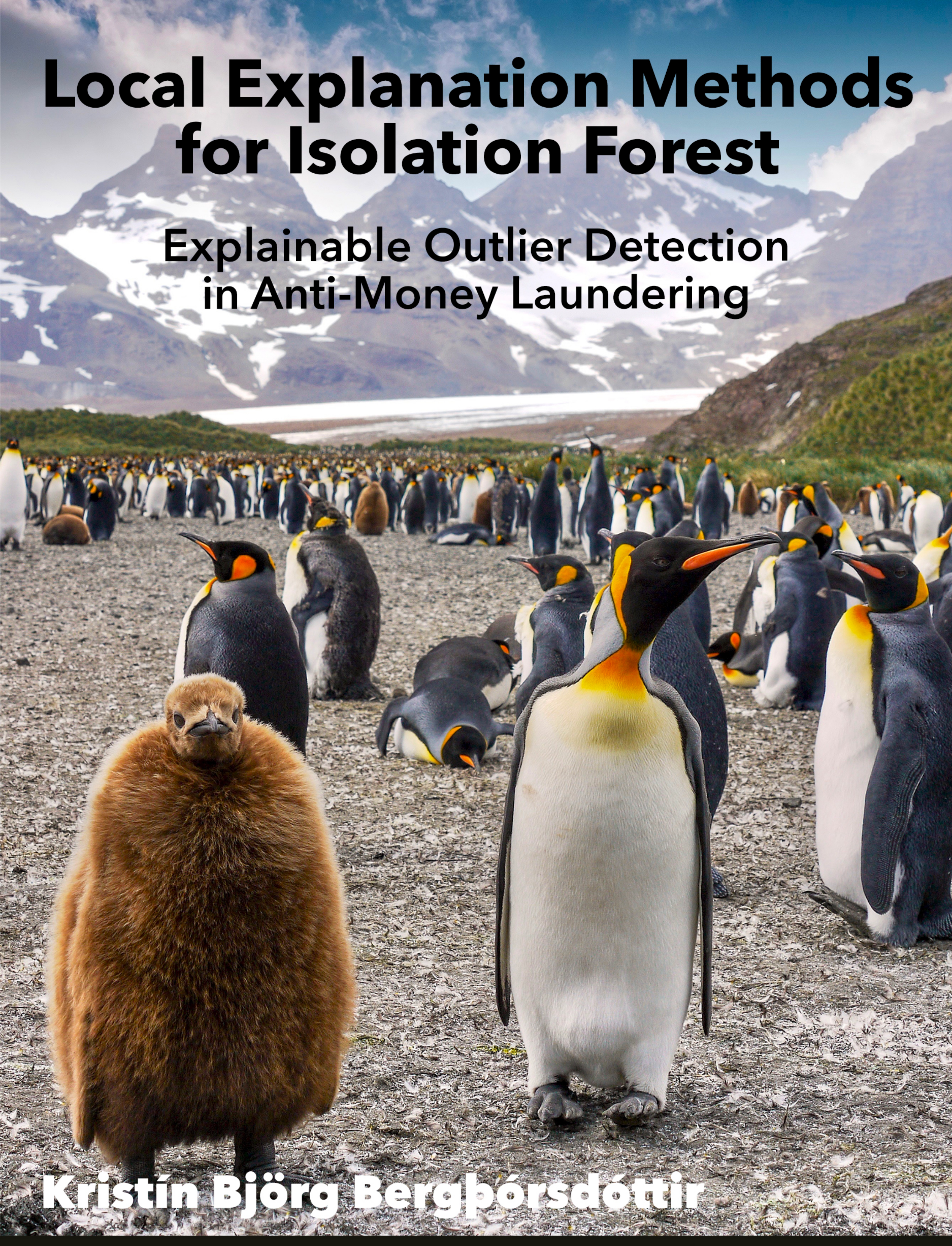
Local Explanation Methods for Isolation Forest

Explainable Outlier Detection in Anti-Money Laundering



Kristín Björg Bergþórsdóttir

Master of Science
Applied Mathematics

TU Delft Delft University of Technology

Local Explanation Methods for Isolation Forest

Explainable Outlier Detection in Anti-Money Laundering

by

Kristín Björg Bergþórsdóttir

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on August 28th 2020 at 15:00.

Student number:	4812808	
Project duration:	December 2019 – August 2020	
Thesis committee:	Prof. dr. ir. C. W. Oosterlee,	TU Delft, supervisor
	Dr. E. Haasdijk,	Deloitte, supervisor
	Dr. A. Fontanari,	CWI, Amsterdam
	Dr. N. Parolya,	TU Delft

Abstract

Machine learning methods like outlier detection are becoming increasingly more popular as tools in the fight against money laundering. In this thesis, we analyse the Isolation Forest outlier detection algorithm in detail and introduce a new local explanation method for Isolation Forest, the MI-Local-DIFFI (Multiple Indicator Local-DIFFI) method. The method uses the structure of the isolation trees and the traversal of individual outliers down the trees to determine an importance weight for each of the features. These weights are then combined into feature importance scores that are used to explain why a specific outlier is identified as such. In anti-money laundering (AML), such explanations are very valuable when determining whether an outlying customer is suspicious or not. MI-Local-DIFFI is based on a global explanation method called DIFFI and while we were conducting our research, another local version, Local-DIFFI, was also introduced. In the thesis, we use a synthetic data set to compare the performance of four different explanation methods including our MI-Local-DIFFI, Local-DIFFI and the state-of-the-art TreeSHAP method. Our MI-Local-DIFFI shows excellent results in terms of performance and runtime. Furthermore, we use a data set from Triodos bank to apply the explainable outlier detection methodology consisting of the combination of Isolation Forest and MI-Local-DIFFI. This resulted in interesting findings like the revealing of data quality issues in the current system and references to EDRs and SARs. However, after further inspection, no SARs were filed but some customers were put to higher risk classes. This procedure will be performed on a monthly basis with the goal of continuing to improve the AML processes of the bank.

Contents

List of Figures

List of Tables

List of Notation and Abbreviations

Notation

X	data set
n	number of observations in a data set
d	number of dimensions in a data set
x	observation/instance in a data set
F	feature in a data set
t	a tree object
T	number of trees
v	a node in a tree
ψ	subsampling size, sometimes equal to n
ℓ	height limit of a tree
e	number of edges from root node to leaf node
$h(x)$	path length of instance x
$c(n)$	average path length of isolation tree with n leaves
$s(x)$	outlier score of instance x
$E(n)$	average external path length of an isolation tree t.
$E_t(n)$	average external path length of a ternary isolation tree t.

Abbreviations

AML	Anti-Money Laundering
FATF	Financial Action Task Force
MI-Local-DIFFI	Multiple Indicator Local-DIFFI
AOF	Alter-One-Feature
AUC	Area Under ROC Curve
AUPRC	Area Under Precision-Recall Curve
FI	Feature Importance
MO	Modus Operandi
MTS	Money Transferring Services
SAR	Suspicious Activity Report
EDR	Event Driven Review

Preface

This report contains the results of my master thesis project, which I have been fortunate enough to work on for the past 9 months. I found this research both challenging and interesting and I learned numerous things on the way. I would like to express my sincerest gratitude to the people that have helped me with my research project. Prof. Kees Oosterlee, my academic supervisor, Dr. Evert Haasdijk, my supervisor at Deloitte and Dr. Andrea Fontanari at CWI Amsterdam, who all gave me very helpful feedback and helped shape this project into its current state. I also want to thank Dr. Nestor Parolya for being apart of my Thesis Committee and my co-workers at Deloitte and Triodos for providing me with a great working environment while I was working on this thesis.

Writing a thesis in the midst of a global pandemic has been interesting, to say the least. While I regret not having been able to meet my supervisors and co-workers more in person, I was fortunate enough to finish my research in Iceland surrounded by my family. Very special thanks go to my parents who have supported me in so many ways during the last 25 years. Their optimistic and joyful way of living has taught me something that cannot be learned from textbooks. I also want to thank my dad for providing me with office space to be able to concentrate on my thesis work. Thanks to my four wonderful younger siblings, I am proud to belong to such a great group. And finally, thanks to my lovely partner and best friend, Andri. From March until August we sat opposite each other working on our master thesis projects, for better company I could not have asked.

Kristín Björg Bergþórsdóttir
Hafnarfjordur, Iceland
August 2020

1 Introduction

Technical solutions for anti-money laundering (AML) have become increasingly popular in the last decade, especially with increased knowledge of the benefits of machine learning. Currently, most banks make use of a rule-based system to monitor AML risk. The rules are created manually based on expert opinion and historical cases of money laundering and thresholds are tuned by the use of historical data. Such a system has proven to be quite effective but it suffers from a lot of false positive alerts as well as inability to detect new risk.

To tackle the system's inability to detect new risk, we look at outlier detection methods. Such methods are less subject to specific rules and thresholds when it comes to finding suspicious customers. Specifically, we look at Isolation Forest, which is an increasingly popular outlier detection method mainly because of its performance, low computational complexity and non-parametric nature. We look into Isolation Forest in depth and see whether there are improvements to be made for optimal performance.

The understanding of why a particular customer is classified as an outlier can be enhanced by using explanation methods on top of the results from the outlier detection algorithm. We introduce a new method, MI-Local-DIFFI, which is a local adaptation of the previously introduced DIFFI [11] method. This method looks at the splitting structure of Isolation Forest and ranks the importance of the features in the data set based on their contribution to the outlier score of a particular observation. We compare the MI-Local-DIFFI to two other explanations methods, the TreeSHAP [45] and an intuitive method that is based on the paper by Breiman [6], which we refer to as AOF. The TreeSHAP is originally designed for tree-based supervised models but we test the performance on the outlier detection model Isolation Forest. As far as we know, this has not been done before in the literature.

We also recognize that the outlier explanations can help us understand the outlying aspects of the outliers which gives a better foundation for the improvement of the algorithm. It is much easier to improve a model that can be explained and is understood [1]. This is evidential in our research, as the explanations lead us to experimentation with a slightly altered version of Isolation Forest, the Ternary Isolation Forest, which has higher performance measures than the original version.

We mainly work with two data sets. First of all, a synthetic data set with ground truth outlier and outlying aspect information and second of all, a data set from Triodos Bank, an industry partner that is seeking to strengthen their AML system. The synthetic data set is used for validation purposes while the data set from Triodos is specifically engineered to represent certain money laundering risk factors and used to explore the benefits of applying outlier detection to uncover money laundering.

While writing up and finalising this thesis project, we found a new contribution on arXiv.org (21/7/2020) from the authors of the DIFFI explanations method [12]. A local variant of the DIFFI method is introduced and referred to as Local-DIFFI. Since this new contribution was introduced near the deadline of our thesis finalisation, we only briefly comment on their new methodology. Initially, we meant to call our new method the DIFFI-local, but because it is too similar to Local-DIFFI, we decide to refer to our method as Multiple Indicator Local-DIFFI (MI-Local-DIFFI) since it uses three indicators related to the tree structure of Isolation Forest to measure the importance. Further description of these methods is provided in section 3.2.1. The authors have provided the code for the Local-DIFFI method, so we can compare our method to theirs as shown in section 4.4. According to our experiments, our MI-Local-DIFFI method shows better performance than the Local-DIFFI method. Clearly, our MI-Local-DIFFI is thus a novel contribution to the literature on explainable methods for Isolation Forest.

1.1. Anti-Money Laundering

Recently, there have been multiple incidents in the Dutch financial sector where the monitoring of money laundering has been seriously lacking [65]. Insufficient monitoring of clients and their activity is a huge issue in the financial sector and failure to comply with AML legislation may result in large fines. In this chapter, we explain the concept of money laundering and the measures that have been taken to combat such criminal activity, commonly referred to as anti-money laundering (AML) measures.

The Financial Action Task Force (FATF) describes itself as being the global money laundering and terrorist financing watchdog. The FATF recommendations aim to prevent these illegal activities and the FATF monitors countries to ensure their compliance with these recommendations [20]. The FATF defines money laundering as the processing of criminal proceeds to disguise their illegal origin [21]. The United Nations Office on Drugs and Crime (UNODC) conducted a study to investigate the extent of this money laundering activity and estimated that 2.7% of global GDP or about $ 1.6 trillion was being laundered through the financial system in 2009 [52].

There are a few stages of money laundering, from when illegal funds enter the financial system up until the funds re-enter the legitimate economy [21].

1. **Placement:** The launderer enters the illegal profits into the financial system.
2. **Layering:** The launderer engages in movements of the funds to create a distance from their source.
3. **Integration:** The launderer re-enters the legitimate economy with the laundered funds, often investing in real estate, luxury assets or business ventures.

A lot can be done to fight money laundering. Most governments have established comprehensive AML regimes, aiming to increase the awareness within the public and the private business sector and providing the regulatory tools necessary to combat the problem. The financial sector plays a vital role in the fight against money laundering and must establish a financial transaction reporting system, customer screening and monitoring [21]. One part of the AML process is the ongoing due diligence and business relationship monitoring that requires financial institutions to monitor unusual and potentially suspicious transactions where any actual suspicion should lead to an obligation to report.

The monitoring of clients is mostly done using a rule-based system where customer activity is compared to fixed thresholds that determine whether the activity is unusual or not. These thresholds are determined by looking at historical activity and are often confined to certain peer groups of customers. The rule-based systems are very interpretable and work well in catching the most common known money laundering scenarios. There are however some disadvantages to such a system, like the possibility that criminals test the rules and thresholds and learn how to circumvent them and that the rules only consider a small subset of all the available data.

Triodos bank recognizes the importance of fighting money laundering and has therefore spent a lot of effort on improving its AML system and practices. Exploring the benefits of machine learning in AML is one part of these efforts and in this thesis, we explore whether outlier detection can be used to identify suspicious customers not detected with the rule-based AML system. Furthermore, we want to explore whether the use of explanation methods on top of outlier detection can help the analysts to pinpoint the outlying aspects of an outlier and thus making it easier for them to decide whether or not the outlier is suspicious or interesting and requires further investigation.

We are not only interested in the customers that are suspected to participate in illicit activity, but also the customers that in some way deviate from the rest. These customers could, for example, be using the bank's product in another way than what was initially intended. These customers could also be leading to false positive alerts in the rule-based system, making them interesting to detect in case they require further handling.

1.2. Research Questions

The main research question to be answered in this thesis is the following:

- **RQ1.** *To what extent can explainable outlier detection increase the ability of AML systems to detect suspicious customers?*

To answer the research question, the following sub-questions are addressed:

- **SQ1.** *Which outlier detection algorithm is best suited for our problem setting?*
- **SQ2.** *Which explanation method should be used to explain the results of the outlier detection algorithm on an individual prediction level?*

Although the main research question revolves around enhanced detection of suspicious customers, this research puts a lot of effort into the methods used. A deep dive into the Isolation Forest algorithm is performed and an analysis into the different ways of locally explaining the output of Isolation Forest. This aspect of the thesis is not less important than the specific use case in outlier detection.

1.3. Contributions

The contributions of this thesis are listed below.

1. We introduce a new explanation method, MI-Local-DIFFI, that measures feature importance on a local level in the Isolation Forest outlier detection algorithm. This method outperforms the very recent Local-DIFFI method when measured on a few synthetic data sets.
2. The performance of three different local explanation methods of Isolation Forest is evaluated on a synthetic data set that contains ground truth outlying feature information.
3. We introduce Isolation Forest with a ternary tree structure that shows better performance than the standard binary tree structure version.
4. We evaluate the performance of the explainable outlier detection methods on a real-world data set from the AML domain.

The first two points are direct contributions to research in explainable outlier detection, the third point is a contribution towards improvements of the popular Isolation Forest algorithm and the fourth point is a direct contribution towards advancements in detection of suspicious customers for financial institutions such as banks that have an obligation to monitor money laundering risks.

1.4. Thesis Outline

In the first chapter, the concepts of money laundering are introduced along with the measures in AML that have been taken by regulators and financial institutions. The research questions are put forward and the contribution of this research is discussed.

In chapter 2, the field of outlier detection is introduced and the pre-existing literature of such methods in an AML setting. Different classes of outlier detection methods are described and reviewed. Finally, an overview and taxonomy of explanation methods in machine learning is put forward with a special focus on outlier detection methods.

In chapter 3, the Isolation Forest outlier detection algorithm is explained in detail followed by an introduction of the explanation methods that are compared and evaluated later on in the thesis. The

MI-Local-DIFFI is our local adaptation of the method DIFFI [11] that looks at the splitting structure of Isolation Forest to formulate explanations for individual outlier predictions. The MI-Local-DIFFI method is compared to the TreeSHAP [45] and AOF explanation methods. The TreeSHAP has shown good performance for tree-based supervised models but has not yet appeared in the literature for Isolation Forest.

In chapter 4, the synthetic data sets are introduced. The synthetic data has outliers with ground-truth explanations of outlying features which can be used to evaluate the performance of the three explanation methods. We perform multiple experiments, both regarding the outlier detection process and the explanation process. We test the performance of Isolation Forest and perform some analysis of the subspace specific detection ability of Isolation Forest. We also identify a possibility of improvement using a ternary version of Isolation Forest. Finally, we compare the recently introduced Local-DIFFI method to our MI-Local-DIFFI.

In chapter 5, the methods described earlier are applied to the real-world data set from Triodos bank. The real-world data does not contain prior information about the ground-truth outliers or outlying features. With the help of a domain expert, we evaluate whether the outliers are truly interesting in terms of AML. The truthfulness of the explanations is also evaluated with an intuitive method described in the chapter.

In chapter 6, we present the conclusion of the thesis by answering our research questions and addressing the possibilities for future work. We also explain the limitations of this research project.

2
Background and Related Work

We look into the literature of machine learning and outlier detection and specifically analyse the use of these methods in anti-money laundering. There are a few different classes of outlier detection methods that use various ways to measure the outlierness of objects. Since we can not measure the performance of different algorithms on our unlabelled financial data set, we use a previous literature review into outlier detection algorithms to select an appropriate algorithm for our problem and thus manage to address the first research subquestion. The Isolation Forest algorithm performed very well compared to other algorithms and is both efficient and scalable. We thus chose to use this algorithm for our AML outlier detection project. Based on this choice, we also review the literature relating to explanations methods in outlier detection.

2.1. Introduction to Machine Learning

In the well-known book Machine Learning by Tom Mitchell [50], a broad definition of learning is put forward.

> A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [50, p. 2].

To define a specific machine learning problem we must specify these three items (E, T, P). In the checkers learning problem, these three items could for example be

- E: playing practice games against itself,
- T: playing checkers,
- P: percent of games won against the opponent.

Machine learning is often classified based on the amount of supervision used during training [24] as presented below.

Supervised Learning

In supervised learning problems, the training data that you feed to the algorithm includes the desired outputs (labels). The goal is thus to minimise $f(x) - y$ where f is the predictor that is trained to predict the correct output y based on the input data x. Supervised learning can further be divided into classification and regression. *Classification* models are used when the desired output is a class, like predicting whether an email is spam (1) or not (0), while *regression* models are used to predict numeric values, like the price of houses (given location, market conditions etc.). Examples of popular supervised learning algorithms are linear/logistic regression, decision trees, random forests, neural networks and support vector machines (SVM).

Unsupervised Learning
In unsupervised learning, the training is done without information about desired outputs or target values. There are a few types of tasks that can be solved with unsupervised learning, one of them being outlier detection. We introduce a few different tasks:

- **Clustering:** The goal is to learn the structure of the data and create clusters of similar observations.
- **Outlier detection:** The goal is to find observations that differ from the rest of the data. In some way the exact opposite of clustering.
- **Dimensionality reduction:** Learning the structure of the data to reduce the number of dimensions while retaining as much information as possible.

Other Types of Learning
Furthermore, there exists a hybrid version that combines these two classes called semi-supervised learning where a small proportion of the input data has labelled outputs but the majority is unlabelled. Such algorithms can work well on top of clustering for example where only one label per cluster is needed to identify all the elements in a cluster. Reinforcement learning is the final category and of a completely different nature than the others. The learning system is an agent that can observe the environment, select and perform actions which result in rewards or penalties. The goal is to learn the optimal strategy (policy) which the agent must do by trial and error. The policy defines which action an agent should perform in a given situation [24].

2.2. Outlier Detection

The term outlier is quite intuitive and most of us might have some preconceived idea of outliers. Hawkins refers to the intuitive definition of an outlier as "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [27, p. 1]. Outliers have been a statistical topic for centuries but as the field of data science started to grow about two decades ago, outliers have also become a hot topic within the data mining community. Zimek and Filzmoser [70] studied the difference between outlier detection from a statistical reasoning viewpoint, on one hand, and a data mining algorithmic viewpoint, on the other hand. The main focus of the data mining community was finding scalable methods for outlier detection resulting in the loss of probabilistic interpretation of outlier scores. Generally speaking, statistical methods are based on presumed distributions of objects which can sometimes be a disadvantage when the underlying assumed distribution does not fit correctly. The data-oriented research shifted the focus towards more algorithm-driven thinking with efficiency and scalability at the core. The field of outlier detection is thus situated somewhere at the boundary of data mining and statistics, where the best suitable method may be either of a statistical or algorithmic-driven nature depending on the application.

Connection to Supervised Learning
As mentioned in section 2.1, outlier detection is normally considered to be an unsupervised learning problem, where the outliers are not known beforehand. However, some data sets contain labelled outliers, which changes the problem to a two-class classification problem with imbalanced classes. In such cases, the problem becomes a supervised learning problem. In this research, we focus on the unlabelled problem, since that is the case in most real-world applications of outlier detection.

In outlier detection, we can distinguish between *instance-based learning methods* and *explicit generalization methods* [2]. In instance-based learning, the model is not constructed beforehand, but rather, each instance is given a score based on other related instances. A popular example is using the k nearest neighbour distance of a data point as the outlier score. On the other hand, explicit generalization methods use a two-step approach where (1) a one-class model is created and (2) each point gets a score based on this model.

Connection to Extreme Value Theory
Extreme value analysis of one-dimensional data can be a very basic form of outlier detection where outliers are specifically assumed to be values that are either very low or very high. However, outliers

are not necessarily defined as extremely low or high values within a set of observations but as Hawkins described, they are observations that deviate in some way from the rest. For example, looking at the following seven observations $\{1, 1, 2, 50, 98, 99, 99\}$, the value 50 would be considered an outlier but definitely not an extreme value since the mean of these observations is 50. There is thus a fundamental difference between extreme value theory and outlier detection but there are also some connections between the two fields. The outlier modelling algorithms often deliver their prediction by the means of an outlier score that represents the deviation of each observation from the rest. Extreme value theory can then be used as the final step to deviate the outliers from the rest as the outlier scores are now represented as univariate values where the values on the extremes correspond to outliers [2].

2.2.1. Methods

We have put forward an intuitive definition of an outlier, but how do we measure whether an observation deviates from the rest? No single correct definition of an outlier exists. Rather, there are a few ways to define outliers using different measures of outlierness like density, distance or isolation. We introduce a few different classes of outlier detection methods that are either of statistical or algorithmic nature. Some even combine aspects from both statistical and algorithmic theory. The methods are listed in table 2.1 and we discuss the properties of each method below. The column IB/EG refers to the categorisation of the method into instance-based (IB) and explicit generalisation (EG) methods while the column ST/AD refers to the categorisation of statistical-based methods (ST) and algorithm-driven (AD) methods.

Method	Description	Complexity	Example	IB/EG	ST/AD
Density-based	Deviation is measured by looking at the number of observations in certain regions of the data space.	$O(N^2)$ but can be reduced in some cases.	Local-Outlier-Factor (LOF)	IB	AD
Distance-based	Deviation is measured by some distance measure.	$O(N^2)$ but can be reduced in some cases.	k-nearest neighbour	IB	AD
Clustering-based	Clustering is performed first and then the distance from cluster centers is used as a measure of deviation.	$O(N)$ but depends on the clustering algorithm	K-means clustering with Euclidean distance measure	IB	AD
Model-based	A model is created to represent normal observations and outliers correspond to the observations that do not fit the model	Depends on the method.	Gaussian Mixture Model, autoencoders	EG	ST,AD
Isolation-based	The susceptibility of observations to be isolated from other observations by means of random splitting	$O(N)$	Isolation Forest	EG	AD
Other	Methods that do not fall under any of the categories listed above	Depends on the method.	One-Class Support Vector Machine, Angle-based Outlier Detection	-	-

Table 2.1: Overview of outlier detection methods.

Below, we denote observations with x and y, a data set with X and the size of a data set with n and d where n refers to the number of observations and d to the number of dimensions. Furthermore, observation i in the data set X is referred to as x_i and its value in dimension j is denoted as x_{ij}.

Density-based Methods

Density-based outlier detection methods measure the deviations by looking at the number of observations in certain regions of the data space. The most commonly known density-based outlier detection method is the Local-Outlier-Factor (LOF) method [7]. The LOF gives each observation x in data set

X a score based on its local density. Let $dist(x, y)$ be some distance metrics between two observations x and y. We define the distance to the k-nearest neighbor of x in X, denoted $d_k(x)$ as the k-th smallest $dist(x, y)$ when computed for all $y \in X$. Let $S_k(x)$ be the set of observations that are within the k-nearest neighbor distance of x. Then the reachability distance of x with respect to y is defined as the maximum of $dist(x, y)$ and $d_k(y)$,

$$r_k(x, y) = \max(dist(x, y), d_k(y)), \tag{2.1}$$

so the reachability distance is smoothed out by using the k-nearest neighbor distance when x and y are close to each other. The average reachability distance of observation x, $\bar{r}_k(x)$, is then defined as

$$\bar{r}_k(x) = \frac{1}{|S_k(x)|} \sum_{y \in S_k(x)} r_k(x, y). \tag{2.2}$$

The Local Outlier Factor is then equal to the average ratio of $\bar{r}_k(x)$ to the corrresponding values of all the points in $S_k(x)$,

$$LOF_k(x) = \frac{1}{|S_k(x)|} \sum_{y \in S_k(x)} \frac{\bar{r}_k(y)}{\bar{r}_k(x)}. \tag{2.3}$$

This technique follows the assumption that the density around a normal observation is similar to the density around its neighbours but the density around an outlier is different from the density around its neighbours [2]. The main drawback of this method is that for high-dimensional data sets, the time complexity is of the order $O(n^2)$ which can be limiting for data sets with many observations. For lower-dimensional data sets, this time can, however, be reduced to $O(n)$ or $O(n \cdot \log n)$. Density-based methods belong to the class of instance-based methods.

Distance-based Methods

Distance-based methods measure the outlierness of objects using some distance measure, as the name suggests and belong to the class of instance-based methods. The k-nearest neighbor distance is a popular metric. However, to define the k-nearest neighbor distance of a point, we first have to choose some distance metric as a basis. Most commonly, the Euclidean distance is used. Suppose we have a data set X with values in $\mathbb{R}^{n \times d}$. The Euclidean distance between two observations of X, $x_i = (x_{i1}, .., x_{id})$ and $x_j = (x_{j1}, .., x_{jd})$, is defined as

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_{il} - x_{jl})^2}. \tag{2.4}$$

The Euclidean distance is thus equivalent to the L^2 norm, but any L^p norm can also be used as the basis distance. For each observation x in X, the k-th lowest distance to the rest of the observations is then used as a measure of outlierness, where larger scores indicate increased probability of being an outlier. Let

$$d(x_i) = (d(x_i, x_1), d(x_i, x_2), ..., d(x_i, x_n)) \tag{2.5}$$

and let $d_s(x_i)$ be the vector that contains the same elements of $d(x_i)$ except sorted in increasing order such that $d_s(x_i)^{(j)} \leq d_s(x_i)^{(l)}$ for $j < l$. Then the k-nearest neighbor distance of the observation x_i is defined as

$$k\text{NN-distance}(x_i) = d_s(x_i)^{(k)} \tag{2.6}$$

The most basic methods use this exact k-nearest neighbor distance for a predetermined k as the outlier score but since the best value of k is often difficult to estimate in unsupervised problems, the average k-nearest neighbor distance for range of values $k \in K$ is chosen as the outlier score. We define the average k-nearest neighbor score as

$$\text{average-}k\text{NN-distance}(x_i) = \frac{1}{|K|} \sum_{k \in K} d_s(x_i)^{(k)} \tag{2.7}$$

where the set of integers K has to be determined beforehand.

The biggest drawback of these methods is that the distance between all pairs of observations must be computed, resulting in an $O(n^2)$ computation time. This can be partly solved by using sampling methods, reducing the computation time to $O(s \cdot n)$ for a sample of size s such that $s \ll n$.

Clustering-based Methods

There is a somewhat complementary relationship between clustering and outliers, with the most simplistic view being that there are two types of observations, outliers and clustered observations. In many cases, the outliers are even a side-product of the clustering algorithm, often evaluated based on their distance to the nearest cluster centre. An important advantage of clustering methods over distance-based methods is increased time efficiency but they tend to work better for larger data sets where the level of detail is not as important. Clustering methods also suffer from their variability because of the randomness of the cluster generation process making it essential to compute an average outlier score from multiple executions to ensure their robustness.

One method combines the K-means clustering and a distance function to compute outlier scores. Given n observations $(x_1, x_2, ..., x_n)$ the K-means algorithm will divide the observations into K different sets $S = \{S_1, ..., S_K\}$ such that the sum of squares within each set is minimised. Let μ_j be the mean of the observations in set S_j and σ_j^2 be the variance. Formally, the goal is to find an optimal set allocation S such that

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - \mu_i\|^2 = \underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} |S_i| \sigma_i^2. \tag{2.8}$$

Once the sets or clusters have been optimally decided based on the criterion above, the distance to the cluster centroid is calculated for each point and used as an outlier score. For an observation x_i we thus find out which cluster S_j it belongs to and we get

$$K\text{-means Outlier Score} = d(x_i, \mu_j) \tag{2.9}$$

where the distance metric d is, for example, the Euclidean distance. Clustering-based methods are instance-based methods but often less complex than the first two methods because for each observation the only distance measure required is to the corresponding cluster center instead of to the entire data set. The overall complexity depends on the clustering algorithm being used but the complexity should not have to exceed $O(n)$.

Model-based Methods

The class of model-based outlier methods relies on creating a model to represent normal observations and looks at outliers as the observations that do not fit the model [35]. We introduce a two of these model-based methods, probabilistic methods and neural networks, which both belong to the class of explicit generalization method.

(i) *Probabilistic Methods*: The probability density function of the data set X is estimated by inference of the model parameters λ. The points that have the smallest likelihood $P(\lambda|x)$ are classified as outliers. A key assumption for these types of models is the specific choice of the underlying data distribution, which may not always be a trivial choice [2]. Examples of probabilistic methods in outlier detection are Gaussian mixture models (GMM) and Dirichlet process mixture models.

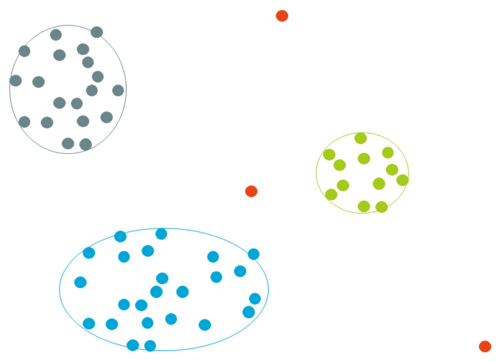


Figure 2.1: Gaussian mixture model for outlier detection, in this case there are three Gaussian blobs and the outliers are located in low density areas.

We describe further how GMMs can be used for outlier detection [59]. The density of the GMM is given by

$$p(x|\lambda) = \sum_{j=1}^{M} w_j \, g(x|\mu_j, \, \Sigma_j) \tag{2.10}$$

where x is a d-dimensional observation, w_j, $j = 1, ..., M$, are the mixture weights, and $g(x|\mu_j, \Sigma_j)$, $j = 1, ..., M$ are the Gaussian densities,

$$g(\mathrm{x}|\mu_j, \, \mathit{\Sigma}_j) = \frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left\{ -\frac{1}{2} (\mathrm{x} - \mu_j)' \, \Sigma_j^{-1} \, (\mathrm{x} - \mu_j) \right\} \tag{2.11}$$

where μ_j and Σ_j are the mean and variance of the corresponding Gaussian density. The mixture weights sum up to 1, $\sum_{j=1}^{M} w_i = 1$. We represent the three parameters of the GMM with $\lambda = \{(\mu_j, \Sigma_j, w_j), j = 1, ..., M\}$.

Given a data set X, the parameters of the GMM must be estimated which is most commonly done with maximum likelihood estimation. Before estimating the parameters, we must determine the number of gaussian densities M that the GMM is comprised of. The estimation of M can however be difficult, especially for high-dimensional data that is hard to visualise. For a set X of n observations $x_1, ...x_n$, the GMM likelihood can be written as

$$p(\lambda|X) = \prod_{i=1}^{n} p(\lambda|\mathrm{x}_i). \tag{2.12}$$

This expression is a non-linear function of the parameters so the maximum likelihood estimated parameters can not be obtained directly but are often calculated with the use of an expectation-maximization (EM) algorithm as described in the paper [16]. The basic idea of the EM method for maximum likelihood estimation is beginning with some λ and iteratively find a λ' such that $p(\lambda'|X) \geq p(\lambda|X)$. In each iteration, the parameters are estimated such that a monotonic increase in the model's likelihood is guaranteed.

Once the parameters have been estimated, $\hat{\lambda} = \{(\hat{\mu}_j, \hat{\Sigma}_j, \hat{w}_j), j = 1, ..., M\}$, the outlier scores are computed using the likelihood function $p(\hat{\lambda}|X)$. The outlier score of an observation x_i

$$\text{GMM outlier score } = p(\hat{\lambda}|\mathrm{x}_i) = \sum_{j=1}^{M} \hat{w}_j \; g(\mathrm{x_i}|\hat{\mu}_j, \, \hat{\mathit{\Sigma}}_j). \tag{2.13}$$

where a lower score means increased outlierness.

The main drawback of such methods is the assumption about the distribution of the underlying data. For example, in a Gaussian mixture model (GMM), the underlying data is assumed to follow a mixture of Gaussian distributions and the outliers are the points that do not fit that distribution [2]. In figure 2.1, we see that the three outliers can be identified effectively using a GMM with $k = 3$ components. In higher dimensional spaces, the choice of k is, however, less obvious which can result in sub-optimal models.

(ii) *Neural Networks*: Autoencoders have been used recently for the detection of outliers. An autoencoder is a neural network that is typically layered and symmetric such that the number of input nodes is equal to the number of output nodes. The nodes in the middle layer are fewer than in the input layer and the goal of the autoencoder is to produce an output that reconstructs the input. The reconstruction loss is then used to identify outliers since they are harder to reconstruct and consequently get a higher reconstruction loss [13]. An example of an autoencoder is shown in figure 2.2. Suppose we have an observation x that is put through the autoencoder, resulting in an output of x'. The corresponding outlier score of observation x is then based on the reconstruction loss, $L(x, x')$.

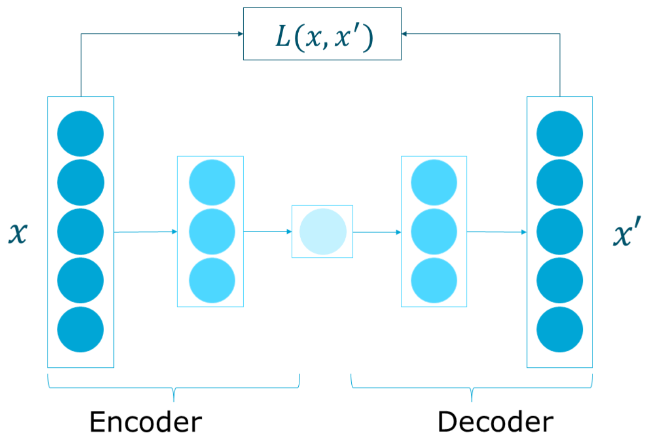


Figure 2.2: An autoencoder with 3 hidden layers.

Given an observation x, we let $f_e(x, \theta_e)$ be the output of an encoder network with parameters θ_e and for some output y of the encoder, we let $f_d(y, \theta_d)$ be the output of a decoder network with parameters θ_d. The output of the autoencoder can thus be computed as follows

$$x' = f_d(f_e(x, \theta_e), \theta_d). \tag{2.14}$$

The parameters $\theta = (\theta_e, \theta_d)$ are then determined such that the sum of the losses $\sum_{i=1}^{n} L(x_i, x_i')$ is minimised over the data set. A common loss function is for example the euclidean distance, $L(x, x') = \|x - x'\|_2^2$. The assumption is that normal observations can easily be reproduced using the network so the reproduction loss $L(x, x')$ is low whereas outliers are harder to reproduce with the network which results in a higher loss $L(x, x')$.

The paper [71] introduces an interesting combination of an autoencoder and a Gaussian mixture model, where the authors first use an autoencoder network and then a GMM estimation network on the reduced dimensional representation of the autoencoder. These two networks are then optimised simultaneously. This method is called the DAGMM and has shown very good results but as with normal GMM networks, it requires the predetermination of the number of Gaussian models M which is not always trivial for unsupervised problems.

Isolation-based Methods

The last class of methods that we introduce are the so-called isolation-based methods with Isolation Forest being the most well known of such methods. Instead of relying on common distance or density measure, the isolation-based methods can be implemented by any means of separation between instances [41]. Isolation Forest constructs binary trees for separation purposes and uses the lengths of the paths in the tree as outlierness indicators. Since computations of distances and densities are not necessary in Isolation Forest, the time complexity is reduced substantially. Furthermore, Isolation Forest is easily scalable for large and high-dimensional data sets. The Isolation Forest is an explicit generalization method and we will discuss this method in detail later in the thesis.

Other Methods

Finally, we mention some well-known methods that are harder to categorise. First of all, the one-class support vector machine (OCSVM) [62] separates the outliers by constructing a boundary between normal and abnormal observations. Second of all, angle-based outlier detection measures outlierness by looking at in which angle the other observations are located. The assumption is then made that the outliers are located at the border of the data distribution meaning that the rest of the observations are located in a similar angle [34].

2.2.2. Method of Choice

In outlier detection, the choice of method is of fundamental importance to the performance of the outlier detection algorithm but unfortunately, this choice is not always obvious due to the unsupervised nature of the problem. In classification problems where labelled data is available, the problem of model selection is less challenging as measures of performance can be used in the decision-making process. The absence of labelled data makes the analysts understanding of relevant outliers a central aspect in the choice of method [2].

We start by identifying a few important characteristics that the outlier detection algorithm should preferably fulfil. Firstly, the performance and time complexity of the algorithm must be good and scale well with the size and dimensionality of the data set (preferably handle data sets with >100.000 instances and >30 dimensions). Secondly, because of the unsupervised nature of the data set, the number of parameters that need to be calibrated or tuned should be as low as possible.

In a recent survey [17], the performance of several state-of-the-art outlier detection methods was compared. We use the result of this comparison as a base for our selection of an outlier detection method. The study benchmarked the average precision, robustness, computation time and memory usage of 14 different outlier detection algorithms on both synthetic and real-world data sets. The study concludes that Isolation Forest is an excellent method to efficiently identify outliers while being both scalable in terms of memory and computation time for large data sets. Moreover, the survey indicates that Isolation Forest should be the method of choice when dealing with large data sets.

In the paper [41], the authors of Isolation Forest also assert that isolation based methods are more appropriate for the outlier detection task than the basic density and distance measures since they can detect both clustered and scattered anomalies instead of only the latter. The study [17] mentioned above, does not include the recently popular neural network based methods; Auto-encoders or RBMs. However, a recent paper [54] has extended the outlier benchmarking by comparing these two methods to Isolation Forest, concluding that overall, Isolation Forest outperforms these neural network based methods, especially when considering the short run time and easy-to-set hyperparameters. Based on the observations above, we decide to use Isolation Forest as the outlier detection method of choice in our research. Note that Isolation Forest is an algorithmic-driven outlier detection method and thus the resulting outlier score suffers from the absence of probabilistic interpretation. However, the tree-based properties of the method offer us the possibility of some interesting mathematical analysis as shown later on in this thesis. We have thus answered our first research sub-question (**SQ1**) about which outlier detection algorithm is best suited for our problem.

2.2.3. Outlier Detection in Anti-Money Laundering

Research into AML is done both in industry and academia, sometimes with cooperation between the two areas, as in our case, which means that a large part of the research is not publicly available. In this project, we base our reasoning both on expert opinion from individuals that work in the AML industry as well as the publicly available literature. The analytics software and solution company SAS is among the biggest players in the technological field of AML. Their recent white paper [61] identifies the main use cases of machine learning techniques in AML as listed in table 2.2.

Use case
Machine learning
1) as a supplement to transaction monitoring
2) for anomaly detection
3) for customer segmentation
4) for customer risk ranking
5) for social network analysis
6) for threshold setting and tuning

Table 2.2: The main use cases of machine learning in AML.

We are most interested in the outlier detection use case. We keep this categorization of use cases in mind when we explore which machine learning methods and models are currently being used in the AML literature that is publicly available.

Now specifically, we look for related work in outlier detection for AML purposes and at the same time, we try to identify possibilities of improvements. In the past two decades, there has been a growing interest in technological solutions in the fight against money laundering as shown in a recently published systematic literature review [37]. The systematic literature review identifies relevant papers by searching on four electronic databases, using search terms related to "money laundering" AND "technology" AND "approach". The 795 publications found are reduced even further by using a few inclusion and exclusion criteria, that agree with our research. This results in a total of 71 studies which are rigorously analysed and categorised based on the approach used, the application domain, the support mechanism, evidence level and context. The term technological solution covers a broader area of research than our machine learning aimed project. We thus use three of the categorization tables from the literature review to get specifically the papers that match our area of interest.

After limiting the scope from the original literature review [37] we end up with a total of 25 relevant papers. We categorise these papers based on their use case using the six categories from table 2.2. The results are shown in table 2.3 below. Since we are most interested in use cases for outlier detection as described earlier, we discuss the papers in this category in depth. Notice that there is considerable overlap between the categories of outlier detection and customer segmentation indicating that the combination of the two has been studied quite a bit. Note that such methods could also be classified as clustering-based outlier detection.

Category	Use case	Papers
1	Machine Learning as a supplement to transaction monitoring	None
2	Machine Learning for outlier detection	[25], [29], [64], [69], [58], [14], [9], [42], [67], [43]
3	Machine Learning for customer segmentation	[25], [69], [58], [14], [3], [10], [56]
4	Machine Learning for customer risk ranking	[33], [32]
5	Machine Learning for social network analysis	[29], [15], [18], [48], [66], [39], [63], [30], [38], [68], [23], [56]
6	Machine Learning for threshold setting and tuning	[3]

Table 2.3: The papers categorised based on their Machine Learning use case.

Papers [25], [14], [69], [58] use clustering-based outlier detection where the clusters are used to establish normal behaviour and then an outlier index is computed to find anomalous behaviour. In particular, paper [25] present a new clustering algorithm TEART and then measures the anomaly score based on deviations from the established transaction behaviour of the cluster the customer belongs to. The authors of [69] explore a new "cross outlier detection" model based on distance definitions that are incorporated with the financial transaction data features, paper [14] explores the use of Expectation Maximization for clustering and genetic search for features selection, paper [58] uses Fuzzy c-means for clustering and a Dynamic Bayesian Network for anomaly index calculation.

Some methods focus purely on anomaly detection. Paper [64] uses a one-class SVM outlier detection algorithm, paper [29] uses a combination of a graph-based similarity matrix and a feature matrix to detect anomalies, paper [9] uses three different types of anomaly detection algorithms (one of them being Isolation Forest) to obtain a suspicious ranking, paper [42] aims to identify suspicious sequences of transactions by studying their pattern and then predicting whether they are abnormal or not. Furthermore, the authors of [67] use DBSCAN clustering to find low-density regions which are marked as anomalous and the authors of [43] use Local Outlier Factor to detect and rank outliers.

Only one of these papers uses Isolation Forest to detect outliers, namely paper [9]. We also notice that none of the methods above tries to use explanation methods to explain the outliers that are identified. We see this as a point of improvement in the existing methods and start by introducing the literature of explanation methods in machine learning.

2.3. Explanation Methods

The outlier detection algorithms can be used to identify outliers but most of them do not describe *why* an observation was identified as an outlier. Since this information is highly relevant in many use cases, including ours, we inspect methods that can identify the main reasons behind the outlier classification. The area of Explainable or Interpretable Machine Learning has become increasingly popular in the last few years since many of the use cases of machine learning require the user to be able to interpret the results in some way.

In the book [51], the author presents a taxonomy of explanation methods by considering four main classification criteria:

- **Intrinsic or post hoc.** Intrinsic explainability refers to the machine learning itself being easily understood, like a decision tree or logistic regression, while post hoc explainability refers to the application of methods that analyse the model after training.

- **Result of the explanation method.** There are a lot of possible ways to present model explanations. For example, it can be in the form of feature importance measures, feature summary statistics or visualisations, internally learned weights or parameters or even an interpretable model trained to approximate the results of a black-box model.
- **Model-specific or model-agnostic.** A model-agnostic explanation method can be used on any machine learning model and is applied after training usually analysing feature input and output relations. Model-specific methods only work on a limited model class and are designed based on the internal mechanism of the specific model.
- **Local or global.** The explanation are global if they explain the entire model behaviour but local if they explain individual predictions.

In the task of finding suspicious customers, both local and global explanations are useful. The global explanation can help us evaluate in which specific subspaces the domain-specific anomalies are located and identify the features that are important for this outlier detection problem in general. On the other hand, the local explanation can provide alert experts with broader information about specific customers that showed indication of anomalous behaviour and have to be inspected further. Explainability in general is also important for the overall understanding of the model, both for the developer, users and regulators. In this thesis, we focus on local explainability.

Isolation Forest, the outlier detection method of choice, is not intrinsically explainable, mainly due to the number of trees it consists of, so post-hoc explanations are needed to explain the outliers. There only exists one model specific method for Isolation Forest that was presented in a recent paper [11]. The method is called DIFFI (Depth-based Isolation Forest Feature Importance) and it measures feature importance in Isolation Forest on a global level. The computational complexity of the DIFFI method is quite high, $O(\text{number of trees} \cdot \text{number of vertices} \cdot \text{number of observations} \cdot \text{max depth})$, especially for a large data set. As mentioned in the introduction, the authors of DIFFI introduced a new contribution [12] that also measures local importance and is referred to as the Local-DIFFI. This work was introduced towards the end of the thesis project timeline and we had already developed our own methodology for local explanations based on the original DIFFI method. We call our method Multiple Indicator Local-DIFFI (MI-Local-DIFFI) since it uses three indicators related to the tree structure of the Isolation Forest to measure the importance. Although the DIFFI method is the only explainability method specifically designed for Isolation Forest, there are several model-agnostic methods available and a few that work on the class of tree-based models.

We also look at model agnostic methods that could be used to explain our model. The SHAP method [44] is a model agnostic explanation method that has shown promising results but its computational complexity is the main concern. Fortunately, an alternative way of computing the SHAP values for tree-based models is introduced in the paper [45] and referred to as the TreeSHAP. These methods are based on the computation of Shapley values and solve an issue regarding the inconsistency of other popular explanation methods such as LIME [60]. A method is said to be inconsistent if a feature's importance is lowered when the true impact of that feature increases. The monotonicity property of the Shapley values ensures that the SHAP values fulfil the consistency property. The TreeSHAP method also demonstrates better agreement with human intuition compared to other popular methods such as LIME [60].

Originally, TreeSHAP was designed for supervised learning problems where labelled data is available. We modify the approach so that it fits our outlier detection setting by considering the outlier score from the outlier detection algorithm as the labels. This can be done even though the data is unlabelled because the explanations are computed after the Isolation Forest outlier scores have been computed. In Isolation Forest, high scores are interpreted as outliers so we want to identify the features that contributed to higher than average score for the outlying observations. That can give us a good explanation as to why a particular observation is identified as an outlier. The scores can also be used to give a global explanation about the importance of different features in the data set.

As a baseline to compare to MI-Local-DIFFI and TreeSHAP, we explore a very intuitive method of measuring feature importance which we refer to as the Alter-One-Feature (AOF) method. These three explanation methods, MI-Local-DIFFI, TreeSHAP and AOF will be described in more detail in section 3 and their performances will be compared in section 4.3.

3
Methods

In this section, we describe in detail the algorithms used in this thesis. First, we describe the Isolation Forest outlier detection algorithm and then the three explanation methods, MI-Local-DIFFI, TreeSHAP and AOF. There are a few variants of the Isolation Forest available such as the extended and the contextual weighted versions and later on, we also introduce our own variant, Ternary Isolation Forest. However, because of scalability and combination with explainability methods, we decide to use the original Isolation Forest in our experiments later on, both on the synthetic and the real-world data set, as the original version has been implemented very efficiently in the *sklearn* package, see runtime analysis in section 4.2.3. The *sklearn* implementation is also necessary for the calculation of the TreeSHAP values with the *shap* package. The complication of the TreeSHAP implementation is such that it did not fit the scope of this thesis to reimplement the Isolation Forest variants (extended, contextual and ternary) and then adjust the shap computations accordingly.

3.1. Isolation Forest

As mentioned earlier, the Isolation Forest algorithm showed very good performance in a quite recent literature review [17]. Isolation Forest can also be scaled up to handle large, high-dimensional data sets. In this section, we introduce the original Isolation Forest algorithm and discuss it in detail. We also introduce a few variants of Isolation Forest.

Isolation Forest [40] is an outlier detection method that uses the outcome of multiple isolation trees to compute an outlier score. An isolation tree is creating with random splits and can be thought of as the unsupervised and randomised version of the decision tree algorithm. Isolation Forest is an explicit generalization method and the training of the model and the scoring is done on the same data set, a so-called non-online setting. This means that it is not necessary to split the data into training and testing sets as done in supervised learning problems. The model can also be deployed in an online setting as done in the Local-DIFFI paper [12], where the model is applied to unseen observations that were not involved in the training phase. However, we focus on the non-online setting in this thesis.

We start by introducing a few basic definitions concerning trees, using notation from [19]. A *tree* is an undirected graph in which any two nodes are connected by exactly one path. We say that a tree is *rooted* when one node has been designated as the root. In a rooted tree, we say that the *parent* of a node v is the node that is connected to v on the path to the root and that the *child* of a node v is a node of which v is the parent. A node with no children is a *leaf*. A *binary tree* is a tree where each node has at most two children, referred to as the *left* and the *right* child. A *proper binary tree* is a tree in which every node has either 0 or 2 children. Using the definitions above we can define an isolation tree.

Definition 3.1.1. An *isolation tree* t is a proper binary tree where each non-leaf node contains a test that consists of an attribute F_i and a split value a such that the test $F_i < a$ determines the traversal of an observation to either the left or the right child nodes.

Let X be an $n \times d$ dimensional data set where the features F_j represent the column vectors of the data set for $j = 1, ..., d$. An observation x_i is the i-th row vector of the data set X. A sample from $X_s \subset X$ of size $\psi \leq n$ is used to build an isolation tree. The sample X_s is recursively split using tests of type $F_j < a$, where the feature $F_j \in \{F_1, ..., F_d\}$ is selected with equal probability and the split value is chosen uniformly in the range of that feature F_j. This splitting mechanism is continued until either (i) the node has only one instance or (ii) all the data in the node have the same values. If the data set X only contains distinct instances, each instance is isolated to a distinct leaf, creating an isolation tree with a total of $2\psi - 1$ nodes.

The isolation tree is used as a means to detect outliers. We define the path length of an observation as follows.

Definition 3.1.2. The *path length* $h(x)$ of an observation x is measured as the number of edges on the path of x from the root to its leaf node.

The path lengths are used to identify the susceptibility to isolation of different observations such that short path lengths mean high susceptibility to isolation and vice versa. Similarly to the relationship of decision trees and random forests, an isolation forest is a set of multiple isolation trees. We now present the Isolation Forest algorithm in detail and note that the algorithm can be divided into two phases, training and scoring phase.

3.1.1. Training Phase

In the training phase, the set of isolation trees that form the isolation forest is created using a subset of the data. This phase can be described by the two algorithms 1 and 2 below.

Algorithm 1 Create isolation forest

```
1: procedure $iForest(X, T, \psi)$
2:     Input: $X$ - input data , $T$ - number of trees, $\psi$ - subsampling size
3:     Initialize $Forest$
4:     Set height limit $\ell = ceiling(\log_2(\psi))$
5:     for $i = 1$ to $T$ do
6:         $X_s = sample(X, \psi)$
7:         $Forest = Forest \cup iTree(X_s, 0, \ell)$
8:     end for
9:     return $Forest$
10: end procedure
```

Algorithm 2 Create isolation tree

```
1:  procedure $iTree(X, e, \ell)$
2:      Input: $X$ - input data, $e$ - current tree height, $\ell$ - height limit
3:      Initialize $iTree$
4:      if $e \geq \ell$ or $|X| \leq 1$ then
5:          return $exNode\{Size = |X|\}$
6:      else
7:          Let $F$ be a list of features in $X$
8:          Randomly select a feature $F_i \in F$
9:          Select a split point $a$ uniformly over the max and min values of attribute $F_i$ in $X$.
10:         $X_l = filter(X, F_i < a)$
11:         $X_r = filter(X, F_i \geq a)$
12:         return $inNode\{Left = iTree(X_l, e+1, \ell),$
                          $Right = iTree(X_r, e+1, \ell),$
                          $SplitFeat = F_i,$
                          $SplitValue = a\}$
13:     end if
14: end procedure
```

Subsampling

One way to improve the computational efficiency is to use a subsample of size ψ to construct isolation trees in a training phase. The training phase returns the tree structure and the split conditions at each node, which are then used for out-of-sample scoring in the testing phase. This has been shown to result in better computational efficiency and better diversity.

Early Termination

A further efficiency gain is to create a threshold on the tree's height. Then, a node is labelled as external (leaf), if it has reached a certain distance from the root node. To estimate the path length of observations in such early terminated nodes, an additional length is assigned to them as they have not been grown to full height. We say that an isolation tree is fully grown if it is built without early termination. For a node that contains n observations, an additional $c(n)$ is added to its path length where $c(n)$ is the average depth of leaves in a fully grown isolation tree,

$$c(n) \approx \begin{cases} 0 & \text{if } n = 1, \\ 1 & \text{if } n = 2, \\ 2(\ln(n) + 0.5772 - 1) & \text{if } n > 2. \end{cases} \tag{3.1}$$

In the original paper, the expected path is $c(n) \approx 2(\ln(n-1) + 0.5772) - \frac{2(n-1)}{n}$ but this computation fails to take into consideration that isolation trees are proper binary trees where each non-leaf node has two children nodes but never one. In the section 3.1.3, we explain how the equation is derived.

Ultimately we end up with T randomly created proper binary trees where each node is either an external (leaf) node or an internal (non-leaf) node. The internal nodes contain information about the corresponding tests, that is the feature and the value which were used to split the data set into the left and right children of that internal node. The external nodes contain information about how many observations ended up in that node. A node becomes external (a leaf) if the tree has reached its maximum height in case of early termination, the node contains a single observation or the node contains multiple identical observations.

3.1.2. Scoring Phase

The outlier scoring of each observation is derived from its expected path length $\mathbb{E}[h(x)]$. The path length, $h(x_i)$ of an observation x_i in an isolation tree is the number of edges from the root node to the termination node e_i. In the case of early termination, if the number of observations in the termination node is $n_t > 1$, then $c(n_t)$ is added to the path length. Here, $c(n_t)$ represents the average path length of an isolation tree that is built with n_t observations as derived in section 3.1.3. The path length is thus,

$$h(x_i) = \begin{cases} e_i + c(n_t) & \text{if } n_t > 1, \\ e_i & \text{otherwise.} \end{cases} \tag{3.2}$$

The final score of observation x_i considering an ensemble of isolation trees is then given by

$$s(x_i) = 2^{-\frac{\mathbb{E}[h(x_i)]}{c(n)}}, \tag{3.3}$$

where n is the number of observations. Note that

$$\lim_{\mathbb{E}[h(x)]\to c(n)} s(x) = 2^{-\frac{c(n)}{c(n)}} = 0.5, \tag{3.4}$$

$$\lim_{\mathbb{E}[h(x)]\to 0} s(x) = 2^{-0} = 1, \tag{3.5}$$

$$\lim_{\mathbb{E}[h(x)]\to n-1} s(x) = 2^{-\frac{n-1}{c(n)}} = 2^{-\frac{1}{2}\cdot\frac{n-1}{\ln(n-1)+0.5772-1}} \to 0 \qquad \text{as } n \to \infty. \tag{3.6}$$

This means that $s \in (0, 1]$ and we can interpret the score in terms of outlierness of observations.

$$\text{If } s(x) \text{ is } \begin{cases} \text{close to 1, then } x \text{ is definitely an outlier.} \\ \text{much smaller than 0.5, then } x \text{ is quite certainly a normal observation.} \\ \text{around 0.5 for all observations, then the data set does not really contain clear outliers.} \end{cases}$$

In our applications of Isolation Forest, we have seen that there is no single correct outlier score threshold that can be used to distinguish between outliers and normal data points. The distribution of the outlier scores differs a lot between data sets.

As we have shown above, the observations in the data set are given an outlier score depending on how far away they are from the root node in each isolation tree. This can be explained visually by considering a 2-dimensional data set as shown in figure 3.1.

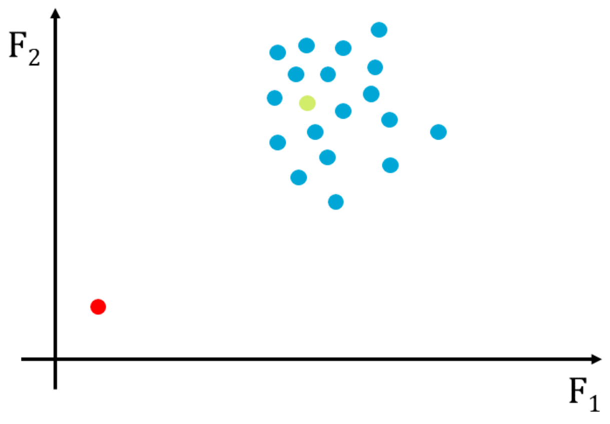
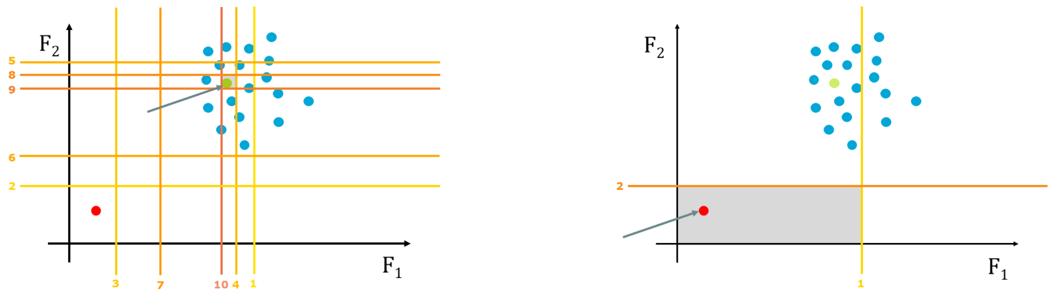


Figure 3.1: A 2-dimensional data set with 20 observations, 19 inliers (green) and one outlier (red).

For illustration purposes, we create an example of how an inlier and an outlier could be isolated in a single isolation tree without using subsampling or early termination. The outlier is coloured in red and the inlier we want to isolate is coloured bright green. When the isolation tree has been built with the random splitting as described above, we can analyse the path of (a) the inlier and (b) the outlier in the corresponding tree.

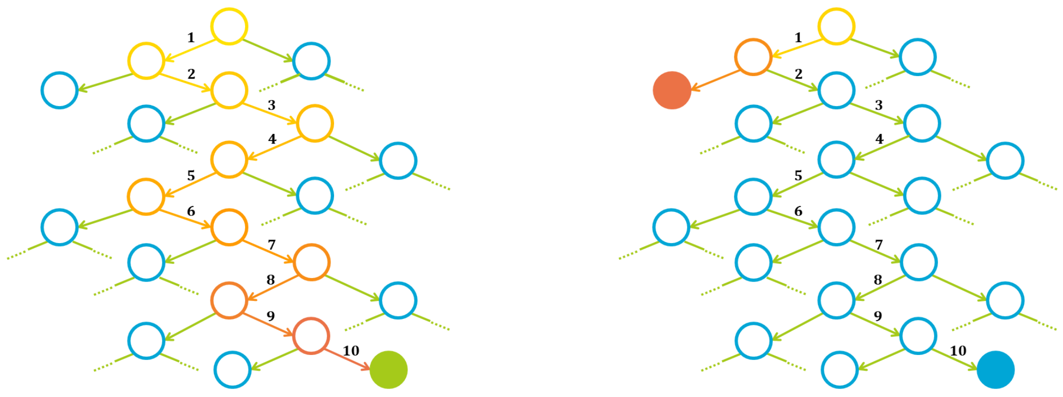


(a) Isolating an inlier. (b) Isolating an outlier.

Figure 3.2: An example of how inliers and outliers are randomly isolated in an isolation tree. The isolation of the inlier requires 10 splits whereas the outlier is isolated with only 2 splits.

In an experiment, there were 10 random splittings performed in the path of the inlier before it was isolated, but only 2 random splittings were needed before the outlier was isolated. In the isolation tree that corresponds to this random splitting (see figure 3.3), the path length of the inlier is 10 while the path length of the outlier is only 2. If we consider an isolation forest with only a single isolation tree, the corresponding scores for the inlier and the outlier would be

$$s(x_i) = e^{-10/c(20)} = 0.002 \text{ and } s(o) = e^{-2/c(20)} = 0.3. \tag{3.7}$$



(a) The path of the inlier. (b) The path of the outlier.

Figure 3.3: The path of the inlier and the outlier in a corresponding isolation tree.

3.1.3. Average Path Length of Leaves

As mentioned above, in case of early termination in which multiple observations end up in the same leaf, the average path length must be estimated based on the number of observations in the leaf. This is done by using similar analysis as performed on unsuccessful searches in binary search trees in the book by B. R. Preiss [55].

We define the *external path length* of a tree t to be the sum of the path lengths from the root to each external node/leaf in the tree. Using the concept of external path lengths, we can compute the average path length of leaf nodes by dividing the external path length with the number of leaves.

Theorem 3.1.1. *The average external path length of a fully grown isolation tree t with n input samples is denoted by $E(n)$ and given by the following equation,*

$$E(n) = \begin{cases} 0 & \textit{if } n = 1, \\ 2 & \textit{if } n = 2, \\ \frac{n}{n-1}E(n-1) + 2 & \textit{if } n > 2. \end{cases} \tag{3.8}$$

Proof. First assume that $n = 1$ and note that the fully grown isolation tree with one input sample is a root node and the average external path length of a single node is zero, $E(1) = 0$.

Now, let $t_n(l)$ be an arbitrary isolation tree where the left subtree has l leaf nodes for some $l \in \{1, 2, ..., n-1\}$. Such an isolation tree consists of a root, a left subtree with l leaves and a right subtree with $n - l$ leaves. The average external path length of such a tree can thus be computed as the sum of the average external path lengths of the left and the right subtree, $E(l)$ and $E(n - l)$ respectively, plus n since the nodes in the two subtrees are one level lower in $t_n(l)$.

We must average the external path lengths of the trees $t_n(l)$ over all possible sizes, $l \in \{1, 2, ..., n-1\}$ where we assume that the $n - 1$ different left vs. right subtree allocations are selected with equal probability. For $n > 1$ we thus get,

$$E(n) = \frac{1}{n-1}\sum_{l=1}^{n-1}(E(l) + E(n-l) + n) \tag{3.9}$$

$$= \frac{2}{n-1}\sum_{l=1}^{n-1}E(l) + n. \tag{3.10}$$

Using this equation we can compute that $E(2) = 2$. Furthermore, we get for $n > 1$ that

$$(n-1)E(n) = 2\sum_{l=1}^{n-1}E(l) + (n-1)\cdot n \tag{3.11}$$

and for $n > 2$,

$$(n-2)E(n-1) = 2\sum_{l=1}^{n-2}E(l) + (n-2)\cdot(n-1) \tag{3.12}$$

Subtracting equation 3.12 from 3.11 gives us for $n > 2$

$$(n-1)E(n) - (n-2)E(n-1) = 2E(n-1) + (n-1)\cdot n - (n-2)\cdot(n-1) \tag{3.13}$$

$$= 2E(n-1) + 2n - 2 \tag{3.14}$$

which can be written as

$$E(n) = \frac{nE(n-1) + 2n - 2}{n-1} \tag{3.15}$$

$$= \frac{n}{n-1}E(n-1) + 2 \tag{3.16}$$

yielding the desired result. □

We can further simplify the average internal path length formula to a non-recursional form. First divide equation 3.16 by n, and get

$$\frac{E(n)}{n} = \frac{E(n-1)}{n-1} + \frac{2}{n} \tag{3.17}$$

which holds for any $n > 2$. By substituting $n-1$ for n in the preceding sequence we can obtain the following sequence of equations

$$\frac{E(n)}{n} = \frac{E(n-1)}{n-1} + \frac{2}{n}, \quad \text{for } n > 2 \tag{3.18}$$
$$\frac{E(n-1)}{n-1} = \frac{E(n-2)}{n-2} + \frac{2}{n-1}, \quad \text{for } n > 3 \tag{3.19}$$
$$\frac{E(n-2)}{n-2} = \frac{E(n-3)}{n-3} + \frac{2}{n-2}, \quad \text{for } n > 4 \tag{3.20}$$
$$\vdots \tag{3.21}$$
$$\frac{E(n-k)}{n-k} = \frac{E(n-k-1)}{n-k-1} + \frac{2}{n-k}, \quad \text{for } n > k+2 \tag{3.22}$$
$$\vdots \tag{3.23}$$
$$\frac{E(3)}{3} = \frac{E(2)}{2} + \frac{2}{3}. \tag{3.24}$$

We can substitute the first term on the right hand side with the right hand side in the equation below until we have an equation containing only the base case,

$$\frac{E(n)}{n} = \frac{E(2)}{2} + \sum_{i=3}^{n} \frac{2}{i} \tag{3.25}$$
$$= \frac{2}{2} + 2\sum_{i=1}^{n} \frac{1}{i} - 3 \tag{3.26}$$
$$= 1 + 2H_n - 3 \tag{3.27}$$
$$= 2H_n - 2 \tag{3.28}$$
$$\tag{3.29}$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ represents the n-th harmonic number and it can be shown that $H_n \approx \ln(n) + \gamma$, where $\gamma \approx 0.5772$ is called Euler's constant. We thus have that the average path length of the leaves in a fully grown isolation tree, created with n observations is

$$c(n) = \frac{E(n)}{n} \approx \begin{cases} 0 & \text{if } n = 1, \\ 1 & \text{if } n = 2, \\ 2(\ln(n) + 0.5772 - 1) & \text{if } n > 2. \end{cases} \tag{3.30}$$

Note that this formula is slightly different from the formula of binary search trees that is used in the original Isolation Forest paper since they do not take into consideration that the isolation tree is a *proper* binary tree, where each node has either 0 or 2 children nodes.

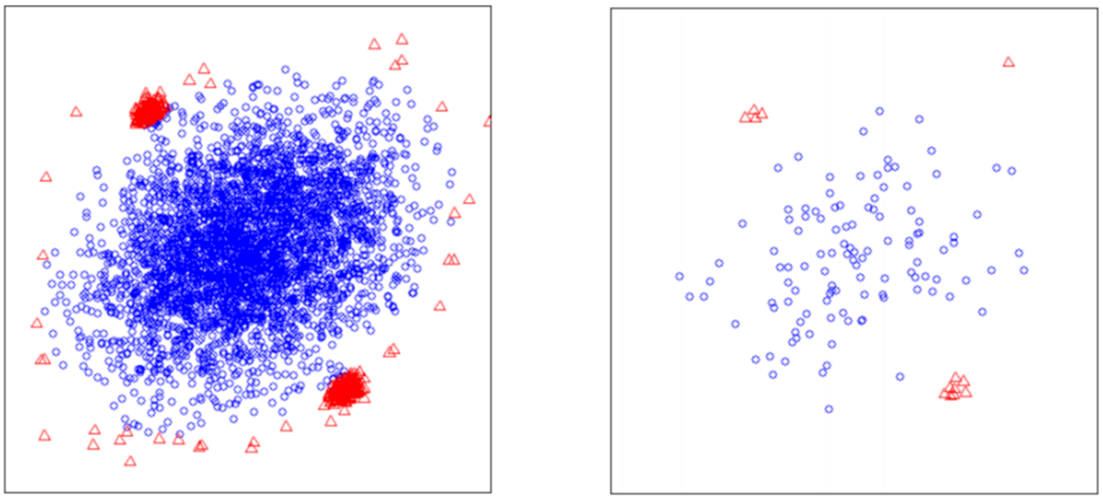
3.1.4. Complexity and Hyperparameters

The computation complexity of the Isolation Forest is $O(T\psi^2)$ for the training phase and $O(T\psi n)$ for the scoring phase where ψ is the subsampling size and T is the number of trees in the forest. In total the complexity is thus $O(T\psi(\psi + n))$. Using the *sklearn* implementation of Isolation Forest one can see that even though ψ is very large, the runtime does not explode, indicating that the runtime complexity is $k \cdot (T\psi(\psi + n))$ where k is a very low constant, see runtime analysis 4.2.3.

The two main configurable parameters of the Isolation Forest are ψ and T. The original paper [40] suggests that an optimal configuration of these parameters is generally $\psi = 256$ and $T = 100$ when considering both performance and runtime. However, since the implementation of Isolation Forest in the *sklearn* library [53] works very efficiently, the main focus should be on optimal performance instead of runtime. This holds especially for non-online applications, where it does not matter that much whether an algorithm takes 5 seconds or 25 seconds to run.

The authors of the Isolation Forest [41] explain that using small subsamples ($\psi = 256$) to create the isolation trees, builds better isolation models and reduces so-called swamping and masking problems. *Swamping* refers to situations where normal instances are wrongly identified as outliers and occur when normal instances become more scattered or their number increases. *Masking* refers to the situation where too many outliers exist, causing them to conceal their presence. This occurs when outlier clusters become increasingly denser and less rare.

To substantiate the claim that the subsampling reduces swamping and masking, an example is introduced where two dense anomaly clusters are placed close to a large cluster of normal observations, see figure 3.4. In this case, subsampling provides a data set with a clearer distinction between inliers and outliers which can increase Isolation Forests ability to detect outliers.

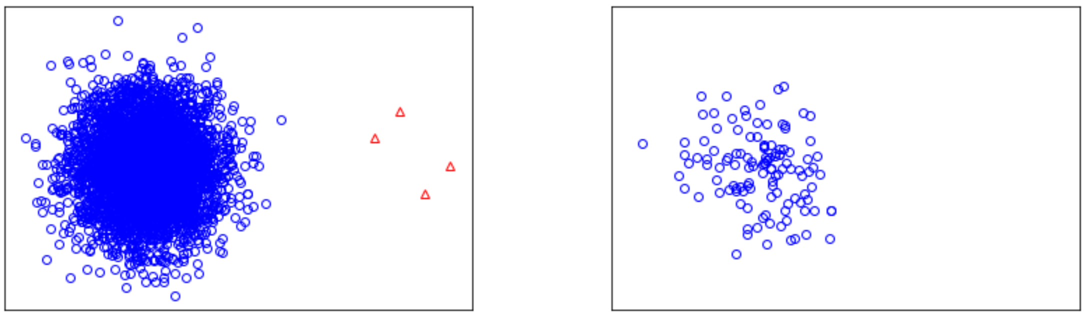


(a) Original sample
(4096 instances)

(b) Sub-sample
(128 instances)

Figure 3.4: An example where subsampling can reduce the problems of swamping and masking.

However, we can also think of cases where subsampling reduces the ability of the Isolation Forest to detect the outlier. Such a case may, for example, be encountered when the number of outliers is very small compared to the size of the data set and the outlier values are at the boundaries of the feature values. In this case, the subsample might not contain any of the outliers because of their scarcity and which causes the isolation trees to be created using only a part of the feature value range. In some cases, this might lead to the inability to isolate some very obvious outliers as shown in figure 3.5.



(a) Original sample
(4096 instances)

(b) Sub-sample
(128 instances)

Figure 3.5: An example where subsampling may lead to the inability of Isolation Forest to isolate obvious outliers.

When building an isolation tree using the subsample in figure 3.5 (b), the right-most split of the x-axis occurs such that the outliers belong to a set that also contains some inliers, see figure 3.6. The vertical line represents the point that is furthest to the right after subsampling. When creating the isolation forest, the split value for the x-axis will thus always be somewhere to the left of this line, meaning that it is impossible to isolate the outliers completely as there will always be inliers involved as well (at least the inliers that are to the right of the vertical line). Thus, the subsampling reduces the ability of the Isolation Forest to identify the four obvious outliers.

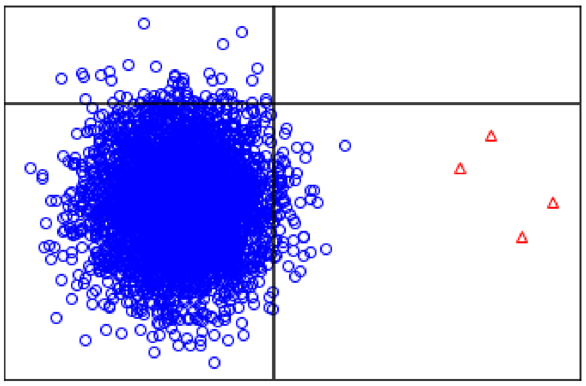


Figure 3.6: The maximum split value for the x-axis does not successfully split the outilers .

Therefore, in cases where the data set is very large and the outliers are few and located towards the boundaries of the feature value range, we suggest that the isolation trees are created without subsampling, that is $\psi = n$.

3.1.5. Add-Ons

Some add-ons of the original Isolation Forest have been suggested to increase the performance of the method. We discuss two of them, the extended version and the contextual weighted version. Since the Python implementation of these methods is not as computationally efficient as the *sklearn* Isolation Forest implementation and they are also incompatible with the *shap* package that is used for creating explanations, we choose the original Isolation Forest instead of the add-ons in our experiments later on. In future work, we suggest the use of these extensions for possible performance improvements.

Extended Isolation Forest

Extended Isolation Forest (EIF) [26] addresses some issues in the branching process of the standard Isolation Forest. The idea is to use hyperplanes with random slopes instead of axis parallel slopes for splitting the data when creating the isolation trees. The main change can be summarized in the *iTreeExtended* algorithm below.

Algorithm 3 Create extended isolation tree

```
1:  procedure $iTreeExtended(X, e, \ell)$
2:      Input: $X$ - input data, $e$ - current tree height, $\ell$ - height limit
3:      Initialize $iTree$
4:      if $e \geq \ell$ or $|X| \leq 1$ then
5:          return $exNode\{Size = |X|\}$
6:      else
7:          Randomly select a normal vector $\vec{n} \in \mathbb{R}^{|X|}$ by drawing each coordinate of $\vec{n}$ from a standard
    Normal distribution.
8:          Randomly select an intercept point $\vec{a} \in \mathbb{R}^{|X|}$ in the range of $X$.
9:          Set coordinates of $\vec{n}$ to zeros according to extension level.
10:         $X_l = filter(X, (X - \vec{a}) \cdot \vec{n} \leq 0)$
11:         $X_r = filter(X, (X - \vec{a}) \cdot \vec{n} > 0)$
12:         return $inNode\{Left = iTree(X_l, e + 1, \ell),$
                         $Right = iTree(X_r, e + 1, \ell),$
                         $Normal = \vec{n},$
                         $Intercept = \vec{a}\}$
13:     end if
14: end procedure
```

The same paper also introduced a slightly different approach called Rotated Isolation Forest where each subsample of the data that is used to create each of the trees is rotated in the plane by a random angle before the tree is built. This approach was however deemed less desirable by the authors mainly because of additional memory requirements, scaling difficulties and the fact that the rotation only averages out the rectangular bias problem instead of solving it.

The picture below gives a good summary of the rectangular bias problems of the standard Isolation Forest and how the extended version addresses these issues. The underlying observations form two Gaussian clusters in the upper left and lower right corner of the frame. The standard Isolation Forest creates two artificial clusters of lower anomaly scores in the upper right and lower left corners because of the axis parallel splitting mechanism while Extended Isolation Forest manages to circumvent this issue by splitting the data space using hyperplanes that are non-parallel to the axes.

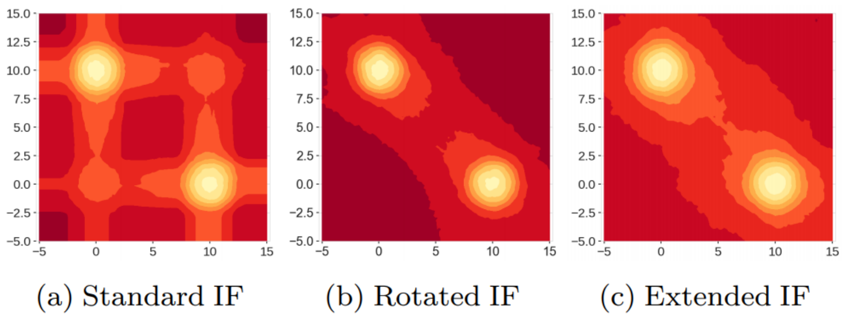


Figure 3.7: Heat map for the anomaly scores of the standard, rotated and extended version of Isolation Forest where the underlying observations form two Gaussian clusters.

The authors compare the performance of the standard Isolation Forest and Extended Isolation Forest on various real-world benchmark data sets and conclude that EIF performed consistently better in all cases considered. The EIF algorithm is available in Python (package: *EIF*) but even though the extended version is supposed to have the same computation complexity as the standard version, the runtime was significantly greater compared to the *sklearn* implementation of Isolation Forest [53]. This is most likely since the C++ integration of *EIF* is not working.

We also note that the Extended Isolation Forest algorithm does not select a single feature for splitting but rather uses a hyperplane. Therefore the splitting structure of the forest can not be used to determine the feature importance measures as done in the MI-Local-DIFFI method (discussed in section 3.2.1). Since the MI-Local-DIFFI feature importance measures can not be computed from Extended Isolation Forest and the *shap* package can not calculate the TreeSHAP values using the EIF structure, we do not use EIF in our experiments but note that further work can be done to allow the combination of *shap* and *EIF*.

Contextual Weighted Isolation Forest

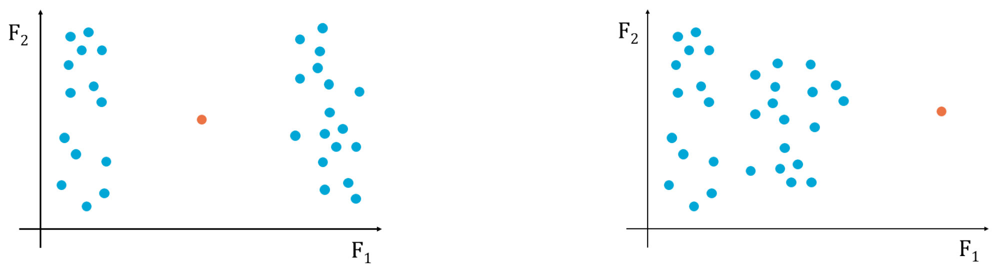
The contextual weighted version of Isolation Forest, introduced in paper [46] splits the features from the data set X into two groups, contextual and behavioural features, based on the properties of different features. Examples of behavioural features in a financial data set would be *the total amount of transactions* and *the number of cash deposits* while the contextual features would include for example *risk classification* and *customer type*. The contextual features are thus static and describe the customers' characteristics while the behavioural features may vary between different time points and describe the financial behaviour of customers. Only the behavioural features are then used to build the isolation trees and the leaves are considered to be certain subspaces of the feature space depending on the splits that lead to the leaves. The observations that end up in the same leaf-subspace but differ from the majority of observations in that leaf with regards to their contextual features, will be regarded more outlying than the remaining observations in that leaf. That way, the contextual information of the data is used to weigh the final outlier score.

In financial data, this version of Isolation Forest might be beneficial to locate outliers that are present in some contextual subspaces of the data. For example when certain transaction behaviour might be normal within one group of customers but abnormal within another group, using this group indication as a contextual feature might allow us to catch outliers that would not have been found with the standard Isolation Forest. We suggest that a contextual weighted isolation forest might improve the performance of the regular version but since the current implementation of that algorithm is not very efficient and does not scale well, we do not use the contextual weighted version in our experiments.

3.1.6. Ternary Isolation Forest

We suggest an alternative version of Isolation Forest using ternary trees as building blocks instead of the binary trees. In this section, we explain why the use of ternary trees could result in better detection ability and then define Ternary Isolation Forest in detail. Later on, in section 4.2.2, we compare the performance of Isolation Forest and Ternary Isolation Forest.

We say that an outlier is a *boundary outlier* if the value of the outlying feature is located in the extremes of the corresponding feature distribution, for example in the 5% lower or upper quantiles. We call an outlier an *interior outlier* if it is not a boundary outlier.



(a) Interior outlier. (b) Boundary outlier.

Figure 3.8: An interior and a boundary outlier.

The Isolation Forest algorithm detects outliers by splitting the subspace randomly. In each node of a tree, the data is split into two subsets and because of this, the boundary outliers have a higher probability to be isolated than the interior outliers. To visualize this, we look at figure 3.8, where we see that the interior outlier needs at least two splits to be isolated while the boundary outlier can be isolated in only a single split. Note that in some cases, especially where the features are many and the trees are short, each feature occurs only once in each path from the root to leaf. In that case, the interior outlier would be impossible to isolate. To correct this bias towards boundary outliers, we suggest that for each randomly drawn feature, there will be two randomly chosen split values, such that the data is split into three subsets instead of two. Then Isolation Forest will not consist of binary trees but ternary trees, as shown in figure 3.9. By performing this double split, the isolation of interior outliers becomes more probable than before.

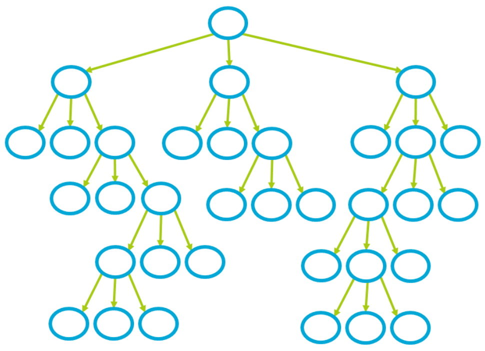


Figure 3.9: A proper ternary tree.

Ternary Trees

We begin by giving a definition of a (proper) ternary tree.

Definition 3.1.3. A *ternary tree t* is a tree where each vertex has at most three children, referred to as the *left, middle* and *right* children. A *semi-proper ternary tree* is a tree in which every node has either 0, 2 or 3 children and in the case of 2 children, only the middle child is missing.

and then a ternary isolation tree is defined as follows,

Definition 3.1.4. A *ternary isolation tree t* is a semi-proper ternary tree where each non-leaf vertex contains a test that consists of an attribute F_i and two split values a_1, a_2 such that the tests $F_i < a_1$, $a_1 \leq F_i < a_2$, $F_i \geq a_2$ determine the traversal of an observation to either the left, middle or right child vertices respectively.

The semi-proper property of the ternary isolation tree reflects the fact that there will always be observations in the left and right child nodes (minimum and maximum value) but there might not necessarily be an observation that has a value between a_1 and a_2, resulting in an empty middle child. The Ternary Isolation Forest algorithm can be described with the two algorithms 4 and 5 below and is a natural extension of the regular Isolation Forest algorithm (with binary trees).

Algorithm 4 Create Ternary Isolation Forest

```
1: procedure $iForest(X, T, \psi)$
2:     Input: $X$ - input data , $T$ - number of trees, $\psi$ - subsampling size
3:     Initialize $Forest$
4:     Set height limit $\ell = ceiling(\log_2(\psi))$
5:     for $i = 1$ to $T$ do
6:         $X_s = sample(X, \psi)$
7:         $Forest = Forest \cup iTree(X_s, 0, \ell)$
8:     end for
9:     return $Forest$
10: end procedure
```

Algorithm 5 Create Ternary Isolation Tree

```
1: procedure $iTree(X, e, \ell)$
2:     Input: $X$ - input data, $e$ - current tree height, $\ell$ - height limit
3:     Initialize $iTree$
4:     if $e \geq \ell$ or $|X| \leq 1$ then
5:         return $exNode\{Size = |X|\}$
6:     else
7:         Let $F$ be a list of features in $X$
8:         Randomly select an feature $F_i \in F$
9:         Select two split points $a_1, a_2$ uniformly over the max and min values of attribute $F_i$ in $X$.
10:        $X_l = filter(X, F_i < a_1)$
11:        $X_m = filter(X, a_1 \leq F_i < a_2)$
12:        $X_r = filter(X, F_i \geq a_2)$
13:        return $inNode\{Left = iTree(X_l, e + 1, \ell),$
                          $Middle = iTree(X_m, e + 1, \ell),$
                          $Right = iTree(X_r, e + 1, \ell),$
                          $SplitFeat = F_i,$
                          $SplitValues = (a_1, a_2)\}$
14:    end if
15: end procedure
```

Average Path Length of Leaves

In the scoring phase of Ternary Isolation Forest, the average path length that is added to the score of the observations in leaves that have not been fully grown has to take into account the structure of ternary trees instead of binary trees. We thus estimate the average path length of a ternary tree and put our results forward in the following theorem. The estimation follows a similar procedure as in the case of binary trees.

As before, we define the *external path length* of a tree t to be the sum of the path lengths from the root to each external vertex/leaf in the tree.

Theorem 3.1.2. *The average external path length of a fully grown ternary isolation tree t with n input samples is denoted by $E_t(n)$ and given by the following equation,*

$$E_t(n) = \begin{cases} 0 & \text{if } n = 1, \\ \frac{4}{n(n-1)} \sum_{l=1}^{n-1} \sum_{m=0}^{n-l} E_t(m) + n & \text{if } n > 1. \end{cases} \tag{3.31}$$

Proof. First assume that $n = 1$ and note that the fully grown ternary isolation tree with one input sample is a root vertice and the average external path length of a single vertex is zero, $E_t(1) = 0$.

Now, let $t_n(l, m)$ be an arbitrary ternary isolation tree where the left subtree has l leaf vertices for some $l \in \{1, 2, ..., n-1\}$ and the middle subtree has m leaf vertices for some $m \in \{0, 1, ..., n-l-1\}$. Such a ternary isolation tree consists of a root, a left subtree with l leaves, a middle subtree with m leaves and a right subtree with $n-l-m$ leaves. Note that by the definition of ternary isolation trees, the left and right subtree must contain at least one leaf vertex while the middle can contain down to zero leaves. The average external path length of such a tree can thus be computed as the sum of the average external path lengths of the left, the middle and the right subtree, $E_t(l)$, $E_t(m)$ and $E_t(n-l-m)$, respectively, plus n since the three subtrees are missing the connection with the root node in $t_n(l,m)$.

We must average the external path lengths of the trees $t_n(l,m)$ over all possible sizes, $l \in \{1,2,...,n-1\}$ and $m \in \{0,1,...,n-l-1\}$, where we assume that the

$$\sum_{l=1}^{n-1}\sum_{m=0}^{n-l-1} 1 = \sum_{l=1}^{n-1}(n-l) = n(n-1) - \sum_{l=1}^{n-1} l = n(n-1)/2$$

different left-middle-right subtree allocations are selected with equal probability. For $n > 1$ we thus get,

$$\begin{aligned}
E_t(n) &= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}\sum_{m=0}^{n-l-1}(E_t(l)+E_t(m)+E(n-l-m)+n)\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}\left((n-l)E_t(l)+\sum_{m=0}^{n-l-1}(E_t(m)+E(n-l-m))\right)+\frac{2}{n(n-1)}\sum_{l=1}^{n-1}\sum_{m=0}^{n-l-1} n\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}\left((n-l)E_t(l)+\sum_{m=0}^{n-l}(E_t(m)+E(n-l-m))-(E_t(n-l)+E_t(0))\right)+n\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}\left((n-l)E_t(l)-E_t(n-l)+\sum_{m=0}^{n-l}(E_t(m)+E(n-l-m))\right)+n\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}\left((n-l)E_t(l)-E_t(l)+2\sum_{m=0}^{n-l}E_t(m)\right)+n\\
&= \frac{2}{n(n-1)}\left(\sum_{l=1}^{n-1}(n-l-1)E_t(l)+2\sum_{l=1}^{n-1}\sum_{m=0}^{n-l}E_t(m)\right)+n\\
&= \frac{2}{n(n-1)}\left(\sum_{l=1}^{n-1}(n-l-1)E_t(l)+2\sum_{l=1}^{n-1}\sum_{i=1}^{n-l}E_t(l)\right)+n\\
&= \frac{2}{n(n-1)}\left(\sum_{l=1}^{n-1}(n-l-1)E_t(l)+2\sum_{l=1}^{n-1}(n-l)E_t(l)\right)+n\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}((n-l-1)E_t(l)+2(n-l)E_t(l))+n\\
&= \frac{2}{n(n-1)}\sum_{l=1}^{n-1}(3(n-l)-1)E_t(l)+n.
\end{aligned}$$

□

We further simplify this equation such that it depends on only the three preceding path lengths instead of all the preceding path lengths. Unfortunately, we were unable to find an equation for $E_t(n)$ that did not include any preceding terms $E_t(n-m)$ where $m > 0$. However, it speeds up computation to use only a proportion of the preceding terms. We use $A(n)$ to represent the size $A(n) = n(n-1)E_t(n) - (n-1)(n-2)E_t(n-1)$ and compute for $n > 2$,

$$\begin{aligned}
A(n) &= n(n-1)E_t(n) - (n-1)(n-2)E_t(n-1) \\
&= \left(2\sum_{l=1}^{n-1}(3(n-l)-1)E_t(l) + n^2(n-1)\right) \\
&\quad - \left(2\sum_{l=1}^{n-2}(3(n-1-l)-1)E_t(l) + (n-1)^2(n-2)\right) \\
&= 4E_t(n-1) + 2\sum_{l=1}^{n-2} E_t(l)\,(3(n-l)-1) - ((3(n-1-l)-1)) \\
&\quad + n^2(n-1) - (n-1)^2(n-2) \\
&= 4E_t(n-1) + 6\sum_{l=1}^{n-2} E_t(l) + (n-1)(3n-2) \\
&= 4E_t(n-1) + 6\sum_{l=1}^{n-2} E_t(l) + (n-1)(3n-2).
\end{aligned}$$

So for $n > 4$ we get

$$\begin{aligned}
A(n) - A(n-2) = n(n-1)E_t(n) &- (n-1)(n-2)E_t(n-1) \\
&- \left[(n-2)(n-3)E_t(n-2) - (n-3)(n-4)E_t(n-3)\right],
\end{aligned}$$

and

$$\begin{aligned}
A(n) - A(n-2) &= 4E_t(n-1) + 6\sum_{l=1}^{n-2} E_t(l) + (n-1)(3n-2) \\
&\quad - \left[2E_t(n-3) + 6\sum_{l=1}^{n-4} E_t(l) + (n-3)(3n-8)\right] \\
&= 4E_t(n-1) + 6E_t(n-2) + 6E_t(n-3) - \\
&\quad 2E_t(n-3) + (n-1)(3n-2) - (n-3)(3n-8) \\
&= 4E_t(n-1) + 6E_t(n-2) + 4E_t(n-3) + (n-1)(3n-2) - (n-3)(3n-8).
\end{aligned}$$

Comparing the equations above gives us for $n > 4$

$$\begin{aligned}
n(n-1)E_t(n) = E_t(n-1)\,&[(n-1)(n-2)+4] \\
+ E_t(n-2)\,&[(n-2)(n-3)+6] \\
+ E_t(n-3)\,&[-(n-3)(n-4)+4] \\
+ [(n-1)(3n-2)&-(n-3)(3n-8)]\,.
\end{aligned}$$

We can thus write

$$E_t(n) = C_1(n)\cdot E_t(n-1) + C_2(n)\cdot E_t(n-2) + C_3(n)\cdot E_t(n-3) + C_4(n), \tag{3.32}$$

where the coefficients $C_i(n)$ for $i = 1, 2, 3, 4$ are obtained with the equations below

$$C_1(n) = \frac{(n-1)(n-2)+4}{n(n-1)}, \tag{3.33}$$

$$C_2(n) = \frac{(n-2)(n-3)+6}{n(n-1)}, \tag{3.34}$$

$$C_3(n) = \frac{-(n-3)(n-4)+4}{n(n-1)}, \tag{3.35}$$

$$C_4(n) = \frac{(n-1)(3n-2)-(n-3)(3n-8)}{n(n-1)}. \tag{3.36}$$

Finally, using $E_t(n)$ we can calculate the average depths of the leaf vertices as

$$\text{Average depth of leaves } = c_t(n) = \frac{E_t(n)}{n}. \tag{3.37}$$

The scoring phase of Ternary Isolation Forest is thus the same as in the binary version except we use c_t to estimate the average leaf depth instead of c. In figure 3.10, we see how the average path length of leaf nodes increases in the number of observations n.

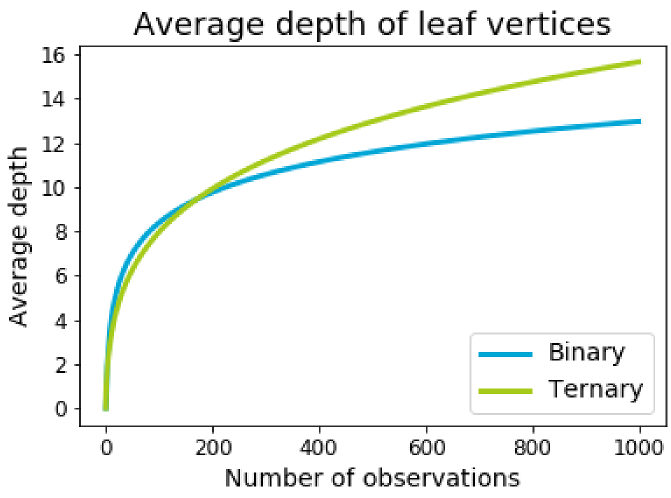


Figure 3.10: The average path length of leaf nodes in the regular binary Isolation Forest and the ternary version for a different number of input samples.

For $n \approx 170$ and less, the binary version of Isolation Forest has a larger average path length but for n over 170, the ternary version surpasses the binary version with regards to average path length. This can be explained with two reasons, (1) the binary version needs on average more nodes until the data samples are isolated resulting longer trees but (2) the ternary version has more nodes on the maximum depth level. For n less than 170, the first reason outweighs the second reason, and vice versa for n greater than 170.

3.2. Explanation Methods

We introduce three different explanation methods that can be used to explain individual outliers predicted by Isolation Forest. The first method is our own adaptation of the DIFFI method [11] that delivers feature importance explanations on a local level instead of globally. This method is referred to as the MI-Local-DIFFI method. We also introduce the TreeSHAP method that is originally designed for tree-based regression models but we use it for outlier explanations. Finally, we introduce a baseline method, Alter-One-Feature (AOF), that identifies the feature importance of each feature by measuring the change in outlier score when that same feature is altered.

The MI-Local-DIFFI calculates the explanations by looking at the splitting structure of the isolation forest, analysing the splitting points on the outlier's path while the other two methods, TreeSHAP and AOF, analyse the change in outlier score when various features are removed, added or altered. The TreeSHAP takes into account the relationship between different features while the AOF only considers one feature at a time.

3.2.1. MI-Local-DIFFI

The DIFFI method [11] is a global feature importance method for the Isolation Forest method that looks at the splitting structure of the isolation trees. We suggest an adaptation of this method that delivers local explanations for individual outliers. The main assumption of this method is that the information in the splitting points of an outlier path can be used to determine the importance of the different features in the outlier identification process. As mentioned before, the authors of DIFFI released a local adaptation of their method, Local-DIFFI, on the arXiv in late July 2020, near the deadline of this thesis project [12] when we had already developed our multiple indicator version. Thus, we first describe our Multiple Indicator Local-DIFFI (MI-Local-DIFFI) method and then compare it to Local-DIFFI from [12].

In an isolation tree, each observation gets a specific path depending on the structure of the tree and the values of the observation. To understand the reason why an instance is detected as an outlier, we look at how an outlier traverses down each tree in the forest, observing:

(a) **Path length indicator:** the length of the outlier's path in each tree,

(b) **Split proportion indicator:** looking at each split, we observe the proportion of observations that follow the same path as the outlier vs. the observations that end up in the opposite node. Depending on this proportion we allocate certain importance to the corresponding splitting feature.

(c) **Split interval length indicator:** the length of the subinterval containing the outlier o after a split is performed in proportion to the length of the interval before the split.

The DIFFI method for global explanations uses both a path length indicator and a split proportion indicator but instead of looking at individual outlier paths, it looks at the path of inliers and outliers separately. As the outliers traverse down each isolation tree, a quantity based on path length and the split proportion of the set of outliers is added to the importance of the corresponding split feature. The same is done for the inliers. The global importance score is then obtained by dividing the importance weights of the outliers with the importance weights of the inliers. A more exact description of this method can be found in the original paper [11].

The name of our method, Multiple Indicator Local-DIFFI (MI-Local-DIFFI) refers to the fact that the multiple indicators above are used to calculate the explanations. To incorporate (a), we adjust the feature importance score by observing the path length such that shorter paths get more weight. For the second point (b), we observe which features are used for splitting and adjust the corresponding feature importance weight with the proportion of observations that follow the same path as the outlier. The fewer the observations following the outlier to the same child node, the higher the feature importance score for that particular feature. Finally, for (c) we compare the length of the interval after the feature split (the part that contains the outlier) to the length of the interval before the split. This is because a small split proportion is expected if the split interval is also small. We thus increase the importance of features that have a small split proportion but a relatively large split interval.

We also considered incorporating the depth of each specific split into the feature importance score, but as shown in the analysis chapter, this information did not improve the performance of the explanation methods. In this case, the depth of a specific split refers to the path length of the corresponding splitting node to the root. Although at first, it may seem to be informative to know whether a good split occurs at a low or high depth, this is outweighed by the fact that the features are chosen randomly and may even appear only once in each path. Adjusting the feature importance scores with the location of the splits does thus not add value and may even wrongly reduce the importance of truly important features.

Below we discuss how each of the indicators is incorporated into the feature importance measures.

(a) **Path length indicator.**
Let $PL(o, i)$ be the path length of outlier's o leaf node in isolation tree t_i. We create a standardized measure of path lengths influence on the feature importance, by first defining the lower and upper path lengths as

$$\begin{aligned} PL_{\text{lower}} &= 1 \\ PL_{\text{upper}} &= \lceil 2(\log(\psi) + \gamma - 1) \rceil \end{aligned}$$

where the lower estimate corresponds to the lowest possible path length of an outlier and we define the upper estimate as the average path length of the leaves in an isolation tree, since outliers are identified by having shorter than average path lengths. Then the path length weight is computed as

$$w^{PL}(o, i) = \max\left\{0.1, \min\left[1, 1 - \left(1 - \frac{PL(o, i) - PL_{\text{lower}}}{PL_{\text{upper}} - PL_{\text{lower}}}\right)\right]\right\} \tag{3.38}$$

This path length weighing scheme thus reflects the fact that features that are occurring in a long outlier path are less important than those that occur in shorter paths. Even though a feature occurs in a longer path and provides a very good split we would still like to increase its importance. Therefore, the lowest possible weight is set at 0.1 instead of 0. Note that $w^{PL}(o, i) \in [0.1, 1]$.

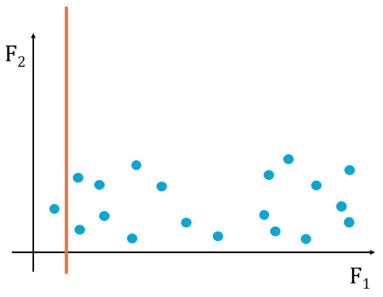
(b) **Split proportion indicator.**
Now let's consider the split proportion weight. Let v_{ij} be a node in the isolation tree t_i and let F_{ij} be the feature that is used to split the data in node v_{ij}. Let q_{ij} be the number of observations in node v_{ij} and let q^o_{ij} be the number of observations in the child node of v_{ij} that contains the outlier o. Let $\vec{w}^{SP}(o, i) = (w^{SP}_1(o, i), ..., w^{SP}_{PL}(o, i))$ where $PL = PL(o, i)$ and

$$w^{SP}_j(o, i) = \begin{cases} 0 & \text{if } q_{ij} = 2 \\ 1 - \frac{q^o_{ij} - 1}{q_{ij} - 2} & \text{otherwise} \end{cases} \tag{3.39}$$

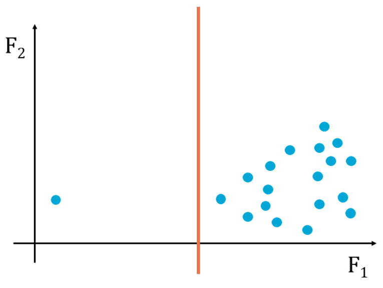
By using this weighing scheme, features that were seriously involved in the isolation of the outlier, get a higher weight than those that did not have much impact in isolating the outlier. We say that a feature split becomes better as the split proportion w_{SP} becomes higher. A good split thus has a higher than average w_{SP} weight while a bad split has a lower than average w_{SP} weight.

If a feature occurs more than once in a path of an outlier, then only the best split (highest w^{SP}) is used for feature importance calculations. This is because a very important feature can result in a bad split because of the random choice of the splitting value. We are thus more interested in increasing the importance of features that result in good splits rather than punishing the features with negative weights when the split is not so good. We did some experiments where an average of the split proportion was considered rather than the best one but that resulted in worse explanations.

(c) **Split interval length indicator.**
We also try to incorporate information about where the split value is chosen within the range of the uniform distribution. We rely on the assumption that, naturally, a low split proportion occurs if the split value is chosen near the minimum or maximum of the feature values since the number of observations in an interval should decrease as the interval decreases. Additional importance weight should be added if the split proportion is low and the split value is chosen such that it is relatively far away from the endpoints of the values range. We visualize two scenarios in figure 3.11 below.



(a) Split value chosen close to an end point.



(b) Split value chosen away from the end points.

Figure 3.11: Looking at the figure, the leftmost observation in plot (b) is more of an outlier than the leftmost observation in plot (a). We use this property in our MI-Local-DIFFI method. If a single observation is isolated with a split very close to the boundary (a), this feature shold get less importance weight than if a single observation is isolated with a split in the middle of the feature values range (b).

In the figure above, the split proportion is the same (1 vs. 18) in both scenarios (a) and (b) but we would like to give feature F_1 more importance in scenario (b) than (a) because the F_1 is more important in terms outlierness of the leftmost node in scenario (b). This increased weight can be incorporated by observing the length of the split intervals as described below.

First, we define a few notions related to split intervals in isolation trees. Each non-leaf node v in an isolation tree has a corresponding sample of observations X_s, a split feature F and a split value s. We define the *feature interval* of a node to be the interval $[a, b]$ where a,b are the minimum and maximum values of the split feature F, looking at the sample of observations X_s in that node v. Furthermore, we define the *outlier split interval* of the node v with respect to outlier o to be the interval $[a, s[$ if the outlier o traverses to the left child node but $[s, b]$ if the outlier o traverses to the right child node.

For a node v_{ij} that is in a path of an outlier o in the tree t_i, we observe

$$si(o, v_{ij}) = \frac{|\text{outlier split interval of } v_{ij} \text{ with respect to } o|}{|\text{feature interval of } v_{ij}|}. \tag{3.40}$$

For the node v_{ij} in the path of an outlier o in tree t_i we measure the split interval weight as $\vec{w}^{SI}(o, i) = [w_1^{SI}(o, i), w_2^{SI}(o, i), ..., w_{PL}^{SI}(o, i)]$ where $PL = PathLength(o, i)$ and

$$w_j^{SI}(o, i) = 1.5 - \frac{1}{si(o, v_{ij}) + 1}. \tag{3.41}$$

Note that $w_j^{SI}(o, i) \in [0.5, 1]$ and for an outlier split interval that is large (for example 0.9) the weight $w_j^{SI}(o, i)$ is also large (≈ 0.97) and for an outlier split interval that is small (for example 0.1), the weight $w_j^{SI}(o, i)$, is also small (≈ 0.59). In the visual scenario in figure 3.11 we see that the proportion is around 0.1 in scenario (a) but around 0.5 in scenario (b). The split interval weight will thus give a weight of ≈ 0.59 to the feature in scenario (a) whilst the feature in scenario

(b) will get a higher weight of ≈ 0.83.

Let $Path(o, i)$ denote the path in tree t_i that outlier o traverses down. The MI-Local-DIFFI algorithm can thus be presented as follows.

Algorithm 6 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 the isolation forest. We want
      to compute the feature importance for each $G_k$, $k = 1, ..., n_F$.
    Initialise feature importance vector $FI = \vec{0}$ with length equal
      to number of features $n_F$.
    Initialise feature occurence vector occurrence($F$) $= \vec{0}$ 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 3.38.
        Calculate $\vec{w}^{SP}(o, i)$ using equation 3.39.
        Calculate $\vec{w}^{SI}(o, i)$ using equation 3.41.
        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 $Path(o, i)$.
                occurrence($k$) = occurrence($k$) + 1
                $\sigma(k) = w^{PL}(o, i) \cdot w^{SP}_{j_k}(o, i) \cdot w^{SI}_{j_k}(o, i)$
                $FI(k) = FI(k) + \sigma(k)$
            end if
        end for
    end for
    $FI = FI/$occurrence
    return $FI$
end procedure
```

Consider an outlier o in the data set X that was detected with the isolation forest IF $= \{t_1, ..., t_T\}$ where t_i are the isolation trees. As before, $Path(o, i)$ and $PL(o, i)$ represent the path of the outlier o in the isolation tree t_i and its length, respectively. To calculate the MI-Local-DIFFI feature importance score we look at each tree t_i and zoom in on the path of the outlier, $Path(o, i)$. In each node on the path, we observe which features were used for splitting. We give each of these features a weight based on the proportion of observations that go to the same children node as the outlier vs. the opposite node calculated with equation 3.39. If the same feature occurs more than once in the same path, we only consider the feature that has the highest w^{SP} ratio. Furthermore, the importance of the features is weighted with the length of the outlier path in t_i (eq. 3.38) and length the outlier split interval in proportion to the feature interval (eq. 3.41).

In the end, the feature importance measures are weighted by how many times a feature has occurred in a path of the outlier in all the trees. Note that $w^{PL} \in [0.1, 1]$, $w^{SP} \in [0, 1]$ and $w^{SI} \in [0.5, 1]$ so the feature importance scores are in the interval $[0, 1]$ where a higher score represents more important features according to the weighing scheme. The MI-Local-DIFFI algorithm thus returns a feature importance measure for each outlier where the feature with the highest value can be interpreted as the feature contributing most to the outlier being classified as such. The complexity of calculating the feature importance for all the outliers is $O(T \cdot n_o \cdot \text{average outlier leaf depth})$, since, for each outlier, we measure the splits in all the nodes that are present in the path of the outlier, for each of the isolation trees. By definition, the average outlier leaf depth is often shorter than in the case of inliers but we can use the maximum height as a worst-case estimate, $\log_2 \psi$.

Local-DIFFI

In this section, we discuss Local-DIFFI, which is very recent work from the authors of the DIFFI method [12]. The paper is available on the arXiv and has been submitted to the IEEE for possible publication. The authors also include a link to their github that contains all relevant code.

We start by describing Local-DIFFI. Let V_i be the set of nodes in an isolation tree t_i and $Path(o,i)$ the path in tree t_i that the outlier o traverses down. Let I_O be d-dimensional vectors, where the j-th component represents the feature importance of the j-th feature. First, the vector I_O is initialised, $I_O = \mathbf{0}_d$ and then I_O is updated in an additive fashion. For each predicted outlier o, an iteration over the internal nodes in the path $Path(o,i)$ is performed and if the splitting feature associated with the internal node is F_j, then the j-th component of I_O is updated by adding the quantity

$$\Delta = \frac{1}{e_o^i} - \frac{1}{h_{\max}} \tag{3.42}$$

where e_o^i represents the number of edges on the path of outlier o from the root node to the termination node in isolation tree t_i and $h_{\max} = \lceil \log_2(\psi) \rceil$ is the maximum path length before early-termination. This iteration is done for each isolation tree t_i in the isolation forest. The Local-DIFFI scores are then given by

$$\frac{I_O}{C_O} \tag{3.43}$$

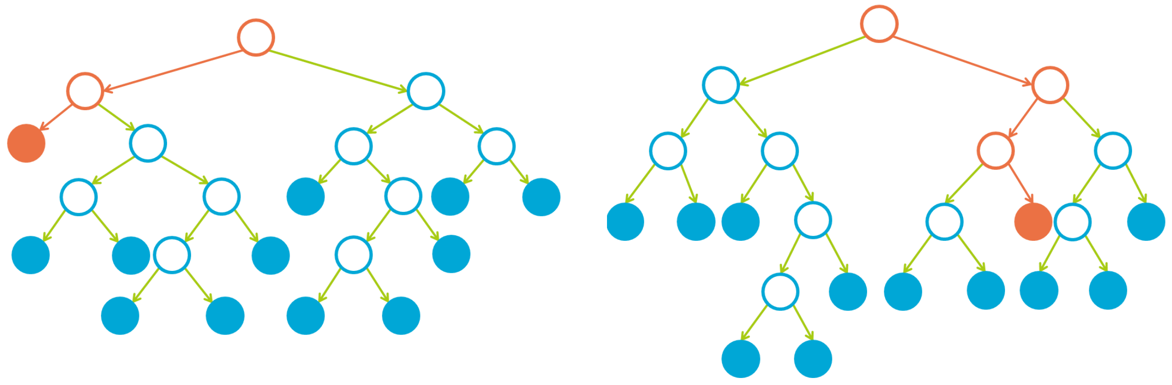
where the features counters C_O are d-dimensional vectors where the j-th component represents how many times the j-th feature appeared while updating the I_O vector.

The main technical differences between our MI-Local-DIFFI and this method is that the Local-DIFFI only uses the path length as an importance indicator while we also incorporate the split proportion and the split interval length to calculate the importance. The split proportion indicator requires that the traversal of other observations is also taken into account, which is not done in Local-DIFFI. Furthermore, they use e_o^t for their path length indication instead of the full path length adjusted with the number of samples in leaf $c(n_{t_i})$.

We discuss a few more differences in the set up of the two explanation methods and compare them further in section 4.4.

MI-Local-DIFFI Example

Suppose we have an isolation forest consisting of two isolation trees, t_1 and t_2. The isolation forest is created using a 3-dimensional data set X. The observation $o \in X$ gets the lowest outlier score and is thus predicted as an outlier. We want to identify the reason why this observation o was predicted as an outlier. The two isolation trees and the path of the outlier o are shown in figure 3.12.

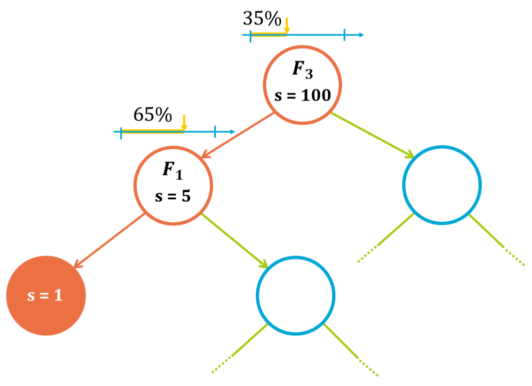


(a) The first isolation tree, t_1 and the path that the outlier o traverses down.

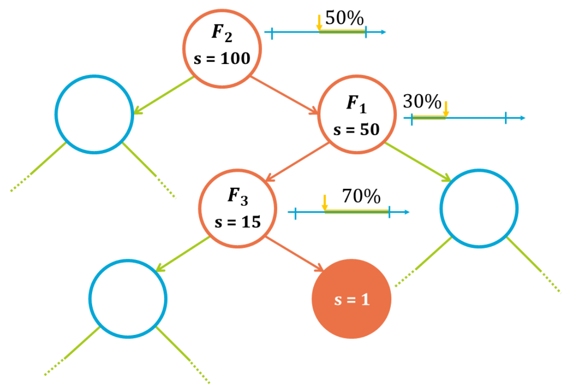
(b) The second isolation tree, t_2 and the path that the outlier o traverses down.

Figure 3.12: The full isolation forest consisting of two isolation trees.

To understand why o was predicted as an outlier, we zoom in on the path of the outlier and identify which features were used to split, the number of samples in each node and the length of the path. Below we see that only feature 1 and 3 occurred in the path of outlier o in tree t_1 while all three features occurred in tree t_2.



(a) The first isolation tree, t_1 and the path that the outlier o traverses down.



(b) The second isolation tree, t_2 and the path that the outlier o traverses down.

Figure 3.13: The path of the outlier o in the two isolation trees. The size s represents the number of observations (sample size) in each node.

In t_1, feature F_3 is used for the initial split where only 5 of the original 100 observations follow the same path as o. In the second split on the outliers path, feature F_1 is used for splitting and manages to isolate the outlier with a 4-1 split. Based on these two splits, we observe that the path length of this outlier is 2 and the split proportion is **5**-95 for feature F_3 and **1**-4 for feature F_1. The split interval length proportions are 35% for feature F_3 and 65% for feature F_1 meaning that feature F_1 will receive a higher weight. Using this information, we can calculate the weights corresponding to tree t_1, w_{PL}, w_{SP} and w_{SI} according to equations 3.38, 3.39 and 3.41.

In t_2 feature F_2 is used for the initial split resulting in a 50-**50** split of the original 100 observations. Next, feature F_1 splits the 50 data samples such that **15** follow the outliers path while 35 samples go to the opposite children node. Finally, feature F_3 manages to isolate the outlier with a 14-**1** split. The split interval length weights are also incorporated and in this case feature F_3 gets the highest weight. Using this information, we can calculate the weights corresponding to tree t_2, w_{PL}, w_{SP} and w_{SI} according to equations 3.38, 3.39 and 3.41.

In both trees, feature F_3 gets the highest importance since it gets the highest w_{SP} weight and is therefore considered as the biggest reason for o being an outlier. Note also that in tree t_2, the outlier split interval is really large (70%) and still this split manages to isolate the outlier. Feature F_1 also gets some importance weights while feature F_2 gets the lowest score since it is not on the path of the outlier in tree t_1 and only gives a 50-50 split in tree t_2.

Standardization

The explanations are local and thus concern individual predictions. Looking at the feature importance metrics for individual outliers, it is possible to standardize the importance scores such that the most important feature receives a score of 1 and the least important a score of 0. Considering some outlier o, one can use the following min-max standardization equation,

$$\text{standardized } FI(o) = \frac{FI(o) - \min FI(o)}{\max FI(o) - \min FI(o)}. \tag{3.44}$$

Subsampling

There is a possibility to use only a subsample of the data set to calculate the split proportion indicator. This is of course not as accurate but can decrease the runtime of the method, especially in the case of larger data sets.

3.2.2. TreeSHAP

The TreeSHAP method [45] is based on so-called Shapley values from coalitional game theory. The game-theoretical notations can be extended to predictions of a machine learning model where the *game* in this setting represents the prediction task for a single observation, the *payoff* is the difference between the prediction of this observation and the average prediction of the whole data set and the *players* are the features. In the SHAP (SHapley Additive exPlanations) [44] method, a conditional expectation based estimation of the Shapley values is used to measure the contribution of each of the features to the prediction by calculating the change in prediction when each feature enters different feature coalitions. A *coalition* is just a subset of the set of players so a feature coalition is a subset of the set of features.

First, we explain how the SHAP values of an observation x and prediction model f is computed. Let F be the set of features in data set X. Ultimately, the SHAP value of a feature F_j is just the weighted sum of the change in the prediction that occurs when the feature F_j is added to all feature coalitions S that do not contain F_j. The SHAP value $\bar{\phi}_j(f_x)$ of a feature F_j for an observation x in the prediction model f is computed as

$$\phi_j(f,x) = \sum_{S \subset \{F_1,\ldots,F_n\}\;\{F_j\}} \frac{|S|!(n-|S|-1)!}{n!}(\mathbb{E}[f(x)|x_{S\cup\{F_j\}}] - \mathbb{E}[f(x)|x_S]) \tag{3.45}$$

where $\mathbb{E}[f(x)|x_S]$ is the conditional expectation of the prediction $f(x)$ where only the features in S are used to calculate the prediction. According to the paper [44], they can be used to explain the prediction $f(x)$ in an additive way

$$f(x) = \mathbb{E}[f(X)] + \phi_1(f,x) + \ldots + \phi_d(f,x) \tag{3.46}$$

where d is the number of features.

For a tree-based model f, the conditional expectation $\mathbb{E}[f(x)|x_S]$ can be estimated by only taking into account the splits that contain the features in S and averaging out the splits that contain other features, as shown in the example below. This is explained in algorithm 10. The estimation can be performed in $O(TL)$ time where T is the number of trees in the model and L is the maximum number of leaves in any tree [45]. Thus, the total computational complexity is $O(TL2^n)$. In algorithm 10, the tree is notated by the v - the nodes and their corresponding prediction values in case of leaf nodes, l - the left children of v, r - the right children of v, a - the split values of v, q - the number of observations in v and F - the features used for splitting in each node of v.

Algorithm 7 Estimating $\mathbb{E}[f(x)|x_S]$

```
procedure ExpectedValue($x$, $S$, tree = $\{v, l, r, a, q, F\}$)
    procedure G($j$,$w$)
        if $v_j \neq$ internal then
            return $w \cdot v_j$
        else
            if $d_j \in S$ then
                return $G(l_j, w)$ if $x_{F_j} \leq a_j$ else $G(r_j, w)$
            else
                return $G(l_j, wq_{l_j}/q_j) + G(r_j, wq_{r_j}/q_j)$
            end if
        end if
    end procedure
    return $G(1, 1)$
end procedure
```

Suppose we have a data set X with three features $F = \{F_1, F_2, F_3\}$ and we want to calculate the influence of the three features on the prediction f of the observation x using Shapley values. We consider a single tree model f shown in figure 3.14 and note that $f(x) = 10$. Note that this example is not for outlier detection or isolation forest particularly, but any tree-based model. In the case of isolation forest, the prediction values would represent the path length from root to leaf.

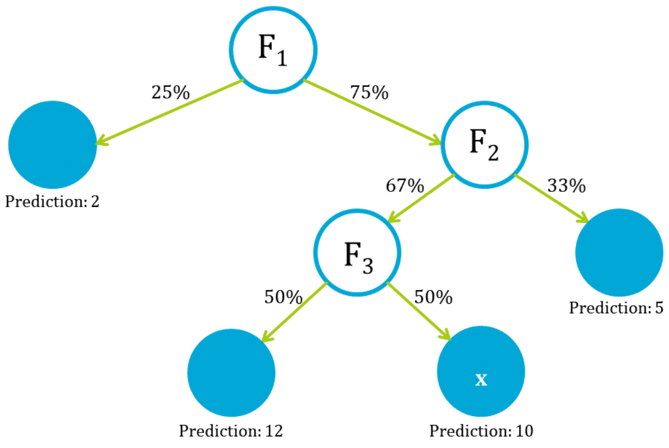


Figure 3.14: The tree-based model f with information about the features used for splitting, the splitting proportion in each node and the prediction value of the leaves.

Let $\mu_{f,x}(S)$ represent the conditional expectation $\mathbb{E}[f(x)|x_S]$, that is $\mu_{f,x}(S) = \mathbb{E}[f(x)|x_S]$. In order to calculate the SHAP value of features $\phi_j(f_x)$ for $j \in \{1,2,3\}$, we need to estimate $\mu_{f,x}(S)$ for all coalitions S of the features in F using algorithm 10,

$$\begin{aligned}
\mu_{f,x}(\emptyset) &= 0.25 \cdot 2 + 0.75 \cdot (0.67 \cdot (0.5 \cdot 12 + 0.5 \cdot 10) + 0.33 \cdot 5)) = 7.25,\\
\mu_{f,x}(\{F_1\}) &= 0.67 \cdot (0.5 \cdot 12 + 0.5 \cdot 10) + 0.33 \cdot 5 = 9.0,\\
\mu_{f,x}(\{F_2\}) &= 0.25 \cdot 2 + 0.75 \cdot (0.5 \cdot 12 + 0.5 \cdot 10) = 8.75,\\
\mu_{f,x}(\{F_3\}) &= 0.25 \cdot 2 + 0.75 \cdot (0.67 \cdot 10 + 0.33 \cdot 5) = 6.75,\\
\mu_{f,x}(\{F_1, F_2\}) &= 0.5 \cdot 12 + 0.5 \cdot 10 = 11,\\
\mu_{f,x}(\{F_1, F_3\}) &= 0.67 \cdot 10 + 0.33 \cdot 5 = 8.333,\\
\mu_{f,x}(\{F_2, F_3\}) &= 0.25 \cdot 2 + 0.75 \cdot 10 = 8,\\
\mu_{f,x}(\{F_1, F_2, F_3\}) &= 10.
\end{aligned}$$

Using the estimations above we can compute the SHAP values for each feature in F as follows,

$$\begin{aligned}
\phi_1(f_x) &= \frac{1}{3}(f_x(\{F_1\}) - f_x(\emptyset)) + \frac{1}{6}(f_x(\{F_1, F_2\}) - f_x(\{F_2\})) + \frac{1}{6}(f_x(\{F_1, F_3\}) - f_x(\{F_3\}))\\
&\quad + \frac{1}{3}(f_x(\{F_1, F_2, F_3\}) - f_x(\{F_2, F_3\}))\\
&\approx 1.89,\\
\phi_2(f_x) &= \frac{1}{3}(f_x(\{F_2\}) - f_x(\emptyset)) + \frac{1}{6}(f_x(\{F_1, F_2\}) - f_x(\{F_1\})) + \frac{1}{6}(f_x(\{F_3, F_2\}) - f_x(\{F_3\}))\\
&\quad + \frac{1}{3}(f_x(\{F_1, F_2, F_3\}) - f_x(\{F_1, F_3\}))\\
&\approx 1.60,\\
\phi_3(f_x) &= \frac{1}{3}(f_x(\{F_3\}) - f_x(\emptyset)) + \frac{1}{6}(f_x(\{F_1, F_3\}) - f_x(\{F_1\})) + \frac{1}{6}(f_x(\{F_2, F_3\}) - f_x(\{F_2\}))\\
&\quad + \frac{1}{3}(f_x(\{F_1, F_2, F_3\}) - f_x(\{F_1, F_2\}))\\
&\approx -0.74.
\end{aligned}$$

The average prediction is $\mathbb{E}[f(X)] = f_x(\emptyset) = 7.25$ so the SHAP explanation of the prediction value 10 is

$$\begin{aligned} f(x) &= \mathbb{E}[f(X)] + \phi_1(f_x) + \phi_2(f_x) + \phi_3(f_x) \\ 10 &= 7.25 + 1.89 + 1.60 + (-0.74). \end{aligned}$$

The SHAP values show that feature 1 is the most important for predicting the score of 10 for observation x and feature 3 is least important. Looking at the sign of the SHAP values we also see that because of its values in feature 1 and 2, x gets an increased prediction score while its value in feature 3 decreases the score. All this goes in hand with the intuition of the importance of the three features looking at the tree model f.

The computational complexity and runtime of algorithm 10 has fortunately been reduced for tree-based models. The paper [45] introduces a method to calculate the values $\mathbb{E}[f(x)|x_S]$ in $O(TLD^2)$ time, where D is the maximum depth of the trees. This method reduces the computation time from being exponential in the number of features to being quadratic in the tree's depth. For the case of multiple features, this is a major improvement. This algorithm is shown in [45] and implemented in the python library *shap* as `TreeExplainer`.

The SHAP additive feature attribution method satisfies three important properties mentioned in the original paper, one being the Consistency property which states that if a model changes such that some features contribution increases or stays the same regardless of the other features, then that feature's impact should not decrease. The authors of [44] also did some experiments to evaluate the consistency of the explanation with human intuition compared to author explanations methods such as LIME [60]. The experiments showed a much stronger agreement between human explanations and SHAP than with the other methods. For a more extensive description of how the SHAP values are derived, we refer to Appendix B.

Explaining Isolation Forest

The SHAP values are, as previously stated, additive feature attributions meaning that for an observation x_i, the average path length h, is a sum of the attribution of different features. Note that the additive property of the SHAP values is less important in case of outlier explanations since we are mainly interested in the features that contribute to a higher outlier score. This is different from supervised regression problems where features that influence the prediction score either by decreasing or increasing it are both interesting to the user. For a data set with d features, the SHAP values $\phi_1, ..., \phi_d$ are such that

$$h(x_i) = \mathbb{E}(h(X)) + \phi_1(x_i) + ... + \phi_d(x_i). \tag{3.47}$$

Note that the prediction model is the average path length instead of the outlier score after normalisation. It would perhaps be more applicable to use the outlier score instead of the average path length but to match the *sklearn* implementation of Isolation Forest and the implementation of the TreeExplainer from the shap package, we use the average path length $h(x)$.

Suppose x_j is identified as an outlier by the Isolation Forest algorithm, meaning that $h(x_j)$ is low compared to $\mathbb{E}(h(X))$, then the feature ℓ such that $\ell = \text{argmin}_l \phi_l(x_j)$ is the feature that contributed most to x_j being an outlier. Furthermore, the second-lowest shap value of x_j is the runner up in contributing to the outlier classification of x_j and so on. The feature with the highest shap value $\kappa = \text{argmax}_l \phi_l(x_j)$ can be interpreted as the feature that pulls the average path length up, reducing the anomaly score and making the instance more normal. The SHAP values with an absolute value close to 0 contribute the least to the outlier score.

In classification problems, the features with the highest absolute SHAP values are the most important ones. However, in outlier detection, we want to know which features are causing the score to be abnormally low compared to the majority of the data. Therefore, we are interested in the features that have the lowest SHAP values. We thus use the SHAP values of each feature as a ranking of their importance where the lowest value corresponds to the most important feature and the highest to the least important feature. In fact, the features that correspond to the higher SHAP values can be interpreted as "the most normal" features for that outlier.

3.2.3. Alter-One-Feature (AOF)

Breiman [6] introduced a method of feature importance by measuring the change in classification error when individual columns are permuted. We want to use a similar approach but on a local level and using the change in outlier score since we do not have classification error information in unsupervised learning.

For each outlier, we calculate the change in outlier score when a column j is permuted or replaced with the columns' mean, median or with zeros. The same Isolation Forest structure is used for a new prediction based on the altered data set. Note that each column represents a specific feature in the data set. If an outlier gets a significantly lower score when a certain column is altered, then the feature that corresponds to that column is considered important for the outlierness of the observation. We consider this method as a good baseline method but note that the runtime can become very high as the number of features increases.

Algorithm 8 Alter-One-Feature

```
procedure AOF(X)
    Train an isolation forest, IF, on data set $X$.
    $scores$ = IF scores of data set $X$.
    Initialise feature importance matrix $FI$ of same size as $X$
    for $j$ in features do
        $X^j = \text{copy}(X)$
        $X^j_{:,j}$ = Alter column $j$ fx. permute, use mean($X^j_{:,j}$), median($X^j_{:,j}$) or zeros
        $scores^j$ = IF scores of the altered data set $X^j$.
        $FI_{:,j} = scores - scores^j$
    end for
    return $FI$
end procedure
```

The resulting scores can be interpreted in a similar fashion as the shap values but are not additive. Thus, a positive feature importance value indicates that the corresponding feature increased the outlierness of that observation. The main disadvantage with this approach is that it only considers changes in the score when a single feature is removed from the data set so it does not incorporate interactions of different features. The complexity of calculating the feature importance using AOF is $O(d \cdot nT\psi)$ since for each altered feature, the outlier scores must be re-evaluated on the isolation forest with the altered data set. Due to the very efficient implementation of the Isolation Forest scoring phase in *sklearn*, the AOF method has a reasonable runtime for data sets with a limited amount of features and the runtime is independent of the number of outliers.

3.2.4. Summary and visualisation

To understand the fundamental difference of these three methods we give a short summary of all three methods.

- **MI-Local-DIFFI:** The Multiple Indicator Local-DIFFI (MI-Local-DIFFI) is our own local variant of the DIFFI method. The feature importance for an individual outlier o is calculated by following the outliers traversal down each of the isolation trees in the isolation forest. For each node, the split feature of the node is given an increased importance weight based on three indicators; *the path length* of the outlier in that tree, the *split proportion* corresponding to this node and the *split interval* length that is calculated by observing the location of the split value on the feature interval.

- **TreeSHAP:** The TreeSHAP uses Shapley values to allocate a fair importance to each feature in the data set. The explanations are of additive nature and for outlier detection, the smallest values are interpreted as the most important outlying aspects and the largest values indicate the most normal features.

- **AOF:** The AOF method calculates the importance of different features by observing the change in outlier score when a single column in altered or permuted. One variant of the AOF is the median replacement method. Isolation Forest is first trained on the original data set and the outlier scores are calculated. To calculate the importance of feature j, we replace column j with its median value and calculate again the outlier score using this altered data set. The change in outlier score is then used as an indication of the feature importance of feature j.

We conclude this chapter by providing an example of an outlier dataset where we use the explanations to interpret the outliers. We create a data set with three dimensions (three features) and 200 observations. The normal observations are three-dimensional vectors where each value is randomly chosen on the interval [0,1]. We select 10 observations to be outliers by adding the constant $k = 3$ to their value of feature 1 so that feature 1 should be the most important feature when it comes to the outlierness of these observations. The data set is shown in figure 3.15 and the outliers are coloured red.

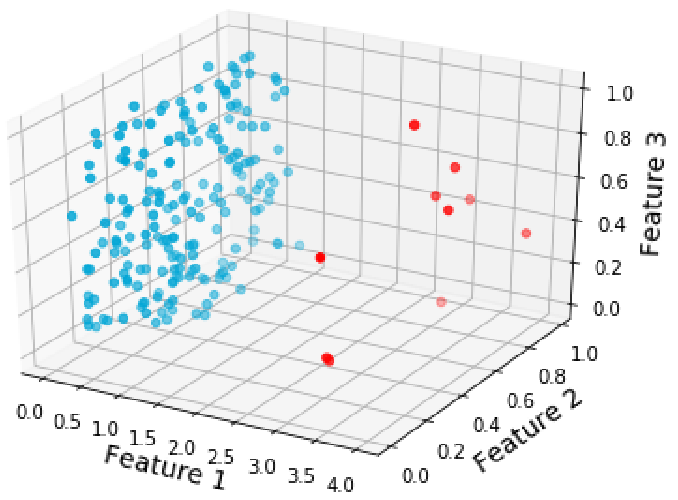


Figure 3.15: A scatter plot of 200 observations of which 10 are clear outlier because of their values in feature 1. The outliers are shown in red.

We observe the average explanation score of each feature considering the ten artificially created outliers. The scores for the three different methods are shown in the bar plot below 3.16. For the MI-Local-DIFFI and AOF, higher values represent more important features but for the TreeSHAP, lower values represent more important features. To simplify the comparison we change the sign of the SHAP values such that higher values also mean more important features.

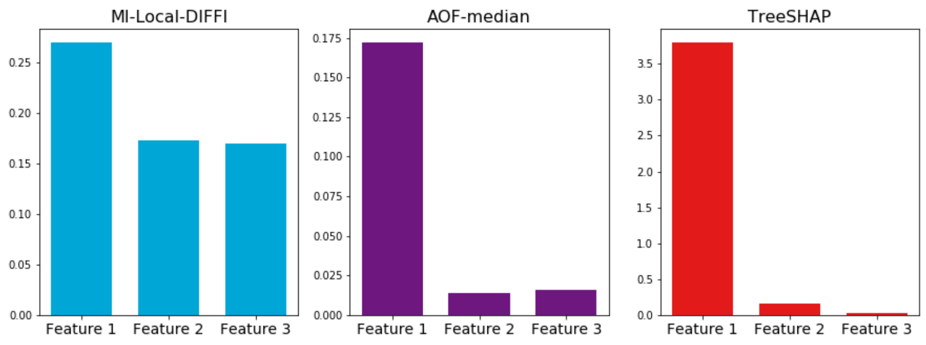


Figure 3.16: Average explanations score of each feature considering all 10 outliers. We show the results of the three different explanation methods and see that all three methods manage to identify feature 1 as the most important.

All three method agree that the first feature is indeed the most important outlying aspect but the values of the scores are different. Since features 2 and 3 are both chosen at random from the interval [0,1] we would expect that they get similar importance scores as is the case for MI-Local-DIFFI and AOF while the TreeSHAP allocates considerably more importance to feature 2 than 3. We also see that AOF and TreeSHAP distinguish more clearly between the importance of feature 1 and the other two features. We analyse the performance of these different methods in detail in chapter 4.1.1.

4 Experiments Using Synthetic Data

We use two types of data sets in our research, synthetic data sets and a real-world data set from Triodos bank. In this chapter, we will focus on the synthetic data sets. These data sets contain non-trivial outliers and information about the ground truth outlying aspects (features) for each outlier. We can thus evaluate the performance of the three different explanation methods by comparing their features importance scores with the ground truth.

The processing of data for outlier detection is slightly different than for other machine learning applications like classification or clustering. We make use of guidelines from the Outlier Analysis book by Aggarwal [2] and we list a few of these guidelines here. Normalization of the columns is not important for the Isolation Forest algorithm although the Extended version requires normalizations. Note that noise filtering methods are not applicable in outlier detection since they could also remove interesting outliers. Therefore, all noise removal must be done by-hand using domain knowledge. Feature selection should also be done implicitly with domain knowledge because normal feature selection methods require the determination of aggregate trends which are hard to determine in a relevant way for outlier analysis. Domain knowledge can be further leveraged in some outlier algorithms by increasing the weight of specific features that are known to be more important than others. Labelled data and human-in-the-loop processes can be of great help if it is available and should be incorporated if possible. Finally, one should consider using an ensemble of the same algorithm or various methods to reduce risk and create more consistent results.

4.1. Synthetic Data

Paper [31] introduces HiCS, an outlier detection method that works well in identifying outliers that are hidden in lower-dimensional subspace projections of the data. To validate their HiCS detection algorithm, the authors produce synthetic data sets that contain non-trivial outliers and a ground-truth knowledge of the features that induce the outlierness. The data set can thus be used to verify the performance of the explanation methods as done by the authors of [47]. A 2-5 dimensional subspace is randomly selected out of the full data space in which high-density clusters are generated. In each subspace, up to 5 objects are modified to deviate from all the clusters in the selected subspace in such a way that the object will not be visible as an outlier in any lower-dimensional subspaces. This is done to ensure that there are no trivial outliers. The synthetic data sets are in total 21, with three sets for each of the seven features sizes, 10, 20, 30, 40, 50, 75 and 100. The data sets all contain 1000 observations but the numbers of outliers differ between them. The different number of outliers in each of the data sets are presented in table 4.1 below.

	Number of features						
	10	**20**	**30**	**40**	**50**	**75**	**100**
Set 1	19	25	44	53	68	110	136
Set 2	19	29	36	50	67	111	123
Set 3	15	35	44	57	68	111	143

Table 4.1: Number of outliers in the 21 different synthetic data sets.

For a better understanding of the synthetic data set, we take a closer look at data set nr. 1. This data set contains 1000 observations, 10 features and 19 outliers. The outliers can be divided into five groups based on the feature subspaces where they clearly deviate from the rest of the observations.

- **Group 1:** The outliers in group 1 are clear outliers in the feature subspace spanned by features $\{8, 9\}$. In total there are 5 outliers in group 1.
- **Group 2:** The outliers in group 2 are clear outliers in the feature subspace spanned by features $\{0, 1\}$. In total there are 4 outliers in group 2.
- **Group 3:** The outliers in group 3 are clear outliers in the feature subspace spanned by features $\{6, 7\}$. In total there are 4 outliers in group 3.
- **Group 4:** The outliers in group 4 are clear outliers in the feature subspace spanned by features $\{2, 3, 4, 5\}$. In total there are 5 outliers in group 4.
- **Group 5:** The outlier in group 5 is clear outliers in the feature subspace spanned by features $\{0, 1\}$ and $\{6, 7\}$. In total there is 1 outlier in group 5.

Let us look at a scatter plot of data set nr. 1 and mark the outliers from Group 2, 3, and 5 with purple, red and yellow colours respectively. We mark the outliers from Group 1 and 4 with green and inliers with blue. We make six 2-dimensional plots with different axes such that each 2-dimensional projections of the feature $\{0, 1, 6, 7\}$ is depicted. We see in figure 4.1 that the outliers from group 2, 3 and 5 are not clear outliers in any two-dimensional subspace of features $\{0, 1, 6, 7\}$ except $\{0, 1\}$ and $\{6, 7\}$ (the top-left and the bottom-right plots).

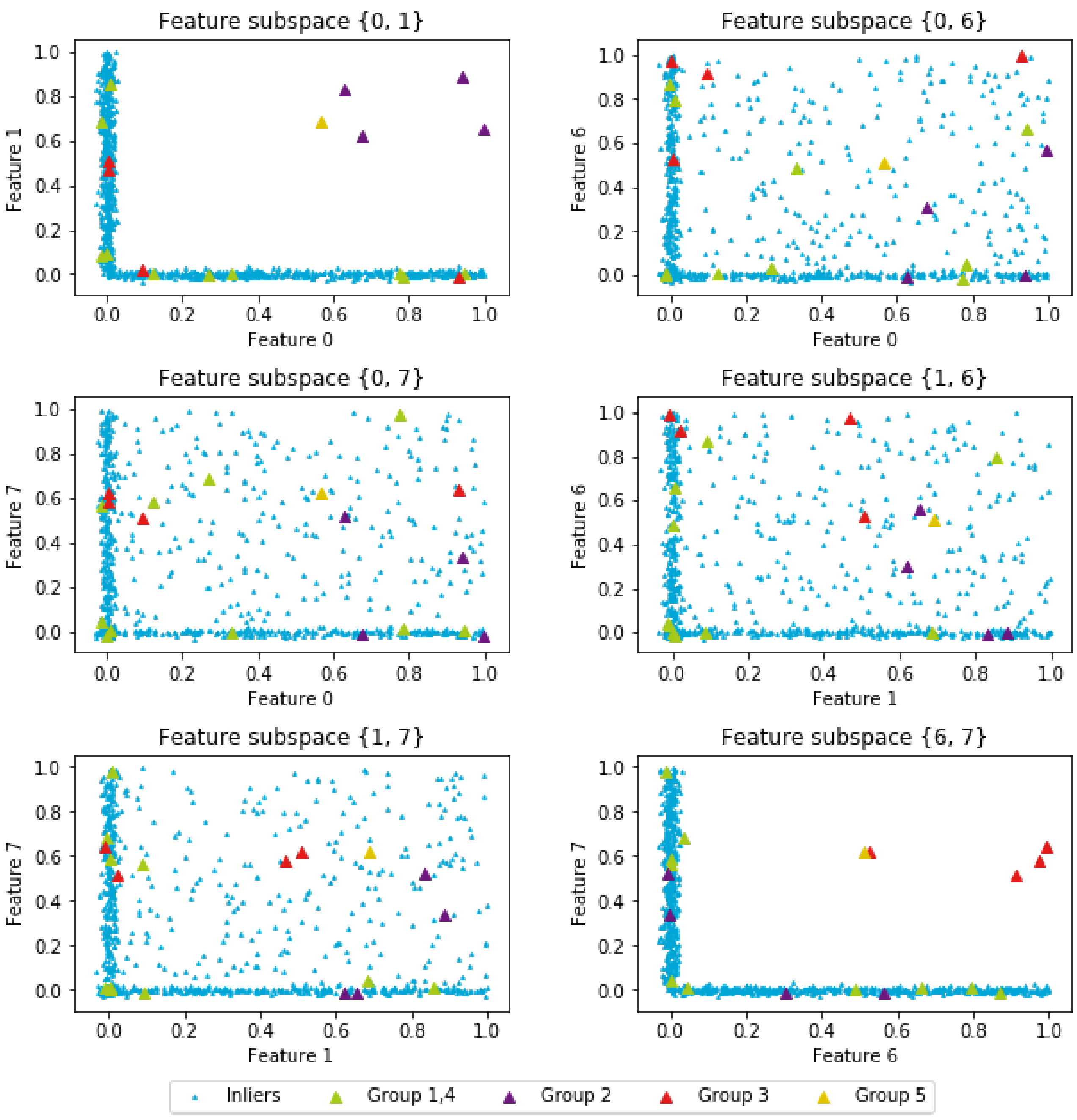


Figure 4.1: We look at two-dimensional projections of the first HiCS synthetic data set. We look at all pairs of features from the subset $\{0, 1, 6, 7\}$ a total of 6 pairs and thus 6 plots. We see that the outliers from groups 2, 3 and 5 (purple, red and yellow) are not clear outliers in any two-dimensional projection of the features $\{0, 1, 6, 7\}$ except $\{0, 1\}$ (purple and yellow) and $\{6, 7\}$ (red and yellow) as seen on the top-left and the bottom-right plots.

4.1.1. Evaluation Metrics

We use two performance metrics, the AUC and AUPRC to evaluate the performance of the explanations and also to evaluate the ability of the Isolation Forest algorithm to identify outliers. By definition, these measures can only be used when the ground-truth/actual information is available. Therefore, we can only use them on the synthetic data set.

To define these two metrics we start by introducing the confusion matrix that compares the actual class of an observation with its predicted class. Suppose we have an algorithm that predicts whether a feature is important ($y = 1$) or unimportant ($y = 0$). The algorithm predicts correctly if either, it predicts important when the feature is actually important (true positive) and it predicts unimportant when the feature is actually unimportant (true negative). There are also two types of incorrect predictions, namely the algorithm predicts important but the feature is actually unimportant (false positive) and conversely, the algorithm predicts unimportant but the feature is actually important (false negative). These different combinations of actual importance and predicted importance can be observed by looking at table 4.2.

		Predicted importance	
		0	**1**
Actual importance	**0**	True negatives (TN)	False positives (FP)
	1	False negatives (FN)	True positives (TP)

Table 4.2: The confusion matrix comparing the predicted and the actual importance.

Using the confusion matrix in table 4.2 above, we can then define a few important metrics, called the accuracy, precision, recall and false positive rate as follows.

$$\begin{aligned}
\text{accuracy} &= \frac{\text{true positives} + \text{true negatives}}{\text{number of observations}} = \frac{\text{TP} + \text{TN}}{\text{TP+TN+FP+FN}} \\
\text{precision} &= \frac{\text{true positives}}{\text{predicted positives}} = \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{recall} &= \frac{\text{true positives}}{\text{actual positives}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\text{false positive rate} &= \frac{\text{false positives}}{\text{actual negatives}} = \frac{\text{FP}}{\text{FP} + \text{TN}}
\end{aligned}$$

The accuracy measures the proportion of correctly predicted observations out of the total number of observations. The precision represents the percentage of the predicted positives that were actually positives while the recall represents the percentage of actually positive observations that were correctly predicted as such. Finally, the false positive rate gives us the percentage of the actually negative observations that were falsely identified as positives.

Using the foundation above, we can define and explain the AUC and AUPRC measures. The **AUC** measure the **A**rea **U**nder the ROC **C**urve. The ROC curve can be obtained by plotting the recall against the false positive rate and is an indicator of the classification ability of a binary classifier as the threshold that divides the two classes is altered. The area under this curve can be computed with the following formula [8],

$$\text{AUC} = \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \mathbb{1}\{f(t_0) < f(t_1)\}}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|},$$

where f is the classifier being evaluated and $\mathcal{D}^0$, $\mathcal{D}^1$ are the sets of negative (unimportant) and positive (important) observations, respectively. The AUC is thus equal to the probability that a classifier will rank a positive instance higher than a negative instance when both are chosen at random. If the AUC is equal to 1 that means that there exists a threshold value such that the negative and positive classes can be perfectly separated.

Similarly, the AUPRC measures the **A**rea **U**nder the **PR C**urve. The precision-recall (PR) curve can be obtained by plotting the precision against the recall for different decision thresholds. Let $p(r)$ be the precision as a function of the recall, r, looking at the PR curve. Then

$$\text{AUPRC} = \int_0^1 p(r)dr,$$

which can be approximated by the average precision score

$$\text{AUPRC} \approx \frac{\sum_{k=1}^{n} P(k) \cdot c(k)}{|\mathcal{D}^1|}$$

where n is the number of observations ($n = |\mathcal{D}^0| + |\mathcal{D}^1|$), k is the rank in the score of observations, $P(k)$ is the precision at cut-off k and $c(k)$ is the actual class of the item at rank k.

Since PR curves do not depend on true negatives (TNs) they are particularly useful for data sets with many true negatives where a high AUC score can be a false indicator of good performance. In explanation evaluation, this holds since the class of true important features (positive) is often rarer than the class of unimportant ones (negative). In outlier detection, this is also the case since the class of outliers (positive) is rare compared to the class of inliers (negative). We thus always look at both the AUC and the AUPRC measures.

Explanations

For each outlier in the synthetic HiCS data, there is a binary indicator vector, $\tau \in \{0, 1\}^d$, that indicates which features represent the ground truth outlying features. For example, if an outlier o is outlying in the feature subspace spanned by features $\{0, 1\}$, then $\tau(o) = (1, 1, 0, 0, ..., 0)$. We evaluate our explanation methods by comparing the resulting feature importance ranking FI, with this ground-truth binary indicator vector τ. Note however that the explanations indicate the importance of the features based on the predictions of the model while the ground truth explanations relate to the structure of the data itself. These two viewpoints, model vs. data, can be different, especially for inaccurate models.

If the true outlying features get the highest feature importance scores, the AUC and AUPRC are 1. We thus compute AUC($\tau(o), FI(o)$)) and AUPRC($\tau(o), FI(o)$)) for each outlier o in the data set X and to measure the performance across all the outliers in the data set we take the mean,

$$\text{AUC}_{FI} = \frac{1}{n_o} \sum_{o=1}^{n_o} \text{AUC}(\tau(o), FI(o)) \tag{4.1}$$

$$\text{AUPRC}_{FI} = \frac{1}{n_o} \sum_{o=1}^{n_o} \text{AUPRC}(\tau(o), FI(o)) \tag{4.2}$$

where n_o is the number of outliers being evaluated and $FI(o)$ are the feature importance explanation scores for outlier o based on some explanation method.

Outliers

Although we explained the AUC and AUPRC metrics above by using the classes of important versus unimportant features, these measures can also be used for the evaluation of the performance of an outlier detection algorithm instead of the explanations. In that case, we have the two classes; outliers (1) and inliers (0), and the results of the model is given with the outlier scores. A perfect AUC and AUPRC score of 1 is reached if there exists a threshold value such that the outliers scores can perfectly separate the outliers from the inliers.

4.2. Experiments with Isolation Forest

We first test the performance of Isolation Forest on the synthetic data set, using the ground truth information about which observations are outliers and which are not. We notice that the performance of Isolation Forest decreases swiftly as the number of features increases and we thus analyse Isolation Forests ability to find subspace specific outliers within a data set. We also test the Ternary Isolation Forest algorithm that was presented earlier which shows better performance results than Isolation Forest on the synthetic data sets.

We report the performance of Isolation Forest on the synthetic data sets in terms of AUC and AUPRC. Since Isolation Forest has a stochastic aspect to it (each isolation tree is randomly built), we calculate the average performance of 10 different randomly generated forests. Note that all our experiments are done with an average over multiple isolation forests and the plots have a black error line which represents +/- one standard deviation of the results over multiple runs. We use Isolation Forest without subsampling and use various numbers of trees, $T \in \{1, 5, 10, 20, 50, 100, 200, 500\}$. Since the outliers in the synthetic data set do not reside in big clusters, subsampling should not increase the performance of the algorithm by reducing masking or swamping effects. In figures 4.2 and 4.3, we see that the highest performance in terms of AUC and AUPRC of Isolation Forest is obtained with 500 trees but a similar performance and faster computation can also be obtained using only 100 trees.

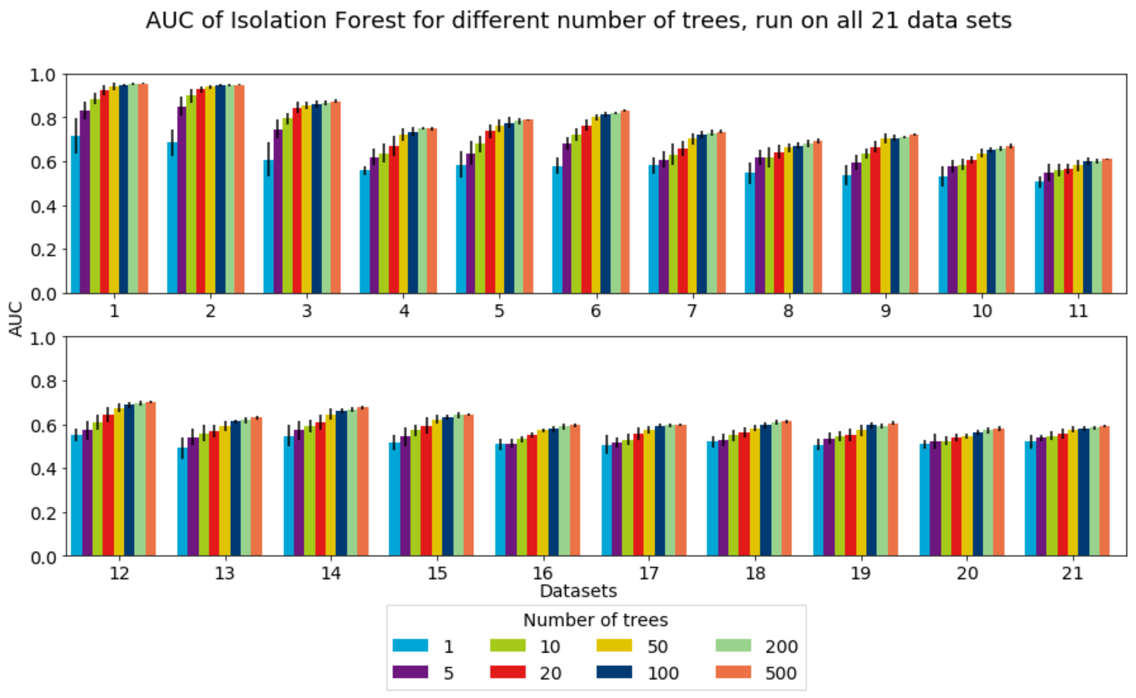


Figure 4.2: The AUC for the 21 data sets with various number of trees in Isolation Forest. The AUC is averaged over 10 runs of Isolation Forest and the standard deviation is shown with the black error line on top of each bar. We see that the AUC increases as the number of trees increases but we believe that satisfactory performance is reached for 100 trees which is also the setting suggested in the original paper. We also see that the AUC decreases in general as the number of features in the data set is higher.

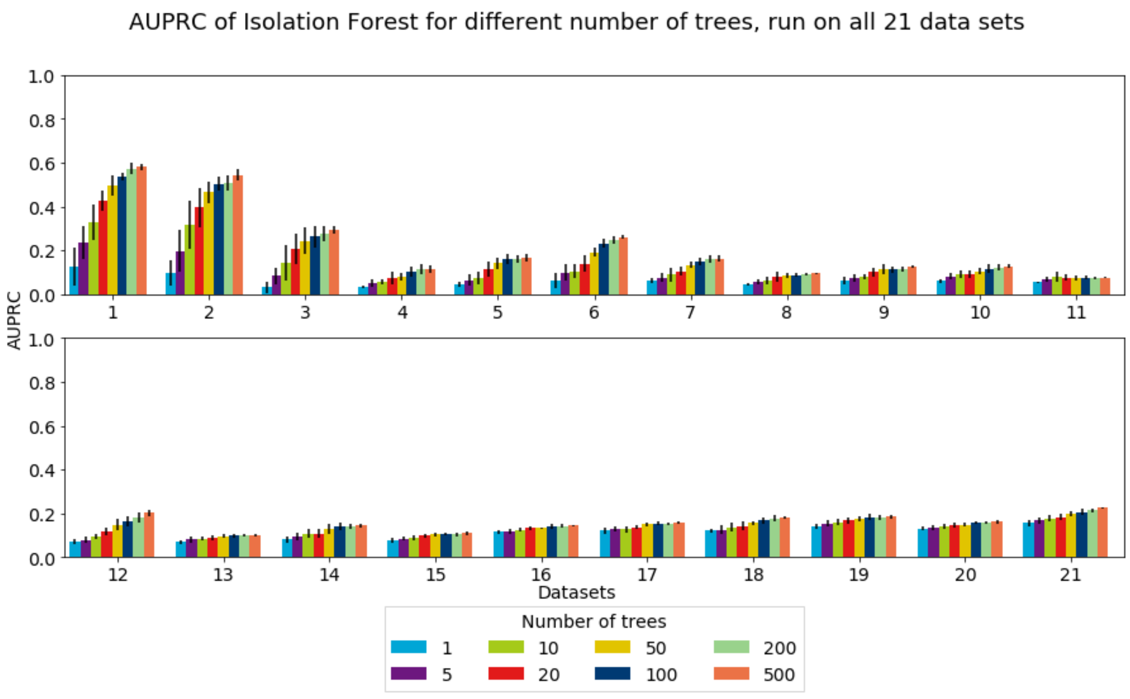


Figure 4.3: The AUPRC for the 21 data sets with various number of trees in Isolation Forest. The AUPRC is averaged over 10 runs of Isolation Forest and the standard deviation is shown with the black error line on top of each bar. The AUPRC is low for most data sets but increases slightly as the number of trees increase.

Now we focus on the values of the AUC and the AUPRC and use only the results with 100 trees. Using 100 trees gives us good performance results and reduces the run time compared to 200 or 500 trees. These measures show that the best performance in terms of accuracy (AUC) is for the data sets with only 10 features and the AUC is decreasing in the number of features. An AUC of 0.5 equals no discriminating ability at all, while usually an AUC of 0.7-0.8 is considered acceptable discrimination [28]. Based on this, Isolation Forest does not manage to distinguish outliers from the inliers sufficiently for all data sets. The AUPRC measures are not great for any of the 21 data sets. We see a trend that the AUC decreases as the number of features increases. That can also be expected since the outliers become harder to identify when the number of subspaces to investigate increases.

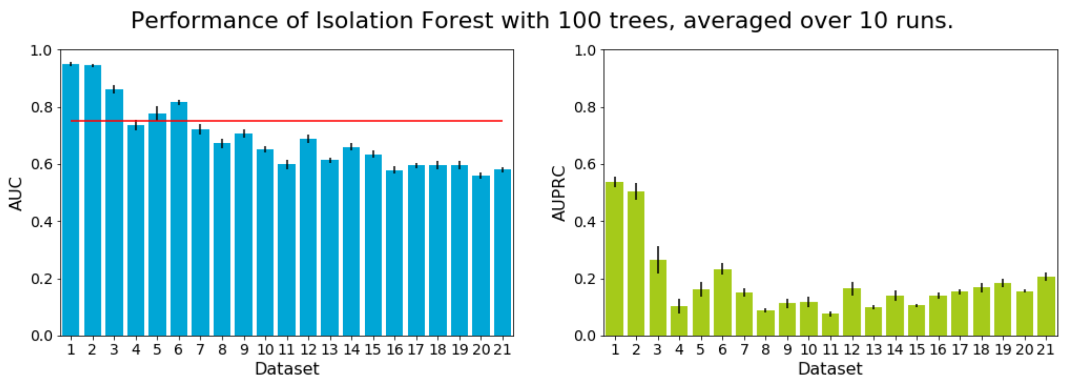


Figure 4.4: The AUC and AUPRC for the 21 data sets with 100 trees. The results are averaged over 10 runs of Isolation Forest and the standard deviation is also depicted.

We see that Isolation Forest is surely not the optimal algorithm for outlier detection in these synthetic outlier data sets. However, these are the only multi-dimensional data sets that we found containing non-trivial outliers with ground-truth outlying feature information. Since the performance of Isolation Forest is not acceptable for the sets with more than 20 features, we choose to evaluate our methods on only the data sets that receive an AUC score of 0.75 or higher, in total 5 data sets as shown in figure 4.4. Three of these data sets contain 10 features (sets 1,2 and 3) and two contain 20 features (sets 5 and 6). Note that these synthetic data sets were specifically generated for the HiCS outlier detection algorithm where the outliers are placed in certain non-trivial subspaces such that the subspace contrast where they are located is high. The results of the HiCS algorithms are very good, maybe because the algorithm specifically searches for these high contrast subspaces. Unfortunately, there is still no convenient implementation in Python or R of the outlier detection component of HiCS so we could not make a direct numerical comparison of the AUC of HiCS vs. Isolation Forest. There are however some graphs in the original paper, indicating that the HiCS manages to have an AUC above 0.75 for each of the different feature sized data sets.

We again point out that the explanations measure the importance of the features to the model while the ground truth outlying aspects relate to the distribution of the data itself. So an outlier x_o may have certain ground truth outlying aspects when looking at the data distribution but Isolation Forest manages to detect this as an outlier based on other features. In that case, the explanations do not match but it does not mean that the explanations were wrong as they could be pointing at those outlying aspects found by Isolation Forest.

In terms of explainability, we are more interested in knowing how the model works than in trying to extract the ground-truth information. Of course, it is best when the two coincide. Therefore we use the data sets that Isolation Forest performed well on to have at least some indication of whether the explanations are pointing at the most interesting features.

Before we use these five synthetic data sets to evaluate the performance of the explanations, we wish to understand why the performance of Isolation Forest decreased so rapidly with increasing feature size of the data sets. In the HiCS paper, the performance indeed decreases as the number of features increases but not as rapidly as in Isolation Forest case. We believe that the answer to this question is found by looking at the ability of Isolation Forest to identify outliers that reside only in specific subspaces.

4.2.1. Subspace Specific Outliers

The synthetic data sets are created such that they contain ground truth outliers that have certain ground truth outlying features. These outliers can only be found when looking at these specific features. To detect such outliers using Isolation Forest, these specific outlying features must be present in the outlier's path in some isolation tree. Then the score corresponding to that path will most likely be lower than the score corresponding to the other paths. However, when the data set is high dimensional and the maximum depth of the trees is limited, the probability of specific features to occur all at once in an outlier path becomes increasingly smaller. Thus, the ability of Isolation Forest to detect such subspace specific outliers decreases with an increasing number of features (dimensionality).

To formalize this, let us look at a data set with d features and consider an outlier path of length s. The features that occur in each split in the path are chosen at random. To simulate the features that occur in a path, we suppose we sample from a set $X = \{1, 2, ..., d\}$ with replacement and let A_i be the event that i is included in our sample. An outlier path of length s corresponds to a sample S of size s and the probability that none of the values in the sample are i equals $\left(\frac{d-1}{d}\right)^s$.The probability that some feature i is not in the outlier path of length s when sampling from d features is then given by

$$\mathbb{P}(A_i^c) = \left(\frac{d-1}{d}\right)^s,$$

and thus

$$\mathbb{P}(A_i) = 1 - \mathbb{P}(A_i^c) = 1 - \left(\frac{d-1}{d}\right)^s.$$

We can also compute the probability that neither i nor j is included in our sample, i.e.,

$$\mathbb{P}\left(A_i^c \cap A_j^c\right) = \left(\frac{d-2}{d}\right)^s$$

and for general A_{i_j}, with $j = 1, ..., l$,

$$\mathbb{P}\left(\bigcap_{j=1}^{l} A_{i_j}^c\right) = \left(\frac{d-l}{d}\right)^s \tag{4.3}$$

Suppose we have an outlier that can only be located in a specific l dimensional subspace of the data spanned by features $i_1, ..., i_l$. In the case of our data set this could for example be an outlier located in an $l = 3$ dimensional subspace spanned by features $i_1 = 6, i_2 = 8, i_3 = 9$. We want to find the probability that all of these features are included in a single path, that is we want to compute $\mathbb{P}(\bigcap_{j=1}^{l} A_{i_j})$.

Theorem 4.2.1. *Let X be a data set with d features. The probability that l specific features $i_1, ..., i_l \in \{1, ..., d\}$ ($i_j \neq i_p$ for $j \neq p$) all occur in a path of length s, $\mathbb{P}(\bigcap_{j=1}^{l} A_{i_j})$, is given by*

$$1 + \sum_{k=1}^{l} \left((-1)^k \binom{l}{k} \left(\frac{d-k}{d}\right)^s\right). \tag{4.4}$$

Proof. Using the inclusion-exclusion criteria we get that

$$\mathbb{P}\left(\bigcup_{j=1}^{l} A_{i_j}\right) = \sum_{k=1}^{l} \left((-1)^{k-1} \sum_{\substack{I \subseteq \{1,...,l\} \\ |I|=k}} \mathbb{P}\left(\bigcap_{j \in I} A_{i_j}\right)\right).$$

Using De Morgan's law and the complement law from set theory, along with equation 4.3 we get

$$\begin{aligned}
\mathbb{P}\left(\bigcap_{j=1}^{l} A_{i_j}\right) &= 1 - \mathbb{P}\left(\left(\bigcap_{j=1}^{l} A_{i_j}\right)^c\right) \\
&= 1 - \mathbb{P}\left(\bigcup_{j=1}^{l} A_{i_j}^c\right) \\
&= 1 - \sum_{k=1}^{l} \left((-1)^{k-1} \sum_{\substack{I \subseteq \{1,...,l\} \\ |I|=k}} \mathbb{P}\left(\bigcap_{j \in I} A_{i_j}^c\right)\right) \\
&= 1 - \sum_{k=1}^{l} \left((-1)^{k-1} \sum_{\substack{I \subseteq \{1,...,l\} \\ |I|=k}} \left(\frac{d-k}{d}\right)^s\right) \\
&= 1 - \sum_{k=1}^{l} \left((-1)^{k-1} \binom{l}{k} \left(\frac{d-k}{d}\right)^s\right) \\
&= 1 + \sum_{k=1}^{l} \left((-1)^{k} \binom{l}{k} \left(\frac{d-k}{d}\right)^s\right) \qquad (4.5)
\end{aligned}$$

□

Results

Consider a data set with $n = 1000$ observations that contains an outlier hidden in a subspace spanned by 3 different features. We calculate the probability that the path of an outlier contains all three of these features for two such data sets with 10 and 100 features respectively. We consider a path of length $\lceil \log_2(1000) \rceil = 10$. According to our formula, the probability of these three features occurring in the same path of length 10 is

$$1 + \sum_{k=1}^{3} \left((-1)^k \binom{3}{k} \left(\frac{10-k}{10} \right)^{10} \right) = 0.2478,$$
$$1 + \sum_{k=1}^{3} \left((-1)^k \binom{3}{k} \left(\frac{100-k}{100} \right)^{10} \right) = 0.0006.$$

which means that such a subspace will be located in a path with $\approx 25\%$ probability for the data set with 10 features but only with 0.06% probability for the data set with 100 features.

This explains why the performance of Isolation Forest is much lower for the data set with 100 features than 10 features. The subspaces that would allow us to isolate the outliers become harder to capture using the random tree path method of Isolation Forest. Not only is it unlikely that they occur in a path but if they occur, the split value is also chosen randomly and may not always result in an isolating split.

4.2.2. Ternary Isolation Forest

Ternary Isolation Forest is defined in section 3.1.6 and below we show that this method outperforms regular Isolation Forest when using the synthetic data sets. Note, however, that in our real-world application and experiments with explanation methods, we stick to the regular Isolation Forest because Ternary Isolation Forest has yet to be implemented efficiently in python.

Using MI-Local-DIFFI we can see how Isolation Forest detects boundary outliers rather than interior outliers. Take a ground truth outlier in the data set with 100 features that has three ground truth outlying aspects. In figure 4.5 we, first of all, see the outlier in its true outlying subspace and then the subspace spanned by the top 3 features from MI-Local-DIFFI. The outlier is more clearly an outlier in the true outlying subspace when considering the number of surrounding observations but it seems like Isolation Forest gives higher importance to features where the outlier is located at the extremes. This supports the fact that a ternary tree should be better in outlier detection because it does not give as much weight to the extremes.

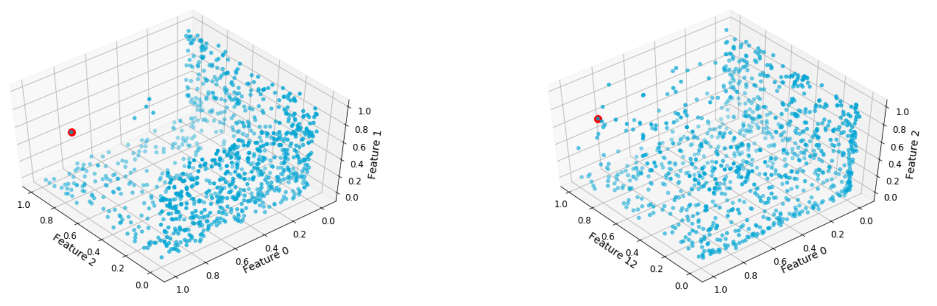


Figure 4.5: Scatter plot of one of the synthetic data sets with 100 features. The outlier is shown in its true outlying subspace and then the subspace identified by MI-Local-DIFFI. One can see the outlier is more clearly an outlier in the true outlying subspace when considering the number of surrounding observations but it seems like Isolation Forest gives higher importance to features where the outlier is located at the extremes. This support the fact that a ternary tree should be better in outlier detection because it does not give as much weight to the extremes.

Results

We compare the performance of regular Isolation Forest and the ternary version on the synthetic data sets. We know that some of the outliers in the synthetic data sets are interior outliers that are only distinguishable from the rest in certain subspaces of the feature space. Therefore, we expect that the ternary version of Isolation Forest can better detect the non-trivial outliers in the synthetic data set which indeed turns out to be the case, both in terms of AUC and AUPRC as shown in figure 4.6. Note that the performance metrics are an average over 10 runs and we also take the average results for data sets with the same number of features.

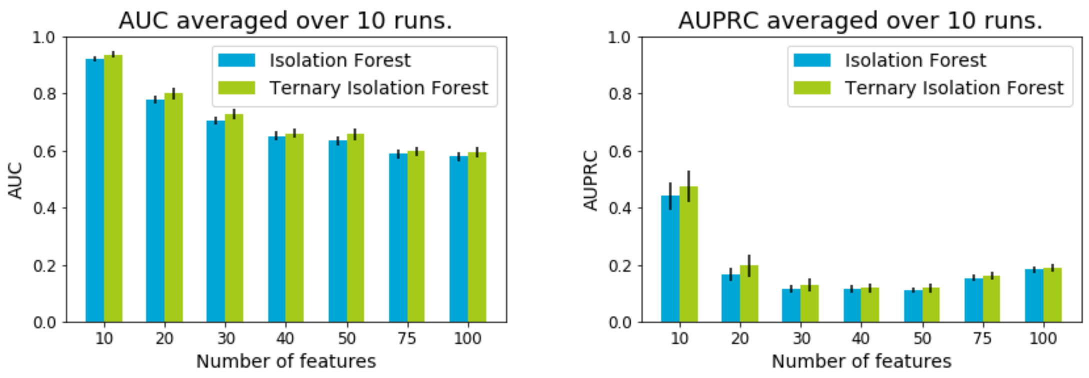


Figure 4.6: The AUC and AUPRC for Isolation Forest and Ternary Isolation Forest. We take the average results for data sets with the same number of features and we also average over 10 runs of the two algorithms. We see that Ternary Isolation Forest performs consistently better than Isolation Forest.

The explanations have successfully helped in understanding the shortcomings of the binary Isolation Forest algorithm which indicates that the explanations do not only give us an understanding of the outlying features of individual outliers but can also be used to improve the method being used, which in this case is the Isolation Forest algorithm. This is a very strong indication of the usefulness of explanations in outlier detection and other machine learning applications.

4.2.3. Runtime Analysis

The runtime (execution time) of Isolation Forest differs greatly between the different implementation of the algorithm. Below we show the difference in runtime for the *sklearn* implementation of Isolation Forest and a more basic implementation of Isolation Forest that does not use C++ for faster computations and does not include any specific runtime optimisations. This more basic version has better alteration possibilities than the *sklearn* version which we, for example, use to implement Ternary Isolation Forest. We measure the runtime of these three implementations; sklearn Isolation Forest, basic Isolation Forest and basic Ternary Isolation Forest, on 9 different randomly created data sets of different sizes. We try a combination of three different number of features, $\{10, 50, 100\}$, with three different number of observations, $\{1000, 10000, 100000\}$.

We tried these data set sizes because they reflect approximately the scope of outlier detection data sets available on ODDS [57]. We run these experiments both for the settings that are suggested in the original paper $T = 100$ and $\psi = 256$ and also with the settings that we use in this thesis $T = 100$ and $psi = n$ where n is the number of observations. We show the results in table 4.3 below.

We first observe that the *sklearn* version runs much quicker than the basic implementation of Isolation Forest indicating that the quality of the implementation matters a lot when it comes to efficiency. Although the computational complexity of Isolation Forest does not depend on the number of features, we see that the runtime is very much influenced by this factor. We also observe that the runtime seems to depend much more on the number of features than the number of observations. The runtime of Ternary Isolation Forest is also greater than of Isolation Forest with the basic implementation. Despite having the same maximum height, the ternary tree has more nodes than the binary tree meaning that the training process takes more time.

Runtime of Isolation Forest in seconds		$\psi = 256$			$\psi = n$		
		Number of observations			**Number of observations**		
Number of features	**Implementation**	1000	10000	100000	1000	10000	100000
10	*sklearn* version	0.31	0.34	0.36	0.31	0.47	0.46
	basic version	2.14	2.29	1.9	3.13	7.67	7.02
	ternary, basic version	4.01	3.19	2.79	6.57	7.51	6.49
50	*sklearn* version	1	1.67	2.71	1.26	2.5	3.56
	basic version	19.13	15.7	14.74	33.74	37.96	41.17
	ternary, basic version	18.88	18.93	17.64	72.09	79.68	79.59
100	*sklearn* version	7.03	20.7	38.41	13.94	28.32	44.57
	basic version	155.73	154.13	142.74	372.1	429.85	455.31
	ternary, basic version	184.19	168.87	193.65	799.37	825.17	865.59

Table 4.3: The runtime in seconds of different implementations of Isolation Forest with subsampling sizes $\psi = 256$ and $\psi = n$ in the two seperate columns. We measure the runtime for data sets of different sizes, where the number of features are in $\{10, 50, 100\}$ and the number of observations in $\{1000, 10000, 100000\}$.

An article on runtime evaluations [36] points out the challenge of correctly evaluating the efficiency and scalability of an algorithm because the runtime evaluation can be very dependent on the implementation of the algorithm. Our analysis shows exactly this as the runtime of the basic implementation can be up to 30 times slower than the *sklearn* version for equally large data sets. We summarize the main findings from our runtime experiment.

- The runtime of Isolation Forest is very dependent on the way it is implemented. The *sklearn* version is much faster than the basic implementation.
- The runtime of Isolation Forest increases more in the number of features than the number of observations.
- Ternary Isolation Forest has about twice the runtime of Isolation Forest considering the basic implementation.
- Increasing the subsampling size has less influence on the runtime of the *sklearn* version than on the basic versions.

4.3. Experiments with Explanation Methods

Next, we use the synthetic data set to evaluate the performance of the explanations. In this case, the ground truth outlying feature information is used instead of the ground truth outlier information. This experiment addresses our second research sub-question about which explanation method to use for local outlier explanations.

We first test a few variations of the MI-Local-DIFFI and the AOF methods and then compare the performance of the three explanations methods, MI-Local-DIFFI, TreeSHAP and the AOF. Note that all the methods deliver the explanations in terms of a feature importance score, where the feature with the highest score can be interpreted as the feature that contributes the most to the corresponding observation being considered an outlier.

4.3.1. Configuration of MI-Local-DIFFI

The new method that we propose, the MI-Local-DIFFI, was developed with influence from the DIFFI method [11]. We used the synthetic data set to validate the performance of the MI-Local-DIFFI and compared the performance of different weighing schemes. The method is described in detail in the Methods chapter. We start by using the synthetic data set to measure the performance of the explanations when incorporating different information on the tree structure. We consider four different sources of information; path length, split proportion, split depth and split interval. The exact weighting scheme is described in the methods section 3.2.1.

- **Path Length.** Shorter paths correspond to more successful isolation of outliers. This means that we are more likely to find important features in those paths than in the longer ones. We incorporate this into the feature importance calculations by weighing up features that occur in shorter paths.
- **Split Proportion.** We also measure the so-called split proportion of each feature in an outlier path, namely the proportion of the data that follows the direction of the outlier in a corresponding feature split. A high split proportion means better isolation power so we expect that features with a high split proportion are more important. We thus give more weight to the features that result in a higher split proportion.
- **Split Depth.** We also check whether the depth of the split in the tree can give us information about the importance of features. It turns out that merely the depth is not a good indicator of how important the feature is. The reason for this is that the split features are chosen randomly and even though a good split located at the beginning of the outlier path could mean that the feature is specifically more important it may also be that the split features that are closest to the leaves are the ones that manage to isolate the outliers. Because of this, we check whether the combination of split proportion and depth provides a better feature importance indicator than if we look at the indicators individually.
- **Split Interval.** Finally, we measure the proportion of the length of the outlier split interval compared to the full feature interval length of each node.

Comparison

For each of the different versions of the method, we use the weighing scheme that gives us the best results on the synthetic data set. Individually, it turns out that merely the split proportion is a good indicator of the feature importance. We test whether combinations of two or more indicators increase the performance of MI-Local-DIFFI. We begin by testing these four indicators below.

1. split proportion (sp)
2. split proportion and path length (sp, pl)
3. split proportion and split depth (sp, sd)
4. split proportion, path length and split depth (sp, pl, sd)

In figure 4.7 below, we see that the path length indicator definitely increases the performance of the explanations. However, the split depth indicator does not seem to make much of a difference in terms of performance. The reason for this could be that the location of specific features is based on a random choice so even though a good split at the beginning of a path would represent significant importance of the corresponding features, a good split at the end of the path can also represent the importance of that features. We thus ignore the split depth indicator.

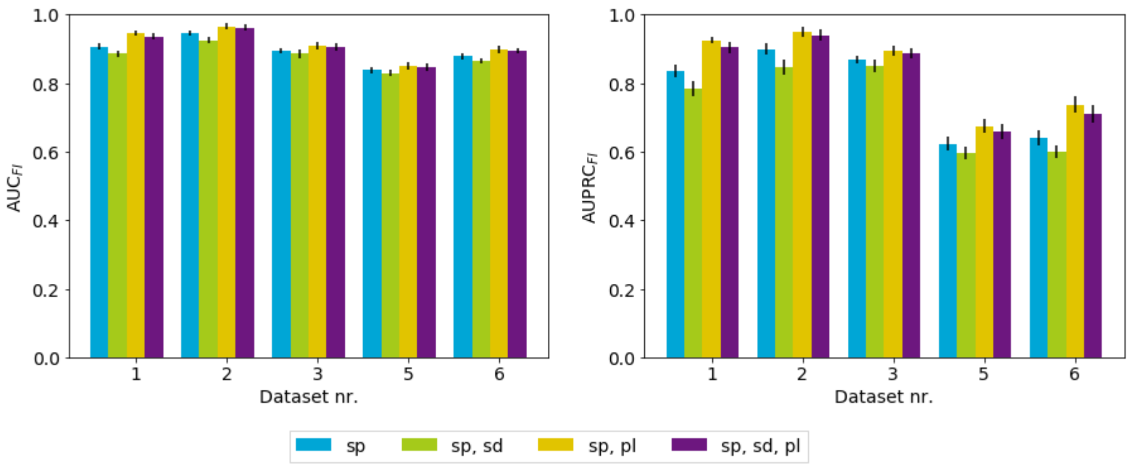


Figure 4.7: The performance of MI-Local-DIFFI measured with AUC_{FI} and $AUPRC_{FI}$ for synthetic data sets 1, 2, 3, 5 and 6. This figure shows four different indicators out of which the yellow bar (sp, pl) shows the best results.

Finally, we add the split interval indicator to the split proportion and path length, thus comparing

5. split proportion and path length (sp, pl)
6. split proportion, path length and split interval (sp, pl, si)

and observe the performance of the explanations, see figure 4.8.

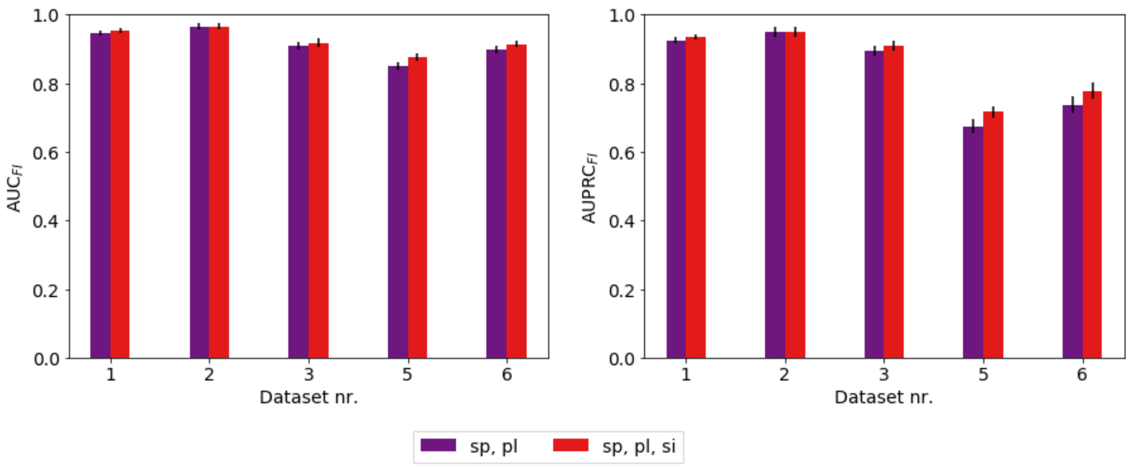


Figure 4.8: The performance of MI-Local-DIFFI measured with AUC_{FI} and $AUPRC_{FI}$ for synthetic data sets 1, 2, 3, 5 and 6 comparing two different combination of indicators. The addition of the split interval indicator shows better results. We thus use the three indicators; split proportion, path length and split interval, as the indicators of MI-Local-DIFFI.

The combination of split proportion, path length and split interval gives the best explanations when evaluating the AUC_{FI} and $AUPRC_{FI}$ measures on the five well-performing synthetic data sets. We thus compare the MI-Local-DIFFI that incorporates these three indicators as described in algorithm 6 with the other two methods.

4.3.2. Configuration of AOF

Next, we test the different versions of the Alter-One-Feature (AOF) method. The different versions are the following;

- Version 1: Permute column j.
- Version 2: Replace column j with its mean.
- Version 3: Replace column j with its median.
- Version 4: Replace column j with zeros.

The AUC_{FI} and AUPRC_{FI} measures for the performance of the feature importance explanations are shown in figure 4.9. We see that the replacement methods (V2-V4) perform similarly well in terms of the AUC_{FI} measure and the zero replacement method (V4) performs best in terms of AUPRC_{FI}. We thus choose to use Version 4, zero-replacement, as the AOF-method to compare with MI-Local-DIFFI and TreeSHAP.

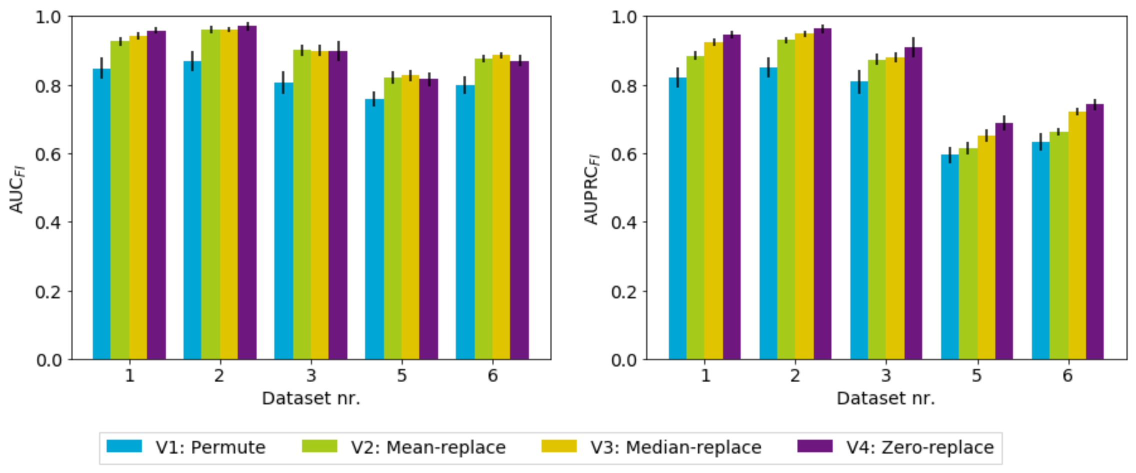


Figure 4.9: The AUC_{FI} and AUPRC_{FI} for the different version of AOF explanation methods. Overall, the V4: Zero-replace indicator gives the best results with the V3: Median-replace following closely behind, giving even better AUC_{FI} resuls for data sets 3, 5 and 6.

However, we remark that the zero-replacement is not likely to be the best version for all data sets. We recommend that the median replace (Version 3) is used if the explanation performance can not be measured because it is most likely to remove the outlying value of the corresponding feature. In the synthetic data set, values around zero are frequent and thus zero becomes a good choice for removal of outlying feature values.

4.3.3. Results

Finally, we compare the three explanation methods, TreeSHAP, MI-Local-DIFFI (sp, pl, si) and AOF (V4: zero-replacement). We continue to use the AUPRC_{FI} and AUC_{FI} measures but also compare the runtime of the three different methods and two α measures that are explained below.

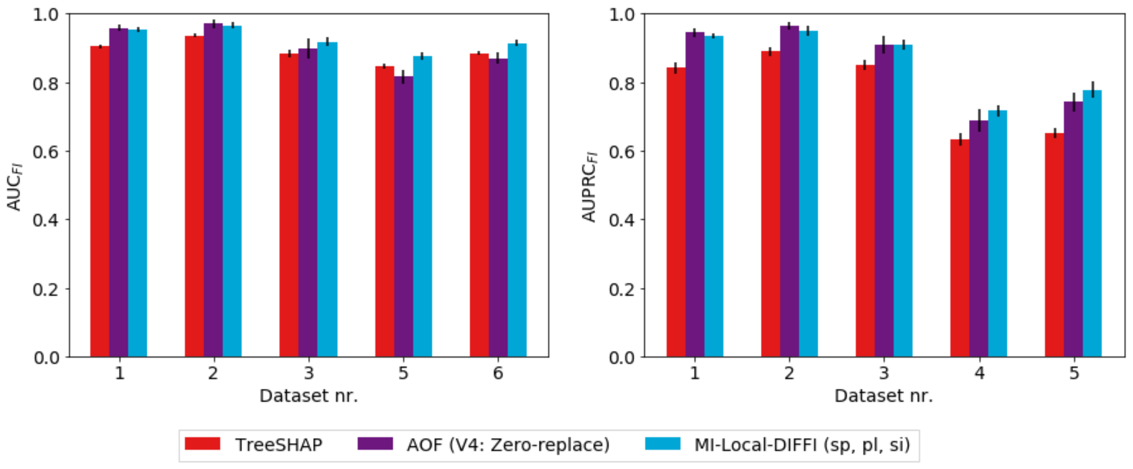


Figure 4.10: The AUC$_{FI}$ and AUPRC$_{FI}$ for the three different explanation methods; TreeSHAP, AOF and MI-Local-DIFFI. Results are averaged over 10 different Isolation Forests. AOF performs better for data sets nr. 1 and 2 whilst MI-Local-DIFFI performs better for data sets nr. 3, 5 and 6.

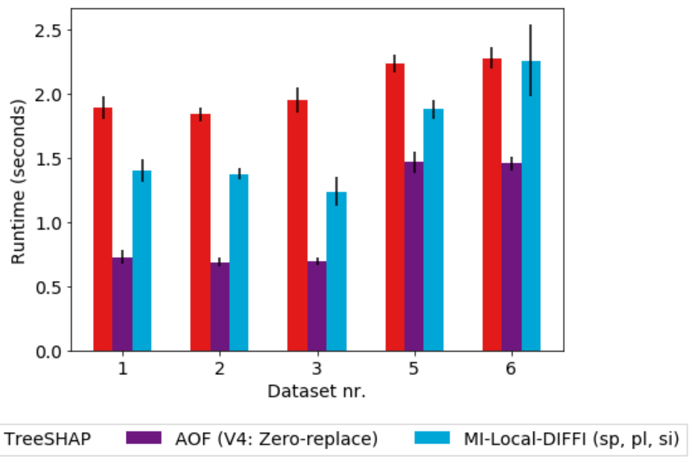


Figure 4.11: The overall runtime (s) for the three different explanation methods. AOF is the fastest but all methods are quite fast for these five synthetic data sets. The runtime however depends on the size of the data set and the setting of the hyperparameters of Isolation Forest so this runtime comparison only holds for the specific data sets and Isolation Forest setting being used in this experiment.

Figure 4.10 shows that the AUC$_{FI}$ measure is similar for the three methods, although the MI-Local-DIFFI performs best on average over the five different data sets. Looking at the AUPRC$_{FI}$ measure, the MI-Local-DIFFI and AOF outperform the TreeSHAP, with MI-Local-DIFFI again receiving the highest average score over the five data sets. In terms of runtime, Figure 4.11 shows that TreeSHAP is significantly slower than the other methods but more detailed runtime analysis of the method is provided in table 4.8. We conclude that the performance is very good for all methods but the MI-Local-DIFFI and AOF outperform the TreeSHAP especially if we look at the AUPRC$_{FI}$. It is interesting that the AOF method, that only observes the changes as one feature at a time is altered, has a very high performance measure on these data sets.

We continue to estimate the ability of these methods to distinguish between the true outlying features and the rest. In addition to the AUC and AUPRC, we want to look explicitly at the explanations scores of the true outlying features vs. the regular features. Ideally, the explanation scores are on average higher for the outlying features than the rest. Before we can make this comparison we standardize the explanation scores for each outlier. Then the first measure we are interested in is defined as

$$\alpha_1 = \frac{|\text{average standardized explanations score of true outlying features}|}{|\text{average standardized explanations score of regular features}|}.$$

We want this measure to be as high as possible as a higher ratio indicates more separability between the explanation score of true outlying features and the rest. Furthermore, we measure the difference between the lowest score of a true outlying feature and the highest score of a regular feature, i.e.,

$$\alpha_2 = \text{Lowest explanation score of a true outlying feature} - \text{Highest explanation score of a regular feature},$$

where a higher α_2 means increased separation ability of the explanation methods. A negative score indicates a failure of the method to give true outlying features higher scores than regular features.

We compare these scores across the different data sets for the three methods. Results are shown in Tables 4.4 and 4.5 below, where we see that the AOF method has the best score in terms of α_1 and MI-Local-DIFFI has the best score in terms of α_2 on average.

α_1		Data set nr.				
		1	2	3	5	6
Method	TreeSHAP	3.96	4.71	4.6	**2.71**	**2.96**
	AOF	**6.45**	**7.32**	**4.86**	2.32	2.74
	MI-Local-DIFFI	3.4	3.58	3.51	2.23	2.51

Table 4.4: The α_1 scores of different methods over the five synthetic data sets. In this case, higher scores represent increased seperation ability between truly outlying and non-outlying featres. Out of the three methods, AOF has the highest scores for data sets nr. 1, 2 and 3 while TreeSHAP has the highest scores for data sets nr. 5 and 6.

α_2		Data set nr.				
		1	2	3	5	6
Method	TreeSHAP	-0.05	0	-0.08	-0.45	-0.36
	AOF	**0.21**	**0.25**	0.1	-0.31	-0.2
	MI-Local-DIFFI	0.19	0.23	**0.11**	**-0.22**	**-0.12**

Table 4.5: The α_2 scores of different methods over the five synthetic data sets. In this case, higher scores represent increased seperation ability between truly outlying and non-outlying featres. Out of the three methods, AOF has the highest scores for data sets nr. 1 and 2 while MI-Local-DIFFI has the highest scores for data sets nr. 3, 5 and 6.

In most practical outlier detection models, the information about outliers and their explanations is however not available beforehand. We must thus decide which explanation method to use without having the ground truth explanations at hand. Since the methods above appear to give quite good results, we look at how they differ in terms of computational complexity, see table 4.6 for general data sets.

	Complexity
TreeSHAP	$O(n_o T L h^2)$
AOF	$O(ndT\psi)$
MI-Local-DIFFI	$O(T n_o \log_2 \psi)$

Table 4.6: Computational complexity of the different explanation methods.

Here, T represents the number of isolation trees in an isolation forest, d represents the number of features (dimensions) in the data set, n represents the number of observations in the data set, n_o represents the number of outliers to be explained, ψ is the size of the subsample that is used to build the isolation forest, L represents the number of leaves in the isolation tree and h represents the height/depth of the isolation trees. We would like to write L and h in the order of the subsampling size ψ since these sizes depend on the isolation tree built with ψ observations. Note that the average height of an isolation tree is $O(\log \psi)$ as computed earlier. In case of a fully grown isolation tree, the number of leaves equals the number of observations ψ. In case of early termination, the height of the isolation trees can at most reach $\log_2 \psi$. A worst-case estimate of the maximum number of leaves in the tree is then $2^h = \psi$. This means that the complexity of the TreeSHAP becomes order $O(T \cdot n_o \psi \log^2 \psi)$, worst case.

In table 4.7, the complexity of the three methods can be shown by looking at only five different factors. However, note that the complexity of TreeSHAP is formulated using a worst-case estimate of the maximum number of leaves. We also notice that higher subsampling size increases the complexity of all the methods, though especially the TreeSHAP.

	Influencial factors				
	Subsampling size	No. of observations	No. of features	No. of outliers	No. of Isolation Trees
TreeSHAP	$\psi \log^2 \psi$			n_o	T
AOF	ψ	n	d		T
MI-Local-DIFFI	$\log_2 \psi$			n_o	T

Table 4.7: Comparing the factors that affect the complexity of the three different methods. TreeSHAP depends most heavily on the subsampling size compared to the other two methods. AOF also depends more on the size of the data set while the other two rather depend on the number of outliers.

Computational complexity can give us an idea of the structure of the algorithms but in applications, the runtime is much more useful as a measure of efficiency and scalability. We thus perform runtime analysis of the three different explanation methods on 9 randomly generated data sets of different sizes, where the number of features is in $\{10, 50, 100\}$ and the number of observations in $\{1000, 10000, 100000\}$. The measures are based on the time it takes to explain a single outlier but we take an average over ten outliers being evaluated at a time. This is because for some of the methods there is a base-runtime that averages out when more than one outlier is evaluated at a time. Furthermore, the runtimes are averaged over 10 evaluation experiments, except in the cases where it takes more than 5 seconds to explain an individual outlier. Then we use only a single runtime evaluation experiment and skip the decimals to avoid the impression that the numbers are more accurate than they actually are. We also measure the runtime for two subsampling size settings, $\psi = 256$ and $\psi = n$. The results are shown in table 4.8 below.

Runtime in seconds		$\psi = 256$			$\psi = n$		
Number of features	**Method**	**Number of observations**			**Number of observations**		
		1000	10000	100000	1000	10000	100000
10	TreeSHAP	**0.05**	0.07	**0.07**	0.13	1.23	8
	AOF	0.08	0.36	4.55	0.07	0.55	8
	MI-Local-DIFFI	**0.05**	**0.06**	0.32	**0.05**	**0.08**	**0.58**
50	TreeSHAP	**0.05**	**0.07**	**0.07**	0.12	1.22	8
	AOF	0.47	3.74	36	0.44	4.55	62
	MI-Local-DIFFI	**0.05**	0.08	0.49	**0.05**	**0.1**	**0.76**
100	TreeSHAP	0.06	**0.07**	**0.08**	0.11	1.07	7
	AOF	1.24	11	110	1.07	14	160
	MI-Local-DIFFI	**0.04**	0.11	0.71	**0.05**	**0.12**	**1.07**

Table 4.8: The runtime in seconds of different explanation methods Isolation Forest with subsampling size equal to 256 and n in two separate columns. We measure the runtime for data sets of different sizes, where the number of features is in $\{10, 50, 100\}$ and the number of observations in $\{1000, 10000, 100000\}$. The runtime corresponds to the time it takes to explain a single outlier but we take an average over ten outliers being evaluated at a time and then average over 10 runs of this process. However, in cases where the runtime is over 5 seconds we do not perform multiple runs.

The AOF method is by far the slowest one and it is only comparable to the other two methods in the case of the smallest data set of size 1000×10. Note, however, that these measures are based on explaining 10 outliers at a time but the AOF method has the same runtime independent on the number of outliers being evaluated which should be taken into consideration when choosing a suitable method. The TreeSHAP and MI-Local-DIFFI are much faster and from the table 4.8, we see that for the smaller subsampling size of $\psi = 256$, the TreeSHAP is on average faster while for a larger subsample size, $\psi = n$, MI-Local-DIFFI is faster.

We conclude this chapter by addressing the second research sub-question (**SQ2**) about which explanation method should be used to explain the results of the outlier detection algorithm on a local level. All the three explanation methods that we tested can be applied for outlierness explanations, although MI-Local-DIFFI and AOF seem to agree more with the ground-truth outlying aspects of the synthetic data sets than TreeSHAP. However, be aware that it does not necessarily mean that those are the best method for all data sets. Finally, with regards to runtime, TreeSHAP and MI-Local-DIFFI are generally faster than AOF as shown in table 4.8. MI-Local-DIFFI is thus the method that shows very good performance both in terms of runtime and performance, although the best runtime performance is very dependent on the size of the data set, number of outliers and hyperparameter setting of the Isolation Forest.

To test completely the performance of the explanations, we will need more diverse data sets that contain ground truth outlying aspects information. However, we did not find such data sets publicly available. Notice also, that our explanation methods measure the most important outlying features according to the Isolation Forest algorithm which may sometimes disagree with the ground truth information.

4.4. Comparison of MI-Local-DIFFI and Local-DIFFI

In general, there are a few more differences to our MI-Local-DIFFI and Local-DIFFI than the technical differences that were discussed is section 3.2.1. First of all, we use the same data set to train and score our outliers, as is the case for most unsupervised outlier detection cases [2], while the authors of DIFFI split the data into training and testing sets. In their experiment with the glass data (described below), they first train Isolation Forest on the non-outlying observations and then create the scores for the labelled outliers using the Isolation Forest. Such an experimental setup simulates a real scenario where ”given a trained instance of the Isolation Forest, the user is interested in deploying the model in an online setting to get the prediction and the corresponding local feature importance score...” [12, p.6]. However, we point out that in most real-world cases we can not exclude the outliers from the training set since the outliers are unknown, so the experiments may not always accurately represent a real-world application.

In case of an online setting, we suggest that the Isolation Forest is trained on a random sample of the full data set that contains both inliers and outliers, as the authors of Local-DIFFI also do in their experiments with synthetic data. This trained Isolation Forest can then be used to score the individual observations in an online fashion. For the Local-DIFFI, only the data of that individual observation is needed for computations but for the MI-Local-DIFFI, the training data has to be merged with the individual observation data in order to calculate the explanations. Despite this, the runtime of the individual MI-Local-DIFFI explanations remains low and can be used in an online fashion. Furthermore, we suggest that if an Isolation Forest model is used in an online setting, then the model should be trained regularly, for example, once per hour or once per day depending on the application, in order to incorporate new observations in the training set. In the case of a non-online outlier detection setting, we suggest that the data is not split into training and testing sets but the model is both trained and scored on the full data set.

Using the online setting suggestions described above, we perform the same experiments as done in the Local-DIFFI paper [12]. The two data sets that are used for these experiments are described in detail in the paper. We discuss them briefly below.

1. *Local-DIFFI synthetic data set*: A data set that consists of observations x_i where each observation is represented by 6-dimensional vector

$$x_i = [\rho\cos(\theta), \rho\sin(\theta), n_1, n_2, n_3, n_4]$$

where n_i are white noise samples and ρ and θ are random variables drawn from

$$\theta \sim \mathcal{U}(0, 2\pi), \qquad \rho \sim \mathcal{U}(0, 3)$$

in case of a normal observation (inlier) while ρ and θ are drawn from

$$\theta \sim \mathcal{U}(0, 2\pi), \qquad \rho \sim \mathcal{U}(4, 30)$$

in case of an outlier.
For the experiment, the training set consists of 1000 of these 6-dimensional observations where 10% are outliers. The Isolation Forest is trained with $T = 100$ and $\psi = 256$. For the testing phase, 300 outliers are generated where 100 are on the x-axis (blue points), 100 are on the y-axis (orange points) and 100 are on the bisector line $y = x$ (green points) as shown in figure 4.12. Prior knowledge for this explainable outlier detection task is that feature F_1 is only relevant for outliers on the x-axis, feature F_2 is only relevant for outliers on the y-axis and both F_1 and F_2 are relevant for outliers on the bisector. All other features are white noise samples and thus irrelevant.

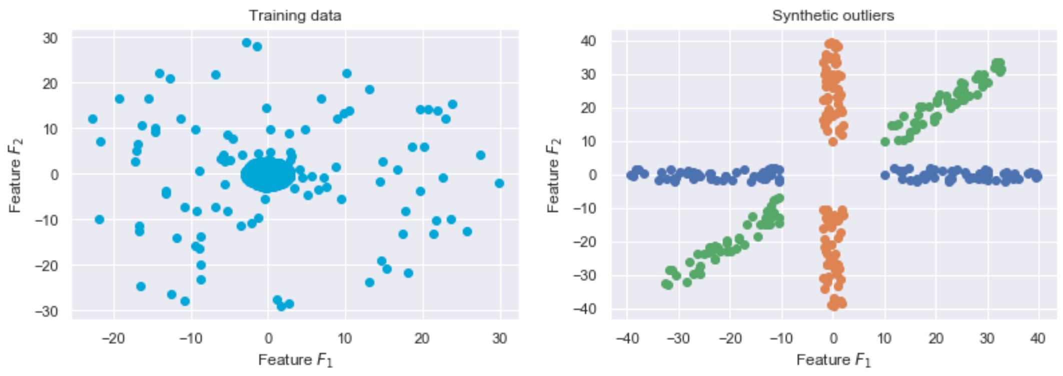


Figure 4.12: The synthetic training data (left) and artificially created outliers (right) projected on the F_1 - F_2 plain. The thick blob in the middle (left plot) represent the normal data points and the scattered points around represent the outliers.

2. *Glass Identification ICU dataset*: This data set consists of 213 glass samples with 9 features which indicate the refractive index (RI), and the concentration of different chemical elements. The original data set had seven classes of glass, but for outlier detection, the first four classes are grouped together (window glass) to form the normal observations and then there are three outlier classes; containers glass (class 5), tableware glass (class 6) and headlamps glass (class 7). The performance of the explanations method is then assessed on outliers from class 7, where the following prior knowledge on headglass is used; the concentration of Aluminum and Barium should be important features when distinguishing between headlamps glass and window glass.
   In the paper by M. Carletti et al. [12], the authors train the Isolation Forest on only the inliers. We suggest that the Isolation Forest should be trained on both outliers and inliers but we exclude half of the class 7 outliers which can then be used for testing. The Isolation Forest was trained with $T = 100$ and $\psi = 64$. We evaluate the explanations on 15 outliers, which were not included in the training set.

Finally, we evaluate the results of Local-DIFFI on our HiCS synthetic data set.

4.4.1. Results

The authors of Local-DIFFI compare their results to the TreeSHAP method as we have also done in this thesis. However, we have a different opinion regarding the usage and interpretation. First of all, they use the absolute value of the SHAP values as the explanation scores and interpret the features that correspond to the highest absolute values as most important. We argue that the absolute value should not be used but instead interpret the lowest values (negative) as the most important outlying features whilst the highest values (positive) represent the most ”normal” (non-outlying) features. Second of all, we disagree with how the runtime (execution time) of the SHAP values is measured. The shap *explainer* (see *shap* python package) only has to be fitted one time, after which it can be used to evaluate the SHAP values on multiple observations. In the Local-DIFFI paper, the runtime is measured such that the explainer is fitted and evaluated for each observation. In an online setting, the explainer can already be fitted before the model is deployed, resulting in very fast runtime.

Now we look at the results of the *Local-DIFFI synthetic data set* experiments, see figure 4.13. We observe the feature rankings for the predicted outliers from the test set and compare the rankings of the Local-DIFFI, the MI-Local-DIFFI and SHAP (TreeSHAP). All the methods manage to identify the top outlying features but the MI-Local-DIFFI outperforms the Local-DIFFI and SHAP in two ways. First of all, the MI-Local-DIFFI manages to identify that for the F_1-outliers, the feature F_2 is the least important feature and the white noise features are then positioned in 2nd to 5th place. We believe that it is indeed true to say that feature F_2 is the least important since the F_1-outliers have F_2 values that are very common among the normal data points. The white noise features are more arbitrarily situated and thus their ranking varies between the 2nd and 5th place. The converse holds for the F_2-outliers. Second of all, for the F_1-and-F_2-outliers on the bisection $y = x$, the most important and second most important feature rank is more equally split between F_1 and F_2 in the case of MI-Local-DIFFI than the other two methods were F_1 is more frequently labelled as the most important feature than F_2. We thus conclude that MI-Local-DIFFI has the most favourable results out of the three explanation methods when evaluated on this specific data set.

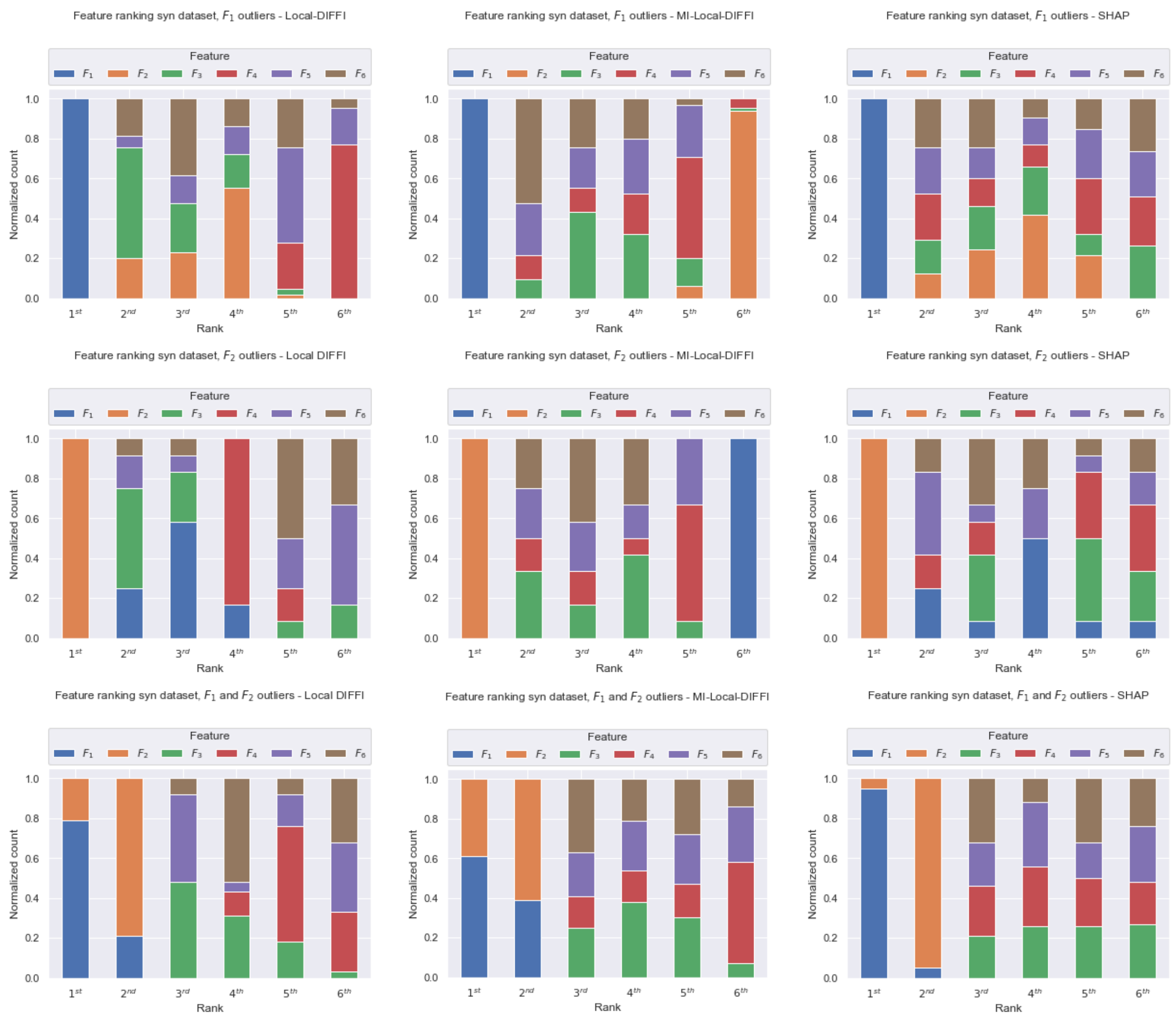


Figure 4.13: The feature rankings for the *Local-DIFFI synthetic data set* based on the scores of Local-DIFFI, MI-Local-DIFFI and TreeSHAP. The first row represents F_1 outliers, the second row represents F_2 outliers and the last row represents outliers on the bisector of F_1 and F_2. We see that all three methods manage to identify F_1, F_2 and F_1, F_2 as the most important features in the case of data sets with F_1-outliers, F_2-outliers and F_1, F_2-outliers, respectively. However, MI-Local-DIFFI also manages to identify F_2 as least outlying in case of the F_1-outliers data and conversely identify F_1 as least outlying for the $F-2$-outliers data. MI-Local-DIFFI also rightfully recognizes F_1 and F_2 as almost equally important in the data with F_1, F_2-outliers while the other two methods allocate considerably more importance towards feature F_1 than F_2.

Next, we evaluate the results on the *Glass data set*, see figure 4.14. Both our MI-Local-DIFFI and Local-DIFFI correctly rank Barium (Ba) and Aluminum (Al) as the most important chemical elements when distinguishing window glass from headlamp glass. Using our interpretation of SHAP values in outlier detection, we also see that the SHAP method gives better results compared to the Local-DIFFI paper. The SHAP recognizes Barium and Aluminum as the most important elements but still gives considerable importance to Magnesium (Mg). We thus conclude that all three methods manage to recognize the most important features Ba and Al, although the segmentation could be clearer, as some outliers still rank Ba and Al among the least important features. Our MI-Local-DIFFI has the most concentration of grey (Ba) and red (Al) out of the three methods when only looking at the 1st and 2nd most important features (first two bars).

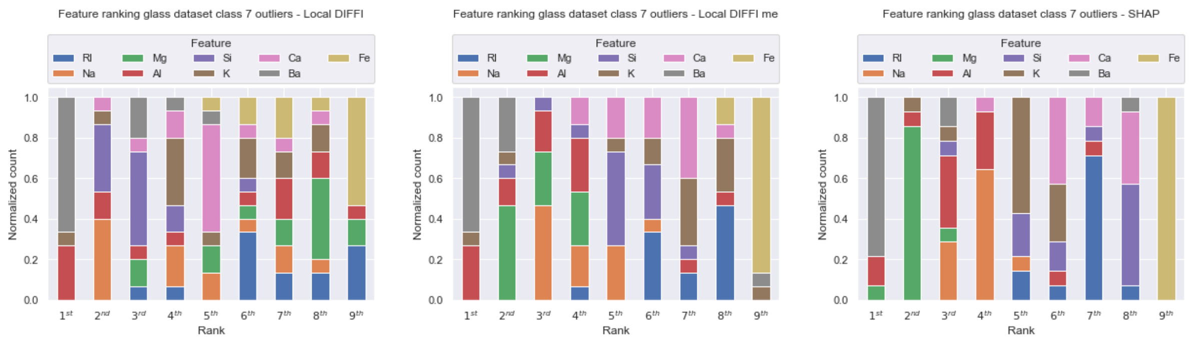


Figure 4.14: The feature rankings for the *Glass data set* based on the scores of Local-DIFFI, MI-Local-DIFFI and SHAP. The most important features are correctly identified as Barium (Ba, grey) and Aluminum (Al, red) for all three methods. However, looking at the first and second most important features, MI-Local-DIFFI allocates most importance to Ba and Al out of the three methods.

Finally, we compare the three methods using the HiCS synthetic data set and our performance measures described in section 4.1.1. We perform the experiments both in an online fashion as done in the Local-DIFFI method, and a non-online fashion as we do throughout this thesis. The results are averaged over 10 runs where the structure of the isolation trees change between runs.

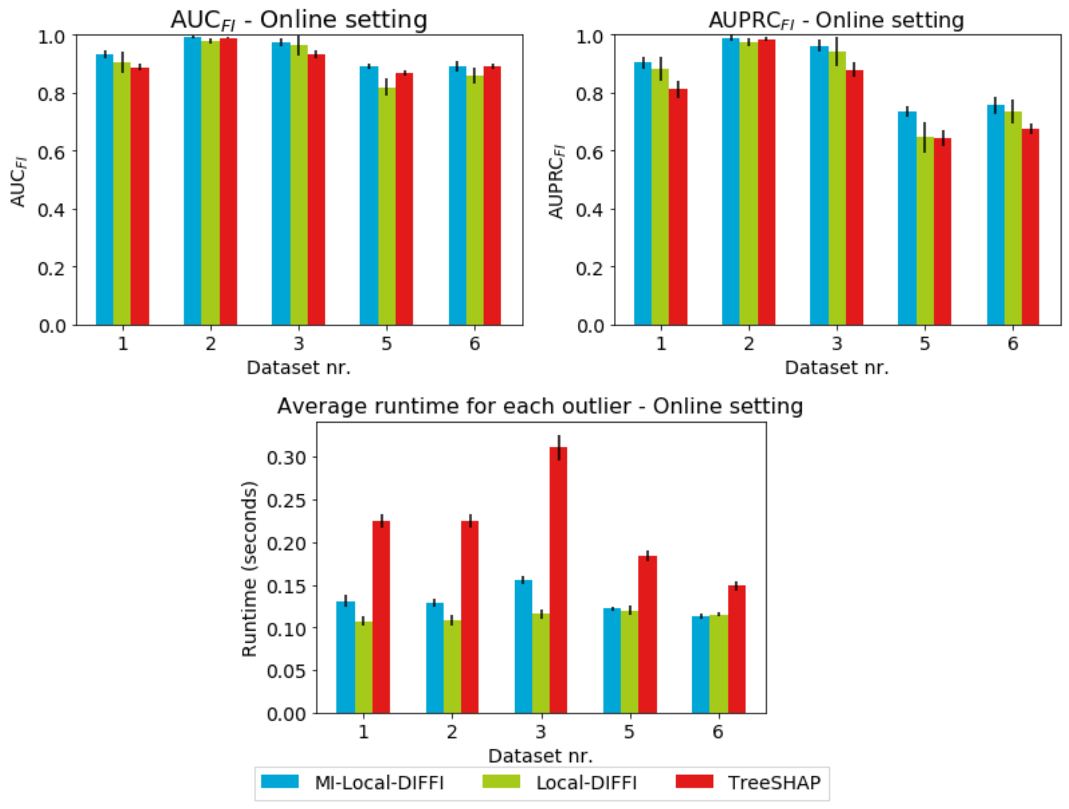


Figure 4.15: The performance measure and average runtime for the three different explanation methods when computed in an online fashion. The numbers are averaged over 10 runs of Isolation Forest. MI-Local-DIFFI consistently outperforms the other methods in terms of AUC_{FI} and $AUPRC_{FI}$.

Looking at figures 4.15 and 4.16, we see that our MI-Local-DIFFI has the highest performance out of the three methods, both in terms of AUC_{FI} and $AUPRC_{FI}$ in the online and non-online setting. Looking at runtime in the online-setting, Local-DIFFI performs most favourably but note that our MI-Local-DIFFI has only a slightly higher runtime. All methods can compute the explanations under one-fourth of a second which should be adequate in most online deployments. We also compare runtime in the non-online setting and see that the total runtime of the MI-Local-DIFFI method is the lowest since it can be optimized when multiple in-sample outliers are being explained at a time. However, we also believe that the runtime of Local-DIFFI can be optimized further in the non-online setting.

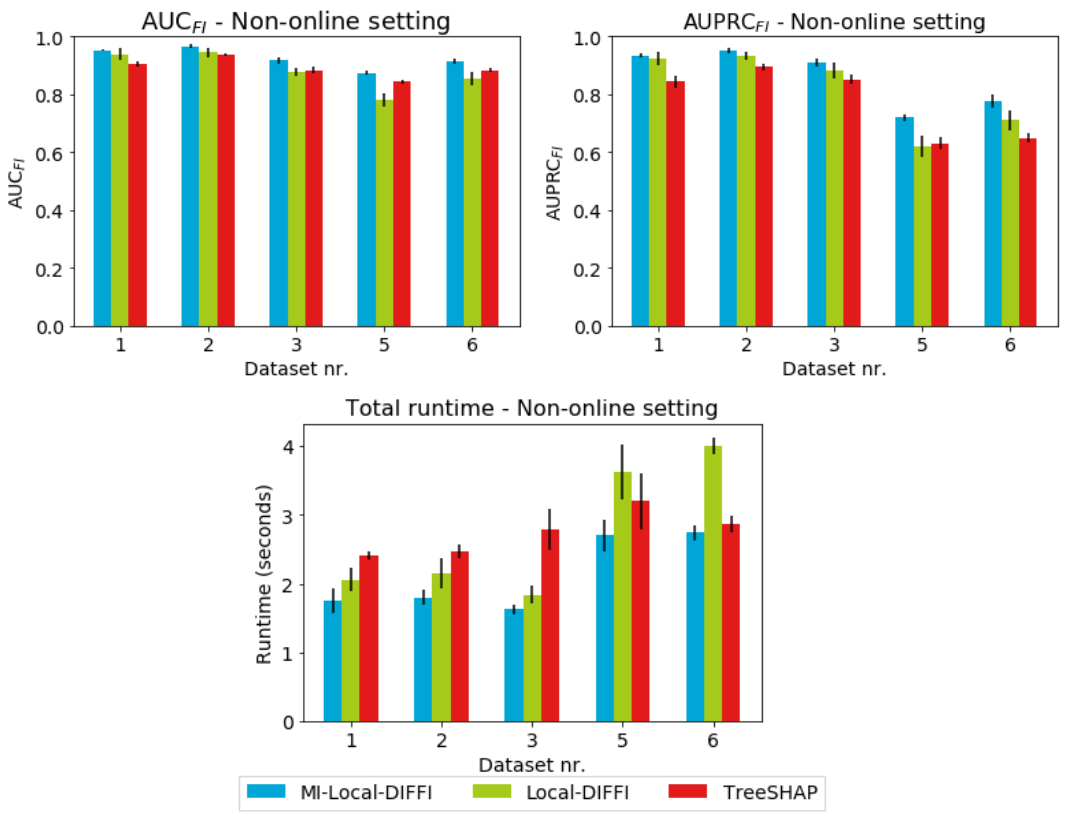


Figure 4.16: The performance measure and total runtime for the three different explanation methods when computed in a non-online fashion. The numbers are averaged over 10 runs of Isolation Forest. MI-Local-DIFFI consistently outperforms the other methods in terms of AUC_{FI} and $AUPRC_{FI}$.

To conclude this chapter we point out that our method MI-Local-DIFFI showed better performance than Local-DIFFI and TreeSHAP in all three experiments. The runtime of MI-Local-DIFFI is slightly higher in the online setting than Local-DIFFI but lower in the non-online setting. However, these runtime measurements only correspond to the data sets in question. We did not find it fair to make a more thorough runtime analysis since the Local-DIFFI method has yet to optimised according to the authors. We have also pointed out a more optimal way to use the TreeSHAP for outlier detection and made some suggestions regarding the splitting of outlier detection data into training and testing sets. However, we do recognise that the Local-DIFFI paper is yet to be published and is subject to change.

5 Application on Real-World Data

We use the methodology presented and tested in the preceding chapters to discover suspicious behaviour in a real-world data set from our industry partner, Triodos Bank. In this chapter, we describe how the data is selected, how the methods are evaluated and the ethical concerns are addressed. We present the results both qualitatively and quantitatively .

The real-world data is quite large so selecting an efficient algorithm is important. Since the *sklearn* version of Isolation Forest is implemented incredibly efficiently, where the same optimized tree structure used in their Random Forest implementation is leveraged, we decide to use the binary Isolation Forest for our problem. The ternary version did show better performance on the synthetic data set but is yet to be implemented efficiently. The *sklearn* version of Isolation Forest also gives faster explanation results due to the efficient structure of the trees. We use the MI-Local-DIFFI to explain the results of the Isolation Forest since it gave the best performance results on the synthetic data sets and it also has a faster runtime when $\psi = n$, which is the setting that we will use for this data set. Furthermore, we use an AOF-based method to give an intuitive evaluation of the MI-Local-DIFFI results which will be described later on.

5.1. Real-World Data

As mentioned, the goal is to identify abnormal customer activity that might be indicative of money laundering or other interesting behaviour that can help the bank to identify potential risks or improve their rule-based system. We decide to extract a tabular data set for our research where the rows correspond to different customers and the columns contain information that can be used to distinguish between money launderers and regular customers. This data set does not contain any ground truth information but rather we would like to use our locally explainable outlier detection methodology to identify interesting customers that may be suspected of money laundering.

5.1.1. Feature Selection

When selecting relevant features for our data set, we must use domain-specific knowledge about money laundering. Therefore, we start by looking at known scenarios and modus operandi (MO) in money laundering that are identified with the help of industry experts both from Deloitte and Triodos Bank. Using these MOs, we identify meaningful information that can be extracted from the bank's data collection. This information can either be of characteristic nature like the business vs. private classification, which industry the customer belongs to in case of a business customer, the general behaviour of a customer, etc. The information can also be related directly to the transactional behaviour of a customer like the amount of cash transactions or the frequency of international transactions. These two classes of information differ mainly in regards to their dependence on time, where the characteristic information is mostly static and does not have to be measured over specific time periods while the transaction information changes from month to month and a specific time window should be determined before measuring.

Modus Operandi/Scenarios	Information to extract	
	Characteristic	Transactions
High Risk Areas		
Certain countries pose a higher money laundering risk than others. Such countries can for example be found on lists from the FATF.		• Amount of transactions from and to high risk areas • Frequency of transactions from and to high risk areas
Cash/MTS Usage		
Money launderers seeking to hide the origin of their funds using cash or Money Transfering Services (MTS) deposits/withdrawals.	• Type of customer • Links to High Risk Areas • Activity Status	• Amount of cash or MTS transactions • Frequency of cash or MTS transactions
Cryptocurrency Usage		
Money launderers seeking to hide the origin of their funds using cryptocurrencies.	• Type of customer. • Links to High Risk Areas • Activity Status	• Amount of transactions related to cryptocurrencies • Frequency of transactions related to cryptocurrencies
Money mule		
Personal customers that enable third parties to gain direct or indirect access to their account.	• Type of customer • Years as a customer	• Amount of cash transactions • Frequency of cash transactions
Smurfing		
Structuring of funds such that they do not go above reporting thresholds.		• Frequency of cash transactions just below regulatory reporting thresholds
Expected Behavior		
Money launderers can possibly be detected by their modes of transactions if they are different from the majority of customers.	• Type of customer • PEP indication • Industry type • Activity status	• Proportion of cash, international, MTS and cryptocurrenc-related transactions out of the total transactions
Change in Behavior		
A sudden peak in transaction activity may be suspicious in terms of money laundering		• Various transaction activity in the last month compared to average in the last year

Table 5.1: Identifying important information by looking at known money laundering Modus Operandi or Scenarios.

In table 5.1 above, we list these scenarios and the corresponding relevant information we would want to extract from the bank's data collection to our own data set. Using this information we can create a data set for our outlier detection project. The transaction information is collected both on a monthly and a yearly basis so that the data set can be continuously updated by constantly looking at the most recent time period. In this case, a month is considered to be 28 days rather than a calendar month to adjust for the imbalance that comes from having months of different lengths. Note that for some scenarios there is no specific characteristic information to be taken into account. In those cases, the table cell is left blank. Below is a full list of the features that are used in the experiment.

Contextual features	**Values**
• Links to High Risk areas • Links to terrorism financing risk related areas • Type of Customer (Business or Private) • PEP (politically exposed person) • NPO (non-profit organization)	• Binary indication (0/1)
Behavioral features	**Measurement Type**
Basic transactions statistics: • All transactions • All CASH transactions • All MTS transactions • All CRYPTOCURRENCY transactions • All INTERNATIONAL transactions • All FISCAL PARADISE RELATED transactions • All HIGH RISK COUNTRY RELATED transactions	*Measure for each category of transaction:* • Sum of amounts on a monthly basis • Sum of amounts on a yearly basis • Count number of transactions on a monthly basis • Count number of transactions on a yearly basis • Percentage of each category monthly amount out of the total monthly amount.
More transactions statistics: • Count different recipients • Count different payers • Count different INTERNATIONAL recipients • Count different INTERNATIONAL payers • Count different payers of large amounts • Count CASH transactions with round amounts • Count number of large amount transactions	*Measure for each category of transaction:* • Counting performed on a monthly basis

Table 5.2: Features that are included in the data set.

The features are 45 in total, 5 contextual and 40 behavioural. We do the analysis separately for business and private customers since the two groups exhibit fundamentally different financial behaviour. For our experiment, we use the last 28 days of 2019 for the 'monthly' computed information and the last $28 \cdot 13 = 364$ days of 2019 for the 'yearly' computed information. We disregard customers that had less than 1000 euros in total yearly transactions during that period. After this filtering, we end up with two 45-dimensional data sets of length appr. 25,000 and 100,000 for business and private customers, respectively.

In the case of supervised learning, where the goal is to predict the correct outcome based on the input data, we can analyse the dependence of different features to the outcome feature. However, in outlier detection, such analysis is not possible due to the absence of outcome features. We thus rely solely on domain expertise.

5.1.2. Evaluation Metrics

In the real-world data set, we do not have a predetermined set of outliers to measure the detection performance of Isolation Forest. On the contrary, we look at the top outliers as predicted by Isolation Forest. Furthermore, we do not have any ground truth information about the most important outlying features so the performance of the explanations can not be confirmed in the same way as with the synthetic data set.

However, we use an intuitive way to verify the most important features indicated by the explanations by computing the change in outlier score when the most important features are one-by-one being altered to the corresponding median feature value. The methodology resembles the AOF method but is used for validation in this case. We choose the median replacement version (V3) instead of zero or the mean because for a random data set, the median will most likely not be an outlying number. If the explanations are correctly identifying the most outlying aspects we expect that the outlier score will decrease. We are focusing on creating human-friendly explanations and one of the observations of Millers [49] extensive research on Explainable AI was the biased selectiveness of humans when it comes to explanations. Furthermore, as pointed out by Molnar [51], we are used to selecting only one explanation as THE primary cause. His advice is to give only 1-3 reasons, even though we know that there is often more to be looked at. We take the upper bound of that interval and evaluate the top three reasons with the method described above.

Suppose we have an outlier o and the corresponding outlier explanation in the form of a feature importance vector, $FI(o)$. We extract the features F_{i_1}, F_{i_2} and F_{i_3} that have the highest, second highest and third highest importance scores, respectively, and calculate the change in outlier score for outlier o when the corresponding feature columns are altered to their median value.

Algorithm 9 Evaluate with an AOF-based method

```
procedure Evaluate($o$ - outlier , $i_1, i_2, i_3$ - three most important features, $X$ - data set, $IF$ - a trained Isolation Forest)
    $s_0(o) = IF$ score of outlier $o$ using data set $X$.
    for $j$ from 1 to 3 do
        $Y = \text{copy}(X)$
        $Y_{:,i_j} = \text{median}(Y_{:,i_j})$                ▷ Alter column $i_j$ to its median value.
        $s_j(o) = IF$ score of outlier $o$ using the altered data set $Y$
    end for
    $Y = \text{copy}(X)$
    $Y_{:,i_1} = \text{median}(Y_{:,i_1})$                ▷ Alter column $i_1$ to its median value.
    $Y_{:,i_2} = \text{median}(Y_{:,i_2})$                ▷ Alter column $i_2$ to its median value.
    $s_{1,2}(o) = IF$ score of outlier $o$ using the altered data set $Y$
    $Y_{:,i_3} = \text{median}(Y_{:,i_3})$                ▷ Alter column $i_3$ to its median value.
    $s_{1,2,3}(o) = IF$ score of outlier $o$ using the altered data set $Y$
    return $s_0(o), s_1(o), s_2(o), s_3(o), s_{1,2}(o), s_{1,2,3}(o)$
end procedure
```

We use the algorithm above to compute the scores $s_0(o), s_1(o), s_2(o), s_3(o), s_{1,2}(o), s_{1,2,3}(o)$, where $s_0(o)$ is the original outlier score of o, $s_i(o)$ is the outlier score of o when the i-th most important feature has taken the median column value for $i = 1, 2, 3$, $s_{1,2}(o)$ is the outlier score of o when the most and second most important feature have both been altered to their corresponding column median value and similarly, $s_{1,2,3}(o)$ is the outlier score when the most, second-most and third-most important features have all been altered to their corresponding column median value. To assess the individual importance of the three most important features we calculate the following sizes,

$$\beta_j(o) = \frac{s_j(o) - s_0(o)}{s_0(o)},$$

for $j = 1, 2, 3$. To assess the combined importance when the two and three most important features are jointly altered to their median, we calculate

$$\begin{aligned}\eta_{12}(o) &= \frac{s_{12}(o) - s_0(o)}{s_0(o)},\\ \eta_{123}(o) &= \frac{s_{123}(o) - s_0(o)}{s_0(o)}.\end{aligned}$$

We show the results of the explanation evaluation for the top 20 outliers in both the private and business customers data sets. We expect that the η and β measures should be negative if the explanations correctly manage to identify the truly outlying features. Note that these measures do not fully validate the truthfulness of the explanations but are decent indicators in the absence of ground truth. To be fully sure that these are the most important features, we would have to have some ”true” outlying aspects information that is unfortunately not available in most use cases.

The customers that are classified as outliers are evaluated with help of a human domain-expert that classifies their relevance into one of the four categories that are shown in table 5.3.

Category	Description
I. Not interesting	The outliers that do not appear to be specifically of interest to us.
II. Transaction behavior is interesting	The outliers that were not suspicious in terms of money laundering but have in some way helped us improve the current system, for example by exhibiting behaviour that we did not anticipate or exposed any data quality issues that could lead to improvements of the current system.
III. Referred for possible EDR	The outliers that showed behavior that was not suspicious per se but indicate that the customer should perhaps belong to a higher risk class are put in a so-called event driven review (EDR) where the current risk class is altered if needed.
IV. Referred for possible SAR	The outliers that exhibited suspicious behaviour are put through a more extensive investigation that ultimately may lead to a suspicious activity report (SAR) filing if the customer is truly suspicious.

Table 5.3: The classification of outliers done by an AML domain expert.

Furthermore, we give a qualitative evaluation of the outlying customers.

5.1.3. Ethical Considerations

Before using machine learning algorithms, a thorough investigation into the ethical aspects of the system is always required. Triodos, being a sustainable bank, looks at these considerations with great care. The Dutch Central Bank (DNB) introduced a guideline for the use of AI in finance [4], mentioning six principles that should stand as a framework to assist financial firms in assessing how responsible their application of AI is. Note that DNB mentions Machine Learning and particularly unsupervised learning as one area of AI research so the following principles surrounding the use of AI in finance clearly apply to our outlier detection application. The six principles (SAFEST) are

- **Soundness**: The AI system should be reliable and accurate.
- **Accountability**: The firm has operationalised accountability for the AI system.
- **Fairness**: The AI system should not inadvertently disadvantage certain groups of customers
- **Ethics**: The AI system should be aligned with the firm's ethical standard.
- **Skills**: The understanding of the strengths and limitations of the organisations AI systems must be present in the firm.
- **Transparency**: The firm should be able to explain how they use AI in their business processes and how the AI systems function.

Before we address these principles we would like to point out that the results from the outlier detection algorithm are never solely used to report suspicious customers. Each outlier is first reviewed by human analysts that evaluate the customer before any action is made. Organizational aspects like ensuring the skills of the professionals, creating an accountability framework and reviewing the alignment with the firm's ethical standards have to be taken care of before the algorithm is put to use.

We look at the principles that relate directly to the design of the algorithms. First of all, the outlier detection algorithm must be accurate. Since we do not know beforehand which instances are outliers, regular classification accuracy measures can not be used. However, we can ensure that the outliers that are detected are indeed abnormal by using the explanation methods mentioned above. If we see that the outliers are indeed outlying when looking at certain features indicated by the explanation methods, we know that the algorithm indeed points to instances that are abnormal compared to the majority of the observations.

Fairness is ensured by avoiding to use sensitive features as input to the outlier detection algorithm. The transparency of the algorithm is greatly enhanced by the explanation methods although it must be ensured that the performance of the explanations is also up to standards. The explanations allow the users to understand the results of the AI system making it easier for them to amend and adapt it.

We conclude that for an outlier detection algorithm to be used in AML, it must be carefully integrated into the organizational framework considering all the ethical aspects involved. High performing explanation methods make the algorithm more transparent and their soundness becomes easier to measure, especially in the case of unsupervised algorithms.

5.2. Results

We now use the Isolation Forest algorithm to detect the outliers in the real-world industry data set from Triodos bank. There are five features of binary type and the rest are numerical features. The binary features are treated as numerical features since the implementation in *sklearn* does not offer other options.

Using our hyperparameter research, we decide to use the full sample in the training phase $\psi = n$ since the number of outliers is very small compared to the size of the data set and the outlier values are suspected to be at the boundaries of the feature values in some cases. For example, the amount of transactions involving cryptocurrency is 0 for the vast majority of customers so if we build isolation trees using only 256 samples, the probability that all the samples have *cryptocurrency amount = 0* is quite high, making the isolation process less effective. Same goes for other more rare modes of transactions like those involving MTS or Cash.

We use 100 trees to limit the runtime and run the algorithm separately for business and private customers. We run Isolation Forest 20 times for both business and private customers. We select a threshold to cut off the outliers from the inliers based on the outlier scores. We use histograms to decide a good cut off threshold but the exact cut off is not important since we only have the capacity to review around 30-50 outliers.

Our goal is to catch the money launderers but we would also like to obtain valuable information that could help us improve the rule-based system. For example, an indication of business usage on personal accounts is a valuable discovery as such misclassification can reduce the monitoring ability of the rule-based system. The outliers are reviewed by a human domain expert that categorizes the outliers into the four categories introduced in table 5.3.

Before we review the outliers from Isolation Forest, we perform two filtering steps. First, we ignore all the outliers that represented customers who had already received an alert from the rule-based system during a similar time period. We do this because we want to identify customers that were not previously detected by the rule-based system and had thus been reviewed before.

Second of all, we filter out the outliers where their corresponding top three explanations refer to amounts that are lower than 1000 euros. For example, an outlier might be an outlier because of 300 euros use of cryptocurrency-related transaction but such a small amount is not subject to further inspection. This is an example of how the explanations can be used for further improvements of results. After these two filtering steps have been applied, the outliers are delivered to a domain expert for review.

5.2.1. Explanation Evaluation

Now, we evaluate the MI-Local-DIFFI explanations using the method described in section 5.1.2 where we defined five measures, $\beta_1(o)$, $\beta_2(o)$, $\beta_3(o)$, $\eta_{1,2}(o)$ and $\eta_{1,2,3}(o)$. We report these measures on the top 20 business and the top 20 private outliers as shown in table 5.4.

We start by looking at the β measures which only take into account the change for one feature at a time. Note that most of the measures are negative meaning that the outlier score decrease as the three most important features according to the MI-Local-DIFFI are replaced by their median value individually. For the outliers that have a positive β, we inspect the corresponding η measures to see whether the combination of features explains the positive β value. For example, the 8th business outlier has a positive β_2 score but since the $\eta_{1,2}$ is lower than the β_1 we can conclude that indeed, combined with the most important feature, the second most important feature induces an even higher outlier score. Such analysis can be made for most positive β scores. However, in some cases, the explanations seem to be incorrect like the third private outlier. When the most important feature is altered to the median, the outlier score increases and combined with the second and third most important feature, it still increases. Individually, the second and third most important feature have a decreasing impact of the outlier score. It thus seems like the second and third most important features are more important than the alleged most important feature.

	Measures for business outliers				
	$\beta_1(o)$	$\beta_2(o)$	$\beta_3(o)$	$\eta_{1,2}(o)$	$\eta_{1,2,3}(o)$
1	-1.5%	-0.9%	-2.3%	-2.7%	-4.8%
2	-0.3%	-3.6%	-4.4%	-8.0%	-12.3%
3	-8.3%	-8.3%	-8.0%	-21.9%	-32.6%
4	-0.1%	-0.3%	-1.4%	-3.2%	-3.5%
5	-1.0%	-9.9%	-4.1%	-11.7%	-17.6%
6	-4.7%	-0.3%	-0.8%	-6.7%	-10.1%
7	-2.0%	-0.4%	-1.2%	-1.9%	-4.5%
8	-2.1%	**0.5%**	**2.9%**	-2.3%	-2.5%
9	-6.6%	-2.6%	**1.4%**	-6.2%	-6.9%
10	-3.4%	**1.2%**	-3.1%	-2.1%	-4.4%
11	-4.4%	-7.7%	-3.2%	-9.4%	-12.9%
12	**0.6%**	-2.2%	**3.6%**	-2.9%	-2.1%
13	-12.1%	-11.0%	-10.2%	-22.2%	-37.7%
14	-9.9%	-6.9%	-3.1%	-14.3%	-20.5%
15	-8.7%	-10.3%	**3.7%**	-22.4%	-23.4%
16	-8.8%	**0.5%**	-0.3%	-12.7%	-16.7%
17	**0.0%**	-7.0%	**1.5%**	-5.0%	-8.8%
18	-2.4%	-14.4%	-5.6%	-19.9%	-27.7%
19	-5.0%	-6.1%	-3.3%	-8.5%	-14.3%
20	-9.8%	-0.9%	-3.7%	-15.0%	-18.9%

	Measures for private outliers				
	$\beta_1(o)$	$\beta_2(o)$	$\beta_3(o)$	$\eta_{1,2}(o)$	$\eta_{1,2,3}(o)$
1	-6.6%	-11.1%	-16.5%	-5.7%	0.0%
2	-3.2%	-10.7%	-17.5%	-5.4%	-3.0%
3	**0.8%**	-3.9%	-10.2%	**0.3%**	**0.0%**
4	-4.4%	-8.2%	-9.4%	-0.9%	-3.3%
5	-7.6%	-17.2%	-24.1%	-8.9%	-2.3%
6	-12.5%	-12.0%	-15.8%	-6.0%	-6.4%
7	-11.9%	-13.6%	-17.4%	-6.2%	-3.5%
8	-7.6%	-13.2%	-16.1%	-0.6%	**0.4%**
9	-5.8%	-10.4%	-11.8%	-4.2%	**2.1%**
10	**0.0%**	-3.9%	-7.1%	**2.3%**	-2.6%
11	-4.3%	-6.0%	-14.7%	-4.7%	-6.6%
12	-7.1%	-17.7%	-16.9%	-9.9%	-1.4%
13	-4.9%	-9.8%	-22.4%	-4.1%	-9.1%
14	-10.3%	-24.3%	-21.9%	-10.2%	-3.8%
15	-2.7%	-9.7%	-10.5%	-2.2%	-1.4%
16	-4.9%	-16.9%	-19.3%	-7.9%	-5.9%
17	-8.9%	-13.7%	-27.7%	-6.9%	-10.7%
18	-13.3%	-15.0%	-12.3%	-0.6%	-6.9%
19	-4.2%	-6.7%	-6.5%	-1.6%	**1.5%**
20	-7.5%	-8.9%	-17.2%	-4.3%	-1.8%

Table 5.4: Explanation evaluation measures for the top 20 business outliers and the top 20 private outliers.

In most cases, the MI-Local-DIFFI explanations are pointing towards features that induce an increased outlier score of the outliers and thus can be regarded as the so-called outlying aspects. The AOF method does not always agree with the order of importance as identified by the MI-Local-DIFFI method. For example in the case of the first business outlier, where the AOF would identify the MI-Local-DIFFIs third most important feature as more important that the MI-Local-DIFFIs most important feature. However, it is difficult to assess which method is correct and in general, we can not verify the effectiveness of the explanations completely in the absence of ground truth knowledge.

The human domain expert that reviewed the outliers found the explanations very helpful and they seemed to indeed point at outlying features. The explanations helped expedite the process of ignoring non-interesting outliers when the explanations pointed to something abnormal but not interesting. They also made it easier to review the outliers that seemed interesting at first, as the reviewer knew where to start looking. Using the explanations we could also manually filter out some low amount outliers, reducing the amount of work for a human expert. We conclude that the outlier explanations are a very important part of the outlier detection task in the context of finding abnormal behaviour that will be reviewed further by human experts.

5.2.2. Outlier Evaluation

We look at the histograms of the outliers scores for both business and private customers, see figure 5.1. The threshold is chosen such that there are about 300-500 outliers identified and that the resulting outliers clearly have a higher outlier score than the large majority of the data set. We do not use a specific thresholding method since there is limited capacity to review outliers. If the capacity would be unlimited, we would select the outliers whose scores are in some upper quantile q. There are also some thresholding methods available in the literature, for example [5], but this particular method did not work well in our case, either resulting in no outliers at all or far too many outliers.

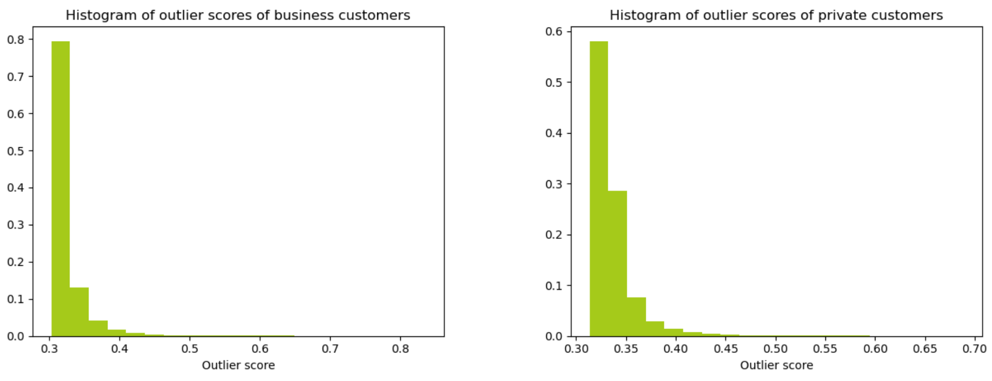


Figure 5.1: The histogram of outlier scores for business and private customers.

For the business customers, an outlier score threshold of 0.4 was determined by looking at the histogram of the outlier scores which resulted in a total of 396 outliers of which 248 had already received an alert from the rule-based system. We review a few of the top outliers from the remaining $396 - 248 = 148$ customers that had not received an alert.

For the private customers, an outlier score threshold of 0.45 was determined by looking at the histogram of the outlier scores. This resulted in 326 outliers of which 115 had already received an alert. This overlap between the rule-based system and the outlier detection system shows that there is a set of customers that both systems flag but also some customers that are only flagged in one of the systems.

Once the already alerted customers had been removed from the list of outliers we created the MI-Local-DIFFI explanations and performed the second filtering step based on the explanations. After these two filtering steps had been performed, a total of 41 outliers were reviewed in detail by an AML domain expert. There were 15 outliers from the set of private customers, 13 from the set of business customers and another 13 from the set of business customers after a few features had been removed from the data set. The features that were removed were the PEP and NPO indicators since these features are already imbalanced, meaning that there are very few positive PEP and NPO customers, making them more prone to be identified as outliers for merely the sake of being PEPs or NPOs. The results are shown in table 5.5.

	Data set		
	Private	**Business**	**Business, no PEP+NPO**
I. Not interesting	3	8	7
II. Transaction behavior is interesting	3	3	2
III. Referred for possible EDR	7	2	2
IV. Referred for possible SAR	2	0	2
Total number of outliers reviewed	**15**	**13**	**13**

Table 5.5: The results of the outlier review done by an AML domain expert.

The results of the review indicate that outlier detection is indeed a useful tool in AML since more than 50% of the outliers were in categories II., III. or IV. We were able to detect customers that were using private accounts for business purposes, we identified possible improvements related to data quality, a few outliers were also moved to a higher risk class and some were sent on for further possible SAR investigations. The possible SAR cases were false positives but one resulted in an increased risk classification.

Although this project has not resulted in any SAR filings yet, we have shown that reviewing the outliers can help us improve the AML system in the above-mentioned way. Note that compared to the number of alerts that are raised with the rule-based system, there are few SARs reported monthly (high false-positive ratio). Thus, even though the 41 outliers that were reviewed in this data set did not result in any SAR filings, we will need more monthly data sets to test the effectiveness of the system to identify SAR cases. Since this first review already leads us to some interesting cases, the plan is to do a similar experiment on a monthly basis and continue to identify interesting customers that can help us improve the current system.

6 Conclusion

In this thesis, we set out to answer the main research question (**RQ1**) regarding the extent in which explainable outlier detection can increase the ability of AML systems to detect suspicious customers. In order to find a suitable explainable outlier detection method, we set out to answer two sub-questions (**SQ1** and **SQ2**) which we will also address in this conclusion section. We also discuss the possibilities for future work in the field of local explanations for outlier detection and the limitations of our work.

6.1. Research Questions

We start by answering the two sub-questions and then answer our main research question.

SQ1: *Which outlier detection algorithm is best suited for our problem setting?*

We analysed the literature of outlier detection methods and concluded that the Isolation Forest was the most suitable method especially when taking into consideration scalability and effectiveness. We give a detailed description of the Isolation Forest algorithm in section 3.1 and in section 4.2 we carry out some experiments with Isolation Forest. We show that the ability of Isolation Forest to detect subspace specific outliers reduces significantly as the number of features increases. We also suggest an alteration of Isolation Forest which uses ternary trees instead of binary, giving better performance results on the synthetic HiCS data set without increasing the complexity.

SQ2. *Which explanation method should be used to explain the results of the outlier detection algorithm on an individual prediction level?*

At the beginning of our research project, we did not find any methods in the current literature that delivered local explanations for the Isolation Forest specifically. We thus decided to compare three different kinds of local explanation methods that could be applied to predictions from Isolation Forest. These methods are MI-Local-DIFFI, TreeSHAP and AOF, each described in detail in section 3.2. As mentioned, Local-DIFFI was introduced by the authors of DIFFI towards the end of this thesis project but fortunately, the authors provided their code on github, so we could also compare this method to our MI-Local-DIFFI.

The MI-Local-DIFFI is our own local variant of the previous DIFFI method and we dived deep into the Isolation Forest algorithm to formulate this method. The TreeSHAP had previously been used in supervised classification problems but we did not find anything in the literature where the SHAP values were used for outlier explanations as we tested in this thesis. Furthermore, the AOF method has not been used in this format before but is very intuitive and similar methods are presented in the literature for supervised problems.

To answer the second research subquestion, we did some experiments to compare the performance of different explanation methods. All the methods seemed to work well in identifying the most important outlying features of individual outliers but MI-Local-DIFFI and AOF showed more favourable performance than the TreeSHAP. These methods take into account different aspects of the Isolation Forest algorithm to reach their explanation scores so their runtime depends on different elements. This should be taken into account when choosing a suitable method for a specific problem. Finally, we showed that the MI-Local-DIFFI performed more favourably than the Local-DIFFI [12] so we can conclude that MI-Local-DIFFI is a novel contribution to the literature on explainability methods for Isolation Forest.

RQ1. *To what extent can explainable outlier detection increase the ability of AML systems to detect suspicious customers?*

Once we had established an explainable outlier detection methodology by combining Isolation Forest and MI-Local-DIFFI, we could apply this methodology to the real-world data set from Triodos bank that we specifically engineered for AML purposes. The explanations turned out to be very important when assessing the level of interest of different outliers. A domain expert reviewed a total of 41 outliers of which 8 had interesting transaction behaviour that could possibly help us improve the rule-based system, 11 were referred to possible Event-Driven-Review and 4 were referred to possible SAR filings. After further investigation, none of the outliers resulted in SAR filings but a few interesting observations came to light. Moving forward, there will be a monthly run of the explainable outlier detection experiment with the goal of finding more interesting observations that can help improve the current system.

We conclude that although there were no SARs filed, explainable outlier detection can increase the ability of AML systems to detect suspicious indicators within the group of customers as our methodology detected some customers that had not received an alert from the rule-based system but were put to higher risk classes as a result of our outlier detection. While reviewing the outliers, we also saw some interesting behavioural patterns and possible points of improvements of rules in the rule-based system. There were also a number of outliers referred for *possible* EDR or SAR which indicates that the detected outliers indeed show signs of suspicious activity although after further inspection none resulted in a SAR filing.

6.2. Limitations and Future Work

In this project, we have addressed our research questions and made some interesting findings. We would like to discuss the potential for further work in the field of locally explainable outlier detection and at the same time we point out some limitations to our work.

First, Isolation Forest only performed well on a few of the synthetic data sets that were used to evaluate the explanation methods. The existence of a data set with ground truth outlying feature information on which Isolation Forest has good performance (AUC > 0.75 and AUPRC > 0.5) would help in evaluating the explanation methods even better. Such a data set could be created using real data combined with an expert opinion about the true outlying features for each outlier. We believe that our proposed methodology of locally explainable outlier detection can be used in more domains than AML, for example in fault detection, medical research or network intrusion. In those cases, it might be possible to create an evaluation data set as described before.

In this work, we have focused on the performance of the outlier detection and explanation scores without using cut-off thresholds to distinguish between inliers/outliers and important/unimportant explanations, respectively. Further analysis is needed to explore which method is best suited when such a cut-off threshold is desirable.

We also mention that an efficient implementation of Ternary Isolation Forest is very important for the scalability of that method and its future use in problems involving large data sets. Since the method showed better performance than the binary version of Isolation Forest, we believe that the implementation of Ternary Isolation Forest in *sklearn* would be very interesting.

Finally, we want to point out that we mainly worked with numerical and binary-categorical features. Further extensions of the methods must be made so that Isolation Forest and the explanation methods also work for any kind of tabular data, such as non-binary categorical data.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 09 2018.

[2] Charu Aggarwal. *Outlier Analysis*. Springer International Publishing, 2011.

[3] Claudio Alexandre. A multi-agent system based approach to fight financial fraud: An application to money laundering. *Preprints*, 01 2018. URL http://dx.doi.org/10.20944/preprints201801.0193.v1.

[4] Nederlandsche Bank. General principles for the use of artificial intelligence in the financial sector. 2019.

[5] Mohamed Bouguessa. Modeling outlier score distributions. In *Advanced Data Mining and Applications*, pages 713–725. Springer Berlin Heidelberg, 2012.

[6] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 01 2001.

[7] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, page 93 104, New York, NY, USA, 2000. Association for Computing Machinery.

[8] Toon Calders and Szymon Jaroszewicz. Efficient auc optimization for classification. In *Knowledge Discovery in Databases: PKDD 2007*, volume 4702, pages 42–53, Berlin, Heidelberg, 09 2007. Springer Berlin Heidelberg.

[9] Ramiro Camino, Radu State, Leandro Montero, and Petko Valtchev. Finding suspicious activities in financial transactions and distributed ledgers. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 787–796, 11 2017.

[10] Dang Cao and Phuc Do. Applying data mining in money laundering detection for the vietnamese banking industry. In *Intelligent Information and Database Systems*, pages 207–216, Berlin, Heidelberg, 03 2012. Springer Berlin Heidelberg.

[11] Mattia Carletti, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. Explainable machine learning in industry 4.0: Evaluating feature importance in anomaly detection to enable root cause analysis. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 21–26, 2019.

[12] Mattia Carletti, Matteo Terzi, and Gian Antonio Susto. Interpretable anomaly detection with diffi: Depth-based feature importance for the isolation forest. *arXiv preprint arXiv:2007.11117*, 2020.

[13] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. *Outlier Detection with Autoencoder Ensembles*, pages 90–98. 06 2017.

[14] Zhiyuan Chen, Le Khoa, Amril Nazir, Ee Teoh, and Ettikan Karuppiah. Exploration of the effectiveness of expectation maximization algorithm for suspicious transaction detection in anti-money laundering. *ICOS 2014 - 2014 IEEE Conference on Open Systems*, pages 145–149, 02 2015.

[15] Tat-Man Cheong and Yain Whar Si. Event-based approach to money laundering data analysis and visualization. *VINCI 2010: 3rd Visual Information Communication - International Symposium*, pages Article No.: 21, pp 1?11, 01 2010.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[17] Remi Domingues, Maurizio Filippone, and Jihane Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 02 2018.

[18] Rafal Drezewski, Jan Sepielak, and Wojciech Filipkowski. The application of social network analysis algorithms in a system supporting money laundering detection. *Information Sciences*, 295:18–32, 02 2015.

[19] S. Gill Williamson Edward A. Bender. *Lists, Decisions and Graphs - With an Introduction to Probability*. University of California, 2010.

[20] FATF. International standards on combating money laundering and the financing of terrorism & proliferation, 2012–2019.

[21] FATF. Money laundering, 2019. URL https://www.fatf-gafi.org/faq/moneylaundering/#d.en.11223.

[22] T.S. Ferguson. *Game Theory*. 2014. URL https://www.math.ucla.edu/~tom/Game_Theory/coal.pdf.

[23] Ayshan Gasanova, Alexander Medvedev, Evgeny Komotskiy, Kamen Spassov, and Igor Sachkov. On the use of data mining methods for money laundering detection based on financial transactions information. In *AIP Conference Proceedings*, volume 2040, page 050021, 11 2018.

[24] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.

[25] Sajjad Haider. Clustering based anomalous transaction reporting. *Procedia Computer Science*, 3: 606–610, 12 2011.

[26] Sahand Hariri, Matias Kind, and Robert Brunner. Extended isolation forest with randomly oriented hyperplanes. *IEEE Transactions on Knowledge and Data Engineering*, PP:1–1, 10 2019.

[27] D. M. Hawkins. *Identification of outliers*. Monographs on applied probability and statistics. Chapman and Hall, 1980.

[28] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[29] Dongxu Huang, Dejun Mu, Libin Yang, and Xiaoyan Cai. Codetect: Financial fraud detection with anomaly feature detection. *IEEE Access*, 6:19161–19174, 03 2018.

[30] Vikas Jayasree and Siva Balan. Money laundering identification on banking data using probabilistic relational audit sequential pattern. *Asian Journal of Applied Sciences*, 8:173–184, 03 2015.

[31] Fabian Keller, Emmanuel Muller, and Klemens Bohm. Hics: High contrast subspaces for density-based outlier ranking. *Proceedings - International Conference on Data Engineering*, pages 1037–1048, 04 2012.

[32] Harmeet Khanuja and Dattatraya Adane. Monitor and detect suspicious transactions with database forensic analysis. *Journal of Database Management*, 29:28–50, 10 2018.

[33] Harmeet Kaur Khanuja and Dattatraya S. Adane. Detection of suspicious transactions with database forensics and theory of evidence. In *Security in Computing and Communications*, pages 419–430, Singapore, 2019. Springer Singapore.

[34] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452, 2008.

[35] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Outlier detection techniques. *Tutorial at KDD*, 10:1–76, 2010.

[36] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowledge and Information Systems*, 52(2): 341?378, 2017.

[37] Gleidson Sobreira Leite, Adriano Bessa Albuquerque, and Pl cido Rogerio Pinheiro. Application of technological solutions in the fight against money laundering a systematic literature review. *Applied Sciences*, 9:4800, 11 2019.

[38] Xurui Li, Xiang Cao, Xuetao Qiu, Jintao Zhao, and Jianbin Zheng. Intelligent anti-money laundering solution based upon novel community detection in massive transaction networks on spark. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 176–181, 08 2017.

[39] Yuhua Li, Dongsheng Duan, Guanghao Hu, and Zhengding Lu. Discovering hidden group in financial transaction network using hidden markov model and genetic algorithm. volume 5, pages 253–258, 01 2009.

[40] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08)*, pages 413–422, 2008.

[41] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery From Data - TKDD*, 6:1–39, 03 2012.

[42] Xuan Liu and Pengzhu Zhang. A scan statistics based suspicious transactions detection model for anti-money laundering (aml) in financial institutions. In *Proceedings - 2010 International Conference on Multimedia Communications, Mediacom 2010*, pages 210–213, 09 2010.

[43] Devendra Luna, Girish Palshikar, Manoj Apte, and Arnab Bhattacharya. Finding shell company accounts using anomaly detection. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 167–174, 01 2018.

[44] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[45] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

[46] M. Y. Meghanath, Deepak Pai, and Leman Akoglu. Conout: Contextual outlier detection with multiple contexts: Application to ad fraud. In *Machine Learning and Knowledge Discovery in Databases*, pages 139–156. Springer International Publishing, 01 2019.

[47] Barbora Micenkova, Raymond Ng, Xuan-Hong Dang, and Ira Assent. Explaining outliers by subspace separability. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 518–527, 12 2013.

[48] Krzysztof Michalak and Jerzy Korczak. Graph mining approach to suspicious transaction detection. In *2011 Federated Conference on Computer Science and Information Systems, FedCSIS 2011*, pages 69–75, 01 2011.

[49] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[50] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[51] Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

[52] United Nations Office on Drugs and Crime. Estimating illicit financial flows resulting from drug trafficking and other transnational organized crimes, 10 2011.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[54] Mark Pijnenburg and Wojtek Kowalczyk. Extending an anomaly detection benchmark with auto-encoders, isolation forests, and rbms. In *Information and Software Technologies*, pages 498–515. Springer International Publishing, 10 2019.

[55] Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. John Wiley Sons, Inc., New York, NY, United States, 1998.

[56] Anagha Rao and Kanchana V. Dynamic approach for detection of suspicious transactions in money laundering. *International Journal of Engineering & Technology*, 7:10, 07 2018.

[57] Shebuti Rayana. ODDS library, 2016. URL http://odds.cs.stonybrook.edu.

[58] Saleha Raza and Sajjad Haider. Suspicious activity reporting using dynamic bayesian networks. *Procedia Computer Science*, 3:987–991, 12 2011.

[59] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009.

[60] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you? : Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–101, 02 2016.

[61] SAS. How ai and machine learning are redefining anti-money laundering, 09 2019. URL https://communities.sas.com/t5/Fraud-AML-and-Security/How-AI-and-Machine-Learning-Are-Redefining-Anti-Money-Laundering/td-p/595031.

[62] Bernhard Sch lkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, volume 12, pages 582–588, Cambridge, MA, USA, 01 1999. MIT Press.

[63] Reza Soltani, Uyen Nguyen, Yang Yang, Mohammad Faghani, Alaa Yagoub, and Aijun An. A new algorithm for money laundering detection based on structural similarity. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, pages 1–7, 10 2016.

[64] Jun Tang and Jian Yin. Developing an intelligent data discriminating system of anti-money laundering based on svm. In *2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3453–3457, 09 2005.

[65] Bart H. Meijer Toby Sterling. Dutch bank ing fined $900 million for failing to spot money laundering, 09 2018. URL https://www.reuters.com/article/us-ing-groep-settlement-money-laundering/dutch-bank-ing-fined-900-million-for-failing-to-spot-money-laundering-idUSKCN1LK0PE.

[66] P. Umadevi and E. Divya. Money laundering detection using tfa system. In *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012)*, pages 1–8, 01 2012.

[67] Yan Yang, Bin Lian, Lian Li, Chen Chen, and Pu Li. Dbscan clustering algorithm applied to identify suspicious financial transactions. In *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 60–65, 10 2014.

[68] Cheng-wei Zhang and Yu-bo Wang. Research on application of distributed data mining in anti-money laundering monitoring system. In *2010 2nd International Conference on Advanced Computer Control*, volume 5, pages 133–135, 01 2010.

[69] Tianqing Zhu. An outlier detection model based on cross datasets comparison for financial surveillance. In *Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages 601–604. IEEE Computer Society, 01 2006.

[70] Arthur Zimek and Peter Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1280, 08 2018.

[71] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae ki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *ICLR*, 2018.

A
Code

In this section, the code that is needed for the explanation methods is provided along with a working example for better understanding of how they are applied. The following python packages are required for the code to work.

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
import shap
import matplotlib.pyplot as plt
import time
from sklearn.metrics import auc,  average_precision_score,  roc_curve
```

A.1. MI-Local-DIFFI

The code for the MI-Local-DIFFI explanations. The inputs required are (a) the tree estimators of the Isolation Forest (type: `sklearn` ExtraTreeRegressors), (b) the outliers that are supposed to be explained, (c) the original data set X and include_si is set to *True* to use the split interval weight version of MI-Local-DIFFI, but *False* otherwise. The output is then the feature importance explanations and the runtime of the algorithm.

```python
def MI_Local_DIFFI(estimators, outliers, data, standardize = False, subsampling =False):

    time_start = time.time() # start timer
    X = data.values
    min_pl = 1 # minimum path length
    max_pl = np.ceil(2 * (np.log(len(X)) +0.5772-1))
    path_length = np.zeros((len(outliers), len(estimators)))
    feature_importance = np.zeros((len(outliers), X.shape[1]))
    feature_occurrence = np.zeros((len(outliers), X.shape[1]))

    if(subsampling):
        sub_size = min(1000, len(X) - len(outliers))
        inl = np.array(list(set(list(range(len(X)))).difference(set(outliers))))
        s = np.random.choice(inl, sub_size)
        Xs = np.concatenate((X[outliers,:], X[s,:]), axis = 0)
    else:
        inl = np.array(list(set(list(range(len(X)))).difference(set(outliers))))
        Xs = np.concatenate((X[outliers,:], X[inl,:]), axis = 0)

    min_X = X.min(axis = 0) # the minimum feature values of each feature in the data set
    max_X = X.max(axis = 0) # the maximum feature values of each feature in the data set

    for i, tree in enumerate(estimators): # For each iTree in the Isolation Forest:
        n_samples_in_node = tree.decision_path(Xs).sum( axis = 0).A1 # Number of samples in each node
        for j, o in enumerate(outliers): # For each outlier:
            node_path = tree.decision_path(Xs[j, :].reshape(1, -1)).indices # Find the path of outlier o,
     in tree t_i
            path_length[j,i] = len(node_path) + average_path_length( n_samples_in_node[node_path[-1]].
     reshape(1, -1)) #calculate the path length
            feature_path =  tree.tree_.feature[node_path][:-1] # find the features in the outlier path
            threshold_path = tree.tree_.threshold[node_path][:-1] # find the splitting values in the
     outlier path

```

```python
            w_sp = 1 - np.nan_to_num((n_samples_in_node[node_path][1:] - 1 ) / np.maximum(1,(n_samples_in_node[node_path][:-1]  - 2) ))
            w_pl = np.maximum(0.1  ,np.minimum(1, 1 - (path_length[j,i] - min_pl)/(max_pl  - min_pl)))

            w_si = np.zeros(len(feature_path))
            min_feature_value = min_X.copy()
            max_feature_value = max_X.copy()
            for i_fp, fp in enumerate(feature_path):
                interval_length = max_feature_value[fp] - min_feature_value[fp]
                if Xs[j,fp] <= threshold_path[i_fp]: # outliers in left subinterval
                    si = (threshold_path[i_fp] - min_feature_value[fp])/interval_length
                    w_si[i_fp] = (3 - 2/(si + 1))/2
                    max_feature_value[fp] = threshold_path[i_fp]
                else:  # outliers in right subinterval
                    si = (max_feature_value[fp] - threshold_path[i_fp])/interval_length
                    w_si[i_fp] = (3 - 2/(si + 1))/2
                    min_feature_value[fp] = threshold_path[i_fp]


             # Add an importance weight for each feature that occurs in the outliers path
            for p in np.unique(feature_path):
                feature_occurrence[j,p] += 1
                pi = np.where(feature_path == p)[0]
                if(len(pi) > 1):
                    pi_m = np.argmax(w_sp[pi])
                    feature_importance[j, p] += float(w_pl * w_sp[pi][pi_m] * w_si[pi][pi_m])
                else:
                    feature_importance[j, p] += float(w_pl * np.max(w_sp[pi])* w_si[pi])

    feature_importance = np.nan_to_num(np.divide(feature_importance, feature_occurrence)) # divide by occurrence
    if(standardize):
        n_o = len(outliers)
        feature_importance = np.divide((feature_importance - np.amin(feature_importance, axis = 1).reshape(n_o, 1)),
                                      (np.amax(feature_importance, axis = 1) -
                                       np.amin(feature_importance, axis = 1)).reshape(n_o, 1)) # standardize for each outlier
    runtime = time.time() - time_start
    return feature_importance, runtime


def average_path_length(n_samples_leaf):
    if(n_samples_leaf > 2):
        average_path_length =  2.0 * (np.log(n_samples_leaf) + np.euler_gamma - 1)
    elif n_samples_leaf == 2:
        average_path_length = 1
    else :
        average_path_length = 0
    return average_path_length
```

A.2. TreeSHAP

The code for the TreeSHAP explanations. The inputs are (a) the fitted Isolation Forest classifier of `sklearn` type, (b) the outliers that are supposed to be explained and (c) the original data set X. The output is the feature importance explanations and the runtime of the algorithm.

```python
def TreeSHAP(classifier, outliers, data, standardize = False):
    start = time.time()
    explainer = shap.TreeExplainer(classifier)
    shap_values = explainer.shap_values(data.iloc[outliers,:])
    runtime = time.time() - start
    # normalize for each outlier:
    if(standardize):
        n_o = len(outliers)
        shap_values =  np.divide((-shap_values - np.amin(-shap_values, axis = 1).reshape(n_o, 1)),
                                      (np.amax(-shap_values, axis = 1) -
                                       np.amin(-shap_values, axis = 1)).reshape(n_o, 1))
    return shap_values, runtime
```

A.3. AOF

The code for the AOF with median replacement explanations. The inputs are (a) the fitted Isolation Forest classifier of `sklearn` type, (b) the outliers that are supposed to be explained and (c) the original data set X. The output is the feature importance explanations and the runtime of the algorithm.

```python
def AOF_median(classifier, outliers, data, scores, standardize = False):
    X = data.values
    start = time.time()
    scores_m_i = np.zeros(X.shape)
    n_o = len(outliers)
    for i in range(X.shape[1]):
        X_m_i = X.copy()
        X_m_i[:,i] = np.median(X_m_i[:,i]) * np.ones(X.shape[0])
        scores_m_i[:,i] = 0.5 - classifier.decision_function(X_m_i)
    scores_diff_m = -np.subtract(scores_m_i[outliers,:] , scores[outliers].reshape(n_o, 1))
    if(standardize):
        scores_diff_m = np.divide((scores_diff_m -
                                    np.amin(scores_diff_m, axis = 1).reshape(n_o, 1)),
                                       (np.amax(scores_diff_m, axis = 1) -
                                        np.amin(scores_diff_m, axis = 1)).reshape(n_o, 1))
    runtime = time.time() - start
    return scores_diff_m, runtime
```

A.4. Working Examples

To understand how the explanation methods are used, we provide a working example using a randomly generated gaussian data set X with outliers.

A.4.1. Example 1

We create ten outliers by changing the third feature to a random number on the interval $[2, 3]$ plus the maximum value of feature 3. The randomly generated data set with outliers is 3-dimensioal and is shown in Figure A.1. The code is shown below and it turns out that the standardized explanation scores all give the highest importance to feature 3.

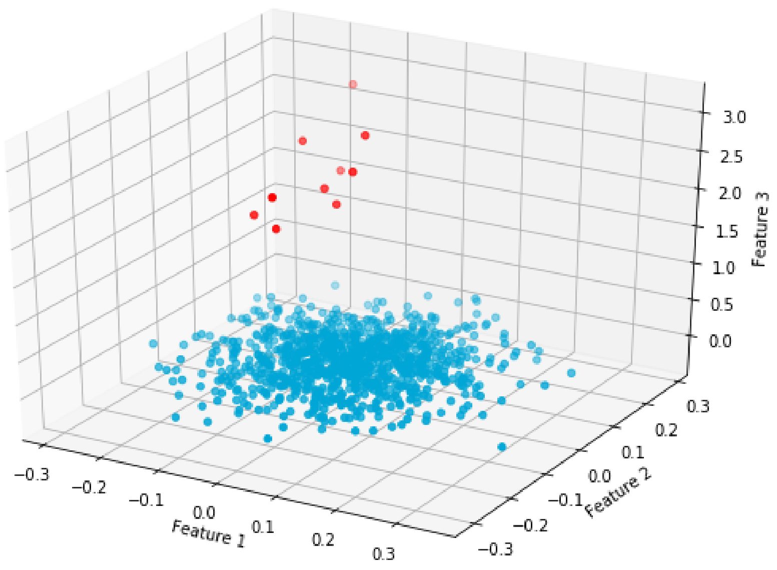


Figure A.1: Scatter plot of the randomly generated Gaussian data set. The outliers are colored red.

```
# Create a synthetic dataset with outliers where the outlying feature is feature 3
rnd = np.random.RandomState(30)
n = 1000
mu, sigma = 0, 0.1
x1 = rnd.normal(mu, sigma, n)
x2 =  rnd.normal(mu, sigma, n)
x3 = rnd.normal(mu, sigma, n)
outliers = rnd.randint(0,n,10)
inliers = np.array(list(set(list(range(0,n))).difference(set(outliers))))
x3[outliers] = np.max(x3) + 2 + rnd.random(10)
X = pd.DataFrame(np.transpose([x1, x2, x3]))
y_true = np.zeros(n)
y_true[outliers] = np.ones(len(outliers))

# Plot the synthetic dataset, outliers are colored red
```

```
fig = plt.figure(figsize =[10,7])

ax = fig.add_subplot(111, projection='3d')
ax.scatter(X.iloc[inliers, 0], X.iloc[inliers, 1], X.iloc[inliers, 2], color = (0, 166/255, 214/255),)
ax.scatter(X.iloc[outliers, 0], X.iloc[outliers, 1], X.iloc[outliers, 2], color = 'red')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')


# Run the Isolation Forest on the synthetic data set
clf = IsolationForest(random_state = 20)
clf.fit(X)
scores = 0.5 - clf.decision_function(X) # to get the scores on a [0,1] interval
IF_estimators = clf.estimators_ # extract the tree estimators from the isolation forest

# Evaluate the performance of the Isolation Forest (AUC, AUPRC)
fpr, tpr, thresholds = roc_curve(y_true, scores, pos_label=1)
auc_ = auc(fpr, tpr)
print(auc_)
auprc_ =  average_precision_score(y_true, scores)
print(auprc_)


# DIFFI-local
explanation_DIFFI_local, runtime_DIFFI_local = MI_Local_DIFFI(IF_estimators, outliers, X)
# TreeSHAP
explanation_TreeSHAP, runtime_TreeSHAP = TreeSHAP(clf, outliers, X)
# AOF-median replacement
explanation_AOF_median, runtime_AOF_median = AOF_median(clf, outliers, X, scores)
```

A.4.2. Example 2

We create ten outliers by changing (a) the third feature to a random number on the interval $[2,3]$ plus the maximum value of feature 3 and (b) the first feature to a random number on $[-3,-2]$ plus the minimum value of feature 1. The randomly generated data set with outliers is 3-dimensioal and is shown in Figure A.2. The code is shown below and it turns out that the standardized explanation scores all give the highest importance to feature 1 and 3.

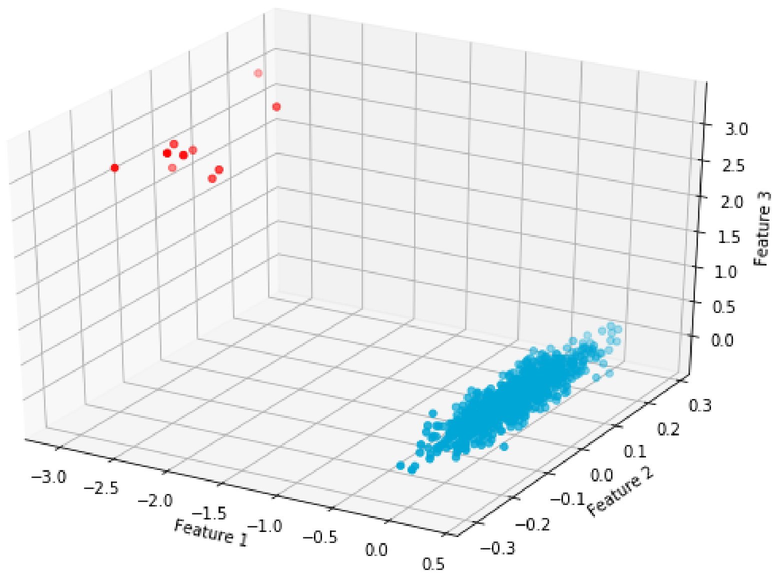


Figure A.2: Scatter plot of the randomly generated Gaussian data set. The outliers are colored red.

```
# Create a synthetic dataset with outliers where the outlying feature is feature 1 and 3

rnd = np.random.RandomState(30)
n = 1000
mu, sigma = 0, 0.1
x1 = rnd.normal(mu, sigma, n)
```

```
x2 =  rnd.normal(mu, sigma, n)
x3 = rnd.normal(mu, sigma, n)
outliers = rnd.randint(0,n,10)
inliers = np.array(list(set(list(range(0,n))).difference(set(outliers))))
x1[outliers] = np.min(x1) - 2 - rnd.random(10)
x3[outliers] = np.max(x3) + 2 + rnd.random(10)
X = pd.DataFrame(np.transpose([x1, x2, x3]))
y_true = np.zeros(n)
y_true[outliers] = np.ones(len(outliers))

# Plot the synthetic dataset, outliers are colored red
fig = plt.figure(figsize =[10,7])

ax = fig.add_subplot(111, projection='3d')
ax.scatter(X.iloc[inliers, 0], X.iloc[inliers, 1], X.iloc[inliers, 2], color = (0, 166/255, 214/255),)
ax.scatter(X.iloc[outliers, 0], X.iloc[outliers, 1], X.iloc[outliers, 2], color = 'red')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')


# Run the Isolation Forest on the synthetic data set
clf = IsolationForest(random_state = 20)
clf.fit(X)
scores = 0.5 - clf.decision_function(X) # to get the scores on a [0,1] interval
IF_estimators = clf.estimators_ # extract the tree estimators from the isolation forest

# Evaluate the performance of the Isolation Forest (AUC, AUPRC)
fpr, tpr, thresholds = roc_curve(y_true, scores, pos_label=1)
auc_ = auc(fpr, tpr)
print(auc_)
auprc_ =  average_precision_score(y_true, scores)
print(auprc_)

# DIFFI-local
explanation_DIFFI_local, runtime_DIFFI_local = MI_Local_DIFFI(IF_estimators, outliers, X)
# TreeSHAP
explanation_TreeSHAP, runtime_TreeSHAP = TreeSHAP(clf, outliers, X)
# AOF-median replacement
explanation_AOF_median, runtime_AOF_median = AOF_median(clf, outliers, X, scores)
```

B

TreeSHAP Method in Detail

The TreeSHAP method and SHAP values are based on so-called Shapley values from coalitional game thoery. In this appendix we introduce the theory behind the TreeSHAP method. We first introduce the basic concepts of Shapley values and the four axioms that the Shapley values satisfy. We then show how these four axioms can be reduced to only two axioms in the case of prediction models f. Then we introduce the SHAP values which are defined by the relationship of the $f(x, S) = \mathbb{E}[f(x)|x_S]$. We then explain how the Shapley axioms translate to three desirable properties of local explanation methods. Finally, we explain how the TreeSHAP algorithm can be used to calculate the SHAP values in the case f is a tree-based model in a very efficient way.

B.1. Shapley Values

To define Shapley values, we first introduce the notion of games in coalitional form. The following definitions are taken from [22].

Definition B.1.1. Let $d \geq 2$ denote the number of players in the game and D denote the set of players, $D = \{1, 2, ..., n\}$. A *coalition*, S, is defined to be a subset of D, $S \subset D$, and the set of all coalitions is denoted by 2^D.

Definition B.1.2. The *coalitional* form of an d-person game is given by the pair (N, v), where $D = \{1, 2, ..., d\}$ is the set of players and v is a real-valued function, called the *characteristic function* of the game, defined on the set, 2^D, of all coalitions, and satisfying

(i) $v(\emptyset) = 0$, and

(ii) if S and T are disjoint coalitions, then $v(S) + v(T) \leq v(S \cup T)$.

Then we define the *value function* of the characteristic function.

Definition B.1.3. A *value function*, ϕ, is a function that assigns to each possible characteristic function of an d-person game, an d-tuple, $\phi(v) = (\phi_1(v), \phi_2(v), ...\phi_d(v))$ where $\phi_j(v) \in \mathbb{R}$ for each v and $j \in \{1, 2, ..., d\}$.

Intuitively, $\phi_j(v)$ should represent the value or worth of player j in the game with characteristic function v. The worth of a player should fulfil certain notions of fairness. These fairness notions are represented by four axioms called the Shapley Axioms and it can be shown that there is exactly one value function that satisfies these four axioms, namely the Shapley values. We present these four axioms in the game setting of model predictions for a prediction model f and an individual observation x drawn from d dimensional data set X with n observations. The Shapley values ϕ_j corresponding to the model f and observation x thus represent the credit that is assigned to each of the features $j \in \{1, ..., d\}$ when considering the prediction outcome $f(x)$. We let $f(x, S)$ represent the prediction outcome of model f for observation x when only considering the features in the subset S. The exact way in how such a prediction is obtained will be described later. The axioms are the following,

Axiom 1 (Shapley Axioms)**.**

I **Efficiency**

$$f(x) = \sum_{j=1}^{d} \phi_j$$

Interpretation: The total credit of each feature should add up to the prediction outcome.

II **Symmetry** If j and k are such that $f(x, S \cup \{j\}) = f(x, S \cup \{k\})$ for every coalition S not containing j and k, then $\phi_j = \phi_k$.
Interpretation: The Shapley values of two features j and k should be the same if their contribution to all coalitions $S\ \ \{j,k\}$ is the same.

III **Dummy Axiom** If j is such that $f(x, S) = f(x, S \cup \{j\})$ for every coalition S not containing j, then $\phi_j = 0$.
Interpretation: If feature j neither increases nor decreases the prediction value of any coalition that it joins, then it should have a value of 0.

IV **Additivity** For any two models f and f', it should hold that $\phi(f(x)+f'(x)) = \phi(f(x)))+\phi(f'(x))$.
Interpretation: The effect a feature has on the sum of two models should be equal to sum of the effects that the features has on the two models individually.

It can be shown that there exists a unique function ϕ that satisfies the Shapley Axioms and there is a closed form formula that can be used to calculate the Shapley value of a coalition game. For the proof of the following two theorems, we point to chapter IV.3 in Ferguson's book [22].

Theorem B.1.1. *There exists a unique function ϕ satisfying the Shapley Axioms.*

Theorem B.1.2. *The Shapley value is given by $\phi = (\phi_1, \phi_2, ...\phi_d)$, where for $j = 1, 2, ..., d$,*

$$\phi_j = \sum_{S \subset D, j \in S} \frac{(|S|-1)!(d-|S|)!}{d!}[f(x, S) - f(x, S\ \ \{j\})]. \tag{B.1}$$

Using the supplementary material of the SHAP paper from Lundberg [44], we further show that these four axioms can be simplified to only two axioms. In 1985, Young proved that the additivity and dummy axioms can be eliminated using a monotonicity axiom.

Monotonicity For any two prediction models f and f', if for all feature subsets S that do not contain j it holds that

$$f(x, S \cup \{j\}) - f(x, S) \geq f'(x, S \cup \{j\}) - f'(x, S),$$

then $\phi_j(f(x)) \geq \phi_j(f'(x))$.
Interpretation: If the difference between the prediction outcome on coalition $S \cup \{j\}$ and S is always greater for some model f than some model f', then the Shapley value of model f corresponding to feature j should also be greater than that of f'.

Now, we show that the symmetry axiom is implied by the monotonicity axiom. Assume that f' is the same as f except the inputs j and k are swapped. This means that for all S that do not contain j or k it holds that

$$f'(x, S \cup \{j\}) = f(x, S \cup \{k\}) \quad \text{and} \quad f'(x, S) = f(x, S). \tag{B.2}$$

Furthermore, for a subset S that does not contain k, the following is guaranteed to hold

$$f'(x, S\ \ \{k\} \cup \{j,k\}) = f'(x, S\ \ \{k\} \cup \{k,j\}).$$

Let us state the monotonicity axiom again, for all S that do not contain j,

$$f(x, S \cup \{j\}) - f(x, S) \geq f'(x, S \cup \{j\}) - f'(x, S) \Rightarrow \phi_j(f(x)) \geq \phi_j(f'(x)). \tag{B.3}$$

Using this axiom, equation B.2 and ignoring the terms that contain k we get,

$$\text{for all } S \text{ that do not contain } j, k\text{: } f(x, S \cup \{j\}) \geq f(x, S \cup \{k\}) \Rightarrow \phi_j(f(x)) \geq \phi_j(f'(x)), \tag{B.4}$$

Repeating this but swapping j and k in equations B.3 and B.4, we get

$$\text{for all } S \text{ that do not contain } j, k\text{: } f(x, S \cup \{k\}) \geq f(x, S \cup \{j\}) \Rightarrow \phi_k(f(x)) \geq \phi_k(f'(x)), \tag{B.5}$$

Finally, by combining equations B.4 and B.5 and observing that by definition of f', $\phi_j(f, x) = \phi_k(f', x)$ and $\phi_k(f, x) = \phi_j(f', x)$, we get the symmetry axiom,

$$\text{for all } S \text{ that do not contain } j, k\text{: } f(x, S \cup \{j\}) = f(x, S \cup \{k\}) \Rightarrow \phi_j(f(x)) = \phi_k(f(x)). \tag{B.6}$$

So we have now shown that we only need the efficiency and monotonicity axioms to uniquely constrain ourselves to using the Shapley values.

B.2. SHAP values

The SHAP values are defined as *the Shapley values of the conditional expectation of the original prediction model* f, that is $f(x, S) = \mathbb{E}[f(x)|x_S]$. To show the properties of the SHAP values, we begin by introducing the class of additive feature attribution methods which are methods that explain the model outcome as sum of real values that correspond to the different input features.

Definition B.2.1. *Additive feature attribution methods* have an explanation model that is a linear function of binary variables,

$$g(z') = \phi_0 + \sum_{j=1}^{d} \phi_j z'_j, \tag{B.7}$$

where $z' \in \{0, 1\}^d$ are the simplified features, d is the number of simplified input features and $\phi_j \in \mathbb{R}$.

Here, the z'_j variables represent a feature being observed ($z'_j = 1$) or unknown ($z'_j = 0$) and the values ϕ_j are the feature attributions. We now introduce three properties that are desirable for an explanation model and show that the attribution values ϕ_j that satisfies these three properties are the Shapley values.

Property 1 (Local accuracy)

$$f(x) = \phi_0(f) + \sum_{j=1}^{d} \phi_j(f, x) \tag{B.8}$$

Interpretation: The sum of the feature attributions $\phi_j(f, x)$ *should match the original model output* $f(x)$ *where* $\phi_0(f) = \mathbb{E}[f(X)]$

Property 2 (Consistency) For any two models f and f', if

$$f'_x(S) - f'_x(S \quad j) \geq f_x(S) - f_x(S \quad j) \tag{B.9}$$

for all subsets of features $S \subset D$, then $\phi_j(f', x) \geq \phi_j(f, x)$.
Interpretation: If a model changes so that some simplified input's contribution either increases or stays the same regardless of the other inputs, then that input's impact should not decrease.

Property 3 (Missingness) If

$$f_x(S \cup i) = f_x(S) \tag{B.10}$$

for all subsets of features $S \subset D$, then $\phi_i(f, x) = 0$.
Interpretation: Features that have no effect on the prediction output $f(x)$ should have no assigned impact.

We showed in the previous section that the Shapley values are the only values that satisfy Property 1 and Property 3 where in this case Property 1 corresponds to the Efficiency Axiom and Property 3 corresponds to the Monotonicity Axiom. Furthermore, Property 2 is required to adapt the Shapley proof to the class of additive feature attribution methods. In order to evaluate the effect of missing features on the prediction model f, we must define some mapping. In the case of SHAP values, the mapping is defined as $f(x, S) = \mathbb{E}[f(x)|x_S]$. Thus the SHAP values can be computed as

$$\bar{\phi}_j(f, x) = \sum_{S \subset D, j \in S} \frac{|S|!(d - |S| - 1)!}{d!} \Big(\mathbb{E}[f(x)|x_S] - \mathbb{E}[f(x)|x_{S\ \{j\}}]\Big) \tag{B.11}$$

and they fulfill the three properties of local accuracy, consistency and missingness due to their direct connection with the Shapley axioms.

B.3. TreeSHAP

In order to calculate the SHAP feature importance values, $\mathbb{E}[f(x)|x_S]$ must be estimated for each subset S of $N = \{1, 2, ..., n\}$, a total of 2^n possibilities. For a tree-based model f, the conditional expectation $\mathbb{E}[f(x)|x_S]$ can be estimated in $O(TL)$ time using algorithm 10, where T is the number of trees in the model and L is the maximum number of leaves in any tree [45]. Thus, the total computational complexity is $O(TL2^n)$. In algorithm 10, the tree is notated by the v - the nodes and their corresponding prediction values, l - the left children of v, r - the right children of v, a - the split values of v, q - the number of data points in v and F - the features used for splitting in each node of v.

Algorithm 10 Estimating $\mathbb{E}[f(x)|x_S]$

```
procedure ExpectedValue($x$, $S$, tree = $\{v, l, r, a, q, F\}$)
    procedure G($j$,$w$)
        if $v_j \neq$ internal then
            return $w \cdot v_j$
        else
            if $d_j \in S$ then
                return $G(l_j, w)$ if $x_{F_j} \leq a_j$ else $G(r_j, w)$
            else
                return $G(l_j, wq_{l_j}/q_j) + G(r_j, wq_{r_j}/q_j)$
            end if
        end if
    end procedure
    return $G(1, 1)$
end procedure
```

However, [45] introduces a method to calculate the values $\mathbb{E}[f(x)|x_S]$ in $O(TLD^2)$ time, where D is the maximum depth of the trees. This method reduces the computation time from being exponential in the number of features to being second order in the trees depth. For the case of multiple feature, this is a major improvement. This algorithm is shown in [45] and implemented in the python library *shap* as `TreeExplainer`.