Isolation-based Anomaly Detection Algorithms for Distributed Data Streams

Visser, Thijs

Award date:
2021

Isolation-based Anomaly Detection Algorithms for Distributed Data Streams

A thesis presented by

Thijs Visser

to

the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Computer Science and Engineering

TU Eindhoven

June 2021

Supervisor:
dr. rer. nat. Morteza Monemizadeh

Defense committee:
prof. dr. Mark de Berg
dr. rer. nat. Morteza Monemizadeh
dr. Odysseas Papapetrou

Abstract

Anomaly detection is a classical topic in machine learning and data mining, with a wide range of applications. As the amount of information shared online grows rapidly, the need for parallelizable anomaly detection algorithms increases. In this thesis, we propose the *Random Shift Forest (RSF)* algorithm to detect anomalies. RSF is an unsupervised anomaly detection algorithm with a low time complexity. It is an ensemble method that uses partitioning to isolate anomalies. To evaluate RSF, we use it on synthetic and real datasets. Compared to a state-of-the-art algorithm, RSF performs favourably in terms of ROC-AUC. The straightforward construction of RSF makes it amenable to big data models. In particular, we show that RSF can be implemented in the distributed and the streaming model using sparse recovery techniques.

Acknowledgments

First of all, I would like to thank my supervisor, Morteza Monemizadeh, for his time and effort. Your guidance, help, and enthusiasm were amazing, and I have learned a lot from you. I would also like to thank prof. dr. Mark de Berg and dr. Odysseas Papapetrou for being a part of my defense committee, I feel honored that you have read my thesis. Furthermore, I want to thank my girlfriend Fleur for being by my side at all times. And James, having you (literally) at my side made writing this thesis a lot more fun. I want to thank my parents for their encouragement and support throughout my studies. I am grateful to have a sister, family, and friends that always believe in me. And finally, since I was told you have to thank your sponsors here, I want to thank Nienke for all the free coffee.

Contents

List of Figures

List of Tables

1 Introduction

In a world where the amount of information shared digitally grows rapidly, it becomes increasingly important to develop ways to cope with massive data. One of the core concepts in analyzing big data is *anomaly detection* [11], also known as *outlier detection* [27]. In the field of anomaly detection, the goal is to find parts of the data that are in some sense different from the majority of the data. The idea behind this is that if some instances differ from most other instances, then they deserve additional attention. There are many applications for anomaly detection, including:

- Detecting irregularities in the power production of a system of solar panels for predictive maintenance [13].
- Finding and removing noise from a dataset before performing statistical analysis, such as linear regression, on the data to improve the validity of the results [42].
- Finding a rarity [12] in a big data stream for further investigation in data mining (novelty detection) [18].
- Removing outliers from a training dataset before training a neural network on it to improve classification accuracy in machine learning [16].
- Detecting anomalies in network traffic to find data that was sent by an attacking party (intrusion detection) [40].
- Finding a part of an IoT system that is faulty by analyzing data streams [34].
- Detecting potential cases of credit card fraud in online banking [8].
- Finding and removing pieces of fruit that show abnormalities, such as rotting, within a fruit sorting machine [31].

Unfortunately, defining an anomaly exactly seems to be a difficult task. A commonly cited definition of an anomaly by Hawkins, author of a book on outlier detection [25], is *an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.* This definition reveals the motivation behind detecting anomalies: we would like to

find those instances that have been generated by other sources than the expected mechanism.

One may think that an anomaly is always different from the rest of the points, or that rare points are always anomalies. However, this is not always the case, as is highlighted in the following two examples:

- **Attacks in network systems:** in the case of a network setting an attack is often identified because of detecting abrupt bursts of activity [47]. The observations from the attack are detected as anomalous because they are unexpectedly similar to each other.

- **Imbalanced datasets:** if in a stream 97% of all data belongs to a cluster A, 1% belongs to a cluster B and the remaining 2% is random noise. Then, whether we want to say that data in cluster B is anomalous depends on the application. As an example, in the case of monitoring a network and attack identification, cluster B could be the data sent by the attacker and should therefore likely be marked as an anomalous cluster. If the data is from some population where we know about 1% of people satisfy a certain condition then this clustering is expected and there is no reason to remove or analyze the data.

These examples show that the definition of an anomaly will always be context-specific, so it is unlikely that there will ever be a single algorithm that finds all anomalies in every setting. It is up to the expert to select the right algorithm(s) and parameters to find anomalies of interest.

Anomaly detection in low- and high-dimensional Euclidean spaces. In this thesis, we consider the anomaly detection task in low- as well as high-dimensional Euclidean spaces. Finding anomalies for small datasets in low dimensional spaces such as 1, 2, and 3 dimensions is often a simple task for a human. These datasets can be visualized, such that we can distinguish anomalies and normal points. The anomaly detection task is much harder once the underlying space is relatively high, mainly because it is not possible to plot the data. For high-dimensional space, we use statistical metrics such as *ROC-AUC* and *contamination* to measure the correctness of the classification behaviour of an algorithm. We will explain these metrics in detail in the next chapter. The anomaly detection problems becomes harder for big data models, such as streaming and distributed models. In particular, in the streaming model, given

a stream of data that is constantly changing, we would like to detect anomalies in real time. In the distributed model, data is spread among a pool of machines and local anomalies on one machine may not be the global anomalies of the union of data. The anomaly detection becomes even more complicated if the data lies in high-dimensional Euclidean spaces. Often anomaly detection algorithms have running times that are exponentially dependent on the number of dimensions. Unfortunately, even the running time that is polynomially dependent on the number dimensions may not be feasible if the number of dimensions is very high. Another issue in high dimensional Euclidean spaces is a phenomenon known as *the curse of dimensionality* [48]. The curse of dimensionality can be described as a series of issues when dealing with highly dimensional data. The main issue in the context of anomaly detection is that in high dimensional spaces, points tend to have relatively equal distances from each other. Therefore it becomes harder and harder to find meaningful patterns in the data that can be used to create a classification model for normal and anomalous points. New methods have been introduced in the past decades to help deal with anomaly detection in big data.

Broadly speaking, anomaly detection algorithms can be categorized into two classes as follows:

- **Statistical algorithms:** in statistical methods, also known as parametric methods [7], we assume there is a certain known underlying distribution from which normal points are generated. A point is classified as an anomaly based on its deviation from this model.
- **Non-statistical algorithms:** other anomaly detection algorithms do not impose such an assumption on data, but instead use metrics such as distance or (local) density to determine the anomalous instances.

Researchers have taken different approaches to the design of anomaly detection algorithms. Isolation Forest (iForest) [33] is an anomaly detection algorithm that introduced a new way of finding anomalies. It uses recursive partitioning of the data to build binary trees. The average number of partitioning iterations it takes to isolate a point is used as a measure of anomalousness by this algorithm. The focus of this thesis is to develop an algorithm that uses isolation to detect anomalies, while also being suitable to use in a distributed streaming setting. We consider iForest as a starting point due to its practical success reported recently in [1] and it is one of the state-of-the-art techniques for general-purpose

anomaly detection.

1.1 Problem statement

Isolation forest is currently one of the best anomaly detection algorithms when it comes to accuracy [9]. However, it is not suitable to use in a distributed streaming setting, as we will explain later. Since big data often arrives in real-time and on multiple machines, it is important to have algorithms that can be used or adapted to work under such circumstances. Therefore the main research question of this thesis is:

Can we develop an anomaly detection algorithm that performs better than Isolation Forest and is suitable for streaming- and distributed settings?

Efforts to use Isolation Forest in a distributed way have been made in the past, for example with the Isolation Forest variant *Extended Isolation Forest* [24]. However, Extended Isolation Forest is known to perform well for simple toy datasets rather than big datasets in high dimensional spaces. Moreover, the authors in [24] do not give any proof of why their approach should or should not work for real big datasets.

1.2 Outline

This chapter introduced the topic of our research and provided a high-level overview of the relevant topics related to the research question. In chapter 2 we provide the necessary background information for the scope of this thesis. In chapter 3 the algorithms developed to solve the research question are introduced and detailed with both intuitive explanations and pseudocode. In chapter 4 we present both quantitative and qualitative comparisons of our algorithms and widely used anomaly detection algorithms. Furthermore, this chapter contains results relating to the running time of our main algorithm. In chapter 5 we discuss the way our algorithms can be used in a streaming and distributed setting. Finally, chapter 6 consists of limitations, ideas for future research, and the conclusion.

2 | Background

In this chapter, we discuss the related work, background, and definitions used in this thesis. First, we discuss the definition of anomalies in detail. Next, we give an overview of the types of existing anomaly detection algorithms, with examples for each type. Then, we focus on Isolation Forest, which is the most important algorithm related to our research. We explain this algorithm in detail, and we highlight its strengths and weaknesses. After all, the aim of our research is to keep those strengths while using different techniques to address the shortcomings as much as possible.

2.1 Anomalies

Before thinking about a way to efficiently detect anomalies, it is important to understand how we can define an anomaly first. We provided one definition of anomalies in the introduction, but more characterizations have been used over the years, including:

- *An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs* [39].
- *An observation in a dataset that appears to be inconsistent with the remainder of that set of data* [29].

Indeed, it is clear that a formal definition of an anomaly does not exist. We can, however, argue about different types of anomalies. Whether a type of anomaly is then expected to occur in a certain context can be identified by a domain expert. There are at least the following three distinct types of anomalies according to [20]:

- **Point anomalies:** these anomalous observations differ significantly from the vast majority of the rest of the data.
- **Collective anomalies:** anomalies that come in bursts or groups. This type of anomaly is particularly common in network intrusion detection. Individually speaking, the observations are not rare, but a group of them differs from the majority of data.

- **Contextual anomalies:** anomalies that do not differ from the rest of the data, unless some context is taken into account. A case of a type of context that matters in certain situations is time. For example, it can be perfectly normal for the sun to shine for 4.5 hours on a day in the Netherlands, but if this happens in January (when it shines on average 1.5 hours per day) that should be considered an interesting anomaly. The anomalies are referred to as *contextual anomalies*.

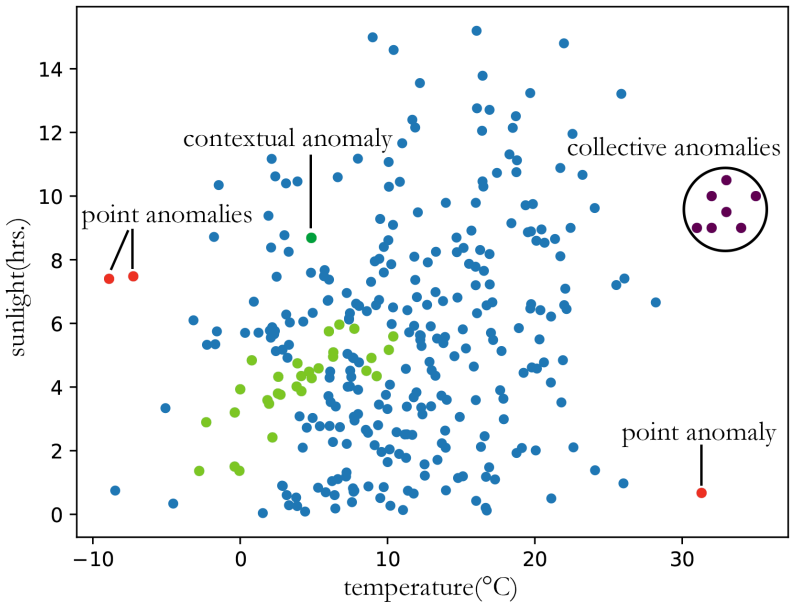


Figure 1: Types of anomalies in a synthetic dataset

Figure 1 shows examples of all of these types in a single dataset. This figure contains a synthetic dataset showing a possible combination of the temperature (x-axis) and hours of sun (y-axis) for every day of a year. The red points are examples of point anomalies: the points on the left have exceptionally many hours of sun, considering the low temperature on those two days. The red point on the right was a very hot day, with a surprisingly little amount of hours of sunshine. The purple points are a collective anomaly: if we consider the full data, a single point from these purple points is not unusual because there are six other points with almost the same values in both dimensions. However,

if one was to consider these purple points as a single observation they would stand out from the entire data. The seven points highlighted with an enclosing circle would be interesting to observe because they may be from a single week, and this week could then be marked as the only heatwave of that year. The dark green point is an example of a contextual anomaly: all points in green are from January, but this one dark green point has an exceptional amount of sun compared to the others. However, if we compare the dark green point to the full data, it is not an observation that stands out. It is highly unlikely that an anomaly detection algorithm would find this contextual anomaly using just the two dimensions in the data. However, if we would include the month (or week/day) as an extra numerical dimension (e.g. January = 1, February = 2, etc...) then it would become possible to detect the contextual anomalies as well. It is often the case that contextual anomalies can be detected by finding a way to include the context as an input dimension.

In the remainder of this thesis, we focus mainly on *point anomalies*, but also discuss ways to detect the other types of anomalies with the same algorithm by making use of its parameters.

2.2 Anomaly detection algorithms

We are in particular interested in general-purpose, *unsupervised* anomaly detection algorithms. Unsupervised anomaly detection algorithms are algorithms that are trained using unlabeled data. In anomaly detection, labeled data means that a point is marked as an anomaly or a normal point. The other types of algorithms are *supervised* and *semi-supervised* learning algorithms. In *supervised* learning, all of the data used to train a model is labeled. In *semi-supervised* learning, a small portion of the data is labeled. In the experiments in chapter 5, labeled data is used to calculate ROC-AUC scores. However, these labels are not passed to the algorithms, so the learning is still unsupervised. The reason for this research to focus on unsupervised anomaly detection is that labeled data is hard to come by [11]. Datasets have to be labeled by a human expert who individually inspects all instances to conclude whether an observation is anomalous or not. This process is labour intensive, especially in big data. Besides this difficulty, data can change over time and anomalies are dynamic, so supervised anomaly detection algorithms are simply not reasonable options for big data analysis in practice.

Over the years, many anomaly detection algorithms have been developed that can roughly be split into four main categories:

- Statistical algorithms
- Distance-based algorithms
- Density-based algorithms
- Randomized algorithms

The next paragraphs describe these four categories using examples of the most well-known algorithms in each of the categories.

Statistical algorithms. Statistical algorithms are characterized by the fact that they assume that the underlying model in the data is roughly known. Based on this model, it is possible to make estimations on how likely it is that an observation came from this model. The obvious advantage is that if the assumption of the underlying model holds, the technique can be both accurate and unsupervised. The main shortcoming of these algorithms is that they are specialized to certain supposed underlying models and generalize poorly to other settings. The question when using these algorithms is therefore usually: *is there something wrong with the anomaly or is there something wrong with the model choice?* [49]. An example of a statistical algorithm is fast-MCD [44]. Another statistical algorithm that is used often is One-Class Support Vector Machine (OCSVM) [4]. An issue with this algorithm is that it is sensitive to the inclusion of anomalies in the training phase [35]. More examples of statistical methods are discussed in [45].

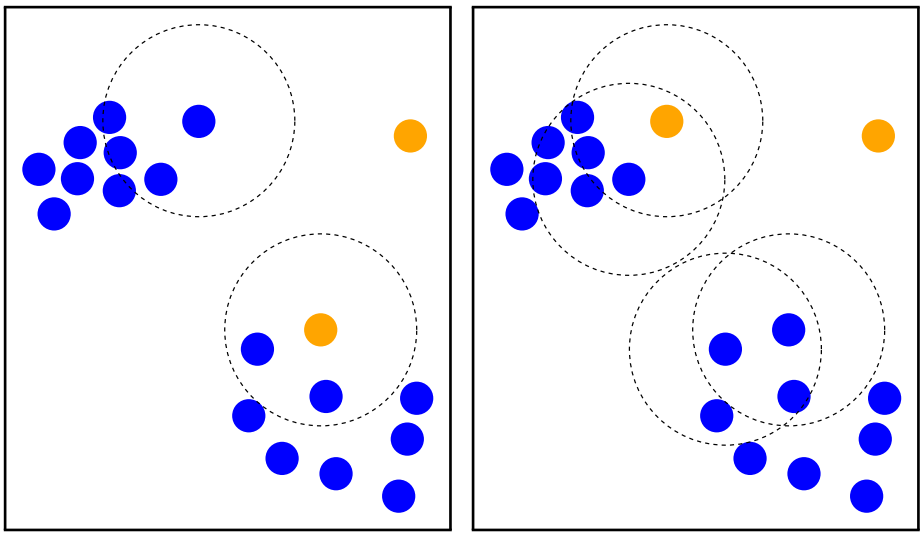
Distance-based algorithms. Distance-based algorithms use the distance of an instance to a subset of the other instances as a measure of anomalousness. Some of the most widely used distance-based anomaly detection algorithms are k-Nearest Neighbour approaches. These algorithms typically involve calculating a distance matrix, which has time complexity $\mathcal{O}(n^2)$. A weakness of these approaches is that they have difficulty handling data with many dimensions, as the distance between points becomes less effective at differentiating between normal and anomalous instances [11]. Furthermore, computing all pairwise distances in large, highly dimensional datasets is expensive. Some of the most cited [2] nearest neighbour approaches are LOF [10], COF [46], (a)LOCI [37],

INFLO [28] and LoOP [32]. These algorithms typically have a running time of $\mathcal{O}(n^2)$ or $\mathcal{O}(n^3)$ [2]. Therefore, it is not feasible to use these algorithms on big datasets. LOF, for example, can take over $10,000$ seconds (roughly 3 hours) to run on a dataset of over $11,000$ points [33].

Density-based algorithms. Density-based algorithms use the density of an area to compare observations. These algorithms can either look at the globally dense areas or the locally dense areas. In the former case, those points that are from the least dense areas in the entire data are marked as the anomalous points. In the latter case, those points that are from an area that is less dense than that of its neighbours are seen as the most anomalous. The practical difference is that the algorithms that focus on locally sparse points can detect anomalies of clusters with different densities, that would otherwise be classified as normal because their local area is dense in points. Figure 2 illustrates this difference. In figure 2a, the anomalies (in orange) are the points with the lowest amount of other points in its neighbourhood (dashed circles). In figure 2b, one of the anomalies is the same as in the previous case because it has no points in its neighbourhood. However, the other anomaly in this case is the point with 3 points in its neighbourhood because this number deviates more from the other points in its neighbourhood compared to the points in the other cluster (those all have 2 or 3 points in their local area). One of the most used density-based algorithms is DBSCAN [17]. This algorithm calculates, for each point, how many points are in its local neighborhood. If it has many points in its neighborhood the point is called a *core instance*. Core instances and the points in the neighborhoods of the core instances are marked as the normal points, all other points are considered anomalies.

Randomized algorithms. Randomized algorithms do not calculate absolute densities or distances like the previous two methods. Instead, they rely on partitioning the data to obtain a model that can classify anomalies. These partitions can conveniently be stored in the form of a tree to optimize the running time to $\mathcal{O}(n \log n)$. Isolation Forest [33] and Robust Random Cut Forest [22] are examples of random forest algorithms for anomaly detection.

Of course, distance-based-, density-based-, and randomized anomaly detection algorithms are somewhat related: if a point is close to many points it generally resides in a more dense area of the feature spaces and this point will



(a) Global anomalies (b) Local anomalies

Figure 2: The difference between global- and local density-based anomalies.

usually then be harder to isolate for an algorithm like Isolation Forest. However, we will see later on in this thesis that in practice, results of different methods can vary significantly.

2.3 Isolation Forest

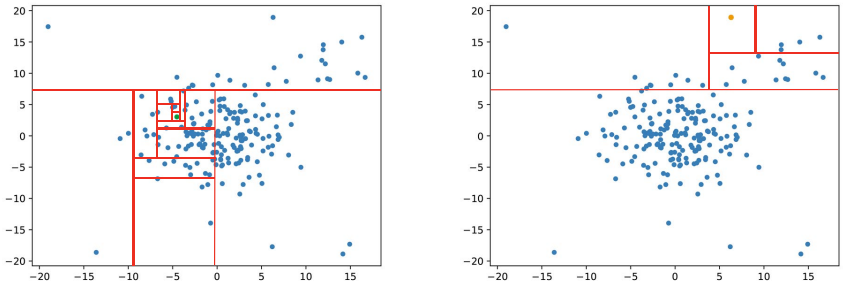
The most important anomaly detection algorithm in the context of this thesis is Isolation Forest, as our algorithm relies on a similar principle to detect anomalies. Therefore, we will explain this algorithm in more detail. Liu et al. [33] proposed an anomaly detection algorithm named Isolation Forest (iForest) in 2008. As its name suggests, this algorithm finds anomalies based on how *isolated* a point is. Isolation in this context is defined as *how easy it is to separate an instance from the rest of the instances*, using a separation process detailed below. In the original iForest paper, isolation is argued to be a good indication of anomalies under the following two assumptions about anomalies:

- Anomalies are rare: compared to normal instances, anomalies are a minority and they do not occur in large clusters.
- Anomalies are different: the attributes of an anomaly differ significantly from those of normal instances.

If these assumptions about the anomalies are true, then isolating an anomaly should be easier (i.e., it takes fewer steps) than isolating a normal point.

iForest construction. The iForest algorithm works by creating an ensemble of trees called *Isolation Trees* (iTrees) that are created by partitioning the data recursively. The authors of iForest [33] showed that they can use subsamples to generate iTrees that act as sketches of the full data. A random dimension is chosen and the data is split at a random splitting point, between a minimum and a maximum value. Because anomalies are assumed to be rare and well-separated, these points are more likely to become isolated (i.e., the only inhabitant of their part) sooner than normal points. The information about the partition is stored in the form of an iTree, where internal nodes are empty and every point is stored in an outer node (leaf). When the trees have been constructed, an anomaly score is calculated for every point. The anomaly score of a point in the iTree is based on the path length from the root of the tree to the leaf containing the point. Having a lower average path length corresponds to the point receiving a higher anomaly score. Figure 3 shows how iForest isolates points and illustrates why anomalous points are usually isolated more quickly than normal points. The normal point that we isolate in 3a is surrounded by many points from the large cluster it is a part of. Therefore, after any random split, the point is likely still in the same part of the partitioning as many other points from the cluster. It takes 13 random cuts to isolate this normal point. The anomalous point that is marked as orange in figure 3b is far from most other points in at least one of the dimensions. Therefore, a split in that dimension is likely to cut off most other points from the anomalous point. In this example, it only takes 4 cuts to completely isolate the anomalous point. In other words, it has a path length of 4 in the iTree.

As argued before the path length for anomalous points is expected to be shorter than average. Therefore it is not necessary to fully build the iTree, but only to isolate points at levels up to a shorter than average depth in the tree. This reduces the running time of the algorithm. The fact that not the full tree is built is reflected in the scoring of the points: if the point resides at the maximum depth for that tree a penalty is added to the anomaly score to account for the part of the tree that was not completely built. In that case, the path length will be calculated as the path length in the tree, plus the expected average path length of the sub-tree that would occur beyond the height limit of the iTree. The expectation is based on the number of points residing at that node. Because



(a) Isolating a normal point in 13 random cuts

(b) Isolating an anomalous point in 4 random cuts

Figure 3: Partial construction of an iTree in two examples, showing why anomalies generally take fewer cuts to isolate.

of the random nature of the algorithm scores from multiple iTrees have to be combined to obtain a robust anomaly score for a point.

Weaknesses of iForest. *The algorithm does not take absolute distance into account because of scale independence* [22]. This means that in data where purely distance-based anomaly detection algorithms would perform well, iForest can have problems because it is making random cuts between a data dependent minimum and a maximum, no matter the absolute distance between those extremes. This can cause high false positive rates and inconsistent results.
The algorithm is sensitive to the sample size [22]. Isolation Forest uses a sample of the full data to build iTrees. This mechanism makes it unlikely that many anomalies are used in building the iTrees. This is beneficial to reducing *swamping* and *masking* effects [33]. Swamping refers to the effect that normal instances are classified as anomalous. Masking refers to the opposite effect: anomalies are classified as normal. This can happen because there is a small, but dense cluster of anomalies, that take many partitions to isolate. However, by using a sample this small cluster will likely be represented by only a few points, making the anomalies once again easy to isolate.

However, it has been found that if iForest does not include any anomalies in the sample and, for example, two clusters are present in the sample, it will later classify the anomalous point as a part of one of the two clusters (normal point). If the anomaly would have been included in the sample, then this behaviour would not occur [22]. This example shows that iForest is sensitive to those

anomalies that may be included in the sample.
The algorithm suffers from ghost clusters because of the binary partitioning [23]. The fact that each individual cut in iForest is along a single dimension (i.e. a straight line), can lead to unwanted classification behaviour. An example of this phenomenon is a *ghost cluster*: if we have, for example, a cluster at (0,0) and (5,5) in the sample, then iForest will have a bias towards classifying points around (0,5) and (5,0) as normal. A visualization of this issue can be found in Extended Isolation Forest [23].

2.4 Evaluation metrics and data

Now that we have discussed widely used anomaly detection techniques, it is also important to have a way to compare them to one another. In low-dimensional spaces, it is easy to visualize results, but it is still challenging for an algorithm to detect anomalies in this setting. In high-dimensional space it is more difficult to visualize results and we need to have an automatic way to measure and benchmark different anomaly detection algorithms.

ROC-AUC score. Receiver operating characteristic (ROC) is a way of measuring the accuracy of a binary classifier. Since anomaly detection is essentially a binary classification task, characterized by a great class imbalance, ROC also works well as a measure of performance of anomaly detection algorithms. The ROC-curve can be understood as the true positive rate on the y-axis, against the false positive rate on the x-axis. The true positive rate is defined as the number of true positives (TP) divided by the total number of positives (P). The false positive rate (FPR) is defined as the number of false positives (FP) divided by the number of negatives (N). The area under the ROC-curve (ROC-AUC) is a measure that indicates the probability that the classifier correctly predicts the class of a random instance. A completely random classifier is therefore expected to have a ROC-AUC of 0.5. Good classifiers have scores close to 1, and if the score is below 0.5, one can simply reverse the decision function of the classifier to get a score above 0.5.

Contamination. In our analysis, we consider two types of contamination:

1. **Data contamination** refers to the degree of contamination in the data. This is calculated by dividing the number of true anomalies in the data

by the total size of the data (times 100% if we want to express it as a percentage).

2. **Algorithm contamination** refers to the fraction of the full data that the anomaly detection algorithm is instructed to return as anomalous. Isolation Forest and the algorithm we propose both have this contamination as an input parameter. If we set it to 0.01, that means the algorithm returns the top 1% of points with the highest anomaly score as the anomalies.

Ideally, the algorithm contamination parameter should be set to closely match the data contamination. If we set the algorithm contamination lower, then the algorithm is guaranteed to miss anomalies. If we set it too high, then it is guaranteed to classify some normal points as anomalies. Therefore, we argue that if two algorithms score similarly in terms of ROC-AUC, the algorithm that achieves this score with its contamination parameter set closer to the data contamination is performing better.

Gaussian ditributions. In our results and other figures, we often use *Gaussian distributions* [21]. A Gaussian distribution, often called *normal distribution*, is a distribution that is characterized by the following probability function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Where roughly 68% of the data is expected to fall within 1 standard deviation σ of the expected mean μ, and almost 95.5% is expected to fall within 2σ of μ. It is important to use Gaussian distributions in evaluation because many variables found in the real world resemble a Gaussian distribution. For example, the amount of error a machine arm has when placing down an object at a specified point resembles a Gaussian distribution.

3 Algorithms

In this chapter, we describe the two anomaly detection algorithms we develop in this research, *random shift forest* and *randomly shifted quadtree forest*. Furthermore, we highlight the differences and similarities to Isolation Forest.

3.1 Random Shift Forest

The main algorithm that we develop is the Random Shift Forest (RSF). This algorithm works for any number of dimensions by using random dimension selection to partition the data. The algorithm works in multiple stages that we break down in detail in the upcoming sections.

Preliminary steps. First of all, we assume the input of RSF is a d-dimensional Euclidean point set P. The value of point $p \in P$ in dimension i is referred to as p_i. For each dimension i of P, we compute the minimum, $\min_{p \in P} p_i$, and maximum, $\max_{p \in P} p_i$, valued points. From these points, we compute the minimum $\min(P)$ and maximum $\max(P)$ values across all dimensions. The range r of P is calculated as $r = \max(P) - \min(P)$. We initialize k empty trees. For each tree T, and for each dimension i ($i \in 1, \ldots, d$), a random shift rs_i^T is picked satisfying $0 \leq rs_i^T \leq r$. These i random shifts are applied to each point by addition: $p_i^T = p_i + rs_i^T$.

Then, a d-dimensional hypercube B with side length $2 \cdot r$ is created that can fit all points in P, even after randomly shifting them. All k trees in the ensemble will use a copy of B as a root node. Lastly, a maximum number of points $p_{\max}$ that can reside in the same leaf is set in this step.

A sub-sample $\mathcal{S}$ of the data is used to build the k trees that will be used to generate the anomaly scores. We typically use a fixed sample size $n_{samples}$ of 128 or 256, similar to iForest [33]. A maximum depth $depth_{\max}$ is determined for the trees, it depends on the expected depth for a point in the sample.

A random order $\mathcal{V}$ is generated for each tree. This is a vector of length $depth_{\max}$ of integer values, chosen uniformly at random (with repetition) from the range $\{0, 1, \cdots, d-1\}$. This vector decides in which order the dimensions will recursively be cut in half when we build the tree. At depth j in the tree, the element with index j from $\mathcal{V}$ will be the dimension that is cut in half at that depth, for every node at that depth that still needs to be divided.

If the difference between the ranges of the dimensions is large, some dimensions can disproportionately influence the number of splits needed to isolate a point. In this case, we first perform min-max normalization on the data to equalize the relevancy of the dimensions. For each dimension i of each $p \in P$ we calculate a new value p'_i with the following equation:

$$p'_i = \frac{p_i - \min_{p \in P} p_i}{\max_{p \in P} p_i - \min_{p \in P} p_i}$$

Building the trees. We build k trees $T_1, T_2, \cdots, T_k$ where T_i is linked to the sample of shifted points $\mathcal{S}_i$. For a point in sample $\mathcal{S}_i$ we recursively insert it in tree T_i until it either reaches $\max_{depth}$, or the point is isolated.

The insertion process is as follows: for a point $s \in \mathcal{S}_i$ we first insert it in the root node of T_i. Then we recursively do the following steps: check if the number of points in the current node $u \in T_i$ is smaller than $p_{\max}$. If this is the case, add s to the points in u. Otherwise, we check if u has children u_{left} and u_{right}. If it does not, we split node u at depth j in half along dimension $\mathcal{V}_j$, spawning two children nodes. We then check whether the value of s_i for dimension j is smaller or larger than the split value, and recursively insert it in u_{left} or u_{right} based on this check: this is the node whose bounding box contains s. Furthermore, any other points that may have been residing in u are also evaluated with the split value and inserted in the corresponding child node u_{left} or u_{right}. This ensures that all points reside in the leaves of the tree after each update.

To evaluate the points, we want all of them to reside in the leaves of the trees. Therefore, when adding points to the fitting child nodes, they are deleted from the parent node. This is important because otherwise points that are inserted earlier would be more likely to reside at a smaller depth, changing their anomaly score. If a point reaches a leaf at the maximum depth of the tree, it is always added to that leaf, regardless of the number of nodes currently residing in that leaf. For points that reach a leaf at $depth_{\max}$ holding more points than $p_{\max}$, we define a penalty that is applied during the scoring phase of the algorithm.

A notable difference to iForest is that in RSF we have a parameter called *granularity*. When a few points are clustered together, RSF will usually take more splits to isolate these points. These points can form so-called *micro-clusters* [3], which can be a form of a collective anomaly. To allow the user of the algorithm to detect these micro-clusters we use a different $p_{\max}$ in each group of trees. Instead, we create trees with varying values for $p_{\max}$. So, instead

of splitting cells until a point is isolated, we only keep splitting until there are at most $p_{\max}$ points in the cell, where $p_{\max}$ is anywhere between 2 and m (we do not use more than $m = 5$ in our experiments). This helps to detect micro clusters that typically consist of 1 to 4 points in the sample as anomalous. Furthermore, it increases the likelihood that if an anomaly is included in the random sample, the algorithm can still detect points that are close to it as anomalies. Figure 6a is an example of a situation where the algorithm that did not use the granularity principle would have unwanted classification behaviour. Note that in Isolation Forest, granularity is used in a different context. There, granularity of the results can be changed using the height limit of the iTrees [33].

After the previous steps, we have k trees containing $n_{samples}$ points. These trees act as sketches of the full data to evaluate all other points with. Figure 4 shows a tree decomposition of a small set of points. Furthermore, paths to an anomaly (in orange) and a normal points (in blue) are highlighted, illustrating how a normal point generally has a higher path length than an anomalous point.

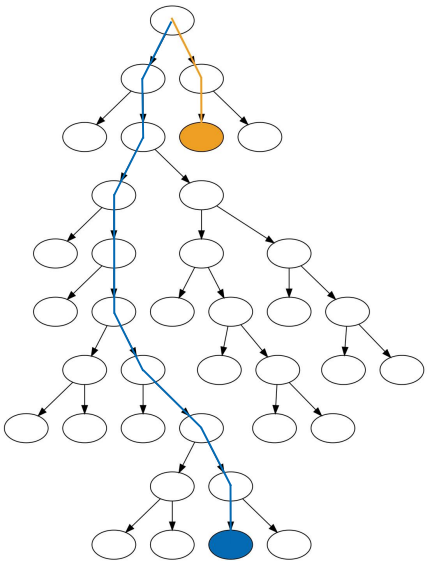
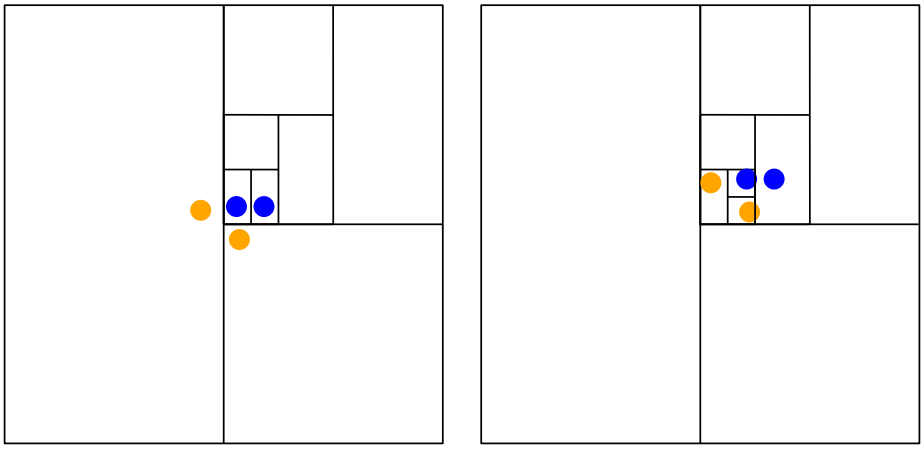


Figure 4: Tree decomposition of a small point set generated by RSF, highlighting a path to an anomaly (in orange), and a normal point (in blue)

Scoring the points. In the first tree, we use the raw input coordinates of the points. But, in the other trees we add a random shift to all of the coordinates of

the points. This is important because we want to avoid having a bias towards certain initial positions of the points as is illustrated in figure 5. In sub-figure 5a it takes more than 3 times as many cuts to isolate the blue points, than it takes to isolate the orange points. This is counter-intuitive because all 4 points are close together. In iForest this problem is addressed by using a random cut value between a minimum and a maximum in each new tree in the forest. In the case of RSF, we eliminate the randomness of the cutting procedure. Instead, we shift the points by a random amount between a minimum and maximum in each dimension. This means that the split values in RSF are still random relative to the points, but not relative to the other splits within the same tree. In sub-figure 5b the points have been shifted relative to the grid, but have the same pairwise distances between them as in sub-figure 5a. However, it takes 5 to 7 cuts to isolate each point in this case. This illustrates that if we shift points randomly in each new tree of the forest, eventually the bias seen in 5a will average out.



(a) The orange points are isolated within 2 cuts, the blue points after 7 cuts

(b) All points take at least 5 cuts to isolate

Figure 5: Effect of the random shift

The points that are not in the sample are then evaluated against this ensemble of trees. This means that they are inserted in these trees to check at what depth they would become isolated, but the point itself is not added to the tree, ensuring that it does not contaminate the sample. Therefore we can score the points without changing the sketch. The score in the tree is the depth of the point in that tree. If the depth of a point is equal to $depth_{\max}$, then a penalty is

applied to its score, since the point would most likely be at greater depth if the full tree was built. The formula we use for the score of a point at the maximum depth is:

$$score(p) = max_{depth} + \log n_{leaf}$$

Where n_{leaf} is the number of points in the parent node of the point. We add this amount, because it is an estimation of the expected depth of the subtree that was never built below the leaf.

After each point has been evaluated in each tree, the scores are accumulated. This means that for each point, its score will be the cumulative depth in all k trees. Then, this score is divided by the number of trees, to obtain the average path length $A(p)$.

To make this score easier to interpret, we apply the same transformation technique that is used in Isolation Forest [33]. First, we compute the expected path length of unsuccessful search in a Binary Search Tree (BST) [41], referred to as $c(n)$:

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

Here, $H(i)$ is the ith harmonic number, estimated by $\ln i + \gamma$, where γ is Euler's constant. n is the number of points in the tree, i.e. the sample size. Now we transform the scores with the following equation:

$$s(p) = 2^{-\frac{A(p)}{c(n)}}$$

As $c(n)$ is the same for all points in a tree, the sorted order of the points will be reversed by this function. In the end, scores of anomalous points will be closer to 1, and scores of normal points will be closer to 0.5.

The user can input the level of contamination c expected in the data, and the algorithm will return the top $c\%$ points with the highest anomaly scores as the anomalies.

Parameters. The RSF algorithm has four parameters: contamination, granularity, sample size, and number of trees. In the coming paragraphs, we explain these 4 parameters and how they can influence our results.

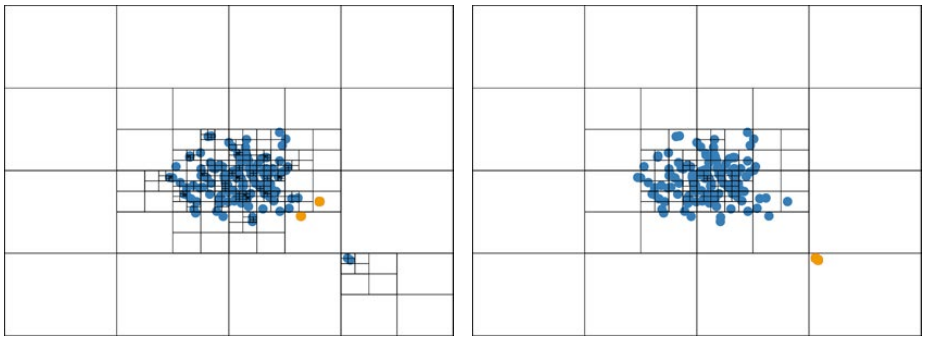
- **Contamination:** the value of the contamination parameter is the fraction of the total number of points that RSF marks as anomalous. RSF ranks all

input points with the anomaly score function and sorts the points on this score. If we set the contamination parameter to 0.1, then RSF will mark the top 10% of most anomalous points as anomalies and the remaining 90% as normal points. It is therefore important that the user of the algorithm has some knowledge of the expected fraction of anomalies in the data: if the contamination parameter is lower than the actual contamination of the data, then we are guaranteed to have false negatives. In some figures we refer to this parameter as c.

- **Granularity:** this parameter controls how the maximum number of points per cell changes as more trees are added to the forest. If we set this parameter to 10, then the first 10 trees can only have 1 point in each leaf (except for those at $depth_{max}$), the next 10 trees can have 2 points per leaf, and so on. By lowering the granularity parameter the user increases the likelihood that RSF detects points in smaller micro-clusters in the data as anomalous. Furthermore, it reduces masking effects because having a few anomalies in the sample has a smaller effect on the way the trees are built, as is illustrated in figure 6. This figure shows a Gaussian distribution of 128 points, and 2 anomalous points that are close together in the bottom right. In sub-figure 6a the maximum number of points that can reside in a cell is 1. Since the anomalous points are so close together, it takes more splits to isolate them. The two points in orange are the most anomalous according to the algorithm, even though they belong to the Gaussian distribution. In sub-figure 6b however, the maximum number of points in a leaf is set to 2. Now, the true anomalous points are isolated in fewer splits than any of the points from the Gaussian distribution. The algorithm will correctly label these two points as anomalous.

- **Sample size:** to build the trees RSF uses a sub-sample of the full data. The parameter allows the user to use any integer below n_{points} to set $n_{samples}$. Therefore, this parameter controls the absolute number of points that will be drawn randomly from the data to create the samples. We typically use 128 or 256 samples for the best results in our experiments.

- **Number of trees:** because the trees RSF builds are highly randomized, we need multiple trees to increase the reliability of the output. This parameter enables the user to control how many trees RSF will build. In other words, it lets the user of the algorithm pick k. Adding more trees



(a) at most 1 point per leaf (b) at most 2 points per leaf

Figure 6: Effect of increasing the maximum number of points per leaf

increases the stability of the output, but it also increases the running time of the algorithm linearly.

Complexity analysis. The most time- and memory-intensive stage of the algorithm is the evaluation of the points that were not used to create the trees. If we set the maximum depth of a tree to a multiple of $\log n_{samples}$ then evaluating a point in a single tree has time complexity $\mathcal{O}(\log n_{samples})$. Evaluating all points in all trees then has time complexity $\mathcal{O}(k \cdot n \log n_{samples})$. We conclude the explanation of RSF by providing pseudocode in algorithm 1, and subroutines 2 (insert) and 3 (score).

Algorithm 1: RSF - Random Shift Forest -

Let P be a set of points with d dimensions;
Let k be the number of trees;
for *j from* 1 *to* k **do**
    Let $rs_1^j, rs_2^j \dots rs_d^j$ be uniformly at random chosen real numbers between 0 and $max(P) - min(P)$;
    for *p in* P **do**
        for *i from* 1 *to* d **do**
            $p_j.i = p.i + rs_i^j$;
        end
    end
end
The root cell ranges from $min(P)$ to $min(P) + 2 \cdot (max(P) - min(P))$ in all d dimensions ;
Pick a sample S of size $n_{samples}$ to construct k trees stored in set T;
for *j from* 1 *to* k **do**
    for *s in* S **do**
        INSERT(s_j, T_j)
    end
    for *p in* $P \setminus S$ **do**
        SCORE(p_j, T_j)
    end
end
Sort all points of P by anomaly score;
return top c fraction of P as the anomalous points

Algorithm 2: INSERT(p, C)

```
1  if C_depth = depth_max then
2  |   add p to C_points;
3  |   p_score = C_depth + penalty;
4  |   return;
5  else if C_npoints < max_points and C is a leaf then
6  |   add p to C_points;
7  |   p_score = C_depth;
8  |   return;
9  if C is a leaf then
10 |   Split C in half, spawning child nodes C_left and C_right ;
11 Let C_fitting be the child node fitting p;
12 INSERT(p, C_fitting);
13 for Every point p_C still in C do
14 |   Let C_fitting be the child node fitting p_C;
15 |   INSERT(p_C, C_fitting);
16 end
```

Algorithm 3: SCORE(p, C)

```
1  if C_depth = depth_max then
2  |   p_score = C_depth + penalty;
3  |   return;
4  else if C_npoints < max_points and C is a leaf then
5  |   p_score = C_depth;
6  |   return;
7  Let C_fitting be the child node fitting p;
8  SCORE(p, C_fitting);
```

3.2 RSQT Forest

For the 2-dimensional setting, we developed a simplified variation of the RSF algorithm that we call RSQT Forest (Randomly Shifted QuadTree Forest). This algorithm does not use random dimension ordering, but cuts in the middle in both dimensions simultaneously, dividing the space into 4 equal parts, this data structure is otherwise known as a quadtree. Since we add the random shift, it becomes a quadtree with random shift. A quadtree with a random shift is a data structure that has been used in the past, for example for the travelling salesman problem [5]. A quadtree has depth at most $log(\frac{sl_{root}}{min_{dist}}) + \frac{3}{2}$, where sl_{root} is the side length of the root node and min_{dist} is the minimum distance between any two points [14]. Since min_{dist} can be arbitrarily small in the input data, a max_{depth} was used in this algorithm, but it still had issues detecting anomalous micro-clusters of a few points. Whether a small cluster of points should be seen as an anomaly or not depends on the context, so the granularity parameter was added to RSF to control the size of the clusters that can be detected as an anomaly. We should mention that RSQT Forest does not use random sampling. These techniques were added to RSF because they are critical to performance on big datasets with many dimensions. For the 2-dimensional case with a few thousand points these techniques barely change the behaviour of the algorithm in practice. Since RSQT Forest does not use random sampling, it builds the full trees with all points in the data. Therefore, RSQT Forest has a running time of $\mathcal{O}(k \cdot n \log n)$. Pseudocode of the algorithm is presented below in algorithm 4. This algorithm shows how one tree is created, in practice we execute this subroutine up to 100 times to obtain more stable results. The scores are combined in the same way as in the RSF algorithm.

Algorithm 4: RSQT - Randomly Shifted Quadtree -

Let P be a set of points with 2 dimensions;
Let a, b be uniformly at random chosen numbers between 0 and $max(P) - min(P)$;
for *p in P* **do**
    $p.x+ = a$;
    $p.y+ = b$;
end
The root square cell now ranges from $min(P)$ to $min(P) + 2 \cdot (max(P) - min(P))$ in both dimensions ;
Consider cell c with four possible children c_1, c_2, c_3, c_4;
if *cell c has more than* max_{points} *points* **then**
    Split c in the middle in both dimensions to create the children. Recursively call $RSQT(\mathbf{c_1})$, $RSQT(\mathbf{c_2})$, $RSQT(\mathbf{c_3})$, and $RSQT(\mathbf{c_4})$;
    Return the union of the obtained partitions as the partition of c;
else if *c has fewer points* **then**
    Return $\{c\}$ itself as the partition of c;

4 | Results and discussion

In this chapter, we describe the experiments we performed with RSF and RSQT. First, we highlight the 2-dimensional synthetic datasets used to demonstrate the performance of RSQT Forest. Then, we use these datasets to show how RSQT Forest compares to other widely used anomaly detection algorithms. Afterwards, we compare RSF to other anomaly detection algorithms. To this end, we use both synthetic and real-world data. The former type is useful to analyse the behaviour of the algorithm under different circumstances, while the latter type is suited to demonstrate real-life effectiveness compared to other algorithms. Finally, we analyze the influence of the four parameters of RSF, and show how scalable the algorithm is.

4.1 Experimental setup

RSF and RSQT Forest are implemented in Python, and are publicly available on GitHub. [1]. We use the following 4 anomaly detection algorithms as baselines:

- **Robust covariance** is an implementation of the statistical algorithm FAST-MCD [44].
- **One-Class SVM** is a statistical algorithm that tries to find a non-linear decision boundary that separates normal points from anomalies [43].
- **Local Outlier Factor (LOF)** is a nearest neighbour approach that uses distance to neighbours as an approximation for local density [10].
- **Isolation Forest** is an ensemble method that is detailed in chapter 2 of this thesis.

The implementations of these algorithms that we used are all part of the scikit-learn package for Python [38]. Furthermore, this package was used to generate some of the synthetic datasets. All experiments were carried out on a 2.4 GHz Windows laptop with 8 GB of RAM.

[1] https://github.com/Thijs3/Anomaly-Detection-with-RS-Quadtree

4.2 RSQT Forest on 2-dimensional datasets

In this section, we compare RSQT Forest to other anomaly detection algorithms on 2-dimensional synthetic datasets.

Dataset types. We generated multiple synthetic datasets to compare the characteristics of our algorithm to those of other anomaly detection algorithms. They are mainly used to observe the behaviour of different algorithms when data has different types of shapes and distributions. Our synthetic datasets are all in 2-dimensional Euclidean space. They can be divided into 3 groups. One group consists of datasets of geometric shapes constructed from straight lines. These datasets resemble structures commonly found on geographical maps, such as T-junctions and highways. The second group consists of curved shapes. These types of shapes could also occur on geographical maps (a river, for example). The last group contains datasets consisting of *Gaussian distributions* [21].

Noise. The coordinates of the points making up the noise are generated uniformly at random from the domain of the normal points. This means that all values between the minimum and the maximum are evenly likely to appear as a coordinate of the noise. The goal of the anomaly detection algorithms is to detect as much of the background noise as possible as anomalies. The contamination parameter of the anomaly detection algorithms (if this parameter is present) is set to match the fraction of random noise in the data. We use the following synthetic datasets to compare the performance of RSQT to other anomaly detection algorithms:

- **Two-moon dataset:** The two moons dataset from scikit-learn. The moon shapes have equally many points.
- **Nested squares dataset:** Two squares with the same center but different side lengths.
- **Cross dataset:** Two lines placed in a plus shape.
- **T-junction dataset:** Two lines placed in a T-shape. This simulates a T-junction that could occur in geographical data. Anomalies could then be seen as faulty GPS data.
- **Parallel lines dataset:** Two lines following a Gaussian distribution in one dimension and a fixed value in the other dimension.

- **Rectangles dataset:** Two rectangles with the same center point.
- **Nested circle dataset:** Four circles of different sizes, with the same center point.
- **Two-circle dataset:** Two sets of points on the outlines of two circles, both containing equally many points. One has a diameter twice as big as the other one, and the smaller circle is completely on the inside of the larger one. Both circles have the same center point.
- **Gaussian dataset:** A single Gaussian distribution
- **Small cluster dataset:** Two Gaussian distributions with different center points. The first Gaussian distribution has center point (0,0), has a standard deviation of 2.0, and contains 80% of all normal points. The second Gaussian distribution contains the remaining 20% of normal points, has center point (5,5) and a standard deviation of 0.2.
- **Mixed Gaussians dataset:** Two Gaussian distributions with different center points and standard deviations, each containing half of the normal points.

Each of these synthetic datasets consists of 1000 points in total: 75 points of random noise and 925 normal points. The contamination is therefore $\frac{75}{1000} = 0.075$.

Discussion. In figure 7 we show the result of running iForest and RSQT Forest on the two moons dataset. Furthermore, for RSQT Forest we visualize the partitioning. We observe that RSQT Forest is able to detect the random noise in the center as anomalous, whereas iForest classifies these points as normal. On the other hand, iForest is wrongly classifying points in the tails of both moons as anomalies. This phenomenon is caused by the fact that iForest does shrinking after each cut, such that absolute distance between points is not reflected in the anomaly scores of iForest. RSQT Forest is not perfect either: it classifies some points that are part of the moon shapes as anomalies.

Figure 8 shows the comparison of RSQT Forest to Robust covariance, One-Class SVM, LOF, and iForest on 2-dimensional datasets consisting of straight lines, resembling roads on geographical maps. In the nested squares dataset, RSQT Forest is classifying the random noise points as anomalies more often

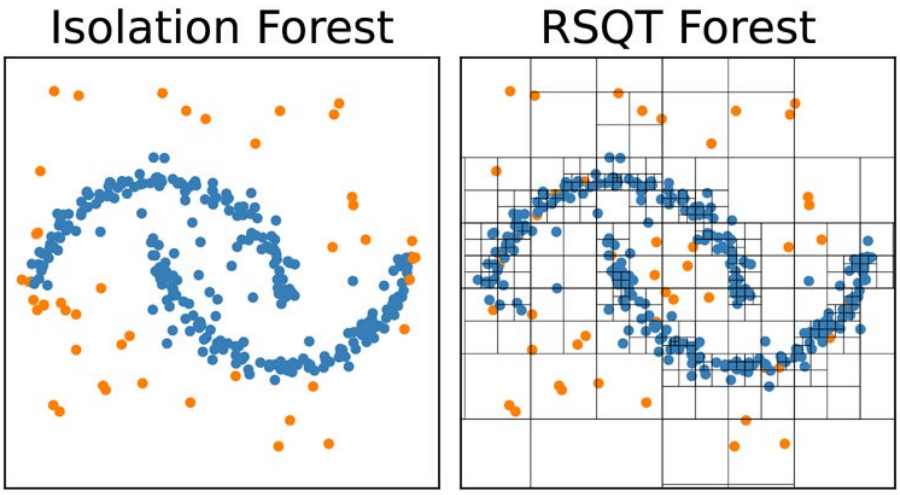


Figure 7: Comparison of Isolation Forest to RSQT Forest on the moons dataset

than all other algorithms. The other algorithms mainly have trouble with classifying the noise on the interior of the squares as anomalies. Even on such a simple dataset, RSQT Forest is the only algorithm out of these 5 that is classifying the points in the same way a human would. In the cross dataset and T-junction dataset, there is a similar pattern: Robust covariance and One-Class SVM include much of the noise as normal points, Isolation Forest makes a few misclassifications, and Local Outlier Factor and RSQT Forest classify almost all points correctly. In the parallel lines dataset, Isolation Forest classifies much of the noise as normal points. Only RSQT Forest is able to classify most noise as anomalous and to report the points near the ends of the lines as normal. In the rectangles dataset, we see similar results as in the squares dataset: only RSQT Forest is able to classify all the points that are part of the rectangles as normal points.

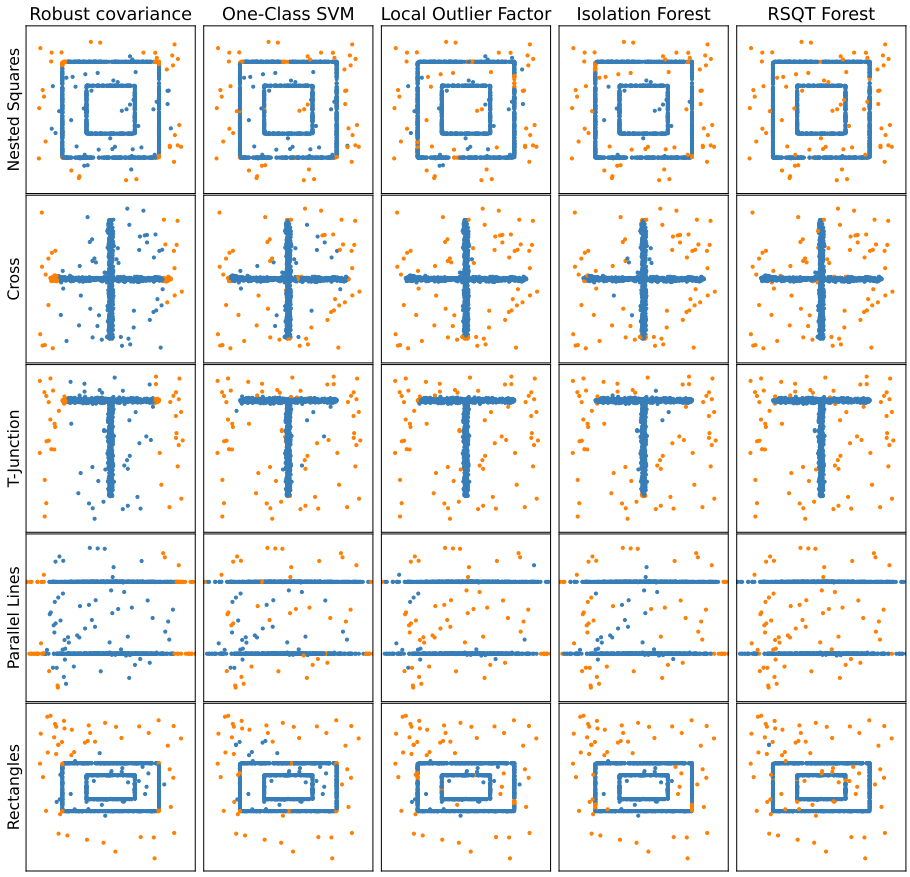


Figure 8: Comparing anomaly detection algorithms to RSQT Forest on map-like data.

In figure 9, RSQT Forest is compared to the other algorithms on the datasets including curved shapes. In the nested circles dataset, only RSQT Forest is able to identify almost all of the random noise as anomalies. The other algorithms all classify some of the points that are clearly on one of the circles as anomalous. In the two circles dataset, LOF is also able to identify points in the interior as anomalous, but it still classifies many points between the inner and outer circles as normal. Once again, RSQT Forest is the most accurate at classifying random noise as anomalous. In the moons dataset, only LOF and RSQT Forest are classifying points in the center as anomalies, whereas all other algorithms classify points belonging to the outer tails of the moons as anomalies.

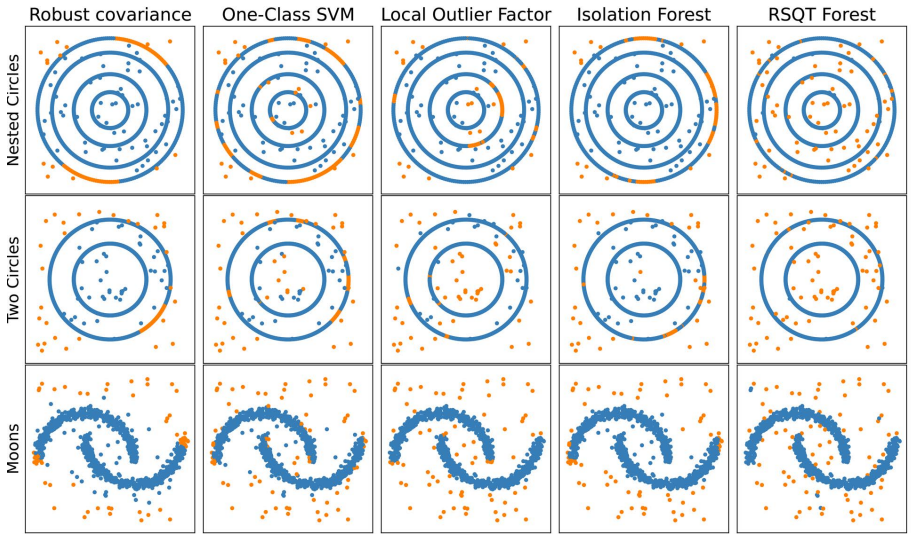


Figure 9: Comparing anomaly detection algorithms to RSQT Forest on curved shapes.

Finally, in figure 10 the same 5 algorithms are compared on datasets consisting of 1 or 2 Gaussian distributions. In the first dataset, a single Gaussian distribution, all algorithms classify most of the noise as anomalies. RSQT Forest is making some mistakes when it comes to noise points that are in micro-clusters. We attribute this to the fact that RSQT Forest does not use sampling, so this issue should be resolved in RSF. In the next dataset, with a small cluster and a larger cluster, we see that RSQT Forest and iForest classify global outliers best. A local approach like LOF will classify some points belonging to the small cluster as anomalies because they have a lower local density than their

tightly packed together neighbouring points. One-class SVM is classifying some points near the center of the large cluster as anomalies. In the mixed Gaussians dataset, where both clusters have the same number of points, something similar happens. RSQT Forest and Isolation Forest classify the points in the cluster with lower variance as normal, whereas LOF identifies anomalies in both clusters.

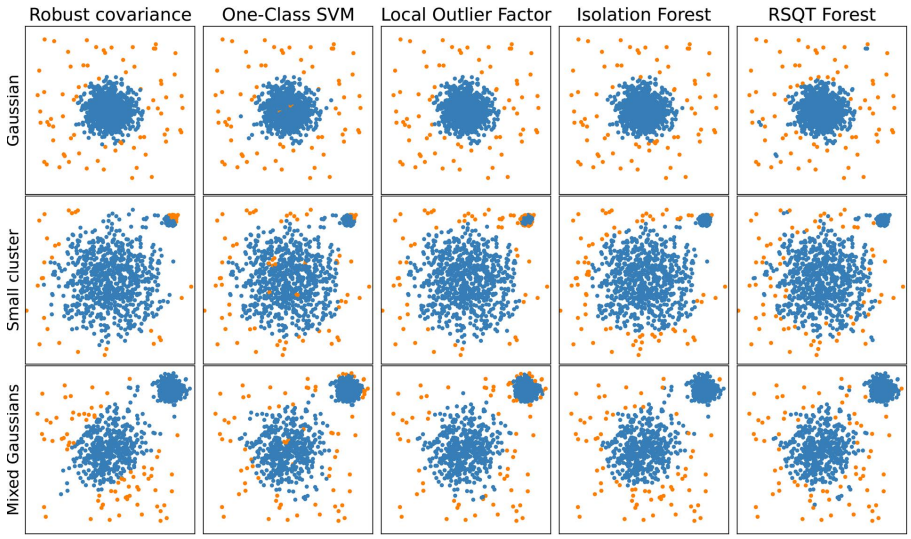


Figure 10: Comparing anomaly detection algorithms to RSQT Forest on Gaussian distributions.

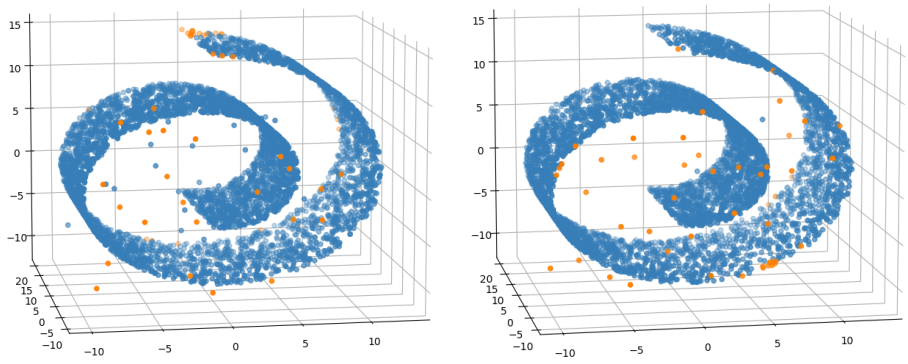
We tested existing widely used anomaly detection algorithms and RSQT Forest on a variety of datasets. We conclude that in a majority of these cases RSQT Forest is performing similarly to the best performing other anomaly detection algorithm, and in some cases, it is outperforming all of the other algorithms. Especially in the two circles, two squares, and two moons datasets, RSQT Forest is the only algorithm capable of detecting anomalies positioned between the normal clusters.

4.3 RSF on 3-dimensional datasets

In this section, we compare RSF to iForest on 3-dimensional datasets. We use two synthetic and one real dataset to visualize the results of using both algorithms.

Datasets. We use the **swiss roll** dataset and the **s-curve** dataset to evaluate performance on a 3-dimensional dataset. These datasets are planes that are formed into a roll and an "S" shape, respectively. They consist of 4950 points in our results, and 50 points with coordinates chosen uniformly at random are added as the noise. Therefore both 3-dimensional synthetic datasets consist of 5000 points. Finally, we use the KKDCUP99 **SMTP** dataset to demonstrate results on a real dataset in 3 dimensions.

Discussion. In figure 11 the anomalies predicted by iForest and RSF are plotted in orange. Similar to the 2-dimensional results, iForest is not detecting the random noise in the center. RSF does a much better job at this, detecting almost all of the random noise as anomalies. At the same time, iForest is classifying points near the edges and top of the roll as anomalies. RSF also makes some mistakes, but to a much smaller extent than Isolation Forest.



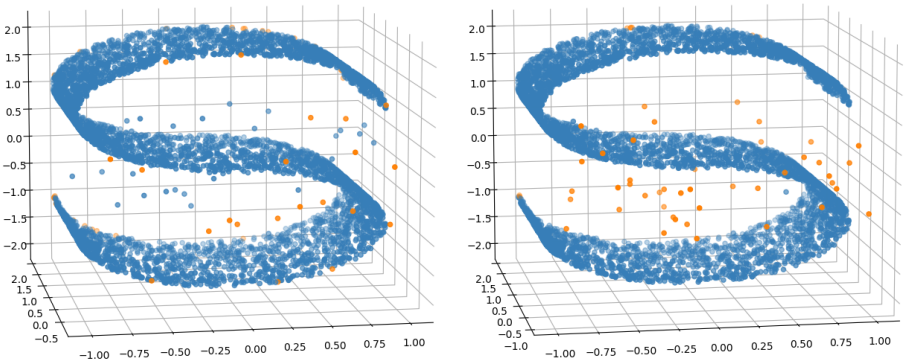
(a) iForest anomalies (c=0.01) (b) RSF anomalies (c=0.01)

Figure 11: Predicted anomalies in the swiss roll dataset for iForest and RSF

In figure 12 something similar occurs: this time iForest is not detecting most of the random noise in the center, while RSF is predicting close to every point of noise as anomalous. One of the reasons we think iForest is not good at finding anomalies is because it has a bias introduced by the shrinking mechanism. Be-

cause of this mechanism, iForest will split 2 points near the end of a distribution from each other in 1 cut, no matter the distance between them. Therefore the path length of a point near the edge of a cluster will likely have a lower average path length in iForest than in RSF.

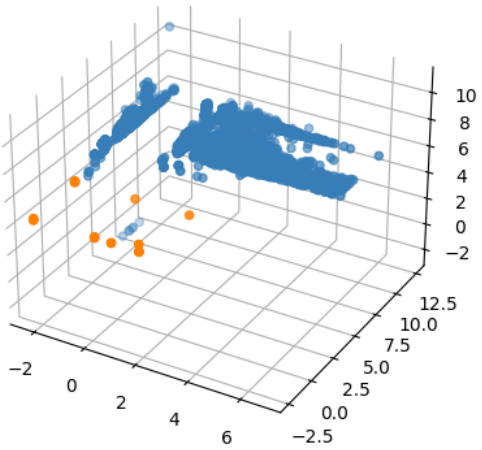
Since we are plotting a 3-dimensional dataset in figures 11 and 12, not every point is visible. Additional figures on these 2 datasets, from a variety of viewpoints, can be found in the appendix.



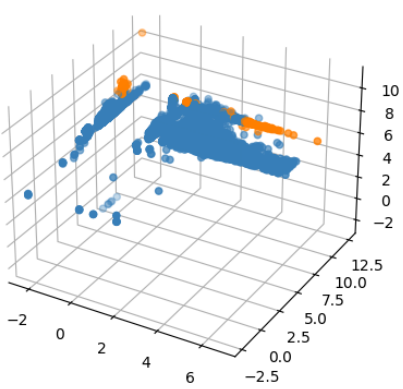
(a) iForest anomalies (c=0.01) (b) RSF anomalies (c=0.01)

Figure 12: Predicted anomalies in the s-curve dataset for iForest and RSF

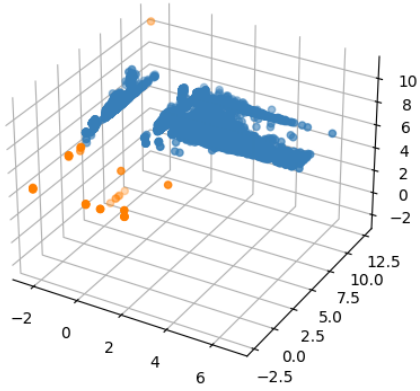
The result in figure 13 is remarkable, as RSF is able to detect most of the anomalies. On the other hand, iForest reports none of the anomalies, despite using a much higher contamination setting. RSF was consistently able to detect more than 20 of the anomalies in this dataset, while only reporting 40 points in total. We assume that the reason why iForest is not reporting the anomalies is because it does not take absolute distance between points into account enough. As stated before, iForest will cut points near the outside of a cluster off in 1 cut. Therefore, it is reporting normal points near the edge of the dense clusters as anomalous. The fact that the true anomalies are much further away does not play a big role in iForest, but it does in RSF.



(a) The labelled dataset, used as ground truth



(b) iForest predicted anomalies (c=0.005)



(c) RSF predicted anomalies (c=0.0003)

Figure 13: Anomalies in SMTP dataset: ground truth, iForest, and RSF.

4.4 RSF on real datasets

In this section, we compare RSF to iForest on a variety of real datasets. These datasets are labeled, so we can quantitatively compare the algorithms. We use ROC-AUC score as the accuracy measure.

Datasets. To compare the performance of our algorithm to that of iForest in a real setting, we use datasets that are also used in the original iForest paper [33]. Our motivation for this choice of datasets is that we can replicate the performance from this paper, verifying their results and ensuring we benchmark RSF and RSQT Forest on datasets that iForest performs well on.

We used two datasets from the KDDCUP99: The **HTTP** and the **SMTP** dataset [26]. These contain data about network traffic, and the goal is to detect intrusion. Intrusion in this context means that an attacker tries to connect to the network. In the HTTP dataset, about 0.4% of the connections are labeled as *bad*, meaning they are created by attackers. In the SMTP dataset only 30 connections (0.03% of all data) are marked as *bad*. This makes the SMTP dataset the dataset with the smallest fraction of anomalies that we used. Both of the datasets have 3 dimensions, therefore they can also be visually presented and analyzed.

We also use the **statlog landsat satellite** dataset that consists of multi-spectral values in a 3-by-3 neighborhood in a satellite image. The instances are placed in 7 classes (but class 6 never appears in the data) as follows:

1. red soil
2. cotton crop
3. grey soil
4. damp grey soil
5. soil with vegetation stubble
6. mixture class (all types present)
7. very damp grey soil

In the iForest paper, the least common 3 classes of this dataset (class 2, 4, and 5) were considered to be anomalous. It was found during the experiments that using just one of the classes as the anomalous class would lead to a more

accurate classification in terms of ROC-AUC. This is the class called *cotton crop*. We therefore both replicate the experiment with classes 2, 4, and 5 as anomalous classes, and create a new version with just class 2 as the anomalous class.

The **Pima Indians Diabetes Database** is a dataset of diagnostic measurements that can be used to predict whether or not a patient has diabetes. Patients with diabetes are less common (35% of all cases), so they will be the anomalous class. Since the anomalous class is relatively large, classifying anomalies is expected to be difficult: the effect of masking will be much greater since many anomalies are expected to be included in the sampling phase of both iForest and RSF.

The next dataset is the **statlog shuttle** dataset. It is a dataset with 9 numerical attributes that are used to classify shuttles in 7 classes. Classes 1 (rad flow) and 4 (high) combined account for 93% of all cases, so they will be seen as the normal instances.

The final real dataset that was used is the **ForestCover** dataset. The full data has 7 classes that denote the most common species of trees in a forest. 10 numerical attributes that correspond to ecological factors in the forest, are used to predict the type of prevalent trees. In the experiment, we focus on class 2 (lodgepole pine) against class 4 (cottonwood/willow). In this case, class 2 makes up 99.1% of all instances and should therefore be classified as normal.

The ForestCover, shuttle, and satellite datasets were all obtained from the UCI Machine Learning Repository [15]. The Pima Indians Diabetes Database can be found on Kaggle [2]. Table 1 gives an overview of all the real datasets described above, including the number of instances, number of dimensions, anomaly class, and anomaly rate.

Dataset	Instances	d	Anomaly class	Anomaly pct
ForestCover	286048	10	cottonwood/willow (class 4)	0.96%
HTTP	567497	3	attack (marked 1)	0.39%
Pima	768	8	positive (marked 1)	34.9%
Satellite	6435	36	classes 2, 4, 5/ class 2	32%/12%
Shuttle	49097	9	classes 2, 3, 5 & 7	7%
SMTP	95156	3	attack (marked 1)	0.03%

Table 1: Overview of real datasets (d = number of dimensions).

[2] `https://www.kaggle.com/uciml/pima-indians-diabetes-database`

Discussion. Table 2 shows the performance of iForest and RSF on a variety of real datasets in terms of ROC-AUC. In all of the cases, the ROC-AUC of the two algorithms is fairly close, with RSF having a higher score in 4 of the 7 cases. The SMTP dataset stands out because RSF is able to get a similar ROC-AUC score with a much lower contamination parameter than iForest. In this dataset, the number of anomalies (30) is very small compared to the total size of the data ($\pm$100.000). In such cases, it is desirable that an algorithm does not need to output many false positives, but iForest is detecting many normal points as anomalous. If we set the contamination parameter to a lower value, such as in sub-figure 13b, iForest does not detect any of the true anomalies. Therefore it will then have a ROC-AUC score close to 0.5, which corresponds to not doing any better than an algorithm that returns random points. In sub-figure 13a we showed the true anomalies as labeled in the dataset, where we noted that there are 10 more anomalies that are not visible in the figure because they are part of the large cluster. Therefore, we can not expect any density-, distance- or tree-based anomaly detection algorithm to find those anomalies based on the three features in the dataset. In sub-figure 13c the contamination parameter is set to match the true anomaly rate in the data (0.03%), and RSF is still able to detect all of the anomalies except for the 10 obscured anomalies mentioned before.

Dataset	**ROC-AUC**		**Contamination**		
	iForest	*RSF*	*iForest*	*RSF*	*Data*
HTTP	0.998	0.992	0.05	0.05	0.039
ForestCover	0.876	0.892	0.24	0.18	0.0096
Pima	0.641	0.675	0.45	0.42	0.35
SMTP	0.853	0.831	0.06	0.0005	0.0003
Shuttle	0.983	0.974	0.075	0.075	0.007
Satellite (3 anomaly classes)	0.688	0.723	0.32	0.32	0.32
Satellite (1 anomaly class)	0.946	0.968	0.13	0.125	0.12

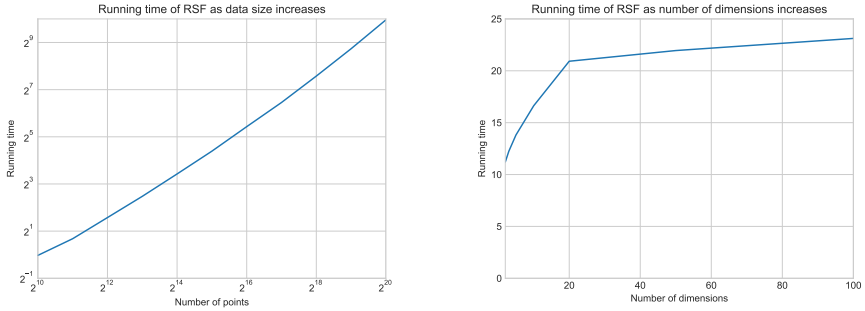
Table 2: ROC-AUC score in real datasets for iForest and RSF

These real dataset experiments show that RSF has a similar performance to iForest in practice, and in some cases, can do better than iForest in terms of false-positive rate.

4.5 Running time and parameters

We have seen that RSF is as good or better than other anomaly detection algorithms when it comes to finding anomalies in a variety of synthetic and real datasets. In this section, we analyze the running time of RSF. Running time is important for anomaly detection algorithms, as real-world applications often include big data. Finally, we investigate the influence of the RSF parameters on its performance in terms of ROC-AUC score.

Running time. Running time is important for every algorithm, especially algorithms that are made to be used on big data. In chapter 3, we argued that RSF has a theoretical running time of $\mathcal{O}(k \cdot n \log n_{samples})$. Since we do not have to increase k and $n_{samples}$ necessarily when we deal with bigger datasets, the running time is expected to increase linearly with respect to input size n. One assumption we made when arguing about this running time is that all dimensions in the data are relevant. If some of the dimensions turn out to be irrelevant when it comes to distinguishing anomalies, then we need to make more cuts to isolate points. Therefore we may need to increase $depth_{max}$, which in turn increases the running time of the algorithm. Another solution in the case of high-dimensional data is to select a subset of dimensions, as is argued in [33].



(a) Running time of RSF as the number of points increases

(b) Running time of RSF as the number of dimensions increases

Figure 14: Running time of RSF

Figure 14 shows how the running time of RSF is affected by the size of the dataset. The number of points in sub-figure 14a changes from 1024 (2^{10}) to $1,048,576$ (2^{20}) and comes from a single generated Gaussian distribution.

The observed increase in running time corresponds to the theorized $\mathcal{O}(k \cdot n \cdot \log n_{samples})$ from chapter 3, in terms of the increase in the total number of points. If assume the number of samples is constant, the running time scales linearly with the input size. Additionally, when the number of dimensions is increased, while keeping the same number of points, the running time also increases. The increase is smaller with each added dimension because once the number of dimensions is too large compared to the number of points, the curse of dimensionality causes each point to be isolated quickly. Furthermore, the relative extra overhead of an extra dimension decreases as the number of dimensions grows.

Parameters. In chapter 3, we introduced the 4 parameters of RSF: contamination, granularity, sample size, and number of trees. In figure 15 we plot the values of these parameters against the ROC-AUC scores on the shuttle dataset. This figure provides an indication of the way ROC-AUC scores are influenced by changing the parameters. Does the score rely heavily on the parameters set by the user, or does RSF perform well with a wide range of parameter values? Ideally, the score is robust to any reasonable amount of change to the parameters.

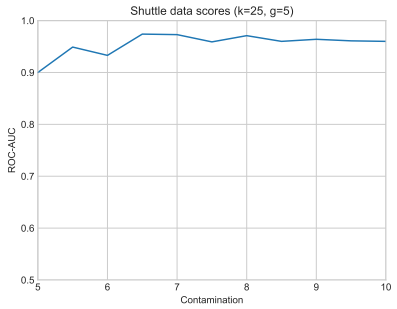
In sub-figure 15a, the contamination (c) parameter has the highest score when it is set to 6.5% (or 0.065). In general, it is advisable to have the contamination parameter higher than the true contamination in the data, otherwise there will be guaranteed false negatives in the result. Simultaneously, depending on the application the contamination parameter should be as low as possible to avoid too many false positives.

The next examined parameter is the granularity (g) in sub-figure 15b, ranging from 1 (after each tree max_{points} increases by 1) to 25 (max_{points} is 1 for each of the 25 trees in the ensemble). We notice a decline in scores when $g > 10$. The optimal value for this parameter is highly dependent on the number of (micro)clusters in the data. In general, if some of the anomalies form a cluster, it is advisable to use a lower value for g. Since this parameter can have a large impact on the outcome of the algorithm the expert using RSF must have some understanding of the domain and typical anomalies in the data.

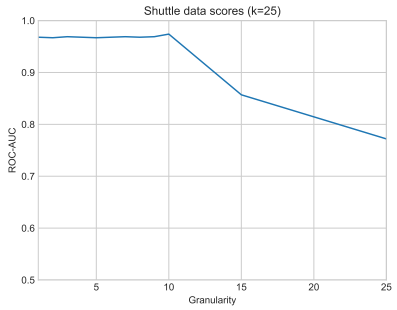
The third parameter is the number of trees in the ensemble. Since we have many degrees of randomness in the algorithm, results per tree can vary greatly. Therefore multiple trees should be used to increase the reliability of the results. In sub-figure 15c the number of trees ranges from 1 to 100. Experimentally the

results converge quite rapidly: using more than 25 trees does not change the ROC-AUC scores significantly.

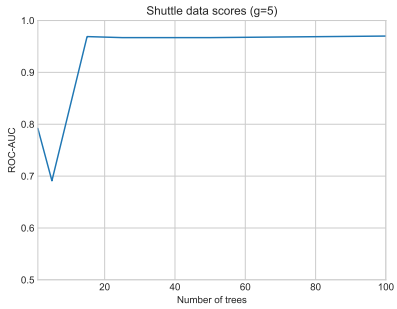
The final parameter is the sample size (s). This parameter can greatly affect the accuracy of the results, as is clear from sub-figure 15d. If it is too small, there is no good representation of the full data. If it is too large, masking effects are introduced by the inclusion of anomalies in the sample. In general, we advise using 128 to 512 samples, using larger sample size if there are more clusters in the data.



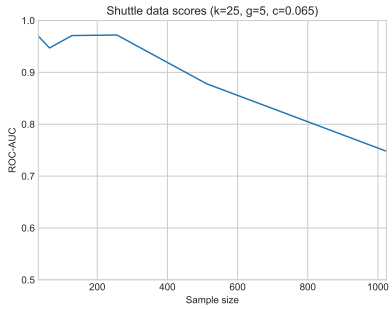
(a) Effect of contamination on ROC-AUC score



(b) Effect of granularity on ROC-AUC score



(c) Effect of the number of trees in the forest on ROC-AUC score



(d) Effect of sample size on ROC-AUC score

Figure 15: Effects of the parameters of RSF on performance

5 | Streaming and parallelism

In this chapter, we show that RSF construction can be easily implemented in the streaming and parallel models using z-sparse recovery sketches. A z-sample recovery algorithm [19; 36; 30] recovers $\min(z, \|x\|_0)$ elements from x such that sampled index i has $x_i \neq 0$ and is sampled uniformly, where $\|x\|_0$ of a vector x counts the number of non-zero entries of x. Constructions of z-sample recovery data structures are known which require space $\mathcal{O}(z)$ and fail only with probability polynomially small in n [6].

Algebraically, the streaming and distributed anomaly detection algorithms use union and subtraction set operators that are done using z-sparse recovery sketches. Indeed, we use the subtraction and union operators to develop dynamic streaming algorithms where batches of points are inserted or deleted in a streaming fashion. As an example, assume that we have three data sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ where $\mathcal{P}_3 \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$ and $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$. Suppose we would like to find the anomalies of $\mathcal{P} = \mathcal{P}_2 \cup \mathcal{P}_1 \backslash \mathcal{P}_3$. Since the size of these sets may be extremely large, we cannot afford to store them entirely. Instead, we store a sketch of each of them. To this end, we compute the sketches $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ of $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, respectively, and then apply the sketching operation $\mathcal{S}_1 + \mathcal{S}_2 - \mathcal{S}_3$ to obtain anomalies of $\mathcal{P}$. Similarly, we can use the union operator to implement a parallel algorithm. In particular, suppose we have multiple data sets $\mathcal{P}_1, \cdots, \mathcal{P}_m$ and our goal is to compute anomalies of the union set $\mathcal{P} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_m$. We first compute a sketch $\mathcal{S}_i$ of the point set $\mathcal{P}_i$. Then, we compute the summation $\mathcal{S}_1 + \cdots + \mathcal{S}_m$ using which we can compute anomalies of the point set $\mathcal{P}$, efficiently. Below, we detail the streaming approach for RSF.

Streaming Algorithm. Let $\mathcal{S}$ be a dynamic stream of inserts and deletes of points of a point set $P \subset \mathbb{R}^d$. We let P_t be the set of points that are inserted but not deleted from the beginning of the stream until time t. Let $\mathcal{F}$ be the RSF that we build offline if we run the RSF algorithm with the input point set P_t. We would like to develop a dynamic streaming algorithm that at any time t of the stream reports the same anomalies as the offline forest $\mathcal{F}^{offline}$ finds for P_t. To this end, in the course of the stream, we maintain z-sample recovery sketches for every internal node and leaf of trees of the RSF $\mathcal{F}$. In particular, for each tree $\mathcal{T}$ in the RSF $\mathcal{F}$ we do the following.

We denote by $(p_{\max}+1)$-SR(u), the $(p_{\max}+1)$-sparse recovery sketch that we maintain for the internal node or the leaf u in the tree $\mathcal{T}$, where $p_{\max}$ is the maximum number of points that a leaf can have before we split it. Upon insertion of a point p, starting from the root of $\mathcal{T}$, we traverse the path of the nodes whose rectangles contain p till we end up with the leaf u whose corresponding rectangle contains p. We add p to the $(p_{\max}+1)$-sparse recovery $(p_{\max}+1)$-SR(v) of any node v that we see along this path including the leaf u. Observe that this is different than the offline version of RSF where points are stored only in leaves.

Suppose the leaf u is at level ℓ and the rectangle that corresponds to u is $rect(u)$. If the leaf u has fewer than $p_{\max}$ points, we do nothing; otherwise (i.e., if u has $p_{\max}$ points) we pick the entry $\mathcal{V}[\ell]$ of the random vector $\mathcal{V}$ and then split the rectangle $rect(u)$ at the middle point of the side of $rect(u)$ that is along the dimension $\mathcal{V}[\ell]$. Then, we create two leaves u_1 and u_2. Let u be the parent of these two and let u_1 and u_2 be the children of u. We also initialize $(p_{\max}+1)$-sparse recovery sketches for both u_1 and u_2. From the $(p_{\max}+1)$-sparse recovery sketch of u, we extract the point set $P(u)$ of u and add each point $q \in P(u)$ to the $(p_{\max}+1)$-sparse recovery of the leaf whose rectangle contains q. Later, we add p to the $(p_{\max}+1)$-sparse recovery of the leaf whose rectangle contains p. If both u_1 and u_2 have fewer than $p_{\max}$ we add p to the $(p_{\max}+1)$-SR(v) of the leaf whose rectangle contains p and stop. Otherwise, that is if one of them, say u_1 is empty and the other one, say u_2 contains $p_{\max}+1$ points, we recurse the splitting procedure with the leaf u_2.

Upon deletion of a point p, starting from the root of $\mathcal{T}$, we traverse the path of the nodes whose rectangles contain p till we end up with a leaf u whose corresponding rectangle contains p. We delete p from the $(p_{\max}+1)$-SR(v) of any node v that we see along this path including the leaf u. Suppose the leaf u is at level ℓ and the rectangle that corresponds to u is $rect(u)$. Let v be the parent of u in level $\ell-1$. If after deletion of p from $(p_{\max}+1)$-SR(v), the node v has more than $p_{\max}$ points in its subtree, we stop and do nothing; otherwise (i.e., if v has fewer than $p_{\max}$ points), we then delete the children of v and recursively check the parent of the node v till we have a node whose subtree contains more than $p_{\max}$ points, but its children have fewer than $p_{\max}$ points each.

Observe that such sketching algorithm cannot be done if we use iForest instead of RSF. Suppose we have two data sets $\mathcal{P}$ and $\mathcal{P}'$ where $\mathcal{P}' \subseteq \mathcal{P}$. Suppose that we cannot store these point sets in the memory, so we need to sketch them. In particular, suppose for the point set $\mathcal{P}$, iForest finds a random split

value C along a randomly chosen dimension, that splits $\mathcal{P}$ into two subsets $\mathcal{P}_1$ and $\mathcal{P}_2$. We store z-sparse recovery sketches S_1 and S_2 of $\mathcal{P}_1$ and $\mathcal{P}_2$. Similarly, suppose for the point set $\mathcal{P}'$, iForest finds a random split value C' along a randomly chosen dimension, that splits $\mathcal{P}'$ into two subsets $\mathcal{P}'_1$ and $\mathcal{P}'_2$, respectively. Observe that iForest picks C and C' at a random value along a randomly chosen dimension, so with very high probability, C and C' are not the same. We store z-sparse recovery sketches S'_1 and S'_2 of $\mathcal{P}'_1$ and $\mathcal{P}'_2$, respectively. Now, suppose the set $\mathcal{P}\backslash\mathcal{P}'$ has fewer than z points in one side of the split value C. Since the split values C and C' are not the same, we cannot use the subtraction operations $S_1 - S'_1$ and $S_2 - S'_2$ to recover all points that are in the side of C that has fewer than z points. However, using RSF, the above is possible. Suppose we have two RSF trees, one for $\mathcal{P}$ and one for $\mathcal{P}'$. Furthermore, suppose both trees use the same random order. Then, any node at a level i of both trees use the same set of split values starting from the root of both trees going along the path that ends at that node. Therefore, we can use set minus operations on sketches of ancestors of the node to find anomalies of the ancestors of the node.

6 Conclusion

In this final chapter, we reflect on the extent to which the goals set in the introduction were met by our research. First, we discuss how future work could address the limitations of our research. Finally, we end the thesis with concluding remarks.

6.1 Limitations and future research

In this section we address the limitations of this thesis and discuss how these issues could be resolved in future research.

We introduced a way to implement RSF in a distributed and streaming setting. However, due to time constraints, we did not implement and test this method. Therefore, empirical evidence that the technique works is missing. Future research should focus on implementing this algorithm to verify that it performs well through experiments.

Additionally, future work should focus on exploring the limits of the sample sizes used in this research. For the datasets used in the experiments, a small sample size was enough to achieve good ROC-AUC, and increasing sample size did generally not improve the results in terms of ROC-AUC. When the algorithm is used in a real-world streaming setting, there could be cases where there are so many normal clusters in the data that a larger sample size is necessary to represent all of them consistently.

Furthermore, we did not implement a way to sample the data when it is arriving in a stream. For a static dataset, it is straightforward to take a random sample from the whole set, but in a stream of unknown length and time of arrival, it becomes more difficult to create a sample that represents the stream as a whole accurately. A possible way this could be achieved is to include newly arriving points with a probability that is based on the anomaly score of the point: anomalous points should have a smaller probability to be included in the sample. However, the probability should be greater than 0. This way, a new normal cluster of sufficient size will still have a good probability of being included in the sample, regardless of the length of stream and time of arrival.

Another issue that was not solved for the streaming setting is that we assume that we know the limits of the numerical range of a feature. In a static dataset, it is easy to find the minimum and maximum of each of the features and

normalize them accordingly. In the streaming setting, if we know nothing about the arriving data, this range changes over time as new points arrive. Therefore the algorithm only works correctly if we have prior knowledge of the minimum and maximum values a feature can have.

Lastly, future work could explore different types of data. We focused on multidimensional numerical datasets, but other types of data should also be compatible with RSF. For example, graph data could be converted in some way such that RSF can extract anomalies. Furthermore, ways to include categorical data in the input would be useful.

6.2 Concluding remarks

In the introduction, we stated the goal of this thesis was to explore the possibility of a general-purpose unsupervised anomaly detection algorithm that uses isolation to detect anomalies that can be used in big data models. The RSF algorithm proposed in chapter 3 is such an algorithm that can be adapted to the distributed and streaming model. RSF uses recursive partitioning to isolate points, similarly to iForest. At the same time, constructing RSF has a different source of randomness than iForest, making it suitable for parallel computation and infinite data streams. Furthermore, RSF showed promising results on both synthetic and real datasets. It has similar or better ROC-AUC scores compared to iForest, while often having to report fewer points as anomalies. Moreover, it is better at finding anomalies that are surrounded by normal points, as was demonstrated in figures 11 and 12.

References

[1] Ahmed, S., Lee, Y., Hyun, S.-H., and Koo, I. Unsupervised machine learning-based detection of covert data integrity assault in smart grid networks utilizing isolation forest. *IEEE Transactions on Information Forensics and Security 14*, 10 (2019), 2765–2777.

[2] Alghushairy, O., Alsini, R., Soule, T., and Ma, X. A review of local outlier factor algorithms for outlier detection in big data streams. *Big Data and Cognitive Computing 5*, 1 (2021).

[3] Amer, M., and Goldstein, M. Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer. In *Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012)* (2012), pp. 1–12.

[4] Amer, M., Goldstein, M., and Abdennadher, S. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description* (2013), pp. 8–15.

[5] Arora, S. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM) 45*, 5 (1998), 753–782.

[6] Barkay, N., Porat, E., and Shalem, B. Efficient sampling of non-strict turnstile data streams. *Theor. Comput. Sci. 590* (2015), 106–117.

[7] Ben-Gal, I. Outlier detection. In *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 131–146.

[8] Bolton, R. J., Hand, D. J., et al. Unsupervised profiling methods for fraud detection. *Credit scoring and credit control VII* (2001), 235–255.

[9] Braei, M., and Wagner, S. Anomaly detection in univariate time-series: A survey on the state-of-the-art. *arXiv preprint arXiv:2004.00433* (2020).

[10] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (New York,

NY, USA, 2000), SIGMOD '00, Association for Computing Machinery, p. 93–104.

[11] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 1–58.

[12] Datar, M., and Muthukrishnan, S. Estimating rarity and similarity over data stream windows. In *European Symposium on Algorithms* (2002), Springer, pp. 323–335.

[13] De Benedetti, M., Leonardi, F., Messina, F., Santoro, C., and Vasilakos, A. Anomaly detection and predictive maintenance for photovoltaic systems. *Neurocomputing 310* (2018), 59–68.

[14] De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications (3d edition)*. Springer, 2008.

[15] Dua, D., and Graff, C. UCI machine learning repository, 2017.

[16] Escalante, H. J. A comparison of outlier detection algorithms for machine learning. *Programming and Computer Software* (01 2005), 228–237.

[17] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD-96 Proceedings* (1996), vol. 96, pp. 226–231.

[18] Faria, E. R., Gama, J. a., and Carvalho, A. C. P. L. F. Novelty detection algorithm for data streams multi-class problems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (2013), Association for Computing Machinery, p. 795–800.

[19] Frahling, G., Indyk, P., and Sohler, C. Sampling in dynamic data streams and applications. In *Proceedings of the 21st ACM Symposium on Computational Geometry* (2005), pp. 142–149.

[20] Goldstein, M., and Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one 11*, 4 (2016), e0152173.

[21] Goodman, N. R. Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction). *The Annals of mathematical statistics 34*, 1 (1963), 152–177.

[22] Guha, S., Mishra, N., Roy, G., and Schrijvers, O. Robust random cut forest based anomaly detection on streams. In *Proceedings of the 33nd International Conference on Machine Learning* (2016), M. Balcan and K. Q. Weinberger, Eds., vol. 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 2712–2721.

[23] Hariri, S., Kind, M., and Brunner, R. J. Extended isolation forest. *IEEE Transactions on Knowledge and Data Engineering 33*, 04 (apr 2021), 1479–1489.

[24] Hariri, S., and Kind, M. C. Batch and online anomaly detection for scientific applications in a kubernetes environment. In *Proceedings of the 9th Workshop on Scientific Cloud Computing* (2018), ScienceCloud'18, Association for Computing Machinery.

[25] Hawkins, D. M. *Identification of outliers*, vol. 11. Springer, 1980.

[26] Hettich, S., and Bay, S. D. The UCI KDD Archive, 1999.

[27] Hodge, V., and Austin, J. A survey of outlier detection methodologies. *Artificial intelligence review 22*, 2 (2004), 85–126.

[28] Jin, W., Tung, A. K., Han, J., and Wang, W. Ranking outliers using symmetric neighborhood relationship. In *Pacific-Asia conference on knowledge discovery and data mining* (2006), Springer, pp. 577–593.

[29] Johnson, R. A., Wichern, D. W., et al. *Applied multivariate statistical analysis*, vol. 5. Prentice hall Upper Saddle River, NJ, 2002.

[30] Jowhari, H., Saglam, M., and Tardos, G. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2011), pp. 49–58.

[31] Keresztes, J. C., Diels, E., Goodarzi, M., Nguyen-Do-Trong, N., Goos, P., Nicolai, B., and Saeys, W. Glare based apple sorting and iterative algorithm for bruise region detection using shortwave infrared hyperspectral imaging. *Postharvest biology and technology 130* (2017), 103–115.

[32] Kriegel, H.-P., Kröger, P., Schubert, E., and Zimek, A. Loop: local outlier probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), pp. 1649–1652.

[33] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 413–422.

[34] Liu, Y., Pang, Z., Karlsson, M., and Gong, S. Anomaly detection based on machine learning in iot-based vertical plant wall for indoor climate control. *Building and Environment 183* (2020), 107–212.

[35] Manevitz, L. M., and Yousef, M. One-class svms for document classification. *Journal of machine Learning research 2*, Dec (2001), 139–154.

[36] Monemizadeh, M., and Woodruff, D. P. 1-pass relative-error l_p-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (2010), M. Charikar, Ed., SIAM, pp. 1143–1160.

[37] Papadimitriou, S., Kitagawa, H., Gibbons, P. B., and Faloutsos, C. Loci: Fast outlier detection using the local correlation integral. In *Proceedings 19th international conference on data engineering (Cat. No. 03CH37405)* (2003), IEEE, pp. 315–326.

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

[39] Pincus, R., Barnett, V., and Lewis, T. Outliers in statistical data. 3rd edition. *Biometrical Journal 37*, 2 (1995), 256–256.

[40] Portnoy, L., Eskin, E., and Stolfo, S. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security* (2001), pp. 5–8.

[41] Preiss, B. R. *Data Structure and Algorithms: With Object-oriented Design Patterns in Java*. John Wiley & Sons, 1999.

[42] Rakotosaona, M.-J., La Barbera, V., Guerrero, P., Mitra, N. J., and Ovsjanikov, M. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 185–203.

[43] Rätsch, G., Schölkopf, B., Mika, S., and Müller, K.-R. *SVM and boosting: One class*. GMD-Forschungszentrum Informationstechnik, 2000.

[44] Rousseeuw, P., and Driessen, K. A fast algorithm for the minimum covariance determinant estimator. *Technometrics 41* (08 1999), 212–223.

[45] Rousseeuw, P. J., and Hubert, M. Anomaly detection by robust statistics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8*, 2 (2018), e1236.

[46] Tang, J., Chen, Z., Fu, A. W.-c., and Cheung, D. A robust outlier detection scheme for large data sets. In *6th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2001), Citeseer.

[47] Thottan, M., and Chuanyi Ji. Anomaly detection in ip networks. *IEEE Transactions on Signal Processing 51*, 8 (2003), 2191–2204.

[48] Verleysen, M., and François, D. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks* (2005), Springer, pp. 758–770.

[49] Zimek, A., and Filzmoser, P. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8*, 6 (2018), e1280.

Appendices

A. Viewpoints s-curve

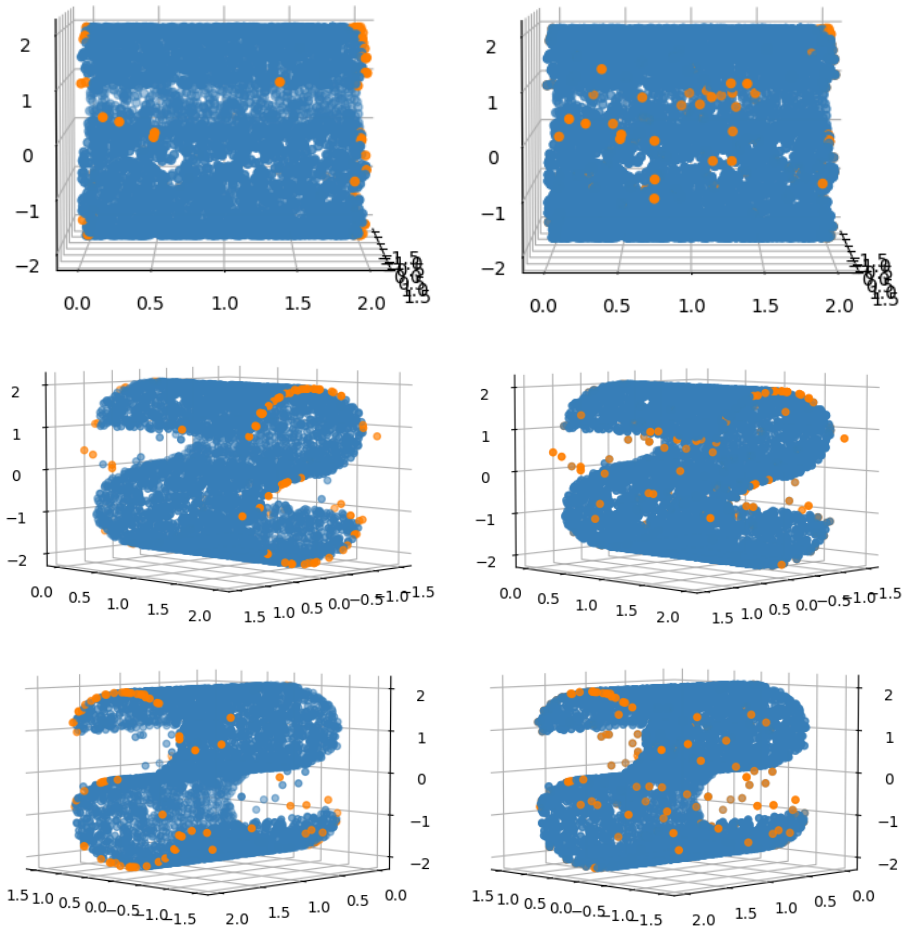


Figure 16: Additional viewing angles s-curve data (left: iForest, right: RSF)

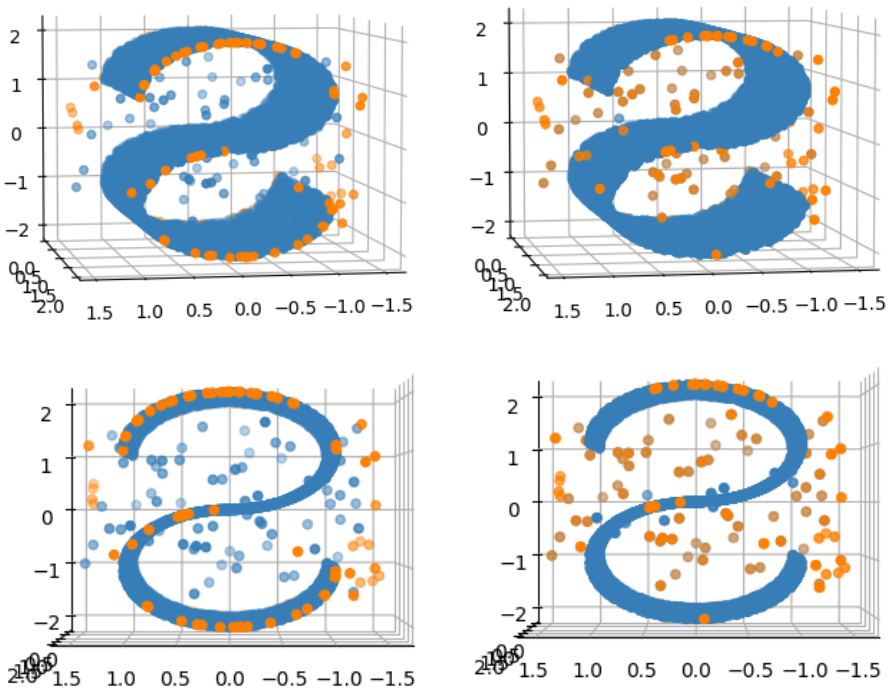


Figure 17: Additional viewing angles s-curve data (left: iForest, right: RSF)

B. Viewpoints swiss roll

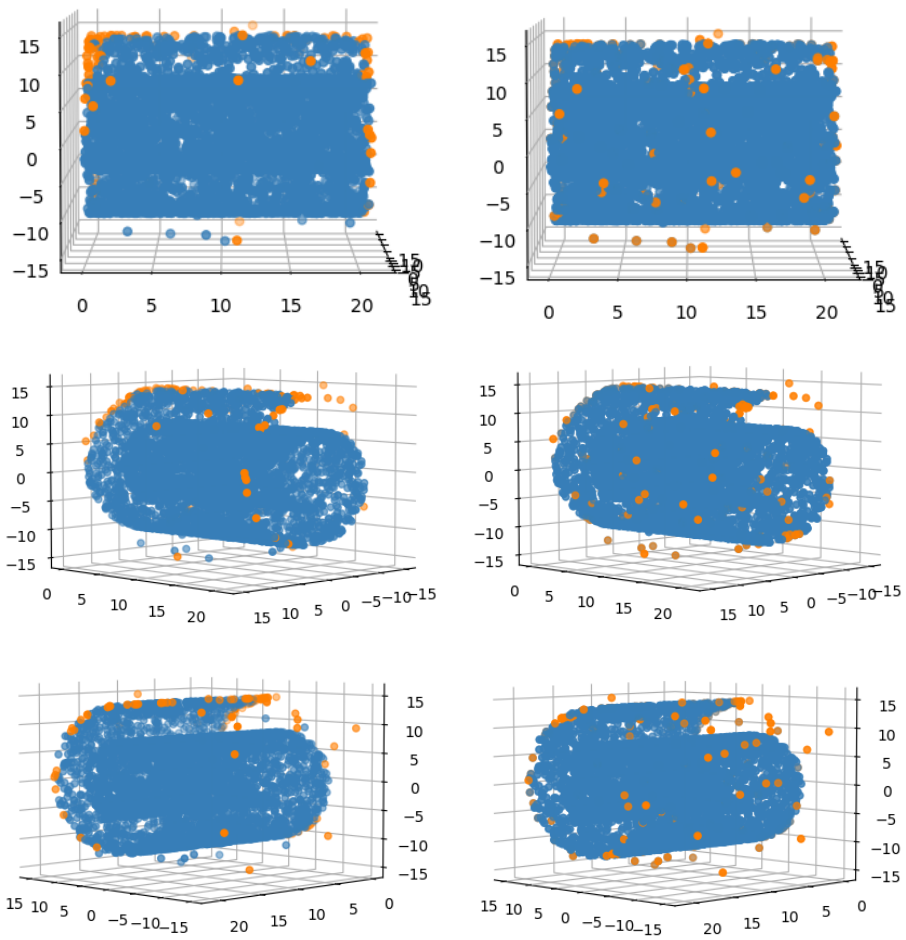


Figure 18: Additional viewing angles swiss roll data (left: iForest, right: RSF)

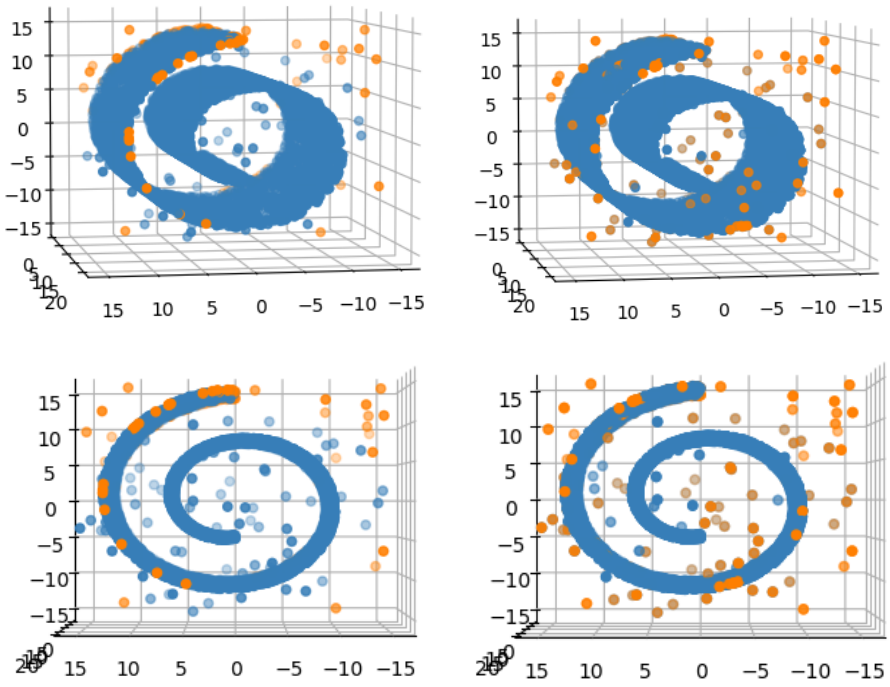


Figure 19: Additional viewing angles swiss roll data (left: iForest, right: RSF)