PIDForest: Anomaly Detection and Certification via Partial Identification

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We consider the problem of detecting anomalies in a large dataset. We propose a definition that captures the intuition that anomalies are easy to distinguish from the overwhelming majority of points by relatively few attribute values: we call this partial identification. Formalizing this intuition, we propose a geometric anomaly measure for a point that we call PIDScore, which measures for the minimum density of data points over all subcubes containing the point. We present PIDForest: a random forest based algorithm that finds anomalies based on this definition. We show that it performs favorably in comparison to several popular anomaly detection methods, across a broad range of benchmarks. PIDForest also provides a succinct explanation for why a point is labelled anomalous, by providing a set of features and ranges for them which are relatively uncommon in the dataset.

1 Introduction

An anomaly in a dataset is a point that does not conform to what is normal or expected. Anomaly detection is a ubiquitous machine learning task with diverse applications including network monitoring, medicine and finance [1, Section 3]. There is an extensive body of research devoted to it, see [1, 2] and the references therein. Our work is primarily motivated by the emergence of large distributed systems, like the modern data center which produce massive amounts of heterogenous data. Operators need to constantly monitor this data, and use it to identify and troubleshoot problems. The volumes of data involved are so large that a lot of the analysis has to be automated. Effective anomaly detection algorithms can help humans prioritize attention and hone in on important events. Here we highlight some of the challenges that an anomaly detection algorithm must face.

1. **High dimensional, heterogeneous data:** The data collected could contains measurements of metrics like cpu usage, memory, bandwidth, temperature, in addition to categorical data such as day of the week, geographic location, OS type. This makes finding an accurate generative model for the data challenging. The metrics might be captured in different units, hence algorithms that are unit-agnostic are preferable. The algorithm needs to scale to high dimensional data.
2. **Scarce labels:** Most of the data are unlabeled. Generating labels is time and effort intensive and requires domain knowledge. Hence supervised methods are a non-starter, and even tuning too many hyper-parameters of unsupervised algorithms could be challenging.
3. **Irrelevant attributes:** Often an anomaly manifests itself in a relatively small number of attributes among the large number being monitored. For instance, a single machine in a large datacenter might be compromised and behave abnormally.
4. **Interpretability of results:** When we alert a datacenter administrator to a potential anomaly, it helps to point to a few metrics that might have triggered it, to help in troubleshooting.

In the generative model setting, anomalies come with a simple explanation: a model that fits the data, under which the anomalous observation is unlikely. Interpretability is more challenging for algorithms that do not assume a generative model. In this work, we are particularly interested in random forest based methods for anomaly detection, namely the influential work on Isolation Forests [3] (we refer to this algorithm as iForest) and subsequent work [4, 5]. iForest is a remarkably simple and efficient algorithm, that has been found to outperform other anomaly detection methods in several domains [6]. Yet, there is no crisp definition of ground truth for what constitutes an anomaly: the anomaly score is more or less *defined* as the output of the algorithm. We believe that a necessary step for interpretability is a clear articulation of *what* is an anomaly, separate from the algorithmic question of *how* it is found.

Our contributions. We summarize the main contributions of this work:

1. In Section 2, we motivate and propose a new anomaly measure that we call $\mathrm{PIDScore}$. Our definition corresponds to an intuitive notion of what is an anomaly and has a natural geometric interpretation. It can be viewed as a natural generalization of the well studied notion of teaching dimension [7, 8].
2. Our definition sheds light on the types of points likely to be labeled as anomalies by the iForest algorithm, and also on the types of points it might miss. We build on this intuition to design an efficient random forest based algorithm—PIDForest, which finds anomalies according to $\mathrm{PIDScore}$, in Section 3.
3. We present extensive experiments on real and synthetic data sets showing that our algorithm consistently outperforms or matches six popular anomaly detection algorithms. PIDForest is the top performing algorithm in 6 out of 12 benchmark real-world datasets, while no other algorithm is the best in more than 3. PIDForest is also resilient to noise and irrelevant attributes. These results are in Section 4 and 5.

We begin by describing our proposed anomaly measure, $\mathrm{PIDScore}$ at a high level. Let the *sparsity* of a dataset $\mathcal{T}$ in a subcube of the attribute space be the volume of the subcube divided by the number of points from $\mathcal{T}$ that it contains. For a dataset $\mathcal{T}$ and a point x, $\mathrm{PIDScore}(x, \mathcal{T})$ measures the maximum sparsity of $\mathcal{T}$ in all subcubes C containing x. A point x is labelled anomalous if it belongs to a region of the attribute space where data points are sparse. While notions of density have been used in previous works on clustering and anomaly detection, our approach differs from prior work in important ways.

1. **Dealing with heterogenous attributes:** Dealing with subcubes and volumes allows us to handle heterogenous data where some columns are real, some are categorical and possibly unordered. All we need is to specify two things for each coordinate: what constitutes an interval, and how length is measured. Subcubes and volumes are then defined as products over coordinates. This is in sharp contrast to methods that assume a metric space. Notions like ℓ_1/ℓ_2 distance add different coordinates and might not be natural in heterogenous settings.
2. **Scale invariance:** For a subcube, we only care about the ratio of its volume to the volume of the entire attribute space. Hence we are not sensitive to the units of measurement.
3. **Considering subcubes at all scales:** In previous works, density is computed using balls of a fixed radius, this radius is typically a hyperparameter. This makes the algorithm susceptible to masking, since there may be a dense cluster of points, all of which are anomalous. We take the minimum over subcubes at all scales.

Given this definition, one could aim for an algorithm that preprocesses $\mathcal{T}$, then takes a point x and computes $\mathrm{PIDScore}(x, \mathcal{T})$. Such an algorithm is likely to suffer from the curse of dimensionality like in Nearest Neighbor based methods, and not scale to high volumes of data. Instead we adopt the approach of iForest [3] which focuses on what is anomalous, rather than the entire dataset. We call the resulting algorithm PIDForest. Like in iForest, PIDForest builds a collection of decision trees that partition space into subcubes. In PIDForest, the choice of the splits at each node favors partitions of greatly varying sparsity, the variance in the sparsity is explicitly the quantity we optimize when choosing a split. In contrast, previous work either choose splits randomly [3] or based on the range [4]. Choosing coordinates that have greater variance in their marginal distribution lets us hone in on the important coordinates, and makes our algorithm robust to irrelevant/noisy attributes, which are unlikely to be chosen. Secondly, we label each leaf by its sparsity rather than depth in the tree. The score of a point is the maximum sparsity over all leaves reached in the forest.

We present a detailed comparison between PIDForest and iForest in Section 3 and a detailed discussion of related work in Appendix B.

2 Partial Identification and $\mathrm{PIDScore}$

A motivating example: Anomalous Animals. Imagine a tabular data set that contains a row for every animal on the planet. Each row then contains attribute information about the animal such as the species, color, weight, age and so forth. The rows are ordered. Say that Alice wishes to identify a particular animal in the table unambiguously to Bob, using the fewest number of bits.

If the animal happens to be a *white elephant*, then Alice is in luck. Just specifying the attributes species and color narrows the list of candidates down to about fifty (as per Wikipedia). At this point, specifying one more attribute like weight or age will probably pin the animal down uniquely. Or she can just specify its order in the list.

If the animal in question happens to be a *white rabbit*, then it might be far harder to uniquely identify, since there are tens of millions of white rabbits, unless that animal happens to have some other distinguishing features. Since weight and age are numeric rather than categorical attributes, if one could measure them to arbitrary precision, one might be able to uniquely identify each specimen. However, the higher the precision, the more bits Alice needs to communicate to specify the animal.

We will postulate a formal definition of anomaly score, drawing on the following intuitions:

1. **Anomalies have short descriptions.** The more exotic/anomalous the animal Alice has in mind, the more it stands out from the crowd and the easier it is for her to convey it to Bob. Constraining just a *few carefully chosen* attributes sets anomalies apart from the vast majority of the population.
2. **Precision matters in real values.** For real-valued attributes, it makes sense to specify a range in which the value lies. For anomalous points, this range might not need to be very narrow, but for normal points, we might need more precision.
3. **Isolation may be overkill.** The selected attributes need not suffice for complete isolation. *Partial identification* aka narrowing the space down to a small list can be a good indicator of an anomaly.

First some notation: let $\mathcal{T}$ denote a dataset of n points in d dimensions. Given indices $S \subseteq [d]$ and $x \in \mathbb{R}^d$, let x_S denote the projection of x onto coordinates in S. Logarithms are to base 2. As a warm-up, we consider the Boolean setting where the set of points is $\mathcal{T} \subseteq \{0,1\}^d$.

2.1 The Boolean setting

First assume that $\mathcal{T}$ has no duplicates. We define $\mathrm{idLength}(x, \mathcal{T})$ to be the minimum number of co-ordinates that must be revealed to uniquely identify x among all points in $\mathcal{T}$. Since there are no duplicates, revealing all coordinates suffices, so $\mathrm{idLength}(x, \mathcal{T}) \leq d$,

Definition 1. (IDs for a point) We say that $S \subseteq [d]$ is an ID for $x \in \mathcal{T}$ if $x_S \neq y_S$ for all $y \in \mathcal{T} \setminus \{x\}$. Let $\mathrm{ID}(x, \mathcal{T})$ be the smallest ID for x breaking ties arbitrarily. Let $\mathrm{idLength}(x, \mathcal{T}) = |\mathrm{ID}(x, \mathcal{T})|$.

While on first thought $\mathrm{idLength}$ is an appealing measure of anomaly, it does not deal with duplicates, and further, the requirement of unique identification is fairly stringent. Even in simple settings points might not have short IDs. For example, if $\mathcal{H}$ is the Hamming ball consisting of 0^d and all d unit vectors, then $\mathrm{idLength}(0^d, \mathcal{H}) = d$, since we need to reveal all the coordinates to separate 0^d from every unit vector. One can construct examples where even the average value of $\mathrm{idLength}(x, \mathcal{T})$ over all points can be surprisingly high [8].

We relax the definition to allow for *partial identification*. Given $x \in \mathcal{T}$ and $S \subseteq [d]$, the set of impostors of x in $\mathcal{T}$ are all points that equal x on all coordinates in S. Formally $\mathrm{Imp}(x, \mathcal{T}, S) = \{y \in \mathcal{T} \ s.t. \ x_S = y_S\}$. We penalize sets that do not identify x uniquely by the *logarithm of the number of impostors*. The intuition is that this penalty measures how many bits it costs Alice to specify x from the list of impostors.

Definition 2. (Partial ID) We define

$$\mathrm{PID}(x, \mathcal{T}) = \arg\min_{S \subseteq [d]} (|S| + \log_2(|\mathrm{Imp}(x, \mathcal{T}, S)|)), \tag{1}$$

$$\mathrm{pidLength}(x, \mathcal{T}) = \min_{S \subseteq [d]} (|S| + \log_2(|\mathrm{Imp}(x, \mathcal{T}, S)|)). \tag{2}$$

It follows from the definition that $\mathrm{pidLength}(x, \mathcal{T}) \leq \min(\log_2(n), \mathrm{idLength}(x, \mathcal{T}))$. The first inequality follows by taking S to be empty so that every point in $\mathcal{T}$ is an impostor, the second by taking $S = \mathrm{ID}(x, \mathcal{T})$ so that the only impostor is x itself. Returning to the Hamming ball example, it follows that $\mathrm{pidLength}(0^d, \mathcal{T}) = \log_2(d+1)$ where we take the empty set as the PID.

We present an alternate geometric view of pidLength, which generalizes naturally to other settings. A subcube C of $\{0,1\}^d$ is the set of points obtained by fixing some subset $S \subseteq [d]$ coordinates to values in $0, 1$. The sparsity of $\mathcal{T}$ in a subcube C is $\rho_{0,1}(\mathcal{T}, C) = |C|/|C \cap \mathcal{T}|$. The notation $C \ni x$ means that C contains x, hence $\min_{C \ni x}$ is the minimum over all C that contain x. One can show that for $x \in \mathcal{T}$, $\max_{C \ni x} \rho_{0,1}(\mathcal{T}, C) = 2^{d - \mathrm{pidLength}(x, \mathcal{T})}$, see appendix C for a proof. This characterization motivates using $2^{-\mathrm{pidLength}(x,\mathcal{T})}$ as an anomaly score: anomalies are points that lie in relatively sparse subcubes. Low scores come with a natural witness: a sparse subcube $\mathrm{PID}(x, \mathcal{T})$ containing relatively few points from $\mathcal{T}$.

2.2 The continuous setting

Now assume that all the coordinates are real-valued, and bounded. Without loss of generality, we may assume that they lie in the range $[0, 1]$, hence $\mathcal{T}$ is a collection of n points from $[0,1]^d$. An interval $I = [a, b], 0 \le a \le b \le 1$ is of length $\mathrm{len}(I) = b - a$. A subcube C is specified by a subset of co-ordinates S and intervals I_j for each $j \in S$. It consists of all points such that $x_j \in I_j$ for all $j \in S$. To simplify our notation, we let C be $I_1 \times I_2 \cdots \times I_d$ where $I_j = [0, 1]$ for $j \notin S$. Note that $\mathrm{vol}(C) = \Pi_j \mathrm{len}(I_j)$. Define the sparsity of $\mathcal{T}$ in C as $\rho(\mathcal{T}, C) = \mathrm{vol}(C)/|C \cap \mathcal{T}|$. $\mathrm{PIDScore}(x, T)$ is the maximum sparsity over all subcubes of $[0,1]^d$ containing x.

Definition 3. For $x \in \mathcal{T}$, let

$$\mathrm{PID}(x, \mathcal{T}) = \arg\max_{C \ni x} \rho(\mathcal{T}, C), \quad \mathrm{PIDScore}(x, \mathcal{T}) = \max_{C \ni x} \rho(\mathcal{T}, C).$$

To see the analogy to the Boolean case, define $\mathrm{pidLength}(x, \mathcal{T}) = -\log(\mathrm{PIDScore}(x, \mathcal{T}))$. Fix $C = \mathrm{PID}(x, \mathcal{T})$. Since $\mathrm{vol}(C) = \prod_{j \in [d]} \mathrm{len}(I_j)$, we can write

$$\mathrm{pidLength}(x, \mathcal{T}) = \log(|C \cap \mathcal{T}|/\mathrm{vol}(C)) = \sum_{j \in [d]} \log(1/\mathrm{len}(I_j)) + \log(|C \cap \mathcal{T}|). \tag{3}$$

This exposes the similarities to Equation (2): $C \cap \mathcal{T}$ is exactly the set of impostors for x, whereas $\sum_{j \in [d]} \log(1/\mathrm{len}(I_j))$ is the analog of $|S|$. In the boolean setting, we pay 1 for each coordinate from S, here the cost ranges in $[0, \infty)$ depending on the length of the interval. In the continuous setting, the $j \notin S$ iff $I_j = [0, 1]$ hence $\log(1/\mathrm{len}(I_j)) = 0$, hence we pay nothing for coordinates outside S. Restricting to an interval of length p costs $\log(1/p)$. If $p = 1/2$, we pay 1, which is analogous to the Boolean case where we pay 1 to cut the domain in half. This addresses the issue of having to pay more for higher precision. Note also that the definition is *scale-invariant* as multiplying a coordinate by a constant changes the volume of all subcubes by the same factor.

Other attributes: To handle attributes over a domain D, we need to specify what subsets of D are intervals and how we measure their length. For discrete attributes, it in natural to define $\mathrm{len}(I) = |I|/|D|$. When the domain is ordered intervals are naturally defined, for instance *months between April and September* is an interval of length $1/2$. We could also allow wraparound in intervals, say *months between November and March*. For unordered discrete values, the right definition of interval could be singleton sets, like *country = Brazil* or certain subsets, like *continent= the Americas*. The right choice will depend on the dataset. Our definition is flexible enough to handle this: We can make independent choices for each coordinate, subcubes and volumes are then defined as products, and PIDScore can be defined using definition 3.

3 The PIDForest algorithm

We do not how to compute PIDScore exactly, or even a provable approximation of it in a way that scales well with both d and n. The PIDForest algorithm described below is heuristic designed to approximate PIDScore. Like with iForest, the PIDForest algorithm builds an ensemble of decision trees, each tree is built using a sample of the data set and partitions the space into subcubes. However, the way the trees are constructed and the criteria by which a point is declared anomalous are very different. Each node of a tree corresponds to a subcube C, the children of C represent a disjoint partition of C along some axis $i \in [d]$ (iForest always splits C into two , here we allow for a finer partition). The goal is to have large variance in the sparsity of the subcubes. Recall that the sparsity of a subcube C with respect to a data set $\mathcal{T}$ is $\rho(C, \mathcal{T}) = \mathrm{vol}(C)/|C \cap \mathcal{T}|$. Ultimately, the leaves with large ρ values will point to the regions with the anomalies.

For each tree, we pick a random sample $P \subseteq \mathcal{T}$ of points, and use that subset to build the tree. Each node v in the tree corresponds to subcube $C(v)$, and a set of points $P(v) = C(v) \cap \mathcal{P}$. For the root, $C(v) = [0, 1]^d$ and $P(v) = \mathcal{P}$. At each internal node, we pick a coordinate $j \in [d]$, and breakpoints $t_1 \leq \cdots \leq t_{k-1}$ which partition I_j into k intervals, and split C into k subcubes. The number of partitions k is a hyper-parameter (taking $k < 5$ works well in practice). We then partition the points $P(v)$ into these subcube. The partitions stop when the tree reached some specified maximum depth or when $|P(v)| \leq 1$. The key algorithmic problem is how to choose the coordinate j and the breakpoints by which it should be partitioned. Intuitively we want to partition the cube into some sparse regions and some dense regions. This intuition is formalized next.

Let $I_j \subseteq [0, 1]$ be the projection of C onto coordinate i. Say the breakpoints are chosen so that we partition I_j into $I_j^1, \ldots, I_j^k$. This partitions C into $C^1, \ldots, C^k$ where the intervals corresponding to the other coordinates stay the same. We first observe that in any partition C, the sparsity of the subcubes weighted by the number of points is the same. Let $\mathrm{len}(I_j^i)/\mathrm{len}(I) = \mathrm{vol}(C^i)/\mathrm{vol}(C) = p_i$ and let $|P \cap C^i|/|P| = q_i$. Hence

$$\rho(C^i) = \mathrm{vol}(C^i)/|P \cap C^i| = (p_i \mathrm{vol}(I))/(q_i|P|) = (p_i \rho(C))/q_i.$$

Since a q_i fraction of points in P have sparsity $\rho(C_i)$, the expected sparsity for a randomly chosen point from P is

$$\sum_j q_j \rho(C_j) = \sum_j p_j \rho(C) = \rho(C).$$

In words, in any partition of C if we pick a point randomly from $P(v)$ and measure the sparsity of its subcube, on expectation we get $\rho(C)$. Recall that our goal is to split C into sparse and dense subcubes. Hence a natural objective is to maximize the *variance* in the sparsity:

$$\mathbf{Var}(\mathcal{P}, k) = \sum_j q_j(\rho(C_j) - \rho(C))^2 = \sum_j q_j \rho(C_j)^2 - \rho(C)^2. \tag{4}$$

In Appendix A, we show that this problem can be reduced to the problem of finding a k-histogram for a discrete function $f : [n] \to \mathbb{R}$, which minimizes the squared ℓ_2 error. This is a well-studied problem [9, 10, 11] and there an efficient one-pass streaming algorithm for computing near-optimal histograms due to Guha *et al.* [12]. We use this algorithm to compute the best split along each coordinate, and then choose the coordinate that offers the most variance reduction. This is the fundamental difference between PIDForest and iForest and its variants. PIDForest zooms on the coordinates with signal - on the coordinates where a split is most beneficial. We continue splitting until a certain predefined depth is reached, (or points are isolated). Each leaf is labeled with the sparsity of its subcube.

```
PIDForest Fit
Params: Num of trees $t$, Samples $m$, Max degree $k$, Max depth $h$.
Repeat $t$ times:
    Create root node $v$.
    Let $C(v) = [0,1]^d$, $P(v) \subseteq \mathcal{T}$ be a random subset of size $m$ .
    Split($v$)

Split($v$):
    For $i \in [d]$, compute the best split into $k$ intervals.
    Pick $i$ that maximizes variance, split $C$ along $i$ into $\{C_i\}_{i=1}^k$.
    For $i \in [k]$ create child $v_i$ s.t. $C(v_i) = C_i, P(v_i) = P(v) \cap C_i$.
    If depth($v_i$) $\leq h$ and $|P(v_i)| > 1$ then Split($v_i$).
    Else, set PIDScore($v_i$) = $\mathrm{vol}(C(v_i))/|P(v_i)|$.
```

Producing an anomaly score for each point is fairly straightforward. Say we want to compute a score for $y \in [0, 1]^d$. Each tree in the forest maps y to a leaf node v and gives it a score $\mathrm{PIDScore}(v)$. We take the 75% percentile score as our final score. (Any robust analog of the max will do).

Finding the optimal split for a node takes time $O(dm \log(m))$. This is repeated at most k^h times for each tree (typically much fewer since the trees we build tend to be unbalanced), and t times to create the forest. We typically choose $m \leq 200$, $k \leq 5$, $h \leq 10$ and $t \leq 50$.

Comparison to Isolation Forests: iForest repeatedly samples a set S of m points from $\mathcal{T}$ and builds a random tree with those points as leaves. The tree is built by choosing a random co-ordinate x_i, and a random value in its range about which to split. The intuition is that anomalous points will be easy to separate from the rest, and will be isolated at small depth. What kind of points are likely to be labeled anomalous by iForest?

In one direction, if a point is isolated at relatively low depth k in a tree, then it probably belongs in a sparse subcube. Indeed, a node at depth k corresponds to a subcube C of expected volume 2^{-k}, which is large for small k. The fact that the sample contains no points from C suggests that C is fairly sparse in it (this can be made precise using a VC-dimension argument).

Being in a sparse subcube is necessary but not sufficient. This is because iForest chooses which coordinate we split on as well as the breakpoint at random. Thus to be isolated at small depth frequently, a point needs to lie in an *abundant* number of low-density subspaces: picking splits at random should have a good chance of defining such a subspace. Requiring such an abundance of sparse subcubes can be problematic. Going back to the animals example, isolating white elephants is hard unless both Color and Type are used as attributes, as there is no shortage of elephants or white animals. Moreover, which attributes are relevant can depend on the point: weight might be irrelevant in isolating a white elephant, but it might be crucial to isolating a particularly large elephant. This causes iForest to perform poorly in the presence of irrelevant attributes, see for instance [5].

PIDForest circumvents this by explicitly seeking dimensions and splits that have large variance in sparsity. Attributes with little signal are unlikely to be chosen for splitting. For concrete examples, see Section 5. The tradeoff is that we incur a slightly higher cost at training time, the cost of prediction stays pretty much the same.

4 Real-world Datasets

We show that PIDForest performs favorably in comparison to several popular anomaly detection algorithms on real-world benchmarks. We select datasets from varying domains, and with different number of datapoints, percentage of anomalies and dimensionality. The code and data for all experiments is included in the supplementary.

Dataset Descriptions: The first set of datasets are classification datasets from the UCI [13] and openML repository [14]. Three of the datsets—*Thyroid*, *Mammography* and *Siesmic*—are naturally suited to anomaly detection as they are binary classification tasks where one of the classes has a much smaller occurrence rate (around 5%) and hence can be treated as anomalous. *Thyroid* and *Mammography* have real-valued attributes whereas *Siesmic* has categorical attributes as well. Three other datasets—*Satimage-2*, *Musk* and *Vowels*—are classification datasets with multiple classes, and we combine the classes and divide them into inliers and outliers as in [15]. Two of the datasets—*http* and *smtp*—are derived from the KDD Cup 1999 network intrusion detection task and we preprocess them as in [16]. These two datasets have have significantly more datapoints (about 500k and 100k respectively) and a smaller percentage of outliers (less than 0.5%).

The next set of real-world datasets—*NYC taxicab*, *CPU utilization*, *Machine temperature (M.T.)* and *Ambient temperature (A.T.)*—are time series datasets from the Numenta anomaly detection benchmark [17]. These are time series datasets which have been hand-labeled with anomalies rooted in real-world causes. The length of the time series is between 10k-20k, with about 10% of the timepoints marked as anomalous. We use the standard technique of *shingling* with a sliding window of width 10, hence each data point becomes a 10 dimensional vector holding 10 consecutive measurements from the time series. Detailed parameters of our datasets can be found in Table 2 in the Appendix.

Methodology: We compare PIDForestwith six popular anomaly detection algorithms: Isolation Forest (iForest), Robust Random Cut Forest (RRCF), one-class SVM (SVM), Local Outlier Factor (LOF), k-Nearest Neighbour (kNN) and Principal Component Analysis (PCA). We implement PIDForest in Python, it takes about 500 lines of code. For iForest, SVM and LOF we used the scikit-learn implementations, for kNN and PCA we used the implementations on PyOD [18] , and for RRCF we use the implementation from [19]. Except for RRCF, we run each algorithm with the default hyperparameter setting as varying the hyperparameters from their default values did not

Data set	PIDForest	iForest	RRCF	LOF	SVM	kNN	PCA
Thyroid	**0.876** ± **0.013**	0.819 ± 0.013	0.739± 0.004	0.737	0.547	0.751	0.673
Mammo.	0.840 ± 0.010	0.862 ± 0.008	0.830 ± 0.002	0.720	0.872	0.839	**0.886**
Siesmic	**0.733** ± **0.006**	0.698 ± 0.004	0.701 ± 0.004	0.553	0.601	**0.740**	0.682
Satimage	0.987 ± 0.001	**0.994** ± **0.001**	**0.991** ± **0.002**	0.540	0.421	0.936	0.977
Vowels	0.741 ± 0.008	0.736 ± 0.026	0.813± 0.007	0.943	0.778	**0.975**	0.606
Musk	**1.000** ± **0.000**	**0.998** ± **0.003**	0.998 ± 0.000	0.416	0.573	0.373	**1.000**
http	0.986 ± 0.004	**1.000** ± **0.000**	0.993 ± 0.000	0.353	0.994	0.231	0.996
smtp	**0.923** ± **0.003**	0.908 ± 0.003	0.886 ± 0.017	0.905	0.841	0.895	0.823
NYC	0.564 ± 0.004	0.550 ± 0.005	0.543 ± 0.004	0.671	0.500	**0.697**	0.511
A.T.	**0.810** ± 0.005	0.780 ± 0.006	0.695 ±0.004	0.563	0.670	0.634	0.792
CPU	**0.935** ± **0.003**	0.917 ± 0.002	0.785 ± 0.002	0.560	0.794	0.724	0.858
M.T.	0.813 ± 0.006	0.828 ± 0.002	0.7524 ± 0.003	0.501	0.796	0.759	**0.834**

Table 1: Results on real-world datasets. We bold the algorithm(s) which get the best AUC.

change the results significantly. For RRCF, we use 500 trees instead of the default 100 since it yielded significantly better performance. For PIDForest, we fix the hyperparameters of depth to 10, number of trees to 50, and the number of samples used to build each tree to 100. We use the area under the ROC curve (AUC) as the performance metric. As iForest, PIDForest and RRCF are randomized, we repeat these algorithms for 5 runs and report the mean and standard deviation. SVM, LOF, kNN and PCA are deterministic, hence we report a single AUC number for them.

Results: We report the results in Table 1. PIDForest is the top performing or jointly top performing algorithm in 6 out of the 12 datasets, and iForest, kNN and PCA are top performing or jointly top performing algorithms in 3 datasets each. Detailed ROC performance curves of the algorithms are given in Fig. 3 and 4. While the running time of our fit procedure is slower than iForest, it is comparable to RRCF and faster than many other methods. The time for predict is similar to iForest, and faster than RRCF. Even our vanilla python implementation only takes about 5 minutes to fit a model to our largest dataset which has half a million points.

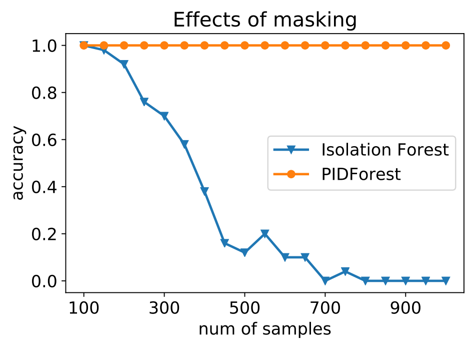
Recall from Section 3 that PIDForest differs from iForest in two ways, it optimizes for the axis to split on, and secondly, it uses sparsity instead of depth as the anomaly measure. To further examine the factors which contribute to the favorable performance of PIDForest, we do an ablation study through two additional experiments.

Choice of split: Optimizing for the choice of split rather than choosing one at random seems valuable in the presence of irrelevant dimensions. To measure this effect, we added 50 additional random dimensions sampled uniformly in the range $[0, 1]$ to two low-dimensional datasets from Table 1—*Mammography* and *Thyroid* (both datasets are 6 dimensional). In the *Mammography* dataset, PIDForest (and many other algorithms as well) suffers only a small 2% drop in performance, whereas the performance of iForest drops by 15%. In the *Thyroid* dataset, the performance of all algorithms drops appreciably. However, PIDForest has a 13% drop in performance, compared to a 20% drop for iForest. The detailed results are given in Table 3 in the Appendix.

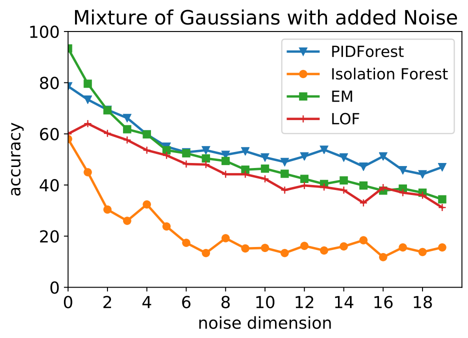
Using sparsity instead of depth: In this experiment, we test the hypothesis that the sparsity of the leaf is a better anomaly score than depth for the PIDForest algorithm. The performance of PIDForest deteriorates noticeably with depth as the score, the AUC for *Thyroid* drops to 0.847 from 0.876, while the AUC for *Mammography* drops to 0.783 from 0.840.

5 Synthetic Data

We compare PIDForest with popular anomaly detection algorithms on synthetic benchmarks. The first set of experiments checks how the algorithms handle duplicates in the data. The second set of uses data from a mixture of Gaussians, and highlights the importance of the choice of coordinates to



(a) 970 points are drawn from $\{-1, 1\}^{10}$ and 30 are all zeros vector. The $y-$axis measures the fraction of the 30 reported in the top 5% of anomalies.



(b) $y-$axis measures how many of the 100 true anomalies were reported by the algorithm in the top 100 anomalies.

Figure 1: Synthetic experiments on masked anomalies and Gaussian data.

split in PIDForest. The third set of experiments tests the ability of the algorithm to detect anomalies in time-series (see Appendix D). In all these experiments, PIDForest outperforms prior art.

Masking and sample size: It is often the case that anomalies repeat multiple times in the data. This phenomena is called *masking* and is a challenge for many algorithms. iForest counts on sampling to counter masking: not too many repetitions occur in the sample. But the performance is sensitive to the sampling rate, see [4, 5]. To demonstrate it, we create a data set of 1000 points in 10 dimensions. 970 of these points are sampled randomly in $\{-1, 1\}^{10}$ (hence most of these points are unique). The remaining 30 are the all-zeros vector, these constitute a masked anomaly. We test if the zero points are declared as anomalies by PIDForest and iForestunder varying sample sizes. The results are reported in Fig. 1a. Whereas PIDForest consistently reported these points as anomalies, the performance of iForest heavily depends on the sample size. When it is small, then masking is negated and the anomalies are caught, however the points become hard to isolate when the sample size increases.

Mixtures of Gaussians and random noise: We use a generative model where the ground truth anomalies are the points of least likelihood. The first two coordinates of the 1000 data points are sampled by taking a mixture of two 2$-$dimensional Gaussians with different means and covariances (the eigenvalues are $\{1, 2\}$, the eigenvectors are chosen at random). The remaining d dimensions are sampled uniformly in the range $[-2, 2]$. We run experiments varying d from 0 to 19. In each case we take the 100 points smallest likelihood points as the ground truth anomalies. For each algorithm we examine the 100 most anomalous points and calculate how many of these belong to the ground truth.

We compare PIDForest with Isolation Forest, Local Outlier Factor (LOF) and with an algorithm that uses EM to fit the data to a mixture of Gaussians. The results are reported in Fig. 1b. Note that even without noise ($d = 0$) PIDForest is the best generic algorithm. As the number of noisy dimensions increase PIDForest focuses on the dimensions with signal, so it performs better. Some observations:

1. The performance of iForest degrades rapidly with d, once $d \geq 6$ it effectively outputs a random set. Noticeably, PIDForest performs better than iForest even when $d = 0$. This is mainly due to the points between the two centers being classified as normal by iForest. PIDForest classifies them correctly as anomalous even though they are assigned to leaves that are deep in the tree.
2. The EM algorithm is specifically designed to fit the data to a mixture of two Gaussians, so it does best for small or zero noise. As d increases the data distribution is further away from a mixture. PIDForest matches or improves once $d > 2$.
3. Local Outlier Factor's reasonable performance depends crucially on the fact that the noise is of the same scale as the Gaussians. If we change the scale of the noise (which could happen if the measuring unit changes), then the performance of LOF drops significantly even for $d = 1$.

Conclusions. We believe that PIDForest is arguably the best off-the-shelf algorithm for anomaly detection on a large, heterogenous dataset. It inherits many of the desirable features of Isolation Forests, while also improving on it in important ways. Developing provable and scalable approximations to PIDScore is an interesting algorithmic challenge.

References

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

[2] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2013. ISBN 9783319475783.

[3] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 413–422, 2008.

[4] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2712–2721, 2016.

[5] Tharindu R. Bandaragoda, Kai Ming Ting, David W. Albrecht, Fei Tony Liu, Ye Zhu, and Jonathan R. Wells. Isolation-based anomaly detection using nearest-neighbor ensembles. *Computational Intelligence*, 34(4):968–998, 2018.

[6] Andrew F Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, pages 16–21. ACM, 2013.

[7] S.A. Goldman and M.J. Kearns. On the complexity of teaching. *J. Comput. Syst. Sci.*, 50 (1):20–31, February 1995. ISSN 0022-0000. doi: 10.1006/jcss.1995.1003. URL http://dx.doi.org/10.1006/jcss.1995.1003.

[8] Eyal Kushilevitz, Nathan Linial, Yuri Rabinovich, and Michael E. Saks. Witness sets for families of binary vectors. *J. Comb. Theory, Ser. A*, 73(2):376–380, 1996. doi: 10.1006/jcta.1996.0031. URL https://doi.org/10.1006/jcta.1996.0031.

[9] Nick Koudas, S Muthukrishnan, and Divesh Srivastava. Optimal histograms for hierarchical range queries. In *PODS*, pages 196–204, 2000.

[10] Anna C Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398. ACM, 2002.

[11] Sudipto Guha, Piotr Indyk, S Muthukrishnan, and Martin J Strauss. Histogramming data streams with fast per-item processing. In *International Colloquium on Automata, Languages, and Programming*, pages 681–692. Springer, 2002.

[12] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31(1):396–438, March 2006. ISSN 0362-5915. doi: 10.1145/1132863.1132873. URL http://doi.acm.org/10.1145/1132863.1132873.

[13] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[14] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

[15] Charu C Aggarwal and Saket Sathe. Theoretical foundations and algorithms for outlier ensembles. *ACM Sigkdd Explorations Newsletter*, 17(1):24–47, 2015.

[16] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

[17] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[18] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588*, 2019.

[19] Tharindu R. Bandaragoda, Kai Ming Ting, David W. Albrecht, Fei Tony Liu, Ye Zhu, and Jonathan R. Wells. rrcf: Implementation of the robust random cut forest algorithm for anomaly detection on streams. *Journal of Open Source Software*, 4(35):1336, 2019.

[20] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[21] Animesh Patcha and Jung-Min Jerry Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51:3448–3470, 2007.

[22] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM. ISBN 1-58113-217-4. doi: 10.1145/342009.335388. URL http://doi.acm.org/10.1145/342009.335388.

[23] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, pages 15–26, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44037-2. URL http://dl.acm.org/citation.cfm?id=645806.670167.

[24] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335437. URL http://doi.acm.org/10.1145/335191.335437.

[25] Tao Shi and Steve Horvath. Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics*, 15(1):118–138, 2006.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996. URL http://dl.acm.org/citation.cfm?id=3001460.3001507.

[27] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998. ISSN 0163-5808. doi: 10.1145/276305.276314. URL http://doi.acm.org/10.1145/276305.276314.

[28] Balas K. Natarajan. *Machine learning: A theoretical approach*. Morgan Kaufmann Publishers, Inc., 1991.

[29] Martin Anthony, Graham Brightwell, Dave Cohen, and John Shawe-Taylor. On exact specification by examples. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 311–318, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130420. URL http://doi.acm.org/10.1145/130385.130420.

[30] Avi Wigderson and Amir Yehudayoff. Population recovery and partial identification. *Mach. Learn.*, 102(1):29–56, January 2016. ISSN 0885-6125. doi: 10.1007/s10994-015-5489-9. URL http://dx.doi.org/10.1007/s10994-015-5489-9.

A Finding optimal splits efficiently

In this section we present the algorithm used to split a single dimension. The problem is easy in the discrete setting, so we focus on the continuous case. We first restate the problem we which solve.

For interval I, a k-interval partition of I is a partition into a set $\{I_1, \ldots, I_k\}$ of disjoint intervals.

Optimal k-split: Given a set $\mathcal{P}$ of m points $x_1 \leq \cdots \leq x_m$ from an interval I, and a parameter k, find a k-interval partition of I where I_i contains m_i points from $\mathcal{P}$ so as to maximize

$$\operatorname{cost}(\mathcal{P}, k) = \sum_{i=1}^{k} m_i \rho(I_i)^2 \tag{5}$$

where $\rho(E_i) = \operatorname{len}(E_i)/m_i$. Comparing this to Equation (4), we have

$$\operatorname{cost}(\mathcal{P}, k) = m\mathbf{Var}(\mathcal{P}, k) + m\rho(C)^2.$$

The second term does not depend on the partition, so we can drop it without changing the optimum.

By shifting and scaling, we will assume that the bounding interval $I = [0, 1]$. We also assume the x_is are distinct (this is not needed, but eases notation). To simplify matters, we restrict to those intervals whose start and end points are either $e_0 = 0$, $e_m = 1$, or $e_i = (x_i + x_{i+1})/2$ for $i \in [m-1]$. This avoids issues that might arise from the precision of the end points, and from having points lie on the boundary of two intervals (to which interval should they belong?). It reduces the search space of intervals to $O(m^2)$. One can use dynamic programming to give an $O(m^2 k)$ time and $O(mk)$ space algorithm to solve the problem exactly. However this is too costly, since the procedure runs in an inner loop of PIDForest Fit. Rather we show the problem reduces to that of computing optimal k-histograms for an array, for which there are efficient streaming approximation algorithms known.

First some notation. An interval in $[m]$ is $J = \{i : \ell \leq i \leq u\}$. Given a function $f : [m] \to \mathbb{R}$ and an interval $J \subseteq [m]$, let $\bar{f}(J) = \sum_{i \in J} f(i)/|J|$ denote the average of f over the interval J. A k-interval partition of $[m]$ is a set of pairwise disjoint intervals $\{J_1, \ldots, J_k\}$ whose union is $[m]$. Given $j \in [m]$, let $J(j) \in \mathbb{I}$ denote the interval containing it.

Optimal k-histograms: Given $f : [m] \to \mathbb{R}$, find a k-interval partition of $[m]$ which maximizes

$$\operatorname{cost}(f, k) = \sum_{i \in [k]} |J_i| (\bar{f}(J_i))^2. \tag{6}$$

Consider the k-histogram h where we approximate f by its average over each interval J_i. Maximizing $\operatorname{cost}(f, k)$ is equivalent to minimizing the squared error of the histogram since

$$\sum_{j \in m} (\bar{f}(I(j)) - f(j))^2 = \sum_{j \in [m]} f(j)^2 - \sum_{i \in [k]} |J_i| (\bar{f}(J_i))^2,$$

hence the name.

We now give the reduction from computing k-splits to k-histograms. For each $i \in [m]$, let $f(i) = e_i - e_{i-1}$ denote the length of the interval $[e_{i-1}, e_i]$ which contains x_i. There is now a natural correspondence between the discrete interval $J_{\ell,u} = \{\ell, \cdots, u\}$ and the continuous interval $I_{\ell,u} = [e_{\ell-1}, e_u]$ which contains the points $\{x_\ell, \cdots, x_u\}$ from $\mathcal{P}$, where

$$\bar{f}(J_{\ell,u}) = \frac{\sum_{i=\ell}^{u}(e_i - e_{i-1})}{u - \ell + 1} = \frac{e_u - e_{\ell-1}}{u - \ell + 1} = \rho(I_{\ell,u})$$

Thus a k interval partition of $[m]$ translates to a k-interval partition of I, with objective function

$$\operatorname{cost}(f, k) = \sum_{i \in [k]} |J_i| (\bar{f}(J_i))^2 = \sum_{i \in [k]} |\mathcal{P} \cap I_i| \rho(I_i)^2 = \operatorname{cost}(\mathcal{P}, k).$$

An efficient streaming algorithm for computing approximately optimal k-histograms is given by Guha *et al.* [12], which requires space $O(m + k^2)$ and time $O(m + k^3)$ (we set their parameter ε to 0.1). We use their algorithm in Fit procedure to find good splits.

B Relation to prior work

Anomaly detection is a wide area with a number of surveys and books [1] [20] [21]. In the following we discuss the basic techniques and how they relate to our work.

Proximity based methods: Many works detect anomalies by computing distances to nearest neighbors in various methods [22][5] [23],[24], [25]. There are many different methods with pros and cons, which we do not discuss here. The important common feature is that these algorithms utilize a notion of *distance*, typically Euclidean, as a proxy for similarly across points. Once the notion of distance is established it is used find the closest neighbor, or the close k neighbors, it is used to define a ball of a given radius around a point and so on. There are few major drawbacks with these approaches. First, as the number of dimensions increases the distances between points become more and more similar, and notion of a local neighborhood looses its meaning. But more importantly, while in some contexts distance is naturally defined and then a crisp notion of an anomaly could be obtained, we claim that distance is an ill suited notion to the general case where some columns are categorical and some numeric and where we different columns employ different units.

Density based algorithms: Density has been used as a criterion for several works on clustering. For instance, DBSCAN [26] builds clusters from maximal connected components out of dense balls (in a chosen metric) of a certain radius. Outliers are points that do not belong in any such cluster. The work of Agrawal *et al.* [27] builds clusters that can be expressed as connected unions of subcubes. A key difference from these works is that we do not attempt to discover the structure (such as clusters) in the normal data. Further, rather than only consider balls/subcubes at a particular scale (this scale is a hyperparameter), our algorithms attempt to minimize density over subcubes at all scales.

Isolation Forest: Perhaps the most relevant algorithm is the widely used Isolation Forest [3] and its important to understand the two fundamental differences between the algorithms. The first is that [3] chooses *randomly* which column to split. We choose columns that could be split well. This randomness causes Isolation Forest's accuracy to degrade when extra dimensions are introduced, (this is a well known issue, see [4], [5]). Robust Isolation Forest [4] deals with this by choosing a column based on the size of its range. This makes the algorithm scale-sensitive and results change based on the units with which the data is reported. PIDForest on the other hand zooms on the columns which have the most *signal*, and thus can overcome uniform noise. The other difference is that Isolation Forest insists on full isolation and then computes the average leaf depth. This means it is sensitive to duplicates or near duplicates. See Section 5 where we demonstrate these points via experiments.

IDs and PIDs: The notion of IDs for a point is natural and has been studied in the computational learning literature under various names: the teaching dimension of a hypothesis class [7], discriminant [28], specifying set [29] and witness set [8]. The terminology of IDs is from Wigderson and Yehudayoff [30], so is the notion of partial identification and impostors. Our definition of $\mathrm{pidLength}$ and its application to anomaly detection is new. [30] consider PIDs, but with a different goal in mind (to minimize the depth of a certain graph constructed using the PID relation).

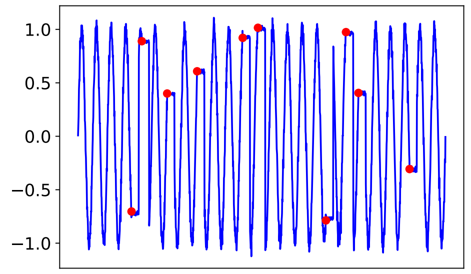
C Proofs

Lemma 4. For $x \in \mathcal{T}$,
$$\max_{C \ni x} \rho_{0,1}(\mathcal{T}, C) = 2^{d - \mathrm{pidLength}(x, \mathcal{T})}.$$

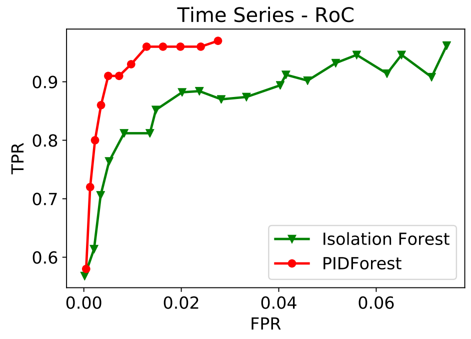
Proof: Given $S \subseteq [d]$, let
$$C_x(S) = \{y \in \{0,1\}^d \ s.t.\ y_S = x_S\}$$
be the subcube consisting of $2^{d-|S|}$ points that agree with x on S. Since $C_x(S) \cap \mathcal{T} = \mathrm{Imp}(x, \mathcal{T}, S)$,
$$\rho_{0,1}(\mathcal{T}, C_x(S)) = \frac{|C_x(S)|/|C_x(S) \cap \mathcal{T}|}{|\mathcal{T}|} = \frac{2^{d-|S|}}{|\mathrm{Imp}(x, \mathcal{T}, S)|} = 2^{d-(|S|+\log_2(|\mathrm{Imp}(x,\mathcal{T},S)|))}$$
Iterating over all subsets S gives all the subcubes that contain x. The RHS is minimized by taking $S = \mathrm{PID}(x, \mathcal{T})$ by Definition 2. This gives the desired result. □

D Experiments on Synthetic Time Series Data

Time Series: We create a periodic time series using a simple $\sin$ function with a period of 40. We choose 10 locations and fix the value for the next 20 points following each of these locations.



(a) An example of a time series, the red dots represent the beginning of an anomalous segment.



(b) Performance comparison on time series data.

Figure 2: Synthetic experiments on time series data.

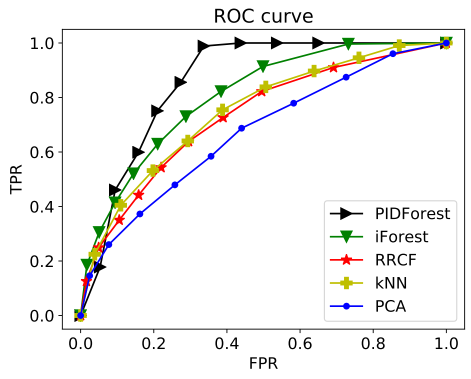
Data set	n	d	#outliers (%)
Thyroid	7200	6	534 (7.42%)
Mammography (Mammo.)	11183	6	250 (2.32%)
Siesmic	2584	15	170 (6.5%)
Satimage-2	5803	36	71 (1.2%)
Vowels	1456	12	50 (3.4%)
Musk	3062	166	97 (3.2%)
http	567479	3	2211 (0.4%)
smtp	95156	3	30 (0.03%)
NYC taxicab	10321	10	1035 (10%)
Ambient Temperature (A.T.)	7267	10	726 (10%)
CPU utilization	18050	10	1499 (8.3%)
Machine temperature (M.T.)	22695	10	2268 (10%)

Table 2: Details of real-world datasets. The first 8 datasets are derived from classification tasks, and the last 4 are from time series with known anomalies.

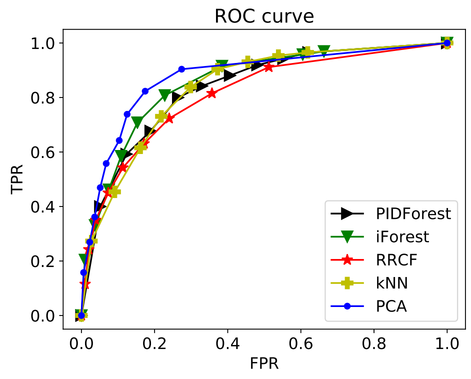
These regions are the anomalies. Finally we add small Gaussian noise to the series. See Fig. 2a for an example. As in Section 4, we shingle the time series with a window of 10. Fig. 2b shows the ROC curve (true positive rate (TPR) vs. false positive rate (FPR)), averaged over 10 runs. Since all dimensions are a priori identical, choosing splits at random seems natural. So we expect iForest to perform well, and indeed it achieves a precision of almost 1 while catching 5 out of the 10 anomalies. But iForest however struggles to catch all anomalies, and PIDForest has a significantly better precision for high recall.

Data set	PIDForest	iForest	RRCF	LOF	SVM	kNN	PCA
Thyroid*	**0.751 ± 0.035**	0.641 ± 0.023	0.530 ± 0.005	0.492	0.494	0.495	0.614
Mammography*	0.829 ± 0.016	0.722 ± 0.016	0.797 ± 0.013	0.628	**0.872**	0.817	0.768

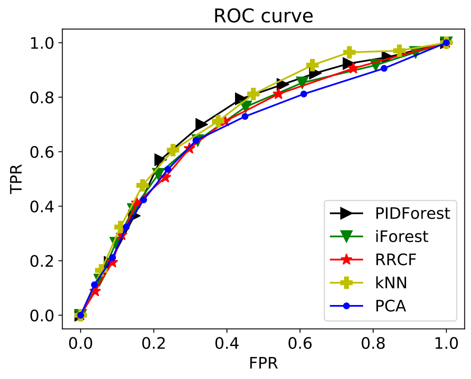
Table 3: For the first two datasets in Table 1 we add 50 noisy dimensions to examine the performance of algorithms in the presence of irrelevant attributes. We bold the algorithm(s) which get the best AUC, up to statistical error.



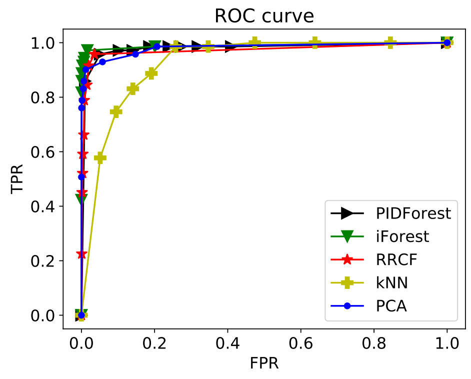
(a) ROC curve for *Thyroid* dataset.



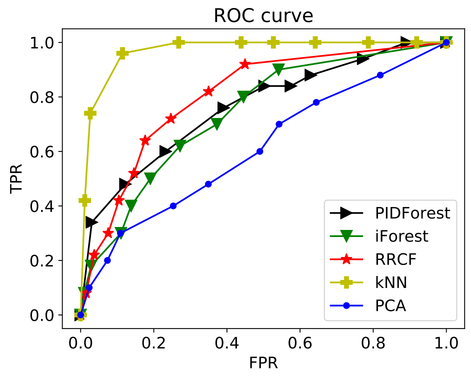
(b) ROC curve for *Mammography* dataset.



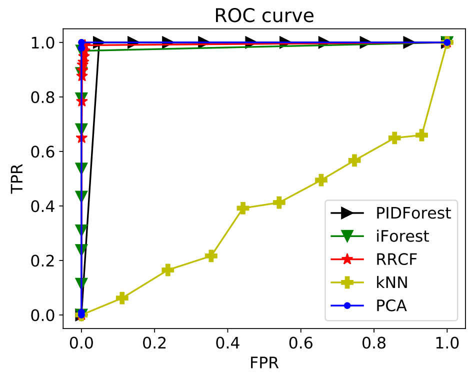
(c) ROC curve for *Siesmic* dataset.



(d) ROC curve for *Satimage-2* dataset.

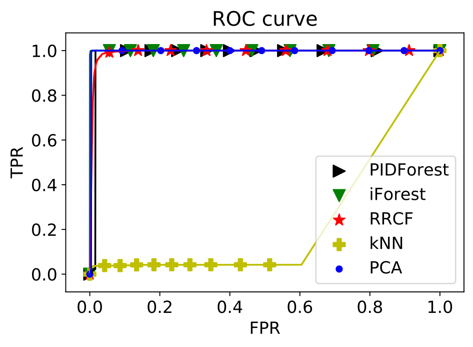


(e) ROC curve for *Vowels* dataset.

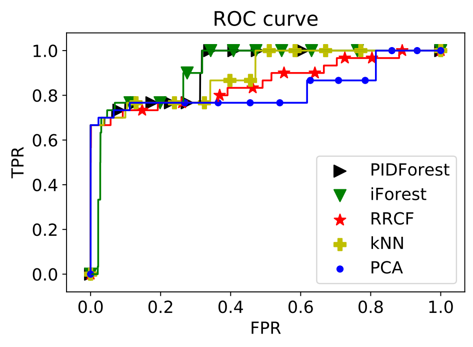


(f) ROC curve for *Musk* dataset.

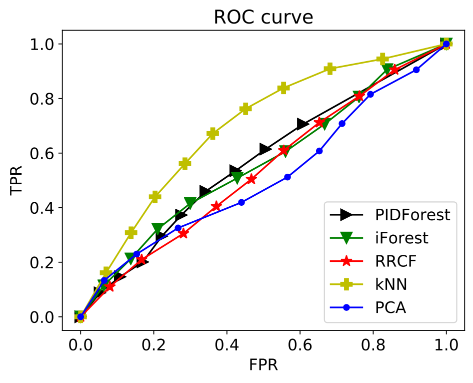
Figure 3: ROC curves for the first six datasets from Table 1. For visual clarity, we omit LOF and SVM which did not perform as well as the other algorithms.



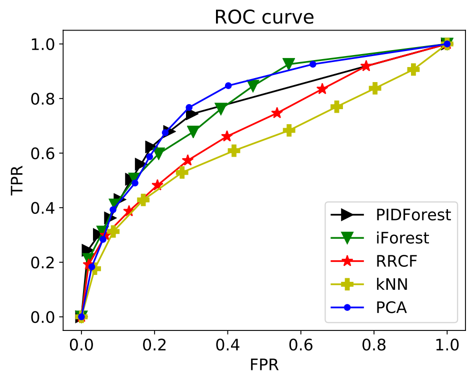
(a) ROC curve for *http* dataset.



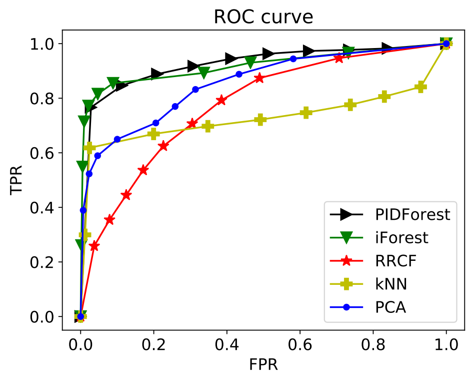
(b) ROC curve for *smtp* dataset.



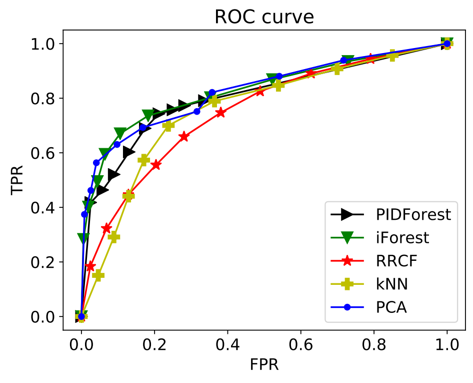
(c) ROC curve for *NYC taxi* dataset.



(d) ROC curve for *Ambient temperature* dataset.



(e) ROC curve for *CPU utilization* dataset.



(f) ROC curve for *Machine temperature* dataset.

Figure 4: ROC curves for the last six datasets from Table 1. For visual clarity, we omit LOF and SVM which did not perform as well as the other algorithms.