Isolation Forest

Conference Paper · January 2009

DOI: 10.1109/ICDM.2008.17 · Source: IEEE Xplore

3 authors, including:

Fei Tony Liu
Monash University (Australia)
17 PUBLICATIONS **1,276** CITATIONS
SEE PROFILE

Zhi-Hua Zhou
Nanjing University
459 PUBLICATIONS **30,420** CITATIONS
SEE PROFILE

Isolation Forest

Fei Tony Liu, Kai Ming Ting
Gippsland School of Information Technology
Monash University, Victoria, Australia
{tony.liu},{kaiming.ting}@infotech.monash.edu.au

Zhi-Hua Zhou
National Key Laboratory
for Novel Software Technology
Nanjing University, Nanjing 210093, China
zhouzh@lamda.nju.edu.cn

Abstract

Most existing model-based approaches to anomaly detection construct a profile of normal instances, then identify instances that do not conform to the normal profile as anomalies. This paper proposes a fundamentally different model-based method that explicitly isolates anomalies instead of profiles normal points. To our best knowledge, the concept of isolation has not been explored in current literature. The use of isolation enables the proposed method, iForest, to exploit sub-sampling to an extent that is not feasible in existing methods, creating an algorithm which has a linear time complexity with a low constant and a low memory requirement. Our empirical evaluation shows that iForest performs favourably to ORCA, a near-linear time complexity distance-based method, LOF and Random Forests in terms of AUC and processing time, and especially in large data sets. iForest also works well in high dimensional problems which have a large number of irrelevant attributes, and in situations where training set does not contain any anomalies.

1 Introduction

Anomalies are data patterns that have different data characteristics from normal instances. The detection of anomalies has significant relevance and often provides critical actionable information in various application domains. For example, anomalies in credit card transactions could signify fraudulent use of credit cards. An anomalous spot in an astronomy image could indicate the discovery of a new star. An unusual computer network traffic pattern could stand for an unauthorised access. These applications demand anomaly detection algorithms with high detection performance and fast execution.

Most existing model-based approaches to anomaly detection construct a profile of normal instances, then identify instances that do not conform to the normal profile as anomalies. Notable examples such as statistical methods [11], classification-based methods [1], and clustering-based methods [5] all use this general approach. Two major drawbacks of this approach are: (i) the anomaly detector is optimized to profile normal instances, but not optimized to detect anomalies—as a consequence, the results of anomaly detection might not be as good as expected, causing too many false alarms (having normal instances identified as anomalies) or too few anomalies being detected; (ii) many existing methods are constrained to low dimensional data and small data size because of their high computational complexity.

This paper proposes a different type of model-based method that explicitly isolates anomalies rather than profiles normal instances. To achieve this, our proposed method takes advantage of two anomalies' quantitative properties: i) they are the minority consisting of fewer instances and ii) they have attribute-values that are very different from those of normal instances. In other words, anomalies are 'few and different', which make them more susceptible to isolation than normal points. We show in this paper that a tree structure can be constructed effectively to isolate every single instance. Because of their susceptibility to isolation, anomalies are isolated closer to the root of the tree; whereas normal points are isolated at the deeper end of the tree. This isolation characteristic of tree forms the basis of our method to detect anomalies, and we call this tree Isolation Tree or iTree.

The proposed method, called Isolation Forest or iForest, builds an ensemble of iTrees for a given data set, then anomalies are those instances which have short average path lengths on the iTrees. There are only two variables in this method: the number of trees to build and the sub-sampling size. We show that iForest's detection performance converges quickly with a very small number of trees, and it only requires a small sub-sampling size to achieve high detection performance with high efficiency.

Apart from the key difference of isolation versus profiling, iForest is distinguished from existing model-based

[11, 1, 5], distance-based [6] and density-based methods [4] in the follow ways:

- The isolation characteristic of iTrees enables them to build partial models and exploit sub-sampling to an extent that is not feasible in existing methods. Since a large part of an iTree that isolates normal points is not needed for anomaly detection; it does not need to be constructed. A small sample size produces better iTrees because the swamping and masking effects are reduced.

- iForest utilizes no distance or density measures to detect anomalies. This eliminates major computational cost of distance calculation in all distance-based methods and density-based methods.

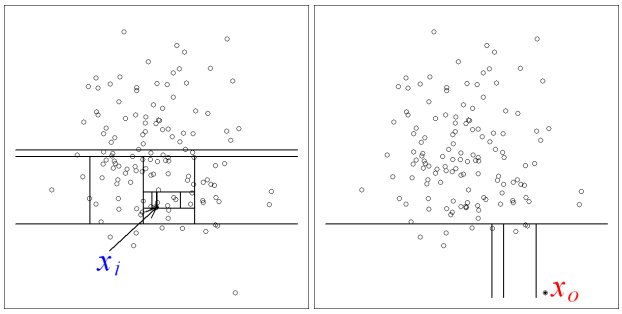
- iForest has a linear time complexity with a low constant and a low memory requirement. To our best knowledge, the best-performing existing method achieves only approximate linear time complexity with high memory usage [13].

- iForest has the capacity to scale up to handle extremely large data size and high-dimensional problems with a large number of irrelevant attributes.

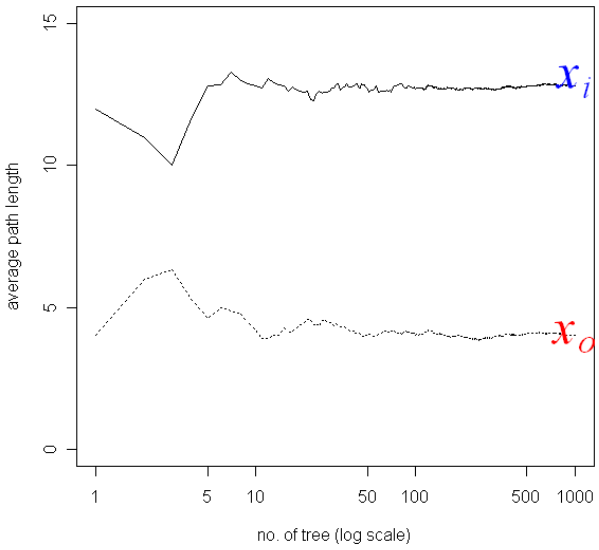
This paper is organised as follows: In Section 2, we demonstrate isolation at work using an iTree that recursively partitions data. A new anomaly score based on iTrees is also proposed. In Section 3, we describe the characteristic of this method that helps to tackle the problems of swamping and masking. In Section 4, we provide the algorithms to construct iTrees and iForest. Section 5 empirically compares this method with three state-of-the-art anomaly detectors; we also analyse the efficiency of the proposed method, and report the experimental results in terms of AUC and processing time. Section 6 provides a discussion on efficiency, and Section 7 concludes this paper.

2 Isolation and Isolation Trees

In this paper, the term *isolation* means 'separating an instance from the rest of the instances'. Since anomalies are 'few and different' and therefore they are more susceptible to isolation. In a data-induced random tree, partitioning of instances are repeated recursively until all instances are isolated. This random partitioning produces noticeable shorter paths for anomalies since (a) the fewer instances of anomalies result in a smaller number of partitions – shorter paths in a tree structure, and (b) instances with distinguishable attribute-values are more likely to be separated in early partitioning. Hence, when a forest of random trees collectively produce shorter path lengths for some particular points, then they are highly likely to be anomalies.



(a) Isolating x_i (b) Isolating x_o



(c) Average path lengths converge

Figure 1. Anomalies are more susceptible to isolation and hence have short path lengths. Given a Gaussian distribution (135 points), (a) a normal point x_i requires twelve random partitions to be isolated; (b) an anomaly x_o requires only four partitions to be isolated. (c) averaged path lengths of x_i and x_o converge when the number of trees increases.

To demonstrate the idea that anomalies are more susceptible to isolation under random partitioning, we illustrate an example in Figures 1(a) and 1(b) to visualise the random partitioning of a normal point versus an anomaly. We observe that a normal point, x_i, generally requires more partitions to be isolated. The opposite is also true for the anomaly point, x_o, which generally requires less partitions to be isolated. In this example, partitions are generated by randomly selecting an attribute and then randomly selecting

a split value between the maximum and minimum values of the selected attribute. Since recursive partitioning can be represented by a tree structure, the number of partitions required to isolate a point is equivalent to the path length from the root node to a terminating node. In this example, the path length of x_i is greater than the path length of x_o.

Since each partition is randomly generated, individual trees are generated with different sets of partitions. We average path lengths over a number of trees to find the expected path length. Figure 1(c) shows that the average path lengths of x_o and x_i converge when the number of trees increases. Using 1000 trees, the average path lengths of x_o and x_i converge to 4.02 and 12.82 respectively. It shows that anomalies are having path lengths shorter than normal instances.

Definition : **Isolation Tree.** Let T be a node of an isolation tree. T is either an external-node with no child, or an internal-node with one test and exactly two daughter nodes (T_l,T_r). A test consists of an attribute q and a split value p such that the test $q < p$ divides data points into T_l and T_r.

Given a sample of data $X = \{x_1, ..., x_n\}$ of n instances from a d-variate distribution, to build an isolation tree (iTree), we recursively divide X by randomly selecting an attribute q and a split value p, until either: (i) the tree reaches a height limit, (ii) $|X| = 1$ or (iii) all data in X have the same values. An iTree is a *proper binary tree*, where each node in the tree has exactly zero or two daughter nodes. Assuming all instances are distinct, each instance is isolated to an external node when an iTree is fully grown, in which case the number of external nodes is n and the number of internal nodes is $n - 1$; the total number of nodes of an iTrees is $2n - 1$; and thus the memory requirement is bounded and only grows linearly with n.

The task of anomaly detection is to provide a ranking that reflects the degree of anomaly. Thus, one way to detect anomalies is to sort data points according to their path lengths or anomaly scores; and anomalies are points that are ranked at the top of the list. We define path length and anomaly score as follows.

Definition : **Path Length** $h(x)$ of a point x is measured by the number of edges x traverses an iTree from the root node until the traversal is terminated at an external node.

An anomaly score is required for any anomaly detection method. The difficulty in deriving such a score from $h(x)$ is that while the maximum possible height of iTree grows in the order of n, the average height grows in the order of $\log n$ [7]. Normalization of $h(x)$ by any of the above terms is either not bounded or cannot be directly compared.

Since iTrees have an equivalent structure to Binary Search Tree or BST (see Table 1), the estimation of average $h(x)$ for external node terminations is the same as the unsuccessful search in BST. We borrow the analysis from BST to estimate the average path length of iTree. Given a data set of n instances, Section 10.3.3 of [9] gives the average path length of unsuccessful search in BST as:

iTree	BST
Proper binary trees	Proper binary trees
External node termination	Unsuccessful search
Not applicable	Successful search

Table 1. List of equivalent structure and operations in iTree and Binary Search Tree (BST)

$$c(n) = 2H(n-1) - (2(n-1)/n), \tag{1}$$

where $H(i)$ is the harmonic number and it can be estimated by $\ln(i)$ + 0.5772156649 (Euler's constant). As $c(n)$ is the average of $h(x)$ given n, we use it to normalise $h(x)$. The anomaly score s of an instance x is defined as:

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

where $E(h(x))$ is the average of $h(x)$ from a collection of isolation trees. In Equation (2):

- when $E(h(x)) \to c(n)$, $s \to 0.5$;
- when $E(h(x)) \to 0$, $s \to 1$;
- and when $E(h(x)) \to n - 1$, $s \to 0$.

s is monotonic to $h(x)$. Figure 2 illustrates the relationship between $E(h(x))$ and s, and the following conditions applied where $0 < s \leq 1$ for $0 < h(x) \leq n - 1$. Using the anomaly score s, we are able to make the following assessment:

- (a) if instances return s very close to 1, then they are definitely anomalies,
- (b) if instances have s much smaller than 0.5, then they are quite safe to be regarded as normal instances, and
- (c) if all the instances return $s \approx 0.5$, then the entire sample does not really have any distinct anomaly.

A contour of anomaly score can be produced by passing a lattice sample through a collection of isolation trees, facilitating a detailed analysis of the detection result. Figure 3 shows an example of such a contour, allowing a user to visualise and identify anomalies in the instance space. Using the contour, we can clearly identify three points, where $s \geq 0.6$, which are potential anomalies.

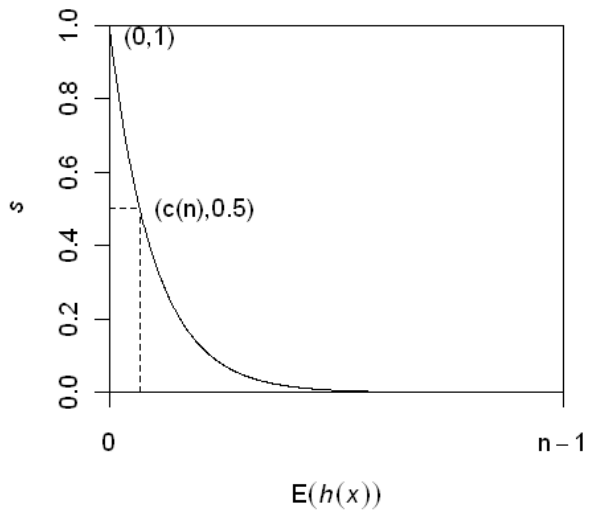


Figure 2. The relationship of expected path length $E(h(x))$ and anomaly score s. $c(n)$ is the average path length as defined in equation 1. If the expected path length $E(h(x))$ is equal to the average path length $c(n)$, then $s = 0.5$, regardless of the value of n.

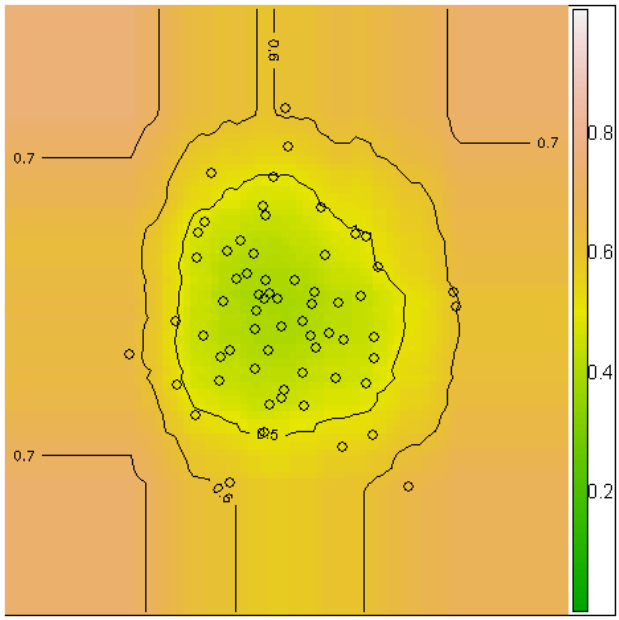


Figure 3. Anomaly score contour of iForest for a Gaussian distribution of sixty-four points. Contour lines for $s = 0.5, 0.6, 0.7$ are illustrated. Potential anomalies can be identified as points where $s \geq 0.6$.

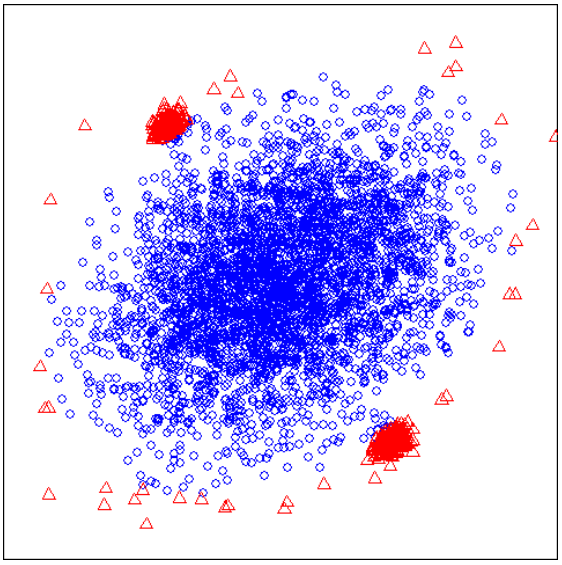
3 Characteristic of Isolation Trees

This section describes the characteristic of iTrees and their unique way of handling the effects of swamping and masking. As a tree ensemble that employs isolation trees, iForest a) identifies anomalies as points having shorter path lengths, and b) has multiple trees acting as 'experts' to target different anomalies. Since iForest does not need to isolate all of normal instances – the majority of the training sample, *iForest is able to work well with a partial model without isolating all normal points and builds models using a small sample size*.

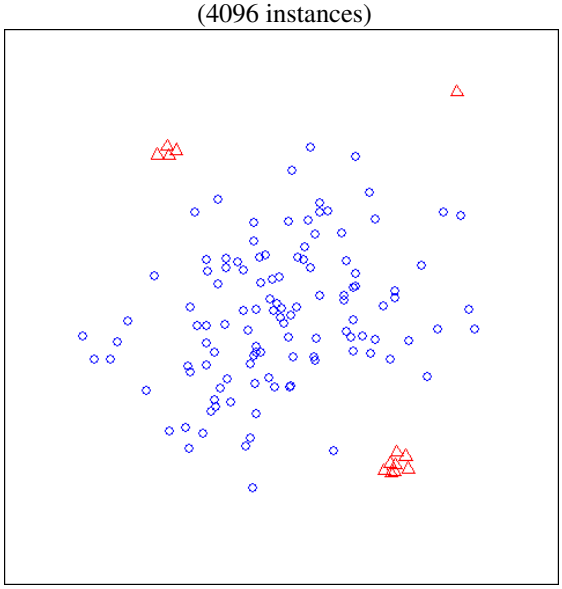
Contrary to existing methods where large sampling size is more desirable, isolation method works best when the sampling size is kept small. Large sampling size reduces iForest's ability to isolate anomalies as normal instances can interfere with the isolation process and therefore reduces its ability to clearly isolate anomalies. Thus, sub-sampling provides a favourable environment for iForest to work well. Throughout this paper, sub-sampling is conducted by random selection of instances without replacement.

Problems of swamping and masking have been studied extensively in anomaly detection [8]. Swamping refers to wrongly identifying normal instances as anomalies. When normal instances are too close to anomalies, the number of partitions required to separate anomalies increases – which makes it harder to distinguish anomalies from normal instances. Masking is the existence of too many anomalies concealing their own presence. When an anomaly cluster is large and dense, it also increases the number of partitions to isolate each anomaly. Under these circumstances, evaluations using these trees have longer path lengths making anomalies more difficult to detect. Note that both swamping and masking are a result of too many data for the purpose of anomaly detection. The unique characteristic of isolation trees allows iForest to build a partial model by sub-sampling which incidentally alleviates the effects of swamping and masking. It is because: 1) sub-sampling controls data size, which helps iForest better isolate examples of anomalies and 2) each isolation tree can be specialised, as each sub-sample includes different set of anomalies or even no anomaly.

To illustrate this, Figure 4(a) shows a data set generated by Mulcross. The data set has two anomaly clusters located close to one large cluster of normal points at the centre. There are interfering normal points surrounding the anomaly clusters, and the anomaly clusters are denser



(a) Original sample
(4096 instances)



(b) Sub-sample
(128 instances)

Figure 4. Using generated data to demonstrate the effects of swamping and masking, (a) shows the original data generated by Mulcross. (b) shows a sub-sample of the original data. Circles (○) denote normal instances and triangles (△) denote anomalies.

than normal points in this sample of 4096 instances. Figure 4(b) shows a sub-sample of 128 instances of the original data. The anomalies clusters are clearly identifiable in the sub-sample. Those normal instances surrounding the two anomaly clusters have been cleared out, and the size of anomaly clusters becomes smaller which makes them easier to identify. When using the entire sample, iForest reports an AUC of 0.67. When using a sub-sampling size of 128, iForest achieves an AUC of 0.91. The result shows iForest's superior anomaly detection ability in handling the effects swamping and masking through a sigificantly reduced sub-sample.

4 Anomaly Detection using iForest

Anomaly detection using iForest is a two-stage process. The first (training) stage builds isolation trees using sub-samples of the training set. The second (testing) stage passes the test instances through isolation trees to obtain an anomaly score for each instance.

4.1 Training Stage

In the training stage, iTrees are constructed by recursively partitioning the given training set until instances are isolated or a specific tree height is reached of which results a partial model. Note that the tree height limit l is automatically set by the sub-sampling size ψ: $l = ceiling(\log_2 \psi)$, which is approximately the average tree height [7]. The rationale of growing trees up to the average tree height is that we are only interested in data points that have shorter-than-average path lengths, as those points are more likely to be anomalies. Details of the training stage can be found in Algorithms 1 and 2.

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size
Output: a set of t *iTrees*

1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to t **do**
4: $\quad X' \leftarrow sample(X, \psi)$
5: $\quad Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: **end for**
7: **return** $Forest$

There are two input parameters to the iForest algorithm. They are the sub-sampling size ψ and the number of trees t. We provide a guide below to select a suitable value for each of the two parameters.

Sub-sampling size ψ controls the training data size. We find that when ψ increases to a desired value, iForest detects reliably and there is no need to increase ψ further because it increases processing time and memory size without any gain in detection performance. Empirically, we find that setting ψ to 2^8 or 256 generally provides enough details to perform anomaly detection across a wide range of

Algorithm 2 : $iTree(X, e, l)$

Inputs: X - input data, e - current tree height, l - height limit

Output: an iTree

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

data. Unless otherwise specified, we use $\psi = 256$ as the default value for our experiment. An analysis on the effect sub-sampling size can be found in section 5.2 which shows that the detection performance is near optimal at this default setting and insensitive to a wide range of ψ.

Number of tree t controls the ensemble size. We find that path lengths usually converge well before $t = 100$. Unless otherwise specified, we shall use $t = 100$ as the default value in our experiment.

At the end of the training process, a collection of trees is returned and is ready for the evaluation stage. The complexity of the training an iForest is $O(t\psi \log \psi)$.

4.2 Evaluating Stage

In the evaluating stage, an anomaly score s is derived from the expected path length $E(h(x))$ for each test instance. $E(h(x))$ are derived by passing instances through each iTree in an iForest. Using $PathLength$ function, a single path length $h(x)$ is derived by counting the number of edges e from the root node to a terminating node as instance x traverses through an iTree. When x is terminated at an external node, where $Size > 1$, the return value is e plus an adjustment $c(Size)$. The adjustment accounts for an unbuilt subtree beyond the tree height limit. When $h(x)$ is obtained for each tree of the ensemble, an anomaly score is produced by computing $s(x, \psi)$ in Equation 2. The complexity of the evaluation process is $O(nt \log \psi)$, where n is the testing data size. Details of the $PathLength$ function can be found in Algorithm 3. To find the top m anomalies, simply sorts the data using s in descending order. The first m instances are the top m anomalies.

Algorithm 3 : $PathLength(x, T, e)$

Inputs : x - an instance, T - an iTree, e - current path length; to be initialized to zero when first called

Output: path length of x

if T is an external node **then**
  return $e + c(T.size)$ $\{c(.)$ is defined in Equation 1$\}$
end if
$a \leftarrow T.splitAtt$
if $x_a < T.splitValue$ **then**
  return $PathLength(x, T.left, e+1)$
else $\{x_a \geq T.splitValue\}$
  return $PathLength(x, T.right, e+1)$
end if

5 Empirical Evaluation

This section presents the detailed results for four sets of experiment designed to evaluate iForest. In the first experiment we compare iForest with ORCA [3], LOF [6] and Random Forests (RF) [12]. LOF is a well known density based method, and RF is selected because this algorithm also uses tree ensembles. In the second experiment, we examine the impact of different sub-sampling sizes using the two largest data sets in our experiments. The results provide an insight as to what sub-sampling size should be used and its effects on detection performance. The third experiment extends iForest to handle high-dimensional data; we reduce attribute space before tree construction by applying a simple uni-variate test for each sub-sample. We aim to find out whether this simple mechanism is able to improve iForest's detection performance in high dimensional spaces. In many cases, anomaly data are hard to obtain, the fourth experiment examines iForest's performance when only normal instances are available for training. For all the experiments, actual CPU time and Area Under Curve (AUC) are reported. They are conducted as single threaded jobs processed at 2.3GHz in a Linux cluster (www.vpac.org).

The benchmarking method is ORCA - a k-Nearest Neighbour (k-nn) based method and one of the state-of-the-art anomaly detection methods, where the largest demand of processing time comes from the distance calculation of k nearest neighbours. Using sample randomisation together with a simple pruning rule, ORCA is claimed to be able to cut down the complexity of $O(n^2)$ to near linear time [3].

In ORCA, the parameter k determines the number of nearest neighbourhood, increasing k also increases the run time. We use ORCA's default setting of $k = 5$ in our experiment unless otherwise specified. The parameter N determines how many anomalies are reported. If N is small, ORCA increases the running cutoff rapidly and pruning off more searches, resulting in a much faster run time. However, it would be unreasonable to set N below the number

of anomalies due to AUC's requirement to report anomaly scores for every instances. Since choosing N has an effect on run time and the number of anomalies is not supposed to be known in the training stage, we will use a reasonable value $N = \frac{n}{8}$ unless otherwise specified. Using ORCA's original default setting ($k = 5$ and $N = 30$), all data sets larger than one thousand points report AUC close to 0.5, which is equivalent to randomly selecting points as anomalies. In reporting processing time, we report the total training and testing time, but omit the pre-processing time "dprep" from ORCA.

As for LOF, we use a commonly used setting of $k = 10$ in our experiment. As for RF, we use $t = 100$ and other parameters in their default values. Because RF is a supervised learner, we follow the exact instruction as in [12] to generate synthetic data as the alternative class. The alternative class is generated by uniformly sampling random points valued between the maximums and minimums of all attributes. Proximity measure is calculated after decision trees are being constructed and anomalies are instances whose proximities to all other instances in the data are generally small.

We use eleven natural data sets plus a synthetic data set for evaluation. They are selected because they contain known anomaly classes as ground truth and these data sets are used in the literature to evaluate anomaly detectors in a similar setting. They include: the two biggest data subsets (*Http* and *Smtp*) of KDD CUP 99 network intrusion data as used in [14], *Annthyroid*, *Arrhythmia*, Wisconsin Breast Cancer (*Breastw*), Forest Cover Type (*ForestCover*), *Ionosphere*, *Pima*, *Satellite*, *Shuttle* [2], *Mammography*[1] and Mulcross [10]. Since we are only interested in continuous-valued attributes in this paper, all nominal and binary attributes are removed. The synthetic data generator, Mulcross generates a multi-variate normal distribution with a selectable number of anomaly clusters. In our experiments, the basic setting for Mulcross is as following: contamination ratio = 10% (number of anomalies over the total number of points), distance factor = 2 (distance between the center of normal cluster and anomaly clusters), and number of anomaly clusters = 2. An example of Mulcross data can be found in Figure 4. Table 2 provides the properties of all data sets and information on anomaly classes sorted by the size of data in descending order.

It is assumed that anomaly labels are unavailable in the training stage. Anomaly labels are only available in the evaluation stage to compute the performance measure, AUC.

[1]The Mammography data set was made available by the courtesy of Aleksandar Lazarevic

	n	d	anomaly class
Http (KDDCUP99)	567497	3	attack (0.4%)
ForestCover	286048	10	class 4 (0.9%) vs. class 2
Mulcross	262144	4	2 clusters (10%)
Smtp (KDDCUP99)	95156	3	attack (0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%)
Mammography	11183	6	class 1 (2%)
Annthyroid	6832	6	classes 1, 2 (7%)
Satellite	6435	36	3 smallest classes (32%)
Pima	768	8	pos (35%)
Breastw	683	9	malignant (35%)
Arrhythmia	452	274	classes 03,04,05,07, 08,09,14,15 (15%)
Ionosphere	351	32	bad (36%)

Table 2. Table of Data properties, where n is the number of instances, and d is the number of dimensions, and the percentage in bracket indicates the percentage of anomalies.

5.1 Comparison with ORCA, LOF and Random Forests

The aim of this experiment is to compare iForest with ORCA, LOF and RF in terms of AUC and processing time. Table 3 reports the AUC score and actual run time for all methods. From the table, we observe that iForest compares favourably to ORCA. It shows that iForest as a model-based method outperforms ORCA, a distance based method, in terms of AUC and processing time. In particular, iForest is more accurate and faster in all the data sets larger than one thousand points.

Note that the difference in execution time is huge between iForest and ORCA, especially in large data sets; this is due to the fact that iForest is not required to compute pairwise distances; this happens despite the fact that ORCA only reports $\frac{n}{8}$ anomalies where iForest ranks all n points.

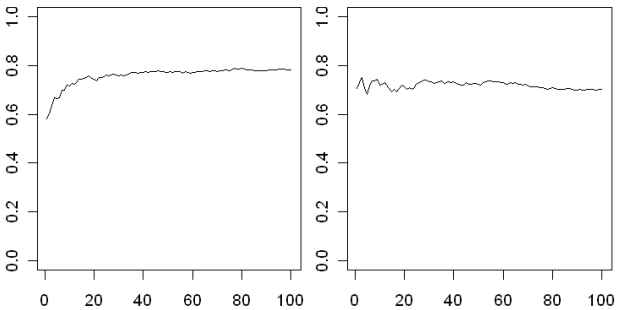
iForest compares favourable to LOF in seven out of eight data sets examined and iForest is better than RF in all the four data sets tested in terms of AUC. In terms of processing time, iForest is superior in all the data sets as compared with LOF and RF.

The performance of iForest is stable in a wide range of t. Using the two data sets of highest dimension, Figure 5 shows that AUC converges at a small t. Since increasing t also increases processing time, the early convergence of AUC suggests that iForest's execution time can be further reduces if t is tuned to a data set.

As for the *Http* and *Mulcross* data sets, due to the large

	AUC				Time (seconds)					
	iForest	ORCA	LOF	RF	iForest			ORCA	LOF	RF
					Train	Eval.	Total			
Http (KDDCUP99)	**1.00**	0.36	NA	NA	0.25	15.33	**15.58**	9487.47	NA	NA
ForestCover	**0.88**	0.83	NA	NA	0.76	15.57	**16.33**	6995.17	NA	NA
Mulcross	**0.97**	0.33	NA	NA	0.26	12.26	**12.52**	2512.20	NA	NA
Smtp (KDDCUP99)	**0.88**	0.80	NA	NA	0.14	2.58	**2.72**	267.45	NA	NA
Shuttle	**1.00**	0.60	0.55	NA	0.30	2.83	**3.13**	156.66	7489.74	NA
Mammography	**0.86**	0.77	0.67	NA	0.16	0.50	**0.66**	4.49	14647.00	NA
Annthyroid	**0.82**	0.68	0.72	NA	0.15	0.36	**0.51**	2.32	72.02	NA
Satellite	**0.71**	0.65	0.52	NA	0.46	1.17	**1.63**	8.51	217.39	NA
Pima	0.67	**0.71**	0.49	0.65	0.17	0.11	0.28	**0.06**	1.14	4.98
Breastw	**0.99**	0.98	0.37	0.97	0.17	0.11	0.28	**0.04**	1.77	3.10
Arrhythmia	**0.80**	0.78	0.73	0.60	2.12	0.86	2.98	**0.49**	6.35	2.32
Ionosphere	0.85	**0.92**	0.89	0.85	0.33	0.15	0.48	**0.04**	0.64	0.83

Table 3. iForest performs favourably to ORCA, especially for large data sets where $n > 1000$. AUC reported with boldfaced are the best performance. iForest is significantly faster than ORCA for large data sets where $n > 1000$. We do not have the full results for LOF and RF because: (1) LOF has a high computation complexity and is unable to complete some very high volume data sets in reasonable time; (2) RF has a huge memory requirement, which requires system memory of $(2n)^2$ to produce proximity matrix in unsupervised learning settings.



(a) *Arrhythmia* (b) *Satellite*

Figure 5. Detection performance AUC (y-axis) is converged at a small t (x-axis).

anomaly-cluster size and the fact that anomaly clusters have an equal or higher density as compared to normal instances (i.e., masking effect), ORCA reports a poorer-than-average result on these data sets. We also experiment ORCA on these data sets using a higher value of k (where $k = 150$), however the detection performance is similar. This highlights one problematic assumption in ORCA and other similar k-nn based methods: they can only detect low-density anomaly clusters of size smaller than k. Increasing k may solve the problem, but it is not practical in high volume setting due to the increase in processing time.

5.2 Efficiency Analysis

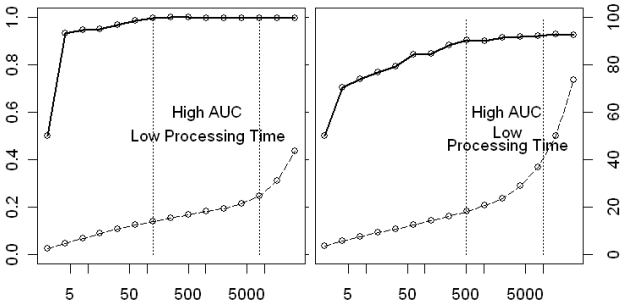
This experiment investigates iForest's efficiency in relation to the sub-sampling size ψ. Using the two largest data sets, Http and ForestCover, we examine the effect of sub-sample size on detection accuracy and processing time. In this experiment we adjust the sub-sampling size $\psi = 2, 4, 8, 16, ..., 32768$.

Our findings are shown in Figure 6. We observe that AUC converges very quickly at small ψ. AUC is near optimal when $\psi = 128$ for Http and $\psi = 512$ for ForestCover, and they are only a fraction of the original data (0.00045 for Http and 0.0018 for ForestCover). After this ψ setting, the variation of AUC is minimal: ± 0.0014 and ± 0.023 respectively. Also note that the processing time increases very modestly when ψ increases from 4 up to 8192. iForest maintains its near optimal detection performance within this range. In a nutshell, a small ψ provides high AUC and low processing time, and a further increase of ψ is not necessary.

5.3 High Dimensional Data

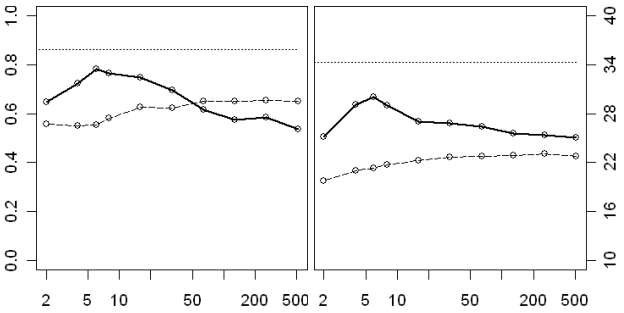
One of the important challenges in anomaly detection is high dimensional data. For distance-based methods, every point is equally sparse in high dimensional space — rendering distance a useless measure. For iForest, it also suffers from the same 'curse of dimensionality'.

In this experiment, we study a special case in which high



(a) *Http* (b) *ForestCover*

Figure 6. A small sub-sampling size provides both high AUC (left y-axis, solid lines) and low processing time (right y-axis, dashed lines, in seconds). Sub-sampling size (x-axis, log scale) ranges $\psi = 2, 4, 8, 16, ..., 32768$.



(a) *Mammography* (b) *Annthyroid*

Figure 7. iForest achieves good results on high dimensional data using Kurtosis to select attributes. 506 irrelevant attributes are added. AUC (left y-axis, solid lines) improves when the subspace size (x-axis), comes close to the number of original attributes and processing time (right y-axis, dashed lines, in seconds) increases slightly as subspace size increases. iForest trained using the original data has slightly better AUC (shown as the top dotted lines).

dimension data sets has a large number of irrelevant attributes or background noises, and show that iForest has a significant advantage in processing time. We simulate high dimensional data using Mammography and Annthyroid data sets. For each data set, 506 random attributes, each uniformly distributed, valued between 0 and 1 are added to simulate background noise. Thus, there is a total of 512 attributes in each data sets. We use a simple statistical test, Kurtosis, to select an attribute subspace from the sub-sample before constructing each iTree. Kurtosis measures the 'peakness' of univariate distribution. Kurtosis is sensitive to the presence of anomalies and hence it is a good attribute selector for anomaly detection. After Kurtosis has provided a ranking for each attribute, a subspace of attributes is selected according to this ranking to construct each tree. The result is promising and we show that the detection performance improves when the subspace size comes close to the original number of attributes. There are other attribute selectors that we can choose from, e.g., Grubb's test. However, in this section, we are only concern with showcasing iForest's ability to work with an attribute selector to reduce dimensionality of anomaly detection tasks.

Figure 7 shows that a) processing time remains less than 30 seconds for the whole range of subspace sizes and b) AUC peaks when subspace size is the same as the number of original attributes and this result comes very close to the result of ORCA using the original attributes only. When ORCA is used on both high dimensional data sets, it reports AUC close to 0.5 with processing time over one hundred seconds. It shows that both data sets are challenging, however, iForest is able to improve the detection performance by a simple addition of Kurtosis test. It may well be possible for other methods to apply similar attribute reduction technique to improve detection accuracy on high-dimensional data, but the advantage of iForest is its low processing time even in high dimensional data.

5.4 Training using normal instances only

"Does iForest work when training set contains normal instances only? " To answer this question, we conduct a simple experiment using the two largest data sets in our experiment. We first randomly divide each data set into two parts, one for training and one for evaluation, so that the AUC is derived on unseen data. We repeat this process ten times and report the average AUC.

When training with anomalies and normal points, Http reports $AUC = 0.9997$; however, when training without anomalies, AUC reduces to 0.9919. For ForestCover, AUC reduces from 0.8817 to 0.8802. Whilst there is a small reduction in AUC, we find that using a larger sub-sampling size can help to restore the detection performance. When we increase the sub-sampling size from $\psi = 256$ to $\psi = 8,192$ for Http and $\psi = 512$ for ForestCover and train without anomalies, AUC catches up to 0.9997 for Http and 0.884 for ForestCover.

6 Discussion

The implication of using a small sub-sample size is that one can easily host an online anomaly detection system with

minimal memory footprint. Using $\psi = 256$, the maximum number of nodes is 511. Let the maximum size of a node be b bytes, t be the number of trees. Thus, a working model to detect anomaly is estimated to be less than $511tb$ bytes, which is trivial in modern computing equipments.

iForest has time complexities of $O(t\psi \log \psi)$ in the training stage and $O(nt \log \psi)$ in the evaluating stage. For Http data set, when $\psi = 256$, $t = 100$ and evaluating 283,748 instances, the total processing time is 7.6 seconds only. We increase the sub-sampling size 64 times to $\psi = 16384$ and the processing time increases by only 1.6 times to 11.9 seconds. It shows that iForest has a low constant in its computational complexity.

iForest's fast execution with low memory requirement is a direct result of building partial models and requiring only a significantly small sample size as compared to the given training set. This capability is unparallel in the domain of anomaly detection.

7 Conclusions

This paper proposes a fundamentally different model-based method that focuses on anomaly isolation rather than normal instance profiling. The concept of isolation has not been explored in the current literature and the use of isolation is shown to be highly effective in detecting anomalies with extremely high efficiency. Taking advantage of anomalies' nature of 'few and different', iTree isolates anomalies closer to the root of the tree as compared to normal points. This unique characteristic allows iForest to build partial models (as opposed to full models in profiling) and employ only a tiny proportion of training data to build effective models. As a result, iForest has a linear time complexity with a low constant and a low memory requirement which is ideal for high volume data sets.

Our empirical evaluation shows that iForest performs significantly better than a near-linear time complexity distance-based method, ORCA, LOF and RF in terms of AUC and execution time, especially in large data sets. In addition, iForest converges quickly with a small ensemble size, which enables it to detect anomalies with high efficiency.

For high dimensional problems that contain a large number of irrelevant attributes, iForest can achieve high detection performance quickly with an additional attribute selector; whereas a distance-based method either has poor detection performance or requires significantly more time. We also demonstrate that iForest works well even when no anomalies are present in the training set. Essentially, Isolation Forest is an accurate and efficient anomaly detector especially for large databases. Its capacity in handling high volume databases is highly desirable for real life applications.

Acknowledgements The authors thank Victorian Partnership for Advanced Computing (www.vpac.org) for providing the high-performing computing facility.

Z.-H. Zhou was supported by NSFC (60635030, 60721002) and JiangsuSF (BK2008018).

References

[1] N. Abe, B. Zadrozny, and J. Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 504–509. ACM Press, 2006.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38. ACM Press, 2003.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. *ACM SIGMOD Record*, 29(2):93–104, 2000.

[5] Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recogn. Lett.*, 24(9-10):1641–1650, 2003.

[6] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 392–403, San Francisco, CA, USA, 1998. Morgan Kaufmann.

[7] D. E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, April 1998.

[8] R. B. Murphy. *On Tests for Outlying Observations*. PhD thesis, Princeton University, 1951.

[9] B. R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. Wiley, 1999.

[10] D. M. Rocke and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.

[11] P. J. Rousseeuw and K. V. Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

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

[13] M. Wu and C. Jermaine. Outlier detection by sampling with accuracy guarantees. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 767–772, New York, NY, USA, 2006. ACM.

[14] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. Online unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 320–324. ACM Press, 2000.