A Novel Anomaly Detection Algorithm Based on Trident Tree

Chunkai Zhang[1(✉)], Ao Yin[1], Yepeng Deng[1], Panbo Tian[1], Xuan Wang[1], and Lifeng Dong[2]

[1] Department of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China
ckzhang812@gmail.com, yinaoyn@126.com, erpim@qq.com, panbo_tian@qq.com, wangxuan@cs.hitsz.edu.cn

[2] Department of Physics, Hamline University, 1536 Hewitt Avenue, St. Paul, MN 55104, USA

Abstract. In this paper, we propose a novel anomaly detection algorithm, named T-Forest, which is implemented by multiple trident trees (T-trees). Each T-tree is constructed recursively by isolating the data outside of 3 *sigma* into the left and right subtree and isolating the others into the middle subtree, and each node in a T-tree records the size of datasets that falls on this node, so that each T-tree can be used as a local density estimator for data points. The density value for each instance is the average of all trees evaluation instance densities, and it can be used as the anomaly score of the instance. Since each T-tree is constructed according to 3 *sigma* principle, each tree in TB-Forest can obtain good anomaly detection results without a large tree height. Compared with some state-of-the-art methods, our algorithm performs well in AUC value, and needs linear time complexity and space complexity. The experimental results show that our approach can not only effectively detect anomaly points, but also tend to converge within a certain parameters range.

Keywords: Anomaly detection · Isolation · Forest · 3 *sigma* Gaussian

1 Introduction

Anomaly points are the data points that deviate from most data and do not obey the distribution of most data points [1–3]. Anomaly detection has been a widely researched problem in several application domains such as system health management, intrusion detection, healthy-care, bio-informatics, fraud detection, and mechanical fault detection. For these applications, anomaly detection as an unsupervised learning task is very important. The significance of anomaly detection is due to the fact that anomaly data can be translated into important operational information in various application domains. For example, with the

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 295–306, 2018.
https://doi.org/10.1007/978-3-319-94295-7_20

development of cloud computing [4–7], it is very important to detect abnormal traffic in the cloud in a timely manner. And with the development of the cloud storage [8,9], timely detection of abnormal disk reads and writes in cloud storage can greatly reduce the potential risk of cloud storage.

Many unsupervised anomaly detection approaches, including classification-based [10,11], clustering-based [12,13], density-based [14–17], and angle-baed [18], calculate the distance between data points to determine their similarity, and then determine whether the data points are abnormal. There are many ways to calculate the distance, such as Euclidean distance, DTW, and so on. By analyzing the characteristics of these distance calculation formulas, we can get that the result can be easily affected by the data values and the number of data attributes. And many of the above algorithms are constrained to low dimensional data and small data size due to the high computational complexity of its original algorithm. In order to solving the above problems, Liu *et.al* proposed a different approach that detects anomalies by isolating instances, without relying on any distance or density measure [19,20]. In this approach, since the attribute and the split value of each node in the isolation tree are randomly selected, the built isolation tree can also be called a completely random tree. Consequently, there is often a certain degree of randomness in the anomaly detection results by using such a model.

In response to these challenges, we propose a novel anomaly detection algorithm on the basis of isolation-forest [19]. The key insight of our proposed algorithm is a fast and accurate local density estimator implemented by multiple trident trees(T-trees), called T-Forest. The proposed approach can isolate anomaly data faster and more accurately. To achieve this, we extend the binary tree structure of the original isolation tree to the structure of the trident tree. And instead of selecting the split value randomly, we take advantage of *sigma* principle to select two split values at a time to split the inconsistent attribute data as soon as possible. In this paper, we will show that trident trees can effectively isolate anomaly points.

The main contributions of this paper are summarized as follows:

- We propose a novel anomaly detection algorithm, called TB-Forest. Data points which have short average path on the T-tree, may be seen as anomaly points.
- We propose a local density assessment method, which is used to estimate the anomaly degree of data points. We perform many experiments on benchmark dataSets. The experiment results show that our proposed approaches outperforms the competing methods on most of the benchmark dataSets in AUC score.

The remainder of this paper is organized as follows. In Sect. 1, we review the related work. In Sect. 2, we present the proposed anomaly detection algorithm. In Sect. 3, we perform some empirical experiments to demonstrate the effectiveness of our algorithm. Lastly, the conclusion will be shown in Sect. 4.

2 Anomaly Detection by Our Method

In this section, we will show the detailed steps of our proposed detection algorithm. We first show some definitions used in this paper. Then we present the implementation details. Table 1 summarizes the symbols used in this paper.

2.1 Definitions

In this section, we will present the definition of T-tree and introduce the attributes of each node in a T-tree. And we definite the formula for calculating the anomaly score.

Table 1. Symbols and descriptions

Symbols	Description
N	Number of instances in a dataset
x	An instance
X	A dataset of N instances
Q	A set of attributes
q	An attribute
u_q	The mean of the attribute q
σ_q	The standard deviation of the attribute q
T	A tree
T_r	A right tree of a node τ
T_l	A left tree of a node τ
T_m	A middle tree of a node τ
p_l	The split value between left tree and middle tree, $u_q - 3\sigma_q$
p_r	The split value between middle tree and left tree, $u_q + 3\sigma_q$
t	Number of trees
ψ	Sample ratio
$den(x)$	The number of contained instances of the external node that x belongs to
$hlim$	Height limit
$slim$	Size limit in training a tree

Definition 1. T-tree: *Let T denote a T-tree and τ be a node in this T-tree. It is either an external-node with no child, or an internal-node with a set and exactly three child trees(T_l,T_m,T_r). Each node in a T-tree contains the following elements:(1) variables p_l and p_r, which are used to divide data points into T_l, T_m, and T_r subtree; (2) variable size, which is used to record the instances number located in this node; (3) three node pointers, which are used to the left subtree, the right subtree, and the middle subtree.*

Given a sample of data X=$\{x_1, x_2, ..., x_n\}$ of n instances, in which each x_i has d attributes, to build a trident tree(T-tree). Figure 1 is used as an example

to illustrate the structure of a T-tree. We recursively divide X by randomly selecting an attribute q and calculating the left split value p_l and the right split value p_r, until either: (1) the tree height reaches limit, $hlim$, (2) the number of instances at a node τ is less than the size limit, $slim$ or (3) all points of the selected attribute have the same value in X. An T-tree has the property as BST, that is, the value of an attribute in left child tree is less than the middle side and the middle size is less than the right side. Assuming all instances are distinct, the number of instances in left child tree is equal to the number in the right side. Since each leaf node in the MB-Tree contains no fewer than one instance and each internal node contains exactly three children, the total number of nodes in a T-tree is less than $\frac{3N}{2} - 1$. Hence, an MB-Tree is only linear storage overhead.

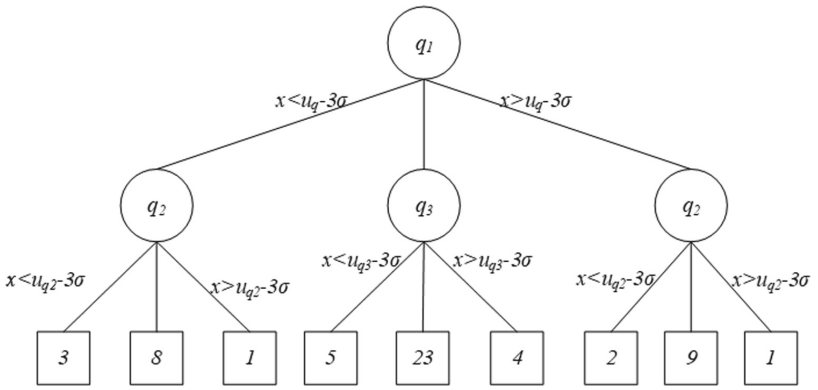

Fig. 1. This figure is used as an example to illustrate the structure of a T-tree and show the process of the division. Round nodes represent internal nodes, and square nodes represent external nodes.

In this paper, calculating a score for each data point that reflects the anomaly degree of each data point. Since all instances in the dataSets fall on different external nodes after being divided several times according to different attribute values, each node in a T-tree forms a local subspace and instances located on each node have similar data characteristics. Instances on each node become K-nearest-neighbors to each other. Hence, we calculate the local density of the instance by counting the number of instances in each node where the instance falls, to determine the anomaly degree of the instance.

Anomaly Score: The anomaly score of each point x is measured by the average local density, $den(x)$. $den(x)$ denotes the number of instances contained in the terminational node of x. Since we have adopted the idea of ensemble learning and the number of instances used to train each T-tree may be different, we need to normalise the value $den(x)$ in order to calculate the final anomaly score by using all results in the TB-Forest. Therefore, the anomaly score of an instance can be defined as follows:

$$score(x) = \frac{1}{t}\sum_{i=1}^{t}\frac{den_i(x)}{N_i} \tag{1}$$

where $den_i(x)$ is the local density in a T-tree and N_i is the instances number in T-tree$_i$. Because of the characteristics "less and different" of anomaly points, the value of $score(x)$ will be less than the normal points. Therefore, the smaller the value of $score(x)$, the more abnormal it is. In other words, the closer of the value of $score(x)$ is to 0, the more likely the corresponding instance is an anomaly point.

2.2 Training for T-Forest

In this section, we will introduce the concrete steps of building a T-Forest, which is composed of many T-trees. Each T-tree is built on the dataset sampled by using the variable bagging technique.

Building a T-tree: A T-tree can be constructed by recursively dividing the training datasets until the tree reaches height limit or the instances number is less than the size limit. During the construction of each node of a T-tree, an attribute will be randomly selected from the attributes of the datasets as the splitting attribute. And these two splitting values, p_l and p_r, will be obtained by calculating the mean value and the standard derivation of this attribute, and then are used to divide the training datasets. The concrete steps to build a T-tree are described in Algorithm 1. The time complexity of this algorithm is only $O(log_3(N))$, and the space complexity is liner $O(N)$.

Building a T-Forest: There are three parameters that need to be set manually to the TB-Forest. They are the height limit of a T-tree $hlim$, the size limit of each node $slim$, and the tree number t. There is also a variable, sample ratio ψ, which is generated uniformly from the range between 0.2 and 0.8. This parameter may be different in each T-tree, so it can increase the diversity of samples in each T-tree. The changes of these parameters will affect the evaluation effect. For example, many trees or a high T-tree can increase the accuracy of the result. However, after increasing to a certain value, the result will tend to converge. If these two parameters are set too large, it not only did not improve the detection performance, but increase the model's runtime and memory consumption. The size limit is always set to 15, because if the number of samples is less than 15, the criteria 3 $sigma$ will no longer apply. The concrete steps of TB-Forest algorithm are described as Algorithm 2. The time complexity of building a TB-Forest is $O(t * \psi * log_3(N))$. The space complexity of building a TB-Forest $O(t * \psi * N)$.

2.3 Evaluation Stage

In this evaluation stage, an anomaly score of instance x can be estimated by the average local density $E(den(x))$. $den(x)$ can be calculated by Algorithm 3. After getting all $den(x)$ of instance x by TB-Forest, we can use the formula

Algorithm 1: Building a T-tree($Data$, cur, $hlim$, $slim$)

Input: A dataSet $Data$, a current height cur, a height limit $hlim$, a size limit $slim$.

Output: A T-tree

if $|Data| \leq slim$ *and* $cur \geq hlim$ **then**
    return node(size=|Data|,exteral=TRUE);
end
Randomly select an attribute $q \in Q$.
Calculate the mean value, u_q, of the attribute q.
Calculate the standard derivation, σ, of the attribute q.
left-split $\leftarrow u_q - 3\sigma$
right-split $\leftarrow u_q + 3\sigma$
$Data_l \leftarrow$ data-filter($Data$, $x \leq$ left-split)
$Data_m \leftarrow$ data-filter($Data$, $x >$left-split and $x <$right-split)
$Data_r \leftarrow$ data-filter($Data$, $x \geq$ right-split)
Left-Tree $\leftarrow$ Building-Tree($Data_l$,$cur + 1$, $hlim$, $slim$)
Middle-Tree $\leftarrow$ Building-Tree($Data_m$,$cur + 1$, $hlim$, $slim$)
Right-Tree $\leftarrow$ Building-Tree($Data_r$,$cur + 1$, $hlim$, $slim$)
return node(Left-Tree,Middle-Tree,Right-Tree,size=|Data|,p_l =left-split,p_r =right-split,exteral=FALSE);

1 to calculate the average local density $E(den(x))$. The closer of the value of $E(den(x))$ is to 0, the more likely the instance x is an anomaly point. For the test dataSet, we can sort the anomaly scores of these instances to get the top K anomaly points. The time complexity of getting the average local density $E(den(x))$ of this test dataSet is $O(M * t * log_3(N * \varphi))$, where N is the instances number of training dataSets, and M is the instances number of this test dataSet.

3 Experimental Evaluation

In this section, we will present the performance of our proposed algorithm from many experiments using the public dataset from UCI. For comparability, we implemented all experiments on our workstation with 2.5 GHz, 6 bits operation system, 4 cores CPU and 16 GB RAM, and the algorithms codes are built in Python 2.7.

3.1 Experimental Metrics and Experimental Setup

Metrics: In our experiment, we use Area Under Curve(AUC) as the evaluation metric with other classic anomaly detection algorithms. AUC denotes the area under of Receiver Operating Characteristic(ROC) curve and illustrates the diagnostic ability of a binary classifier system. AUC is created by plotting the true positive rate against the false positive rate at various threshold settings[1].

[1] https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

Algorithm 2: Building a TB-Forest

Input: A dataSet *Data*,number of trees t, height limit *hlim*, size limit *slim*
Output: A set of TB-Trees, TB-Forest

```
Forest ← set();
sample-size ← uniform(min())
for i=1 to t do
    ψ ← uniform(0.2,0.8)
    Data~ ← sample(Data, ψ).
    tree ← Building-Tree(Data~,0,hlim,slim)
    Forest ← [Forest, tree]
end
return Forest;
```

Algorithm 3: Local density of an instance($den(x)$)

Input: An instance x, a height limit *hlim*, A T-tree *root*
Output: The anomaly score of this instance

```
if root is external or cur >= hlim then
    return root→size / N;
end
if x_q ≤ root.q_l then
    return Local-density(x,cur + 1,hlim, root → left);
else if x_q ≥ root.q_l then
    return Local-density(x,cur + 1,hlim, root → right);
else
    return Local-density(x,cur + 1,hlim, root → middle);
end
```

The detection algorithm with larger AUC has the better detection accuracy, otherwise, the detection algorithm is less effective.

Experimental Setup: There are two types of experiments. First, we compare the differences in the AUC value between our proposed algorithm and other classic algorithms. Second, we compare the effect of parameters on our proposed algorithm and iForest. All above experiments are performed on the selected twelve datasets from public UCI datasets [21], which are summarized in Table 2. Most of these datasets include two labels, and we use the most class as the normal class and the less class as the anomaly class, for example, Http which is from KDD CUP 99 network intrusion data [22] includes two classes 0, 1. For other datasets that include multiple classes, we need process these into two labels. Arrhythmia has 15 classes, and we choose 3, 4, 5, 7,8,9, 14, 15 as the anomaly class and other classes as the normal class. We choose *NON-MUSK-252*, *NON-MUSK-j146*, and *NON-MUSK-j147* as the normal class, and choose *MUSK-213*, *MUSK-211* as the anomaly class in musk dataset. In HAPT dataset, we choose

the class 5 as the normal class and the class 11 as the anomaly class to form the *hapt511* dataset.

In order to present the efficiency of our proposed algorithm, we choose four representative anomaly detection algorithms, which include LOF [23], iForest [19], HS-Forest [24], RS-Forest [25]. LOF is the same as our proposed algorithm to determine the abnormality of data points by calculating the local density of data points, but LOF calculate the local density of data points by calculating the similarity between data points and it needs $O(N^2)$ time complexity. HS-Forest, RS-Forest are the same as our proposed algorithm to calculating the local density of data points by counting the instances number of a terminal node in model tree, but each model tree in these two algorithms is constructed without training datasets. And these two algorithms can be used to stream data, but we only use their function on static datasets. Because our algorithm is an improvement on iForest, we choose it as a comparision algorithms. As for LOF, we set $k = 10$ in our experiment. As for HS-Forest, RS-Forest, and iForest, we set height limit $hlim = 6$ and the trees number $t = 25$. For our proposed algorithm, we set the height limit $hlim = 4$ and the trees number $t = 25$.

Table 2. Benchmark data sets, where n denotes the size of datasets, and d denotes the number of attributes of datasets.

Datasets	n	d	Anomaly ratio
Http	567497	3	0.4%
Satellite	6435	37	31.6%
ann_throid	7200	6	7.4%
Cardiotocography	1831	20	9.6%
Musk	5682	165	1.7%
Epileptic	11500	178	20%
hapt511	1513	561	5.9%
Breast	569	10	35.7%
Arrhythmia	452	273	14.6%
Shuttle	14500	10	5.9%
Pima	768	9	35%
Ionosphere	351	34	35.8%

3.2 Performance on AUC

This experiment is to compare our proposed algorithm with other algorithms in term of AUC. Table 3 presents the results of all compared algorithms on benchmark datsets. From this table, we can find that our algorithm outperforms other algorithms on most of this benchmark datasets. The AUC performance of our algorithms are approximative to RS-Forest, HS-Forest, and iForest, but it is much better than LOF algorithm on all these datasets. From this table,

we can find that our proposed algorithm outperforms iForest on nine of twelve datasets, which can illustrate that our improvement is successful. And we can observe that our algorithms performs well on these datasets, which contain a small percentage of anomaly class from Table 2.

As we all known, the Http dataset is a network traffic data set provided by KDD99. There is a small amount of traffic data in this dataset as abnormal traffic, and we treat these small amount of abnormal data as data with *class 0*. Our detection algorithm can efficiently detect such abnormal network traffic, and our algorithm only requires a logarithmic level of runtime. So our algorithm can be used to detect whether there is abnormal traffic in the public cloud of private cloud.

Table 3. Performance comparison of different methods on different benchmark data sets. AUC score is measured, and the bold font indicates that the algorithm performs significantly better.

Data sets	**TB-Forest***	iForest	LOF	RS-Forest	HS-Forest
Http	0.9925	**1.00**	NA	0.999	0.996
Satellite	0.68	**0.71**	0.52	0.7	0.59
ann_thyroid	**0.85**	0.81	0.72	0.68	0.8
Cardiotocography	**0.93**	0.92	0.539	0.88	0.74
Musk	**0.77**	0.64	0.531	0.64	0.66
Epileptic	**0.984**	0.98	0.57	0.88	0.94
hapt511	**0.999**	0.998	0.595	0.997	0.982
Breast	0.88	0.84	0.6293	0.518	**0.94**
Arrhythmia	**0.836**	0.80	0.69	0.695	0.686
Shuttle	0.992	**1.00**	0.55	0.998	0.999
Pima	**0.71**	0.67	0.513	0.49	**0.71**
Ionosphere	**0.94**	0.85	0.546	0.89	0.78

Parameters Analysis: In this experiment, we show the effect of two parameter (*hlim* and t) values on the detection results of all compared algorithms on benchmark datasets. Due to the space limitations, we only show the experiment results on *Http* and *shuttle* datasets.

Figure 2 shows that the value of AUC changes as the tree height changing in the *Http* dataset, when the number of trees is set to 25. This figure shows that our proposed algorithm performs better when the height limit is in the range 2 to 6. Since *Http* dataset has only three attributes, it does not require a very high tree to get good performance. Figure 3 shows that the value of AUC changes as the tree height changing in the *shuttle* dataset, when the number of trees is set to 25. In this figure, with the increasement of height limit, all comparable algorithms perform better and better and tend to converge. From these two

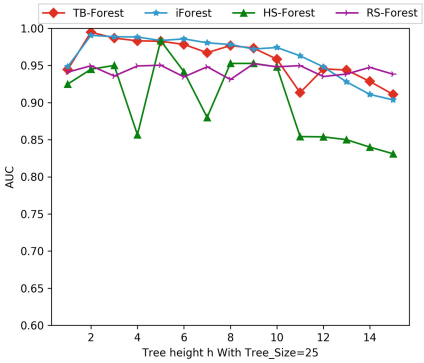

Fig. 2. AUC changes with *hlim*, when fixed $t = 25$, on http data.

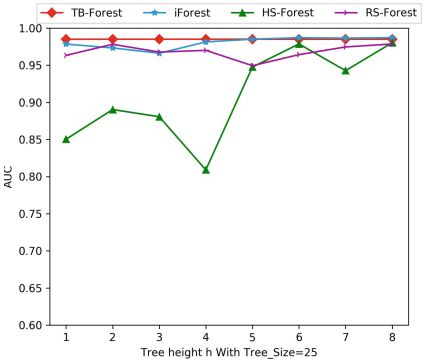

Fig. 3. AUC changes with *hlim*, when fixed $t = 25$, on shuttle data.

examples, we can find that the tree height limit of each T-tree in T-Forest can be in the range of 4 to 6, in our proposed algorithm.

Therefore, we run two experiments to detect the effect of changes in the number of trees on the AUC value with height limit $hlim = 6$. Figure 4 shows that the value of AUC changes as the number of trees changing in the *http* dataset, when the height limit of trees is set to 6. Figure 5 shows that the value of AUC changes as the number of trees changing in the *shuttle* dataset, when the height limit of trees is set to 6. From these two figures, we can observe that changes in the number of trees have little effect on the effectiveness of our algorithm. To accommodate most data sets, the parameters t can be in the range of 10 to 25.

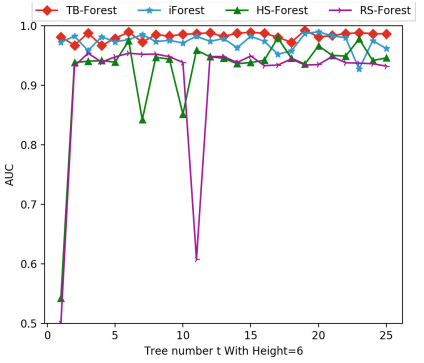

Fig. 4. AUC changes with t, when fixed $hlim = 6$, on http data.

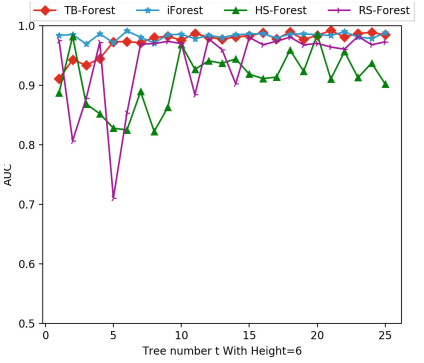

Fig. 5. AUC changes with t, when fixed $hlim = 6$, on shuttle data.

4 Conclusions

In this paper, we propose a novel anomaly detection, T-Forest, based on isolation principle. Our algorithm constructs many TB-Trees using sampled datasets by variable sample technique, and each T-tree is a trigeminal tree which is built by recursively segmenting dataset to map dataset to different subtrees by 3 *sigma* principle. Then, we have performed some experiments to illustrate the detection effect of our algorithm. The experiment results show that our algorithm can detect anomaly data points effectively and efficiently. In the future, we will focus on how to improve the detection accuracy of our algorithm on the datasets, in which normal and anomaly points are mixed distributions.

Acknowledgment. This study was supported by the Shenzhen Research Council (Grant No. JSGG20170822160842949, JCYJ20170307151518535).

References

1. Yu, X., Tang, L.A., Han, J.: Filtering and refinement: a two-stage approach for efficient and effective anomaly detection. In: IEEE International Conference on Data Mining, pp. 617–626 (2009)
2. Xie, M., Han, S., Tian, B., Parvin, S.: Anomaly detection in wireless sensor networks: a survey. J. Netw. Comput. Appl. **34**(4), 1302–1325 (2011)
3. Liu, S., Chen, L., Ni, L.M.: Anomaly detection from incomplete data. ACM Trans. Knowl. Discov. Data (TKDD) **9**(2), 11 (2014)
4. Baucke, S., Ali, R.B., Kempf, J., Mishra, R., Ferioli, F., Carossino, A.: Cloud atlas: a software defined networking abstraction for cloud to WAN virtual networking. In: IEEE Sixth International Conference on Cloud Computing, pp. 895–902 (2014)
5. Sayeed, Z., Liao, Q., Grinshpun, E., Faucher, D., Sharma, S.: Cloud analytics for short-term LTE metric prediction-cloud framework and performance. In: IEEE CLOUD (2015)
6. Kauffman, R.J., Ma, D., Shang, R., Huang, J., Yang, Y.: On the financification of cloud computing: an agenda for pricing and service delivery mechanism design research. Int. J. Cloud Comput. Featur. Article (2015)
7. An, B., Zhang, X., Tsugawa, M., Zhang, Y., Cao, C., Huang, G., Fortes, J.: Towards a model-defined cloud-of-clouds. In: Collaboration and Internet Computing, pp. 1–10 (2016)
8. Bellini, P., Cenni, D., Nesi, P.: A knowledge base driven solution for smart cloud management. In: IEEE International Conference on Cloud Computing, pp. 1069–1072 (2015)
9. Chen, W.-S.E., Huang, M.-J., Huang, C.-F.: Intelligent software-defined storage with deep traffic modeling for cloud storage service. Int. J. Serv. Comput. (IJSC), 1–14 (2016)
10. Abe, N., Zadrozny, B., Langford, J.: Outlier detection by active learning. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 504–509. ACM (2006)
11. Shi, T., Horvath, S.: Unsupervised learning with random forest predictors. J. Comput. Graph. Stat. **15**(1), 118–138 (2006)

12. He, Z., Xiaofei, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24**(9–10), 1641–1650 (2003)
13. Niu, K., Huang, C., Zhang, S., Chen, J.: ODDC: outlier detection using distance distribution clustering. In: Washio, T., et al. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4819, pp. 332–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77018-3_34
14. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J.: LOF: identifying density-based local outliers. In: ACM Sigmod Record, vol. 29, pp. 93–104. ACM (2000)
15. Fan, H., Zaïane, O.R., Foss, A., Wu, J.: A nonparametric outlier detection for effectively discovering top-N outliers from engineering data. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 557–566. Springer, Heidelberg (2006). https://doi.org/10.1007/11731139_66
16. Salehi, M., Leckie, C., Bezdek, J.C., Vaithianathan, T., Zhang, X.: Fast memory efficient local outlier detection in data streams. IEEE Trans. Knowl. Data Eng. **28**(12), 3246–3260 (2016)
17. Yan, Y., Cao, L., Kulhman, C., Rundensteiner, E.: Distributed local outlier detection in big data. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1225–1234. ACM (2017)
18. Kriegel, H.-P., Zimek, A., et al.: Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 444–452. ACM (2008)
19. Liu, F.T., Ting, K.M., Zhou, Z.-H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, ICDM 2008, pp. 413–422. IEEE (2008)
20. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. ACM Trans. Knowl. Discov. Data **6**, 1 (2012)
21. Dheeru, D., Taniskidou, E.K.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
22. Yamanishi, K.: On-line unsupervised outlier detection using finite mixture with discounting learning algorithms. Data Min. Knowl. Discov. **8**(3), 275–300 (2004)
23. Breunig, M.M., Kriegel, H.P., Ng, R.T.: LOF: identifying density-based local outliers. In: ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
24. Tan, S.C., Ting, K.M., Liu, T.F.: Fast anomaly detection for streaming data. In: Proceedings of the International Joint Conference on Artificial Intelligence, Barcelona, IJCAI 2011, Catalonia, Spain, pp. 1511–1516, July 2011
25. Wu, K., Zhang, K., Fan, W., Edwards, A., Philip, S.Y.: RS-forest: a rapid density estimator for streaming anomaly detection, pp. 600–609 (2014)