Supplementary Materials for the paper entitled: Reconstruction-based Anomaly Detection with Completely Random Forest

Yi-Xuan Xu[*] Ming Pang[*†] Ji Feng[*‡§] Kai Ming Ting[*] Yuan Jiang[*] Zhi-Hua Zhou[*]

1 Discussion

The discussion part is organized as follows: First, we discuss the similarities and differences between RecForest and two related methods: neural network autoencoder [1] and isolation forest (iForest) [3]. Second, we compare their runtime complexities.

1.1 Relationships with Other Methods. Both the RecForest and neural network autoencoder are reconstruction-based anomaly detectors. However, they employ fundamentally different mechanisms of anomaly detection with sample reconstruction. The training stage of RecForest does not require iterative optimization on model parameters like the neural network autoencoder, and the latter has many hyper-parameters. In contrast, RecForest has two hyper-parameters only: maximum tree depth and number of trees in the forest.

Both RecForest and iForest are based on a completely random forest. However, they are with totally different mechanisms of anomaly detection. iForest uses the average tree path length as the anomaly score; whereas RecForest exploits the bounding boxes in the forest to perform sample reconstruction for each test sample. In addition, iForest is unable to identify outlying attributes of an anomaly. This is demonstrated in Fig 1 (c), where iForest produces a large anomaly score for each anomaly, but no other information can be further extracted from the score. In contrast, the same figure shows that one anomaly deviates from normal samples mainly on the X-axis; and the other mainly from the Y-axis, which can be successfully identified by comparing differences between the original sample and its reconstructed sample produced by RecForest.

1.2 Time Complexities. In the training stage, RecForest generates m completely random trees with maximum tree depth h from a training set of n samples.

[*]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China. {xuyx, pangm, fengj, tingkm, jiangy, zhouzh}@lamda.nju.edu.cn

[†]JD.com, Beijing, China

[‡]Baiont Technology, Nanjing, China.

[§]Sinovation Ventures AI Institute, Beijing, China

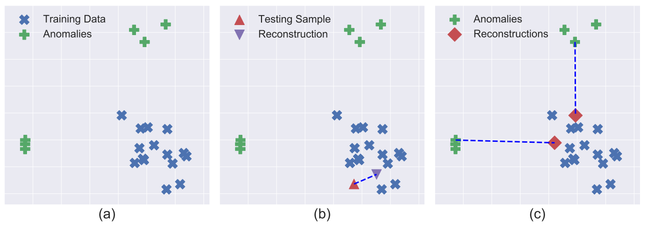

Figure 1: Demonstration of RecForest using a 2-dim toy dataset. (a) Training data, and randomly-generated anomalies; (b) The reconstructed sample for a normal sample; (c) Reconstructed samples for two anomalies. Dashed lines indicate the distance between each anomaly and its reconstructed sample.

Therefore, the overall complexity of the training stage is $O(nmh)$, which is linear to the size of training data.

In the evaluation stage, RecForest requires traversal through every tree to find the bounding boxes for each test sample. Therefore the time complexity grows linearly with the testing set size t and the number of decision trees m. Since the number of internal nodes traversed in each tree is upper bounded by the maximum tree depth h, the overall time complexity of the evaluating stage is $O(tmh)$. Consider that h and m are both constants, the testing time complexity is linear to the testing set size only.

Table 1 summarizes the time complexities of RecForest, iForest, and the neural network autoencoder. For the neural network autoencoder, we denote the cost of a forward pass and back-propagation on a sample as f and b, respectively. They are mainly related to operations on matrix multiplication in the neural network, and can be considered as constants once the architecture of autoencoder is determined. In addition, e is the number of training epochs. For iForest, ψ is the subsampling size which is a constant typically smaller than the size of training set n.

As shown in Table 1, the training time complexity of RecForest is larger than iForest because all training samples are used in RecForest. However, as shown in the runtime comparison part, the difference is small in practice as internal nodes in completely random trees

Table 1: Runtime complexities of the neural network autoencoder, iForest, and RecForest.

Model	Training	Evaluating
Autoencoder	$O(en(f+b))$	$O(tf)$
iForest	$O(m\psi \log \psi)$	$O(tm \log \psi)$
RecForest	$O(nmh)$	$O(tmh)$

randomly select an attribute and cut-off for splitting. On the other hand, the training cost on autoencoder is much greater than iForest and RecForest because the iterative optimization on parameters over many training epochs is required to achieve good performance.

2 Experiment

2.1 Configuration. Setting hyper-parameters for the neural network autoencoder is highly heuristic, we use a deep neural network with five hidden layers. The number of hidden neurons is chosen from $\{1, 2^1, 2^2, \cdots, 2^7\}$ via grid search, satisfying that: (1) The architecture of the neural network is symmetric; (2) The number of neurons in the first and last hidden layer is smaller than the number of input dimensions. The Relu activation function and Dropout [4] with probability 0.2 are used to mitigate the overfitting problem. For DAGMM, we use the publicly available implementation[1]. The structure of the compression network is set as $12-4-1-4-12$, and the size of the estimation network is $3-10-2$. Remaining hyper-parameters are set as values suggested in the original paper [5]. For VAE, we use the implementations in PyOD[2]. The network structure is same as the deep autoencoder. Since Autoencoder, DAGMM, and VAE are all neural networks that require iterative optimization on network parameters, we use the Adam optimizer to update network parameters [2]. The learning rate, number of training epochs, and batch size are further determined via grid search. The search range is listed as follows:

- Learning rate: $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$;
- Number of epochs: Discrete uniform over $[30, 100]$;
- Batch size: $\{2^5, 2^6, \cdots, 2^9\}$;
- Weight decay: $\{0, 5 \times 10^{-4}, 10^{-3}, 10^{-2}\}$

For OC-SVM and iForest, we use the implementations in Scikit-Learn with the default parameter settings[3]. For PIDForest, we use the code provided by the authors[4]. Same as iForest and RecForest, the number of trees is set as 100. Remaining hyper-parameters are set as the suggested values [6].

[1]https://github.com/danieltan07/dagmm
[2]https://pyod.readthedocs.io/
[3]https://scikit-learn.org/stable/index.html

2.2 Reconstruction Comparison. In this section, we examine the differences between RecForest and other reconstruction-based methods. The deep autoencoder is selected because it achieves better performance than DAGMM and VAE on anomaly detection accuracy. We investigate the capability of RecForest and deep autoencoder on reconstructing normal samples and anomalies, which is critical to their anomaly detection performance. The goal is to validate that RecForest is able to provide a larger reconstruction gap between normal data and anomalies compared to neural network autoencoders.

For comparison, a performance measure called anomaly exhibition indicator (AEI) is defined, which computes the ratio of the mean reconstruction error on all anomalies in the testing set to that on all testing normal samples:

$$\text{AEI} = \left(\frac{\sum_{\mathbf{x} \in \text{Anoamly}} l(\mathbf{x}, \mathbf{x}^{\text{rec}})}{n_{\text{Anoamly}}} \right) \Big/ \left(\frac{\sum_{\mathbf{x} \in \text{Normal}} l(\mathbf{x}, \mathbf{x}^{\text{rec}})}{n_{\text{Normal}}} \right),$$

where n_{Normal} and n_{Anoamly} denote the number of normal samples and anomalies in the testing set, and $l(\mathbf{x}, \mathbf{x}^{\text{rec}})$ is the reconstruction error. In reconstruction-based anomaly detectors, anomalies are considered to be much harder to be accurately reconstructed than normal samples [1]. Therefore, a larger value of AEI indicates the better performance of reconstruction-based methods. Notice that this metric is computed based on the entire testing set and ground-truth, and is only used to evaluate reconstruction-based anomaly detectors after the evaluating stage.

Table 2 presents the AEI of RecForest and the neural network autoencoder on all benchmark datasets. First, it can be observed that the value of AEI is positively correlated to the performance of the two methods shown in Table 2 and 3 of the main paper. For instance, both methods have AEI ≤ 1 on the dataset optdigits. Accordingly, the average AUC and Precision@K of the two methods are also low on the dataset optdigits. Second, the AEI of RecForest is much larger than the neural network autoencoder in general. This result indicates that RecForest is better at distinguishing anomalies from normal samples using the reconstruction error. The reasons of these differences are as follow:

The model complexity of the neural network autoencoder is fixed once its architecture is determined. However, there is no general rule on how to design a

[4]https://github.com/vatsalsharan/pidforest

Table 2: Comparison between the neural network autoencoder and RecForest using the metric anomaly exhibition indicator (AEI).

Dataset	**Autoencoder**	**RecForest**
glass	1.1	190.2
ionosphere	2.5	127.0
wbc	7.0	430.4
vowels	1.2	48.2
letter	1.0	6.8
musk	2.9	35.6
optdigits	0.7	0.8
satimage-2	5.9	377.4
satellite	2.1	211.1
pendigits	2.1	12.3
shuttle	12.5	13697.9

good architecture in practice. As a result, when the training set is small, the neural network autoencoder with many parameters are unlikely to be well-trained. Its performance can be unpredictable in regions with few training samples caused by over-fitting.

On the other hand, the model complexity of completely random forest is proportional to the data size. For instance, given n training samples, the average tree depth in completely random forest will be $O(\log n)$ [3]. Therefore, RecForest will not encounter the issue that the training set is not large enough for the model to be well-trained. Such data-dependent characteristic is critical to RecForest's higher detection accuracy on anomaly detection, especially on a large number of small datasets where the neural network autoencoder cannot learn well from normal data.

2.3 Robustness against Irrelevant Attributes. In this section, we provide details on the high-dimensional dataset used in Section 5.4.

Normal samples are sampled from 10 Gaussian clusters in $\mathbb{R}^{1000}$. We set the standard deviation of Gaussian cluster as 0.5, and ensure that the centroids of different clusters are non-overlapping. Anomalies are generated by adding one offset to randomly selected normal samples. For each Gaussian cluster, we randomly set r percentage of attributes as relevant attributes and generate 500 normal samples, whereas other attributes of normal samples are replaced with random noise being uniform in $[-1, 1]$. Besides, 10 normal samples are randomly selected from each cluster, treated as anomalies after adding one offset being unifrom in $[-2, 2]$ onto each relevant attribute.

References

[1] C. C. Aggarwal, "Outlier analysis," Data Mining, Springer, 2015.

[2] D. P. Kingma, J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[3] F.-T. Liu, K.-M. Ting and Z.-H. Zhou, "Isolation forest," in *ICDM*, 2008, pp. 413-422.

[4] N. Srivastava, G. E. Hinton, A. Krizhevsky, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[5] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu et al., "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *ICLR*, 2018.

[6] P. Gopalan, V. Sharan and U. Wieder, "PIDforest: Anomaly detection via partial identification," in *NeurIPS*, 2019, pp. 15783-15793.