Unsupervised log message anomaly detection

Amir Farzad & T. Aaron Gulliver

July 2020

KICS The Korean Institute of Communications and Information Sciences

Available online at www.sciencedirect.com
ScienceDirect

ICT Express 6 (2020) 229–237

www.elsevier.com/locate/icte

Unsupervised log message anomaly detection

Amir Farzad*, T. Aaron Gulliver

Department of Electrical and Computer Engineering, University of Victoria, PO Box 1700, STN CSC, Victoria, BC, V8W 2Y2, Canada

Received 17 February 2020; received in revised form 11 June 2020; accepted 25 June 2020
Available online 2 July 2020

Abstract

Log messages are now broadly used in cloud and software systems. They are important for classification and anomaly detection as millions of logs are generated each day. In this paper, an unsupervised model for log message anomaly detection is proposed which employs Isolation Forest and two deep Autoencoder networks. The Autoencoder networks are used for training and feature extraction, and then for anomaly detection, while Isolation Forest is used for positive sample prediction. The proposed model is evaluated using the BGL, Openstack and Thunderbird log message data sets. The results obtained show that the number of negative samples predicted to be positive is low, especially with Isolation Forest and one Autoencoder. Further, the results are better than with other well-known models.

Keywords: Anomaly detection; Classification; Deep learning; Log messages; Unsupervised learning

1. Introduction

Software systems and applications such as online search engines and cloud servers are now widespread. Availability is expected 24/7 and access failures can cause considerable hardship to both organizations and clients. Thus, significant amounts of money are spent to maintain the quality and availability of these services. This is achieved using log messages which are generated to indicate the system state. Storing records for audit or security purposes is called logging [1]. Each logging statement produces log messages related to a specific task. Unstructured log messages consist of runtime data which include verbosity, time stamps and raw content that is a summary of the system operation. The structure can vary significantly in form, making it difficult to identify abnormalities in these logs [2].

Log messages are used for many tasks such as performance monitoring [3] and anomaly detection [4,5]. Most methods use rules to detect abnormalities in logs, but this requires specific domain knowledge [6]. Some consider only one log component such as the time stamps or verbosity which reduces the ability to recognize anomalies. Anomaly detection can be done manually, but this is not feasible for large-scale frameworks because of the complexity and amount of data [7]. Thus, automated log analysis techniques to identify anomalies are needed.

Deep Learning (DL) employs networks with multiple connected layers and belongs to the class of machine learning (ML) techniques. DL can reduce data complexity and find similarities between data [8]. Thus, DL techniques are very suitable for big data. They can also be used for feature extraction and to reduce data dimensionality [9]. DL has produced excellent results for applications in image processing, text classification and Natural Language Processing (NLP) [10]. These techniques can be classified as generative, discriminative or hybrid methods. Discriminative methods are typically used for supervised classification. Generative methods are unsupervised and typically used to reveal patterns in unlabeled data. Hybrid methods combine discriminative and generative methods [11].

Several anomaly detection methods have been developed using ML and DL algorithms. Gaussian Mixture Model (GMM) and Variational Bayesian Gaussian Mixture Model (BGM) are probabilistic models which have been used in intrusion detection systems [12]. Elliptic Envelope (EEnvelope) generates an elliptical space around the center of mass of the data and has been employed for detecting anomalies in audio sensors [13]. Local Outlier Factor (LOF) measures the local deviation of data and has been used for detecting

* Corresponding author.
E-mail addresses: amirfarzad@uvic.ca (A. Farzad), agullive@ece.uvic.ca (T.A. Gulliver).
Peer review under responsibility of The Korean Institute of Communications and Information Sciences (KICS).

https://doi.org/10.1016/j.icte.2020.06.003

anomalies in traffic data [14]. One-Class Support Vector Machine (OC-SVM) has been employed for anomaly detection in networks [15].

A supervised method was proposed in [16] to detect errors and anomalies in logs using decision trees. A hybrid model was presented in [17] to detect anomalies using K-means clustering and decision trees. The anomaly detection method proposed in [18] employs stacked Long Short-Term Memory (LSTM) networks. The online method given in [19] employs an LSTM network and provides better results than Isolation Forest. Deeplog [20] uses a deep learning LSTM network to predict anomalies but it has high computational complexity and only positive (normal) logs are used for training [21]. The proposed model uses both normal and abnormal (negative) logs for training and has lower complexity than other DL algorithms in the literature.

One of the main challenges in anomaly detection is dealing with unlabeled data. Millions of log messages are produced each day in cloud and other systems. Unfortunately, labeling even a small portion of these logs for model training is not possible. One way to deal with this unlabeled data is to use unsupervised methods. Isolation Forest [22] is an ensemble approach that isolates abnormal samples to detect anomalies. It has been used for tasks such as anomaly detection in sensor data [23] and intrusion detection [24]. Isolation Forest has high accuracy and linear time complexity, but it may perform poorly with large and complex data [25]. Recently, Extended Isolation Forest was proposed in [26] to improve anomaly detection with Isolation Forest. Extended Isolation Forest employs slopes and intercepts whereas Isolation Forest uses attributes and values. Functional Isolation Forest was proposed in [27] to detect anomalies in functional data using Isolated Forest. In this model, anomalies are detected using a collection of Functional Isolation Trees [27].

Isolation Forest has been used to find abnormal data, but in this paper it is used to detect normal data with a threshold. An Autoencoder [28] is used to extract features for Isolation Forest. An Autoencoder is a feed-forward Artificial Neural Network (ANN) which can learn data features [29]. These networks have been used for a number of tasks including interpolation [30], representation learning [31] and generative modeling [32]. In an Autoencoder network, the code layer (a hidden layer) is used for feature extraction and dimensionality reduction so the size of this layer is typically small [33]. However, in this paper the last layer output is used to extract features so the code layer need not be smaller than the input. In addition, most state-of-the-art ML/DL methods such as Deeplog use customized parsing models, but in the proposed model text logs with only simple pre-processing are employed. The proposed model is evaluated using the accuracy, precision, recall and F-measure metrics with three log message data sets, namely BlueGene/L (BGL),[1] Openstack[2] and Thunderbird.[3]

The main contributions of this paper are as follows.

1. Isolation Forest is used to find positive logs rather than anomalies.
2. An Autoencoder network is used to extract features for Isolation Forest.

The proposed architecture includes two Autoencoder networks and Isolation Forest. Feature extraction using Autoencoders has two main advantages. First, it improves the Isolation Forest results as Isolation Forest alone does not provide good results for log message anomaly detection. The use of an Autoencoder with Isolation Forest has not been previously investigated. Further, Isolation Forest is used to predict positive data instead of negative data and a threshold is employed. Second, an Autoencoder is well suited to separating anomalies from normal data so two Autoencoders are used to provide better separation.

The rest of this paper is organized as follows. In Section 2, the Isolation Forest and Autoencoder architectures are presented and the proposed model is described. The results and discussion for the three data sets are given in Section 3. Finally, some concluding remarks are given in Section 4.

2. System model

In this section, the Isolation Forest and Autoencoder architectures are given along with the proposed model.

2.1. Isolation forest

Isolation Forest [22] is an ensemble approach which has been used to detect anomalies. It begins with a random attribute and chooses a partition between the lowest and highest values in order to separate a sample. This continues until the samples are isolated. The Isolation Forest is built by adding a number of Isolation Trees separated into different attributes. The number of partitions needed to isolate a sample is equal to the path length passed from the root to a leaf.

The Isolation Tree technique is based on Extra Trees [34]. In Isolation Tree, each division is random. A sample located near the root, i.e. its path is short, implies that it is easier to distinguish and therefore easier to isolate from samples that are in deeper leaves. It is expected that anomalies (abnormal samples) will have a smaller average path length than positive (normal) samples. When a sample is at a leaf deep in the tree, the score will be low (close to 0), while if it is shallow the score will be high (close to 1). The anomaly score of a sample x with Isolation Forest is

$$s(x, N) = 2^{\frac{-E(h(x))}{c(N)}}, \tag{1}$$

where N is the number of samples trained in each tree of the forest, $E(h(x))$ is the average length of the paths in all trees, and $c(N)$ is the normalization factor which is [35]

$$c(N) = \begin{cases} 2H(N-1) - 2(N-1)/N & \text{if} \quad N > 2, \\ 1 & \text{if} \quad N = 2, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

where $H(N-1)$ is the harmonic number given by $\ln(N-1) + 0.5772156649$.

[1] https://github.com/logpai/loghub/tree/master/BGL
[2] https://github.com/logpai/loghub/tree/master/OpenStack
[3] https://github.com/logpai/loghub/tree/master/Thunderbird

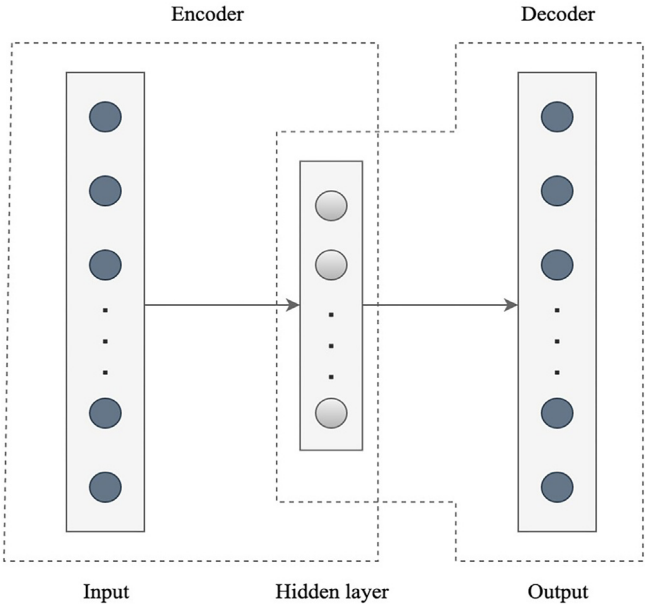

Fig. 1. Autoencoder architecture with an input layer, output layer, and one hidden layer.

2.2. Autoencoder architecture

An Autoencoder [28] is a feed-forward multilayer neural network with the same number of input and output units. Training is performed using a loss function to ensure that the output is close to the input. The aim is to learn a compact representation while minimizing the error for the input data. A deep Autoencoder is an Autoencoder which has more than one hidden layer [36]. It can represent complex distributions if multiple encoder and decoder layers are employed. The encoder and decoder outputs are

$$y = a(Wx + b), \tag{3}$$

and

$$z = a(W'y + b'), \tag{4}$$

respectively, where x is the input, W is the encoder weight matrix, b is the encoder bias vector, W' is the decoder weight matrix, b' is the decoder bias vector, and a is the activation function. Fig. 1 shows the architecture of an Autoencoder with an input layer, output layer, and one hidden layer.

2.3. Proposed model

The proposed model for unsupervised anomaly detection includes two deep Autoencoder networks and an Isolation Forest. First, text pre-processing including tokenization and changing letters to lowercase is applied to the data set. Next, the sentences are padded to 40 words and sentences with less than five words are eliminated. The word frequency is then computed and the data is shuffled. Next, the data set is normalized and scaled between 0 and 1. Then it is divided into the first training set t_1 (around 0.5% for BGL and Thunderbird, and 5% for Openstack), second training set t_2 (around 2% for BGL and Thunderbird, and 17% for Openstack) and testing set t_3 (around 97.5% for BGL and Thunderbird, and 78% for Openstack). The proportion of positive and negative logs in these sets is the same as in the original data sets. The sets t_1 and t_2 are small as their size affects the speed of the algorithm. The BGL and Thunderbird data sets are larger than Openstack so a smaller proportion of the data is used for these sets. Further, the Openstack data set is small so a larger proportion of the data is needed for training convergence.

The first Autoencoder is used for feature extraction from a small amount of data in an unsupervised way. It is trained with t_1 (which contains both positive and negative log messages). This Autoencoder has an encoder layer with 512 units and L1 regularizer, followed by a decoder layer with 256 units. The output layer has the same size as the input which is 40 units. The categorical cross-entropy loss function, Relu activation function and ADAM optimizer are used to train this model with a batch size of 64 and a maximum of 500 training epochs. The ADAM optimizer is used because it has fast convergence and good performance in DL algorithms [37]. Early stopping is used to prevent overfitting. After training the first Autoencoder, the second training set t_2 and testing set t_3 are fed into this Autoencoder to extract features (as the last layer output which has the same size as the input), which are denoted as f_2 and f_3, respectively.

Feature set f_2 is fed into the Isolation Forest with 100 Isolation Trees and this is tested using f_3 to predict the data. Next, a percentage of the positive predicted data (30% for BGL and Thunderbird, and 70% for Openstack), is randomly chosen from the Isolation Forest output and this is denoted as p_1. The remaining positive predicted data and the negative predicted data are denoted as p_2. p_1 is used to train the second Autoencoder which has the same architecture as the first Autoencoder. The maximum number of training epochs is 100 and early stopping is used to prevent overfitting. After training the second Autoencoder, p_1 is input to this Autoencoder to extract features (as the last layer output of the trained Autoencoder), which is denoted as o_1, and then p_2 is input to this Autoencoder and the extracted features are denoted as o_2. Anomalies are detected using a threshold. To determine the threshold, the average and standard deviation of the feature values of each sample in o_1 are computed. The threshold is given by

$$T = stdv \times c, \tag{5}$$

where $stdv$ is the standard deviation of the positive data predicted with Isolation Forest (o_1) and c is a constant. For unlabeled data, this constant can be estimated based on the final predicted test data. In the data sets, it is known that about 10% of the data is negative and the remainder positive. Thus, the constant can be chosen based on the percentages obtained. The larger data sets (BGL and Thunderbird) were divided into smaller size sets and used in the proposed model. It was found that smaller data sets require a larger constant. Based on the results, the constants for BGL, Thunderbird and Openstack are $c = 0.5$, 0.1 and 5, respectively.

Algorithm 1 Proposed Model Algorithm

Require: first training set: t_1; second training set: t_2; testing set: t_3.

for epochs **do**
    Train the first Autoencoder with t_1.
Compute f_2 and f_3 using t_2 and t_3, respectively.
Train Isolation Forest with f_2 and test with f_3.
Randomly select a percentage of the positive predicted logs as p_1, and the remaining predicted logs are p_2.
for epochs **do**
    Train the second Autoencoder with p_1.
Compute o_1 and o_2 using p_1 and p_2, respectively.
Compute the average and standard deviation of the feature values of each sample in o_1 and determine the threshold using (5).
Compute the average of the feature values of each sample in o_2.
If an average is less than the threshold, flag the sample as an anomaly.

The second Autoencoder needs fewer epochs to train because the features have been extracted with the first Autoencoder. Further, there is more input data for the second Autoencoder than the first Autoencoder, and typically deep learning algorithms need large amounts of data to work well. The loss and accuracy of the model were used to determine the number of epochs. The proposed model algorithm is given in Algorithm 1 and the architecture of the proposed model is given in Fig. 2.

3. Results

In this section, the proposed model is evaluated using the BGL, Openstack and Thunderbird data sets. Four criteria, namely accuracy, precision, recall and F-measure, are used to evaluate the performance. Accuracy is the percentage of the data that is correctly predicted and is given by

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}, \tag{6}$$

where T_p is the number of positive samples which are predicted by the model to be positive, F_p is the number of negative samples which are predicted to be positive, T_n is the number of negative samples which are predicted to be negative, and F_n is the number of positive samples which are predicted to be negative. Precision is expressed as

$$P = \frac{T_p}{T_p + F_p}, \tag{7}$$

and recall is

$$R = \frac{T_p}{T_p + F_n}. \tag{8}$$

The F-measure is given by

$$F = \frac{2 \times P \times R}{P + R}. \tag{9}$$

All experiments were run on the Compute Canada Cedar cluster with 24 CPU cores, four P100 GPUs and 125 GB of memory, and the algorithms were implemented using Python in Keras[4] and Scikit-learn.[5]

The hyperparameters of the proposed model were not tuned so for all data sets the default values were used. Each experiment was repeated 10 times, and the minimum, maximum and average testing accuracy, precision, recall, F-measure and time were obtained. Table 1 gives the results for the BGL, Openstack and Thunderbird data sets for (a) Isolation Forest, (b) Isolation Forest with one Autoencoder, and (c) Isolation Forest with two Autoencoders (proposed model). For Isolation Forest alone, the first training set t_1 is input to the Isolation Forest and the results are obtained using the testing set t_3. For Isolation Forest with one Autoencoder, the first training set t_1 is input to the first Autoencoder for training. Then the second training set t_2 and testing set t_3 are input to this trained Autoencoder for feature extraction, giving f_2 and f_3, respectively. Then f_2 is fed into the Isolation Forest for training and the results are obtained using f_3. Isolation Forest with two Autoencoders indicates that the results are obtained with the second Autoencoder using a threshold (proposed model).

3.1. BGL

The BlueGene/L (BGL) data set has 4,399,502 positive logs and 348,460 negative logs. From these, 23,997 logs are used for the first training set, 93,551 for the second training set and the remaining 4,630,414 for testing. With the Isolation Forest model, the average testing accuracy is 88.7% with average precision, recall and F-measure of 28.9%, 36.6% and 32.3% for negative logs, and 94.8%, 92.8% and 93.8% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 90.8% with average precision, recall and F-measure of 42.7%, 76.5% and 54.3% for negative logs, and 98.0%, 91.9% and 94.9% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.6% with average precision, recall and F-measure of 96.8%, 98.7% and 98.1% for negative logs, and 99.8%, 99.7% and 99.8% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 82.5%, 94.7% and 88.2%, respectively, with the nLSALog algorithm [38], and the 83%, 99% and 91%, respectively, with SVM unsupervised learning [39]. The precision, recall and F-measure results for negative logs are also better than the 92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbor supervised algorithm [40]. Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the BGL data set with the Variational Bayesian Gaussian Mixture Model (BGM), Elliptic Envelope (EEnvelope), Gaussian Mixture Model (GMM),

[4] https://github.com/keras-team/keras
[5] https://github.com/scikit-learn/scikit-learn

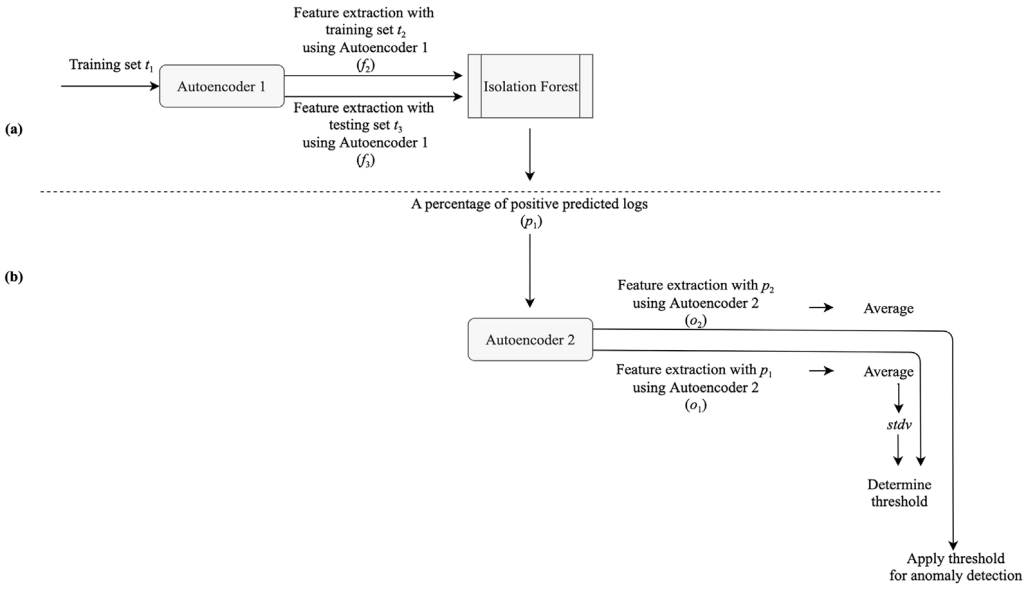

Fig. 2. Architecture of the proposed model: (a) Isolation Forest with one Autoencoder, and (b) the second Autoencoder for anomaly detection (Isolation Forest with two Autoencoders).

Table 1
The testing accuracy, precision, recall, F-measure and time for (a) Isolation Forest, (b) Isolation Forest with one Autoencoder and (c) Isolation Forest with two Autoencoders. The minimum, maximum and average (in parenthesis) values for the BGL, Openstack and Thunderbird data sets for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Data set	Testing accuracy	Label	Precision	Recall	F-measure	Time (s)
(a)	BGL	88.6%–(88.7%)–88.8%	0	28.1%–(28.9%)–29.7%	35.2%–(36.6%)–38.1%	31.2%–(32.3%)–33.4%	276
			1	94.7%–(94.8%)–95.0%	92.8%–(92.8%)–92.9%	93.7%–(93.8%)–93.9%	
	Openstack	86.9%–(86.9%)–87.0%	0	54.5%–(59.2%)–64.0%	52.5%–(57.6%)–62.8%	53.5%–(58.4%)–63.4%	6
			1	92.1%–(92.9%)–93.8%	92.7%–(93.4%)–94.1%	92.4%–(93.1%)–93.9%	
	Thunderbird	78.8%–(79.0%)–79.3%	0	29.9%–(30.8%)–31.8%	19.3%–(19.6%)–20.0%	23.5%–(24.0%)–24.6%	158
			1	84.8%–(84.8%)–84.9%	90.8%–(91.0%)–91.3%	87.7%–(87.8%)–88.0%	
(b)	BGL	90.6%–(90.8%)–91.0%	0	40.2%–(42.7%)–45.2%	57.1%–(76.5%)–95.9%	47.2%–(54.3%)–61.4%	863
			1	96.5%–(98.0%)–99.6%	90.6%–(91.9%)–93.3%	94.9%–(94.9%)–94.9%	
	Openstack	92.9%–(92.9%)–93.0%	0	66.9%–(67.1%)–67.3%	99.4%–(99.5%)–99.7%	80.1%–(80.2%)–80.3%	171
			1	99.5%–(99.7%)–99.9%	91.8%–(91.8%)–91.9%	95.5%–(95.6%)–95.7%	
	Thunderbird	81.8%–(82.9%)–84.1%	0	45.6%–(49.9%)–54.2%	41.9%–(49.6%)–57.4%	43.7%–(49.3%)–54.9%	652
			1	88.4%–(89.8%)–91.2%	89.5%–(90.7%)–92.0%	89.1%–(89.9%)–90.8%	
(c)	BGL	99.4%–**(99.6%)**–99.9%	0	94.1%–**(96.8%)**–99.5%	97.8%–**(98.7%)**–99.6%	96.6%–**(98.1%)**–99.6%	1499
			1	99.8%–(99.8%)–99.9%	99.5%–(99.7%)–99.9%	99.7%–(99.8%)–99.9%	
	Openstack	98.4%–**(99.1%)**–99.7%	0	93.5%–**(96.1%)**–98.7%	95.7%–**(97.5%)**–99.3%	94.6%–**(96.8%)**–99.0%	366
			1	99.3%–(99.6%)–99.9%	98.9%–(99.3%)-99.8%	99.1%–(99.4%)–99.8%	
	Thunderbird	99.0%–**(99.4%)**–99.9%	0	94.7%–**(97.2%)**–99.8%	97.6%–**(98.6%)**–99.6%	97.2%–**(98.4%)**–99.7%	1158
			1	99.6%–(99.7%)–99.9%	98.9%–(99.4%)–99.9%	99.4%–(99.6%)–99.9%	

K-means, Local Outlier Factor (LOF), and One-Class Support Vector Machine (OC-SVM) algorithms using 10-fold cross-validation are given in Table 2. The results show that the proposed model is significantly better than these algorithms. Note that only 5% of the data set was used for LOF and OC-SVM due to the high computational complexity of these algorithms.

3.2. Openstack

The Openstack data set has 137,074 positive log messages and 18,434 negative log messages. From these, 7353 logs are used for first training set, 26,545 for the second training set and the remaining 121,610 for testing. With the Isolation Forest model, the average testing accuracy is 86.9% with average

Table 2
The results for the BGL data set with average testing accuracy, precision, recall, F-measure and time for BGM, EEnvelope, GMM, K-means, LOF and OC-SVM algorithms using 10-fold cross-validation. Positive labels are denoted by 1 and negative labels by 0.

Algorithm	Testing accuracy	Label	Precision	Recall	F-measure	Time (s)
BGM	59.4%	0	42.8%	60.0%	50.0%	3892
		1	71.7%	59.4%	61.4%	
EEnvelope	85.2%	0	12.8%	17.5%	14.8%	6197
		1	93.3%	90.6%	91.9%	
GMM	68.0%	0	46.0%	61.8%	52.5%	1928
		1	76.8%	68.5%	69.9%	
K-means	48.6%	0	1.3%	10.0%	2.3%	18571
		1	88.1%	51.7%	65.1%	
LOF	83.5%	0	7.0%	10.2%	8.3%	1462
		1	92.6%	89.3%	90.9%	
OC-SVM	84.3%	0	8.4%	11.4%	9.7%	56 818
		1	92.7%	90.1%	91.4%	

Table 3
The results for the Openstack data set with average testing accuracy, precision, recall, F-measure and time for BGM, EEnvelope, GMM, K-means, LOF and OC-SVM algorithms using 10-fold cross-validation. Positive labels are denoted by 1 and negative labels by 0.

Algorithm	Testing accuracy	Label	Precision	Recall	F-measure	Time (s)
BGM	57.5%	0	31.6%	60.0%	37.1%	95
		1	77.0%	57.2%	63.3%	
EEnvelope	80.8%	0	16.8%	15.6%	16.2%	318
		1	88.8%	89.6%	89.2%	
GMM	58.0%	0	37.6%	50.0%	40.3%	81
		1	71.3%	59.1%	62.5%	
K-means	40.0%	0	40.0%	40.0%	40.0%	378
		1	40.0%	40.0%	40.0%	
LOF	80.2%	0	14.2%	13.2%	13.7%	1541
		1	88.4%	89.3%	88.8%	
OC-SVM	38.5%	0	0.3%	1.3%	0.5%	41 280
		1	76.6%	43.5%	55.5%	

precision, recall and F-measure of 59.2%, 57.6% and 58.4% for negative logs, and 92.9%, 93.4% and 93.1% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 92.9% with average precision, recall and F-measure of 67.1%, 99.5% and 80.2% for negative logs, and 99.7%, 91.8% and 95.6% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.1% with average precision, recall and F-measure of 96.1%, 97.5% and 96.8% for negative logs, and 99.6%, 99.3% and 99.4% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are similar to the 94%, 99% and 97% obtained with the Deeplog network [20]. Experiments were also conducted with several well-known anomaly detection algorithms. The average testing accuracy, precision, recall, F-measure and time for the Openstack data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 3. The results show that the proposed model is significantly better than these algorithms.

3.3. Thunderbird

The Thunderbird data set has 3,000,000 positive log messages and 600,000 negative log messages. From these, 17,000 messages are used for the first training set, 65,700 for the second training set and the remaining 3,517,300 for testing. With the Isolation Forest model, the average testing accuracy is 79.0% with average precision, recall and F-measure of 30.8%, 19.6% and 24.0% for negative logs, and 84.8%, 91.0% and 87.8% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 82.9% with average precision, recall and F-measure of 49.9%, 49.6% and 49.3% for negative logs, and 89.8%, 90.7% and 89.9% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.4% with

average precision, recall and F-measure of 97.2%, 98.6% and 98.4% for negative logs, and 99.7%, 99.4% and 99.6% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 96%, 96% and 96%, respectively, with the Improved K-Nearest Neighbor supervised algorithm [40]. Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the Thunderbird data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 4. The results show that the proposed model is significantly better than these algorithms. Note that only 5% of the data set was used for LOF and OC-SVM due to the high complexity of these algorithms.

3.4. Discussion

Isolation Forest has been shown to provide good results for anomaly detection problems [24]. However, it is used in this paper to predict positive logs. Table 1(a) shows that Isolation Forest alone does not provide good results for the problem considered here, as the precision, recall and F-measure for the negative logs in all data sets are poor. However, the corresponding results for positive logs are much better. Comparing the results in Table 1(a) with Tables 2–4, it is evident that the precision for positive logs with Isolation Forest is better than that with the other algorithms even though a smaller portion of the data was used for training. Also, Isolation Forest is faster because it has linear time complexity [22]. Table 1(b) shows the effect of feature extraction with an Autoencoder before Isolation Forest. These results indicate that most of the criteria for the positive and negative logs are improved for all three data sets. However, the negative log results are still poor. For the BGL data set, the average precision, recall and F-measure are improved from 28.9%, 36.6% and 32.3% to 42.7%, 76.5% and 54.3%, respectively, for negative logs while for positive logs the corresponding results are 94.8%, 92.8% and 93.8% to 98.0%, 91.9% and 94.9%. While the recall for the positive logs has decreased slightly, here precision is the most important criteria. For the Openstack data set, the average precision, recall and F-measure are mostly improved from 59.2%, 57.6% and 58.4% to 67.1%, 99.5% and 80.2% for negative logs and from 92.9%, 93.4% and 93.1% to 99.7%, 91.8% and 95.6% for positive logs (only the recall for the positive logs has decreased). For the Thunderbird data set, the average precision, recall and F-measure are improved from 30.8%, 19.6% and 24.0% to 49.9%, 49.6% and 49.3%, respectively, for negative logs while for positive logs the corresponding results are 84.8%, 91.0% and 87.8% to 89.8%, 90.7% and 89.9%. Again, the recall for the positive logs has decreased slightly.

Precision for positive logs is the percentage of true positive logs detected out of all logs detected as positive, while recall for positive logs is the percentage of the positive logs that are detected. Table 1(b) shows that the precision for the positive logs is very reliable with 90% to 99% accuracy. Thus, it can be concluded that most logs which are predicted to be positive are correct. In other words, the number of negative logs predicted to be positive (F_P) is low with just Isolation Forest (especially with one Autoencoder). This reliable positive data (p_1) is used to train a second Autoencoder for anomaly detection with a threshold. Table 1(c) gives the test data results with this Autoencoder which has been trained with p_1. Average precision, recall and F-measure for positive and negative logs are improved significantly for all data sets to more than 96%.

Isolation Forest with one Autoencoder is the most important step. Autoencoder networks are typically used for dimensionality reduction and so have fewer units in the hidden layers. However, we use a higher number of units in these layers so the input size is the same as the output. Binary cross-entropy is typically used in Autoencoder networks, but it was found that with categorical cross-entropy the averages of the feature values for each sample in o_2 are more separable into positive and negative logs, which is desirable here. This means that most of the positive logs have high average values and most of the negative logs have low average values. Further, it was noticed that the values for unimportant features (for example features that are repeated in the data set), are near zero with categorical cross-entropy and Relu activation without dimensionality reduction in the code layer. This indicates that the Autoencoder network works well for feature extraction with log messages.

The proposed unsupervised method has three advantages over supervised methods. First, knowledge via log message labels is not required making this approach suitable for many practical applications. This is important as there are a vast variety of systems producing different log messages which makes it difficult to label data for supervised methods. Second, labeling data is a very time-consuming task because of the volumes of data and so is not a practical solution. Third, using an unsupervised method eliminates the human error inherent in labeling. Although supervised methods typically provide better performance than unsupervised methods, the proposed unsupervised method is better than the Improved K-Nearest Neighbor supervised algorithm [40] for the BGL and Thunderbird data sets. This is because they use a simple supervised machine learning algorithm. Conversely, the proposed model employs deep learning to extract features via deep Autoencoder networks.

Only a small amount of training data was used (less than 3% for BGL and Thunderbird, and 22% for Openstack for Isolation Forest with one Autoencoder, and less than 30% for BGL and Thunderbird for Isolation Forest with two Autoencoders), whereas training deep networks usually requires a significant amount of data for convergence. More training data was needed for the Openstack data set because it is small (around 25 times smaller than Thunderbird and 30 times smaller than BGL). The features extracted using the Autoencoder network were used as the input for Isolation Forest. This feature extraction required only a very small amount of training data, so the execution time was fast (1499 s for BGL, 1158 s for Thunderbird and 366 s for Openstack after pre-processing). Further, an Autoencoder network is a very

Table 4
The results for the Thunderbird data set with average testing accuracy, precision, recall, F-measure and time for BGM, EEnvelope, GMM, K-means, LOF and OC-SVM algorithms using 10-fold cross-validation. Positive labels are denoted by 1 and negative labels by 0.

Algorithm	Testing accuracy	Label	Precision	Recall	F-measure	Time (s)
BGM	67.0%	0	21.4%	40.1%	26.8%	3203
		1	84.2%	72.4%	74.3%	
EEnvelope	75.4%	0	10.8%	6.5%	8.1%	5027
		1	82.6%	89.3%	85.8%	
GMM	60.0%	0	25.9%	40.1%	31.5%	987
		1	74.9%	64.0%	67.4%	
K-means	68.8%	0	13.8%	16.7%	15.1%	11 976
		1	82.6%	79.2%	80.9%	
LOF	75.5%	0	17.6%	10.2%	12.9%	1306
		1	82.2%	89.6%	85.7%	
OC-SVM	40.7%	0	8.4%	23.7%	12.5%	50 926
		1	72.9%	44.3%	55.1%	

fast deep learning algorithm, especially compared to LSTM networks, and the time complexity of Isolation Forest is linear. We did not tune the hyperparameters so the results may be improved with tuning such as the learning rate and number of hidden layers.

4. Conclusion

Cloud systems are capable of generating millions of text log messages everyday. Thus, while the detection of anomalies in these logs is very important, the volume makes this a difficult task. In this paper, a model was proposed for unsupervised anomaly detection using Isolation Forest and two deep Autoencoder networks. These networks are used for feature extraction and anomaly detection. Isolation Forest is often used for anomaly detection, but here it is used to predict positive data. The proposed model was evaluated using three well-known log message data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model is better than other models employed in the literature. This is because training and feature extraction with the first Autoencoder network improves the Isolation Forest results, especially for positive log messages. Further, the number of negative logs predicted to be positive (F_p) is low with Isolation Forest, particularly with one Autoencoder. In the future, the performance of other models such as Gaussian Mixture Model and the effects of hyperparameter tuning can be investigated.

CRediT authorship contribution statement

Amir Farzad: Conceptualization, Methodology, Software, Writing - review & editing. **T. Aaron Gulliver:** Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, M.R. Lyu, Tools and benchmarks for automated log parsing, in: International Conference on Software Engineering: Software Engineering in Practice, 2019, pp. 121–130, http://dx.doi.org/10.1109/ICSE-SEIP.2019.00021.

[2] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, S. Pasupathy, SherLog: error diagnosis by connecting clues from run-time logs, in: Architectural Support for Programming Languages and Operating Systems, 2010, pp. 143–154, http://dx.doi.org/10.1145/1736020.1736038.

[3] L. Zhang, The research of log-based network monitoring system, in: G. Lee (Ed.), Advances in Intelligent Systems, Springer, Berlin, Heidelberg, 2012, pp. 315–320.

[4] T. Sipola, A. Juvonen, J. Lehtonen, Anomaly detection from network logs using diffusion maps, in: L. Iliadis, C. Jayne (Eds.), Engineering Applications of Neural Networks, Springer, Berlin, Heidelberg, 2011, pp. 172–181.

[5] Y. Harada, Y. Yamagata, O. Mizuno, E. Choi, Log-based anomaly detection of CPS using a statistical method, in: International Workshop on Empirical Software Engineering in Practice, 2017, pp. 1–6, http://dx.doi.org/10.1109/IWESEP.2017.12.

[6] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, E. Kirda, Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks, in: Annual Computer Security Applications Conference, 2013, pp. 199–208, http://dx.doi.org/10.1145/2523649.2523670.

[7] Q. Lin, H. Zhang, J. Lou, Y. Zhang, X. Chen, Log clustering based problem identification for online service systems, in: IEEE/ACM International Conference on Software Engineering, 2016, pp. 102–111.

[8] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Cambridge, MA, 2016.

[9] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: A review, Data Min. Knowl. Discov. 33 (4) (2019) 917–963, http://dx.doi.org/10.1007/s10618-019-00619-1.

[10] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, 2017, arXiv e-prints arXiv:1708.02709.

[11] E.P. Ijjina, C.K. Mohan, Hybrid deep neural network model for human action recognition, Appl. Soft Comput. 46 (2016) 936–952, http://dx.doi.org/10.1016/j.asoc.2015.08.025.

[12] M. Bahrololum, M. Khaleghi, Anomaly intrusion detection system using Gaussian mixture model, in: International Conference on Convergence and Hybrid Information Technology, 2008, pp. 1162–1167.

[13] M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, C. Perera, Smart audio sensors in the Internet of things edge for anomaly detection, IEEE Access 6 (2018) 67594–67610.
[14] M.X. Ma, H.Y.T. Ngan, W. Liu, Density-based outlier detection by local outlier factor on largescale traffic data, in: Image Processing: Machine Vision Applications, 2016, pp. 1–4.
[15] M. Zhang, B. Xu, J. Gong, An anomaly detection model based on one-class SVM to detect network intrusions, in: International Conference on Mobile Ad-Hoc and Sensor Networks, 2015, pp. 102–107.
[16] T. Reidemeister, M. Jiang, P.A.S. Ward, Mining unstructured log files for recurrent fault diagnosis, in: IFIP/IEEE International Symposium on Integrated Network Management and Workshops, 2011, pp. 377–384.
[17] A.P. Muniyandi, R. Rajeswari, R. Rajaram, Network anomaly detection by cascading K-means clustering and C4.5 decision tree algorithm, Procedia Eng. 30 (2012) 174–182, http://dx.doi.org/10.1016/j.proeng.2012.01.849.
[18] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, in: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2015, pp. 89–94.
[19] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, S. Robinson, Deep learning for unsupervised insider threat detection in structured cybersecurity data streams, 2017, arXiv e-prints arXiv:1710.00811.
[20] M. Du, F. Li, G. Zheng, V. Srikumar, DeepLog: anomaly detection and diagnosis from system logs through deep learning, in: ACM Conference on Computer and Communications Security, 2017, pp. 1285–1298, http://dx.doi.org/10.1145/3133956.3134015.
[21] M. Taghavi, M. Shoaran, Hardware complexity analysis of deep neural networks and decision tree ensembles for real-time neural data classification, in: International IEEE/EMBS Conference on Neural Engineering, 2019, pp. 407–410.
[22] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation forest, in: IEEE International Conference on Data Mining, 2009, pp. 413–422, http://dx.doi.org/10.1109/ICDM.2008.17.
[23] J. Hofmockel, E. Sax, Isolation forest for anomaly detection in raw vehicle sensor data, in: International Conference on Vehicle Technology and Intelligent Transport Systems, 2018, pp. 411–416, http://dx.doi.org/10.5220/0006758004110416.
[24] P.-F. Marteau, S. Soheily-Khah, N. Béchet, Hybrid isolation forest - application to intrusion detection, 2017, arXiv e-prints arXiv:1705.03800.
[25] X. Tao, Y. Peng, F. Zhao, P. Zhao, Y. Wang, A parallel algorithm for network traffic anomaly detection based on isolation forest, Int. J. Distrib. Sens. Netw. 14 (11) (2018) http://dx.doi.org/10.1177/1550147718814471, 1550147718814471.
[26] S. Hariri, M. Carrasco Kind, R.J. Brunner, Extended isolation forest, 2018, arXiv e-prints arXiv:1811.02141.
[27] G. Staerman, P. Mozharovskyi, S. Clémençon, F. d'Alché-Buc, Functional isolation forest, 2019, arXiv e-prints arXiv:1904.04573.
[28] D.E. Rumelhart, J.L. McClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, MIT Press, Cambridge, MA, 1986, pp. 318–362 (Ch. Learning Internal Representations by Error Propagation).
[29] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: International Conference on Machine Learning, 2008, pp. 1096–1103, http://dx.doi.org/10.1145/1390156.1390294.
[30] D. Berthelot, C. Raffel, A. Roy, I. Goodfellow, Understanding and improving interpolation in autoencoders via an adversarial regularizer, 2018, arXiv e-prints arXiv:1807.07543.
[31] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, Beta-VAE: learning basic visual concepts with a constrained variational framework, in: International Conference on Learning Representations, 2017.
[32] Z. Zhang, R. Zhang, Z. Li, Y. Bengio, L. Paull, Perceptual generative autoencoders, 2019, arXiv e-prints arXiv:1906.10335.
[33] M. Alkhayrat, M. Aljnidi, K. Aljoumaa, A comparative dimensionality reduction study in telecom customer segmentation using deep learning and PCA, J. Big Data 7 (1) (2020) 9, http://dx.doi.org/10.1186/s40537-020-0286-0.
[34] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (1) (2006) 3–42, http://dx.doi.org/10.1007/s10994-006-6226-1.
[35] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Trans. Knowl. Discov. Data 6 (3) (2012) 1–39, http://dx.doi.org/10.1145/2133360.2133363.
[36] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, http://dx.doi.org/10.1038/nature14539.
[37] Y. Sun, W. Xu, J. Zhang, J. Xiong, G. Gui, Super-resolution imaging using convolutional neural networks, in: Q. Liang, X. Liu, Z. Na, W. Wang, J. Mu, B. Zhang (Eds.), Communications, Signal Processing, and Systems, Springer, Singapore, 2020, pp. 59–66.
[38] R. Yang, D. Qu, Y. Gao, Y. Qian, Y. Tang, NLSALog: An anomaly detection framework for log sequence in security management, IEEE Access 7 (2019) 181152–181164.
[39] S. He, J. Zhu, P. He, M.R. Lyu, Experience report: system log analysis for anomaly detection, in: IEEE International Symposium on Software Reliability Engineering, 2016, pp. 207–218, http://dx.doi.org/10.1109/ISSRE.2016.21.
[40] B. Wang, S. Ying, G. Cheng, R. Wang, Z. Yang, B. Dong, Log-based anomaly detection with the improved K-nearest neighbor, Int. J. Softw. Eng. Knowl. Eng. 30 (2) (2020) 239–262, http://dx.doi.org/10.1142/S0218194020500114.