Software Log Anomaly Detection Through One Class Clustering of Transformer Encoder Representation

doi: https://doi.org/10.1007/978-3-030-50726-8_85

Software Log Anomaly Detection through One Class Clustering of Transformer Encoder Representation

Rin Hirakawa[1], Keitaro Tominaga[2] and Yoshihisa Nakatoh[1]

[1] Kyushu Institute of Technology, 1-1 Sensuicho, Tobata Ward, Kitakyushu City, Fukuoka Prefecture, Japan
[2] Panasonic System Design Co., Ltd., 3-1-9, Shinyokohama, Kohoku-ku, Yokohama City, 222-0033, Japan
nakatoh@ecs.kyutech.ac.jp
tominaga.keitaro@jp.panasonic.com

Abstract. For smart devices such as smartphones and tablets, developing new software using open source software (OSS) is becoming mainstream. While OSS-based development can greatly increase project productivity, it is more difficult to identify the cause of software defects. In this paper, we propose a deep learning model that performs unsupervised learning based on the log data accumulated in the project and calculates the degree of abnormality per line for newly given logs. The proposed method is evaluated using open supercomputer system log data, Blue Gene / L, and the accuracy of the proposed method is compared with the conventional log anomaly detection method using LSTM AutoEncoder. As a result of the comparative experiment, it was found that the proposed method performed better than the conventional method in the two scores of AUROC and F1 Score at the cutoff point.

Keywords: Anomaly Detection, Software Log, Transformer, Unsupervised Learning.

1 Introduction

In software development for smart devices such as smartphones and tablets, it is necessary to implement abundant functions in a short period of time and release / update them to meet the needs of consumers. Since Open Source Software (OSS) can significantly reduce the time required for development and increase the reusability of programs, software development using these is becoming a global trend. On the other hand, the scale of software development using OSS is enormous, making it more difficult to identify the cause when a problem occurs. Engineers analyzing such complex bugs check a huge log of mixed output from various applications, however, the level of proficiency can make a huge difference in the speed of analysis.

Our ultimate goal is to create a GUI tool that visualizes statistical information on log data and possible causes of defects, so that anyone can analyze bugs smoothly. In this paper, we propose a method to calculate the degree of abnormality for each row of newly given log data based on the log data accumulated in the project. As a result, when performing defect analysis, it is possible to prioritize checking from the line that shows a high degree of abnormality, and it is expected that the analysis time will be shortened.

2 Proposed Method

Our debugging support tool is intended to provide users with two levels of anomaly scores for the current log data. In the first method, a single log data is input in a streaming format, and the time series abnormalities of each row are calculated. In the second method, we treat each line of the log message as a separate input and calculates the degree of anomalies when compared to the entire log data accumulated so far in the project. The first method has been discussed in our past paper [1], and this paper elaborates on the second method.

2.1 Related Works

When detecting abnormalities in log data, it is common to treat log messages as time-series data. Because it is difficult to handle log messages in the same way as natural language, messages can be divided into fixed phrase parts (keys) and embedded values (parameters) so that time series models such as LSTM can learn them [2]. This method may not be able to determine anomaly score efficiently if the training data does not cover all possible normal execution patterns. Our proposed method detects anomalies based on the meaning of sentences by converting log messages into features (distributed expressions) instead of using log key time-series patterns.

2.2 Model Structure

In the proposed model, each line of the log is input to a trained Transformer [3] with fixed weights to obtain the encoder representation (Fig.1). Before being input to Transformer, log messages are broken down into units called subwords by the morphological analyzer WordPiece [4] for neural language models. The subwords are transformed into contextual distributed representations by the Transformer encoder, and we use mean of them as features representing the entire sentence.

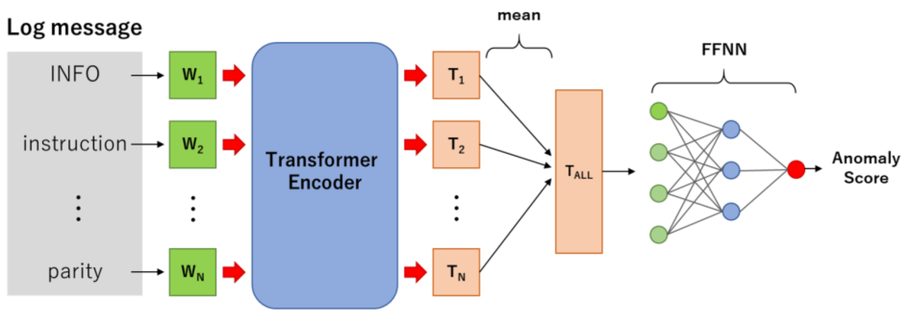

Fig. 1. Structure of proposed log anomaly calculation model.

The method for calculating the anomaly score in our model is inspired by One Class Neural Networks [5]. The distributed representation combined in the previous block is input to the three-layer Feed Forward Neural Networks (FFNN), which is finally converted to a scalar value that represents the distance from the origin. One-class clustering is performed based on the distance, and abnormal lines are detected by thresholding the score of each data as an abnormal score.

3 Experiment

In this chapter, we will use open log datasets to measure performance in anomaly detection and verify the effectiveness of the proposed method.

3.1 Datasets and Setup

BGL Datasets. BGL [6] is an open dataset of logs collected from the BlueGene / L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California. The log contains alert and non-alert messages identified by tags. This dataset is provided by the Loghub repository [7], a large collection of system log datasets for AI-powered log analysis.
Setup. Only the message part of the log about KERNEL of the data is extracted and formatted into a form suitable for the proposed method. Duplicate messages are removed and the entire data set is split for training, testing and validation in a 6: 2: 2 ratio. Details of the data are shown in Table 1.

3.2 Model Condition

The anomaly detection accuracy of the proposed method is compared with the conventional anomaly detection method using LSTM AutoEncoder. This section details the experimental conditions for each model.
Proposed Method. We use BERT-Base (L = 12, H = 768, published by Google [8]) as a Transformer encoder. Table 2 shows the configuration of each layer of FFNN. The loss function of FFNN is shown below (Equation 1).

$$r + \frac{1}{\nu} \cdot \frac{1}{N} \sum_{n=1}^{N} \max(0, \widehat{y_n}(w, V) \ - r) \tag{1}$$

Table 1. Breakdown of the number of normal / abnormal data.

	Training	Test	Validation
Normal	170549	56833	57361
Anomaly	1534	529	508

Table 2. Architecture of Feed Forward Neural Networks (Proposed Method).

Input (feature dimension)	Hidden layer	Output layer
768	72	1

Table 3. Hyperparameters of Proposed Method

Parameter	Value	Optimized by Optuna
ν	2.58e-03	✓
Learning rate	3.19e-05	✓
Epoch	20	✓
Batch size	32	
Max sequence length	128	

where, w and V are the weights between the hidden layer and the output layer and between the input layer and the hidden layer of the FFNN, respectively. Also, $\widehat{y_n}(w, V)$ indicates the final output of FFNN obtained by applying the sigmoid activation function. The values of w and V are updated repeatedly using a backpropagation algorithm on a mini-batch of training data (batch size N). The value of r is updated as νth quantile of $\widehat{y_n}(w, V)$ when all training data is input to FFNN whose weight is frozen at the end of the epoch. The optimal value of ν is determined using the open source hyperparameter auto-optimization framework Optuna [9]. Table 3 shows the details of the hyperparameters determined using Optuna.

LSTM AutoEncoder. To compare the accuracy, we use a conventional LSTM AutoEncoder in this experiment. The AutoEncoder implementation is based on the text-autoencoder repository [10] and its original paper [11]. WordPiece is used for the morphological analyzer as in the proposed method, and the model is trained using only the normal training data in Table 1. The actual anomaly detection uses exactly the same test data as the proposed method. We use the cross-entropy error obtained in entering the log message as a scalar value equivalent to the anomaly score of our proposed method. Table 4 shows the parameters used when training text-autoencoder models.

3.3 Evaluation

In the evaluation stage, the accuracy of anomaly detection on test data is verified using each model after training.

Table 4. Hyperparameters of LSTM AutoEncoder.

Parameter	Value
Model type	Denoising Auto-Encoder
Learning rate	5e-4
Epoch	2
Batch size	32
Max sequence length	128
Embedding dimension	512
Hidden state dimension	128
Number of layer	1

Each model is compared using two types of scores: AUROC and F1 scores. AUROC is an evaluation score that indicates how good the accuracy of the classification model is across the entire threshold. When determining the F1 score, we use a threshold at the cut-off point where the point at which the sensitivity – (1 – specificity) value is highest in the ROC curve.

4 Results & Discussion

Table 5 shows the AUROC value of each model calculated using the test data. Tables 6 and 7 show the F1 score and the other scores used when calculating it for the proposed method and LSTM AutoEncoder, respectively.

These results are the scores for the parameters that each model performs best, and show that the proposed method outperforms the LSTM AutoEncoder in both AUROC and F1 scores. Note that the performance of the proposed method is very sensitive to one of the hyperparameters, v, and the learning rate, and the same conditions may not be optimal when the size of the dataset changes. As a future task, it is necessary to investigate strategies for stable learning on the updated data set.

Table 5. AUROC of Each Models.

Proposed Method	LSTM AutoEncoder
0.823	0.786

Table 6. F1-score of Proposed Method.

		Precision	Recall	F1-score	Support
Class	Normal	1.00	0.82	0.90	56833
	Anomaly	0.04	0.82	0.08	529
Accuracy				0.82	
Macro Avg.		0.52	0.82	0.49	57362
Weighted Avg.		0.99	0.82	0.89	

Table 7. F1-score of LSTM AutoEncoder.

		Precision	Recall	F1-score	Support
Class	Normal	1.00	0.76	0.86	56833
	Anomaly	0.03	0.75	0.05	529
Accuracy				0.76	
Macro Avg.		0.51	0.75	0.46	57362
Weighted Avg.		0.99	0.76	0.85	

5 Conclusion

In this paper, we proposed a deep learning model that calculates an abnormal score for each line of log data by unsupervised learning. In the evaluation of anomaly detection accuracy using the open log data set BGL, it was found that the proposed method can detect anomalous rows more efficiently than the conventional LSTM AutoEncoder.

The degree of abnormality calculated by the proposed method indicates the abnormality in the accumulated log. This can be applied to construction for GUI that helps developers to easily find abnormal lines in the large amount of logs in a short time.

In the future, we will evaluate how much more efficient the debugging work will be when the developer uses a debugging support tool that reflects the anomaly score calculated by the proposed method.

References

1. Rin Hirakawa, Keitaro Tominaga, Yoshihisa Nakatoh: "Study on Real-time Log Anomaly Detection Method using HTM algorithm, Proceedings of the Institute of Electronics," Information and Communication Engineers, vol.2019 Society Conference, pp.74 (2019).
2. Du, Min & Li, Feifei & Zheng, Guineng & Srikumar, Vivek. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," pp.1285-1298. 10.1145/3133956.3134015. (2017).
3. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova: "BERT: Pre-training of deep bidirectional transformers for language understanding," In NAACL-HLT (2018).
4. Wu, Yonghui & Schuster, Mike & Chen, Zhifeng & V. Le, Quoc & Norouzi, Mohammad & Macherey, Wolfgang & Krikun, Maxim & Cao, Yuan & Gao, Qin & Macherey, Klaus & Klingner, Jeff & Shah, Apurva & Johnson, Melvin & Liu, Xiaobing & Kaiser, ukasz & Gouws, Stephan & Kato, Yoshikiyo & Kudo, Taku & Kazawa, Hideto & Dean, Jeffrey: "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," (2016).
5. Chalapathy, Raghavendra & Menon, Aditya & Chawla, Sanjay: "Anomaly Detection using One-Class Neural Networks," (2018).
6. Adam J. Oliner, Jon Stearley: "What Supercomputers Say: A Study of Five System Logs," in Proc. of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), (2007).
7. Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu: "Tools and Benchmarks for Automated Log Parsing," International Conference on Software Engineering (ICSE), (2019).
8. Bert (Github), https://github.com/google-research/bert, last accessed 2020/03/14.
9. Optuna (Github), https://github.com/optuna/optuna, last accessed 2020/03/15.
10. text-autoencoders (Github), https://github.com/shentianxiao/text-autoencoders, last accessed 2020/03/15.
11. Tianxiao Shen, Jonas Mueller, Regina Barzilay, Tommi Jaakkola: "Educating Text Autoencoders: Latent Representation Guidance via Denoising," arXiv preprint arXiv:1905.12777, (2019).