LogLAB: Attention-Based Labeling of Log Data Anomalies via Weak Supervision

Thorsten Wittkopp(✉), Philipp Wiesner(✉), Dominik Scheinert(✉), and Alexander Acker(✉)

Technische Universität Berlin, DOS, TU-Berlin, Berlin, Germany
{t.wittkopp,wiesner,dominik.scheinert,alexander.acker}@tu-berlin.de

Abstract. With increasing scale and complexity of cloud operations, automated detection of anomalies in monitoring data such as logs will be an essential part of managing future IT infrastructures. However, many methods based on artificial intelligence, such as supervised deep learning models, require large amounts of labeled training data to perform well. In practice, this data is rarely available because labeling log data is expensive, time-consuming, and requires a deep understanding of the underlying system. We present LogLAB, a novel modeling approach for automated labeling of log messages without requiring manual work by experts. Our method relies on estimated failure time windows provided by monitoring systems to produce precise labeled datasets in retrospect. It is based on the attention mechanism and uses a custom objective function for weak supervision deep learning techniques that accounts for imbalanced data. Our evaluation shows that LogLAB consistently outperforms nine benchmark approaches across three different datasets and maintains an F1-score of more than 0.98 even at large failure time windows.

Keywords: Anomaly labeling · AIOps · Log analysis

1 Introduction

As more and more companies outsource their IT services to the cloud, the number of servers and interconnected devices is continuously increasing. In the meantime, modern abstraction layers are driving the creation of large multilayered systems while adding technical complexity under the hood. This aggravates the operation and maintenance of systems and services and, therefore, poses new challenges for cloud operators. To maintain control over complexity, monitoring becomes an integral part of cloud infrastructure operations. However, in today's systems the amount of monitoring data is often growing to an extent that cannot be analyzed manually.

The area of artificial intelligence for IT operations (AIOps) is intended to support cloud operators to ensure operational efficiency as well as dependability and serviceability [5]. A core component of any AIOps system is the detection of anomalies in monitoring data such as metrics, logs, or traces. Log data are one

H. Hacid et al. (Eds.): ICSOC 2021, LNCS 13121, pp. 700–707, 2021.
https://doi.org/10.1007/978-3-030-91431-8_46

of the most important resources for troubleshooting because they record events during the execution of service applications. However, even though most types of log messages come with a severity level, these do not necessarily reflect the status of the overall system. Therefore, recent research utilizes deep learning models to analyze log data and perform anomaly detection [2,16,29]. One of the main obstacles in log anomaly detection is the lack of labeled log data [25]. Labeling data is costly and time-consuming, as experts need to analyze every single log message and investigate which messages reflect their corresponding errors. Since supervised models that train on large volumes of labeled data show significant performance in log anomaly detection [27,29], it is important to automate the labeling process to gain a strong accelerator for log anomaly detection [19].

To address this problem, we propose LogLAB, an attention-based model for binary labeling of anomalies in log data via weak supervision. It relies only on rough estimates of when an error has occurred - information that can often be derived from other monitoring systems [23]. Specifically, the contributions of this paper are:

- A problem description for how to label anomalies in log data using monitoring information and weak supervision including a method solving this.
- A custom objective function for weak supervision deep learning techniques that takes class-imbalanced data into account.
- An extensive evaluation of ten different approaches solving the defined problem, including LogLAB and its implementation[1].

The remainder of this paper is structured as follows. Section 2 surveys the related work. Section 3 provided a problem description and explains our approach LogLAB. Section 4 evaluates LogLAB in comparison to nine other approaches. Section 5 concludes the paper.

2 Related Work

We discuss works for text classification, anomaly detection and PU learning.

Text-Based Classification. Many established methods are discussed in [10]. The PCA algorithm [9] is for instance often employed for dimensionality reduction right before the actual classification procedure. Random forests [7] are another technique and a suitable tool due to their ensemble learning design. Logistic regression [8] belongs to the classic statistical methods [4]. Other publications utilize the Rocchio algorithm that is compared against kNN in [22]. In another work [21], the authors design a pipeline involving the Rocchio algorithm.

Log Anomaly Detection. The experience report for anomaly detection on system logs [6] discusses additional methods. Invariant Miners [13] retrieve structured logs using log parsing, further group log messages according to log parameter relationships, and mine invariants from the groups to perform actual anomaly

[1] https://github.com/dos-group/LogLAB.

detection on logs. Decision Trees [18] are another solution often employed in classification problem scenarios. SVMs are evaluated in [14] for document classification and anomaly detection. The authors of [20] propose a boosting-based system and thus ensemble learning method that shows good performance. Deep Learning methods are also more and more used in the realm of log anomaly detection. DeepLog [2] utilizes an LSTM and thus interprets a log as a sequences of templates to performs anomaly detection per log message. More recent works [26–28] also make use of deep learning.

PU Learning. A problem setting also discussed in other works. For instance, the authors in [12] utilize the EM algorithm together with naive Bayesian classification. A more conservative variant of this method is proposed in [3] where the set of reliable negative instances is iteratively pruned using a binary classifier, which ultimately leads to improved final prediction results due to the few but high quality negative instances. An ensemble learning method for PU learning is proposed in [15]. The authors motivate bagging SVM, i.e. the aggregation of multiple SVM classifiers in order to answer sources of instability often encountered in PU learning situations.

3 Automated Log Labeling

3.1 Problem Description

Log messages can describe failures that occur during runtime, such as the crash of a service. We refer to such log messages as 'abnormal'. Modern monitoring solutions raise alerts when a system runs into an abnormality or outages occur by observing metrics, hardware component failures, workload deployment failures and other failure scenarios [23]. Therefore, we assume that in an IT operation center failure time windows of services and systems are roughly known. We use this information in retrospect to identify and label abnormal log messages.

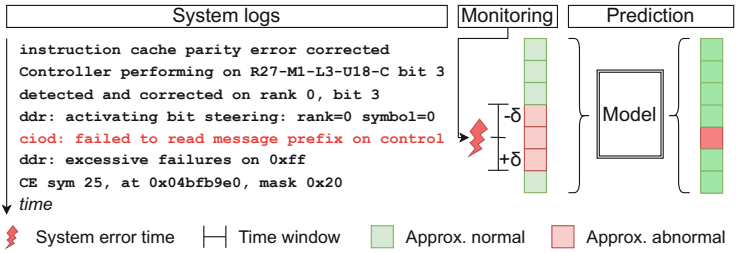

Fig. 1. We use rough estimates for failure times provided by monitoring systems in order to identify and label abnormal log messages via weak supervision. (Color figure online)

Figure 1 provides an example for the described problem. It displays the log of a system with one abnormal log event (colored in red). We utilize monitoring

information to estimate time windows of the length $2 * \delta$ in which we suspect abnormal log events. The model's task is to identify the abnormal log messages in the time window and classify all others as normal.

We describe the log labeling as a weak supervision learning problem with inaccurate labels as defined by Zhou et al. [30]. Thereby, label inaccuracy stems from the imprecision of the failure time windows. We assign inaccurate labels for all log events, depending on whether they are in the failure time windows or not. Further, we utilize PU learning [11,12] which is short for learning from positive and unlabeled data. Thereby, the underlying log data is divided into two classes, positive $\mathcal{P}$ and unlabeled $\mathcal{U}$, where $\mathcal{U}$ consists of all log messages that occur in the aforementioned failure time windows and $\mathcal{P}$ of the remaining log messages.

3.2 LogLAB

For the labeling of logs, we design a processing and modeling pipeline illustrated in Fig. 2. The individual steps are as follows:

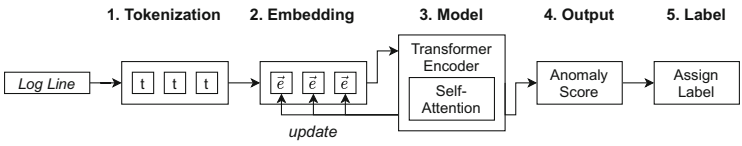

Fig. 2. High level log message labeling pipeline.

First, we convert the content c_i of each log message l_i into a sequence of tokens t_i by splitting on the symbols `.,:/` and whitespaces. Subsequently, we clean the resulting sequence of tokens by replacing certain tokens with placeholders. Thereby placeholder tokens for hexadecimal values `'[HEX]'` and any number greater or equal 10 `'[NUM]'` are introduced. Finally, we prefix the sequence of transformed tokens with a special token `'[CLS]'` which serves as a numerical summary of the whole log message. An exemplary log message: `time.c: Detected 3591.142 MHz` is thus transformed into a sequence of tokens: `['[CLS]', 'time', 'c', 'Detected', '[NUM]', '[NUM]', 'MHz']`.

Since these sequences can vary in length, we truncate them to a fixed size and pad smaller sequences with `'[PAD]'` tokens. For each token w_j of the token sequence t_i, an embedding $\vec{e}_i(j)$ is obtained. The truncated sequences of embeddings $\vec{e_i'}$ serves as the input for the model.

The model computes an output embedding, for each input sequence $\vec{e_i'}$, which summarizes the log message by utilizing the embeddings of all tokens. This output embedding is encoded in the embedding of the `'[CLS]'` token which is also modified during training. For this purpose, we utilize the transformer architecture [1] with additional self-attention [24]. During the training process, the model is supposed to learn the meanings of the log messages, thereby getting

an intuition of what is normal and abnormal. Finally, this model outputs a vector (embedding) for each input sequence $\vec{e_i}$. We denote the output of the model as $z_i = \Phi(\vec{e_i}; \Theta)$ and use it throughout the remaining steps. Thereby the anomaly score is calculated by the length of the output vector $\|z_i\|$. Anomaly scores close to 0 represent normal log messages, whereby large vectors indicate an abnormal log message. The computed anomaly score is used to assign a label $\widehat{y_i}$ to the log message l_i, i.e. either normal or abnormal.

3.3 Objective Function

To label the log data, the model must be trained in a way that it is capable to handle the problem of weak supervision with inaccurate labels. Thus, the objective function must assign log anomaly scores to log messages that occur in class $\mathcal{P}$ and $\mathcal{U}$. Log messages that occur only in $\mathcal{U}$ are likely abnormal and must therefore have higher anomaly scores. In addition, the loss function must be able to handle large amounts of incorrectly labeled log messages, since the class $\mathcal{U}$ can increase quickly for large δ. The objective function consist of two parts. The first part minimize the errors of samples from class $\mathcal{P}$, from which the calculated anomaly scores should be close to 0. The second part of the objective must minimize the errors of samples from class $\mathcal{U}$, by pushing them away from 0. The structure of the objective function is defined as $\frac{1}{m}\sum_{i=1}^{m}((1-\tilde{y}_i)*a(z_i)+(\tilde{y}_i)*b(z_i)$, where $\tilde{y}_i$ is the inaccurate label, z_i the output vector and m the batch size. The function '$a()$' becomes 0 if the sample is from class $\mathcal{U}$, while the second function '$b()$' becomes 0 if the sample is from class $\mathcal{P}$. For a we choose $a(z_i) = \|z_i\|^2$ and for b we choose $b(z_i) = \frac{q^2}{\|z_i\|}$ to minimize the error. Thereby a calculates the squared error of the length of the output for samples from class $\mathcal{P}$. In contrast, we increase the error for all small anomaly scores when the log message is of class $\mathcal{U}$. Thereby q is a numerator between 0 and 1 that represents the relation of the number of samples in $\mathcal{P}$ and $\mathcal{U}$. To ensure that q is representing the relation of $\mathcal{P}$ and $\mathcal{U}$ and remains in the boundaries of 0 to 1, we model q as a limited function $f(x) = \frac{x}{x+1}$, with $\lim_{x\to\infty} f(x) = 1$, that is provided with the relation of $\mathcal{P}$ and $\mathcal{U}$: $q = f(\frac{|\mathcal{P}|}{|\mathcal{U}|}) = \frac{\frac{|\mathcal{P}|}{|\mathcal{U}|}}{(\frac{|\mathcal{P}|}{|\mathcal{U}|}+1)} = \frac{|\mathcal{P}|}{|\mathcal{P}|+|\mathcal{U}|}$. Thus the total loss function can be expressed as: $\frac{1}{m}\sum_{i=1}^{n}\left((1-\tilde{y}_i)*\|z_i\|^2 + (\tilde{y}_i)*\frac{(\frac{|\mathcal{P}|}{|\mathcal{P}|+|\mathcal{U}|})^2}{\|z_i\|}\right)$.

4 Evaluation

To obtain a significant and wide benchmark, we compare LogLAB to several state of the art text-classification and anomaly detection approaches presented in a recent text-classification survey [10] as well as in an established survey for anomaly detection in system logs [6]. Namely, we choose PCA, Invariant Miners, Deeplog, Decision Trees, Random Forests, SVMs, Logistic Regression, the Rocchio algorithm, and boosting approaches as benchmark methods. Thereby we measure the deviation from the ground truth y_i and the calculated labels $\widehat{y_i}$.

4.1 Experimental Setup

We evaluate all methods on three labeled log datasets recorded at different large-scale computer systems[17]. The *BGL* dataset contains 4 747 963 log messages of which 7.3 % are abnormal and records a period of 214 days, with on average 0.25 log messages per second. We selected the first 5 M log messages from the *Thunderbird* dataset of which 4.5 % are abnormal. They account for a period of 9 days, with on average 6.4 log messages per second. Again, we selected the first 5 M log messages from the *Spirit* dataset of which 15.3 % are abnormal. They cover a period of 48 days, with on average 1.2 log messages per second.

We create our evaluation datasets with inaccurate labels by including all abnormal log events as well as their surrounding events within a time window $2*\delta$ in $\mathcal{U}$; all remaining log events are in $\mathcal{P}$. Thereby we investigate the performance at three different time windows: $\pm 1000\,ms$ (2s), $\pm 5000\,ms$ (10s) and $\pm 15000\,ms$ (30s). The amount of samples in $\mathcal{U}$ is changing for BGL: $0.39M$, $0.44M$ and $0.47M$, Thunderbird: $1.42M$, $2.36M$ and $2.90M$ and Spirit: $1.00M$, $2.33M$ and $3.26M$ regarding the respective time window δ.

Each sequence of tokens t_i is truncated to have a length of 20 for *Thunderbird*, 16 for *Spirit*, and 12 for *BGL*. The dimensionality d of our embeddings is set to 128. For the training of our LogLAB model, we use a hidden dimensionality of 256, a batch size of 1024, a total of 8 epochs, and a dropout rate of 10%. We use the Adam optimizer with a learning rate of 10^{-4} and a weight decay of $5 \cdot 10^{-5}$.

4.2 Results

Table 1. Evaluation results: F1-scores above 0.99 and 0.98 are highlighted in blue and cyan, respectively.

Dataset	Metric	Learning $\mathcal{P}$			Learning $\mathcal{P}$ and $\mathcal{U}$						
		PCA	Invariant Miners	Deeplog	Decision Tree	Random Forest	SVM	Logistic Regr.	Boost	Rocchio	**LogLAB**
$\delta = \pm 1000ms$											
BGL	F1-Score	0.5963	0.5102	0.7759	0.9974	0.9830	0.9840	0.9976	0.9908	0.7096	0.9977
TBird	F1-Score	0.3048	0.1824	0.0880	0.3242	0.3144	0.3235	0.3242	0.3361	0.3440	0.9995
Spirit	F1-Score	0.8043	0.5807	0.9926	0.9967	0.9604	0.9857	0.9962	0.9968	0.9971	0.9997
$\delta = \pm 5000ms$											
BGL	F1-Score	0.5930	0.5112	0.7755	0.9874	0.9646	0.9680	0.9875	0.9795	0.8054	0.9949
TBird	F1-Score	0.3053	0.1936	0.0651	0.2669	0.2415	0.2439	0.2678	0.2869	0.3146	0.9995
Spirit	F1-Score	0.7691	0.5740	0.9929	0.6513	0.5453	0.5584	0.6560	0.5830	0.9946	0.9980
$\delta = \pm 15000ms$											
BGL	F1-Score	0.5879	0.5130	0.7760	0.9753	0.9483	0.9523	0.9767	0.9762	0.7898	0.9902
TBird	F1-Score	0.3025	0.1933	0.1113	0.1341	0.1248	0.1348	0.1350	0.1476	0.2241	0.9995
Spirit	F1-Score	0.4958	0.4254	0.8236	0.4909	0.4735	0.4836	0.4917	0.4887	0.5192	0.9825

To compare LogLAB to our baselines, we assess the prediction performance $\tilde{y}_i \sim y_i$ in terms of F1-score metrics. The F1-scores are presented in Table 1. As expected, with increasing δ and thus growing size of $\mathcal{U}$, the performance across all approaches tends to decrease. For $\delta = \pm 1000\,ms$ this was apparently easy to achieve for most of the methods. An exception is the Thunderbird dataset, which is characterized by a large $\mathcal{U}$ class: No baseline manages to achieve an F1-score higher than 0.35, except LogLAB. For $\delta = \pm 5000\,ms$ we notice that the performance degradation previously observed for most approaches on the Thunderbird dataset now also start to manifest on the Spirit dataset. The biggest gap in performance becomes evident at the largest observed time window of $\delta = \pm 15000\,ms$. For the dataset BGL, we notice a considerable drop in F1-scores of other approaches to 0.97, while LogLAB maintains its high performance.

5 Conclusion

This paper presents LogLAB, a novel model for labeling large amounts of log data, such that the usually required need of time-consuming manual labeling through experts is automated. It relies only on rough estimates of failure time windows provided by monitoring systems to generate labeled datasets in retrospect. LogLAB is based on the attention mechanism and uses a custom objective function for weak supervision deep learning techniques that accounts for imbalanced data and deals with inaccurate labels. We evaluated LogLAB on three different datasets in comparison to nine benchmark approaches. LogLAB outperforms other approaches across all experiments and shows a performance of more than 0.98 F1-score, even for large amount of inaccurate labels. As further work we consider to enhance the labeling process by iteratively moving log messages from $\mathcal{U}$ to $\mathcal{P}$ during training, which have significantly lower anomaly scores, calculated by the model. Likewise, this method can be extended by adding other sources for estimating the time windows and therefore improve the training basis.

References

1. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) NAACL-HLT. Association for Computational Linguistics (2019)
2. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: SIGSAC (2017)
3. Fusilier, D.H., Montes-y Gómez, M., Rosso, P., Cabrera, R.G.: Detecting positive and negative deceptive opinions using PU-learning. Inf. Process. Manag. **51**, 433–443 (2015)
4. Genkin, A., Lewis, D.D., Madigan, D.: Large-scale bayesian logistic regression for text categorization. Technometrics **49**(3), 291–304 (2007)
5. Gulenko, A., Acker, A., Kao, O., Liu, F.: Ai-governance and levels of automation for aiops-supported system administration. In: ICCCN. IEEE (2020)
6. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: ISSRE. IEEE (2016)

7. Ho, T.K.: Random decision forests. In: ICDAR. IEEE (1995)
8. Hosmer Jr., D.W., Lemeshow, S., Sturdivant, R.X.: Applied Logistic Regression, vol. 398. Wiley, Hoboken (2013)
9. Jolliffe, I.: Principal component analysis. Encyclopedia of statistics in behavioral science (2005)
10. Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., Brown, D.: Text classification algorithms: a survey. Information **10**(4), 150 (2019)
11. Liu, B., Dai, Y., Li, X., Lee, W.S., Yu, P.S.: Building text classifiers using positive and unlabeled examples. In: ICDM. IEEE (2003)
12. Liu, B., Lee, W.S., Yu, P.S., Li, X.: Partially supervised classification of text documents. In: ICML, Sydney, NSW (2002)
13. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX Annual Technical Conference (2010)
14. Manevitz, L.M., Yousef, M.: One-class SVMs for document classification. J. Mach. Learn. Res. **2**(Dec), 139–154 (2001)
15. Mordelet, F., Vert, J.P.: A bagging SVM to learn from positive and unlabeled examples. Pattern Recognit. Lett. **37**, 201–209 (2014)
16. Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O.: Self-attentive classification-based anomaly detection in unstructured logs. In: ICDM (2020)
17. Oliner, A., Stearley, J.: What supercomputers say: a study of five system logs. In: DSN (2007)
18. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
19. Ratner, A.J., De Sa, C.M., Wu, S., Selsam, D., Ré, C.: Data programming: creating large training sets, quickly. NIPS **29**, 3567–3575 (2016)
20. Schapire, R.E., Singer, Y.: Boostexter: a boosting-based system for text categorization. Mach. Learn. **39**, 135–168 (2000)
21. Selvi, S.T., Karthikeyan, P., Vincent, A., Abinaya, V., Neeraja, G., Deepika, R.: Text categorization using rocchio algorithm and random forest algorithm. In: ICoAC. IEEE (2017)
22. Sowmya, B., Srinivasa, K., et al.: Large scale multi-label text classification of a hierarchical dataset using rocchio algorithm. In: CSITSS. IEEE (2016)
23. Sukhwani, H., Matias, R., Trivedi, K.S., Rindos, A.: Monitoring and mitigating software aging on IBM cloud controller system. In: ISSREW. IEEE (2017)
24. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., et al. (eds.) NeurIPS (2017)
25. Wittkopp, T., Acker, A., et al.: Decentralized federated learning preserves model and data privacy. In: Hacid, H. (ed.) ICSOC 2020. LNCS, vol. 12632, pp. 176–187. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76352-7_20
26. Wittkopp, T., et al.: A2log: attentive augmented log anomaly detection. In: HICSS (2022)
27. Yang, L., et al.: Semi-supervised log-based anomaly detection via probabilistic label estimation. In: ICSE. IEEE (2021)
28. Yang, R., Qu, D., Gao, Y., Qian, Y., Tang, Y.: NLSALog: an anomaly detection framework for log sequence in security management. IEEE Access **7**, 181152–181164 (2019)
29. Zhang, X., et al.: Robust log-based anomaly detection on unstable log data. In: ESEC/FSE (2019)
30. Zhou, Z.H.: A brief introduction to weakly supervised learning. Natl. Sci. Rev. **5**(1), 44–53 (2018)