Anomaly Detection From Log Files Using Unsupervised Deep Learning

Sathya Bursic[0000−0001−8327−5007], Vittorio Cuculo[0000−0002−8479−9950], Alessandro D'Amelio[0000−0002−8210−4457], and

Dipartimento di Informatica
University of Milan, Milano, Italy
{sathya.bursic,vittorio.cuculo,alessandro.damelio}@unimi.it

Abstract. Computer systems have grown in complexity to the point where manual inspection of system behaviour for purposes of malfunction detection have become unfeasible. As these systems output voluminous logs of their activity, machine led analysis of them is a growing need with already several existing solutions. These largely depend on having hand-crafted features, require raw log preprocessing and feature extraction or use supervised learning necessitating having a labeled log dataset not always easily procurable. We propose a two part deep autoencoder model with LSTM units that requires no hand-crafted features, no preprocessing of data as it works on raw text and outputs an anomaly score for each log entry. This anomaly score represents the rarity of a log event both in terms of its content and temporal context. The model was trained and tested on a dataset of HDFS logs containing 2 million raw lines of which half was used for training and half for testing. While this model cannot match the performance of a supervised binary classifier, it could be a useful tool as a coarse filter for manual inspection of log files where a labeled dataset is unavailable.

Keywords: Deep Learning · Anomaly Detection · Log File.

1 Introduction

Today's computer systems in commercial environments are frequently complex and distributed and work on large data throughput. For any part of such a system, be it networking, program execution, machine performance, etc., there is the occurrence of process anomalies and most of these systems generate and keep logs which are intended to be analysed for detecting malfunctions. The commercial systems usually are intended to operate incessantly and reliably with failure to do so having potential to incur costs for the organization. The process of analyzing them has been historically manually done. However, given the scale of these systems the problems that present are that system behaviour may be too complex for a single human to understand and that the systems may generate logs on the order of gigabytes per hour making it infeasible for human comprehension and human anomaly detection [5].

This gave rise to demand for automated log anomaly detection. The problem has been addressed in literature by first performing feature extraction and then applying a linear machine learning model such as PCA, logistic regression or a linear SVM. In this project we propose a two-part model of deep autoencoders that require minimal raw log file preprocessing and detect both anomalous log content and anomalous temporal evolution of logs. The paper is organised as follows: in the next section we review recent approaches to the problem, in Section 3 we present in detail our approach, following which the experiment and results are presented and discussed.

2 Related Work

Traditionally log anomaly detection has had three basic steps: log parsing, which turns unstructured text into structured data; feature extraction, where the text is transformed into a numerical feature vector; and anomaly detection, where a machine learning algorithm is applied to classify log events as anomalous or normal execution [5].

The approaches to the log parsing step can loosely be divided into *clustering based* and *heuristic based*. The clustering based approach parts from calculating distances between logs first and then clustering them into groups using the calculated distance. Heuristic based instead count word occurrences in logs for each position and then frequent words in the positions are selected as event candidates. As an example of clustering based parsing, in [3] the authors separate the changing and constant parts of log messages by first using empirical rules (eg. a regular expression to identify IP addresses), and then by token clustering. A different approach for log message clustering as described in [7] is called the IPLoM Algorithm and pertains to the class of heuristic approaches which are said to perform better than clustering based.

Among the machine learning algorithms that have been applied to the problem are logistic regression, decision tree and SVM of supervised models, and clustering, PCA and invariant mining of unsupervised models. Results indicate that supervised methods achieve performance levels not reachable with unsupervised methods. Also, the performance of the supervised methods are mostly similar with SVM giving greatest and decision tree giving lowest performance, while among unsupervised methods invariant mining gives notably superior performance to other unsupervised methods [5].

Recently, deep learning has been also applied to the problem. Du et al. [2] proposed an LSTM deep neural network to model system logs as a natural language sequence with the purpose of allowing the model to automatically learn log patterns from normal execution, and detect anomalies when log patterns deviate from the model trained on log data under normal execution. For parsing the logs from unstructured text into log keys they used a method similar to [3], ie. a clustering based approach. The results reported outperform all other anomaly detection methods not based on deep learning models. In parallel with supervised deep learning, unsupervised deep learning methods have started being used for

anomaly detection. Thus, Tuor et al. [9] developed an online unsupervised deep learning approach to detect anomalous network activity from system logs in real time outperforming PCA, SVM and isolation forest models. They use a deep neural network (DNN) composed of LSTM units trained to predict the following event in a sequence of events, similar to [2]. Finally, in [1] and [11] the authors used deep autoencoders to detect anomalies by looking at input reconstruction errors. In the former [1], they applied it to assessing the behaviour of high performance computing systems while in the latter [11], they used it for finding anomalies in multivariate time series data such as those encountered in industrial production systems.

3 Proposed Method

3.1 Model Architecture

The goal of this work is to develop a model requiring minimal raw log preprocessing that is capable of detecting both anomalous message content and anomalous temporal evolution of log messages. We follow the work of [8] in developing a deep autoencoder for text. They used a multilayer LSTM to map input sequences to a vector of fixed dimensionality, after which another multilayered LSTM to decode the target sequence from the vector. They achieve then state of the art on English to French translation tasks and furthermore introduce using bidirectional models to counter the problems of performance on long sequences.

A property of this model is that it learns to map an input sequence of variable length to a fixed length vector in an embedding space. In this work, the approach of [8] provides an opportunity to not have to preprocess log files in any elaborate manner, thus making the model suitable to any type of log file with the only cost being that of training the model. The second part of the model is another LSTM autoencoder that has the purpose of detecting anomalies. Following the work of [1] and [11], we suppose that after training an autoencoder is able to reconstruct better those inputs of which it has been exposed to more during training and vice versa. Finally, using a numerical measure of the distance between the inputs to the second part of the model and their reconstruction at its output an anomaly score is obtained.

The full definition of the model is the following, also represented in fig. 1: first the autoencoder embedding log messages is trained on log text (without timestamp) to learn a fixed dimensional embedding of the log messages. After training the decoder is discarded and the autoencoder for detecting anomalies is trained taking as input the embeddings of the message and the numerical timestamp of the message. Finally, a distance measure between the inputs and outputs is calculated and an input is considered anomalous if its distance measure lies above an appropriately chosen threshold. While the anomaly detection autoencoder is the same approach that [1] and [11] had, and the message embedding autoencoder comes from the work of [8], the innovation of this work comes from the model not imposing requirements on the log message structure

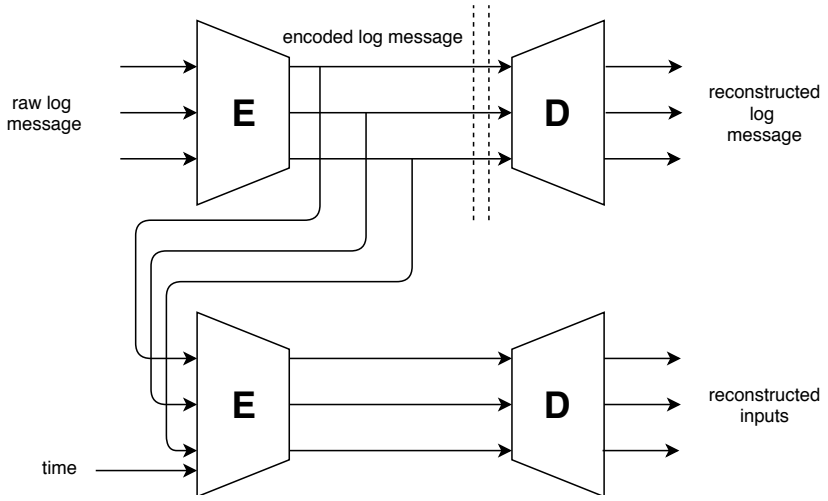

Fig. 1: Schema of the model architecture

and not requiring preprocessing of the log messages which ensures generality and applicability to any type of logs.

3.2 Implementation and Model Training

The dataset used to train and evaluate the proposed model came out of a work of Xu et al. [10] where they mined console logs to detect system runtime problems by parsing them. Combining source code analysis based on C *printf* statements with information retrieval and after extracting features from the parsed logs, they detect operational problems using machine learning. The dataset contains 11 million lines of Hadoop File System logs and is furthermore labeled which will provide a measure of performance for the model. An example of a log entry from the dataset would be:

> 081109 204453 34 INFO dfs.FSNamesystem: BLOCK*
> NameSystem.addStoredBlock: blockMap updated: 10.250.11.85:50010 is
> added to blk_2377150260128098806 size 67108864

The date and time format is "$DDMMYYhhmmss$" with D representing day, M month, Y year, h hour, m minute and s second, respectively.

After separating the date and time, all non-alphanumeric characters are replaced by whitespace and whitespace is inserted between single digit numbers. Furthermore, all multi digit numbers are considered as separate tokens, eg. the number "67108864" from the previous example becomes "6 7 1 0 8 8 6 4". This is done because it is not practical that all numbers, which are frequently variable for a type of log message, be a part of the vocabulary for the text autoencoder. As per how the network encoding text is constructed, all words in the dataset

during preprocessing are put into a vocabulary and assigned a numerical value. Making multi digit numbers that are a log variable part of the vocabulary would constrain the model only to those numbers seen during training. Separating the numbers by whitespace makes the model consider them as separate tokens and allows the model to easily encode numerical values in log messages.

Only a subset of the dataset has been considered, totaling 2 million lines which was further divided into 50% of training data and 50% of test data. The peculiarity of this problem in general and the dataset at hand is the strongly uneven class distribution where there approximately 3% of anoumalous data in the dataset.

The text encoder network is trained with the hyperparameters shown in table 1. These values have been found experimentally. The vectors output by

Table 1: Log Message Embedding Network Parameters

Parameter	**Value**
word embedding size	200
LSTM units	100
initial learning rate	0.01
batch size	64
dropout keep probability	0.75
training epochs	4

Table 2: Anomaly Detection Network Parameters

Parameter	**Value**
LSTM units	64
batch size	64
dropout keep probability	0.8
training epochs	4

the text encoder are then fed as input into the anomaly detection network for training that takes also the cosine transform of the percentage of the seconds passed for that day. The expression transforming the time is thus

$$f(t) = \cos\left(2\pi \frac{t}{86400}\right) \tag{1}$$

with t being the seconds from midnight at which the log event occurred. The aim here is of normalizing the data around zero as deep neural network training has been shown to behave better when inputs are normalized around zero [6]. This transformation of time removes any possibility of detecting inter-day seasonal patterns in the data and hence the model is geared toward detecting short-term anomalies. There are other ways of transforming inputs of cyclical nature to feed into a neural network. Most notably, the corresponding cosine transform could have also been included which would have provided more precise information of the time of day. Also, the same transform could have been included for the day of month and month. However, given that the frequency of log message generation is high, all other temporal information is considered less relevant and discarded for simplicity. The cost function used for training is the MSE function and the experimentally found training parameters for the anomaly detection network can

be found in table 2. The reconstruction error on the anomaly detection network is calculated using an L_1 distance of the inputs and their reconstruction.

4 Experimental Results

On fig. 2 anomaly values for 1 million lines of the test dataset are plotted. Several outliers where the anomaly value is strongly above the mean are particularly interesting. However, most of the values lie within a particular band which could potentially necessitate careful choice of threshold. If the threshold were chosen too low there could be a lot of false positives, and if chosen too high there could be a lot of false negatives.

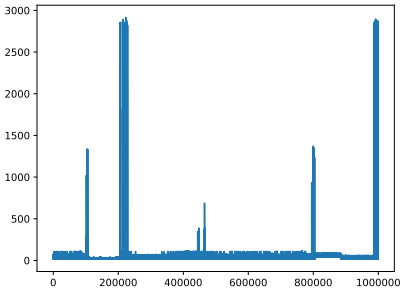

Fig. 2: Test anomaly scores

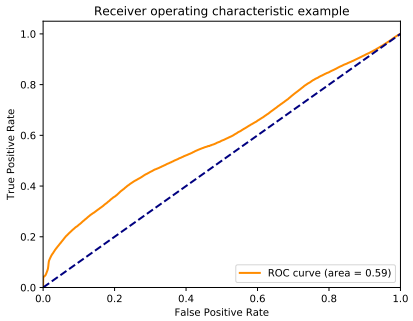

Fig. 3: ROC curve for the test dataset

As the dataset is labeled the problem can be considered a binary classification problem with the classes being *anomalous* and *not anomalous* for any log message. For this reason we can consider techniques usually applied for evaluating binary classifier performance.

The ROC curve for the test dataset can be seen in fig. 3 with AUC equal to 0.59. The value isn't high indicating that the classifier doesn't perform well. This can be supported by visual inspection of the curve showing poor class separation, especially at higher values of FPR and TPR. Regardless, the importance of precision and recall relative one to another can be considered inherent to the problem at hand. This has been considered a critique of the F-measure as a commonly used binary classifier quality indicator [4].

A plot of precision, recall and F-score for the test dataset can be seen in fig. 4. The plot shows the maximum of F-score at a threshold level of 62, after which precision rises rapidly and recall falls rapidly.

The values of true positives, true negatives, false positives and false negatives at eight different threshold values are presented in table 3. A consistently high level of false negatives can be noticed meaning that the largest part of anomalies will be left out by the model. This is also supported by the recall curve parting

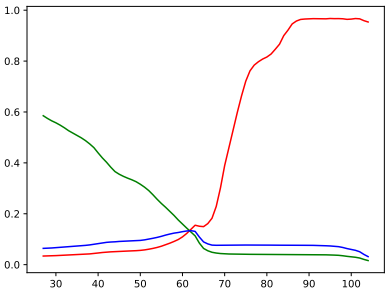

Fig. 4: A plot of precision (red), recall (green) and F-measure (blue) on the test dataset for threshold values between 25 and 106

Table 3: True positives, true negatives, false positives and false negatives for eight different threshold values.

Thresh	**TP**	**TN**	**FP**	**FN**
27	17279	473614	493378	12244
37	14395	630785	336207	15128
47	10035	789325	177667	19488
57	6185	899853	67139	23338
67	1435	960561	6431	28088
77	1195	966663	329	28328
87	1169	966940	52	28354
97	1083	966955	37	28440

from a value of around 0.6 for a low threshold value with the precision being low. This could possibly indicate that certain erroneous events are indeed fairly common. However, at higher threshold values the false positives fall faster than the true positives which is also supported by the rapidly rising precision in fig. 4. This is in line with what can be seen on fig. 2 as there are outliers which have an anomaly score vastly higher than the average. As supported by the values in table 3 most of these outliers seem to be anomalies. Given the above, a potential use for this kind of model is as a coarse filter for human inspection of log files of high enough frequency such that human inspection of the whole stream is infeasible. By choosing higher threshold values (for example at 77), while most anomalies won't be detected, the anomalies with the highest anomaly values will, and the precision will be high. This means that if all values which the model will classify as anomalous are given to a human for inspection most of them indeed will be. One might consider that optimizing for recall rather than precision might be more adapt for application as a filter for manual inspection, but the fundamental ceiling on recall of the approach is that the definition of anomalous here is that which occurs rarely. While the choice of optimal threshold isn't directly addressed here, in potential applications several approaches might be taken depending on the problem at hand. Firstly, a validation dataset could be used with manual choice of threshold. Secondly, the threshold can be dynamically changed and optimized by the user based on experience and circumstances. Thirdly, any number of statistical measures can be adopted as a threshold, such as the mean plus n standard deviations.

5 Conclusions

In this work we proposed a two-part unsupervised deep learning model for detecting anomalies in system log files. The model is composed of two autoencoders

with LSTM units of which one is applied to text as per the work of [8], and the other that takes the text embeddings and a temporal value as input and serves to detect anomalies. Using a publicly available labeled dataset of HDFS logs, experimental results show that at most thresholds recall is low but at higher thresholds precision is high showing that most errors aren't detected by the model at higher thresholds, but those detected are mostly errors. The properties of the model being it doesn't require any log preprocessing or feature extraction and works on generic log data, a potential use could be as a filter for human inspection for anomaly detection in systems generating logs with high frequency. The results presented in this note represent a preliminary proof of concept and in future work we plan to provide a more exhaustive experimental part.

References

1. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Anomaly detection using autoencoders in high performance computing systems. arXiv preprint arXiv:1811.05269 (2018)
2. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1285–1298. ACM (2017)
3. Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: 2009 ninth IEEE international conference on data mining. pp. 149–158. IEEE (2009)
4. Hand, D., Christen, P.: A note on using the f-measure for evaluating record linkage algorithms. Statistics and Computing **28**(3), 539–547 (2018)
5. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: System log analysis for anomaly detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). pp. 207–218. IEEE (2016)
6. LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural networks: Tricks of the trade, pp. 9–48. Springer (2012)
7. Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1255–1264. ACM (2009)
8. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
9. Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., Robinson, S.: Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
10. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Largescale system problem detection by mining console logs. Proceedings of SOSP09 (2009)
11. Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H., Chawla, N.V.: A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. arXiv preprint arXiv:1811.08055 (2018)