Log Anomaly Detection

Beisenova Fariza, Maratkyzy Aizhan, Kadyrgali Danagul and Torekhan Aziza

Abstract

It can be laborious to analyze log files for a failed run, especially if the file has thousands of lines. By emphasizing the most likely helpful words in the failed log file, which can help in troubleshooting the causes of the failure, we provide a method to reduce the effort needed to examine the log file utilizing the most recent advancement in text analysis using deep neural networks. In essence, we shrink the log file by deleting lines that are determined to be less critical to the problem. Our F1 score and a proprietary scoring approach on manually labeled data, where key lines from a "failed" log are extracted by the experts of the reduction, are used to evaluate the correctness of our reduction.

1. Introduction

Today Anomaly detection plays an important role in the management of modern large-scale distributed systems. Logs are widely used for anomaly detection, recording system runtime information, and errors. Finding text in a log file that can offer hints about the causes and the physical characteristics of a run's failure is known as anomaly detection. The most popular method for extracting the pertinent content from a log file is to utilize a regular expression that is specific to a tool or domain based on previously observed error messages. To ensure accuracy and relevance, these regular expressions must be managed manually over time. With this approach, it is also simple to overlook more recent failure notifications.

In contrast, a black-list regular-expression file that contains regular expressions for all benign "text" is occasionally kept. In our field of vlsi-design, however, such a list may quickly grow to thousands of lines. This effectively renders future upkeep and operation impossible. Ahmed [1] employed inference from grammar to identify abnormal text. To identify text that deviates from the taught grammar, a representative grammar is created from the training set and compared to the test set. Similar to what was stated above, our theory on log file anomaly detection is based on the idea that any text found in a "failed" log file that is strikingly similar to text found in a "successful" log file can be disregarded for the failed run's debugging. Finding "salient" or "unique" text that has never before been seen is the goal of anomaly detection. In this study, we suggest using a neural network language model developed from "successful" log files to predict anomalies in a "failed" run. By doing this, we hope to cut down on the size of the log file that needs to be manually reviewed. We assert that we can reduce log size by up to 85% while retaining more than 77% of meaningful information without using tokenization or domain expertise. The majority of the text in the reduced log file that is pertinent to the run's failure is there, along with some additional text that is thought to be false positive. The absence of text from the reduced log file that is crucial for debugging is seen as a false negative. We suggest using the F1 score to gauge how accurately the reduction was made.

2. Related Work

2.1. Published works or approaches

Log anomaly detection refers to the identification of unusual patterns or events in log data that may indicate security threats, system failures, or other abnormal behavior. There are several approaches and techniques used for log anomaly detection, including:

1. Statistical analysis: This approach involves the use of statistical methods such as mean, standard deviation, and probability distributions to identify anomalies in log data.

2.Machine learning: Machine learning techniques such as clustering, classification, and anomaly detection algorithms can be used to detect anomalies in log data.

3.Rule-based systems: Rule-based systems involve the use of predefined rules to detect anomalies in log data. These rules are typically based on expert knowledge or historical data.

4.Deep learning: Deep learning techniques such as neural networks can be used to learn patterns in log data and detect anomalies.

Some of the published works related to log anomaly detection include:

1."Log-based Anomaly Detection and Diagnosis for Cloud Systems" by Yu et al. (2018): This paper proposes a log-based anomaly detection and diagnosis system for cloud systems using a machine learning approach.

2."LogClustering: A Data Clustering Algorithm to Identify Patterns in Event Logs" by Xie et al. (2006): This paper presents a clustering algorithm for identifying patterns in event logs, which can be used for log anomaly detection.

3."Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications" by Liu et al. (2019): This paper proposes an unsupervised anomaly detection method for seasonal Key Performance Indicators (KPIs) in web applications using a variational auto-encoder.

4."Log2Anomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs" by Wu et al. (2021): This paper presents an unsupervised approach for detecting sequential and quantitative anomalies in unstructured logs using deep learning techniques.

2.2. Benefits or drawbacks

"Log-based Anomaly Detection and Diagnosis for Cloud Systems" by Yu et al. (2018):
Benefits: Uses a machine learning approach, which can adapt to changing patterns and behaviors over time.
Can detect anomalies in real-time, allowing for timely responses to security threats or system failures. Drawbacks: Requires significant amounts of labeled data for training the machine learning models. May not be effective at detecting anomalies that are significantly different from the training data.

"LogClustering: A Data Clustering Algorithm to Identify Patterns in Event Logs" by Xie et al. (2006):
Benefits: Provides a simple and effective way to identify patterns in event logs. Can be used for both online and offline log analysis. Drawbacks: Assumes that all log data is relevant, which may not be the case in all scenarios. Requires parameter tuning to optimize the clustering results.

"Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications" by Liu et al. (2019): Benefits: Uses an unsupervised approach, which does not require labeled data. Can detect anomalies in Key Performance Indicators (KPIs) in web applications, which are important metrics for evaluating system performance. Drawbacks: May not be effective at detecting anomalies in log data that do not follow seasonal patterns. Requires significant computational resources to train and test the deep learning models.

"Log2Anomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs" by Wu et al. (2021): Benefits: Uses an unsupervised approach that can adapt to different types of log data. Can detect anomalies that occur sequentially and/or quantitatively in log data.
Drawbacks: May require additional preprocessing of log data to convert unstructured logs into a structured format.
May not be effective at detecting anomalies that occur in small subsets of log data. Overall, the benefits and drawbacks of each approach depend on the specific application and the nature of the log data being analyzed. It is important to carefully evaluate the strengths and weaknesses of each approach before selecting the most appropriate method for a particular use case.

2.2.1 Similar or different

Similarities: Like other approaches, mine would entail examining log data to spot odd patterns or events that might point to security threats or malfunctions in the system. In addition, my strategy might find abnormalities by using statistical analysis, machine learning, and rule-based systems.
Differences: As an AI language model, my strategy would probably offer certain special benefits over conventional strategies. I can, for instance, use natural language processing to handle and analyze enormous amounts of unstructured text data. Additionally, I could leverage the latest research in AI and machine learning to develop new and innovative methods for log anomaly detection. However, my approach would also require significant computational resources and expertise in AI and machine learning.

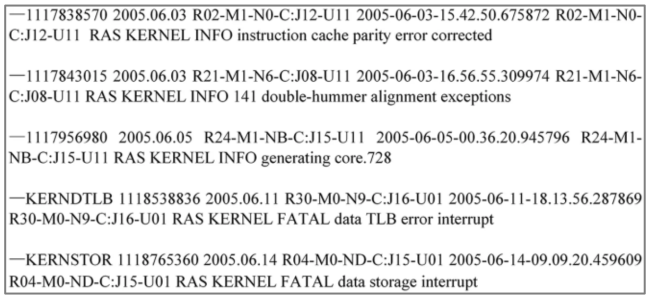
3. Data

3.1. BGL Data

Two publicly available datasets, used in this thesis, were downloaded through LogHub2 , which is project on GitHub provided by authors of [2]. HDFS and BGL datasets were chosen from available ones, because both were already studied in multiple articles and benchmark script from Loglizer implements loading of HDFS data with labels and provide partial implementation for loading BGL.

BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768GB memory. The log contains alert and non-alert messages identified by alert category tags. In the first column of the log, "-" indicates non-alert messages while others are alert messages. The label information is amenable to alert detection and prediction research. It has been used in several studies on log parsing, anomaly detection, and failure prediction. BGL logs have richer header with some redundant fields. Header fields are UNIX timestamp, human readable date, job id, human readable time to µs, another job id, user, group and log level. Log messages themselves are a bit more cryptic when compared to HDFS. Some messages are still mostly English sentences as first log in example below. Other messages, as second line in example, are more dense and often use hexadecimal parameters values and other not human friendly formats

Most of the log files are original text and unstructured files. We take a dataset from the Blue Gene/L (BGL) supercomputer system of Lawrence Livermore National Laboratory (LLNL) as an example. Fig. 1 lists some fragments of the BGL dataset:


```
—1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-C:J12-U11  RAS KERNEL INFO instruction cache parity error corrected

—1117843015 2005.06.03 R21-M1-N6-C:J08-U11 2005-06-03-16.56.55.309974 R21-M1-N6-C:J08-U11 RAS KERNEL INFO 141 double-hummer alignment exceptions

—1117956980  2005.06.05  R24-M1-NB-C:J15-U11  2005-06-05-00.36.20.945796  R24-M1-NB-C:J15-U11 RAS KERNEL INFO generating core.728

—KERNDTLB 1118538836 2005.06.11 R30-M0-N9-C:J16-U01 2005-06-11-18.13.56.287869 R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt

—KERNSTOR 1118765360 2005.06.14 R04-M0-ND-C:J15-U01 2005-06-14-09.09.20.459609 R04-M0-ND-C:J15-U01 RAS KERNEL FATAL data storage interrupt
```

Figure 1: Examples of BGL original datasets

As shown in Tab. 1, BGL data contains a lot of irrelevant information such as time, node, message type, kernel, etc. For example: "1118765360 2005.06.14 R04-M0-ND-C:J15-U01 RAS KERNEL FATAL data storage interrupt". The timestamp is 1118765360, the record date is 2005.06.14, the node is R04-M0-ND-C:J15-U01, the type of the log message is RAS, the location where the message is generated is KERNEL, the corresponding level of the log message is FATAL, and the content is "data storage interrupt". We remove irrelevant information and keep useful log messages, that is, "data storage interrupt".

Table 1: Details of BGL log

Project	Data
Timestamp	1118765360
Date	2005.06.14
Node	R04-M0-ND-C:J15-U01
Information type	RAS
Generate location	KERNEL
Information level	FATAL
Content	Data storage interrupt

BGL is labeled by log statement with 348k anomalous logs out of 11M logs in dataset. Dataset was split in time window because other methods in benchmark require windows. Labels by line are preserved within windows but additional label for whole window is created. Window is considered anomalous if it contains one or more anomalous statement. Length of time window used is 60 hours, which is proposed default values in BLG loading method in Loglizer benchmark script. It resulted to 3132 windows, with 2481 normal and 651 labeled as anomalous. Number of logs in window varied extensively from some empty windows which were discarded to maximum of 184265 log statements. But most windows contain reasonable number of logs since mean is 1504.73 and median is 78. Figure 6.2 show histogram o window lengths for BGL dataset.

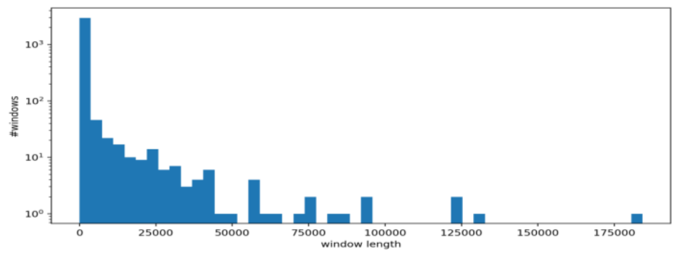


Figure 2: Window length distribution BGL

3.2. HDFS Data

HDFS stands for Hadoop Distributed File System3. LogHub(https://github.com/logpai/loghub/tree/master/HDFS) provides two parts, or in fact two separate datasets, for HDFS. Smaller part is labeled dataset which was originally presented in [26]. Description of this part from Loghub project: This log set is generated in a private cloud environment using benchmark workloads, and manually labeled through handcrafted rules to identify the anomalies. The logs are sliced into traces according to block ids. Then each trace associated with a specific block id is assigned a groundtruth label: normal/anomaly (available in anomaly_label.csv). Second larger part are unlabeled HDFS logs collected by LogHub authors in labs of The Chinese University of Hong Kong. This huge dataset (over 16GB) consist of logs from one name node and 32 data nodes. Example of one HDFS log statement is shown below. HDFS logs have standard header composed from date, time, pid number, log level and component. Log message is then simple English sentence with some parameters in human readable form.

```
081109 203645 175 INFO dfs.DataNode$PacketResponder:
Received block blk_8482590428431422891 of size 67108864 from /10.250.19.16
```

Large unlabeled dataset is used for training fastText language model. And smaller labeled dataset is used for anomaly detection training and evaluation. Unfortunately labels are provided only on block level, as already mentioned in description above. This suggest creation of windows containing logs from one block. Such windows are essentially session windows, and differences are only in HDFS terminology. There is 575061 windows in total, when split by block ID. With 16838 labeled as anomaly, which makes about 2.93% windows in dataset. Figure 6.1 shows histogram of window lengths, which varies from 2 to maximum of 298 log statements, with mean 19.43 and median 19.

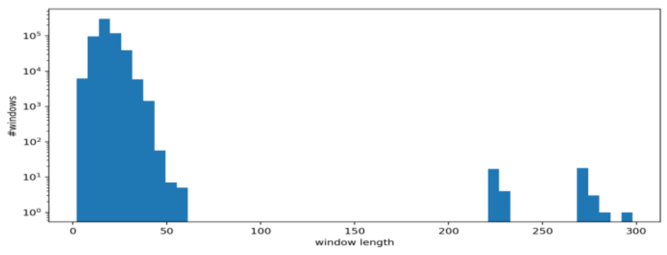


Figure 4: Window length distribution HDFS

3.3. Collected Data

Custom fastText models were trained for log embedding on each dataset. Large unlabeled part of HDFS dataset was used for HDFS model and whole BGL dataset for BGL model. Additionally, several parameter combinations were tried for each dataset. FastText has three parameters which directly affect resulting embedding.

Embedding dimension with default value 100, and bounds for n-grams lengths with default range 3-6. Two embedding dimensions (50, 300) and n-gram range 1-1, were used in addition to default fastTest values. So total of 6 models with different parameter combination were trained for each dataset. Embedding values were computed for selected sample of log statements containing 11 and 12 log templates for HDFS and BGL respectively. Hypothesis is that log statements belonging to the same template will create clusters. In theory, separated clusters allow LSTM based models extract information about template, while variance within cluster represents different parameter values.

All embedding models use high dimensions which cannot be examined directly, so methods of dimension reduction are used for visualization. PCA and t-SNE were tried as they are common and popular methods for dimension reduction. Presented figures use t-SNE reduction to 2 dimension, because it showed results with cleaner separation of clusters.

4. Methods and techniques

4.1. Anomaly detection techniques

The following gives various techniques used in system log data anomaly detection.

A. Statistical Anomaly Detection Models

In Statistical methods, the log dataset is organized in terms of its overall statistic distribution and the data points which stand out or do not conform to this distribution are removed or examined [8]. One possible method is to count the relative frequency of words/events and to examine those with less frequency since it's assumed that anomalies would be sudden and infrequent. Another approach would be to set certain threshold for various parameters above or below of which would denote an anomaly. However such a method is heavily biased. Additionally it can be used only for detecting point anomalies.

B. Nearest Neighbor Based Techniques

This concept is based on the assumption that normal data instances locate in dense neighborhoods, while anomalies lie far from their closest neighbors. Euclidean, Hamming (for equal length), Mahalanobis distances or K-Nearest neighbor technique can be used for anomaly detection. Though this method works well in certain situations, it fails when applied to datasets with an unpredictable distribution with both sparse and dense regions.

C. Clustering Based Anomaly Detection Methods

Clustering methods fundamentally rely on the assumption that normal data instances belong to a cluster in the data, while anomalies either do not belong to any cluster or lie far away from their closest cluster centroid. One such approach is to maintain micro clusters which will be continuously updated and deleted as the log size grows. K-Means clustering, D-Stream, DBSCAN, Modified and hybrid clustering approach, FCM, DENstream, BIRCH with k-means and BIRCH with CLARANS, CURE with k-Means and CLARANS, HDDstream[10] are some popular techniques used for anomaly detection.

D.Support Vector Machines

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side [13]. Often, SVMutilizes the sequential nature of log file. One major disadvantage of this technique is that as the number of features grow exponentially, it will be difficult to balance message information with sequence length.

E. Bayesian Networks

A Bayesian network represents the causal probabilistic relationship among a set of random variables, their conditional dependences, and it provides a compact representation of a joint probability distribution. It consists of two major parts: a directed acyclic graph and a set of conditional probability distributions. This approach is like a classification problem, where a trained Bayesian network on training dataset aggregates information from different variables and provides an estimate on the expectancy of that event to belong to normal/abnormal class for unseen test dataset. The biggest disadvantage of this technique is that they rely on the availability of accurate labels for various classes, which is, most often not possible.

F. Neural Network Based Models

Recurrent Neural Network (RNN) and auto encoder-decoder are two popular techniques used presently. There has been a proliferation of Neural Network based anomaly detection methods in the past decade. At present, several commercially available tools make use of Artificial Neural Network (ANN) advantages. An RNN is a class of ANN where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feed forward neural networks, RNNs can usetheir internal state (memory) to process sequences of inputs.

4.2. Existing Methods

In this section, we introduce six existing methods, which utilize popular neural networks to conduct log-based anomaly detection. They have a particular choice of the model loss and whether to employ the semantic information of logs. We would like to emphasize different combinations (with respect to the model's characteristics and the problem at hand) would yield different methods. For example, by incorporating different loss functions, LSTM models can be either unsupervised or supervised; one method uses the index of log events purely may also accept their semantics; model combinations are also possible as demonstrated by Yen et al, i.e., a combination of CNN and LSTM. The selected four unsupervised methods are introduced as follows:

DeepLog. which is the first work to employ LSTM for log anomaly detection. Particularly, the log patterns are learned from the sequential relations of log events, where each log message is represented by the index of its log event. It is also the first work to detect anomalies in a forecasting-based fashion, which is widely-used in many follow-up studies.

LogAnomaly. To consider the semantic information of logs, [proposed LogAnomaly. Specifically, they proposed template2Vec to distributedly represent the words in log templates by considering the synonyms and antonyms therein. For example, the representation vector of the word "down" and "up" should be distinctly different as they own opposite meanings. To this end, template2Vec first searches synonyms and antonyms in log templates and then applies an embedding model, dLCE, to generate word vectors. Finally, the template vector is calculated as the weighted average of the word vectors of the words in the template. Similarly, LogAnomaly adopts forecasting-based anomaly detection with an LSTM model. In this paper, we follow this work to evaluate whether log semantics can bring performance gain to DeepLog.

Logsy is the first work utilizing the Transformer to detect anomalies on log data. It is a classification-based method, which learns log representations in a way to better distinguish between normal data from the system of interest and abnormal samples from auxiliary log datasets. The auxiliary datasets help learn a better representation of the normal data while regularizing against overfitting. Similarly, in this work, we employ the Transformer with multi-head self-attention mechanism. The procedure of anomaly detection follows that of DeepLog, i.e., forecasting-based. Particularly, we use two types of log event sequences: one only contains the indices of log events as in DeepLog, while the other is encoded with log semantic information as in LogAnomaly .

Autoencoder, were the first to employ autoencoder combined with isolation forest for log-based anomaly detection. The autoencoder is used for feature extraction, while the isolation forest is used for anomaly detection based on the produced features. In this paper, we employ an autoencoder to learn representation for normal log event sequences. The trained model is able to encode normal log patterns properly. When dealing with anomalous instances, the reconstruction loss becomes relatively large, which serves as an important signal for anomalies. We also evaluate whether the model performs better with logs' semantics.

Log-based Anomaly Detection. The selected two supervised methods are introduced as follows:

LogRobust observed that many existing studies of log anomaly detection fail to achieve the promised performance in practice. Particularly, most of them carry a closed-world assumption, which assumes: 1) the log data is stable over time; 2) the training and testing data share an identical set of distinct log events. However, log data often contain previously unseen instances due to the evolution of logging statements and log processing noise. To tackle such a log instability issue, they proposed LogRobust to extract the semantic information of log events by leveraging off-the-shelf word vectors, which is one of the earliest studies to consider logs' semantics as done.

More often than not, different log events have distinct impacts on the prediction result. Thus, LogRobust incorporates the attention mechanism into a Bi-LSTM model to assign different weights to log events, called attentional BiLSTM. Specifically, LogRobust adds a fully-connected layer as the attention layer to the concatenated hidden state h . It calculates an attention weight (denoted as a), indicating the importance of the log event at time step t as $at = tanh(W\ a\ t \cdot ht)$, where $W\ a\ t$ is the weight of the attention

5. Experiments

5.1. Evaluation

In this section, we evaluate six DL-based log anomaly detectors on two widely-used datasets and report the benchmarking results in terms of accuracy, robustness, and efficiency. They represent the key quality of interest to consider during industrial deployment.

Accuracy measures the ability of a method to distinguish anomalous logs from normal ones. This is the main focus in this field. A large false-positive rate would miss critical system failures, while a large false-negative rate would incur a waste of engineering effort.

Robustness measures the ability of a method to detect anomalies with the presence of unknown log events. As modern software systems involve rapidly, this issue starts to gain more attention from both academia and industry. One common solution is leveraging logs' semantic information by assembling word-level features.

Efficiency gauges the speed of a model to conduct anomaly detection. We evaluate the efficiency by recording the time an anomaly detector takes in its training and testing phases. Nowadays, terabytes and even petabytes of data are being generated on a daily basis, imposing stringent requirements on the model's efficiency.

5.2. Experiment Design

Log Dataset Loghub, a large collection of system log datasets. Due to space limitations, in this paper, we only report results evaluated on two popular datasets, namely, HDFS and BGL. Nevertheless, our toolkit can be easily extended to other datasets. Table 1 summarizes the dataset statistics.

HDFS. HDFS dataset contains 11,175,629 log messages, which are generated by running map-reduce tasks on more than 200 Amazon's EC2 nodes [6]. Particularly, each log message contains a unique block_id for each block operation such as allocation, writing, replication, deletion. Thus, identifier-based partitioning can be naturally applied to generate log event sequences. After preprocessing, we end up with 575,061 log sequences, among which 16,838 samples are anomalous. A log sequence will be predicted as anomalous if any of its log windows, W, is identified as an anomaly.

BGL. BGL dataset contains 4,747,963 log messages, which are collected from a BlueGene/L supercomputer at Lawrence Livermore National Labs. Unlike HDFS, logs in this dataset have no identifier to distinguish different job executions. Thus, timestamp-based partitioning is applied to slice logs into log sequences. The number of the resulting sequences depends on the partition size (and stride). In the BGL dataset, 348,460 log messages are labeled as failures. A log sequence is marked as an anomaly if it contains any failure logs.

Evaluation Metrics. Since log anomaly detection is a binary classification problem, we employ precision, recall, and F1 score for accuracy evaluation. Specifically, precision measures the percentage of anomalous log sequences that are successfully identified as anomalies over all the log sequences that are predicted as anomalies: $precision = (TP) / (TP+FP)$; recall calculates the portion of anomalies that are successfully identified by a model over all the actual anomalies: $recall = (TP)/ (TP+FN)$; F1 score is the harmonic mean of precision and recall: $F1\ score = 2 \times precision \times recall\ /\ precision + recall$. TP is the number of anomalies that are correctly disclosed by the model, FP is the number of normal log sequences that are wrongly predicted as anomalies by the model, FN is the number of anomalies that the model misses.

Experiment Setup. For a fair comparison, all experiments are conducted on a machine with 4 NVIDIA Titan V Pascal GPUs (12GB of RAM), 20 Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, and 256GB of RAM. The parameters of all methods are fine-tuned to achieve the best results. To avoid bias from randomness, we run each method five times and report the best result. For all datasets, we first sort logs in chronological order and apply log partition to generate log sequences, which will then be shuffled. Note we do not shuffle the input windows, W, generated from log sequences. Next, we utilize the first 80% of data for model training and the remaining 20% for testing. Particularly, for unsupervised methods that require no anomalies for training, we remove them from the training data. This is because many unsupervised methods try to learn the normal log patterns and alert anomalies when such patterns are violated. Thus, they require anomaly-free log data to yield the best performance. Nevertheless, we will evaluate the impact of anomaly samples in training data. For log partition, we apply identifier-based partitioning to HDFS and fixed partitioning with six hours of partition size to BGL. The default values of window size and step size are ten and one, which are set empirically based on our experiments. For HDFS and BGL, we set k as ten and 50, respectively. In particular, a log event sequence will be regarded as an anomaly if any one of its log windows, W, is predicted as anomalous.

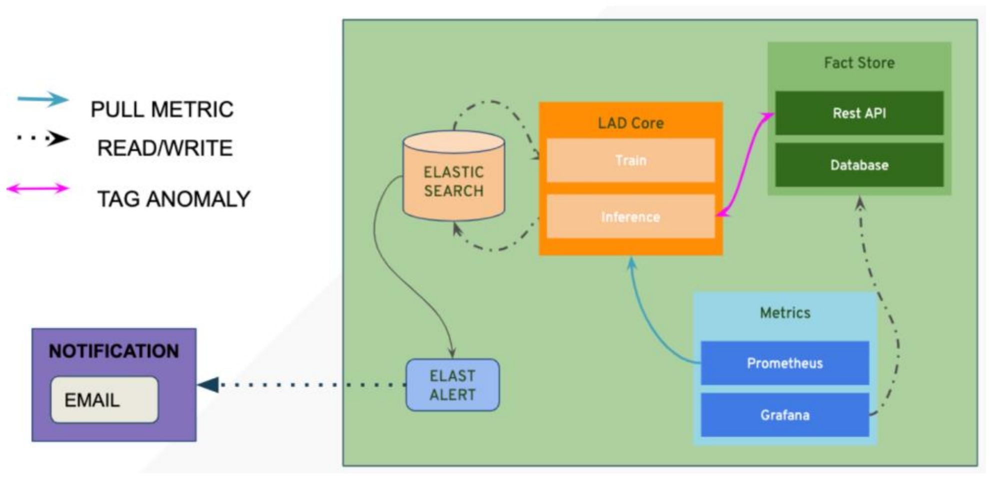


Figure 5. System Architecture Overview

6. System overview

6.1. Architecture

This is a high level overview of the architecture for log anomaly detector which consists of three components.

Machine Learning (LAD-Core) Contains custom code to train model and predict if a log line is an anomaly. We are currently using W2V (word 2 vec) and SOM (self organizing map) with unsupervised machine learning. We are planning to add more models.

Monitoring (Metrics) To monitor this system in production we utilize grafana and prometheus to visualize the health of this machine learning system.

Feedback Loop (Fact-Store) In addition we have a metadata registry for tracking feedback from false_positives in the machine learning system and to providing a method for ML to self correcting false predictions called the “fact-store”.

Figure 5 shows the architecture of our project. Log anomaly detection architecture generally involves the following steps:

1.Data collection: Logs are collected from different sources such as applications, servers, network devices, and security devices.

2.Data preprocessing: Raw log data is preprocessed to remove irrelevant or redundant information, and transformed into a format suitable for analysis. This step may also involve normalization, data cleaning, and data reduction techniques.

3.Feature extraction: Relevant features are extracted from the preprocessed log data, which may include timestamps, log types, message content, and other contextual information.

4.Model selection: Appropriate machine learning models are selected for anomaly detection based on the type of log data and the problem domain. Commonly used models include rule-based systems, clustering algorithms, statistical models, and deep learning models.

5.Model training: The selected model is trained using labeled log data, where anomalies are identified and labeled as such.

6.Anomaly detection: The trained model is used to detect anomalies in new log data by comparing it with the learned patterns from the training data. If an anomaly is detected, an alert is generated for further investigation.

7.Alert generation and reporting: When an anomaly is detected, an alert is generated to notify the appropriate stakeholders, such as system administrators or security analysts. This step may also involve generating reports and visualizations to help understand and diagnose the anomaly.

8.Model evaluation and improvement: The performance of the anomaly detection model is regularly evaluated using metrics such as precision, recall, and F1 score, and the model is continuously improved through feedback and retraining.

6.2. Results

This document provides a high-level overview of the architecture for log anomaly detection, which consists of three components. These include machine learning (LAD-Core), monitoring (Metrics), and feedback loop (Fact-Store). The document also details the steps involved in log anomaly detection architecture, including data collection, preprocessing, feature extraction, model selection, model training, anomaly detection, alert generation and reporting, and model evaluation and improvement.

Through this document, we have learned about the importance of log anomaly detection and how it works. We have also discovered some of the challenges involved in log anomaly detection, such as the need for data preprocessing and feature extraction. However, we can solve these problems by using appropriate machine learning models and continuously improving the model through feedback and retraining.

For future extensions or new applications, we suggest exploring additional machine learning models for anomaly detection and incorporating natural language processing techniques for better feature extraction. We also recommend exploring how log anomaly detection can be used in other domains, such as cybersecurity and finance.

7. Conclusion

Log anomaly detection is a critical area of research in the field of cybersecurity and IT operations. The goal of log anomaly detection is to automatically detect abnormal events or behaviors in log data that indicate potential security threats or operational issues. Over the years, various approaches and techniques have been developed for log anomaly detection, ranging from rule-based systems to more advanced machine learning models such as deep learning.

Recent research in log anomaly detection has focused on developing more accurate and efficient machine learning models, improving feature extraction techniques, and addressing the challenges of large-scale log data analysis. There has also been a growing interest in leveraging unsupervised learning techniques for log anomaly detection, as they can detect unknown or previously unseen anomalies without the need for labeled training data.

While significant progress has been made in log anomaly detection research, there are still challenges that need to be addressed, such as the issue of data imbalance, the lack of interpretability of machine learning models, and the need for more robust evaluation metrics. Additionally, the constantly evolving nature of security threats and IT environments presents a continuous challenge for log anomaly detection research.

Overall, log anomaly detection is an important area of research that has the potential to enhance the security and reliability of IT systems, and further advancements in this field are expected to have a significant impact on the cybersecurity and IT operations industry.

The fundamental hypothesis of this work, concerning ability to differentiate the text which is similar to previously during training Vs newer text, through RNN based language modeling worked quite well. We were able to show reduction in log file by up to 85% using raw log files, while keeping more that 77% of useful information intact. For the smaller sequence length, addition of domain knowledge in terms of tokenization helped improve the accuracy, however even better accuracies could be achieved by simply increasing the sequence length, and using GRU cell. Analysis on false positives reveal a few trends, such as words which are used only in one or two unique phrases tend to get high loss in the during testing. Also that token which contain domain specific information, which is subject to change each run, sometimes results in high loss. Not all “newer” text is of same importance in debugging the log file. And the next step is to train the language model using the additional text from “failed” log files which is found to be not useful for debug, however it’s somewhat new. Addition of such text into training model is expected to further compress the final log file size. We also believe that using the language model, and suitable training data, we can remove the high variability tokens such containing information such as ‘block name’ and ‘design environment variables’, and then they can be replaced by fixed tokens. This should improve the performance of the anomaly detection significantly.

In this work we have analyzed data mining methods for anomaly detection and proposed an approach for discovering security breaches in log records. Such approach generates rules dynamically from certain patterns in sample files and is able to learn new types of attacks, while minimizing the need of human actions.

8. References

[1] Ahmed Umar Memon (2008). “Log file categorization and anomaly analysis using grammar inference”. Master thesiss, Queen’s University Kingston, Ontario, Canada

[2]Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 1-58.

[3]Wang, Y., Jiang, J., & Yuan, C. (2019). A survey on log anomaly detection. Journal of Network and Computer Applications, 129, 89-105.

[4] Gao, J., Wang, Y., Li, Z., & Li, J. (2021). Anomaly detection for log data: A deep learning approach. Journal of Parallel and Distributed Computing, 151, 1-13.

[5] https://github.com/topics/log-based-anomaly-detection

[6] Log Anomaly Detection Using Machine Learning. Larry Lancaster https://www.zebrium.com/blog/using-machine-learning-to-detect-anomalies-in-logs

[7] https://github.com/logpai

[8] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” 2017.

[9] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, “Dynamic log file analysis: An unsupervised cluster evolution approach detection,” Computers & Security, vol. 79, pp. 94 – 116, 2018.

[10] Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection. https://arxiv.org/pdf/2107.05908.pdf