Anomaly Detection in Log Events

Aamir M. Khan
University of British Columbia
Kelowna, Canada
jadoon.engr@gmail.com

Conrad Yeung
University of British Columbia
Kelowna, Canada
conrad.yeung08@gmail.com

ABSTRACT

Modern large-scale distributed systems use logs to record system run-time information. Anomaly detection methods are widely used to maintain integrity of those systems/logs. Traditionally, logs were manually inspected using search queries and matching rules. But with the advent of big data, such logs can no longer be inspected manually. In this paper we analyse some of the anomaly detection methods applied to a particular case study production log dataset. Our work provides insight into the working anomaly detection mechanism and provides guidelines for adoption of these methods and references for future development.

KEYWORDS

anomaly detection, log events, datasets, feature extraction

ACM Reference Format:
Aamir M. Khan and Conrad Yeung. 2021. Anomaly Detection in Log Events. In *Proceedings of ACM Conference (Conference'21).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As the name suggests, anomaly detection is the identification of rare events that differ from the norm [2]. Anomaly detection has a wide range of applications from being a method of fraud detection to identifying abnormalities in health systems or even identifying performance issues in applications. One of the more popular applications of anomaly detection is in log analysis. As the world becomes more technologically advanced, systems generate more data recorded in logs. Logs record run-time information of systems such as the time of the event, description of the event and state of the event [3]. In the case of security breaches or system failures, developers are able to use the information stored in log files to identify and address issues that arise. While log analysis was manually done in the past, it is becoming increasing infeasible due to the necessary domain knowledge of modern systems and the sheer size of log files. For example, it was estimated that modern systems generate around 50 GB of log events (120 to 200 million lines) per hour [9]. Though there are many automated log analyzers available online, finding the most optimal combination of log parsing and anomaly detection algorithms is still an active area of research.

Conference'21, May 2021, Kelowna, BC, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper we aim to recreate some of the results seen in the papers by P.He et al. [3] and S.He et al. [4] We will be using the following system logs and methods in for our anomaly detection:

- Log files: Hadoop File System (HDFS, with labels) and Distributed System Coordinator (Zookeeper)
- Log parsing: Iterative Partitioning Log Mining (IPLoM)
- Feature Extraction: Session window (for HDFS only), fixed window and sliding window
- Supervised Model (HDFS only): Logistic Regression
- Unsupervised Model: Principal Component Analysis

2 LITERATURE REVIEW

In this section we will explain our choice in log parsing methods, feature extraction algorithms and machine learning models.

2.1 General Log Anomaly Detection Process

Log anomaly detection typically consists of four main steps: log collection, log parsing, feature extraction and anomaly detection [5]. The log collection phase is relatively straight forward and involves acquiring log files. Log parsing is the processing of digesting the log file and generating an event template. As logs are plain text consisting of constant parts and variable parts, it is necessary to isolate the constant part in order to create a log event. For example, the content from a log from the Zookeeper system might be "Notification time out: 3200", where "Notification time out:" would be the constant part and labelled "Event1", and "3200" the variable part. Feature extraction involves converting the parsed log into an event count matrix where each column represents an event, each row represents some aggregation of logs (for example with a certain block_id, or within five hour windows), and each value represents the count of the event within the aggregate. Finally, in order to detect any anomalies, the event count matrix is fed into machine learning models for training and testing.

2.2 Log Parsing Methods

There are 2 main types of log parsing methods: clustering based and heuristic based. Clustering based log parsers calculate distances between all logs, group them into clusters and then assign each cluster to an event [4]. Examples of cluster based parsers are Log Key Extraction (LKE) and LogSig. On the other hand, heuristic based parsers count each word at each position in a log, the most frequent words are selected as potential candidates, from the potential candidates, some are chosen as events[4]. Examples of heuristic based parsers are Simple Logfile Clustering Tool (SLCT) and Iterative Partitioning Log Mining (IPLoM).

In a paper published by P.He et al. [3], the accuracy and of the four parsing methods mentioned above as well as their affect on anomaly detection was evaluated. The four parsing methods

were ran on five different log types: BlueGene/L Supercomputer (BGL), High Performance Cluster (HPC), Proxy Client (Proxifier), Hadoop File System (HDFS) and Distributed System Coordinator (Zookeeper). With the exception of parsing HPC logs, IPLoM had the highest parsing accuracy of the four methods while SLCT and LogSig performed quite evenly. It is also worthwhile to mention that the runtime of heuristic parsing were much quicker than their counterparts. When comparing the anomaly detection results using logs parsed with IPLoM, LogSig and SLCT while the detected anomaly rate of all three methods were very similar (63 to 64%) the main differences arose in the false positive rate with IPLoM being the lowest and SLCT being the highest (2.5%, 3.5% and 40% respectively). On top of this, IPLoM requires no parameterization and pre-processing of the logs have little to no effect on its accuracy.

A similar paper by Makanju et al. [8] also concluded that the IPLoM had a significantly higher accuracy than other methods such as SLCT, Loghound and Teiresia.

For this reason, we chose to use the IPLoM algorithm to parse our logs.

2.3 Feature Extraction

We are able to partition our log file to create our feature matrix in three different ways: session window, fixed window and sliding window [4]. The session window method requires that the system log a unique identifier for each entry. For example, HDFS logs contain block_ids which allows us to group events by block_id. We can then count the occurrence of each even for a given blocK_id to create a feature vector. Combining feature vectors together will give us our feature matrix. Similarly, fixed windows group log entries by timestamp. Any entry within a certain fixed time span are grouped and used to create a feature vector similarly to the session window method. For example: with a 5 hour window size, starting at 00:00, feature_vector_1 = [00:00,05:00], feature_vector_2 = [05:00,10:00] etc. Lastly, sliding windows are similar to fixed widows except we 'slide' the widow by some time value. For example: with a 5 hour window size and 1 hour 'slide', starting at 00:00, feature_vector_1 = [00:00,05:00], feature_vector_2 = [01:00,06:00], feature_vector_1 = [02:00,07:00] etc.

In a paper published by S.He et al. [4], the accuracy of supervised methods on the three extraction methods mentioned above were determined. While the precision, recall and F-measure of supervised methods on HDFS data with session windows was high (0.95 to 1) for both Training and Test set, the authors did not report the accuracies on the HDFS data with fixed or sliding sessions. They did however compare fixed and sliding windows on the BGL data and found that while the precision of the two methods are similar, the sliding window method tends to increase recall and the f-measure by around 0.2 and 0.1 respectively.

We chose to implement all three feature extraction methods, especially to quantify the differences between session windows and the two methods utilizing timestamps.

2.4 Machine Learning Models

The paper by S.He et al.[4] compares three supervised machine learning methods: Logistic regression, Decision Tree and Support Vector Machine (SVM) and three unsupervised methods: Log clustering, Principal Component Analysis (PCA) and Invariants Mining. In terms of supervised learning, all three methods performed equally well on BGL test data using fixed windows with a precision, recall and f-measure of 0.95, 0.57 and 0.71 respectively. On the other hand, there was a large disparity between unsupervised methods when tested on BGL data with Invariant Mining having the highest precision, recall and f-measure, followed by Log clustering and PCA (differences of 0.3, 0.2, 0.4 for precision, recall and f-measure respectively). When the unsupervised methods were applied to HDFS data, while Invariant Mining had the highest recall and f-measure, the disparity between methods was much smaller with PCA even having the highest precision (differences of 0.1, 0.2, 0.1 for precision, recall and f-measure respectively).

As we are dealing with HDFS and Zookeeper data, we chose to implement Logistic regression as our supervised method and PCA as our unsupervised method.

3 BACKGROUND

This section will aim explain the workings and purpose of methods and models used in this paper.

3.1 IPLoM

Introduced in papers by A.Makanju et al. [6][7][8], IPLoM was specifically created as a way to identify event patterns within logs. The IPLoM method works in four hierarchical steps:

First, it partitions each log entry by length. For example a log with a message "Deleting block (*) file (*)" will have a token length of five. This is on the assumption that similar log messages have the same token length.

Second, logs partitioned by token length are further partitioned by token position. For example if a cluster with 'm' logs with token size 'n' will be represented as a 'm x n' matrix with the values being the token value at position 'n'. Clusters are then further partitioned by identifying the column with the least number of unique values. This is on the assumption that this column will most likely contains a constant value.

Third, in the final partitioning step, logs are partitioned based on the mapping relationship of two columns (token position). The two columns are chosen based on having the highest unique count values at that token position. The values at these 2 positions are then compared to derive their relationship. There are 4 possible relationships: 1 to 1 relationship, 1 to Many or Many to 1 and Many to Many. In the 1 to 1 case, logs that contain the same 1 to 1 relationship are partitioned together. In the case of 1 to Many or Many to 1, the Many side could represent either constant or variable values. If the many side is found to be constant, logs are partitioned based on unique values at the Many position. Many to Many relationships are partitioned together.

Finally, cluster descriptions are given to each partition. At this point it is assumed every partition contains log entries with the same line format. Within a partition, unique values at each position are counted. If there is only one value at that position, that value is considered constant, else variable. Values at variable positions are then replaced with a (*).

In terms of parameterization, there are 4 optional parameters that have shown little effect on performance.

3.2 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF [12][11] is used in Natural Language Processing (NLP) to score words thus allowing textual data to be fed into machine learning algorithms. In log anomaly detection, it is used as part of the feature extraction step where event vectors (textual data) are converted to feature vectors (numerical data). Rather than using a Bag of Words approach were counts of events are used, TF-IDF forms the log sequence vector based on the frequency and weight of the event.

3.3 Supervised Learning: Logistic Regression

Supervised machine learning models [4] derive a model with the aid of labelled training data. In the case of log anomaly detection, the labels would correspond to whether a log entry is "Normal" or an "Anomaly". Logistic regression classifies logs using a probability model. Using the logistic function, the probability of group membership for each state can be calculated. For example, the probability that block_id123 is an anomaly. Once the probability passes a certain threshold, 0.5, the feature vector will be classified as being part of that state (i.e. anomaly). Feature vectors with a probability below 0.5 will not be part of that state (i.e. normal).

3.4 Unsupervised Learning: PCA

On the other hand, unsupervised machine learning models [10][4][1] do not utilize labelled training data. PCA is commonly used for dimension reduction. Essentially it projects high dimensional data onto a reduced dimensional space composed of principal components. The chosen principal components are chosen that they encompass most of the variation in the high dimensional data. In terms of log anomaly detection, PCA is used to identify patterns between dimensions of feature vectors. Once we have a reduced space that captures 95% of the original variation (by dropping principal components), we can project our feature vectors onto the space to determine the reconstruction error (length of the projection). Feature vectors with large reconstruction errors greater than some threshold (Qa) can then be classified as an anomaly with confidence (1-a). Capturing too much variance (i.e. when the number of dimensions equals number of principal components) will lead to little reconstruction error and therefore little to no anomalies detected.

4 METHODOLOGY

The two main packages we used in this paper are Logparser [3],[13] which contains the IPLoM code and Loglizer [4] which contains code for feature extraction as well as Logistic regression and PCA. The HDFS and Zookeeper logs were supplied from Loghub [4].

4.1 Log Parsing

Both Zookeeper and HDFS log files contain 2000 log entries. Dates on the Zookeeper log ranged from 17:41:44 29-07-2015 to 11:26:28 24-08-2015 and were unlabelled. Dates on the HDFS log had a much smaller time range beginning from 00:01:48 09-11-2008 to 10:20:17 11-11-2008 with labels available. Log files were parsed using the IPLoM method mentioned previously. Descriptions were then standardized into "Event1", "Event2" etc.

4.2 Feature Extraction

A Log event matrix comprised of log event vectors was then generated from the log files. HDFS log entries contain "block_ids" which can be used as unique identifiers, as such we are able to use session windows. The HDFS log data was then partitioned via Session window, fixed window and sliding window. The Zookeeper log data was partitioned via fixed and sliding window. Log event vectors were then split into training and testing sets. For the labelled HDFS log data, training and testing sets were split uniformly so that both sets would have the same ratio of Normal:Anomaly logs. The log event matrix was then transformed and normalized into a feature matrix using the tf-idf method.

4.3 Supervised & Unsupervised Anomaly Detection

Once the data processing phase is over, we proceed with trying different models on our data. As we have labels available for anomalies we try both supervised and unsupervised methods.

We use Support Vector Machines (SVM) and Logistic Regression (LR) for supervised classification. This helped us to ensure that there are no issues with our approach by comparing results. Training data from the splitting phase is used to train models. Then test data is used for performance evaluation.

For the unsupervised learning, we used Principal Component Analysis (PCA). We used the training data again (but without having any associated labels) to train the model which is then used to predict the test data outcomes. In practical unsupervised settings we don't have known labels, but here we use the liberty of having test data labels to evaluate model performance.

4.4 Results

As the provided dataset is highly skewed, this is an imbalanced classification problem for which 'accuracy' will not be the right evaluation metric. Both supervised methods result in a precision of 100% which means that all the cases marked by the model as anomaly were correct. However recall, which represents the proportion of detected anomalous cases out of total anomaly cases, is very low at around 14%. This shows us that the model is not working efficiently and is also reflected by a low F1 score of mere 25%. For the case of unsupervised learning, we see that our models are not able to show any significant results.

Highly skewed dataset makes it difficult to train models. Approximately only 3% of the data events are anomalies which are not enough to train models properly for such a small sized dataset. Upon observation of the misclassified entries, we see that all those five events were anomaly data events. In essence, we noted that the performance of these anomaly detection models can be improved by data gathering techniques or providing more anomaly cases.

5 CONCLUSION

Log data is widely used to detect anomalies but traditional methods are prohibitive in large scale systems. This paper explored various

parsing methods, feature engineering, model selection and evaluation techniques for efficient anomaly detection. We compared the performance of various supervised and unsupervised methods and conclude that the given dataset is not large enough and is highly skewed to train machine learning models effectively.

REFERENCES

[1] Corporation, M. PCA-Based Anomaly Detection. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/pca-based-anomaly-detection, June 5th, 2019. [Online].

[2] DeepAI. What is Anomaly Detection? https://deepai.org/machine-learning-glossary-and-terms/anomaly-detection. [Online].

[3] He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. An evaluation study on log parsing and its use in log mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2016), pp. 654–661.

[4] He, S., Zhu, J., He, P., and Lyu, M. R. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (2016), pp. 207–218.

[5] Liu, D. Log Analysis for Anomaly Detection. https://davideliu.com/2019/10/26/log-analysis-for-anomaly-detection/, 2019. [Online].

[6] LogPAI. Iterative Partitioning Log Mining (IPLoM). https://logparser.readthedocs.io/en/latest/tools/IPLoM.html, 2018. [Online].

[7] Makanju, A., Zincir-Heywood, A. N., and Milios, E. E. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering 24*, 11 (2012), 1921–1936.

[8] Makanju, A. A., Zincir-Heywood, A. N., and Milios, E. E. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), KDD '09, Association for Computing Machinery, p. 1255–1264.

[9] Mi, H., Wang, H., Zhou, Y., Lyu, M. R.-T., and Cai, H. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems 24*, 6 (2013), 1245–1255.

[10] Patel, A. A. *Hands-On Unsupervised Learning Using Python.* O'Reilly Media, Inc., March 2019.

[11] Stecanella, B. What is TF-IDF? https://monkeylearn.com/blog/what-is-tf-idf/, May 10th, 2019. [Online].

[12] Wang, J., Tang, Y., He, S., Zhao, C., Sharma, P. K., Alfarraj, O., and Tolba, A. Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors 20*, 9 (2020).

[13] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. Tools and benchmarks for automated log parsing, 2019.