The Value of Data-Centric Approaches to Log Anomaly Detection

Luka Vukovic
Computer Science and Statistics
Faculty of Science
University of British Columbia
luka.vuko@outlook.com

April 7th, 2021

ABSTRACT

With the growing use of online services for distributed computing, log anomaly detection has become a huge area of research for monitoring and maintaining distributed systems. Handling massive data on hundreds, if not thousands of machines is not an easy process and when problems arise, it becomes an even harder process to discover which machines are the culprit. Engineers turned to logging to discover such hidden problems; however, semi-structure logs quickly become uninterpretable in massive production settings rendering manual anomaly detection a near impossible task. With many complex approaches and expansive models already available for such tasks [2,3,6,14,16], we asked if a data-centric approach with a simple LSTM model can yield similar performance to more model-centric approaches [2,3,6]. As such, feature selection was done with great care and we determined that only one was necessary for anomaly detection in Hadoop logs: sequential events within given block IDs. Other features were also explored but yielded negative results, suggesting anomalies in the Hadoop Distributed File System (HDFS) log can be defined by a single feature. After parsing, a bidirectional, 2-layer LSTM was trained in a supervised manner on anomalous and normal block IDs. Model performance was evaluated on a test set of over 115,000 blocks, yielding a precision of 0.99 and recall of 0.99. These results were both impressive but not without the cost of model generalizability. An advantage of knowing higher level features of the anomaly space allows for extremely powerful feature engineering; however, a supervised model requires the labeling of new data for every new environment, which further demands both domain knowledge and manual labour. Our approach illustrates the power of a data-centric approach to anomaly detection and in future work hope to develop a more generalizable unsupervised model which should be tested on other logging environments.

1 INTRODUCTION

Anomaly detection is the identification of rare observations which can serve as indicators for unobserved problems. Depending on the system, anomalies may precede problems acting as warnings, may be the result of unseen problems, or may simply coincide with problems at the time of occurrence. With this flexibility, anomaly detection has applications in all fields serving to prevent and help address underlying issues in any type of system. For example, satellite engineers may need methods to detect irregular vibrations in rocket engines while healthcare professionals need methods to detect biometric anomalies in patients. This paper focuses on anomaly detection in log files, particularly those coming from large-scale online service systems such as the Hadoop Distributed File System (HDFS) [1]. These systems contain hundreds, if not thousands of distributed nodes and machines to help process big data for which typical debugging tools no longer apply when attempting to monitor system performance in production settings [2]. Engineers turned to logging which record events that occur in these massively distributed systems for further examination and troubleshooting purposes. Unfortunately, the nature of massively distributed systems quickly obscures log files rendering them nearly impossible to understand without a certain level of domain knowledge. The question then begs, why use machine learning when we can simply parse log files for messages containing the words “error” and “exception”? The issue is, despite such messages being raised, not all errors are anomalies since distributed systems can simply restart and complete the process elsewhere without any true problem. This is further elaborated on in section 2.2. Secondly, parsing and searching for key words is time consuming and inefficient since returned messages need further examination. This may have initially allowed service providers to address malfunctioning services, but with increasing scales of operation, complexity, and data volume, this is no longer feasible [2].

Researchers responded with new state-of-the-art techniques for such tasks, such as LogCluster [2], LogRobust [3], DeepLog [6], and other older but effective unstructured log analysis techniques [4]. Despite these success stories the sheer variety and growth of distributed online services will always demand up-to-date techniques as even current methods may not perform optimally across all logging systems. To ensure online service providers have options to multiple high performing techniques, we highlight the importance of two items: model simplicity and optimal feature engineering. A stronger data-centric approach can still achieve state-of-the-art performance despite basic model design. In this case, sequential event features remained the best suited feature for our approach using a bidirectional LSTM model.

Sequential event features refer to the order of occurrence of general event types on a given node in a distributed file system. Such information has already proved to be extremely useful for identifying anomalous log events [2,3,5,6]. This is accomplished by converting log information back to a language that is understood by computers. In essence, a numeric value is assigned to each general message structure, so that every event is categorized numerically. Events are then strung together as a sequence in time. One can imagine each event representing a “word” and the event sequence representing a “sentence.” By capturing the event sequences in a system, we essentially capture data in the form of a language like any other that can be interpreted using natural language processing techniques. From there, “grammatically correct sentences” are treated as regular events while “grammatically incorrect sentences” are labeled as anomalies. We validate two things: the predictive power of this sequential feature, and the predictive power of a generic LSTM architecture.

The rest of the paper is organized as follows: a brief overview of anomaly detection, background on logging systems and relevant literature, motivation for our research, our methodology, experiments, results, and finally a brief conclusion to summarize our work.

2 BACKGROUND

In this section we briefly discuss general solutions to anomaly detection, some of their strengths and weaknesses, and then narrow down our discussion to the domain of log anomaly detection. Here we comment on related work in the field and set the stage for our motivation and research questions in the last part.

2.1 A Broad Look at Anomaly Detection

Anomaly detection techniques can fall under three general learning schemes: unsupervised, supervised, or semi-supervised. In the unsupervised context, we do not explicitly know what the anomalies are. Instead, we use various statistical concepts to identify them. As an extremely simple example, an anomaly might just be a data point beyond three standard deviations from the mean. In the supervised context we can train models to explicitly learn known data patterns for anomaly detection. A major disadvantage of supervised models is that it requires domain knowledge to initially understand how to label anomalous events. In many cases, this information is not available. Lastly is semi-supervised learning, where a model is trained on known “normal” data. The model defines for itself what “normal” data looks like then discriminates against anomalies based on how far input data deviates from the norm. Autoencoders for example are neural network-based classifiers which perform dimensionality reduction and learn to recreate data, after which a loss is calculated between generated and input data [7]. If the loss exceeds a certain threshold, we may label the data as anomalous.

Without consideration for learning schemes, we can identify several other general categories for anomaly detection techniques. These include classification techniques, nearest neighbour techniques, clustering techniques, and statistical techniques. Classification techniques include neural network-based models such as replicator networks [8, 9], Bayesian networks [10], support vector machine techniques [11], and rule-based decision trees. These techniques rely on the availability of accurately labeled data, and as classifiers they tend to provide non-probabilistic labels to anomalies when in many cases it is more applicable to have a probabilistic score. Despite this there have been potential solutions to provide probabilistic scores [12]. Nearest neighbour techniques assume that anomalies occur in groups outside neighborhoods of normal data. This is not always true, particularly if anomalous log messages look exactly like normal log messages. An advantage however would be that these techniques are unsupervised in nature. It is also worth mentioning that the computational complexity of nearest neighbor techniques may come as a disadvantage when dealing with massive amounts of data. Clustering techniques carry a similar assumption, that anomalies will occur outside normal clusters. This may be problematic in two situations: if anomalies cannot be differentiated from normal clusters, or if anomalies form their own sizeable clusters like normal clusters. Lastly are the statistical anomaly detection which assume anomalies occur in low probability regions of a given stochastic model [13]. This technique category is advantageous if the assumed data distributions are realistic and if labels for anomalies are not available. A disadvantage however is that data is not always generated from a known or consistent distribution, especially with increasing data dimensionality [13].

In general, a successful model is highly dependent on the application and domain, which makes technique selection among hundreds of techniques, a very problematic task. Technique selection thus depends on at least two things:

1. The nature of the data (labelled/unlabeled, structured/unstructured, uni/multivariate, continuous/discrete, and so forth).
2. One's domain knowledge and if it is enough to manually distinguish anomalies from normal data.

When no distinguishing features are known, a more generic and flexible approach may be preferred. If distinguishing features are known, a state-of-the-art model can be constructed relatively easily based on that one known anomaly distinguishing feature. This is fortunately the case with the nature of log information since sequential event features are a primary identifier of anomalous data. This is further discussed in the next section.

2.2 Background in Log Anomaly Detection

As mentioned in the introduction, logging and manual examination of abstract log information is not the most effective way to identify problems in online services, despite being a common practice for service monitoring and troubleshooting [2, 14, 15]. Distributed computing systems are generally extremely resilient to failure, so when a particular process fails it can just be restarted elsewhere. Because of this, message keywords such as 'killed' and 'failed' can appear thousands of times in a log and yet be perfectly normal events. The definition of anomaly must go beyond simply finding 'killed' or 'failed' processes. To understand what log data looks like, a sample from the HDFS log dataset is provided in figure 1.

```
081109 204925 673 INFO
dfs.DataNode$DataXceiver: Receiving block
blk_-5623176793330377570 src:
/10.251.75.228:53725 dest:
/10.251.75.228:50010

081109 205035 28 INFO dfs.FSNamesystem:
BLOCK* NameSystem.allocateBlock:
/user/root/rand/_temporary/_task_200811092030
_0001_m_000590_0/part-00590. blk_-
1727475099218615100

081109 205056 710 INFO
dfs.DataNode$PacketResponder: PacketResponder
1 for block blk_5017373558217225674
terminating

081109 205157 752 INFO
dfs.DataNode$PacketResponder: Received block
blk_9212264480425680329 of size 67108864 from
/10.251.123.1
```

Figure 1: An HDFS log sample.

We can see that log data is semi-structured with date time information in the first two columns, a session code in the third column, a message type in the fourth column (*INFO*, *WARN*, etc.), and the log message itself in the final column. This information can be used to extract features of interest with basic parsing techniques. Notice it may be tempting to extract all possible features from this type of data, yet there is no guarantee of successful anomaly detection. For example, Reidemeister *et al.* [16] use decision trees with a recall of 78%, which is quite good but not perfect. The Hadoop log structure presents a generic print structure where the information block ID and machine IP addresses are unique. From the HDFS documentation,

> "*A DataNode identifies block replicas in its possession to the NameNode by sending a block report. A block report contains the block ID, the generation stamp and the length for each block replica the server hosts.*" [20]

When anomalies occur, they are thus defined by the block ID which may reappear thousands of times. It should be noted that anomalies are not individual logging events, but instead unanticipated blocks of information. As such, the sequence of messages for a given block ID is what we can define as an anomaly. For example, a block that has been ***received*** cannot be ***receiving*** in later events, thus is anomalous. This is the primary assumption for anomaly detection in this domain and why sequential feature engineering is currently the best approach for anomaly detection [2,3,6]. Other previously mentioned models, (LogCluster, LogRobust, and DeepLog) all use sequential features to attain state-of-the-art performance [2,3,6]. This is by no means a coincidence.

Despite this, the complexity of such methods is somewhat higher than it needs to be, and we would like to propose a similar but more simplistic approach using the same primary feature. The motivation is that simplicity allows for better generalization across logging systems. A very generic parsing structure, as well as vanilla model design, can easily be adapted to new systems in contrast to a highly specialized model. Our generic LSTM model can achieve similar performance and further validates the importance of sequential feature engineering.

2.3 Research Questions

The primary question explores the value of data-centric approaches to log anomaly detection. This is relative to more model-centric approaches. As such, we asked if a simple LSTM model can match the performance of more complex models (such as those referenced in 2.2) using only one predictor feature? Since anomaly labels were generated by a strict set of rules, our main assumption was that if our feature can reflect those rules, then anomalies can be defined by a as little as one predictor variable. Note that these anomaly identifying features are not always known across domains. In those cases, unsupervised approaches may be more suitable. Fortunately, sequential log features are known to have merit in anomaly detection within this domain. As such, we trained our primary model using this.

A second and more minor research question relates to a problem that is often seen in supervised anomaly detection: the issue of highly imbalanced class distributions in training data [21]. The data used for this project is highly imbalanced, at roughly 2.9% being anomalous and the other 97.1% being normal. This begs the question, is data imbalance be an issue for our basic LSTM model?

Lastly, we wanted to briefly explore if there is value in any non-sequential features. The motivation being that there could be other anomaly predictors not captured by sequential data or by data grouped on block IDs. If such features exist, performance on anomaly detection models could be further improved in later projects.

3 METHODOLOGY

In this section, we detail how the research problem was approached computationally starting from dataset selection, feature extraction (data processing), model design, and the statistical methods used for model evaluation.

3.1 Dataset

The dataset used in our experiments was a Hadoop system log file with over 11 million events and 0.5 million block IDs. It was originally published by Lin, Q. et al. in 2016. The log was generated from a Hadoop cluster with 46 cores across five machines, where each machine had an Intel® Core™ i7-3770 CPU with 16GB RAM. During logging, two testing applications were executed to simulate a production environment, *WordCount* and *PageRank.* During execution, three types of failure were injected (*Machine Down*, *Network Disconnection*, and *Disk Full*) to simulate service failures in the production environment. Further details can be found in the original paper [2].

3.2 Hardware and Software

3.2.1 Hardware Specification

Model training and testing was performed on hardware detailed in table 1:

Artifact	Information
CPU	Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz, 6 Cores
RAM	16 GB
GPU	NVIDIA GeForce RTX 2070 SUPER

Table 1: Hardware specification

3.2.2 Software Specification

The software used for all elements of this paper was done in Python 3.8. The machine learning libraries primarily used were Pytorch [17], TensorFlow [18], and Scikit-learn [19].

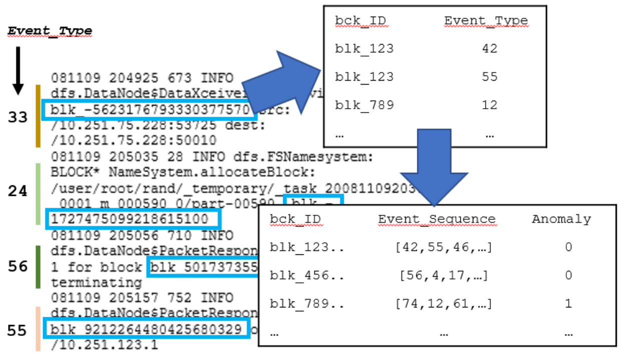
3.3 Experimental Setup

3.3.1 Data Processing and Feature Extraction

To answer our primary research question, we had to parse the log file for event sequences, group them by block ID, then label them as anomalies based on a separate CSV, *anomaly_label.csv*, containing labels for each block. Since the log file contained over 11 million rows of information, much of the parsing code used vectorized operations to:

1) Extract block IDs
2) Extract message structure without block IDs.

Messages are no longer unique without block IDs and could be converted to numeric representations of generalized message types. In total there were 75 different message types. To save RAM in memory intensive processes, data was split into chunks. Each chunk variable was processed, appended to a collector object, and then deleted to save memory drastically reducing processing time and the risk of crashing. Events were then grouped by the rows' respective block ID to produce a final feature as seen in figure 2. The notebooks are available for viewing at the original GitHub repository.



```
081109 204925 673 INFO
dfs.DataNode$DataXceive
blk_-5623176793330377570 src:
/10.251.75.228:53725 dest:
/10.251.75.228:50010
081109 205035 28 INFO dfs.FSNamesystem:
BLOCK* NameSystem.allocateBlock:
/user/root/rand/_temporary/_task_20081109203
_0001_m_000590_0/part-00590 blk_-
1727475099218615100
081109 205056 710 INFO
dfs.DataNode$PacketRespon
1 for block blk_501737355
terminating
081109 205157 752 INFO
dfs.DataNode$PacketRespon
blk_9212264480425680329
/10.251.123.1
```

bck_ID	Event_Type
blk_123	42
blk_123	55
blk_789	12
…	…

bck_ID	Event_Sequence	Anomaly
blk_123..	[42,55,46,…]	0
blk_456..	[56,4,17,…]	0
blk_789..	[74,12,61,…]	1
…	…	…

Figure 2: Sequential Event Feature Extraction.

3.3.2 Model Design

Since the first research question, begs a data-centric approach, a relatively vanilla model had to be chosen. In this case, a two-layer bidirectional LSTM architecture was used and trained solely on the sequential occurrence of events. For valid model evaluation, we had to ensure optimal hyperparameters were selected. One way of doing so is a grid search approach, although this was not feasible with computational and time constraints. Despite this, of the sampled parameters and training conditions, a near optimal model was quickly found. To avoid overfitting a validation set was used to monitor potential discrepancies between training and validation accuracies. The ratio splits for training, validation, and testing are in Table 2.

Data Group	Ratio	Block IDs
Train	0.70	402,542
Validation	0.10	57,506
Testing	0.20	115,013

Table 2: Data split ratios and counts.

To explore our final research question (the usefulness of non-sequential features), we developed two additional semi--supervised autoencoder models trained on 32 principal components derived from 128 engineered features such as event frequencies, message categories, keywords, and so forth. Note that anomalous data was excluded when training the autoencoders. In addition, principal component analysis was performed on these features to see if anomalies could be distinguished based on the labeled data. These models served as an indirect form of exploratory data analysis to understand the usefulness of non-sequential or quasi-sequential features, where quasi-sequential is defined as models trained on sliding windows of data. Thus, information is partially sequential within the window, while no relationship is learned between windows. As discussed in 2.1, each model encodes then decodes input data and if the decoding loss exceeds a certain threshold (determined by an *x* number of standard deviations from the mean loss), an anomaly is labeled. Since multiple block IDs might appear in the same window, All models, hyperparameters, and architectures are found in table 3.

Model	**LSTM Model**
Learning Scheme	Supervised
Feature Type	Sequential
Architecture	64 Feature Embedding Layer 128 Feature, 2-Layered, Bidirectional LSTM Fully Connected Dense Layer with Sigmoid Activation
Epochs	3
Learning Rate	0.0001
Batch Size	1000
	Single Event Autoencoder
Learning Scheme	Semi-supervised
Feature Type	Non-Sequential
Architecture	32 Dense (ReLU) 16 Dense (ReLU) 8 Dense (ReLU) 16 Dense (ReLU) 32 Dense (ReLU) 128 Dense (Sigmoid)
Epochs	8
Learning Rate	-
Batch Size	512
	Convolutional Autoencoder
Learning Scheme	Semi-supervised
Feature Type	Quasi-Sequential
Window Size	6
Architecture	16 feature 2DConv 3x3 (ReLU) 8 feature 2DConv 3x3 (ReLU) 8 feature Deconvolution 3x3 (ReLU) 16 feature Deconvolution 3x3 (ReLU) 1 feature 2DConv 3x3 (Sigmoid)
Epochs	10
Learning Rate	-
Batch Size	512

Table 3: Experimental Models.

3.3.3 Model Evaluation

Models were evaluated on precision and recall, all of which could be computed from a confusion matrix comparing true values to predicted values. Precision refers to the proportion of positive anomaly predictions were correct (i.e., zero false positives would indicate a precision score of 1.0). Recall refers to the proportion of anomalies detected out of all anomalies in the dataset (i.e., Recapturing of all anomalies would indicate a recall score of 1.0). Accuracy (proportion of correct predictions) was not used since it is misleading in the context of binary classification on highly imbalanced datasets. For example, if 97% of log data is normal and not a single anomaly is detected, we will still achieve 97% accuracy while entirely failing at anomaly detection.

4 EXPERIMENTS AND RESULTS

RQ1: *How well does a basic LSTM perform in anomaly detection trained only on the sequential event feature?*

A basic LSTM can match performance and event outperform state-of-the-art methods for log anomaly detection. This is both in terms of precision and recall. A bidirectional 2-layer LSTM had a precision and recall of 0.99 while the next best model we were aware of had a precision and recall of 0.96. These are both comparable performances, however if a million block IDs need to be checked daily in a logging system with data that is on average 2% anomalous, a 3% difference in model recall translates to 600 undetected anomalies daily. As such we stress the importance of maximizing recall. Results are summarized in Table 4.

Model	Precision	Recall
Basic LSTM^1	0.99	0.99
LogRobust[3]	0.96^2	0.96^2
DeepLog[6]	0.95	0.96
LogCluster[2]	0.60	-

Table 4: Model performance comparisons to related state-of-the-art log anomaly detection models.
^{1}Basic LSTM corresponds to a 2 Layer bidirectional LSTM (see Table 3). ^{2}Averaged from 5 final precision/recall scores reported from the original source [3].

It is important to note that all performance metrics in table 4 were derived from HDFS datasets however none uses a single standardized dataset. Results from models not our own were taken directly from the publication source. One case had multiple results based on different operating conditions, so an average of their five conditions was used. In future work, training and testing all models on the same data can provide more validity. Despite this, as a proof of concept, these results are still comparable and suggest the importance of data-centric approaches.

To better understand how such performance could be achieved, we asked what played a larger role, model architecture or feature engineering? Table 5 shows results of varying LSTM architectures and their performance relative to uninformative features trained on autoencoders. Architectural variations in LSTM models yielded almost identical results with consistent training parameters. On the other hand, poor feature engineering led to essentially useless models.

Model			Precision	Recall
One Sequential Event Feature				
LSTM	Bidirectional	2 Layer	0.99	0.99
		1 Layer	0.99	0.98
	Single direction	2 Layer	0.98	0.98
		1 Layer	0.98	0.98
First 32 Principal Components from 128 Non-Sequential Features				
Autoencoder	By event	-	0.12	0.24
Convolutional Autoencoder	By Window	-	0.05	0.25

Table 5: Comparison of model performances based on trained feature types and architectures.

There are however caveats to validity of experimental results in Table 5. Autoencoders are extremely flexible and useful for exploring hidden feature spaces, however the training data structure could have been improved for these results to be more comparable to LSTMs. For example, feature data for the autoencoders was not grouped by block ID unlike the LSTM models. Despite being a flexible model for learning hidden features, the way data was structured for training the windowed autoencoder may fail to give a reliable baseline comparison model. Secondly, semi-supervised learning with manual selection of anomaly thresholds is an extremely technical process and may never yield effective models, even after many design iterations. Thirdly, the 6-event rolling window autoencoder is forced to identify the same anomaly 6 times (once for each rolling window) to label an event as anomalous, otherwise an entire window of what could be normal events would be labeled as anomalous. This impacts the precision in regions where anomalies occur more frequently than normal events partially explaining why the windowed model performed worse than the "by row" model. Additionally, having to identify the same anomaly 6 times may also harshly impact recall if even 1 rolling window of the 6 containing the anomaly is incorrectly labeled. A second, and arguably better metric for studying uninformative feature selection (principal component analysis) is explored in RQ3.

RQ2: *Is data imbalance an issue for our LSTM model?*

Since the model identified nearly all anomalies correctly, there is no strong evidence that data imbalance was an issue in the final LSTM model. This may be due to a few reasons. Firstly, if the data contained reliable hidden features from which to predict anomalies, then it should not require large amounts of either data class to learn these patterns. This would imply that event sequences are not as noisy as previously thought. The lack of noise in event sequences may also reflect an overly consistent anomaly type in the dataset. This could be problematic if true. If we are seeking generalizable features and models, we do not want to overfit to only one type of anomaly assuming there could be many subtypes hinting at different issues. For a solution to be generalizable, larger diversities of anomalies need to be used in testing. To further investigate the subject, new datasets should be used in testing such as those from actual production environments. This would require domain expertise to correctly label and identify anomalies beforehand. Although, a benefit of using production level environments is that logging is produced from a larger number of machines as well as a larger diversity of programs running on these machines. Spending more time with the data is where performance comes from when following a data-centric approach to anomaly detection.

RQ3: *Are there potentially other useful non-sequential features?*

Two feature sets were created. The first set had 128 features that included log message priorities (*info/warn/error*/…), log message categories (DataXceiver, PacketResponder, Name-System…), words per message, time between events, event frequencies in varying time frames of 10 minutes, 1 minute, and 1 second, and message keywords from the messages itself. Categorical features were one-hot encoded, and numeric columns scaled to be between zero and 1. The second feature set was like the first minus the message keyword features, containing 16 features in total. Of these features, most had to do with the timing of events, explaining why their main principal components (PCs) look to be following some line through space. The first two PCs of both feature sets are visible in figure 3, with anomalies in red. Not all points are plotted simply due to computational constraints. Instead, every 1000th point is plotted in the upper quadrants with every other point plotted in the magnified lower quadrants.

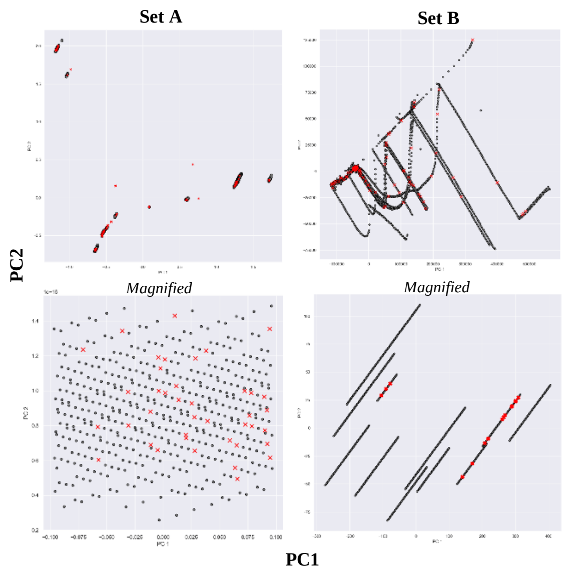


Figure 3: First two principal components (PCs) from uninformative feature engineering. (**A**) data includes structural, time, and message keyword features while (**B**) data only includes structural and time features.

In set A, almost all anomalies are grouped with regular events, which tend to organize themselves into clusters. These clusters are likely representative of message classes in the log file. Looking more closely in the bottom left quadrant, anomalies and normal events are practically homogenous with respect to the first two principal components. This is not ideal. There are however small, isolated clusters of anomalies however these likely represent no more than 15% of the data judging by cluster density and size. If true and if we only trained on the first two principal components, our autoencoder model will only identify these anomalous clusters to score no higher than around 15% on recall. When trained on the first 32 components recall turns out to be around 25% which in certain respects supports this supposition.

With the exclusion of message keyword features, anomalous events in set B fall directly on the paths of regular events through the feature space. It is interesting how set B's feature space is very threadlike in nature, as in, the points follow almost predictable lines through time reflecting the time-based features. Clearly, these first two principal components provide no predictive power to anomaly detection despite being a tempting feature to initially extract. It could be deemed a naïve feature set since these are features likely to be extracted if one has never seen a logging system and how it is structured in higher space (relating to block IDs and what they represent).

Despite these uninformative PCs, we still cannot completely rule out these features because anomalies are underrepresented in the data. As such, even PCs explaining only 2% of variation may be the PCs that distinguish the 2% of anomalies from normal events. In the future, it may be a good idea to visualize other PCs to understand if any do represent these distinctions. Another reason figure 3 may not be representative of the true log feature space, is that the red anomaly labels are not block IDs (which encompass multiple events) but single message events themselves. According to our initial definition in 2.2, multiple red labels could then correspond to the same anomaly. Knowing this, log data from these sets first needs to be grouped by block ID before feature extraction and PCA.

5 CONCLUSIONS

The results of this project serve as an initial source of validation for the power of a data-centric approach to anomaly detection. Understanding the nature of anomalies prior to feature engineering can yield much more performance than countless hours spent on parameter tuning and model architecture development. If one has a higher understanding for what causes or labels anomalies, then one can try to extract features that relate to these higher causes or labels.

In HDFS logs, event sequences act as the primary distinguishing feature. Therefore, an appropriate machine learning model, such as a recurrent neural network, can be applied to find hidden patterns within these higher sequential features. Our parsing method and model may be generic in nature, but it does not yet guarantee generalizability for two reasons. Different logging structures may have anomalies defined by features other than sequential types, and the approach used follows a supervised learning scheme which cannot easily be applied to new log files from new logging systems. This is the greatest pitfall of our approach as it would require domain expertise to label new files for training before the approach could be used. If time and cost is not an issue, then investing in labeling new datasets can yield state-of-the-art performances for anomaly detection. Otherwise, reiterating the importance of sequential features in log analysis, a future unsupervised model may be developed to better generalize a data-centric approach for future and untested logging systems.

ACKNOWLEDGMENT

I thank Naveen Chalasani for his insightful discussions on approaching this problem, in addition to his guidance surrounding the nature of log file systems.

REFERENCES

[1] Hadoop. http://hadoop.apache.org/core

[2] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, Xuewei Chen. Log Clustering Based Problem Identification for Online Service Systems, in Proc. of International Conference on Software Engineering (ICSE), 2016.

[3] Xu Zhang, Yong Xu, Qingwei Lin, et al. 2019. Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019). Association for Computing Machinery, New York, NY, USA, 807–817. DOI:https://doi.org/10.1145/3338906.3338931

[4] Q. Fu, J. Lou, Y. Wang and J. Li, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," 2009 Ninth IEEE International Conference on Data Mining, 2009, pp. 149-158, doi: 10.1109/ICDM.2009.60.

[5] Shang, Weiyi & Jiang, Zhen & Hemmati, Hadi & Adams, Bram & Hassan, Ahmed E. & Martin, Patrick. (2013). Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds. Proceedings - International Conference on Software Engineering. 402-411. 10.1109/ICSE.2013.6606586.

[6] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 1285–1298. DOI:https://doi.org/10.1145/3133956.3134015

[7] Kingma, D. P., and Welling, M. 2019. An Introduction to Variational Autoencoders. Foundations and Trends in Machine Learning: Vol. 12: No. 4, pp 307-392. http://dx.doi.org/10.1561/2200000056

[8] Hawkins, S., He, H., Williams, G. J., And Baxter, R. A. 2002. Outlier detection using replicator neural networks. In Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery. Springer-Verlag, 170–180.

[9] Williams, G., Baxter, R., He, H., Hawkins, S., And Gu, L. 2002. A comparative study of RNN for outlier detection in data mining. In Proceedings of the IEEE International Conference on Data Mining. IEEE Computer Society, 709

[10] Baker, D., Hofmann, T., McCallum, A., And Yang, Y. 1999. A hierarchical probabilistic model for novelty detection in text. In Proceedings of the International Conference on Machine Learning

[11] Vapnik, V. N. 1995. The Nature of Statistical Learning Theory. Springer-Verlag

[12] Platt, J. 2000. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. Adv. Large Margin Classif. 10.

[13] Chandola, V., Banerjee, A., and Kumar, V. 2009. Anomaly detection: A survey. ACM Comput. Surv. 41, 3, Article 15 (July 2009), 58 pages. DOI = 10.1145/154

[14] R. Ding, Q. Fu, J. Lou, Q. Lin, D. Zhang, and T. Xie, Mining historical issue repositories to heal large-scale online service systems. In Proc. 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014), Atlanta, GA, USA, pp. 311–322.

[15] Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In Proc. of the 36th International Conference on Software Engineering (ICSE 2014), June 2014, pp. 24-33.

[16] T. Reidemeister, Miao Jiang and P. A. S. Ward, "Mining unstructured log files for recurrent fault diagnosis," 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, 2011, pp. 377-384, doi: 10.1109/INM.2011.5990536.

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[19] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, pp.2825-2830.

[20] Chansler, R., Kuang, H., Radia, S., Shvachko, K., and Srinivas, S. The Hadoop Distributed File System. http://aosabook.org/en/hdfs.html

[21] Ranga Suri, N. N. R. et al. (2019) Outlier Detection: Techniques and Applications A Data Mining Perspective. 1st ed. 2019. [Online]. Cham: Springer International Publishing.