Anomaly Detection using Sequential Data Modeling

Abstract

Modern systems generate huge amounts of data in the form of log files during execution. The information in log files help us understand the system behavior. This is especially true during maintenance and troubleshooting phases where error messages can be analyzed to understand the root causes of system failures. However, this process involves a lot of manual work by experts to parse the log messages and infer system states. This process becomes unrealistic as the systems grow larger and generate vast amounts of data that is unrealistic for manual human intervention.

Machine learning techniques can be very useful in detecting patterns, and therefore a natural choice in inferring system behavior. Given the complexity of systems – with multiple components interacting with each other – domain expertise still plays a vital role in state inference based on these log messages. Ideally, we would want to leverage domain expert knowledge to some extent in designing the patterns and then use machine learning to identify those patterns.

Natural language techniques are very effective at modeling sequences of data. If a system state can be identified at a particular time, the changes in system state over-time can be modeled as sequential data. This allows recurrent neural networks to identify patterns in state changes, which in turn can be used to classify a state into an anomaly or normal state. Leveraging the patterns identified by the LogCluster algorithm, we have successfully implemented a LSTM based recurrent neural network to identify anomalies in Hadoop file system.

Keywords — Anomaly detection, logfile, LogCluster, LSTM, machine learning, NLP, RNN
Source Code — Publicly available on GitHub.[1]

1. Introduction

In systems management and monitoring, log files are critical in identifying significant or unusual events. Nearly all software contains log files that document a history of actions performed and the results of those actions by the system.[2] To mitigate risks, businesses regularly analyze log files for deviations from the norm: anomalies, errors, suspicious, or unauthorized activity. This is because log analysis provides valuable insight into what has happened across the infrastructure. In fact, log analysis has the potential to detect issues before or as they happen, thus reducing cost, time wasted, and unnecessary delays.

The difficulties in log analysis are two-fold: (1) log files are continuously generated, thus become much too large in volume to examine manually; (2) unstructured data in log files are often too difficult to analyze automatically.[2] Therefore, this paper explores different approaches of log analysis anomaly detection. Our goal is to provide an improved approach in anomaly detection that minimizes human intervention while improving accuracy of these automated

approaches. This review aims to accomplish this by combining the effectiveness of modern NLP techniques and deep learning networks such as LSTM to identify relationships in sequential data with the advantages of data pre-processing and template generation.

Our contribution is a 3–step methodology that utilizes:

1) Data Pre-processing
2) Pattern Recognition using LogCluster
3) Sequential Data Modeling using LSTM

This paper is organized as follows: section 2 provides a background and literature review of effective approaches in anomaly detection; section 3 details our methodology of how we combined pattern detection and sequential data modeling; section 4 discusses our experimental design; section 5 presents our results from different models; and section 6 discusses future work.

2. Background & Literature Review

2.1 Anomaly Detection

Identification of states plays a significant role in inferring system behavior and identifying problems and anomalies. Traditionally, log messages have been parsed to infer system states or find anomalies using data mining approaches such as (1) rules-based approaches; (2) association rules analysis. In particular, identification of states through feature extraction—a technique where raw console logs are parsed into templates and states are extracted from these templates to be used as features in unsupervised learning—is helpful in detecting problems in large-scale systems.

Additionally, applying machine learning starting with feature extraction from log messages has also been a popular alternative for anomaly detection. This includes applying Principal Component Analysis (PCA) to detect anomalies and a decision tree to visualize the results of the PCA. High accuracy from this approach demonstrates that there is valuable information in the states derived from patterns in log messages. However, deep learning techniques such as Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNN) have shown promising results when compared to other algorithms such as Gradient Boosting Decision Tree (GBDT).

Recently, modern natural language processing techniques have emerged as a promising approach to extract information from log files due to the recent advances in sequential data modeling and the potential to find valuable information hidden in the sequence of words within the log message. Wang et al. proposed combining natural language processing for feature extraction with deep learning for anomaly detection.[3] Though Word2vec and Term Frequency-Inverse Document Frequency (TF-IDF) are suitable methods for feature extraction, Google's Word2vec has recently emerged as the popular choice for feature extraction.

Word2vec is a neural network-based word embedding technique that takes text corpus as input and outputs a high-dimensional vector space. Relation between different words is preserved in this vector space as similar words are mapped to nearby points. In fact, Bertero et al.

demonstrated that standard out-of-the-box classifiers performed very well (~ 90% accuracy) when word2vec output is used as the feature space for anomaly detection in log files.[4] This indicates that treating log message data as regular text is highly effective in anomaly detection.

2.2 Log Parsing

Instead of using rules-based techniques and association analysis to minimize human intervention, log clustering algorithms were developed as an approach to identify patterns in messages that was scalable to complex systems. One of the earliest open-source event log clustering algorithms—Simple Logfile Clustering Tool (SCLT) considered the position of the word when generating patterns from log messages.[5] He et al. studied the effectiveness of log parsers such as SLCT and Iterative Partitioning Log Mining (IPLoM) in finding anomalies on multiple large-scale system logs.[6] A consistent improvement is shown across the board when log parsers and pre-processing is used in tandem, thus demonstrating the advantages of log parsers in log mining tasks, ie. anomaly detection.

However, some of SCLT's shortcomings, lack of wildcard prefix detection, sensitivity to shifts in word positions, and overfitting, led to the development of a new technique—LogCluster. LogCluster algorithm generalizes well to finding patterns as it is less sensitive to the exact position of words. LogCluster algorithm executes in two phases: (1) words frequently occurring in the log messages are identified; (2) patterns are identified in lines based on these words at a support threshold specified by the user.[7]

We aim to incorporate these log parsing methods in our approach, while also leveraging modern natural language processing techniques and recurrent neural network architectures to find anomalies with high precision.

3. Methodology

3.1 Data pre-processing and wrangling

The data used in this study is a Hadoop distributed file system dataset generated in a private cloud environment. The data is organized as approximately 11 million log records with multiple records belonging to a trace (block). Utilizing handcrafted rules by domain experts, each block is identified as either an anomaly or a normal log with labels from a separate file as part of this HDFS1 data set.

Each log record is structured in the form of six columns: date, time, process id, type, component, and log message. The *type* column indicates whether the log record is a warning or information. The *component* column contains different sub-systems that interact with one another: file system, packet responder, data receiver, among others. The column of interest for our methodology is *Log Message*, which provides us information about what is happening at a particular point in time.

Data pre-processing starts with extracting the six columns from the raw log files. For readability, dates are converted from UNIX epoch format to regular date format. Time of the day is converted to overall datetime so that we can compare records from different days and investigate the difference in time. For modeling purposes, one-hot encoding was applied to the type of the message, component that sent the message, and the patterns found in the message.

The valuable information within these log messages would allow us to infer the state of the system and the type of interaction that is occurring between different components.

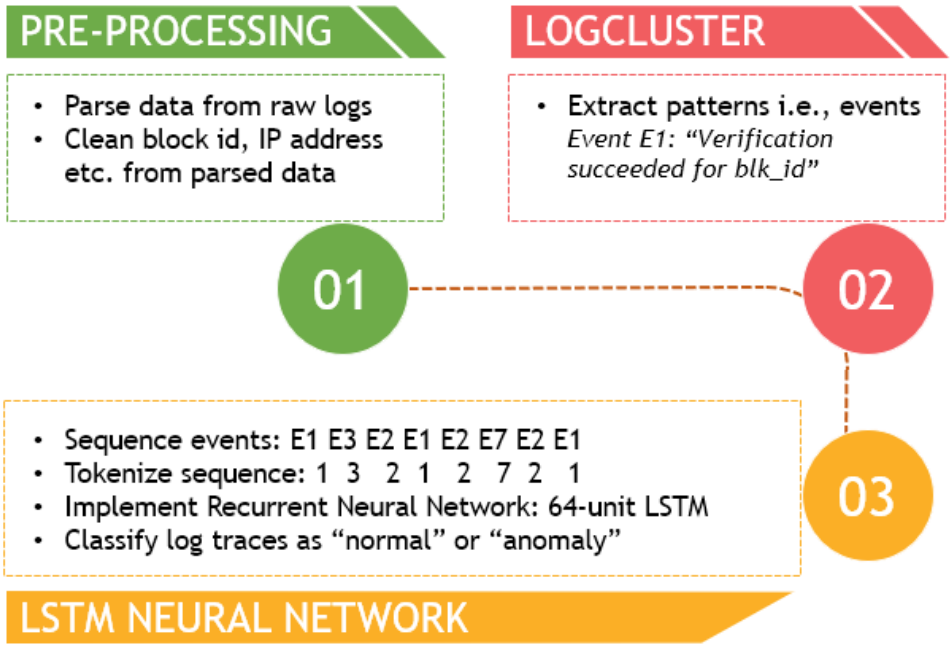

Figure 1. Workflow.

3.2 Pattern recognition using LogCluster

To infer the states, it is critical to identify the pattern or event associated with the system during the generation of the log message. However, the log messages include some distinct identifiers such as block ids, ip addresses, and mount folders. If we ignore these unique identifiers, which we consider as noise, patterns can be easily detected that would infer system behavior.

LogCluster, a perl-based tool inspired by the original SCLT, includes several novel features and data processing techniques such as custom support and filtering options. The sixth column of our data contains long messages generated by the system. LogCluster is used in this study to identify all patterns that appear in at least 0.0005% of these records. These patterns are then attached to each log record in the parsed dataset.

Example of patterns include:

- `ip:Got exception while serving blk_id to /ip`
- `Received block blk_id of size *{1,1} from /ip`
- `Verification succeeded for blk_id`

3.3 Supervised learning

In the first approach, all the columns including events are treated as individual predictors, and the data is modeled using supervised techniques such as logistic regression and random forests. In the second approach, we treated the sequence of patterns as regular text and applied natural language processing techniques to identify the anomalies based on patterns of events.

In the first approach, the type of message and component of the system which generated the log message are one-hot encoded as predictors. For example, a message type of "info" might be less associated with patterns in anomalies compared to the message type "warning". This information along with the component identifier can be helpful in finding the patterns associated with anomalies.

Patterns identified in log messages using LogCluster are all one-hot encoded for each record. By utilizing this approach, we are ignoring the sequence in which these events occurred, and instead are treating them as separate events. Parsed time column is used, with the help of some basic data transformation, to identify the sequence in which the log messages are generated during each block. The column can be treated as a quasi-identifier to model the sequence in which the events are happening in each block.

In the second approach, sequences of events are treated as natural language. Understanding the order in which events occurred can be valuable in understanding the state of the system. Modern systems are complex with frequent interaction between several components. Not all failures are considered anomalies. For example, a delay in one component might very well cause a warning in another component, but might lead to an unexpected failure in another system that is time dependent on this information. A domain expert would then classify this as an anomaly. Yet in another instance, the system failure might be because of some expected issue, which would indicate normal behaviour. In other words, identifying the sequence of events that led to the failure of a component differentiates whether the failure can be considered an anomaly or a normal one.

Natural language techniques are very useful in modeling sequences of data. If the events associated with each message in our data is aggregated to the level of traces, then we get a textual column with the list of all the events that occurred during that block. This information can be treated as regular text in sequential data form, and we can leverage the modern NLP techniques to understand these patterns.

4. Experiment

4.1 Experimental Setup

Anomaly detection is a widely researched topic with a variety of proposed solutions to applications in different domains. Some methods use unsupervised techniques such as PCA, clustering and mixture models. Other techniques include: (1) rules-based techniques; (2) supervised methods. These techniques require human assistance and expertise to infer system states from log messages and to differentiate expected and unexpected system behavior.

One of our goals is to minimize needed human intervention by standardizing the pattern recognition procedures. The dataset used for this study, HDFS, was selected because the log data comes along with labels that are carefully labelled by domain experts.[8] We intend to leverage this domain expertise and create a supervised model utilizing pattern recognition to detect the anomalies with higher precision. These labels created by experts are used for training and validation of accuracy, precision and recall of the model.

4.2 Experiment 1

For the first experiment, HDFS log data with approximately 11 million log records are pre-processed into separate columns and the pattern associated with each log message is also attached to this data. LogCluster script is used to identify these patterns for all log messages. A support of 0.0005% is used while generating these patterns. This means that the pattern must have appeared in at least 1 out of every 200,000 records. The records with no identifiable pattern are marked as ‘other’.

All the patterns are one-hot encoded to be used as individual predictors for the supervised algorithms. However, the sequence of events within the block are also added as a numerical predictor, which might slightly help with recognizing the order of event occurrences. Labels created by the domain experts are treated as response variables. In terms of performance, the base models we used for comparison included logistic regression and random forests.

These models did not perform well in identifying the anomalies, which was expected because we did not leverage the sequence of events completely. Our contributions are discussed in detail in the following section—4.3.

4.3 Experiment 2

Recurrent neural networks comprise of recurrent neurons, which receives inputs at time *t* as well as their own output from the previous time steps. This structure in essence acts as a form of memory that retains information across time. This memory is especially useful in processing sequences of data. Training of recurrent neural networks is done through backpropagation through time, which is essentially a regular backpropagation unrolled through time.

Long Short-Term Memory (LSTM) cells are designed to address the short-term memory problem in RNNs.[9] LSTM consists of an input gate that controls information to be added to long-term

state, a forget gate that controls information to be removed from long-term state, and a output gate that controls access to the long-term gate. LSTMs have been proven very successful in modeling sequential data as they capture the long-term dependencies in the data very well.

Since the HDFS log traces have many states (the longest block has almost 300 transitions over time), we used an LSTM cell in the recurrent neural network to capture the long-term relationships between these states, and identify patterns and anomalies during the process. Output from the 64-unit LSTM is sent to a dense layer with ReLu activation, and finally another dense layer with sigmoid activation is used to classify the sequence data into an anomaly or a normal record.

Shuffling and stratified sampling are used before training the models because the HDFS1 data is unbalanced with approximately 3% of the traces marked as anomalies. Keras preprocessing tokenizer is used to tokenize the sequence of patterns in each block to a sequence of numbers. The number of events in a particular trace can range from just 2 events to a maximum of 300 events. Given this irregular length of traces, zero padding is done to create the training data matrix of same length. This padded data is masked while training the model with the mask zero parameter during embedding.

5. Results

5.1 The Results of Preprocessing

The LogCluster algorithm requires data to be without distinct identifiers in order to recognize patterns in log messages. In some cases, a distinct block identifier might be used in thousands of records making it a pattern by itself, which is not expected. To avoid such scenarios, log messages are cleaned up by replacing block id, ip addresses, and mount folders by generic text with the help of Linux bash commands.

For example:

```
Receiving block blk_5792489080791696128 src: /10.251.30.6:33145 dest:
/10.251.30.6:50010
```

will be converted into

```
Receiving block blk_id src: /ip dest: /ip
```

An example bash command for removing ip address is:

```
sed 's/[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\:\{0,1\}[0-
9]\{0,6\}/ip/g' content.txt >> content_ip_removed.txt
```

And an example bash command for removing block id is:

```
sed 's/blk_\-\{0,1\}[0-9]\{1,20\}/blk_id/g' content_ip_removed.txt >>
content_block_removed.txt
```

5.2 The Results of Pattern Recognition

Cleaned up data is then passed through the LogCluster algorithm and tested with support threshold levels. We selected 0.0005% as a threshold level for pattern identification as further lowering the threshold seems to be overfitting, thus might not generalize well during validation.

An example command for using LogCluster algorithm is:

```
perl logcluster.pl --support=1000 --input=content.txt --
writewords=frequent_words.txt >> output.txt
```

Frequent words can be helpful during experiments and support threshold checks. The LogCluster algorithm automatically generates wildcard characters during pattern recognition as seen in an example below. Using a 0.0005% support level, 45 patterns are identified from the 11 million records. Records that do not meet the threshold level are marked as 'other'.

Example of patterns include:

- `ip:Got exception while serving blk_id to /ip`
- `Received block blk_id of size *{1,1} from /ip`
- `Verification succeeded for blk_id`

Output from the LogCluster algorithm also includes line numbers associated with all these patterns. These are loaded and joined to the original data using pandas so that we have a pattern associated with each log message.

5.3 The Results of Anomaly Detection

Handcrafted rules by domain experts are used to create the labels for this HDFS1 dataset. These labels indicate whether a log trace is an anomaly or a normal one. Unsupervised techniques or ensemble methods are better suited when we do not have labels. In this case, however, we wanted to examine the effectiveness of combining pattern recognition and sequential data modeling in anomaly detection, which we then used for validating our approach.

Two types of supervised learning methods are performed on this dataset. To test the effectiveness of sequential data modeling, recurrent neural networks are compared to out-of-box models that treat patterns as individual predictors. Theoretically, leveraging relationships and long-term dependencies of data in sequential form should help in improving the accuracy of anomaly detection, which is what we observed during these experiments, as well.

In the first approach, all the patterns are one-hot encoded for each row and treated as individual predictors with no dependency. A simple logistic Regression on this data scored 72.24% precision and an extremely poor 10.7% recall rate in detecting anomalies. Random forest model improved upon this by scoring 86.6% precision and 17.2% recall.

This shows that the out-of-box supervised learning models that treat patterns in log messages as individual predictors are not able to capture the relationships between them, which is evident in the poor recall of these models. Sequence within the block predictor, which we created to capture some of the information encoded in the order of events only helped to improve precision and

accuracy by a few percentage points. This shows that just the order of events is not enough, and there is much more information to extract from the dependencies between these events.

In the second approach, all the patterns for log records are grouped into a sequence of text for each log trace and treated as a regular text for modeling sequential data. All the other predictors such as sequence within block, message type, component, minutes are ignored in this test as we wanted to examine the effectiveness of this model in finding anomalies based solely on the events or patterns and the order in which they occur.

Results from the recurrent neural network were promising: the simple baseline LSTM model without any parameter tuning predicted most of the anomalies accurately. Precision of the model, which denotes how many of the predicted anomalies were in fact anomalies scored 99.7%. Recall of the model, which denotes how many of the true anomalies were we able to predict scored 99.7%. These results indicate that this is a highly effective approach – leveraging recurrent neural networks to understand relationships and long-term dependencies within the events.

Feature Extraction	**Learning Algorithm**	**Precision**	**Recall**	**Accuracy**
LogCluster	Logistic Regression	72.2%	10.8%	97.6%
LogCluster	Random Forest	86.7%	17.2%	97.8%
LogCluster	LSTM neural network	99.7%	99.7%	99.9%

6. Future work

Promising results from combining pattern detection and sequential data modeling demonstrate that information extracted from the relationships and dependencies between the events can be especially useful in detecting anomalies. One of our main goals is to minimize human intervention in the system log analysis process by leveraging modern machine learning methods. However, our methodology still includes human intervention in both log parsing and pattern detection. This might suggest that similar custom changes by users are necessary while doing system log analysis on another system with different log message structure. We aim to generalize our methodology by reducing the custom parsing elements in the process and incorporating changes that generalize well to any system.

Simplifying the existing pattern detection process and integrating Word2vec algorithm, which computes vector representations of words, and groups similar words together looks promising. Unsupervised learning methods such as PCA and clustering can also be particularly useful in the anomaly detection modeling process. Moreover, our current approach only takes events and their order of occurrence into account. Incorporating timing of the events can be extremely useful in understanding long-term dependencies within the system. We aim to continue fine-tuning our methodology with these changes to further generalize our model and minimize human intervention.

7. References

(1) https://github.com/naveen-chalasani/natural-language-processing-and-anomaly-detection

(2) W. Xu, L. Huang, A. Fox, D. Patterson, M. I. Jordan, “Detecting Large-Scale System Problems by Mining Console Logs,” *International Conference on Machine Learning*, 2010.

(3) M. Wang, L. Xu and L. Guo, "Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning," *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, 2018, pp. 140-144, doi: 10.1109/ICFSP.2018.8552075.

(4) C. Bertero, M. Roy, C. Sauvanaud and G. Tredan, "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection," *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, 2017, pp. 351-360, doi: 10.1109/ISSRE.2017.43.

(5) R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*, 2003, pp. 119-126, doi: 10.1109/IPOM.2003.1251233.

(6) P. He, J. Zhu, S. He, J. Li and M. R. Lyu, "An Evaluation Study on Log Parsing and Its Use in Log Mining," *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 654-661, doi: 10.1109/DSN.2016.66.

(7) R. Vaarandi and M. Pihelgas, “LogCluster - A data clustering and pattern mining algorithm for event logs,” *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 1–7, doi: 10.1109/CNSM.2015.7367331.

(8) https://zenodo.org/record/3227177

(9) S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, 1997, 9 (8): 1735–1780.