An Intelligent Framework for Log Anomaly Detection Based on Log Template Extraction

Lei Pan, Nanjing Institute of Technology, China*
Huichang Zhu, Jiangsu Huasheng Information Technology Co., Ltd., China

ABSTRACT

Log anomaly detection holds great significance in computer systems and network security. A large amount of log data is generated in the background of various information systems and equipment, so automated methods are required to identify abnormal behavior that may indicate security threats or system malfunctions. The traditional anomaly detection methods usually rely on manual statistical discovery, or match by regular expression which are complex and time-consuming. To prevent system failures, minimize troubleshooting time, and reduce service interruptions, a log template-based anomaly detection method has been proposed in this context. This approach leverages log template extraction, log clustering, and classification technology to timely detect abnormal events within the information system. The effectiveness of this method has been thoroughly tested and compared against traditional log anomaly detection systems. The results demonstrate improvements in log analysis depth, event recognition accuracy, and overall efficiency.

KEYWORDS

Anomaly Clustering, Anomaly Detection, Anomaly Prediction, Log Analysis, Log Template

With the proliferation of large-scale information systems such as the Internet of Things, cloud computing systems, and big data systems, there has been a significant increase in the scale of log data generated in the backend of these systems. This growth trend is characterized by diverse and rapidly generated log messages that contain crucial information about the operation of system equipment. The log message is a kind of temporal data consisting of timestamp and text messages; it records the operating status of the business in real time. Through the collection and analysis of logs, known faults can be detected and potential failures in the network can be predicted. Traditional non-automated methods involving human participation are inadequate for handling the current scenarios with massive data processing demands. Therefore, this task must be undertaken by programs and algorithms.

On the other hand, machine learning technology has a large number of applications in various fields. Chaudhary et al. (2021) uses multiple machine learning (ML) algorithms that can predict...
future values of solar radiation based on previously observed values and other environmental features measured. Similar applications exist in many other fields, such as recommendation systems (Tanwar et al., 2022), social networks (Azeez et al., 2021), financial systems (Naveed et al., 2023), and medical fields (Gupta & Gupta, 2019).

Anomaly detection plays increasingly essential roles, highlighted in various decision-making systems including machine learning, computer vision, and security. In recent years, the study of log anomaly detection has gained significant attention and a series of representative methods have been proposed. For instance, Laptev et al. (2015) introduced a framework for automatic anomaly detection of large-scale time series data using the Storm system based on the Hadoop platform. This framework enables real-time anomaly detection on millions of time series. Sun et al. (2018) introduced a potential scoring based on the ripple effect to reveal the anomaly propagation. They used the Monte Carlo tree search algorithm and a hierarchical pruning strategy to identify the cause of anomalies within a search space encompassing tens of thousands of combinations in multi-dimensional attribute spaces. Nair et al. (2015) presented a machine learning-driven method to construct a hierarchical monitoring system that can detect and diagnose service problems by employing a weighted combination of detectors at various levels to identify anomalies.

The regular expression-based method fails to address the challenges encountered in log anomaly detection, so Meng et al. (2018) proposed a data-driven framework for detecting and classifying anomalies based on device logs. A word representation method is used to represent the word combination of device logs. In order to solve the challenges caused by lack of labels, PU learning is introduced to construct device-agnostic vocabulary with partial labels to detect anomalies. A framework for modeling log streams as natural language sequences is proposed (Meng et al., 2019). By extracting semantic information hidden in log templates, it can detect sequence and quantitative log anomalies at the same time. Liu et al. (2015) applied anomaly detection to Internet-based services, used machine learning to capture unclear anomaly concepts from real data and operator tags, and automatically combined a large number of existing detectors to train classifiers to identify anomalies. Khatuya et al. (2018) proposed to carefully select features from the system log and identify abnormal signatures with known problems based on the Ridge regression. After establishing a known anomaly model, a hybrid model is proposed that can map unknown cases to known anomalies and detect anomalies in real time.

On the other hand, most methods for detecting outliers rely on manually set thresholds or assumptions about data distribution. Siffer et al. (2017) proposed a method for detecting outliers in univariate flow sequences based on extreme value theory which does not require manual setting of thresholds and makes no assumptions about the distribution. Zhang et al. (2018) applied machine learning methods to extract common patterns used to predict switch failures. The use of new features such as frequency, seasonality, and surge can effectively cope with challenges such as noise, sample imbalance, and computational overhead. Furthermore, a frequent template tree model (FT-Tree) is proposed (Zhang et al., 2017), which is capable of recognizing frequently occurring word combinations within log data and constructing corresponding templates. Leveraging these template structures, subsequent tasks, such as anomaly clustering and prediction, can be efficiently performed.

In log data, numerous alarms and abnormal information are typically present. If these anomalies occur frequently, they may indicate severe system or equipment failures. Therefore, predicting such anomalies in advance holds significant importance for large-scale information systems and network equipment. Various supervised learning-based approaches have been applied to address log anomaly predictions (Chandola et al., 2009; Du et al., 2017). However, collecting annotated data within log data presents challenges, making unsupervised learning beneficial for assisting in anomaly detection. Several methods exist, including the utilization of outlier detection techniques to identify data points distant from clusters, thereby uncovering abnormal information (Liu & Yang, 2011). Log clustering approaches based on the similarity between log events can identify frequent sequences associated with failure events and generate rules for predicting such events. Lin et al. (2016) proposed a
knowledge base-driven approach that reduces redundant log sequences and detects log anomalies through clustering.

In general, operation and maintenance personnel possess limited information about the entire system, often monitoring diary abnormalities from a localized perspective. Given the massive volume of data generated daily by diverse devices and systems, it becomes impractical for humans to manually comprehend and detect anomalies solely based on log messages. Even automated scripts developed by operations engineers may prove ineffective for large-scale systems. Moreover, log anomaly detection differs from detection tasks that primarily involve majority, regular, or evident patterns. Instead, it focuses on identifying minor, uncertain, and rare events, thereby posing unique challenges for all detection methods. In conclusion, there are still several challenging issues in current log anomaly detection research.

- **Diverse data sources.** Numerous equipment models generate logs, with each manufacturer employing its unique data format. Additionally, various factors, such as operating systems, software versions, and original equipment manufacturers (OEMs), influence log formats. Consequently, a uniform definition for data formats is lacking, posing a significant challenge in swiftly collecting and parsing data from these heterogeneous sources.

- **Unstructured data.** Despite the absence of a standardized log message format, the majority of log message formats predominantly consist of semi-structured and unstructured data. Parsing such data differs from processing structured data and cannot be accomplished through SQL statements or universal processing methods. Effectively processing these data structures and extracting analyzable information pose significant challenges.

- **Redundant data.** The system log contains a significant amount of redundant data and noise signals. Not all information within the log is relevant for anomaly detection purposes. Consequently, filtering out redundant and repetitive data becomes necessary to obtain more reliable results in anomaly detection.

- **Time-consuming labeling work.** The process of labeling data anomalies is highly time-consuming. In the context of machine learning, a substantial amount of class-labeled data is necessary for effective training. However, manual labeling is a laborious and time-intensive task.

- **Unbalanced training data.** The training data suffers from class imbalance, as the proportion of abnormal data is significantly smaller compared to the overall dataset. This imbalance poses challenges in machine learning, as the large disparity between positive and negative samples can adversely impact prediction accuracy.

To address the aforementioned challenges, this paper presents a log anomaly detection framework that leverages machine learning techniques. The framework incorporates log template extraction, log clustering, log classification, and other mining methods to construct a log anomaly detection model capable of comprehensively analyzing system anomalies and providing early warnings. The subsequent sections of this paper are organized as follows: Section 2 introduces the method and system architecture, Section 3 presents the experimental design and results of the proposed approach, and Section 4 provides a discussion on the work presented in this paper.

**RELATED WORK**

Currently, log anomaly detection involves several distinct processing steps. In this section, we review some notable practices associated with each step.

**Log Parsing**

The raw log file has a semi-structured text format and cannot be directly used for machine learning and data mining purposes. To enable effective analysis, it is essential to preprocess the raw logs by
log parsing to extract key information and eliminate redundant events and irrelevant elements from the raw logs. The traditional approach of log parsing is to process it through regular expressions which is time-consuming and challenging to implement in practice. Several effective methods for log parsing exist, such as LKE (Fu et al., 2009), LogSig (Mizutani, 2013), LogMine (Hamooni et al., 2016), and SHISO (Zhu et al., 2010). These methods use similarity-based clustering techniques to compute distances between logs and cluster them based on their similarity. Another category is frequency-based clustering, which includes approaches like LFA (Nagappan & Vouk, 2010), SLCT (Vaarandi, 2003), and LogCluster (Vaarandi & Pihelgas, 2015). These methods group log items into multiple clusters based on the frequency of their occurrence. The third category comprises heuristic methods that utilize specific data structures to parse logs into multiple templates. Representative techniques in this category include FT-Tree (Zhang et al., 2017), Drain (He et al., 2017), Spell (Du & Li, 2016), and Logstamp (Tao et al., 2022). These methods employ heuristic rules and data structures to identify common log patterns and generate templates for log parsing.

By employing these various log parsing techniques, the raw log data can be transformed into a structured format suitable for machine learning and data mining tasks, facilitating efficient analysis and knowledge extraction from log files.

Feature Extraction

Feature extraction plays a crucial role in anomaly detection as it enables the identification of relevant and informative characteristics within the data, allowing for the differentiation between normal and anomalous instances.

In various applications, such as intrusion detection (Deokar & Hazarnis, 2012) and network security monitoring (Yang et al., 2019), traditional keyword-based features have found extensive utilization where they depend on identifying the presence or absence of specific keywords or phrases within log messages, encompassing aspects such as log message frequency, message length, and the occurrence of particular error codes. By considering the frequency distribution of words in log messages, Bag-of-words features emerge as an informative approach that provides valuable insights into the content of the logs, with methods such as term frequency-inverse document frequency (TF-IDF) (Soucy & Mineau, 2005) and word embeddings (e.g., word2vec) (Bertero et al., 2017) demonstrating successful applications in extracting these features. Pattern-based features are commonly used in log anomaly detection systems that handle unstructured log messages. Regular expressions are employed to capture error messages, IP addresses, timestamps, and other patterns related to log anomalies (Feremans et al., 2020). Khatuya et al. (2018) extract statistical features from system logs, including event count, event ratio, mean inter-arrival time, mean inter-arrival distance, severity spread, and time-interval spread. These features are then transformed into a score matrix for further analysis. Guo et al. (2023) adopt TF-IDF algorithm to extract more discriminative keywords from the log text. Bhanage et al. (2023) trained a model through TF-IDF, polarity score, and Word2Vec.

Recent studies have demonstrated the effectiveness of deep learning features, which are automatically learned by deep neural networks in log anomaly detection tasks (Guo et al., 2021; Yen et al., 2019; Nedelkoski et al., 2019). Deep learning approaches excel in handling large and complex log datasets due to their ability to capture intricate patterns and relationships within log messages (Pang et al., 2021; Chalapathy & Chawla, 2019). Common deep learning features encompass embeddings obtained through techniques like autoencoders and recurrent neural networks (Yadav et al., 2020).

By employing a combination of these feature extraction techniques, anomaly detection systems can effectively represent log data and enable accurate identification of anomalous behavior within diverse application domains.

Anomaly Detection

Following log parsing and feature extraction, various machine learning methods can be employed for anomaly detection. For instance, ridge regression can be utilized to estimate the characteristics
of the abnormal score (Khatuya et al., 2018). Random forest can be applied to classify data based on the feature matrix. Clustering methods, such as LogCluster (Lin et al., 2016) and Loglens (Debnath et al., 2018), can simplify and compress raw log data, aiming to leverage the historical knowledge base to identify previous occurrences of events and detect anomalies. Liu et al. (2018) employ K-Nearest Neighbor (KNN) to identify new abnormalities in log datasets, thereby reducing computational complexity and time. Henriques et al. (2020) combine K-MEANS and XGBOOST for detection, which proves suitable for applications with massive log sources. Moreover, a wide range of supervised and unsupervised learning methods can be applied in this context. He et al. (2016) conduct a comparative analysis of six state-of-the-art log anomaly detection techniques, including Logistic Regression, Decision Tree, Support Vector Machines (SVM), Principal Component Analysis (PCA), and Invariant Mining.

At present, deep learning methods have gained popularity in log anomaly detection. Prominent examples include DeepLog (Du et al., 2017), LogRobust (Zhang et al., 2019), and SwissLog (Li et al., 2020), all of which are LSTM-based approaches that model specific log key sequences for detection purposes. Moreover, the raw log data contains abundant semantic information about the system state, making the utilization of Natural Language Processing (NLP) techniques advantageous for enhancing the accuracy of LSTM predictions (Meng et al., 2019).

METH oDS AND ARCHITECTURE

The overall framework of the log anomaly detection method presented in this paper is shown in Figure 1 and all symbols in this paper are shown in Table 1. Given a sequence of log messages, this method aims to detect whether there is any anomalous. In order to represent log messages, a collection of log messages denoted as \( DM = \{ M_1, M_2, \ldots, M_n \} \) is defined where each \( M_i \) represents an individual log message. The objective of this task involves predicting whether a new log message \( M_i \) is anomalous or not, based on a training dataset \( D = M^{N}_{i} \) comprising both normal and anomalous log sequences.
Here, it is assumed that similar log patterns exist before the same type of failure occurs. To achieve this, the method involves a sequence of steps. It starts by extracting log templates from historical log data. Next, it constructs log category libraries and anomaly libraries using clustering techniques. Finally, classification methods are applied to identify abnormal log patterns within historical data, enabling accurate prediction of future failures. The specific steps are as follows:

- **Log parsing and template extraction.** The raw log data is preprocessed and the log template is extracted based on FT-Tree (Chandola et al., 2009) to achieve log compression.
- **Log clustering.** Clustering the compressed logs through log clustering based on Levenshtein distance (Levenshtein et al., 1966). After clustering, the categories are labeled, and a log anomaly library is formed.
- **Log anomaly detection.** TF-IDF (Ramos et al., 2003) feature extraction is performed on the log and combined with the classification results after clustering. The Polynomial Bayes algorithm is used for log anomaly training and prediction.

### Data Preprocessing

As described in Figure 2, there are a large amount of irrelevant data and punctuation in the log file, which is cleaned out by preprocessing for subsequent analysis operations, including:

- **Remove the punctuation.** Remove all the punctuation that is not needed for data training, such as commas, carriage return, line feed, etc.
- **Remove the date.** Date removal is achieved through string offset, and corresponding removal rules are defined according to different device log types.
- **Remove the ip address.** Filter out the ip address in the log string by regular expression.
- **Remove other irrelevant data.** Remove the other string such as the description information at the beginning of the log file.

### Log Template Extraction Based on FT-Tree

In log anomaly detection, the presence of a vast volume of log data necessitates modeling and training prior to the detection process. However, due to the significant amount of redundant data, excessive redundancy can lead to imbalanced classes and adversely impact the efficiency of modeling. To

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>The log message</td>
</tr>
<tr>
<td>$DM$</td>
<td>The set of log messages</td>
</tr>
<tr>
<td>$a$</td>
<td>The word contained in the log message</td>
</tr>
<tr>
<td>$T$</td>
<td>The log template extracted from log message</td>
</tr>
<tr>
<td>$L$</td>
<td>The list in descending order of the word frequency using dictionary statistics</td>
</tr>
<tr>
<td>$TC$</td>
<td>The collection of log templates</td>
</tr>
<tr>
<td>$CC$</td>
<td>The collection of cluster labels</td>
</tr>
<tr>
<td>$window_size$</td>
<td>The size of the slide window</td>
</tr>
<tr>
<td>$window_gap$</td>
<td>The interval between sliding window and detected data</td>
</tr>
<tr>
<td>$f(M)$</td>
<td>The prediction of log message $M$</td>
</tr>
<tr>
<td>$y(M)$</td>
<td>The label of log message $M$</td>
</tr>
</tbody>
</table>
address this issue, our method initially undertakes log compression by employing log template extraction techniques. This process helps reduce redundancy and optimize the data for subsequent modeling and training stages.

In log analysis, the log content primarily consists of English characters, numbers, and punctuation. A straightforward approach for log template extraction involves utilizing regular expressions to replace and extract log templates. However, this method proves to be ineffective as it necessitates redefining regular expressions for each log class, limiting its applicability. In practical scenarios, two main types of template extraction methods are commonly employed: clustering-based methods and heuristic methods. The former entails calculating the similarity between log strings using measures such as Euclidean distance, Cosine similarity, or Levenshtein distance. In this study, we employ the latter approach, specifically the log template extraction method based on FT-Tree, which has been demonstrated to be accurate and efficient to extract templates from logs (Niu et al., 2023; Li & Su, 2023; Sun & Xu, 2023). The rationale behind selecting the FT-Tree model lies in its ability to maintain high accuracy while incurring low computational costs. Moreover, its capability for incremental learning makes it particularly well-suited for real-world engineering applications. This method introduces a frequent template tree (FT-Tree) model, which facilitates the rapid identification of frequent word combinations in log text, enabling the construction of templates.

Let $DM = \{M_1, M_2, ..., M_n\}$ denote a collection of system log texts, where each $M_i$ represents an individual system log text. Let $I = \{a_1, a_2, ..., a_m\}$ represent a collection of distinct words present in the system log text collection. The support of a word combination $T$ corresponds to the frequency of its occurrence, determined by the number of system log texts within $DM$ that contain $T$. Higher support indicates that $T$ appears more frequently, suggesting that it represents a template.

The FT-Tree is an extended prefix tree structure which is employed for representing log templates. The fundamental concept behind constructing the FT-Tree is to identify the longest combination of frequently occurring words, which typically corresponds to the subtype of the detailed information field in the system log (Chandola et al., 2009). By recognizing this word combination within the log message, the log template can be derived. The specific construction process of the FT-Tree is outlined in Algorithm 1.

1) During the initial pass, the entire log file is scanned and a list $L$ is generated containing words sorted in descending order of their frequency (based on dictionary statistics). Subsequently, the root node of the tree is created, with the log type serving as its label.

2) During the second pass, the log file is traversed once again to construct a reverse dictionary of words for each log. Starting with the processing of the first log message $M_1$, the initial path/
As subsequent log messages, such as $M_2$, are processed, if their sorted word list shares a common prefix with an existing path/branch in the FT-Tree, a new branch is created to represent a subtree. This process is repeated for each log message, resulting in the complete construction of the FT-Tree.

3) Finally, the FT-Tree undergoes a pruning process to ensure that the degree constraints of the nodes are met. The underlying rationale is that each log type should have a limited number of subtypes, while there should be a sufficient number of distinct system log messages associated with each subtype. Thus, if a node in the FT-Tree possesses an excessive number of child nodes, all of its children are removed, effectively transforming the node into a leaf node. This process is performed to maintain the desired structure of the FT-Tree. Figure 3 illustrates an example of the constructed FT-Tree.

Based on the aforementioned construction approach, the FT-Tree is constructed independently for each log type, resulting in an FT-Tree forest. However, constructing an FT-Tree forest can be a time-consuming process, particularly when dealing with large volumes of system log data. To address this issue, we can persist the generated FT-Tree forest from historical logs and save the extracted template information to a file. This allows for efficient retrieval and utilization of the template information without the need to rebuild the FT-Tree forest each time. By storing the FT-Tree forest, the construction time can be significantly reduced, facilitating faster processing of subsequent log data. In the event of system version upgrades or other modifications, new log types may emerge, and additional data may be introduced within existing log types, necessitating the generation of new log
templates. To address these scenarios, the FT-Tree can be leveraged to facilitate incremental template construction. Specifically, when encountering new log types, a new FT-Tree needs to be rebuilt and persisted. As for new data within existing log types, the data is initially sorted based on the word list, and the sorted list is subsequently inserted into the corresponding FT-Tree for that log type. The process primarily involves the following steps:

1) The new log message is read and the tree structure is traversed. The previous word encountered, denoted as wordprev, and the corresponding previous node, denoted as nodeprev, are recorded based on the log word frequency list.
2) Beginning with the first word, the existence of a node in the tree is searched. If the node exists, the search continues for the next word. The current word is assigned to the previous wordprev, and the current node pointer is assigned to nodeprev.
3) If all words in the log word list are found in the FT-Tree, the function returns Null, indicating that there is no need to add new template nodes as the template for the current log already exists.
4) If a word in the log message is not found in the FT-Tree, a new node is added to the previous node nodeprev, creating a new branch where the new node becomes its child node.
5) Repeat the aforementioned process iteratively until all logs have been traversed. This ensures that each log message is processed, and if any new words are encountered, they are added to the corresponding nodes in the FT-Tree.

Log Clustering Based on Levenshtein Distance

Levenshtein distance, also known as Minimum Edit Distance (MED), was introduced by Vladimir Levenshtein in 1966. It has found applications in various fields, including information theory, linguistics, and computer science. Levenshtein distance is commonly utilized to measure the similarity between two sequences, representing the minimum number of single-character editing operations required to transform one word into another. The available editing operations include insertion, deletion, and substitution. Compared with other similarities, Levenshtein distance has relatively high accuracy and low computational complexity, which is more suitable for code implementation in large-scale text analysis system. The Levenshtein distance of the two strings \( a \) and \( b \) is defined as \( \text{lev}_{a,b}(i,j) \) that is shown below:

\[
\text{lev}_{a,b}(i,j) = \begin{cases} 
  i, & j = 0 \\
  j, & i = 0 \\
  \min \left\{ \text{lev}_{a,b}(i-1,j) + 1, \text{lev}_{a,b}(i,j-1) + 1, \text{lev}_{a,b}(i-1,j-1) + 1, a_i \neq b_j \right\} & \text{otherwise}
\end{cases}
\]

In the context of Levenshtein distance calculation, \( \text{lev}_{a,b}(i,j) \) represents the distance between the first \( i \) characters in string \( a \) and the first \( j \) characters in string \( b \). For clarity, in this explanation, \( i \) and \( j \) can be considered as the lengths of strings \( a \) and \( b \), respectively. The character subscripts in the string start from 1 (in actual calculations, an additional 0 is added before the string), so the
Levenshtein distance can be represented as $lev_{a,b}(i,j)$ when $i = |a|, j = |b|$. Here, $|a|$ and $|b|$ denote the lengths of strings $a$ and $b$, respectively.

If one of the strings $a$ or $b$ is an empty string, then only a single-character editing operation is required to transform it into the other string. When $i \neq 0$ & $j \neq 0$, the Levenshtein distance is calculated as the minimum of the following three cases:

1) $lev_{a,b}(i-1,j) + 1$ represents the distance obtained by deleting the $i$th character of $a$.
2) $lev_{a,b}(i,j-1) + 1$ represents the distance obtained by inserting the $j$th character of $b$ into $a$.
3) $lev_{a,b}(i-1,j-1) + 1$ represents the distance obtained by replacing the $i$-th character of $a$ with the $j$-th character of $b$ if $a_i \neq b_j$.

For log files of each type, clustering is performed separately using the Levenshtein distance based on the extracted log templates. The resulting clusters are then associated with the corresponding alarm file. The classification of each line in the log file is based on the Levenshtein distance of the string. The clustering algorithm is applied to all the extracted template collections (TC), and the program automatically outputs the labels of each cluster into a collection (CC). Algorithm 3 provides the pseudocode for this algorithm.
First, the log category collection is initialized as empty, with the format of the category collection being [index, log template content, result set sorted by time]. Next, the clustering parameters are set, which include: 1) The similarity threshold, denoting the level at which strings with a similarity above the threshold are grouped together. The default value is set to 0.8. 2) The time interval for logs which represents the number of logs recorded during the interval is set. If a certain type of log in the clustering result has already been recorded during this time period, similar logs will not be recorded again. The default interval is set to 15 minutes.

Each log is compared for similarity with the log templates in the category collection. If the Levenshtein distance exceeds the predefined threshold, the log is assigned to the corresponding category and added to the clustering results. If the Levenshtein distance is lower than the threshold for all templates, a new category is defined, and a new record is added to the category collection. These steps are performed separately for each log type, resulting in the final clustering result sets for all categories.

Finally, after the clustering process, the categories are labeled, including the normal category and the anomaly category, in preparation for log anomaly classification. Data labeling can be automated using corresponding keywords such as “FATAL, ERROR, WARNING, EXCEPTION,” etc. In practical applications, the TextRank keyword extraction algorithm (Durant & Smith, 2006) can be employed to extract and record descriptive information for the normal category, which can facilitate data visualization.

Algorithm 2. FT-tree update

Algorithm 2. FT-Tree update

<table>
<thead>
<tr>
<th>Input: DM: log file, T: template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: T: updated template</td>
</tr>
<tr>
<td>function UPDATE_TREE(DM)</td>
</tr>
<tr>
<td>L ← NULL</td>
</tr>
<tr>
<td>for text M in DM do</td>
</tr>
<tr>
<td>words ← WordTokenize(M)</td>
</tr>
<tr>
<td>append words to L</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>L ← Sort(L)</td>
</tr>
<tr>
<td>prev_node ← NULL</td>
</tr>
<tr>
<td>prev_word ← NULL</td>
</tr>
<tr>
<td>for W in L do</td>
</tr>
<tr>
<td>for node in T do</td>
</tr>
<tr>
<td>prev_node ← node</td>
</tr>
<tr>
<td>prev_word ← node.word</td>
</tr>
<tr>
<td>if prev_word == W then</td>
</tr>
<tr>
<td>prev_node ← NULL</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>if prev_word != NULL then</td>
</tr>
<tr>
<td>InsertNode(W)</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>return T</td>
</tr>
<tr>
<td>end function</td>
</tr>
</tbody>
</table>
Log Anomaly Detection Based on TF-IDF

Log anomaly detection is essentially a time series prediction problem. A sliding window method is used to transform it into a supervised learning problem. As mentioned earlier, the initial raw log messages consist of extensive sections of unstructured text, which are subsequently parsed into structured
data through template extraction. By compressing a significant amount of redundant messages and reducing the observation samples of normal categories, the imbalance between different classes can be mitigated. Additionally, similar categories of log data are grouped together and the overall log set is divided into distinct analysis units, facilitating data labeling.

Based on the log clustering results and the characteristics of the log dataset, each log message is transformed into a numerical vector to enable model training. The extracted features encompass both numeric and categorical attributes derived from the raw log messages. In this study, the inverse document frequency index (TF-IDF) feature extraction method is employed. The reason for choosing TF-IDF is that it reduces sparsity and maintains a simple representation while balancing the impact of term frequency. Additionally, in log text analysis, unlike other NLP scenarios, such as news sentiment analysis, a significant portion of text in log data is repetitive. Therefore, using TF-IDF is more appropriate in this context. The TF-IDF is defined as follows:

$$TF-IDF = TF_w \times IDF_w = \frac{N_w}{N} \times \log \frac{Y}{Y_w + 1}$$

where $N_w$ represents the number of occurrences of a specific word $w$ in a given text, while $N$ corresponds to the total number of words in that text. The variable $Y$ represents the total number of documents present in the corpus, and $Y_w$ signifies the number of documents containing the word $w$. To avoid scenarios where the denominator becomes zero due to a word not appearing in any document, one is added to the denominator. The TF-IDF definition demonstrates that common words within documents tend to have limited significance, whereas infrequently occurring words may carry more representational value. If a word appears frequently in one document but rarely in others, it is considered to possess classification power and can be utilized as a distinguishing feature. Consequently, TF-IDF tends to filter out common words while retaining essential keywords.

During the feature extraction process, a sliding window approach is employed to segment the words in the log messages. Then TF-IDF is utilized to transform the words associated with each log type into a shared word vector space. In this particular context, two windows are constructed: the input sliding window and the label sliding window. The window size parameter denoted as $window_size$ defaults to 50, while the interval between the two windows referred to as $window_gap$ defaults to 20. The sliding window method is depicted in Figure 4, illustrating the window arrangement.

All logs within the first window serve as input data for further processing. Following preprocessing steps, TF-IDF features are extracted. At this time, if an anomalous log is detected within the second window corresponding to the first window (determined based on the clustering result and whether it belongs to an anomaly category), the corresponding label is assigned to the feature data associated with the first window. Conversely, if the log is deemed normal, the normal label is assigned. For model training, the polynomial Bayes algorithm, specifically MultinomialNB [44], is employed. 

MultinomialNB is an extension of the naive Bayes algorithm designed for polynomial distributed data, and it is a well-established variant of the classic naive Bayes algorithm. In text classification tasks, data is often represented using word vectors, such as TF-IDF. The algorithm assumes that the prior probability of each feature follows a polynomial distribution, which is defined as follows:

$$P(X_j = x_j|Y = C_k) = \frac{x_j + \lambda}{m_k + n\lambda}$$

where the probability $P$ represents the conditional probability of the $l$-th value of the $j$-th dimensional feature belonging to category $k$. The value $m_k$ denotes the number of samples in the
training dataset that belong to category $k$. To prevent zero probabilities during subsequent calculations and account for non-existent features in the training samples, a constant $\lambda$ greater than or equal to 0 is introduced. When $\lambda = 1$, Laplace smoothing is applied, and when $\lambda < 1$, Lidstone smoothing is employed. The purpose of smoothing is to ensure that all features have non-zero probabilities, even if they are not observed in the training data.

$$y(M_i) = \arg\max_{c_i} \frac{f(M_i)}{\prod_{j=1}^{n} P(X_j = x_j | Y = C_k)}$$

To predict the label of a log sequence $M_i$, an algorithm is utilized, which employs a binary classifier denoted as $c$. This classifier is trained on the feature vectors $f(M_i | M_i \in DM) \in T$, where $T$ represents the training set. The label $y(M_i)$ indicates whether the log sequence is anomalous or not. Specifically, if $y(M_i) = 0$, it signifies that $M_i$ corresponds to a normal event, while $y(M_i) = 1$ indicates an anomalous event. The binary classifier $c$ aims to separate the log sequences into these two categories based on their respective feature representations.

Using the trained model, out-of-sample data can be evaluated to determine whether it corresponds to an anomalous log or a log belonging to a category that has not been encountered before. Once the model is trained, it is saved for future use. During the system’s runtime, anomaly predictions can be generated at regular intervals (typically every 15 minutes). The accuracy and effectiveness of these predictions are assessed by comparing the number of events classified as anomalies with the number of events assigned to the anomaly cluster. This evaluation serves to validate the reliability of the results obtained in this stage.

**EXPERIMENTS**

To evaluate the performance of our method, we established a dedicated experimental environment. The experimental setup consisted of a PC with an Intel(R) Core (TM) i7-5930K CPU running at 3.50GHz and 32GB of memory. The operating system used was Microsoft Windows 10 X64. The programming environments employed were Python 3.6 and Java 1.8.

The datasets utilized in our experiments were sourced from an internal information system of a group company. Specifically, we collected raw log data from four distinct types of systems: Windows IIS service, Linux OS syslog service, H3C switch, and power dedicated equipment. Table 2 provides a summary of the dataset, which comprises a total of 507,196 logs. These logs were arranged in a predetermined format, with the event problem date, the hostname of the device generating the event, and the event delivery process included in the log sequence.

Our experimental workflow began by constructing FT-Tree log templates based on the raw log data. Subsequently, log clustering and category labeling were performed using the constructed templates. Finally, the log data was utilized for training the anomaly detection model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Data Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>72270</td>
<td>Logs from Windows IIS service</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>127561</td>
<td>Logs from Linux OS syslog service</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>179182</td>
<td>Logs from H3C switch</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>128183</td>
<td>Logs from power dedicated equipment</td>
</tr>
</tbody>
</table>
Evaluation

To assess the effectiveness of the proposed method, we employed various metrics to evaluate the performance of log anomaly detection. By comparing the predicted results with the ground truth, we could assess the system’s ability to accurately identify anomalous logs. The adoption of these metrics ensured a comprehensive evaluation of our system’s capabilities in detecting log anomalies. The results obtained from this evaluation were crucial in validating the effectiveness and reliability of our proposed approach.

Cluster evaluation is mainly to evaluate the clustering quality and stability of the clustering algorithm on the data set. The assessment mainly includes estimating the clustering trend, determining the number of clusters in the data set, and determining the clustering quality. The Silhouette Coefficient is used as the evaluation index to evaluate the cohesion and separation of the results. Its definition is as follows:

\[
SC = \frac{1}{N} \sum_{i=1}^{N} \frac{b - a}{\max(b - a)}
\]

where \(N\) is the number of clusters, \(a\) is the average distance between the sample and the sample in the cluster, and \(b\) is the average distance between the sample and other cluster samples. The range of these two indicators is between \([-1, 1]\). The closer the result is to 1, the better the clustering effect.

The evaluation of anomaly prediction involves assessing the performance of the model in identifying and detecting anomalies in the log data. To measure the effectiveness of the prediction model, we utilized two commonly used metrics: accuracy and recall.

\[
\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}
\]

\[
\text{recall} = \frac{TP}{TP + FN}
\]

\(TP\) is the proprietary sample, and the correct number is judged. \(FP\) is the number of errors in a sample that is determined as a positive. \(TN\) is a negative sample to determine the correct number. \(FN\) is a negative sample to determine the number of errors. A higher accuracy value indicates a higher level of correct predictions by the model, indicating its ability to accurately classify logs as normal or anomalous. A higher recall value indicates a higher ability of the model to detect anomalies and minimize false negatives. When evaluating the prediction effect, within 15 minutes we restrict the effective time of the prediction is \(T1\), if the time of the predicted incident occurs \(T2\) to meet \(T1 < T2 < T1 + 30\text{min}\), the prediction is valid, otherwise it is invalid.

Computational Time

Figure 5 depicts the time consumption of different processing stages and algorithms within the framework. The computational time is categorized into three main parts: template extraction, log clustering, and log anomaly detection. Specifically, Figure 5(a) presents the time consumption for log parsing, clustering, model training, and issuing predictions for the entire dataset.

Among these stages, template extraction demonstrates the longest computational time, taking 42.6 seconds for processing 100,000 logs. On the other hand, log training exhibits the shortest time of 4.91 seconds. This can be observed in Figure 5(a). The extended duration for template extraction can be attributed to its comprehensive analysis and extraction process. Conversely, log training requires a relatively shorter time due to the reduced log volume after template extraction. Once the
template extraction stage is completed, subsequent clustering and detection algorithms benefit from the compressed log data, resulting in more efficient processing. The primary time-consuming task remains the preprocessing of historical data. However, it is important to note that historical data is only modeled once, and subsequent template extraction is performed incrementally. Consequently, the computational time in the early stage has minimal impact on the incremental update process.

In Figure 5(b), the computational time for different clustering methods on 100,000 logs is illustrated. The Levenshtein-FTtree clustering method demonstrates a time consumption of 25.99 seconds, whereas the Levenshtein-sampling method requires 43.31 seconds, and the Levenshtein-original method consumes 59.32 seconds. This comparison highlights the impact of template extraction on reducing the time consumption in log clustering and classification. Moreover, it can be observed that different clustering methods have a relatively minor influence on the overall computational time, while the template extraction method has a more substantial impact. The FT-Tree template compression technique significantly reduces the log size, resulting in decreased computational time for the Levenshtein-FTtree clustering algorithm compared to the uncompressed Levenshtein-original clustering algorithm. Similarly, the Levenshtein-sampling clustering algorithm, which incorporates sampling compression, also exhibits reduced computational time. These findings emphasize the efficiency and effectiveness of template extraction in reducing computational time for log clustering and anomaly detection.

Figure 5. Algorithm performance
Figure 5(c) presents the computational time of the training method. The TFIDF-NB-FTtree method requires 4.91 seconds to process 100,000 logs, while TFIDF-LR takes 52.08 seconds and TFIDF-NB consumes 64.08 seconds. Notably, due to the utilization of template extraction, the computational time of the detection algorithm TFIDF-NB-FTtree in this framework remains shorter than that of the algorithm without extraction. For comparison, the algorithms used are the uncompressed naive Bayes classifier (NaiveBayes) and the TF-IDF-based classifier (TFIDF). The results highlight the advantage of incorporating template extraction in reducing the computational time of the TFIDF-NB-FTtree method.

Log Clustering Results

After performing template extraction, the clustering process on more than 500,000 log data of four types was completed in approximately 150 seconds. Comparing the results of clustering using template extraction (Levenshtein-FTtree) with direct clustering without template extraction (Levenshtein-original) and clustering with sampling (Levenshtein-sampling), it is evident from Figure 5 that the performance of Levenshtein-FTtree (0.8827, 0.8812, 0.8918, 0.8961) on the four datasets is superior to that of Levenshtein-original (0.8123, 0.8166, 0.8265, 0.8289) and Levenshtein-sampling (0.8023, 0.8264, 0.8378, 0.8244).

The higher clustering coefficient of the Levenshtein-FTtree method proposed in this paper indicates that the clustering approach has achieved a certain level of effectiveness. Furthermore, the log data after template extraction has been significantly compressed, leading to improved efficiency. Thanks to the effectiveness of the FT-Tree template extraction, a substantial amount of noisy data with interfering information is filtered out. Upon completion of the clustering process, there is no need to manually inspect individual logs; instead, one can simply review the template log within each category along with its corresponding description to gain insights into the specific log details.

Log Anomaly Detection Results

In the prediction of log anomalies, the word vector features are extracted by TF-IDF, and then processed by the sliding window to form training data, and finally trained and predicted through the polynomial Bayes algorithm. The proposed method (TFIDF-NB-FTtree) is compared with the logistic regression algorithm without template compression (TFIDF-LR) and the NaiveBayes algorithm (TFIDF-NB) (Durant & Smith, 2006). To evaluate the performance of the models, the dataset is divided randomly into 10 parts for cross validation. The classification verification was performed on the log dataset, and the results are shown in Table 3.

Regarding the aspect of the detection model, we can see that the anomaly detection performance of TFIDF-FTtree (Accuracy = 0.8292, Recall = 0.7968) are far higher than that TFIDF-LR (Accuracy = 0.6827, Recall = 0.7227), and TFIDF-NB (Accuracy = 0.7127, Recall = 0.7356) on DataSet1. In the case of DataSe2, the performance of TFIDF-FTtree (Accuracy= 0.7966, Recall = 0.7644), TFIDF-LR (Accuracy =0.7156, Recall = 0.7756), and TFIDF-NB (Accuracy = 0.6934, Recall = 0.7666) exhibit similar levels of performance. In the case of DataSe3, TFIDF-FTtree (Accuracy = 0.8315, Recall = 0.8292)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TFIDF-LR</th>
<th></th>
<th>TFIDF-NB</th>
<th></th>
<th>TFIDF-FTtree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Accuracy</td>
<td>Recall</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>0.6827</td>
<td>0.7227</td>
<td>0.7127</td>
<td>0.7356</td>
<td>0.8292</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>0.7156</td>
<td>0.7756</td>
<td>0.6934</td>
<td>0.7666</td>
<td>0.7966</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>0.6918</td>
<td>0.7918</td>
<td>0.7012</td>
<td>0.8105</td>
<td>0.8315</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>0.7028</td>
<td>0.7936</td>
<td>0.6986</td>
<td>0.7821</td>
<td>0.8328</td>
</tr>
</tbody>
</table>
0.7986) outperforms TFIDF-LR (Accuracy = 0.6918, Recall = 0.7918) and TFIDF-NB (Accuracy = 0.7012, Recall = 0.8105). In the case of DataSe4, the performance of TFIDF-FTtree (Accuracy = 0.8328, Recall = 0.8012) demonstrates superior performance compared to TFIDF-LR (Accuracy = 0.7028, Recall = 0.7936) and TFIDF-NB (Accuracy = 0.6986, Recall = 0.7821).

The way of sampling method has minimal impact on the anomaly detection results. Among the three classifiers compared, TFIDF-FTtree outperforms TFIDF-LR and TFIDF-NB. This superiority can be attributed to the effectiveness of template extraction in reducing noise present in the log text. By filtering out irrelevant information, template extraction not only enhances data processing speed but also improves the accuracy of anomaly prediction. However, it should be noted that the prediction accuracy does not exceed 90%. This limitation can be attributed to the assumption of independence between features in the Naive Bayes algorithm, whereas a certain degree of correlation exists among log events in the log sequence.

The obtained results confirm the efficacy of the Naive Bayes model, which is known for its simplicity, fast training, and prompt responses. The average accuracy achieved by our predictions on the four datasets is 82.25%, with an average recall rate of 79.03%. In comparison to the other two methods, our method exhibits higher accuracy and relatively consistent prediction results across the four log datasets, albeit with a slightly lower recall rate. This can be attributed to the nature of anomalies, which are closely associated with log templates and clustering. The model can effectively predict anomalies that have a strong connection to their corresponding logs and have been observed in the training set. However, it may struggle to predict anomalies with low relevance to other logs or those that have not been encountered in the training set, resulting in a slightly lower recall rate. Overall, these findings underscore the significant impact of utilizing TF-IDF embedding on log data, as it allows for the condensation of each logfile into a single representation while preserving sufficient information for accurate classification.

Although advanced machine learning techniques like deep learning were not employed, the accuracy of the Naive-Bayes based approach remains commendable. In comparison with deep learning methods, it holds certain advantages, such as faster training speed and quicker decision efficiency. Its performance on large-scale datasets further validates this advantage. Additionally, we adopted log template compression and semantic vector extraction to significantly reduce log redundancy, resulting in substantial reduction of training data volume.

As for threats to validation, one concern lies in the representativeness of the dataset. Our dataset was sourced from various devices and systems, encompassing both hardware-generated and software-generated logs. However, it’s worth noting that the lengths of these logs vary. We acknowledge that different software systems may have diverse execution times for distinct tasks. As a result, the method’s generalization ability may be impacted, necessitating parameter adjustments based on task and system types.

CONCLUSION AND DISCUSSION

This paper presents a template extraction-based log anomaly detection method designed for handling massive log data, enabling real-time detection and prediction of anomaly events and information within the system. Experimental results demonstrate that the method possesses effective capabilities for processing and identifying large-scale log anomalies. Moreover, the proposed algorithms exhibit lower complexity and faster execution efficiency. By utilizing this method, continuous failures can be avoided, reducing business interruption during incidents, and enhancing the overall emergency response capability of the system. However, it is worth noting that the preprocessing of historical data and template training still require significant time investment. In future research, we aim to explore rapid processing techniques for handling massive log data, leverage deep learning and other advanced methods, establish scientific analysis models, and enhance the design of distributed computing architectures. Additionally, we plan to investigate self-supervised learning frameworks.
to handle unlabeled datasets comprising both normal and anomalous log sequences, with the goal of further improving the depth of log analysis and the accuracy of anomaly detection.

ACKNOWLEDGMENT

This research was funded by the Research Foundation of Nanjing Institute of Technology under grant No. CKJB202107, No.3612407222468 and No.3612407222302.
REFERENCES


Lei Pan received a BS degree in Electronic Information Engineering from Nanjing University of Posts and Telecommunications in 2008 and a PhD degree in Computer Science from Nanjing University in 2014. He is currently working in the School of Computer Engineering, Nanjing Institute of Technology. His research interests include complex network analysis, big data application, etc.

Huichang Zhu received a BS degree and a MS degree from Nanjing University of Posts and Telecommunications in 2008 and 2011, respectively. He is currently working in Jiangsu Huasheng Information Technology Co., Ltd. His research interests include cloud computing, big data application, etc.

Lei Pan received the B.S. degree in Electronic Information Engineering from Nanjing University of Posts and Telecommunications in 2008 and the ph.D degree in Computer science from Nanjing University in 2014. He is currently working in the School of Computer Engineering, Nanjing Institute of Technology. His research interests include Complex network analysis, Big data application, etc.

Huichang Zhu received the B.S. degree, M.S. degree from Nanjing University of Posts and Telecommunications in 2008, 2011 respectively. He is currently working in Jiangsu Huasheng Information Technology Co., Ltd. His research interests include Cloud Computing, Big data application, etc.