IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 25 June 2024, accepted 3 July 2024, date of publication 11 July 2024, date of current version 22 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3426959

RESEARCH ARTICLE

The Usage of Template Mining in Log File Classification

PÉTER MARJAI[1] AND ATTILA KISS[1,2]
[1]Department of Information Systems, Eötvös Loránd University (ELTE), 1117 Budapest, Hungary
[2]Department of Informatics, J. Selye University, 945 01 Komárno, Slovakia

Corresponding author: Péter Marjai (g7tzap@inf.elte.hu)

ABSTRACT The continuing growth of large-scale and complex software systems has led to growing interest in examining the possibilities of using the log files that were created during the runtime of the software. These files can be used for various purposes like error prediction, performance evaluation, learning of usage patterns, improving reliability, and so on. With software systems continuously becoming more and more complicated, the distinction of log files that were generated by different components of the software becomes a new task. The classification of log files is important for several reasons like resource optimization, compliance and auditing, automation and analysis, or understanding the general system health. By classifying log files, organizations can better understand the health and performance of their systems. They can identify patterns, potential security threats, anomalies, errors, and malicious behaviors and storage can also be optimized. In the log files, each line represents a specific event that has occurred. Such events can be identified with the use of template miners that assign a unique ID for each event. In our paper, instead of using the full-sized log files, we change each line to its corresponding event ID and use the resulting smaller file for classification purposes. We use numerous classifying algorithms like Random Forest, K-NN, Ada Boost Classifier, and Decision Tree to assign the files to groups corresponding to their origin types. 75% of the data is used for learning purposes while the remaining 25% is used for testing. We conduct numerous different experiments to verify the effectiveness of our method like evaluating the precision, recall, f-score, and accuracy values and measuring the time it takes to classify the files. Our results yielded that while there is a small fallback in the case of the performance of some of the investigated methods used with the proposed algorithm, it takes significantly less time to classify the log files, which can be profitable, especially in the case of large collections of log files.

INDEX TERMS Document classification, log file, template miner.

I. INTRODUCTION

All software is expected to create log files during its run, which can be used to monitor the states it went through, the operations performed, and other fundamental information. Log files are created via print statements that were inserted into the source code via the developers to save necessary information. They have many areas of use such as data mining and analysis [1], [2], understanding user behavior [3], [4], [5], security checking [6], performance monitoring [7], [8], reliability engineering [9], and anomaly detection [10], [11], [12].

One of the most common types of logs is weblogs. They record which pages were visited by which users, what were their requests, what were the response times and so on. In [1] it is shown how one's supposed to configure a web server to gain useful log files that can be used for analytic purposes. The main advantage of the analyzation is that it can be used by an e-commerce site operator to acquire the global overview of the feedback immediately and an extensive understanding of events occurring on the e-commerce site as well as on social media. The only drawback of this approach is that it does not take information from the physical shopping floor

The associate editor coordinating the review of this manuscript and approving it for publication was Inês Domingues.

into account. The authors of [2] investigate how clustering algorithms such as K-means and DBSCAN can be used to cluster the dataset and retrieve practical information from web server log files. They found that DBSCAN has a better performance in clustering such files than K-means.

Based on information gathered from log files, we can get a more comprehensive picture of user behavior patterns. In [3] the procedure of data cleaning, user recognition, and finally session identification is proposed. With the use of the different sessions of a unique user, they analyze the behavior pattern of that user. While this information is advantageous for business intelligence, keeping it up to date could be time and resource-consuming. This paper also does not consider if the behavior patterns are malicious. The authors of [4] utilize log files about the user's daily activities and e-mailing habitats to be used as input of anomaly detection algorithms so that malevolent insider activities can be detected. This is typically a hard task for network operators since insiders possess extensive knowledge about the system of the organization. The proposed solution remarkably lessens their workload. The limitation of the method is that the models were trained by different datasets which could result in conflicting detections. If the detection results were integrated it may be possible to achieve better detections. The log data available on smartphones (app usage, music consumption, etc.) is used in combination with machine learning algorithms to anticipate the personality type of the owner of the smartphone [5]. Although most of the personality traits could be predicted by the proposed algorithm, some could not be predicted at all. Enhancing performance may require the employment of additional sensors and the integration of their data. The paper also points out the advantages and dangers of collecting and modeling behavioral data.

The authors of [6] provide an extensive summary of recent directions, developments, and possible future orientations in the case of log security analysis with the examination and summarization of 34 different approaches from recent papers. It is stated that although machine learning techniques are most commonly used for log taxonomy, new methods are being proposed that are based on different principles like process mining, event correlation, statistical analysis, and so on. The current problem is also stated: while analyses can alert administrators, they can not quickly address the attack.

With the swift expansion of information technology, the appearance of large-scale systems has increased greatly. These systems are also becoming faster and faster. To monitor such systems by hand has become impossible so software-aided performance monitoring has become a decisive task. In [7], the performance of a new method that is capable of utilizing log files that arrive with a very high frequency is evaluated. They came to the conclusion that with the utilization of log files, higher precision values can be achieved than without them. The greatest improvement is that log files enable the review of effects that were previously unobservable. However, the single use of the proposed method may not be enough and an auxiliary system might be required. Cloud services are also becoming increasingly popular. Due to the troubleshooting complexity, of such services, a new approach called megatables is proposed in [8] that outputs millibottleneck predictions and supporting visualizations based on the automatic analysis of log data. Unlike previous research in this field, their algorithm not only extracts the performance-related data from the log files but also interprets and analyses the performance patterns.

Automated log analysis can be used for reliability engineering purposes. In [9] a survey is provided regarding this subject. They introduce and detail the four main steps in automated log analysis. First, they describe how and when logging should be done and what kind of information is necessary to be included in the log file. The format of the log messages is also investigated. The second step is the compression of the log files, which addresses the significant challenge of storing logs from large-scale systems that run continuously. The main compression approaches considered include bucket-based, statistics-based, and dictionary-based methods. The next step is the parsing of the logs. Log mining tools take structured data as their input however, log lines are typically unstructured or semi-structured. To transform the log lines, previously used methods relied on regular expressions or ad-hoc scripts. This paper introduces 15 modern automated log parsers: 10 run offline, processing historical data, and 5 perform online, operating on logs sequentially for real-time services. The last step is log mining which employs machine learning, statistics, and data mining to analyze large amounts of data and to find meaningful patterns. The general workflow of log mining is proposed. Additionally, the paper incorporates a classification and brief introduction of log anomaly detection algorithms.

The prediction of upcoming anomalies and errors became a vital task with the appearance of large-scale networks and log files produced by them. The authors of [10] provide a complete review of the current state of anomaly detection algorithms. They highlight the limitation that the datasets used in the investigated papers are usually outdated, and not properly labeled. It is also stated that, while it is claimed that deep learning methods outperform traditional machine learning and data mining techniques, the necessity for more extensive research on the optimization of hyperparameters is emphasized. Many researchers are taking advantage of deep learning techniques, to make the detections. A complete survey of such methods is supplied in [11]. First, the methodology of log anomaly detection is explained in the same way as in [9]. This is followed by a brief overview of five advanced deep learning methods: Long Short-Term Memory (LSTM), Transformer, Autoencoder, attentional Bidirectional Long Short-Term Memory (BiLSTM), and Convolutional Neural Network (CNN). These new solutions along with six state-of-the-art methods are evaluated in terms of efficiency, robustness, and accuracy. A significant contribution of this paper is that they prove that including the log's semantics improves a model's robustness against noises, which results in the improvement of their overall reliability in practical utilizations. A new method that uses autoencoders is proposed in [12]. The main advantage is that this approach is not limited by the underlying log structure. Furthermore, the presence of anomalies is not required during the training phase. One of the primary limitations of this study is the requirement for normative logs. The assembly of these logs is a difficult task that is greatly dependent on the characteristics of the particular system generating them. In real-life industrial scenarios, log data can come from various domains. Most deep learning models are typically trained on data from a single domain which can lead to poor generalization performance on multi-domain data. To overcome this problem, the authors of [13] propose a new framework that is based on two main steps. First, the model is pre-trained on logs from a particular source domain to capture domain-specific patterns and features. Then, the obtained knowledge is transported to the target domain with the use of shared parameters. This allows the model to adjust to the new domain while retaining the learned information from the source domain.

Log files are usually unstructured since developers can write free text messages into print statements. In the past few years, various algorithms have been proposed to recover message types from unstructured logs. Each log line can be grouped under an event template. The templates are made from two parts, the constant part, which is the same at any occurrence, and the parameter part, which might differ. A new algorithm called Longest Common Subsequent is proposed in [14]. The log lines are turned into a sequence of tokens that are identified by unique IDs. After the conversion, backtracking is used to recover the event templates. The templates are retrieved based on the assumption that log lines with equivalent length words on the same positions belong to the same template in [15]. The authors of [16] propose a new tree-based template mining technique. Apart from the rate of matching tokens in two distinct log entries, this algorithm also contemplates the tokens at which two log entries disagree.

A. LOG CLASSIFICATION

Document classification is fundamental for managing and ordering large collections of documents, such as contracts, emails, images or log files. Classification is a prevalent supervised task of machine learning. The algorithms are designed to classify the given data points into n number of different classes based on patterns observed within the data. The classification of texts is a part of document classification that deals especially with text. It can be a sentence, a paragraph, or even the whole document. In addition, the classification of text data in general is more complex than document classification because usually it has less context to work with. With document classification, the entire document can be viewed as context, while with text classification, only the text itself can be considered as context.

There are many approaches for automatic document classification, the most common ones being supervised, unsupervised, and semi-supervised. For supervised methods, a training data set with labeled documents is needed to accurately predict the category of new documents. Conceptually, supervised methods attempt to find a connection between the document and its corresponding category by looking at labeled historical data. In contrast to supervised document classification, in the case of unsupervised methods, a learning dataset is not needed. Instead, they attempt to classify documents by looking just at the variance between documents. This results in different clusters that contain related documents, however, it is unclear for that method what those clusters (i.e. categories) are. Semi-supervised involves a mix of supervised and unsupervised methods. The semi-supervised method takes advantage of both a labeled training set and unlabeled data and is capable of improving the performance of supervised and unsupervised document classification methods.

Our motivation was that a collection of classified log files could be used for various tasks. They contain valuable information about the behavior of applications, systems, components, and networks. By classifying log files, organizations can better understand the performance and general health of their systems. They can identify patterns, anomalies, errors, and potential security threats. Log files can take up serious amounts of storage space, especially in the case of large-scale systems. With the classification of logs based on their significance and relevance, organizations can arrange storage resources and retention policies, ensuring that crucial logs are kept while less important ones are appropriately archived or discarded. Many industries and institutions are subject to regulatory requirements according to data privacy, security, or compliance. Properly classified log files can assure that these regulatory requirements are met by the organizations by providing auditors with clear and cataloged records of system activities. To implement advanced analytics techniques such as artificial intelligence or machine learning log file classification is often an essential task. By structuring log data in a logical and categorized manner, organizations can build more precise predictive models, error prediction systems, and other automated tools to enhance system management and monitoring.

While log analysis is a well-researched topic [17], [18], [19] and there are various works on the classification of the actual log lines for different purposes like anomaly detection, there is limited literature on the classification of log files themselves. The authors of [20] identify different types of firewall log messages with the use of support vector machines. For the same purposes, the authors of [21] propose an approach that uses shallow neural networks. The log files generated by MRI systems usually lack information on which body region was examined. In [22] pattern recognition methods are used to classify which body region the log file corresponds to. The type and the origin of a file are not always clear or available and sometimes only fragments of a file are available. In [23] the authors propose a new machine-learning approach, Semi-Supervised Generative Adversarial

Networks (SGAN) to combat this problem. Tree methods that use NLP, statistical features, and one-shot learning to define a file's type have been proposed in [24].

In this paper, we study the performance of a new approach that converts log files into lines that only contain an ID that corresponds to that line's event template. This way, instead of the procession of the full plain text files, we only have to take the ID-s into action while calculating the Term Frequency-Inverse Document Frequency (TFIDF) vectors. The organization of the paper is the following. In Section II concept of log template mining, the preliminary concepts and the definition of the classification algorithms, and our proposed algorithm are presented. The details of the data we used, the experiments that were performed, and their explanations are presented in Section III. Lastly, Section IV contains the conclusions and discusses the different future possibilities to investigate the matter at hand.

II. CONCEPTS AND PROBLEMS

A. LOG PARSING AND THE PROPOSED ALGORITHM

The entries of a log file are usually unstructured and raw due to the fact that programmers can insert free-text messages into their print statements. Each log entry contains runtime information about events that have happened like restarts, messages being sent or received, error occurrences, and so on. A log message usually starts with a list of information like timestamps, the module name that produced the message, and others. In this paper, we only focus on the message part. Each word in a message can be either a constant or a parameter. A constant token is always the same at each occurrence of a log line corresponding to an event type. Parameter tokens can be different on each occasion. A fragment of our working data can be seen in Figure 1.

The corresponding event template of this log message is "Receiving block <*> src: <*> dest: <*>". The "<*>" symbols indicate the presence of a parameter token, and the parameters of this specific entry can be seen in the parameter list. The template that was just discussed is identified by the unique ID "ABC123".

In our experiments, instead of using the whole log files as the input of the classification algorithms, in every file, we replace each line with its corresponding event template ID. By reducing the file size significantly, we enhance the efficiency of calculating Term Frequency-Inverse Document Frequency (TFIDF) [25] vectors and performing classifications. This leads to notable reductions in both computational time and resource usage. Subsequently, the TFIDF vectors are computed. The core principle here is that various types of log files contain different event types. Consequently, we anticipate that TF-IDF vectors derived from the same type of log file will be similar, whereas vectors from different types of log files will be distinct. These vectors, generated from unique identifiers rather than entire lines of text, serve as inputs for the classification algorithms. Class imbalance occurs when one class significantly outnumbers the others in a dataset. This phenomenon can be observed in the case of our data, which is detailed in Section III. Imbalance can lead to a model that is biased towards the majority class, making it perform poorly on the minority class. To deal with this problem, and achieve better results, we use the Synthetic Minority Over-sampling Technique (SMOTE) that was presented in [26]. SMOTE selects examples that are close to each other in the feature space, draws a line in the feature space between such examples, and then draws a new sample at some point along that line. The data obtained this way will serve as input to the classifier algorithms.

B. DECISION TREE AND RANDOM FOREST

Decision tree learning is a supervised learning method commonly used in data mining, machine learning, and statistics [27]. Let's assume that all of the input attributes have finite discrete domains, and there is a single target attribute called the "classification". Classes are the elements that can be found in the classification attribute's domain. All the internal nodes of the decision tree are labeled with an input attribute. The edges coming from a node labeled with an input attribute must be labeled with each of the possible values of the target attribute or the edge has to lead to a "lower" internal node on a different input attribute. Each leaf of a tree represents a class.

The tree is built by splitting a source set (at first, the root of the tree) into subsets. The splitting is based on rules on the different attributes. This process is repeated recursively. The recursion ends when a subset at a node has all the same values on the target attribute. This method of top-down induction of decision trees is a greedy algorithm and one of the most popular plans of action.

Decision trees tend to overfit their training sets. To correct this behavior Random Forest was proposed in [28]. It is a supervised classification method that generates multiple decision trees during training. Several decision tree classifiers are being fit on various sub-samples of the dataset. Then, averaging is used to control the over-fitting and improve the accuracy of the prediction. An extension of the algorithm was proposed in [29] that uses bagging. Let $X = x_1, x_2, \ldots, x_n$ be a training set with $Y = y_1, y_2, \ldots, y_n$ being the classes. With bagging, B times a random sample X_b, Y_b is selected to replace the training set and then that sample is being fitted to the trees by f_b. After training, predictions for unseen samples x' is made by averaging the predictions from all the individual decision trees on x':

$$f = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{1}$$

C. K-NEAREST NEIGHBORS (K-NN)

The k-nearest neighbors algorithm is a supervised non-parametric learning algorithm that can be used for classification and was proposed in [30] and later broadened in [31]. Training examples are represented as vectors in a multidimensional attribute space with each having their

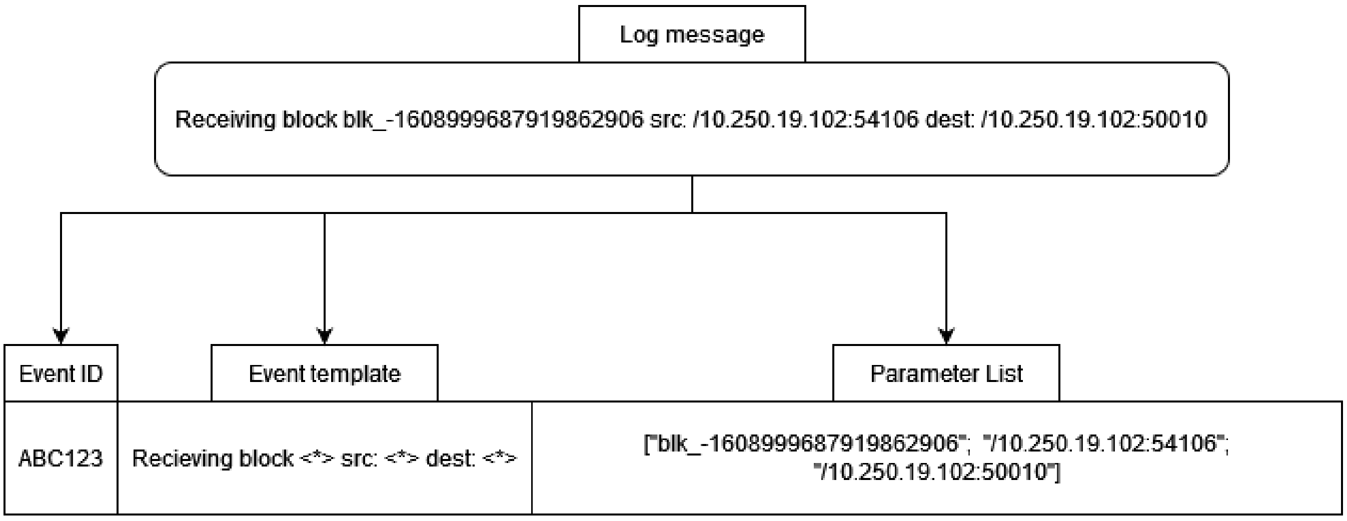
Log message
Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
Event ID
Event template
Parameter List
ABC123
Recieving block <*> src: <*> dest: <*>
["blk_-1608999687919862906", "/10.250.19.102:54106", "/10.250.19.102:50010"]

FIGURE 1. The process of log parsing.

corresponding class label. The attribute vectors and the labels are stored in the training phase. Let k be a user-defined constant, generally a small positive integer. In the classification stage, the test point that is an unlabeled vector is being classified by a plurality vote of its neighbors. In other words, the class that is the most frequent among the k neighbors of the test point is being assigned to it. If $k = 1$, the assigned class is simply the class of the vector that is closest to the test point. Various distance metrics like Minkowski distance [32], Manhattan distance [32], Jaccard distance [33], Cosine distance, Chebysev distance [34] and Hamming distance [35] can be used with the algorithm.

The most commonly used distance is the Euclidean distance:

$$\begin{aligned} d(p, q) &= \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \\ &= \|p - q\| \end{aligned} \tag{2}$$

where p and q are points in the space with coordinates $(p_1, p_2, \ldots, p_n)$ and $(q_1, q_2, \ldots, q_n)$ respectively.

D. ADABOOST CLASSIFIER

Ada-boost or Adaptive Boosting is an ensemble boosting classifier that was proposed in 1996 by Yoav Freund and Robert Schapire [36]. To increase the accuracy of classification, the base idea is to combine multiple classifiers. AdaBoost is an iterative ensemble method. By combining multiple weakly performing classifiers, AdaBoost Classifier builds a strong classifier with high accuracy. The basic concept behind Adaboost is to set the weights of classifiers and train the data sample in each iteration such that it ensures accurate predictions of unusual observations. As the base classifier, any machine learning that accepts weights on the training set can be used. The algorithm works in the following steps. First, a random subset of the training data is selected. Then, the model is iteratively trained by electing the training set based on the accurate prediction of the last training. A higher weight is assigned to wrongly classified observations which results in a high probability in the next iteration for these observations. The trained classifier also gets assigned a weight based on its accuracy. The better the accuracy the higher the weight it gets. The iteration process stops after a specified number of estimators or when the training data fits without error. To classify a vote is performed between all of the classifiers that have been built.

The steps of an AdaBoost Classifier with 3 as the number of estimators can be seen in Figure 2.

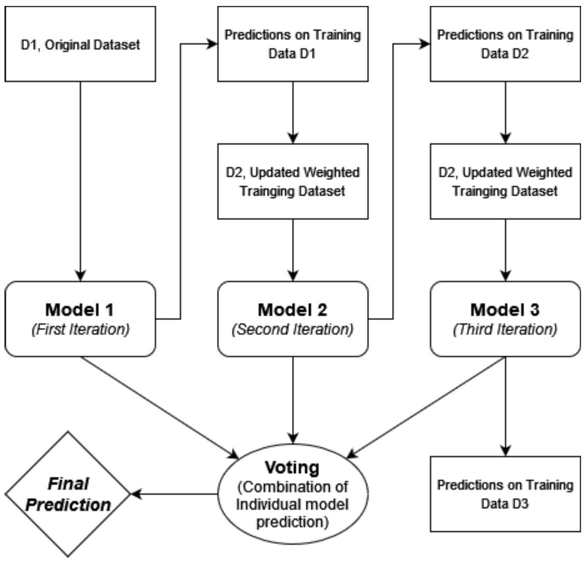
D1, Original Dataset
Predictions on Training Data D1
Predictions on Training Data D2
D2, Updated Weighted Trainging Dataset
D2, Updated Weighted Trainging Dataset
Model 1 (First Iteration)
Model 2 (Second Iteration)
Model 3 (Third Iteration)
Final Prediction
Voting (Combination of Individual model prediction)
Predictions on Training Data D3

FIGURE 2. 3 steps of an AdaBoost Classifier with 3 estimators.

III. EXPERIMENTS AND RESULTS

A. DATA AND EXPERIMENTAL ANALYSIS

Six real-life life log file collections were employed to evaluate the effectiveness of our proposed algorithm. The HDFS_v1, BGL, HealthApp, HPC, Proxifier, and OpenSSH datasets and their corresponding event types were provided in [37]. For our testing purposes, we assume that we already have the event types. Log lines are usually made up of several parts, like a timestamp, ID, log level, source, and message. We only use the unstructured message part of the log lines to classify the files since the other parts are not relevant for the task. For our testing purposes, we cut them into parts of the same size. During the partition, the collections were partitioned into different numbers of log files with different numbers of log lines per file to better represent the variety of real-life log files. Overall, 74 files were created. The details of the data can be seen in Table 1.

TABLE 1. Details of the testing data.

Name	Average size (MB)	Number of lines	Number of files
HDFS_v1	13.26	100K	25
BGL	13.15	100K	25
HealthApp	4.4	50K	5
HPC	3.4	50K	8
Proxifier	0.6	5K	4
OpenSSH	10	100K	7

To evaluate the effectiveness of our algorithm, multiclass classification was used which is the problem of classifying instances into one of three or more classes. For each instance in the dataset, the classifier predicts a class label. The number of instances where the predicted class label matches the true class label based on the set of test data is the number of true positives. We performed various experiments, like comparing the elapsed time, precision, recall, f-score, and accuracy values achieved by our algorithm and other classifiers. The result was obtained as an average of 100 different classification runs. For a set of test data, where the true values are known, the confusion matrix can be used to indicate the performance of a classification model. Figure 3 showcases the structure of such a matrix.

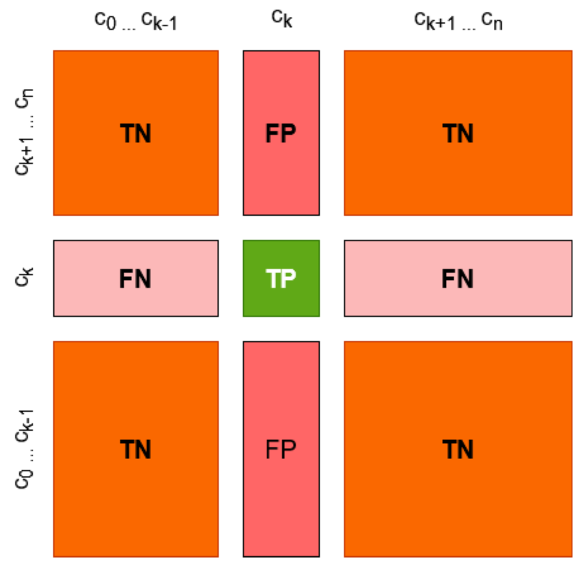

FIGURE 3. Confusion matrix.

where the number of c_i (class i) samples that have been correctly classified as c_i by the model is indicated by T_p (True Positives), T_n (True Negatives) marks the number of samples that are not c_i and have been predicted to be not c_i The number of samples that have been predicted to be c_i even though they are not being represented by F_p (False Positives) and the number of samples that are c_i but have been classified otherwise is F_n (False Negatives).

The precision of a model quantifies the number of positive predictions made by that model.

$$P = \frac{T_p}{T_p + F_p} \tag{3}$$

Recall implies the fraction of relevant samples that were found.

$$R = \frac{T_p}{T_p + F_n} \tag{4}$$

To capture both properties, precision, and recall can be combined into a single measure called F-measure:

$$F = \frac{2 \times P \times R}{P + R} \tag{5}$$

Accuracy indicates the ratio of the data that has been correctly predicted and is denoted as:

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{6}$$

The abbreviations used in the experiments are shown in Table 2. The parameters of the Classifiers are shown in Appendix.

TABLE 2. Abbreviations.

Abbreviation	Algorithm
p_rf	RandomForest Classifier with the proposed algorithm
p_knn	K-NN Classifier with the proposed algorithm
p_ada	AdaBoost Classifier with the proposed algorithm
p_dt	Decision Tree Classifier with the proposed algorithm
o_rf	Original RandomForest Classifier
o_knn	Original K-NN Classifier
o_ada	Original AdaBoost Classifier
o_dt	Original Decision Tree Classifier

B. EXPERIMENT 1: COMPARING THE PRECISION VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

First, the precision values attained by the different classification algorithms on the 74 log files were inspected, to measure the number of the correctly classified log files. The results are shown in Figure 4, while the numerical results are shown in Table 3.

It can be seen that the proposed algorithm achieves 20.2% and 12.5% less score than the original KNN and AdaBoost Classifier respectively, however, outperforms the original Random Forest algorithm with 37.1% and the original Decision Tree Classifier with 10.4%.

C. EXPERIMENT 2: COMPARING THE RECALL VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

To estimate the algorithm's ability to find all relevant instances of a class in a dataset we investigated the recall

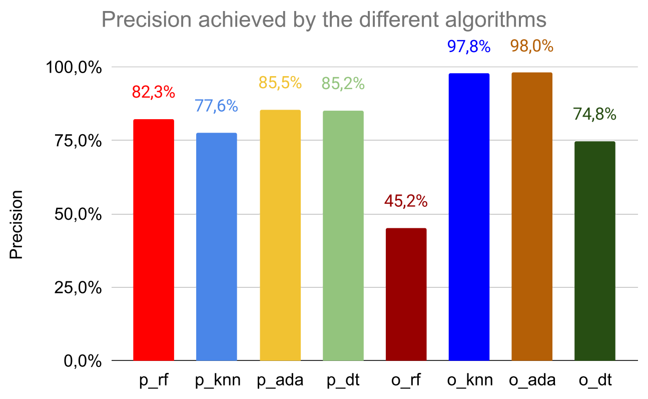

FIGURE 4. **Precision achieved by the different algorithms.**

values that were achieved by the classification algorithms. The results can be seen in Figure 5 and Table 3.

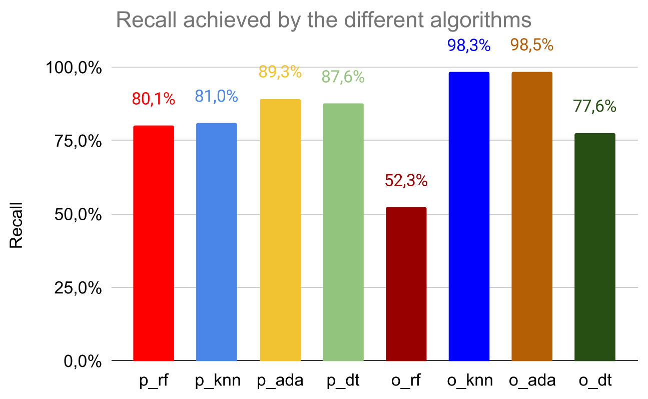

FIGURE 5. **Recall achieved by the different algorithms.**

The proposed algorithm falls behind with 17.3% and 9.2% in the case of KNN and AdaBoost Classifier while achieving better scores than the Random Forest and Decision Tree Classifiers with 27.8% and 10%.

D. EXPERIMENT 3: COMPARING THE F-SCORE VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

In this experiment, the F-score, which combines the properties of both the precision and recall achieved by the different measures were compared. The results are presented in Figure 6 and Table 3

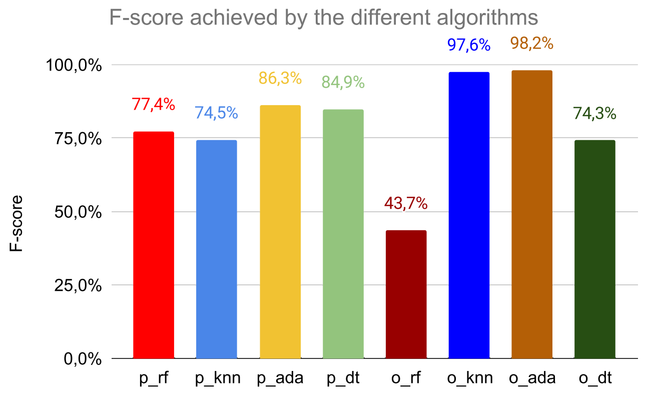

FIGURE 6. **F-score achieved by the different algorithms.**

As in the previous experiments, the proposed algorithm is outperformed by the KNN and AdaBoost Classifiers with 23.1% and 11.9%. On the other hand, the proposed algorithm once again has better scores than the Random Forest and Decision Tree Classifiers with 33.7% and 10.6%.

E. EXPERIMENT 4: COMPARING THE ACCURACY VALUES ACHIEVED BY THE DIFFERENT ALGORITHMS

The closeness or farness of a given set of measurements (reading, observations, etc.) from their true value is indicated by the accuracy, which we investigated in this experiment. The results can be seen in Figure 7 and Table 3.

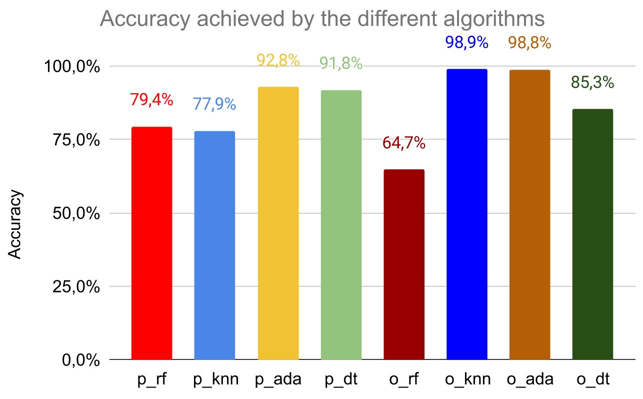

FIGURE 7. **Accuracy achieved by the different algorithms.**

Once again, the original KNN and AdaBoost Classifiers have greater values like 21% and 6% respectively. Just as before, the original Random Forest and Decision Tree classifiers have achieved worse values with 14.7% and 6.5%.

TABLE 3. **Details of the results.**

Classifier	Precision (%)	Recall (%)	F-score (%)	Accuracy (%)	Average (%)	Speed (s)
p_rf	82.3	80.1	77.4	79.4	79.8	8.919
p_knn	77.6	81	74.5	77.9	77.8	8.896
p_ada	85.5	89.3	86.3	92.8	88.5	9.144
p_dt	85.2	87.6	84.9	91.8	87.4	8.906
o_rf	45.2	52.3	43.7	64.7	51.5	462.933
o_knn	97.8	98.3	97.6	98.9	98.1	462.794
o_ada	98	98.5	98.2	98.8	98.4	542.289
o_dt	74.8	77.6	74.3	85.3	78.0	466.877

F. EXPERIMENT 5: COMPARING THE SPEED OF THE DIFFERENT ALGORITHMS

The precision, recall, f-score, and accuracy values achieved by an algorithm are essential, however, the time it takes for an algorithm to classify the log files is also an important measure, especially when numerous log files have to be classified. In this experiment, the run time of the different methods was investigated. The results are shown in Figure 8 and in Table 3.

It can be seen that the classification algorithms that are applied to our simplified data (log files only consisting of event IDs instead of the original log messages) need at least 52 to 59 times less time to classify the log files. Out of the

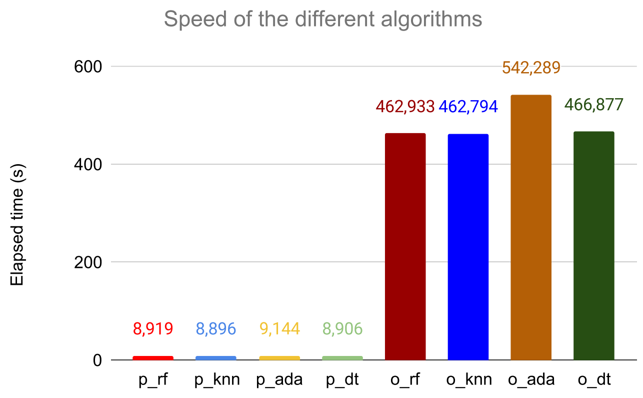

FIGURE 8. The speed of different algorithms.

TABLE 4. ML parameters.

Classifier	Parameter	Value
RandomForest Classifier	Class weight	Balanced
RandomForest Classifier	Criterion	Gini
RandomForest Classifier	Max depth	5
RandomForest Classifier	Number of estimators	10
RandomForest Classifier	Max features	1
K-NN Classifier	Number of Neighbors	19
K-NN Classifier	Metric	Minkowski
K-NN Classifier	Minkowski parameter	2
AdaBoost Classifier	Estimator	Decision tree
AdaBoost Classifier	Number of estimators	50
AdaBoost Classifier	Learning rate	0.01
AdaBoost Classifier	Algorithm	SAMME.R
Decision Tree Classifier	Criterion	Gini
Decision Tree Classifier	Max depth	None
Decision Tree Classifier	Max features	None

original algorithms, AdaBoost Classifier proved to be the slowest.

G. OUR RESULT

Apart from the Decision Tree Classifier, the high accuracy values indicate that both the original models and the ones that are used with our proposed algorithm correctly predict the class labels. The high precision and recall values signal that there are large numbers of true positive classifications that were correctly identified. This is confirmed by the F-scores. The above-mentioned results demonstrate that the classification is meaningful. If the average of the first four experiments (precision, recall, f-score, accuracy) is taken, it can be seen that our proposed algorithm used with any of the classification methods achieves better results than the original RandomForest and Decision Tree classifiers. The proposed algorithm used with KNN and AdaBoost Classifiers falls behind the original algorithms in the case of performance with around 20% and 10% respectively. The main advantage of the use of the proposed algorithm is that the time that is needed to classify the log files is 52, and 59 times less which can pay off, especially in the case of large and less crucial types of log files.

In [20] the authors propose four multiclass support vector machine (SVM) classifiers to classify information based on log files. Based on their experiments, they came to a conclusion that SVM used with RBF activation is the best suited for the task. This approach achieved a 76.4% of F-score, 63% of precision, and 97.1% of recall. The speed of their algorithm is not investigated in their paper. It can be said that classifiers used in pairs with our proposed method achieve better results than the one proposed in the aforementioned paper. To determine the type of a file, the authors of [23] use byte values, distribution, and frequency as input features to classifiers. Apart from measuring the performance of traditional methods like Decision Tree classifiers or KNN, they propose a new approach using neural networks, specifically Semi-Supervised Generative Adversarial Networks (SGAN). Their study, however, does not investigate precision, recall, or F-score metrics, only accuracy is reported. On their data, the Decision Tree classifier, KNN, and SGAN have accuracy values of 90.7%, 89%, and 97.6% respectively. The SGAN method outperforms the individual classifiers paired with our proposed method. However, the time required for classification with SGAN has not been investigated. It is conceivable that SGAN demands more time for classification compared to our proposed method, which achieves results in a significantly short duration.

IV. CONCLUSION

The classification of log files with different origins has become an important assignment in recent years. They are made up from log lines that are usually free-text messages with parameters. Each line belongs to an event type. An event type is a template, where the constant part of the template is the same at any occurrence, while the parameter parts might change.

Classifying log files with full-length log messages is a resource and time-consuming task. To combat this, in this paper, we propose a new algorithm designed to modify log files, which are subsequently used as input for classification algorithms. Our algorithm alters the log files to only contain the IDs of the event types instead of the full text.

To evaluate the performance of the classification methods that had the modified files as their input, we conducted various experiments like investigating the precision, recall, f-score, and accuracy values achieved during the classification. The results yielded that in the case of KNN and AdaBoost Classifier, the original algorithms outperformed the ones using the proposed algorithm with an average of 20% and 10% respectively. In the case of Random Forest, and Decision Tree Classifiers, the ones with the input generated by the proposed algorithm have surpassed their original counterpart with an average of 28% and 9%.In terms of speed, the classification takes 52 to 90 times less time, if the altered log files created by our proposed algorithm are used as input for the classifiers.

Our main contribution is that while the proposed algorithm might lead to slightly reduced classification accuracy when used alongside certain classifiers, it significantly decreases the time required for file classification by several magnitudes.

We only evaluated the performance of a set of classification algorithms, in the future, it could be beneficial to investigate the performance of other methods. It would be also beneficial to use the altered log files as input for machine learning algorithms like CNN or LSTM.

APPENDIX
ML PARAMETERS
See Table 4.

REFERENCES
[1] C. J. Aivalis, "Log file analysis," in *Handbook of E-Tourism*. Springer, 2022, pp. 659–683.
[2] A. Fawzia Omer, H. A. Mohammed, M. A. Awadallah, Z. Khan, S. U. Abrar, and M. D. Shah, "Big data mining using K-means and DBSCAN clustering techniques," in *Big Data Analytics and Computational Intelligence for Cybersecurity*. Springer, 2022, pp. 231–246.
[3] G. Neelima and S. Rodda, "Predicting user behavior through sessions using the web log mining," in *Proc. Int. Conf. Adv. Human Mach. Interact. (HMI)*, Mar. 2016, pp. 1–5.
[4] J. Kim, M. Park, H. Kim, S. Cho, and P. Kang, "Insider threat detection based on user behavior modeling and anomaly detection algorithms," *Appl. Sci.*, vol. 9, no. 19, p. 4018, Sep. 2019.
[5] C. Stachl, Q. Au, R. Schoedel, S. D. Gosling, G. M. Harari, D. Buschek, S. T. Völkel, T. Schuwerk, M. Oldemeier, T. Ullmann, H. Hussmann, B. Bischl, and M. Bühner, "Predicting personality from patterns of behavior collected with smartphones," *Proc. Nat. Acad. Sci. USA*, vol. 117, no. 30, pp. 17680–17687, Jul. 2020.
[6] J. Svacina, J. Raffety, C. Woodahl, B. Stone, T. Cerny, M. Bures, D. Shin, K. Frajtak, and P. Tisnovsky, "On vulnerability and security log analysis: A systematic literature review on recent trends," in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, 2020, pp. 175–180.
[7] C. N. Kabat, D. L. Defoor, P. Myers, N. Kirby, K. Rasmussen, D. L. Saenz, P. Mavroidis, N. Papanikolaou, and S. Stathakis, "Evaluation of the elekta agility MLC performance using high-resolution log files," *Med. Phys.*, vol. 46, no. 3, pp. 1397–1407, Mar. 2019.
[8] J. Kimball, R. A. Lima, and C. Pu, "Finding performance patterns from logs with high confidence," in *Proc. 27th Int. Conf., Held Services Conf. Fed., SCF Web Services–ICWS*, Honolulu, HI, USA. Springer, Sep. 2020, pp. 164–178.
[9] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–37, Jul. 2022.
[10] M. Siwach and S. Mann, "Anomaly detection for web log data analysis: A review," *J. Algebr. Statist.*, vol. 13, no. 1, p. 129, 2022.
[11] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," 2021, *arXiv:2107.05908*.
[12] M. Catillo, A. Pecchia, and U. Villano, "AutoLog: Anomaly detection by deep autoencoding of system logs," *Expert Syst. Appl.*, vol. 191, Apr. 2022, Art. no. 116263.
[13] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, and Q. Tian, "LogFormer: A pre-train and tuning pipeline for log anomaly detection," in *Proc. AAAI Conf. Artif. Intell.*, Mar. 2024, vol. 38, no. 1, pp. 135–143.
[14] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 859–864.
[15] K. Shima, "Length matters: Clustering system log messages using length of words," 2016, *arXiv:1611.03213*.
[16] D. Plaisted and M. Xie, "DIP: A log parser based on 'disagreement index token' conditions," in *Proc. ACM Southeast Conf.*, Apr. 2022, pp. 113–122.
[17] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, 2019, pp. 121–130.
[18] M. Nagappan, "Analysis of execution log files," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, vol. 2, May 2010, pp. 409–412.
[19] J. H. Andrews, "Testing using log file analysis: Tools, methods, and issues," in *Proc. 13th IEEE Int. Conf. Automated Softw. Eng.*, 1998, pp. 157–166.
[20] F. Ertam and M. Kaya, "Classification of firewall log files with multiclass support vector machine," in *Proc. 6th Int. Symp. Digit. Forensic Secur. (ISDFS)*, Mar. 2018, pp. 1–4.
[21] Q. A. Al-Haija and A. Ishtaiwi, "Multiclass classification of firewall log files using shallow neural network for network security applications," in *Proc. Soft Comput. Secur. Appl. (ICSCS)*. Springer, 2021, pp. 27–41.
[22] N. Kuhnert, O. Lindenmayr, and A. Maier, "Classification of body regions based on MRI log files," in *Proc. Int. Conf. Comput. Recognit. Syst.* Springer, 2017, pp. 102–109.
[23] K. S. Germain and J. Angichiodo, "Adversarial networks and machine learning for file classification," 2023, *arXiv:2301.11964*.
[24] S. Lisker, A. Butman, R. Dubin, A. Dvir, and C. Hajaj, "File type identification classifier with classification-by-retrieval and one-shot learning," *SSRN*.
[25] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge, U.K.: Cambridge Univ. Press, 2011.
[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.
[27] O. Z. Maimon and L. Rokach, *Data Mining With Decision Trees: Theory and Applications*, vol. 81. Singapore: World Scientific, 2014.
[28] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, 1995, pp. 278–282.
[29] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
[30] E. Fix and J. L. Hodges, "Discriminatory analysis. Nonparametric discrimination: Consistency properties," *Int. Stat. Rev./Revue Internationale de Statistique*, vol. 57, no. 3, p. 238, Dec. 1989.
[31] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[32] H. Minkowski, *Geometrie Der Zahlen*. German: BG Teubner, 1910.
[33] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New Phytologist*, vol. 11, no. 2, pp. 37–50, Feb. 1912.
[34] C. D. Cantrell, *Modern Mathematical Methods for Physicists and Engineers*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[35] B. Waggener and W. N. Waggener, *Pulse Code Modulation Techniques*. Berlin, German: Springer, 1995.
[36] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[37] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," 2020, *arXiv:2008.06448*.

PÉTER MARJAI received the M.Sc. degree in computer science from the Faculty of Informatics, Eötvös Loránd University, Budapest, in 2021, where he is currently pursuing the Ph.D. degree in information systems. His scientific research interests include centrality measures, log processing, parsing, and compression.

ATTILA KISS received the Ph.D. degree in database theory, in 1991. Since 2010, he has been the Head of the Department of Information Systems, Eötvös Loránd University, Hungary. He is also teaching with J. Selye University, Slovakia. Seven students received their Ph.D. degrees under his supervision. He has more than 190 scientific publications. His research interests include information systems, data mining, and artificial intelligence.