EPiC Series in Computing
Volume 106, 2025, Pages 210–220
Proceedings of the 20th Asia Joint Conference on Information Security

A Lightweight and Accurate Classification Framework for Traffic Log Analysis Based on an Effective Feature Representation Method

Ayako Sasaki[1], Takeshi Takahashi[2], Keisuke Furumoto[3], Chun-I Fan[4], and Tomohiro Morikawa[5]*

[1] University of Hyogo, Japan
ad231025@guh.u-hyogo.ac.jp
[2] National Institute of Information and Communications Technology, Japan
takeshi_takahashi@nict.go.jp
[3] National Institute of Information and Communications Technology, Japan
k.furumoto@nict.go.jp
[4] National Sun Yat-sen University, Taiwan
cifan@mail.cse.nsysu.edu.tw
[5] University of Hyogo, Japan
morikawa@gsis.u-hyogo.ac.jp

Abstract

As cyberattacks become increasingly sophisticated, organizations face an urgent need for timely and accurate incident response to reduce their impact on critical systems. Automating the analysis of network traffic logs has become essential for supporting security analysts and specialists. Although many previous studies have applied machine learning to address this task, they often encounter challenges such as dependence on large-scale analytics platforms, limited exploration of machine learning algorithms, and difficulties in deploying distributed systems due to high costs, complexity, and privacy concerns. To tackle these limitations, we propose a lightweight and accurate machine learning-based framework for the automatic analysis of network traffic logs. Our approach transforms log data into feature vectors using a document-based feature representation method. Experimental results on benchmark datasets demonstrate that our method enables efficient and effective traffic log analysis suitable for practical deployment.

1 Introduction

Cyberattacks are becoming increasingly sophisticated, frequently leading to system intrusions and a subsequent escalation of damage. As such, a prompt and accurate initial response is indispensable for minimizing the impact of cyberattacks on critical systems and ensuring effective threat mitigation. Analyzing network traffic logs is a fundamental aspect of the initial

*Corresponding Author.

response to cyberattacks. Given the substantial volume of logs generated on a daily basis and the shortage of skilled security specialists capable of conducting thorough analyses, there is an increasing demand for automated approaches to support and streamline this process.

In response to this need, several studies [3, 10, 12, 2, 14] have investigated the application of machine learning techniques to automate the analysis of network traffic logs. Some of these works emphasize improving accuracy to substantially reduce the time and effort required for manual analysis. Traditional machine learning algorithms such as Support Vector Machines (SVM), Naive Bayes (NB), and C4.5 Decision Trees (DT) are commonly used to classify and categorize network traffic. In contrast, other studies prioritize scalability, aiming to develop methods capable of efficiently processing large volumes of log data. These approaches frequently leverage the Hadoop Distributed File System (HDFS) for scalable data storage and employ Apache Spark, built on top of HDFS, to execute machine learning tasks and support efficient, large-scale, distributed data processing.

However, the following three issues remain unaddressed: (1) The feature representation methods adopted in previous studies cannot deliver efficient performance without relying on large-scale data analytics platforms. (2) Prior research has evaluated only a limited set of machine learning algorithms, leaving others unexplored and thus limiting a comprehensive comparison of performance. (3) Implementing distributed systems for log classification in typical corporate environments poses challenges due to the high costs and operational complexity associated with their deployment and maintenance. Although utilizing third-party distributed platforms could be a potential solution, many organizations may be reluctant to transmit communication logs externally due to privacy and security concerns.

To address the aforementioned challenges, we propose a lightweight and accurate log classification framework that offers comparable speed and accuracy to previous approaches while requiring only a single workstation. This design enables practical adoption of machine learning-based log analysis in enterprise environments. For feature representation, we employ Doc2Vec [4], a technique that encodes each log line, treated as a separate document, into a fixed-dimensional vector. By increasing the dimensionality of these vectors, we expand the feature space for the machine learning algorithms, aiming to enhance evaluation metrics such as the F1-score. The choice of Doc2Vec in our method is motivated by its potential to improve classification performance, especially for algorithms with relatively short training and inference times that still exhibit room for improvement in accuracy. In addition to the classifiers used in previous studies, we incorporate three additional algorithms: AdaBoost, Multilayer Perceptron (MLP), and Stochastic Gradient Descent (SGD). The performance of each classifier is evaluated in terms of accuracy, recall, F1-score, precision, training time, and testing time when classifying a dataset composed of Zeek Conn Logs [11] generated by a network analyzer.

As a result, all algorithms except NB—namely AdaBoost, MLP, SGD, Logistic Regression (LR), DT, and Random Forest (RF)—achieved F1-scores of over 99%. In particular, when the dataset was converted into 30-dimensional vectors and classified using MLP, the highest F1-score of 99.89% was achieved in our experiment evaluation. Similarly, with 30-dimensional vectors as input, SGD and LR achieved an F1-score of 99.70%. These results are comparable to the F1-scores achieved by RF and DT in the prior study that used the same dataset as ours [3]. Furthermore, MLP achieved a test time of 0.1000 seconds, which is on par with prior research, while SGD showed shorter computation times, with a training time of 2.350 seconds and a test time of 0.0070 seconds. Additionally, similarly, the test time for LR was also very short, at 0.0067 seconds. The results indicate that classifiers such as SGD, LR and MLP, when combined with Doc2Vec preprocessing, are effective for the classification of Zeek Conn Log data on a single workstation.

The main contributions of this paper are summarized as follows:

- We propose a log classification framework that employs a document-based feature representation method and explores several previously untested machine learning algorithms to achieve high performance while maintaining low environmental and computational costs.
- Our approach facilitates practical deployment in standard enterprise environments without the reliance on distributed systems.
- Based on the experimental results, we demonstrate that our method achieves shorter computation times, with a training time of 2.350 seconds and a test time of 0.0070 seconds, without sacrificing accuracy compared to previous works.

The remainder of this paper is organized as follows: Section 2 reviews related work on network log classification. Section 3 introduces the proposed method for classifying network traffic logs. Section 4 evaluates the performance of the proposed method. Section 5 discusses the limitations of this approach, and Section 6 concludes the paper.

2 Related Work

Firstly, this section introduces three related studies that classified the KDD'99 dataset [1], which consists of TCP dump data collected during a nine-week-long attack simulation. In a 2012 study, Swamy et al. [14] classified the KDD'99 dataset using the C4.5 DT and an improved version of the C4.5 DT. Their method employed these two machine learning algorithms as detection mechanisms in a Network Intrusion Detection System (NIDS) to detect attacks within the KDD'99 dataset. The experimental results showed that while the standard C4.5 DT achieved an accuracy of 95.7%, the improved version achieved a higher accuracy of 96.9%. Furthermore, the test time for both DTs was kept as low as 0.09 seconds. In a 2007 study, Panda et al. [12] classified the dataset into normal traffic and four categories of attacks using a Naive Bayes classifier. In their 2010 study, Mulay et al. [10] proposed a method for classifying normal traffic and the same four categories of attacks as in the 2007 study by Panda et al., by combining Support Vector Machines (SVM) with DTs to construct a multiclass SVM.

Next, we introduce two related studies that classified datasets of Zeek connection logs labeled according to the MITRE ATT&CK framework [6]. In a 2022 study, Bagui et al. [2] classified the UWF-ZeekData22 dataset [11], using several machine learning algorithms: NB, RF, DT, SVM, Gradient-Boosting Trees (GBT), and LR. The study conducted two types of binary classification, targeting data representing normal traffic and data labeled with a specified tactic. The tactics selected for classification were "Reconnaissance" [8] and "Discovery" [7]. As a result, DT, GBT, and RF each achieved high F1-scores exceeding 98.6%, regardless of the tactic selected. In a follow-up study in 2023, Bagui et al.[3] conducted classification experiments using a dataset, UWF-ZeekDataFall22[11], which was generated from newly collected Zeek connection logs. In the follow-up study, in addition to the previously used tactics "Reconnaissance" and "Discovery," the tactic "Resource Development" was also included in binary classification tasks[9]. Furthermore, multi-class classification was conducted to distinguish among these three tactics and normal traffic. The results indicated that DT and RF exhibited high performance in both binary and multi-class classification, achieving F1-scores of over 99.9% even in the multi-class classification. In particular, DT completed training in under 14 seconds.

Both studies utilized HDFS for storing logs and employed Apache Spark for performing machine learning tasks. However, adopting such distributed systems for log classification in standard corporate environments is challenging due to the high costs and complexity involved in deployment and maintenance. While the use of distributed platforms provided by third parties is a possible alternative, some companies may be reluctant to transmit communication logs externally due to security concerns. Therefore, we propose a log classification method that does not rely on distributed systems and is capable of delivering practical performance in environments that can be prepared by standard corporations.

3 Classification Framework

3.1 Overview

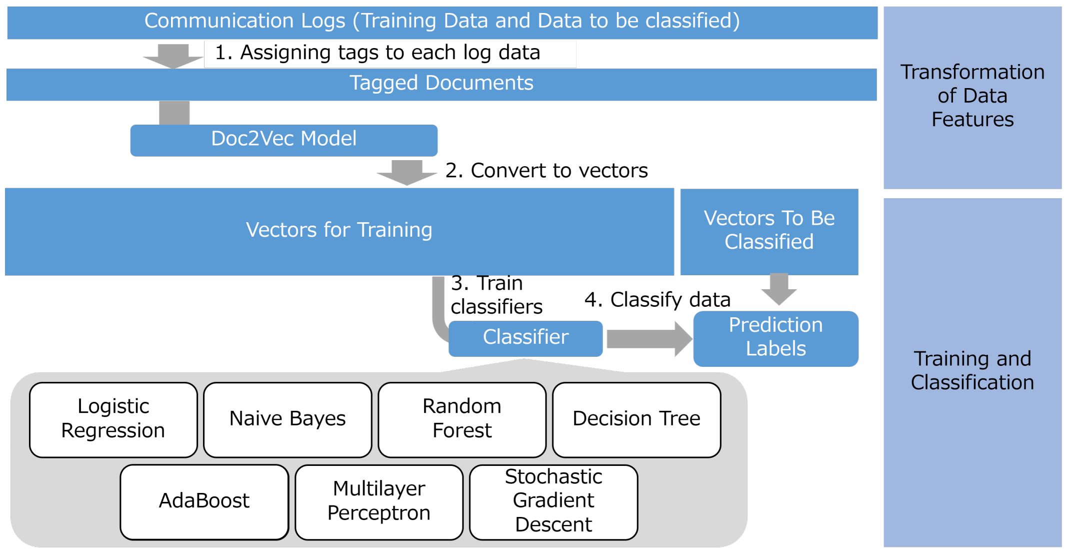

Figure 1: Overview of the proposed log classification method

This subsection presents our proposed method for classifying communication log data by attack type. As shown in Figure 1, our approach consists of two main stages: transformation of data features and machine learning-based training and classification.

We adopt Doc2Vec [4] for transforming the features of the communication logs. Details are provided in Section 3.2, but in brief, Doc2Vec enables documents to be represented as fixed-length distributed vectors using a model trained through unsupervised learning. Before converting the data into vectors, it is necessary to assign tags to each log entry in both the dataset to be classified and the dataset used for training the machine learning models. This allows Doc2Vec to recognize each line of log data as an individual document. Using these tagged datasets, we train a Doc2Vec model and then convert both datasets into vectorized form with the trained Doc2Vec model.

In the next stage, communication log classification, our method employs not only LR, NB, RF, and DT, which were used in the 2023 study by Bagui et al. [3], but also AdaBoost, MLP, and SGD. These models, after being trained on the vectorized training dataset, assign predicted attack type labels to the target dataset.

3.2 Feature Representation

The communication logs generally contain various types of fields, such as numeric values, strings, and boolean values. In our study, Doc2Vec [4], a library for converting documents into vector representations called document vectors, is employed to transform diverse types of data into vector form. It is based on Word2Vec [5], which converts words into distributed representation vectors, achieving vectorization at the document level.

As a preliminary step for explaining Doc2Vec, we describe the model used to obtain distributed word representations. This model predicts the central word based on a set of surrounding words (context) in a sentence. To build the model, we assign an initial unique vector $w_0, w_1, \ldots, w_T$ to each word in the training text data and represent these vectors as columns in a matrix W. Given a sequence of words $w_0, w_1, \ldots, w_T$, the goal is to maximize the average log probability defined by the following formula.

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t \mid w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k}) \tag{1}$$

The vectors are updated so as to maximize this formula. Through this process, the distributed representation vectors are able to capture the meanings of words. The probabilities can be computed using methods such as softmax, as shown below.

$$p(w_t \mid w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{2}$$

y_i is the score assigned to any word w_i among $w_{t-k}, \ldots, w_{t+k}$, and it is computed by the following formula.

$$y = b + Uh(w_{t-k}, ..., w_{t+k}; W) \tag{3}$$

, where b and U are parameters, and the function h is constructed by either concatenating or averaging the input vectors.

The concept of distributed representation vectors for words is extended to document vectors, which can be obtained by building a model that predicts the central word from contexts sampled from a document. When converting a set of documents into document vectors, an initial vector is assigned to each document, and these vectors are organized as columns in a matrix D. The main difference from the word vector framework is Equation (3) is constructed from both D and W; otherwise, the process of updating document vectors, parameters, and W follows the same steps. After the training process, the constructed model has W and other parameters fixed. Using this model, it is possible to infer document vectors for new documents.

4 Experiment

This section introduces our experimental procedure and the dataset used in the experiment, and describes the result obtained. Finally, we present a comparison with related studies.

4.1 Procedures and Evaluation Metrics

First, as described in Section 4.2, we extract data from UWF-ZeekDataFall22 and use it as the dataset for our experiment. The extracted dataset was then transformed into fixed-length vector datasets using Doc2Vec. We used 80% of the converted data for training and evaluated the classification performance on the remaining 20%. We prepared five variations of the transformed

Table 1: Parameters Adopted for Each Machine Learning Algorithm

Classifier		scikit-learn
ML Algo.	**Num. of Attrs.**	**Parameters**
LR	18, 20, 25, 30, 35	'C': 10, 'solver': 'saga'
NB	18, 20, 25, 30, 35	'var_smoothing': 1e-08
RF	18, 20, 25, 30, 35	'criterion': 'entropy'
DT	18, 20, 30, 35	'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 5
DT	25	'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20
AdaBoost	18, 20, 25, 30, 35	'estimator': sklearn.tree.DecisionTreeClassifier(max_depth=5, random_state=1), 'n_estimators': 250, 'learning_rate': 0.14
MLP	18	All parameters are default values.
MLP	20	'alpha': 1e-06, 'max_iter': 400
MLP	25, 30, 35	'alpha': 1e-05, 'max_iter': 400
SGD	18	'alpha': 1e-05, 'eta0': 0.001, 'penalty': 'l1', 'max_iter': 10000
SGD	20	'alpha': 1e-06, 'eta0': 0.001, 'penalty': 'l1', 'max_iter': 10000
SGD	25	'alpha': 1e-05, 'eta0': 0.001, 'penalty': None, 'max_iter': 10000
SGD	30	'alpha': 1e-06, 'eta0': 0.001, 'loss': 'log_loss', 'penalty': None, 'max_iter': 10000
SGD	35	'alpha': 1e-05, 'eta0': 0.001, 'loss': 'modified_huber', 'penalty': None, 'max_iter': 10000

datasets, with vector sizes of 18, 20, 25, 30, and 35, respectively. By increasing the number of features, we aimed to improve the classification performance of algorithms that, while having relatively short training and testing times, still had room for accuracy improvement. In our experiment, we evaluate 35 combinations, as there are five types of input vector sizes and seven types of algorithms. The machine learning algorithms were implemented using the scikit-learn library [13], and NB refers to Gaussian Naive Bayes in the experiment.

Next, grid search was performed to identify more suitable parameters for each algorithm. The parameters adopted in our experiment are shown in Table 1. Parameters not listed in the table were left at their default values. Except for NB, all classifiers have a parameter called 'random_state' for fixing the random seed, which was set to 1 across all classifiers.

Using the identified parameters, the classifiers were trained and tested, and accuracy, recall, F1-score, precision, and the false positive rate (FPR) were calculated to evaluate performance. The metrics other than accuracy are first calculated for each class, and then a weighted average is taken as the overall evaluation score. In addition to the five evaluation metrics, we separately measured the time taken for training and for testing.

Also, the experiment was conducted using a single workstation equipped with an Intel(R) Core(TM) i9-14900K processor and 188 GiB of memory.

4.2 Dataset

We validated our proposed communication log classification framework using the UWF-ZeekDataFall22 dataset, which was employed in the 2023 study by Bagui et al.[3]. The dataset consists of connection logs collected by the network analyzer Zeek during cyber attack exercises conducted in the experimental environment of the University of West Florida. Logs representing normal communication are labeled as ”none”, while logs indicating traces of attacks are labeled with one corresponding MITRE ATT&CK tactic, resulting in a total of 13 types of labels. Since this dataset includes tactics with a small number of samples, our experiments used 694,121 log entries labeled with one of the following four labels: none, Resource Development, Reconnaissance, or Discovery. These labels represent the top four tactics when ranked in descending order of the number of samples.

Given its large size, UWF-ZeekDataFall22 is well-suited for evaluating lightweight methods designed to classify large-scale datasets, such as our method. Another reason for selecting this dataset is that it is already mapped to MITRE ATT&CK tactics, which makes it easy to begin training and evaluating machine learning algorithms. The number of records for each tactic is presented in the previous study by Bagui et al. [3].

4.3 Results

The results of all 35 experimental patterns are shown in Table 2. As shown in the table, the F1-scores exceeded 95% in all patterns, and the other scores also reached similarly high levels. For all algorithms, the highest F1-score was achieved when the number of features was 30, with the maximum being 99.89% by MLP. Additionally, this pattern was the only one in the experiment with an FPR below 0.1%, indicating strong classification performance. MLP consistently achieved the best F1-score compared to the other algorithms across all feature configurations.

In addition to MLP, LR, RF, AdaBoost, and SGD also achieved high F1-scores. Focusing on the computation time of these classifiers, both the training and testing times of SGD and LR were very short when the number of features was 30. In particular, the training time of SGD was 2.350 seconds, which is less than one-fifth of LR’s training time, indicating that it is a well-balanced classifier in terms of classification performance and computational efficiency.

On the other hand, the methods using AdaBoost and RF showed high F1-scores and good performance in other scores as well, but their training and testing times were longer compared to the aforementioned methods. In this experiment, there appears to be no strong reason to adopt AdaBoost and RF in practical.

4.4 Comparison with Related Work

To ensure a fair comparative evaluation and assess the practicality of the proposed classification method, we compare our approach with the work conducted by Bagui et al. (2023) [3] under identical conditions.

Table 3 shows the results of SGD, LR, and MLP with 30 features from our method, as mentioned in Section 4.3, along with the results of RF and DT, which demonstrated the best performance in the related work, specifically the cases with the shortest training and testing times for each. Our method did not exceed the F1-scores reported by related studies, but the difference was no greater than 0.26%, indicating that the performance of our method is comparable to the results of previous research. In particular, MLP achieved a score of 99.89%, which closely approaches that of the related study.

Table 2: Evaluation Results for 35 Classifiers

Classifier		**Score (%)**					**Time (s)**	
ML Algo.	**Num. of Attrs.**	**Pr**	**Re**	**F1**	**Acc**	**FPR**	**Tr**	**Ts**
LR	18	98.63	98.64	98.63	98.64	0.7907	16.639	0.0056
LR	20	99.44	99.44	99.44	99.44	0.4232	7.428	0.0058
LR	25	99.41	99.41	99.41	99.41	0.3788	6.649	0.0058
LR	30	99.70	99.70	99.70	99.70	0.2244	11.997	0.0067
LR	35	98.92	98.95	98.93	98.95	0.7154	14.822	0.0064
NB	18	96.20	95.42	95.68	95.42	1.3400	0.313	0.0251
NB	20	97.25	97.01	97.08	97.01	1.1377	0.316	0.0280
NB	25	96.98	96.65	96.76	96.65	1.1926	0.319	0.0352
NB	30	97.58	97.43	97.48	97.43	1.1912	0.324	0.0534
NB	35	97.08	95.73	96.18	95.73	0.7515	0.345	0.0657
RF	18	99.09	99.09	99.09	99.09	0.6171	211.618	0.5113
RF	20	99.59	99.59	99.59	99.59	0.3236	195.465	0.4029
RF	25	99.58	99.58	99.58	99.58	0.3396	225.307	0.3990
RF	30	99.68	99.68	99.67	99.68	0.2815	229.616	0.3957
RF	35	99.25	99.26	99.25	99.26	0.5947	212.979	0.4702
DT	18	98.35	98.35	98.35	98.35	0.8613	13.039	0.0064
DT	20	99.24	99.24	99.24	99.24	0.5031	13.810	0.0060
DT	25	99.06	99.06	99.06	99.06	0.5504	16.320	0.0063
DT	30	99.34	99.34	99.34	99.34	0.4239	19.108	0.0060
DT	35	98.64	98.61	98.63	98.61	0.7058	20.449	0.0068
AdaBoost	18	99.08	99.09	99.09	99.09	0.5479	1212.757	2.0140
AdaBoost	20	99.63	99.63	99.63	99.63	0.2830	1353.234	1.9148
AdaBoost	25	99.65	99.65	99.65	99.65	0.2408	1660.297	2.0125
AdaBoost	30	99.77	99.77	99.77	99.77	0.1864	1977.570	2.0965
AdaBoost	35	99.21	99.23	99.21	99.23	0.5636	2330.708	2.1027
MLP	18	99.52	99.52	99.52	99.52	0.3117	207.393	0.3890
MLP	20	99.80	99.80	99.80	99.80	0.1470	50.957	0.0685
MLP	25	99.80	99.80	99.80	99.80	0.1042	82.075	0.0920
MLP	30	99.89	99.89	99.89	99.89	0.0897	83.896	0.1000
MLP	35	99.72	99.72	99.72	99.72	0.1271	125.101	0.1298
SGD	18	98.50	98.51	98.50	98.51	0.9613	1.592	0.0061
SGD	20	99.33	99.33	99.33	99.33	0.5842	2.010	0.0059
SGD	25	99.37	99.37	99.37	99.37	0.4236	1.322	0.0070
SGD	30	99.70	99.70	99.70	99.70	0.2187	2.350	0.0070
SGD	35	98.78	98.82	98.79	98.82	0.7700	3.683	0.0068

Pr: Precision, Re: Recall, F1: F1-score, Acc: Accuracy, Tr: Training Time, Ts: Test Time

Table 3: Comparison with Best Performance at Related Work

Research	**Classifier**		**Score (%)**		**Time (s)**	
	ML Algo.	**Num. of Attrs.**	**F1**	**FPR**	**Tr**	**Ts**
Bagui et al. (2023)[3]	RF	6	99.95	0.0012	35.9	0.099
Bagui et al. (2023)[3]	DT	12	99.96	0.0737	12.29	0.143
Our Method	SGD	30	99.70	0.2187	2.350	0.0070
Our Method	LR	30	99.70	0.2244	11.997	0.0067
Our Method	MLP	30	99.89	0.0897	83.896	0.1000

The abbreviations for the evaluation metrics are the same as those used in Table 2.

The next part compares the training and testing times of each algorithm. Firstly, the SGD algorithm has succeeded in significantly reducing both the training and testing times compared to the related research. Specifically, while the shortest training time in the related work was 12.29 seconds for DT, SGD managed to reduce the training time to 2.350 seconds, representing less than one-fifth of the training time required by DT in the related work. The testing time was also reduced to less than one-tenth of the shortest testing time in related research, which was 0.099 seconds for RF. Therefore, the case employing SGD can be considered a lightweight classification method with sufficient classification performance.

Next, the LR algorithm has also succeeded in reducing both the training and testing times compared to related research. Specifically, regarding the testing time, it has been reduced to less than one-tenth of the time for RF in related research, similar to SGD. Therefore, the case employing LR can also be considered a lightweight classification method with sufficient classification performance.

As for the MLP, the testing time is 0.1000 seconds, which is almost identical to the results in related research. On the other hand, the training time shows a significant difference compared to the DT in existing research. However, since the number of training iterations is assumed to be small compared to testing, a processing time of just under one minute would be well within a practical range.

In the related research, both LR and NB, regardless of the features used, did not achieve results comparable to RF and DT. However, as shown in Table 2, the F1-scores for all our classifiers range from 95.68% to 99.89%, and other scores also show stable results at similar levels.

In summary, when the input vectors contained 30 features, the MLP model achieved an F1-score of 99.89%, while both SGD and LR attained 99.70%. Notably, SGD exhibited exceptionally low computation times, with a training time of 2.350 seconds and a testing time of 0.0070 seconds. SGD outperformed the results of previous work by Bagui et al. [3], which utilized a distributed system to classify the same dataset in terms of computation time. Although LR demonstrated a comparable testing time to SGD, its training time was slightly longer at 11.997 seconds. In contrast, MLP required more time for both training and testing than SGD and LR. Based on these findings, we conclude that SGD strikes an effective balance between accuracy and computational efficiency, making it a practical and scalable solution for log classification tasks on a single workstation.

5 Discussion

This section describes the remaining challenges in our approach and potential solutions to be considered in future research.

5.1 Consideration of Deep Learning for Classification

Our method showed slightly lower F1-scores compared to those reported in related research. However, deep learning algorithms, which could potentially improve F1-score, were not adopted. This is because the primary objective of our approach is to propose a lightweight classification method. Compared to the machine learning algorithms used in our method, methods based on deep learning are expected to require significantly longer training and testing times. Furthermore, methods such as MLP and SGD demonstrated sufficient classification performance. For these reasons, deep learning algorithms were not included in the evaluation of our method.

5.2 Classification of Data with Small Sample Sizes

We focus on classifying only three tactics and normal communication, but the actual UWF-ZeekDataFall22 dataset includes nine additional tactics. Compared to the previously mentioned four categories, these nine tactics have relatively fewer samples and were not included in the classification. In particular, seven of the nine tactics contain fewer than 30 samples each. However, the remaining two tactics have more than 3,000 samples. Therefore, it is considered possible to classify these two tactics by applying oversampling techniques to increase the number of training samples. One possible approach involves vectorizing the original data using Doc2Vec and generating similar vectors to augment the dataset.

6 Conclusion

We proposed a lightweight method for classifying a large-scale dataset composed of communication logs by attack type. The classification employed machine learning algorithms including LR, NB, RF, DT, AdaBoost, MLP, and SGD, with feature vectors generated using Doc2Vec to numerically represent the data. In the experiment, for the purpose of evaluating the proposed method, data labeled with four types of MITRE ATT&CK tactics were extracted and classified from the UWF-ZeekDataFall22 dataset, which consists of Zeek connection logs linked to MITRE ATT&CK tactics. As a result of the experiment, MLP, SGD, and LR demonstrated sufficient classification performance and completed both training and testing at practical speeds. In particular, SGD required only 2.350 seconds for training and 0.0070 seconds for testing, both of which were exceptionally short. Based on these findings, we propose SGD as a lightweight and accurate log classification method.

References

[1] KDD Cup 1999 Data. Available online at `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999. Accessed: 2025-07-07.

[2] Sikha Bagui, Dustin Mink, Subhash Bagui, Tirthankar Ghosh, Tom McElroy, Esteban Paredes, Nithisha Khasnavis, and Russell Plenkers. Detecting Reconnaissance and Discovery Tactics from the MITRE ATT&CK Framework in Zeek Conn Logs Using Spark's Machine Learning in the Big Data Framework. *Sensors*, 22(20), 2022.

[3] Sikha S. Bagui, Dustin Mink, Subhash C. Bagui, Pooja Madhyala, Neha Uppal, Tom McElroy, Russell Plenkers, Marshall Elam, and Swathi Prayaga. Introducing the UWF-ZeekDataFall22 Dataset to Classify Attack Tactics from Zeek Conn Logs Using Spark's Machine Learning in a Big Data Framework. *Electronics*, 12(24), 2023.

[4] Quoc V. Le and Tomás Mikolov. Distributed Representations of Sentences and Documents. *CoRR*, abs/1405.4053, 2014.

[5] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, abs/1310.4546, 2013.

[6] MITRE. MITRE ATT&CK. Available at `https://attack.mitre.org/`, 2015. Accessed: 2025-07-07.

[7] MITRE. MITRE ATT&CK "Discovery". Available at `https://attack.mitre.org/tactics/TA0007/`, 2018. Accessed: 2025-07-07.

[8] MITRE. MITRE ATT&CK "Reconnaissance". Available at `https://attack.mitre.org/tactics/TA0043/`, 2020. Accessed: 2025-07-07.

[9] MITRE. MITRE ATT&CK "Resource Development". Available at `https://attack.mitre.org/tactics/TA0042/`, 2020. Accessed: 2025-07-07.

[10] Snehal A Mulay, PR Devale, and Goraksh V Garje. Intrusion detection system using support vector machine and decision tree. *International journal of computer applications*, 3(3):40–43, 2010.

[11] University of West Florida. UWF-ZeekData22 Dataset. Available at `https://datasets.uwf.edu/`, 2022. Accessed: 2025-07-07.

[12] Mrutyunjaya Panda and Manas Ranjan Patra. Network intrusion detection using naive bayes. *International journal of computer science and network security*, 7(12):258–263, 2007.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] KVR Swamy and KV Lakshmi. Network intrusion detection using improved decision tree algorithm. *International Journal of Computer Science and Information Security*, 10(8), 2012.