Received 11 February 2019
Revised 3 May 2019
Accepted 7 May 2019

A hybrid approach for log signature generation

Prabhat Pokharel
Department of Graduate Studies, NCIT, Lalitpur, Nepal, and
Roshan Pokhrel and Basanta Joshi
Department of Electronics and Computer Engineering, IoE, Lalitpur, Nepal

Abstract

Analysis of log message is very important for the identification of a suspicious system and network activity. This analysis requires the correct extraction of variable entities. The variable entities are extracted by comparing the logs messages against the log patterns. Each of these log patterns can be represented in the form of a log signature. In this paper, we present a hybrid approach for log signature extraction. The approach consists of two modules. The first module identifies log patterns by generating log clusters. The second module uses Named Entity Recognition (NER) to extract signatures by using the extracted log clusters. Experiments were performed on event logs from Windows Operating System, Exchange and Unix and validation of the result was done by comparing the signatures and the variable entities against the standard log documentation. The outcome of the experiments was that extracted signatures were ready to be used with a high degree of accuracy.

Keywords Log message, Named entity recognition, Density-based spatial clustering, Similarity measure, Support vector machine

Paper type Original Article

1. Introduction

A Log message is generated for an action performed on a computer system, device or an application. A computer program generates a log message by printing the message with an applicable log level. The log level defines the severity of the log message. This message is forwarded to the logging unit of the respective system, device or application. A Syslog server or an agent forwards the log message to a central log collection server. Log messages [1] are used for various day to day activities such as system and network troubleshooting, performance optimization, auditing actions of users and other objects, and to perform advanced analytics for issues in security, behavior analysis, incident management, operation, and regulatory compliance. Initial processing of log messages is required before the actual log analysis. This initial processing requires classification and parsing of log messages in the form of key-value pairs.

Publishers note: The publisher wishes to inform readers that the article "A hybrid approach for log signature generation" was originally published by the previous publisher of *Applied Computing and Informatics* and the pagination of this article has been subsequently changed. There has been no change to the content of the article. This change was necessary for the journal to transition from the previous publisher to the new one. The publisher sincerely apologises for any inconvenience caused. To access and cite this article, please use Pokharel, P., Pokhrel, R., Joshi, B. (2019), "A hybrid approach for log signature generation", *Applied Computing and Informatics*. Vol. ahead-of-print No. ahead-of-print, https://10.1016/j.aci.2019.05.002. The original publication date for this paper was 14/05/2019.

Applied Computing and Informatics
Vol. 19 No. 1/2, 2023
pp. 108-121
Emerald Publishing Limited
e-ISSN: 2210-8327
p-ISSN: 2634-1964
DOI 10.1016/j.aci.2019.05.002

Log management and SIEM [2] solutions are the most popular applications used for advanced log analysis. A SIEM system is centrally located and ingests log messages from all kinds of data sources. The ingested data is parsed, tagged with relevant information, stored and indexed. The indexed data is used searching, filtering, aggregation, correlation, alerting, reporting and also for advanced machine learning and behavior analytics. However, for all of these tasks, the log data should first be correctly parsed and indexed. And, parsing of these messages requires an understanding of the logging behavior of the respective system. The logging behavior brings some challenges for log parsing. The challenges are as follows:

- Log data repositories store log data from many different data sources such as in case of a data lake Figure 1. A data set practically contains log messages from many different types of log sources with multiple log categories contained within each source. Each log source has a different header and message part. The header part is always the same within a data source and in some cases resemble between different sources. The message part of each data source has multiple categories. So, if a log source in an average has 50 categories and there are 10 data sources, then the total number of categories is 500.
- Some of the log messages are generated very rarely. That is the probability of generation of such a log message is very low. Some example of such messages are changes to policy or even clearing of audit records. The occurrences of the events also depend on the type of system. For example, log message for system reboot or shutdown is not normal in critical servers.
- The log messages do not have uniformity in the available entity types. For example, one log message may contain a file name and not an IP address while the other may contain the IP address but not the file name.

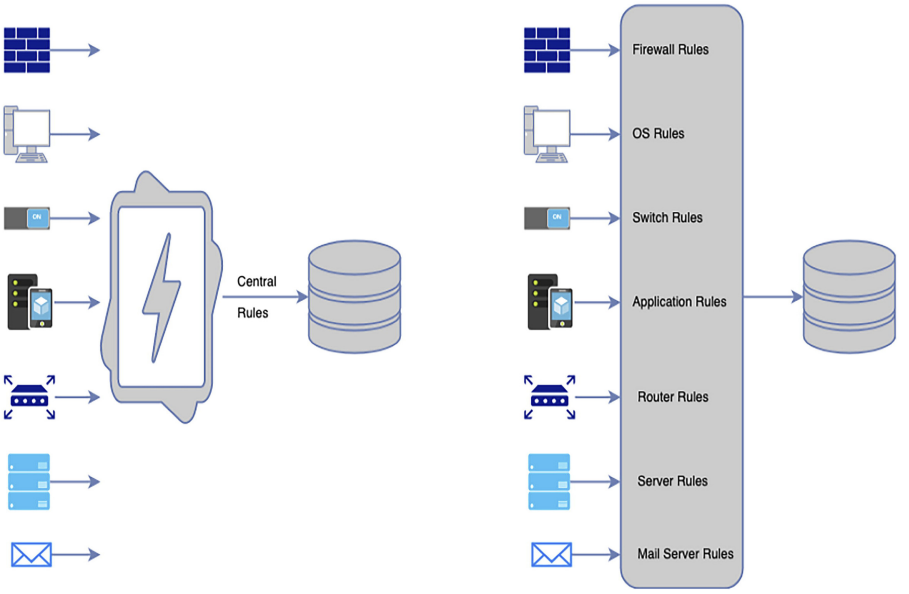

Figure 1.
Log collection with and without data lakes.

- There is no standard taxonomy for the definition of a named entity as there is no uniform protocol for defining the names of the available entities. For example, an IP address can have different names in different log sources.
- There is no uniform structure for defining the format of a log message and developers can generate logs as per their choice. Log messages may be composed of the regular sequence of words or also may be generated in formats like JSON, XML, CSV, etc.

To be able to solve the problem of log parsing through log signature generation we need to solve the above-mentioned challenges by understanding and utilizing the two important properties in messages. The first property is that every log message can be mapped to a log category. This mapping is specifically applicable for log messages belonging to the same source. The log categories are the result of the templates used for generating log messages. The second property is that each log message contains information about the entities involved for a given action performed. And, the values for these entities vary between log messages within the same category. The sequence of strings besides the variables are fixed within a category and can be termed as constants.

Parsing of log messages is typically done using signatures based on regular expressions. James E. Prewett [3] has explained the use of the regular expression to analyze log messages. Risto Vaarandi et al. have used LogCluster [4] tool to propose a framework for identification of anomalous log messages. Similarly, Risto Vaarandi has carried a number of works on clustering and pattern identification in log messages. A number of works have been conducted to compare the performance of LogCluster [5,6] with other publicly available tools. LogCluster is based on frequent itemset mining along with and SLCT [7] which was one of the first tools for log clustering and signature generation. His approaches rely on the fact that within a cluster the non-varying parts is constant whereas the varying parts represent the variable entities. These variable entities are replaced by wildcards to construct log templates. Thresholds should be defined for the frequency of the variable entities falling in each cluster. Improperly set thresholds can give undesired clusters and thus wrong templates. The major limitation can be seen when the varying (variable) entities do not change and remain constant for the samples within the pattern. In such a case, the varying entities cannot be identified. Thus, the formation of a signature is not possible. In recent years some other methods of log pattern identification have also been devised. Such a method for log template generation based on Natural Language Processing was proposed by Satoru Kobayashi et al. [8]. The method showed promising results for variable identification. However, the accuracy for log template generation is fairly moderate, thus leaving sufficient space for improvement. Similarly, some other works such as search based log clustering have been implemented by Basanta Joshi et al. [9], where a log message is compared with a universal set of signatures.

In recent years' new approaches on log parsing and template generation have been conducted. Pinjia He et al. [10] have implemented an online log parsing tool using a tree with a fixed depth. It consists of 5 different steps. The first step uses regular expressions to match the required messages or sections of the log message. The second step counts the number of tokens to create groups. This means log messages with the same length will fall into a group. The third step creates new groups based on the tokens at the beginning of the log messages. The fourth step calculates the similarity between the log group and the actual log message to arrange similar log messages into a group. The fifth step updates the group if the log messages match to the group else creates a new group. This approach has some practical challenges such as log messages with the same length and same or similar string at the beginning of the message can have a higher chance of generating a common parser. While at other times log messages with different lengths will always generate different parsers, which is not always correct. This can be observed particularly when the length of the variable is subject to change. Similarly, for rare log messages where the system generates only one log

message for a given category, the variables can't be identified. Salma Messaoudi et al. [11] have implemented a search-based approach using a genetic algorithm to generate log templates. Hamooni et al. [12] have implemented a tool named as LogMine based on clustering to identify patterns from log messages. Wei Xu et al. [13] have conducted console log mining using the source code analysis for problem identification.

Most of the earlier works in log signature generation are based on clustering and frequent itemset mining. However, there have been works based on parsing tree or machine learning as well. The existing research works have both advantages and limitations. Thus, the idea behind our work is to reduce the limitations of the existing approaches, with a focus on the practical challenges observed in log message parsing.

In this paper, we present a new concept on log message parsing. This concept combines machine learning with log clustering which is one of the trusted methods. The most important aspect of this paper is that it presents a concept to solve the practical challenges for log signature generation and log parsing particularly in relation to log management. Here, we use log clustering for pattern identification and NER for variable recognition. In the remaining section of the paper, we detail the examples of motivation and implementation details of the hybrid approach. Furthermore, the remaining sections outline the experiments performed, conclusions around the results obtained and areas for future work.

2. Materials and methods

2.1 Motivation

Log management systems collect log data from many different sources and each of these sources contain multiple log categories. A log category contains similar log messages which can directly be mapped to a log signature. We adopt clustering to identify a log category and group the log messages together. A correctly identified log message and falling inside a log category can be used as a seed for signature generation. A log category is composed of a defined language pattern. However, this language pattern may not resemble the natural language. For example, "Connection allowed from ⟨host/IP⟩" and "Connection denied to ⟨host/IP⟩". These examples show that entity names may occur after the words to or from. Similar kinds of rules are observed in other log messages. Thus, these rules can be used to identify the constant parts and the variable parts in the log messages. A signature will represent the constants as in the original strings and variables by a fixed pattern. Here, Figure 2 shows a regular log pattern. Most of the log signature generation techniques are based on the concept that within a pattern the varying strings are the entities and the remaining strings are the constants. However, this may not always be the case. There are some limitations to the assumptions made by most of the existing approaches. These are:

- Variable entities do not change within a given dataset. In the example to the right in Figure 3 connection requests end up on the same destination IP address or destination host. In this case, all the destination IP addresses or destination hosts will have the same value and thus interpreted as a constant.
- If the data set contains some rare log samples, with one log message per category, the constant tokens between different categories may be interpreted as variables. In the example to the left in Figure 3 we see that failed and successful are represented as variables though they are actually the constants.
- In approaches such as parsing tree, it is assumed that log messages with different lengths can fall into different categories and thus generate different signatures. However, this cannot always be correct. The absence of a variable entity and change in number tokens used for a variable entity is widely observed behavior.

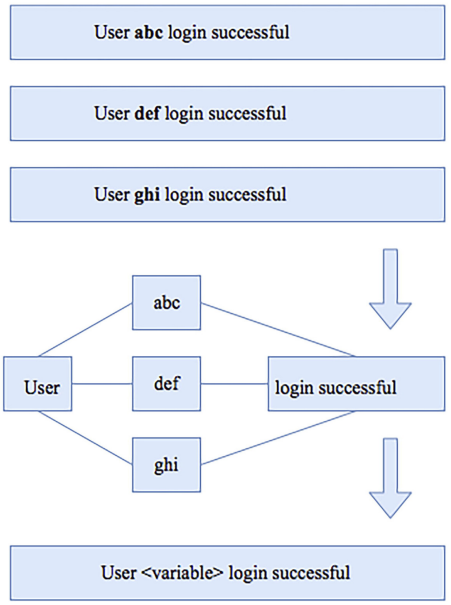

Figure 2.
Regular log pattern.

- It is also observed that certain sequences in the log message are considered more significant in determining a category which again is not correct as most of the log message can have the same or similar header information. In the case of parsing tree sequence at the beginning of the log message is considered for identifying a category. This can cause the algorithm to fail in many cases.

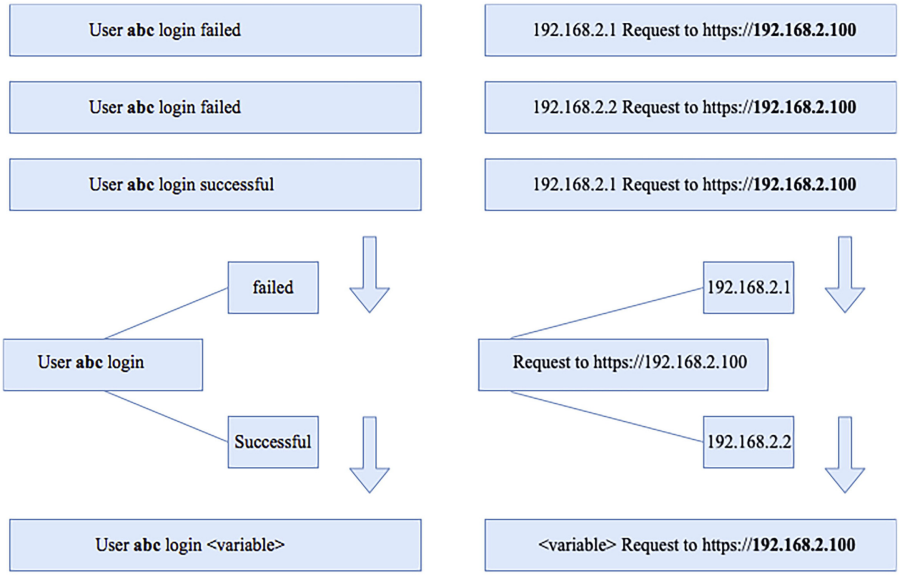

Figure 3.
Log patterns with exceptions.

To solve these limitations, we adapt to use log clustering followed by NER. Clustering is used to identify log patterns. While NER is used to identify the constant terms and variable terms.

2.2 Related algorithms

Clustering is an unsupervised machine learning approach. It is used to create various sub-groups from a group of data points. The data points in a sub-group are such that they are more similar to the other data points within the same sub-group compared to the other sub-groups. Clustering is very efficient for identifying unique log patterns from a given collection of log messages. Our approach uses Density-Based Spatial Clustering (DBSCAN) for the clustering problem. This is because it is able to solve the following problems:

- Outlier samples should be considered for signature extraction. Outlier samples are the samples, which do not fall within the defined similarity criteria. Every batch of data set can have some set of log samples as outliers. If these outlier samples are included in the classifier model, thus creating an under-fitting model.
- As it is practically a difficult problem to know in advance the number of log categories in a batch of log samples. Thus, we want the approach to be able to identify the number of clusters on its own.

We use the cosine similarity measure as a measure of similarity. This is because with this the accuracy of a cluster can be determined by the frequency of the words. Similarly, samples with the same length can fall within multiple clusters. Thus, cosine similarity [14] is a good fit for measuring the similarity between log samples.

NER is used in areas such as classification of entities in documents, news, and web, recommendation systems, search optimization and so on. These systems deal with standard natural language structure. Log messages do not precisely follow standard natural language constructs. Thus, the Parts of Speech (POS) tagging for log messages is not completely aligned to the definition and context given by the natural language. This creates a need to adjust the POS tagging as per the structure of the log messages. Our approach uses Support Vector Machine (SVM) [15] to build the classifier model for extracting named entities. SVM is a supervised machine learning approach to classify a given set of data points. It identifies a hyperplane that classifies the data points with a maximum margin. If f(x) is a classification function. Such that $\times \Rightarrow$ {true, false} for training data points $\times *$ {true, false}. f(x) = true if the test data point falls in the desired class. And f(x) = false if the test data point falls outside the desired class. The choice SVM in our approach is due to the fact that we want higher performance and accuracy with limited labeled samples. Also making the use of preceding and succeeding tokens as features and not just the actual token.

2.3 Proposed approach

We have devised a hybrid approach, which uses clustering for log pattern identification and a classifier model for NER. The approach can be broken down into two modules. The first module identifies patterns by creating clusters from a batch of log messages. A pattern contains log messages falling within a category or an event type. The second module consumes the log patterns to build the classifier model and then extract the named entities. A signature is generated as an output of the second module, which consists of static and variable sections. The signature as a whole represents a log pattern with both variables and constants. Here in Figure 4, we see the block diagram for the overall signature generation approach. The input consists of a batch of n log messages. The log batch may contain log messages from one or more sources. The log messages are broken down into bigram tokens and then sent for a similarity check with the seed. Log messages falling under a defined threshold are grouped together as a cluster or a pattern. The logs are further used to prepare a

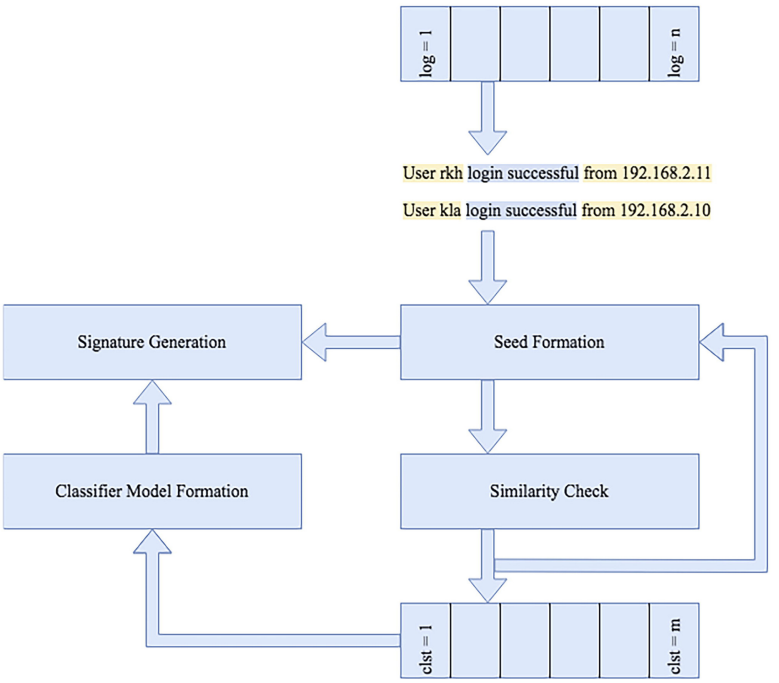

Figure 4.
Signature generation approach.

classifier and identify the named entities. Important parameters used in the proposed approach are listed in Table 1.

2.3.1 Pattern identification. The pattern identification module uses clustering to identify log patterns. Every log message is broken down into bigram tokens and compared with a seed log message. The log messages are appended to a cluster if the angle of separation between them is within a specified threshold limit. When there are more than two log messages within a cluster (clst), the seed is replaced by the new log message having the smallest angle of separation. The threshold angle is adjusted to minimize the cluster pattern error. This is ensured by validating that the outlier does not belong to the set of clustered patterns and no two outliers are similar. Thus, only a log message which does not fall into an

Parameter	Description
N (Number of log samples)	Represented by {1, 2 . . . N}. If n is an instance $n \in \{1, 2 \ldots N\}$
M (Number of log categories)	Represented by {1, 2 . . . M}. If m is an instance $m \in \{1, 2 \ldots M\}$
P (Number of log patterns)	Represented by {1, 2 . . . P}. If p is an instance $p \in \{1, 2 \ldots P\}$
O (Number of outlier logs)	Represented by {1, 2 . . . O}. If o is an instance $o \in \{1, 2 \ldots O\}$
T (Threshold)	Optimal threshold for an extracted pattern in the range of 0 to 1
CPE (Cluster pattern error)	Ratio of number of clusters to number of patterns. $CPE = P/M$
AA (Adjusted accuracy)	Accuracy obtained by removing outlier categories from the data set. $AA = P/(M - O)$
SA (Signature accuracy)	Accuracy of the signatures generated by using the outlier categories. $SA = (P + O)/M$

Table 1.
Key parameters used in the proposed approach.

existing cluster is considered as an outlier. As each of these outliers is represented as an independent pattern and thus these are passed to the second module to extract the named entities.

2.3.2 Pattern identification algorithm.

- For a given data set that contains N messages and M categories where each message is represented by n and each category by m
- Each n log message is represented as a text vector for given word bigrams
- $n \in \{1, 2 \ldots N\}$ and $m \in \{1, 2 \ldots M\}$
- This generates P patterns and O outliers
- $p \in \{1, 2 \ldots P\}$ and $o \in \{1, 2 \ldots O\}$
- For each p there are more than one log messages with one seed message.
- For each o there is one and only one log message which can act as a seed.
- If N is the input and P, O are outputs
- Define a threshold T for which the log vectors fall into a category if their angle of separation is less than the angle of threshold (0<=T<=1)
- Such that $M \approx P$, $P \cap O = \varnothing$, $P + O = M$ and $o \notin P$, $o \in M$

2.3.3 Entity recognition and signature generation. The entity recognition module ingests the log patterns in the form of log clusters. The log messages in the clusters are first used to train the classifier model. The entity recognition module consists of the following steps:

2.3.3.1 Labeled data preparation. The varying (variable) entities are extracted from each cluster by using a proven approach. During the process, we used a standard application [16] for parsing log messages. Furthermore, the results from the application are validated by following the standard log documentation. This data is used for training the model.

2.3.3.2 Preprocessing. Preprocessing does the cleaning of the data samples by removing unwanted characters and tokens. Tokens are created by splitting the log message on whitespace characters. Non-alphanumeric characters from the tokens are removed. The patterns for DateTime format, IP address, and, MAC address is replaced with definitions for each of these types.

2.3.3.3 Feature extraction. Feature extraction is done on the preprocessed tokens. This is done by selecting trigrams in order to preserve the context of a given log sequence. The selection of features considers the use of preceding and succeeding tokens from the selected token. The preceding value is represented by $(n - 1)$, the actual value by n and the succeeding value by $(n + 1)$. POS tagging for each identified token is done before using it as a feature. Lemmatization process is not required as we want our model to preserve the tokens as they are and change in the tokens is sensitive for the final outcome. Following features are used for model preparation:

- POS tag of $(n-1)^{\text{th}}$ token
- POS tag of $(n+1)^{\text{th}}$ token
- POS tag for n^{th} token
- Value of $(n-1)^{\text{th}}$ token
- Value of $(n+1)^{\text{th}}$ token
- The shape of n^{th} token

2.3.3.4 Classifier model formation. Classifier model based on the SVM is constructed using the extracted features.

2.3.3.5 Signature generation. The seed log message for each pattern or an outlier is compared against the model to generate the signature. As the variable entities are extracted, these entities are replaced by variable tags.

2.3.4 Signature generation algorithm.

- **Inputs**: A seed or an outlier log message An empty signature list
- From the log message remove any unwanted characters and replace standard patterns with their definitions
- Split the log message into a list of tokens
- For each token in the list
- Check for a variable entity
- If True: Replace the token with a variable tag and append to the signature list
- If False: Append the token to the signature list
- **Output**: Updated signature list

3. Experiment

3.1 Dataset

Two sets of data are used during the experiment. The first set is synthetically prepared and only used during the research phase for finalizing the algorithm. The second set consists of log dumps from 6 different instances of the Windows Operating System as shown in Table 2. The dumps contain log samples consisting of categories such as successful login, failed login, logoff, successful authentication, failed authentication, account created, account deleted and so on. Sample number 5 contains log messages from Windows and Exchange while the sample number 6 contains log messages from Unix Server as well. The reason for combining three data sources is for evaluating the dependency of the accuracy on the header section or the beginning tokens. Log messages in Windows and Exchange have the same beginning tokens while the header in Unix is different from that of Windows and Exchange. So, with this, we are able to validate our model for two special cases. The first is that a log collection server receives more than just one data source. And the second is that even if the data sources are different, their headers or beginning tokens are similar. We used application [16] with log parsing capability to label the data with identified variable entities. Standard event log documentation [17] is referred for re-verifying the correctness of the results returned by the application. This gives the number of log categories and log messages belonging to each category. The labeled data is used for pattern identification

Table 2.
Data samples.

Sample	Categories	Message Count	Average Tokens	Data Source
1	104	10,400	70	Windows
2	98	11,020	69	Windows
3	67	4599	64	Windows
4	46	9021	66	Windows
5	109	121,503	69	Windows, Exchange
6	121	145,090	68	Windows, Exchange and Unix

and validation against the number of available categories and also for training and testing of extracted named entities.

3.2 Results and discussion

Independent evaluations were done on the results for the accuracy of the signatures and the accuracy of the variable entities. Validation of the signatures was done by comparing with the available categories in the training data set as shown in Figure 5. While the validation of the variable entities was done by testing the classifier models by performing 10-fold cross-validation on the training datasets. The performance for the correctness of variable entities was measured in terms of accuracy, precision and f measure. The correctness of the signatures was evaluated by calculating the cluster pattern error and signature accuracy. The cluster pattern error is the ratio between the expected number of patterns and extracted patterns. While adjusted accuracy is the accuracy obtained by removing the outlier messages from each sample. The outliers are the single message containing patterns as a result of rare log samples in the data set. Finally, we calculate the signature accuracy at the end of the signature generation process, which considers signatures generated from both the clustered patterns and the outliers.

3.2.1 Signature accuracy. We represent the accuracy of generated patterns and the signature in three different ways, which helps us to highlight the advantage of the hybrid approach. The cluster pattern error is the ratio of the number of patterns received after the clustering to the available number categories. Adjusted accuracy is the accuracy obtained by removing the single log instances from the given dataset. Signature accuracy is the final accuracy of the signature generation process. It is the ratio between the extracted signatures and the available number of log categories. For the six different samples, the maximum observed error for pattern identification in the clustering module is 26.44% as shown in Table 3. There is not much change in cluster pattern error as the number of log samples is changed. However, comparing the results from all of the six samples, it is observed that the cluster pattern error increases as the number of log categories are increased. The first sample classifies 104 log categories into 80 patterns leaving behind 5 unclassified samples as outliers. Whereas the sixth sample has nine outliers as the output of the first module. Removing the outliers, the cluster pattern error for all of the samples is reduced and thus adjusting the signature accuracy. This shows that the cluster pattern error is high on samples with a higher number of outliers. Thus, we observe the least adjustment for samples with fewer outliers. The overall signature accuracy for all of the six samples is above 80% and the maximum observed value is 81.73%.

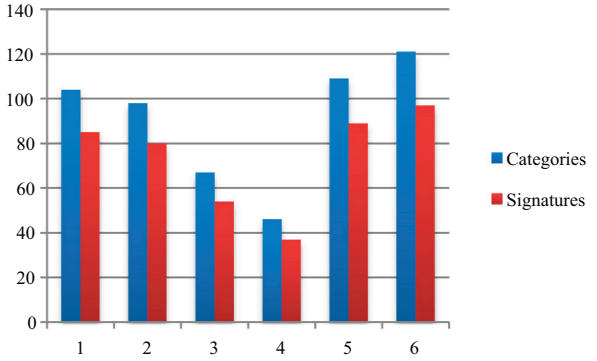

Figure 5.
Signatures extracted by log categories.

3.2.2 Variable accuracy. The accuracy of variable entities shows how correctly the variables are represented in the signatures. We calculate accuracy, precision and f measure as the parameters for measuring the performance. We observe low accuracy when the number of log samples is low. This is because of the improperly created models due to scarce data points. However, once the number of samples crosses a particular limit, increase in accuracy is seen. As in Table 4, all of the four samples exhibited accuracies above 99%. Higher accuracy is observed in sample 2 with fewer categories but with a higher number of log samples compared to sample 1. Similarly, sample 3 and sample 4 with fewer categories have even higher accuracies. While sample 5 and 6 both exhibit higher accuracies even if the number of log categories is higher it is because both of these samples have maximum data points required for model formation. However, for sample 6 we observe a slight decrease in accuracy due to the variation in language constructs between the logs in Windows and Unix Server. Figure 6 shows plots for accuracy, precision and f measure for sample 1,3, 5 and 6 respectively. The performance measures in all of the four plots show similar characteristics. What we observe is that the accuracy has a gradual rise to reach towards the saturation point. However, the precision and the f measure exhibit a relatively sharp change in slope.

3.2.3 Comparison with other approaches. We conducted experiments using LogCluster and Drain and compared the results with our approach performed on data sample 5 and 6. The obtained results are shown in Table 5. An accuracy of 67% was observed with LogCluster while Drain showed an accuracy of 79% on sample 5. The accuracy decreased to 61% and 73% respectively for LogCluster and Drain respectively for experiments on sample 6. The significant decrease in accuracy with Drain was observed on data sample 6. One main reason for it was that the log samples belonging to different categories had the same number of tokens with similarity at the beginning. On the other hand, in the case of LogCluster, the rare samples and frequent behavior of the variable entities was the main reason for the reduction in accuracy.

Table 3.
Signature accuracy by data samples.

Sample	Categories	Patterns	Cluster Pattern Error (%)	Outlier Categories	Adjusted Accuracy (%)	Signature Accuracy (%)	Precision (%)	F Measure (%)
1	104	80	23.08	5	80.80	81.73	88.23	91.45
2	98	76	22.45	4	80.85	81.63	87.50	90.90
3	67	52	22.39	2	80.00	80.59	88.88	91.42
4	46	36	21.74	1	80.80	80.04	89.18	91.65
5	109	82	24.77	7	80.39	81.65	89.88	91.42
6	121	89	26.44	8	78.76	80.16	87.62	90.41

Table 4.
Variable performance measures by data samples.

Sample	Accuracy (Max %)	Precision (Max %)	F Measure (Max %)
1	99.19	87.56	90.27
2	99.24	88.07	91.10
3	99.48	89.15	96.23
4	99.67	90.17	96.13
5	99.69	90.70	92.10
6	99.57	90.19	92.50

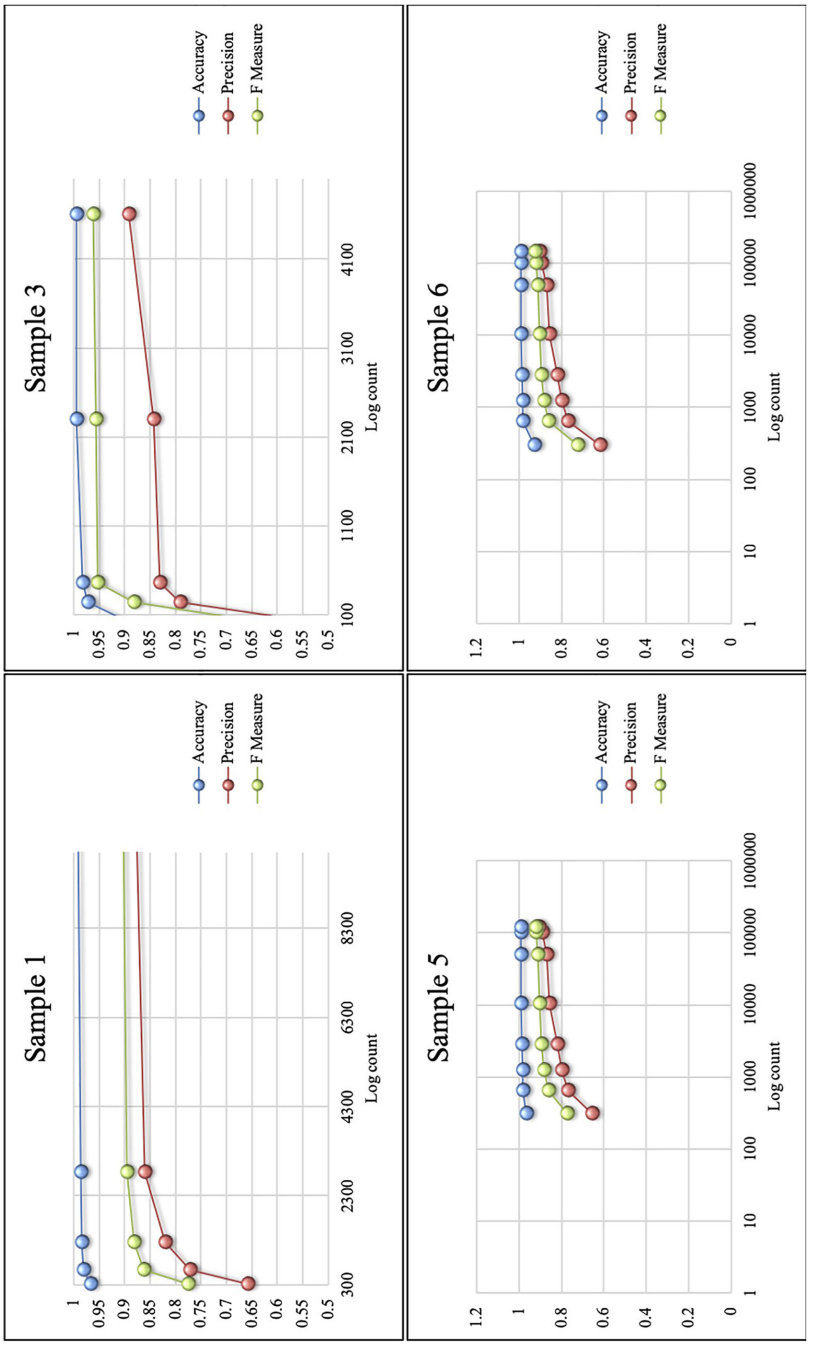

Figure 6.
Performance measures for variable entities.

Table 5. Comparison with other approaches.

Sample	Algorithm	Categories	Signature Accuracy (%)
5	Hybrid	109	81.65
	Drain		79
	LogCluster		67
6	Hybrid	121	80.16
	Drain		73
	LogCluster		61

4. Conclusion and future work

We devised a hybrid approach for signature generation from log messages. We performed experiments on log data from Windows, Exchange and Unix systems to verify that the concept of the hybrid approach can be very effectively used to generate signatures. Our approach uses similarity-based log clustering for log pattern identification and then generates signatures by recognizing variable entities using the SVM based classifier model. As signatures are generated per log cluster, each of these signatures represents a unique log pattern, which in turn maps to a log category. At times there are single log instances for certain patterns, these instances are not grouped into clusters but listed as outliers. However, both clusters and outliers serve for the actual variable extraction and signature generation. In certain cases, a single signature is generated for two or more categories if the pattern for two or more categories is identical. The research work has a number of areas for improvement. One area of improvement is on signature accuracy. Similarly, our approach uses data sets in batches. This can also be extended to work on streaming datasets. Another important area for future work could be to try new algorithms both for log clustering and NER. Similarly, the current scope of the research focuses on the evaluation of the concept of the hybrid approach with experiments conducted on three different data sources. This can be extended in the future by performing experiments on a wide number of data sources.

References

[1] K. Kent, M. Souppaya, Guide to Computer Security Log Management, NIST, Gaithersburg, 2006.

[2] R.D. Miller, S. Harris, A. Harper, S. VanDyke, C. Blask, Security Information and Event Management (SIEM) Implementation, McGraw Hill Professional, New York City, 2011.

[3] E.J. Prewett, Analyzing cluster log files using Logsurfer, Citeseerx (2003) 1–11.

[4] R. Vaarandi, B. Blumbergs, M. Kont, An unsupervised framework for detecting anomalous messages from syslog log files, IEEE Xplore (2018) 1–6.

[5] R. Vaarandi, M. Kont, M. Pihelgas, Event log analysis with the LogCluster tool, IEEE Xplore (2016) 982–987.

[6] R. Vaarandi, C. Zhuge, Efficient Event Log Mining with LogClusterC, IEEE Xplore (2017) 261–266.

[7] R. Vaarandi, A data clustering algorithm for mining patterns from event logs, IEEE Xplore (2003) 119–126.

[8] S. Kobayashi, K. Fukuda, H. Esaki, Towards an NLP-based log template generation algorithm for system log analysis, Assoc. Comput. Mach. (2014) 1–4.

[9] B. Joshi, U. Bista, M. Ghimire, Intelligent clustering scheme for log data streams, Comput. Linguist. Intel. Text Process. (2014) 454–465.

[10] P. He, J. Zhu, Z. Zheng, R. Lyu, M., Drain: an online log parsing approach with fixed depth Tree, IEEE Xplore (2017) 33–40.

[11] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, R. Sasnauskas, A Search-based approach for accurate identification of log message formats, Assoc. Comput. Mach. (2018) 167–177.

[12] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, A. Mueen, LogMine: fast pattern recognition for log analytics, Assoc. Comput. Mach. (2016) 1573–1582.

[13] X. Wei, H. Ling, F. Armando, P. David, I.J. Michael, Detecting large-scale system problems by mining console logs, Assoc. Comput. Mach. (2009) 117–132.

[14] S. Al-Anaz, H. AlMahmoud, I. Al-Turaiki, Finding similar documents using different clustering techniques, Proc. Comput. Sci. (2016) 28–34.

[15] Z. Ju, J. Wang, F. Zhu, Named entity recognition from biomedical text using SVM, IEEE Xplore (2011) 1–4.

[16] C. David, Getting Data, in: C. David (Ed.), Exploring Splunk, CITO Research, New York, 2012, pp. 13–19.

[17] Microsoft. (n.d.). Advanced security audit policy settings. Retrieved from Advanced security audit policy settings: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/advanced-security-audit-policy-settings.

Corresponding author
Prabhat Pokharel can be contact at: prabhat@ncit.edu.np