Improving performances of log mining for anomaly prediction through NLP-based log parsing

Nicolas Aussel, Yohan Petetin, Sophie Chabridon

Improving Performances of Log Mining for Anomaly Prediction through NLP-based Log Parsing

Nicolas Aussel*†, Yohan Petetin†, Sophie Chabridon†
*Zodiac Inflight Innovations, Wessling, Germany
†SAMOVAR, Télécom SudParis, CNRS, Université Paris-Saclay, France

***Abstract*—Failure prediction of industrial systems is a promising application domain for data mining approaches and should naturally rely on log messages which are a prime source of data as they are generated by many systems. However, before extracting relevant information of such log messages, another critical step is to parse the logs, that is to say to transform a raw unstructured text from the log messages into a suitable input for data mining. These two problems (log parsing then log mining) are often studied separately while they are directly related in the context of failure prediction ; moreover, few performance benchmarks are publicly available. In this paper, we focus on the impact of log parsing techniques via natural language processing on the performances of log mining on two datasets. The first one is a log of an industrial aeronautical system comprising over $4,500,000$ messages collected over one year of operation ; the second one is a public benchmark set from an HDFS cluster. On the latter, we show that it is possible to raise the F-score from 96% to 99.2% while using simpler and more robust log parsing techniques that require less parameter tuning provided that they are correctly combined with log mining techniques.**

I. Introduction

The usage of logging systems is one of the most widespread solution for monitoring complex systems and applications. One such example would be the syslog of Linux machines. Such logging systems offer both a high level of flexibility for developers and a high level of expressiveness for users making them very useful to understand potential anomalies. However the high volume of log messages generated by complex systems makes it difficult for human operators to effectively monitor every event. This often leads to a reliance on high level signalling of the gravity of log messages such as "information","warning" and "error". This system has the obvious weakness of being only accurate for events and situations that were accounted for at the time of the implementation of the logging system. In a situation of interaction between multiple subsystems developed independently, the signalling might not be representative of the state of the entire system. That is an "error" message from a subsystem could be the result of a normal operation of the system, for example a subsystem reporting a loss of connectivity when the network services are being reset at the system level. Conversely, a "normal" message from a subsystem could indicate an anomaly in the system, for example, a subsystem reporting communication taking place when the network services are shutdown at the system level. As a result of this limitation and of the high volume of log messages, in practice, many logging systems are often limited in their use as tools for postmortem diagnostics.

In order to enable better usage of logs, many data mining techniques have been adapted to automate log analysis. This process can be split in two main steps: i) how to transform the raw unstructured text from the log messages into a suitable input for data mining; this is called log parsing; ii) how to automatically exploit this structured information with data mining techniques; this is called log mining. Log mining has been the subject of many publications [1], [2], [3], log parsing as well [4], [5]. However, until recently there was no benchmark of publicly available datasets and implementations to systematically evaluate the performances of the log parsing methods. As a result, the interactions between log mining and log parsing have been only partially explored. Moreover, as logs are automated messages, even raw messages display repetitive patterns. Consequently, the first parsing approach adopted has often been rule-based and aimed at distinguishing between constant parts and variable parts to categorize log messages. However, the log messages are made of sentences or partial sentences understandable by a human operator and as such it is also possible to take into consideration their semantic aspect in order to parse them and categorize them. Thus it would make sense to investigate the use of Natural Language Processing (NLP) which is surprisingly scarcely studied [10], [11] and absent from state-of-the-art surveys such as [6].

The goal of this article is to study the influence of a selection of log parsing techniques from the field of NLP on the efficiency of a fixed log mining process aimed at providing log-based failure prediction of an aeronautical system. The results are evaluated on an industrial dataset collected on live systems and, in the end, the best log parsing method found is evaluated on a new benchmark dataset and compared to existing solutions found in [6]. We find that our solution achieve substantial performance increase on the benchmark dataset while also being more robust and easier to parametrize than state-of-the-art solutions.

The paper is organized as follows. First, we study existing works in log parsing and log mining in section II. Next, we detail the log parsing techniques that we implemented in section III. We explain the log mining process that we use in section IV. We present our results and interpret them in section V. Finally, VI concludes the paper.

II. Related work

There are many studies pertaining to both log mining and log parsing. Some of the recent works include [1], [2], [3]. In [1], the authors first parse the constant and variable parts of the logs messages using either the source code of the logging application or the messages themselves then filter the messages on frequency and perform anomaly detection with using Principal Component Analysis (PCA), treating outliers as anomalies.

In [2], the authors filter out the variable parts of log messages using manually defined rules and cluster the rest with a similarity measure based on a custom weighted edit distance. They model the sequence between messages in a Finite State Automaton and perform anomaly detection by identifying sequences where impossible transitions are made.

In [3], the authors first distinguish between state messages and event messages before engineering features from the results of statistical tests run on each distribution separately. Finally, the anomaly detection is performed by analysing outliers.

[4] describes a log parsing method that is still widely used to this day based on multiple passes over the logs to determine the most frequent words and the messages in which they occur.

[7] relies on Multiple Instance Learning (MIL) to predict hard drive failures. Daily aggregates are used to simplify the model. Even though this appears promising for reducing the amount of data, it is not applicable to all kind of data and requires a preliminary domain analysis.

[5] describes another state-of-the-art log parsing algorithm, IPLoM. It is of a particular interest for this study since it has been found in [6] to be the most effective log parsing algorithm available for the HDFS benchmark dataset which shares several characteristics with the industrial dataset that this study is based on and in particular its size and the distributed nature of the system being studied. IPLoM works by performing several steps of hierarchical partitioning of the log messages before splitting them in constant and variable parts.

The study [6] reviews many of the recent work on log mining and log parsing and points out that very little is available in terms of benchmark be it from the perspective of the datasets or of the implementation of the log parsers. This leads to difficulties in performance evaluation for researchers designing new methods and difficulties in applying log mining methods for potential users. This study also highlights the fact that the existing log parsing solutions are not distributed and tend to be very costly to run on large datasets in terms of execution time. Subsequent studies from the same authors in [8] and [9] propose two new log parsing algorithms POP and Drain. POP is based on recursive partitioning and optimized for low time complexity and Drain is an online algorithm which uses fixed depth parse tree. Though they both perform extremely well for log parsing with performances nearly optimal, once evaluated on a log mining task benchmark, their performances are similar to those of IPLoM. Finally, it is worth noting that the proposed methods are all rule-based and none of them are based on Natural Language Processing.

The article [10] presents an approach based on the NLP method of word embedding for log parsing, sharing our observation that off-the-shelf NLP techniques could potentially yield significant improvement over traditional rule-based log parsing. The authors then use three different classifiers for log mining, Random Forest, Naive Bayes and Neural Network, evaluated on one of the public benchmark datasets from [6] that we also use in the present article reaching up to 90% accuracy with the Random Forest technique.

In [11], the authors propose a parsing algorithm that uses detailed semantic analysis and custom dictionaries defined through statistical analysis of their text corpus, taken from logs of several servers at their disposal. Evaluation is then conducted on the same servers rendering comparisons with benchmark performances difficult.

[12] describes a log parser called Spell for Streaming Parser for Event Logs using LCS. It is based, as its name implies, on the Longest Common Subsequence (LCS) algorithm run in a streaming fashion to identify constant and variables part of log messages and classify them into message types. It does not however contain an evaluation on log mining metrics but only on log parsing. In [13], the Spell log parser is used in combination with a Long Short-Term Memory (LSTM) deep neural network to evaluate the full pipeline on two public datasets, including the HDFS benchmark that we also use, and reaching up to 96% F-measure on it. It is worth noting that the deep learning approach used is considerably more complex to implement and parametrize than our proposed solution.

III. Log parsing

Log parsing is understood here as the combination of methods used to transform the unstructured character strings that compose the log messages in a structured form suitable for failure prediction. Since the character strings form sentences or partial sentences understandable by a human operator, we take a special interest in methods from the field of Natural Language Processing (NLP) which aim specifically at extracting structure from documents written in natural languages by opposition to formal languages. The goal of this section is to present several easy to implement log parsing methods that can be used to engineer the features necessary for log mining.

A. Tokenization

The first method that we apply is a tokenizer. Its role is to process a text and break it into individual words in order to enable comparisons between sentences. In our case, we split the words based on whitespaces and punctuation marks and convert the characters to lower case. As an example, the sentence ”Temperature too high: the CPU is overheating” would get broken down into the words ”temperature”, ”too”, ”high”, ”the”, ”cpu”, ”is”, ”overheating”. It is worth noting that this tokenizer has difficulties especially with contracted English forms such as ”don’t” or ”it’s” in which case other solutions based on regular expressions can be used. However

such forms are absent from our logs so these solutions were not considered.

B. Semantic techniques

The next methods we can apply on the individual words are stemming, synonym replacement and stopwords removal. Those methods are not strictly necessary to perform failure prediction as they simply aim to clean up the input text to reduce noise by looking into the meaning of the words considered. We study their impact on the prediction performances in section V. Stemming is the process of removing inflected forms at the end of words to find out the stem of the word. We use it to conflate similar words such as "failure" and "fail" which are semantically close and convey a similar sense under the same stem "fail". This allows us to reduce the vocabulary and increase the similarity between sentences whose meanings are close. In our case, we apply a standard stemmer, the English snowball stemmer from the nltk package [14].

With the synonym replacement method, we replace words by their most common lemma corresponding to their most common meaning based on the standard wordnet module of the nltk package. This allows us to conflate words with similar meaning but different stems that cannot be caught by the stemming method. An example of that would be the words "present" and "detected" that are quite common in the log messages, the synonym replacement method can conflate them to the same representation, reducing further the vocabulary.

Stopwords removal is the process of removing stopwords. Stopwords are words that are frequent but have a low relevance for the classification, one such example is the article "the". If kept, the stopwords will induce noise in the classification model as two unrelated sentences could have many such words in common. In our case, we use the standard list of English stopwords found in Apache Spark MLLib library [15] from which we remove the words "no", "not", "nor", "on", "off" and "any" (i.e. we will leave them in the logs) which are meaningful in the failure messages we are looking for. A tokenizer and stopwords removal on the previous sentence "Temperature too high: the CPU is overheating" would result in "temperature", "high", "cpu", "overheating".

C. Vectorization

The next step in adapting the logs to prediction algorithms is to determine how to transform the sentences from a list of words into a vectorized representation. The first method that we try is the bag-of-words model. In this model, a sentence is represented as a vector of dimension equal to the total vocabulary size of the corpus. Each dimension corresponds to a given word and the value of the vector is equal to the number of times the given word appears in the sentence. This representation is easy to generate and the resulting model is easy to manipulate from a machine learning perspective but it loses the information contained in the ordering of the words in the sentence. For example, the sentences "Error: no network connection" and "Network connection: no error" would have the same representation in this model despite having different meanings.

The next model we try is the bi-gram model. In this model, instead of considering individual words, we consider pairs of consecutive words. This reduces the ambiguity generated by the loss of ordering. In the previous example, the vocabulary of the sentences would be respectively "Error no", "no network", "network connection" and "network connection", "connection no", "no error". This would capture the fact that both sentences are about network connection but have different conclusions. The drawback of this approach is that it increases the size of the vocabulary: given n different words, we can generate n^2 different bi-grams so, theoretically, a sentence should be represented in the bi-gram space by a vector of dimension n^2. Not all possible bi-grams are encountered which reduces this effect. In the example sentences, starting from a vocabulary of 4 words, we only get 5 different bi-grams.

We also study the tri-gram model where tuples of three consecutive words are considered. Given the limitations induced by the volume of logs and the increase in dimensionality of the n-gram model, we do not study n-grams for $n > 3$.

D. Model compression

The final step before we run the classification is the hashing trick method. This method reduces the size of the models in memory by applying a hashing function to the words and replacing the words with their computed hash. It considerably reduces the memory footprint of the models enabling faster and more complex computations at the expense of possible hash collisions and a reduction in the interpretability of the resulting models. In our case, we use the default hashingTF implementation of Apache Spark MLLib library.

E. Classification

The final step is the classification itself. The goal is to automatically categorize the log messages encoded in the previous steps so that failure prediction can be made based on the message categories. Two unsupervised clustering algorithms have been considered for this step, bisecting k-means [16] and Latent Dirichlet Allocation [17].

Bisecting k-means is a variant of the k-means clustering algorithm. It starts with a unique cluster that regroups all samples and then iteratively splits the clusters starting with the one with the highest within set sum of squared errors. In our case, $k = 50$ was found to be the optimal value after a grid search step.

Latent Dirichlet Allocation (LDA) is a topic modelling approach which attempts to generate topics associated with frequently co-occurring words and assign to every sentence a set of weights corresponding to the topic distribution inside the sentence. In our case, we set the number of topics to 50, as with the bisecting k-means and the number of iterations to define the topics to 100.

The final outcome of all the previous steps are features whose format depends on the classification technique used in the last step. In the case of bisecting k-means, the text

associated to each message is replaced by a single value corresponding to its cluster label. An intuitive way to explain how it works is that the messages classified in the same cluster are very similar and are thus supposed to be about the same topic such as "Temperature test" or "Network failure", it is worth noting that topics are in this representation mutually exclusive. In the case of LDA, the log message is replaced by a vector of topic distribution. In that case, we extend the previous model with the possibility to be related to multiple topics. Hence if we have a "Network" topic and a "Failure" topic, we would find network failures by selecting messages with a topic distribution vector with a high value in both of these topics.

IV. Log mining

A. Modelling

In order to evaluate the efficiency of the log parsing, we target a well-known failure that appears in our log system. The industrial dataset used is a record of one year of logs from an aeronautical system currently in use. It contains over 4.5 million log messages including 302 messages pertaining to the failure that must be predicted. We manually label every instance of that failure. The failure messages appear in burst because of retry mechanisms and the propagation of the consequences of the failure to several subsequent tests. The repeated failures have no practical interest since they always happen in fixed sequences and do not reveal actual new events. Thus we filter them out and only keep the failure label of the first message in each sequence. After this filtering, we end up with 188 messages labelled as failures out of over 4.5 million log messages for an imbalance ratio of over $23.000 : 1$ in favour of the non-failure cases. We model the log messages as a time series. The features we associate with each message depend on the classification step that was used. If a bisecting k-means was performed, we associate to each message the sequence of cluster labels of the 20 previous messages. If a LDA was performed, we associate to each message the sequence of topic distribution vectors of the 20 previous messages. If less than 20 previous messages are available, we do not consider the current message for prediction.

B. Classification

The technique that we will use to transform the features we engineered into predictions is the Random Forest (RF) technique [18]. It is one of the state-of-the-art techniques for classification. It requires little parametrisation and is very efficient even in the case of extreme class imbalance and in the presence of noise. Roughly speaking, RF is an ensemble technique based on a decision tree classifier. A decision tree works by splitting the data set into smaller subsets based on measured attributes until either the subsets are each composed of only one class or the maximum depth of the tree has been reached. RF improves on this technique by combining several decision trees each trained on bootstrapped samples with different attributes. New predictions are then made based on a vote among the different decision trees. In this study, we set the number of decision trees at 20 with a maximum depth of 8, constructed using the Gini impurity metric. Two other methods were considered, Gradient Boosted Trees [18] and Support Vector Machine [19]. Their performances however did not match RF on any configuration. This matches the observation from [10].

TABLE IV
Performance comparison against state-of-the-art approaches on the HDFS benchmark dataset

	Precision	Recall	F-score
SLCT	0.593	0.649	0.620
LogSig	0.963	0.634	0.765
IPLoM	0.975	0.665	0.791
Drain	0.975	0.665	0.791
POP	0.975	0.665	0.791
Best log parsing combination	>0.999	0.985	0.992

V. Results and interpretation

The results on the operational dataset are reported in the tables I, II and III. The comparison with the existing method on the benchmark dataset is presented in Table IV. The parameters used are the one described in the previous sections, optimized through grid-search on the industrial dataset unless noted otherwise. All of the following results were validated through 3-fold cross-validation.

A. Metrics

We have chosen to report the standard performance metrics, precision, recall and F-score to evaluate our results. Precision and recall are also used in the benchmark performance evaluation in [6] and F-score is a hybrid metrics that will enable the comparison of techniques that trade-off between precision and recall. Defining the failure messages that we manually labelled as positives and the non-failure ones as negatives, we have:

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \quad (1)$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \quad (2)$$

$$F - score = \frac{2 \times precision \times recall}{precision + recall} \quad (3)$$

B. Industrial dataset

On Table I, the first observation that we can make is that no matter what log parsing techniques are used, the precision remains quite high with its lowest value achieved with the combination LDA, bi-gram, stemming and synonyms replacement at 75.8% and the next lowest at 85.7%. This is likely due to the high precision and resilience to noise of the RF learning algorithm. However, significant improvements can still be achieved as two combinations achieve 0 false positive for a reported precision over 99.5%, those combinations are LDA, tri-gram, stemming and synonym replacement and LDA, bag-of-words, stemming and synonym replacement and stopwords removal.

TABLE I
LOG MINING PRECISION WITH DIFFERENT LOG PARSING SCHEMES ON THE INDUSTRIAL DATASET

		Bisecting k-means			LDA		
		Bag-of-words	Bi-gram	Tri-gram	Bag-of-words	Bi-gram	Tri-gram
Nothing	Nothing	0.865	0.930	0.971	0.962	0.978	0.988
	Stopwords removal	0.955	0.941	0.954	0.989	0.979	0.953
Stemming and synonym replacement	Nothing	0.927	0.883	0.945	0.857	0.758	>0.995
	Stopwords removal	0.944	0.980	0.967	>0.995	0.974	0.906

TABLE II
LOG MINING RECALL WITH DIFFERENT LOG PARSING SCHEMES ON THE INDUSTRIAL DATASET

		Bisecting k-means			LDA		
		Bag-of-words	Bi-gram	Tri-gram	Bag-of-words	Bi-gram	Tri-gram
Nothing	Nothing	0.454	0.670	0.723	0.666	0.899	0.436
	Stopwords removal	0.782	0.643	0.674	0.471	0.512	0.671
Stemming and synonym replacement	Nothing	0.604	0.320	0.723	0.586	0.499	0.459
	Stopwords removal	0.552	0.589	0.642	0.552	0.832	0.486

TABLE III
LOG MINING F-SCORE WITH DIFFERENT LOG PARSING SCHEMES ON THE INDUSTRIAL DATASET

		Bisecting k-means			LDA		
		Bag-of-words	Bi-gram	Tri-gram	Bag-of-words	Bi-gram	Tri-gram
Nothing	Nothing	0.595	0.779	0.829	0.787	0.937	0.605
	Stopwords removal	0.860	0.764	0.790	0.638	0.672	0.788
Stemming and synonym replacement	Nothing	0.731	0.470	0.819	0.696	0.602	0.628
	Stopwords removal	0.697	0.736	0.772	0.710	0.897	0.633

Concerning recall in Table II, the differences between the different combinations are more significant and the motivation for using the different models is more obvious. The most simple combination, bag-of-words and bisecting k-means only achieves 45.4% recall with only one other combination significantly lower, bisecting k-means, bi-gram, stemming and synonym replacement at 32.0% and a few combinations at comparable recall. The highest recall is achieved with LDA and bi-gram at 89.9%.

Finally regarding the overall performances summarized with the F-score in Table III, the improvement brought by the log parsing techniques is even more obvious with every combinations besides the previously mentioned bisecting k-means, bi-gram, stemming and synonym replacement having a higher score than the basic bisecting k-means, bag-of-words. That is to say the top left combination bisecting k-means, bag-of-words can be considered a basic log parser with little to no NLP applied to it and, beside one specific combination, every NLP technique improves on it. The highest score of 0.937 is achieved by the combination which also had the highest recall LDA, bi-gram followed by LDA, bi-gram, stopwords removal and stemming and synonym replacement at 0.897.

Overall, it can be noted that the interactions between the different models and techniques are complex as none of them can be highlighted as systematically increasing or decreasing the metrics used. It is only when the models and methods are combined with each other that their usefulness becomes apparent. This is not surprising considering that these NLP methods are also in use in the field of speech processing and, similarly, the optimal combination of these techniques is still highly dependent on the dataset used with parameters as varied as the target language, the vocabulary size, the number of available audio samples and on target task. The key point is that the best combination we found with regard to F-score is quite robust, displaying the best recall over every other combination and one of the best precision. To ensure further that its performances are not coincidental, beyond the cross-validation, we will proceed to test it on the public benchmark with the same parameters we used for the industrial dataset.

C. Public dataset

The last step presented in Table IV is the comparison of the best combination, LDA, bi-gram, with existing results from the studies [6], [8] and [9] on a public benchmark dataset from an HDFS cluster. The HDFS dataset was chosen among the available benchmark sets in [6] as the largest dataset and also the closest one to the industrial dataset we used. The parameters used for the algorithms were kept at the same value as with the industrial dataset. The comparison shows clearly that the natural language processing inspired approach improves recall significantly while maintaining a

high precision leading to a very significant increase of the F-score from 0.791 to 0.992.

This could be seen as surprising because the traditional rule-based log parsing methods put focus on producing a structure that perfectly transposes the information in the log messages with performances such that significant improvements seemed unlikely. Our interpretation is that building a perfectly unambiguous parsing is actually not a desirable outcome when dealing with log messages because of the logs inherent ambiguity. The messages are implemented by a human operator to be read and understood by another human operator with natural language that is inherently ambiguous and flexible. This flexibility enables a human reader to understand when several messages, though they might be worded differently, refer to a similar issue. For example, when dealing with network problems, it is common that the actual log message vary depending on the subsystem that attempted to connect. It is obvious for someone reading the log messages that they are related but an unambiguous rule-based approach will not detect it because it operates under the assumption that one spelling is equivalent to one lemma. Through the use of natural language processing and clustering techniques in our approach, we introduce ambiguity and flexibility by foregoing the word order and voluntarily confusing semantically similar messages imitating in that sense the approach of a human operator and leading to better results.

VI. Conclusion

In this paper we propose a general scheme for the failure prediction problem in the context of industrial systems. Our approach is based on log mining, which is a promising approach to achieve failure prediction as many systems already implement detailed logs and on log parsing which is a critical preliminary step. We focus on simple natural language processing techniques or combination thereof and on the interaction between log parsing and log mining and their effect on the performances of failure prediction. To that end, we provide a detailed performance analysis on both an industrial dataset of a system currently in use and on a large publicly available benchmark dataset in order to compare the performances of our approach with state-of-the-art algorithms. On the industrial dataset, we achieve 97.8% precision and 89.9% recall using LDA and bi-gram. On the HDFS dataset, the same method improves the precision and recall from respectively 97.5% and 66.5% to over 99.9% and 98.5% and the F-score improves over the study [13] from 96% to 99.2% with a more simple and robust pipeline. We finally offer an interpretation of the improvement yielded by natural language processing based methods over traditional rule-based methods. We considered the possibility to use more complex NLP techniques such as word embedding but, given the excellent performances of the simpler methods, any possible performance increase over them would most likely not be measurable and deemed that obtaining the same performances with simpler methods is a stronger result. Several more advanced NLP techniques such as word embedding, topic segmentation or a more elaborate synonym replacement process would be worth investigating in the future however the most pressing issue would be to gather a more complex benchmark dataset in order to evaluate said techniques.

References

[1] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 117–132.

[2] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ser. ICDM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 149–158. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2009.60

[3] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 26–26.

[4] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*. IEEE, 2003, pp. 119–126.

[5] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

[6] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 2016, pp. 654–661.

[7] R. Sipos, D. Fradkin, F. Mörchen, and Z. Wang, "Log-based predictive maintenance," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, August*, 2014, pp. 1867–1876.

[8] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[9] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, June 2017, pp. 33–40.

[10] C. Bertero, M. Roy, C. Sauvanaud, and G. Trédan, "Experience report: Log mining using natural language processing and application to anomaly detection," in *28th International Symposium on Software Reliability Engineering (ISSRE 2017)*, 2017, p. 10p.

[11] M. Kubacki and J. Sosnowski, "Holistic processing and exploring event logs," in *Software Engineering for Resilient Systems*, A. Romanovsky and E. A. Troubitsyna, Eds. Cham: Springer International Publishing, 2017, pp. 184–200.

[12] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proceedings - 16th IEEE International Conference on Data Mining, ICDM 2016*. United States: Institute of Electrical and Electronics Engineers Inc., 1 2017, pp. 859–864.

[13] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 1285–1298. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134015

[14] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*. Association for Computational Linguistics, 2002, pp. 63–70.

[15] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[16] M. Steinbach, G. Karypis, V. Kumar *et al.*, "A comparison of document clustering techniques," in *KDD workshop on text mining*, vol. 400, no. 1. Boston, 2000, pp. 525–526.

[17] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[18] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends in Computer Graphics and Vision*, vol. 7, no. 23, pp. 81–227, 2012. [Online]. Available: http://dx.doi.org/10.1561/0600000035

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.