Unsupervised signature extraction from forensic logs

Stefan Thaler[1], Vlado Menkovski[1] and Milan Petkovic[1,2]

[1] Technical University of Eindhoven, s.m.thaler@tue.nl, v.menkovski@tue.nl,
Den Dolech 12, 5600 MB Eindhoven, Netherlands
[2] Philips Research Laboratories, firstname.lastname@philips.com
High Tech Campus 34, Eindhoven, Netherlands

Abstract. Signature extraction is a key part of forensic log analysis. It involves recognizing patterns in log lines such that log lines that originated from the same line of code are grouped together. A log signature consists of immutable parts and mutable parts. The immutable parts define the signature, and the mutable parts are typically variable parameter values. In practice, the number of log lines and signatures can be quite large, and the task of detecting and aligning immutable parts of the logs to extract the signatures becomes a significant challenge. We propose a novel method based on a neural language model that outperforms the current state-of-the-art on signature extraction. We use an RNN auto-encoder to create an embedding of the log lines. Log lines embedded in such a way can be clustered to extract the signatures in an unsupervised manner.

Keywords: Information forensic, RNN auto-encoder, Neural language model, Log clustering, Signature extraction

1 Introduction

An important step of an information forensic investigation is log analysis. Logs contain valuable information for reconstructing incidents that have happened on a computer system. In this context, a log line is a sequence of tokens that give information about the state of a process that created this log line. The tokens of each log lines are partially natural language and partially structured data. Tokens may be words, numbers, variables or punctuation characters such as brackets, colons or dots.

Log signatures are the print statements that produce the log lines. Log signatures have fixed parts and may have variable parts. Fixed parts consist of a sequence of tokens of arbitrary length that uniquely identify signatures. The variable parts may also be of arbitrary length and variable parts in log lines that originate from the same signature differ.

The goal of a forensic investigator is to uncover a sequence of events from a forensic log that reveal a security incident. The sequence of events describes the actions that the users of this computer system took. Traces of these actions

are typically stored in logs. Similar events may have different log lines associated with them because the variable part reports the state of the system at that time, which makes finding such events difficult. Knowing the log signatures of a log enable a forensic investigator to group together log lines that belong to the same event, even though the log lines differ. Finding such signatures is challenging because of the unknown number of signatures and the unknown number and position of fixed parts. Signature extraction is the process of finding a set of log signatures given a set of log lines.

State-of-the-art approaches identify log signatures based on the position and frequency of tokens [1, 12, 23]. These approaches typically assume that frequent words define the fixed parts of the signature. This assumption holds if the ratio of log lines per signature in the analyzed log is high. This can be the case for many application logs where the tokens of fixed signature parts are repeated with a high frequency. However, in information forensics, logs commonly have many signatures but few log lines per signature. In this case, the number of occurrence of tokens of variable parts may be higher than fixed tokens, which can cause a confusion of which tokens are fixed and which ones are variable. Confusing fixed tokens with variable ones leads to signatures that match too few log lines, and mixing variable tokens with fixed ones will result into signatures that will match too few log lines.

To address the challenge of signature extraction from forensic logs, we propose to use a method that takes contextual information about the tokens into account. Our approach is inspired by recent advances in the NLP domain, where sequence-to-sequence models have been successfully used to capture natural language [3, 9].

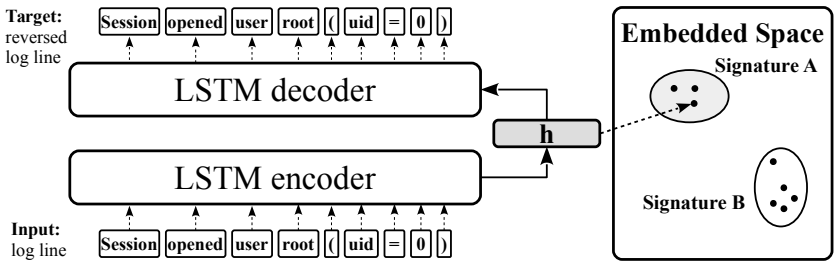

Fig. 1. We first embed forensic log lines using an RNN auto-encoder. We then cluster the embedded log lines and assign them to a signature.

Typically, sequence-to-sequence models consist of two recurrent neural networks (RNNs), an encoder network and a decoder network. The encoder network learns to represent an input sequence, and the decoder learns to construct the output sequence from this representation. We use such a model to learn an encoding function that encodes the log line, and a decoding function that learns to

reconstruct the reversed input log line from this representation. Figure 1 depicts this idea. Based on the findings in the NLP domain, we assume that this embedding function takes into account contextual information, and embeds similar log lines close to each other in the embedded space. We then cluster the embedded log lines and use the clusters as signature assignment.

In detail, the main contributions of our paper are:

- We propose a method, LSTM-AE+C, that uses an RNN auto-encoder to create a log line embedding space. We then cluster the log lines in the embedding space to determine the signature assignment. We detail this method in Section 2.
- We demonstrate on our own and two public datasets that LSTM-AE+C outperforms two state-of-the-art approaches for signature extraction. We detail the experiment setup in Section 3 and discuss the results after that.

2 Method

Our method LSTM-AE+C for signature extraction of forensic logs can be divided into two steps. First, we train a sequence-to-sequence auto-encoder network to learn an embedded space for log lines. Sequence-to-sequence neural networks for natural language translation have been introduced by Sutskever et al. [19] and widely applied since then. We use a similar model, however, instead of using it in a sequence-to-sequence manner, we use it as auto-encoder that reconstructs the input sequence. Secondly, we cluster the embedded log lines to extract the signatures.

We depict a schematic overview of our model in Figure 2. To learn an embedding we train the LSTM auto-encoder to reconstruct each input log line. To do that, the encoder part of the auto-encoder needs to encode the log line into a fixed size vector that is fed into the decoder. The fixed size of the vector limits the capacity of the auto-encoder and provides for a regularization that restricts the auto-encoder from learning an identity function. We use that representation as embedding for the log lines. In the remaining section we will first detail the components of our model and their relationships to each other, then detail the learning objective and finally describe how we extract signatures.

2.1 Model

The input to our model is log lines. We treat log lines as a sequence of tokens of length n, where a token can be a word, variable part or delimiter. The set of unique tokens is our input vocabulary, where each token in the vocabulary gets a unique id.

Since the number of such tokens in a log can be potentially very large, we learn a dense representation for the tokens of our log lines. To get these dense representations, we use a token embedding matrix $E^{(v \times u)}$, where v is the unique number of tokens that we have in our token vocabulary and u is the number

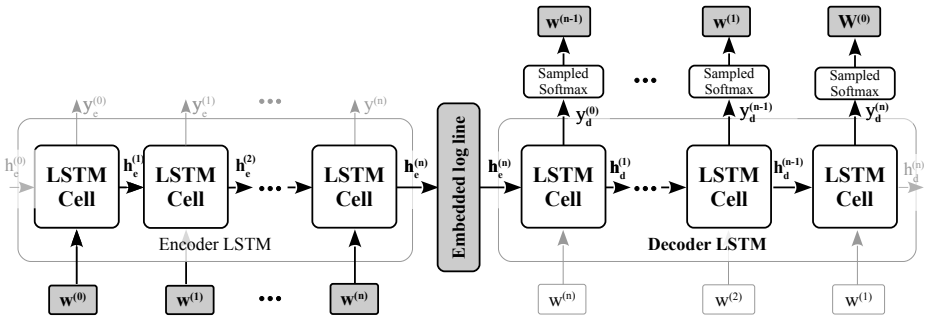

Fig. 2. We use a sequence-to-sequence LSTM auto-encoder to learn embeddings for our log lines.

of hidden units of the encoder network. The index of each row of E is also the position of v in the vocabulary. We denote an in E embedded token as w.

Next, want to learn the log line embedding. To do so, we learn an encoder function ENC using an LSTM [7], which is a variant of a recurrent neural network. We chose an LSTM for both the encoder and decoder, because it addresses the vanishing gradient problem. $h_e\ (t)$ is the hidden encoder state at time step t, and $y_e^{(t)}$ is the encoder output at time step t. $w_e^{(t)}$ is the embedded input word for time step t. We use the encoding state and and input word at each time step to calculate the next state and the next output. We discard all output s of the encoder. We use the final hidden state $h_e^{(n)}$ to embed our log lines. $h_e^{(n)}$ also serves as initial hidden state for our decoder network.

$$(y_e^{(t)}, h_e^{(t+1)} = ENC(w_e^{(t)}, h_e^{(t)})$$

Our decoder function DEC is trained to learn to reconstruct the reverse sequence S' given the last hidden state $h_e^{(n)}$ of our encoding network. The structure of the network is identical to the encoder, except we feed the network the reverse sequence of embedded tokens as input.

$$(y_d^{(t)}, h_d^{(t+1)}) = DEC(w_d^{(t)}, h_d^{(t)})$$

From the decoder outputs $y_d^{(t)}$ we predict the reverse sequence of tokens S'. Calculating a softmax function for a large token vocabulary is computationally very expensive because the softmax function is calculated as the sum over all potential classes. Therefore, we predict the output tokens of our decoder sequence using sampled softmax [8]. Sampled softmax is a candidate training method that approximates the desired softmax function by solving a task that does not require predicting the correct token from all token. Instead, the task sampled softmax solves is to identify the correct token from a randomly drawn sample of tokens.

2.2 Objective

To embed our log lines in an embedded space, the model needs to maximize the probability of predicting a reversed sequence of tokens S' given a sequence of tokens S. In other words, we want to train the encoding network to learn an embedding that contains enough information for the decoder to reconstruct it. However, as an effect of the regularization, we expect the model to use the structure to use the structure of the log lines to create a more efficient representation, that in turn allows us to extract the signatures.

$$\theta^* = arg\,\underset{\theta}{max} \sum_{(S,S')} log\, p(S'|S;\theta)$$

θ are the parameters of our model, S represents a log line, and S' represents a reversed log line.

S is a sequence of tokens of arbitrary length. To model the joint probability over $S'_0, \ldots, S'_{t-1}$ given S and θ, it is common to use the chain rule for probabilities.

$$log\, p(S'|S,\theta) = \sum_{t=0}^{n} log\ p(S'_t|S,\theta,S'_0,\ldots,S'_{t-1})$$

When training the network, S and S' are the inputs and the targets of one training example. We calculate the sum of equation 2.2 per batch using RMSProp [22]. We detail the hyper parameters of the training process in section 3.3.

2.3 Extracting signatures

After the training of our auto-encoder model is complete, we use encoding network to generate the embedded vector. We expect that due to the regularization structurally similar log lines will be embedded close to each other, which enables us to use a clustering algorithm to group log lines which belong to the same signature.

Since forensic logs may be very large, we cluster the embedded log lines using the BIRCH algorithm [27]. BIRCH is an iterative algorithm that dynamically builds a height-balanced cluster feature tree. The algorithm has an almost linear runtime complexity on the number of training examples, and it does not require the whole dataset to be stored in memory. These two properties make the algorithm well suited for applications on large datasets.

3 Experiments

We compare our method to LogCluster [23] and IPLoM [11]. Many algorithms have been designed to be applied to a special type of application log, where the number of signatures is known up front. However, in an information forensic

context the forensic logs that are being analyzed stem from an unknown system, which means that the number of signatures is not known up front. Therefore it is important that IPLoM and LogCluster do not require a fixed amount of clusters as a hyper parameter. Furthermore, in a study by He et al [6], IPLoM and SLCT were amongst the best-performing signature extraction algorithms. LogCluster is the improved version of SLTC that addresses multiple shortcomings of SLCT. We thus assume the LogCluster would have outperformed SLCT in He's evaluation. For IPLoM, we use the implementation provided by [6]. For LogCluster we use the implementation provided by the author online[3]. We implemented our own method LSTM-AE+C in Tensorflow version 1.0.1. Our experiments are available on GitHub[4].

3.1 Evaluation metrics

To assess our approach, we treat the log signature extraction problem as log clustering problem because log clustering and log signature extraction are related problems [20]. The key difference between clustering and signature extraction is, that the goal of a clustering approach is the find the best clusters according to some metric, whereas the goal of signature extraction is to find the right set of signatures. This set of signatures does not have to be the best set of clusters.

We evaluate the quality of the retrieved clusters of all our evaluated approaches with two metrics: the V-Measure [15] and the adjusted mutual information [24]. The V-Measure is the harmonic mean between the homogeneity and the completeness of clusters. It is based on the conditional entropy of the clusters. The adjusted mutual information describes the mutual information of different cluster labels, adjusted for chance. It is normalized to the size of the clusters. Both approaches are independent of permutations on the true and predicted labels. The values of the V-Measure and the adjusted mutual information can range from 0.0 to 1.0. In both cases, 1.0 means perfect clusters and 0.0 means random label assignment.

Additionally, we assess the cluster quality for clusters retrieved with LSTM-AE+C using the Silhouette score. The Silhouette score measures the tightness and separation of clusters and only depends on the partition of the data [16]. It ranges between -1.0 and 1.0, where a negative score means many wrong cluster assignments and 1.0 means perfect clustering.

We validate the stability of our approaches using 10-fold, randomly subsampled 10000 log lines [10]. In Section 3.4, we report the average scores for our metrics and their standard deviation.

3.2 Datasets

We use three logs to evaluate and compare our method: a forensic log that we extracted from a virtual machine hard drive and the system logs of two high-

[3] https://ristov.github.io/logcluster/
[4] https://github.com/stefanthaler/2017-ecml-forensic-unsupervised

performance cluster computers, BlueGene/L(BGL) and Spirit [13]. An overview over the log statistics is presented in Table 1.

We created our forensic log by extracting it from a Ubuntu 16.04 system image disk using the open source log2timeline tool[5]. We manually created the signatures for this dataset by looking at the Ubuntu source code. The difference between a forensic log and a system log is that a forensic log contains information from multiple log files on the examined system, whereas a system log only contains the logs that were reported by the system daemon. The system log is part of the forensic log, but it also contains other logs, which typically leads to more complexity in such log files.

BlueGene/L(BGL) was a high-performance cluster that was installed in the Lawrence Livermore National Labs. The publicly available system log was recorded during March 6th and April 1st in 2006. It consists of 4.747.963 log lines in total. In our experiments, we use a stratified sample which has 474.796 log lines. We manually extracted the signatures for this log file.

Spirit was a high-performance cluster that was installed in the Sandia National Labs. The publicly available system log was recorded during January 1st and the 11th of July in 2006. It consists of 272.298.969 log lines in total. In our experiments, we use a stratified sample which has 716.577 lines. We also extracted the signatures for this log by hand.

The BlueGene/L and the Spirit logs are publicly available and can be downloaded from the Usenix webpage[6]. We publish our dataset on GitHub[7].

Table 1. The log file statistics are as follows:

Log Name	Lines	Signatures	Unique tokens
Forensic	11.023	852	4.114
BlueGene/L	474.796	355	114.495
Spirit2	716.577	691	59.972

For all three log files, we removed fixed position data such as timestamps or dates at the beginning of each log message. In the case of our forensic log we completely removed these columns. In the case of the other two logs, we replaced the fixed elements with special token, such as TIME_STAMP. We added this preprocessing because it reduces the sequence complexity, but it does not reduce the quality of the extracted signatures.

[5] https://github.com/log2timeline/
[6] https://www.usenix.org/cfdr-data
[7] https://github.com/stefanthaler/2017-ecml-forensic-unsupervised

3.3 Hyper parameters and training details

IPLoM supports the following parameters: File support threshold (FST), which controls the number of clusters found; partition support threshold (PST), which limits the backtracking of the algorithm; upper bound (UB) and lower bound(LB) which control when to split a cluster and cluster goodness threshold (CGT) [11]. We evaluate IPLoM by performing a grid search on the following parameter ranges: FST between 1 and 20 in 1 steps, PST of 0.05, UB between 0.5 and 0.9 in 0.1 steps, LB, between 0.1-0.5 in 0.1 steps and CGT between 0.3 and 0.6 in 0.1 steps. We chose the parameters according to the guidelines of the original paper.

LogCluster supports two main parameters: support threshold (ST), which controls the minimum amount of patterns and the word frequency (WF), which sets the frequency of words within a log line. We evaluate LogCluster by performing grid search using the following parameter ranges: ST between 1 and 3000 in and WF of 0.3, 0.6 and 0.9.

We generate each input token sequence by splitting a log line at each special character. Furthermore, we add a special token at the beginning and the end of the sequence that marks the beginning and the end of a sequence. Within a batch, sequences are zero-padded to the longest sequence in this batch, and zero inputs are ignored during training.

All embeddings and LSTM cells had 256 units. Both encoder and decoder network had a 1-layer LSTM. We trained all our LSTM auto-encoders for ten epochs using RMSProp [22]. We used a learning rate of 0.02 and decayed the learning rate by 0.95 after every epoch. Each training batch had 200 examples and the maximum length number of steps to unroll the LSTM auto-encoder was 200. We used 500 samples to calculate the sampled softmax loss. We used dropout on the decoder network outputs [17] to prevent overfitting and to regularize our network to learn independent representations. Finally, we clip the gradients of our LSTM encoder and LSTM decoder at 0.5 to avoid exploding gradients [14].

The hyper parameters and the architecture of our model were empirically determined. We tried LSTMs with attention mechanism [3], batch normalization, multiple layers of LSTMs, and more units. However, these measures had little effect on the quality of the clusters; therefore we chose the simplest possible architecture. We used the same architecture and hyper parameters for all our experiments.

The second step in our method is to cluster the embedded log lines to find signatures. We cluster the embedded log lines using the BIRCH cluster algorithm [27]. We performed the clustering using grid search on distance thresholds between 1 and 50 in 0.5 steps, and a branching factor of either 15, 30 or 50.

3.4 Results

We report the results of our experiments in Table 2. Each value reports the best performing hyper parameter settings. Each score is the average of 10-fold random sub-sampling followed by the standard deviation of this average. We

do not report on the Silhouette score for LogCluster and IPLoM because both algorithms do not provide a means to calculate the distance between different log lines.

Table 2. Log clustering evaluation, best averages and standard deviation.

Log file	Approach	V-Measure	Adj. Mut. Inf.	Silhouette
Forensic	LogCluster [23]	0.904 ±0.000	0.581 ±0.000	N/A
	IPLoM [11]	0.825 ±0.001	0.609 ±0.001	N/A
	LSTM-AE+C (Ours)	**0.935** ±0.002	**0.864** ±0.004	0.705 ±0.001
BlueGene/L	LogCluster [23]	0.592 ±0.004	0.225 ±0.005	N/A
	IPLoM [11]	0.828 ±0.003	0.760 ±0.005	N/A
	LSTM-AE+C (ours)	**0.948** ±0.005	**0.900** ±0.001	0.827 ±0.002
Spirit	LogCluster [23]	0.829 ±0.002	0.677 ±0.004	N/A
	IPLoM [11]	0.920 ±0.004	0.895 ±0.003	N/A
	LSTM-AE+C (ours)	**0.930** ±0.010	**0.902** ±0.008	0.815 ±0.004

3.5 Discussion of results

As can be seen from Table 2, our approach significantly outperforms the two word-frequency based baseline approaches on the three datasets, both regarding V-Measure and Adjusted Mutual Information. The standard deviation is below 0.005 in all reported experiments, which indicates that clustering is consistently stable over the datasets.

For all three log files we obtain a Silhouette score of greater than 0.70, which indicates that the cluster algorithm has found a strong structure in the embedded log lines. The weakest structure has been found in the Forensic log. We hypothesize that the high signature-to-log-line ratio in this log causes the lower Silhouette score.

Finding the optimal number of clusters for a clustering or signature extraction approach is a well-known problem. We do not address the topic of finding the optimal number of signatures in this paper, but it is a fundamental research topic in many methods for finding the optimal number of clusters have been proposed, for example [18, 21].

4 Related Work

Log signature extraction has been studied to achieve a variety of goals such as anomaly and fault detection in logs [5], pattern detection [1, 12, 23], profile building [23], or compression of logs [12, 20].

Most of the approaches use word-position or word-frequency based heuristics to extract signatures from logs. Tang et al. propose to use frequent word-bigrams

to obtain signatures [20]. Fu et al. propose to use a weighted word-edit distance function to extract signatures [5]. Makanju et al. use the log line length as well as word frequencies to extract signatures [11]. Vaarandi et al. use word frequencies and word correlation scores to determine the fixed parts of log lines and thereby the signatures [23]. Xu et al. propose a method that is not base on statistical features of the log lines. Instead, they propose to create to extract the signatures from the source code [26].

Recently, RNN sequence-to-sequence models have been successfully applied for neural language modeling and statistical machine translation tasks [2, 3, 19]. Apart from that, Johnson et al. demonstrated on a large scale that sequence-to-sequence models can be used to allow translation between languages even if explicit training data from source to target language is not available [9].

Auto-encoders have been successfully applied to clustering tasks, such as clustering text and images [25]. Variational recurrent auto-encoders have been used to cluster music snippets [4].

5 Conclusion and future work

We have presented the LSTM-AE+C a method for clustering forensic logs according to their log signatures. Knowing that log lines belong to the same signature enables a forensic investigator to run more sophisticated analysis on a forensic log, for example, to reconstruct security incidents. Our method uses two components: an LSTM encoder and a hierarchical clustering algorithm. The LSTM encoder is trained as part of an auto-encoder on a log in an unsupervised fashion, and then the clustering algorithm assigns embedded log lines to their signature.

Experiments on three different datasets show that this method outperforms two state-of-the-art algorithms on clustering log lines based on their signatures both in V-Measure and adjusted mutual information. Moreover, we find that the Silhouette score of all found clusters by our method are greater than 0.70, which indicates strongly structured clusters.

One potential way of improving this method is to add a regularization term that aids the auto-encoder in embedding the clustering. Adding a regularization term could be a possible way to inject domain knowledge in the learning process and therefore increase the quality of the learned representation. For example, one could penalize the reconstruction loss of likely variable parts such as memory addresses, numbers or dates less.

Furthermore, we intend to investigate whether the attention mechanism of attentive LSTMs could be used to identify mutable and fixed parts of log lines. Finally, another future direction to our approach is to extract signatures that are human-interpretable. One potential way of addressing this is by using the decoder network to sample log lines from the embedding space.

Acknowledgment

This work has been partially funded by the Dutch national program COMMIT under the Big Data Veracity project.

References

1. Aharon, M., Barash, G., Cohen, I., Mordechai, E.: One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 5781 LNAI, pp. 227–243. Springer (2009)
2. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) pp. 1724–1734 (2014), http://arxiv.org/abs/1406.1078
3. Dzmitry Bahdana, Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation By Jointly Learning To Align and Translate. Iclr 2015 pp. 1–15 (2014), http://arxiv.org/abs/1409.0473v3
4. Fabius, O., van Amersfoort, J.R.: Variational recurrent auto-encoders. arXiv preprint arXiv:1412.6581 (2014)
5. Fu, Q., Lou, J.g., Wang, Y., Li, J.: Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In: ICDM. vol. 9, pp. 149–158 (2009)
6. He, P., Zhu, J., He, S., Li, J., Lyu, M.R.: An evaluation study on log parsing and its use in log mining. Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016 pp. 654–661 (2016)
7. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation 9(8), 1735–1780 (1997)
8. Jean, S., Cho, K., Memisevic, R., Bengio, Y.: On Using Very Large Target Vocabulary for Neural Machine Translation (12 2014), http://arxiv.org/abs/1412.2007
9. Johnson, M., Schuster, M., Le, Q.V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., others: Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. arXiv preprint arXiv:1611.04558 (2016)
10. Lange, T., Roth, V., Braun, M.L., Buhmann, J.M.: Stability-based validation of clustering solutions. Neural computation 16(6), 1299–1323 (2004)
11. Makanju, A., Zincir-Heywood, A.N., Milios, E.E.: A Lightweight Algorithm for Message Type Extraction in System Application Logs. IEEE Transactions on Knowledge and Data Engineering 24(11), 1921–1936 (2012)
12. Makanju, A.A.O., Zincir-Heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09. p. 1255. ACM (2009)
13. Oliner, A.J., Stearley, J.: What Supercomputers Say : A Study of Five System Logs Today s Menu Motivation Data Seven Insights Recommendations. In: Dsn. pp. 575–584. IEEE (2007)
14. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. ICML (3) 28, 1310–1318 (2013)

15. Rosenberg, A., Hirschberg, J.: V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In: EMNLP-CoNLL. vol. 7, pp. 410–420 (2007)
16. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20, 53–65 (1987)
17. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15(1), 1929–1958 (2014)
18. Sugar, C.A., James, G.M.: Finding the number of clusters in a dataset: An information-theoretic approach. Journal of the American Statistical Association 98(463), 750–763 (2003)
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Nips. pp. 1–9 (2014)
20. Tang, L., Li, T., Perng, C.s.: LogSig : Generating System Events from Raw Textual Logs. In: Cikm. pp. 785–794. ACM (2011)
21. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63(2), 411–423 (2001)
22. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning 4(2) (2012)
23. Vaarandi, R., Pihelgas, M.: LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In: 12th International Conference on Network and Service Management - CNSM 2015. pp. 1–8. IEEE Computer Society (2015)
24. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research 11(Oct), 2837–2854 (2010)
25. Xie, J., Girshick, R., Farhadi, A.: Unsupervised Deep Embedding for Clustering Analysis. arXiv preprint arXiv:1511.06335 (2015)
26. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting Large-Scale System Problems by Mining Console Logs. In: 22nd ACM Symposium on Operating Systems Principles. pp. 117–131. ACM (2009)
27. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: ACM Sigmod Record. vol. 25, pp. 103–114. ACM (1996)