Discrete & Continuous Models
& Applied Computational Science
ISSN 2658-7149 (Online), 2658-4670 (Print)

2025, 33 (2) 172–183
http://journals.rudn.ru/miph

Research article

UDC 004.62:004.822
PACS 07.05.Mh, 07.05.Kf
DOI: 10.22363/2658-4670-2025-33-2-172-183 EDN: MGCVKV

Predictive diagnostics of computer systems logs using natural language processing techniques

Vladislav A. Kiriachek, Soltan I. Salpagarov

RUDN University, 6, Miklukho-Maklaya St, Moscow, 117198, Russian Federation

(received: December 7, 2024; revised: January 10, 2025; accepted: January 20, 2025)

Abstract. This study aims to develop and validate a method for predictive diagnostics and anomaly detection in computer system logs, using the Vertica database as a case study. The proposed approach is based on semi-supervised learning combined with natural language processing techniques. A specialized parser utilizing a semantic graph was developed for data preprocessing. Vectorization was performed using the fastText NLP library and TF-IDF weighting. Empirical validation was conducted on real Vertica log files from a large IT company, containing periods of normal operation and anomalies leading to failures. A comparative assessment of various anomaly detection algorithms was performed, including k-nearest neighbors, autoencoders, One Class SVM, Isolation Forest, Local Outlier Factor, and Elliptic Envelope. Results are visualized through anomaly graphs depicting time intervals exceeding the threshold level. The findings demonstrate high efficacy of the proposed approach in identifying anomalies preceding system failures and delineate promising directions for further research.

Key words and phrases: machine learning, natural language processing, log analysis, anomaly detection, predictive diagnostics

For citation: Kiriachek, V. A., Salpagarov, S. I. Predictive diagnostics of computer systems logs using natural language processing techniques. *Discrete and Continuous Models and Applied Computational Science* **33** (2), 172–183. doi: 10.22363/2658-4670-2025-33-2-172-183. edn: MGCVKV (2025).

1. Introduction

Effective monitoring and analysis of events occurring in computer systems are key factors in ensuring their reliable, secure and uninterrupted operation. The main source of information on the functioning of such systems are log files - text files containing structured and unstructured data on a wide range of events, including normal activity, warnings, errors and anomalies. Due to the rapid growth of the volume of generated data, measured in millions and billions of lines daily, as well as the diversity of sources and log formats, their systematic manual analysis becomes a virtually impossible task even for teams of qualified specialists. Failure of critical computer systems, such as databases, can lead to the collapse of other systems that rely on them. For example, the operation of web analytics products (dashboards) depends on the operation of databases, and their shutdown due to a database failure costs companies millions in losses. This raises the problem of searching for anomalies in order to prevent failures in computer systems using their logs.

In this regard, the development of intelligent systems for automatic log analysis based on Natural Language Processing (NLP) and machine learning is of paramount importance. Such systems are capable of extracting structured information from unstructured text, identifying patterns and anomalies, and generating reports and alerts automatically. The use of NLP-techniques, in particular, text classification algorithms, clustering, entity and relationship extraction, as well as modern deep learning models based on transformers, opens up new opportunities for intelligent big data analysis in the IT sector.

Having a certain structure, log files contain information about various system events, such as errors, warnings, and other incidents. They record the time and date of the event, as well as its type or importance level, designated by special tags (e.g. `<INFO>`, `<ERROR>`, `<FATAL>`). In addition, logs contain a significant amount of volatile data, including hash-sums, process identifiers, network addresses, etc. This data can be generated dynamically and, as a rule, is not repeated in future records, which requires the use of special preprocessing methods for their normalization and anonymization before further analysis. The literature discusses many parsers built on various architectures, such as Drain [1], Spell [2], and others. However, to meet the requirements of production tasks, which impose additional restrictions in the form of the need for integration into existing software and the ability to flexibly configure individual system components, it was decided to develop our own specialized log parser.

Anomalies in computer systems are events characterized by outlier values of their features and sharply contrasting with typical modes of operation of such systems during periods of their normal operation. Anomalous behavior of systems is often rare and unpredictable, deviating from established patterns based on previous observations. Therefore, the developed anomaly detection approach is based on semi-supervised learning techniques together with NLP algorithms such as fastText [3] and TF-IDF, which does not require labeling of training data. It is only necessary to know the periods of normal, uninterrupted operation of the system.

An analysis of existing solutions in the subject area under consideration reveals a wide variety of approaches to solving the problem of anomaly detection in computer systems, among which a significant share is made up of techniques based on the supervised learning paradigm LogRobust [4], CNN [5] as well as semi-supervised learning techniques, such as DeepLog [6], LogAnomaly [7], LogBERT [8], PLELog [9] or unsupervised approaches, such as Logsy [10].

The most widely used ML architectures include Recurrent Neural Networks (RNN) [4, 6, 7, 11], Convolutional Neural Networks (CNN) [5, 12], Transformers (TF) [8, 10], Graph Neural Networks (GNN) [13], as well as approaches that can do without labeling, such as Autoencoders (AE) [11, 14], Variational Autoencoders (VAE) [12] and classical machine learning techniques such as One Class SVM [15], Isolation Forest [16], Local Outlier Factor [17], Elliptic Envelope [18], k-nearest neighbors, tested in this work.

The developed approach is based on several successive stages: log preprocessing, vectorization and anomaly detection (predictive diagnostics).

2. System description

2.1. Description of the source data format

Before we talk about log preprocessing, let's define the concept of a log. A log is a text file with a certain structure containing information about system events, such as errors, the time and date of these events, and the event tag itself (e.g. `<INFO>`, `<ERROR>`, `<FATAL>`). The logs of various computer systems contain a lot of variable information, such as hash-sums, process IDs, timestamps, etc.

```
2023-03-30  03:41:29.540  Init  Session:0x7f87b2b42700  <ERROR>  @v_dwh_node0006:
42601/4205: Number of columns in the PROJECTION statement must be the same as the
number of columns in the SELECT statement
LOCATION:  transformProjectionStmt, analyze.c:2753

2023-03-30 03:56:15.390 TM Mergeout(02):0x7f6a757f2700 [Txn] <INFO> Commit Complete:
Txn: d000003bcea597 at epoch 0x1d18c9f9 and global catalog version 518874383

2023-03-30 07:57:54.830 Init Session:0x7f865e1fc700 <LOG> @v_dwh_node0006: 00000/2705:
Connection received: host=10.2.66.13 port=49524 (connCnt 78)
```

Figure 1. Example of raw Vertica database log data

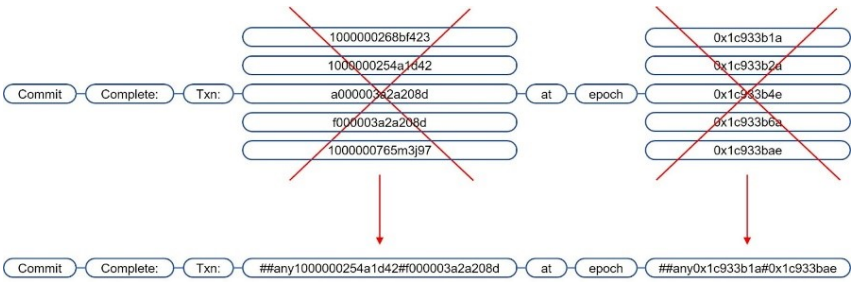

Figure 2. Parser diagram

In the Vertica database, considered in the work as a typical example of a computer system, the logs have a such structure. Examples of several events are shown in Figure 1. First comes information about the date and time of the event, then system information with the process id, the event tag is specified in curly brackets <...>, after which comes the text of the event itself, containing variable information, i.e. id, hosts, ports, etc. It is worth noting that most computer systems, not only databases, have a similar structure.

Log file sizes can be large enough for manual analysis. For example, in the Vertica database considered in the study, log files collected over one day have an average weight of about 400 MB and contain an average of over 10 million events. Thus, the size of the entire training set for 2 months of non-stop operation was 23 GB and over 830 million events.

2.2. Description of the proposed approach

Among the features typical for logs of any database, and not just the Vertica database considered in the work, it is worth noting the presence of SQL-queries in the logs themselves. This data can be useful, since a suboptimally written query can lead to problems in the operation of the database and even its failure. The use of existing log parsers is impossible, since they delete a lot of useful information, including SQL-queries, which was the reason for developing our own log parser, the architecture of which is based on the use of a semantic graph. The general scheme of the proposed approach is illustrated in Figure 2. The key idea is to build a semantic graph in the process of learning on a training data set, where individual lexical units correspond to graph vertices. When the number of graph branches becomes large enough, the graph collapses as shown in Figure 2, and frequent words are replaced with special words containing the constant part ##any and an added part consisting of a range of replaceable words. This is done, firstly, so that different special words appear in different places in the graph, and secondly, so that by looking at this word one can understand its approximate meaning and characteristic values.

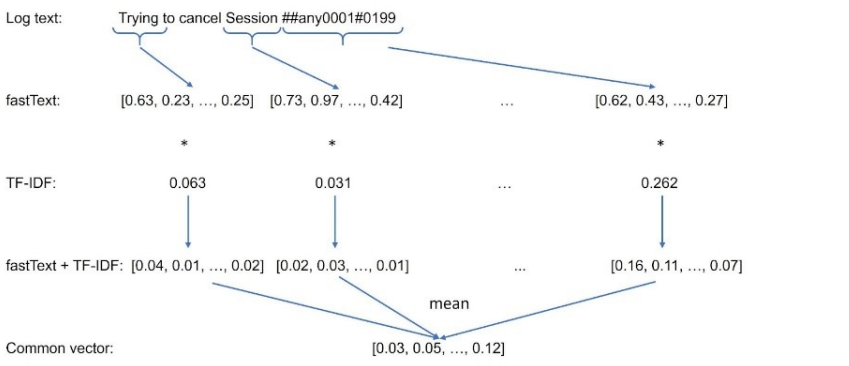

Figure 3. Log vectorization scheme

After preprocessing the data using a parser, the text information is converted into numeric information using the fastText NLP-model [3], which is trained on a training dataset. A special feature of this model is that previously unheard of words will still be assigned vectors using the so-called subword model. This advantage distinguishes this model from models such as Word2Vec [19] or Glove [20].

Since tens of thousands of individual events can be received in the logs per minute, they must first be grouped. In the work, this was done using a sliding window with a step equal to the window size. In this way, the logs are divided into equal time intervals with different numbers of events falling into them. The procedure for averaging vectors within one time interval is as follows: first, the vector of one event is calculated as the arithmetic mean of all words included in it, and then the arithmetic mean of all events in one interval is found. As a result, one time interval corresponds to one vector of the dimension specified during training of the fastText model. Words found in logs are not equivalent, so the ability to weight them using TF-IDF was also added. To do this, for each word within a time interval, its TF-IDF value is calculated, and then this value is multiplied by its vector. The same scheme was done for entire logs, which after the parser were combined into patterns. This approach allows us to reduce the importance of common words (log patterns) and increase it for rare ones. The general scheme of vectorization and averaging is shown in Figure 3. It is worth noting that the use of TF-IDF for weighting words and log patterns is optional and can be disabled. The results of the computational experiment show graphs both with and without weighting using TF-IDF.

Once the vectors for each time interval for both the training and test periods have been obtained, we can proceed to the problem of anomaly detection. In this setting, the time periods are known when there is confidence that the computer system operates without anomalies, and there is no other data labeling. Typically, as noted earlier, approaches with partial teacher involvement are used in such cases. Among the techniques tested in the work, the following algorithms should be noted: One Class SVM, Isolation Forest, Local Outlier Factor, Elliptic Envelope, k-nearest neighbors, and Autoencoder.

For each time period characterized by a vector, it is also possible to determine the contribution of log patterns to the total vector, i.e. decompose the vector into the sum of its subvectors. To do this, the projection of the vector of each pattern is found, and then it is normalized by the length of the total vector so that the sum of their contributions gives one. This allows us to find out which events had the greatest impact on the state vector and take the necessary measures to eliminate them. Figure 4 shows a diagram of this simple but useful interpretation.

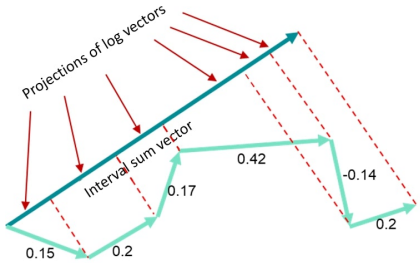

Figure 4. Scheme for determining the contributions of log patterns to the total vector

3. Results of the computational experiment

This section will discuss the experiments performed, the hyperparameters chosen, and the results.

Let's first look at the training data. As noted above, our own dataset, collected from the Vertica database logs, was used to train the model. The training data is 2 months of uninterrupted database operation in March and April, consisting of more than 800 million events. There are two test periods - late October - early November and late December - early January. Both test periods contain anomalies and database failures.

The first stage of building the model is to configure the parser on training data by training the semantic graph. The number of graph branches required for collapse, depending on the position of the token in the log (the closer to the beginning of the log, the more branches there can be), was selected analytically by selecting and evaluating various values, but in the future, automation of this process is planned.

Next, the fastText NLP-model is trained on the logs pre-processed by the parser with a given vector dimension of 100, since varying this parameter did not lead to significant changes in the results.

Along with the fastText model, TF-IDF is also trained, where time intervals act as document context. TF-IDF is trained separately for both words and log patterns. Several experiments were conducted in which TF-IDF was used to weight either only words, only log patterns, or both words and patterns.

Figure 5 shows the anomaly graphs for the test anomaly period with different set parameters: with TF-IDF calculation for log patterns only (Figure 5a), for words only (Figure 5b) and for log patterns and words simultaneously (Figure 5c). The units of anomaly measurement depend on the detection method used. Since the k-nearest neighbors method was used in Figure 5, the anomaly measure is the average distance to a given number of vectors from the training data set. Exceeding the threshold value is indicated by a scarlet indicator, and a long-term excess of this threshold is indicated by a red indicator. The threshold value is estimated based on the training data, for which the anomaly coefficient is also estimated and outliers are removed, for example, using quantiles. For example, in the work, the threshold value was equal to 0.99 quantiles of the anomaly coefficient for the training data set consisting of 830 million events. This is done to filter out single anomalies. You can also filter out outliers in the training dataset using other unsupervised learning techniques, for example, you can use Isolation Forest or Local Outlier Factor.

All graphs clearly show the occurrence of anomalies in the middle of the test period on October 28, which led to the failure on October 30, but other anomalous zones are also highlighted.

Figure 6 shows the same calculations for another test period with an anomaly that caused the failure on January 4. In these graphs, anomalies immediately before the failure are recorded only for the model with the log-only TF-IDF calculation (Figure 6a).

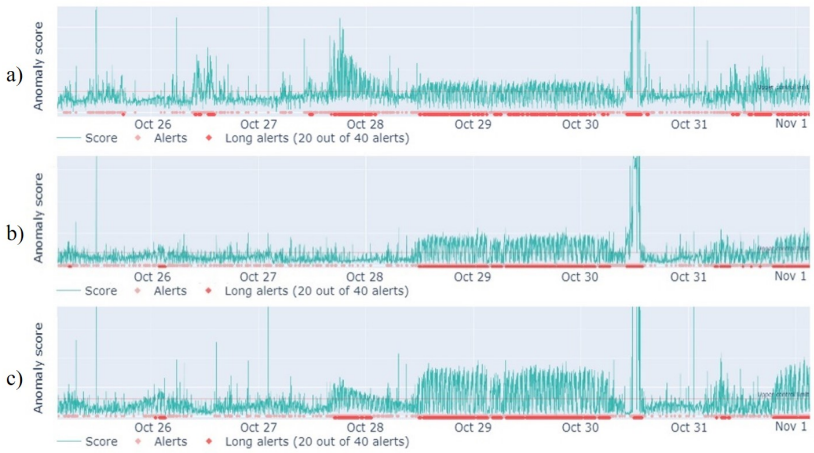

Figure 5. Anomaly graphs for the 1st test period with TF-IDF calculation: a) only for log patterns; b) only for words; c) for log patterns and words simultaneously

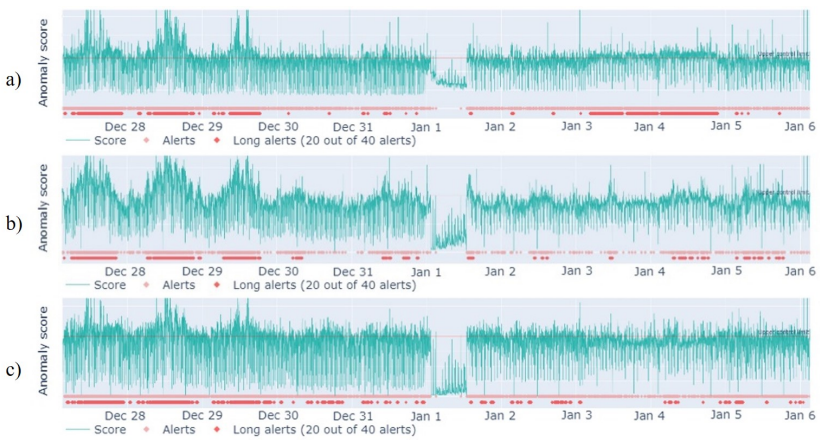

Figure 6. Anomaly graphs for the 2nd test period with TF-IDF calculation: a) only for log patterns; b) only for words; c) for log patterns and words simultaneously

The two cases considered show that calculating TF-IDF only for log patterns allows detecting anomalies before failure better than in other cases, but this is not the only advantage of this approach, which we will follow further.

It is also important to note that each log contains a tag, such as `<INFO>`, `<ERROR>`, etc., which were not involved in training the model, but the results clearly show the moment of database failure in the middle of the day on October 30 (Figure 5a), characterized by a surge in the number of fatal errors, shown in Figure 7a. But the graphs in Figure 5a, where TF-IDF was calculated only for log patterns, clearly show a correlation with the graph of the number of `<ERROR>` errors (Figure 7b), although information about them was not involved in the model. This once again proves the correctness of choosing the TF-IDF calculation method as the best model.

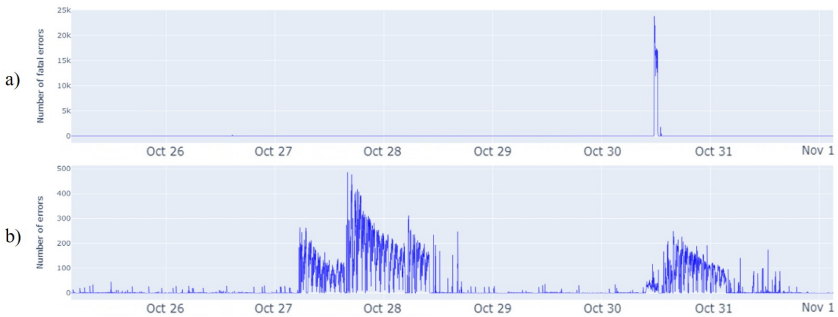

Figure 7. Fatal errors (a) and errors (b) graphs

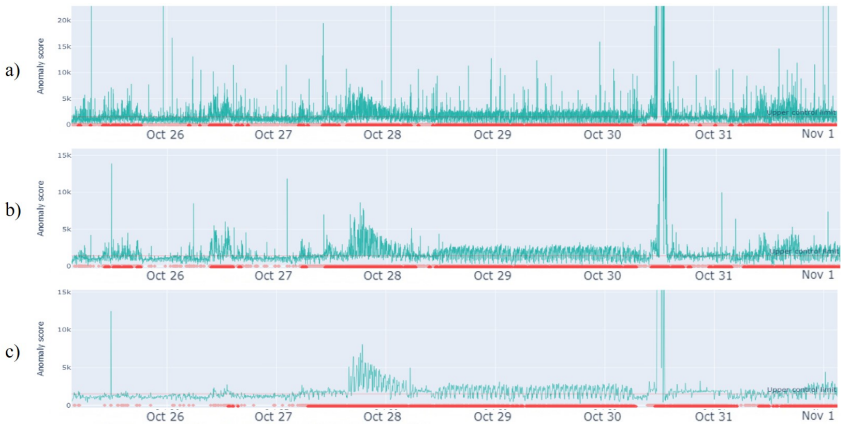

Figure 8. Anomaly graphs for different time interval sizes: 0.2 min (a), 1 min (b) and 5 min (c)

In the previous figures, the time interval size of one minute was the same everywhere. This hyperparameter was also changed to 0.2 and 5 minutes (Figure 8) for the model from Figure 5a. It is worth noting that when the interval size decreases, the computational complexity increases, therefore, the used physical and RAM memory increases, and the calculation time increases. The minimum possible interval size should also not be less than the estimated inference time of the model. A significant increase in the interval leads to excessive smoothing of the resulting vector due to averaging, so there is a high probability of missing an abnormal period.

As can be seen from the graphs, decreasing the interval size resulted in a larger number of anomaly bursts in Figure 8a compared to the one-minute interval (Figure 8b), while increasing it to 5 minutes (Figure 8c) resulted in some anomalies at the beginning of the test period disappearing due to averaging. Thus, it can be concluded that the selection of the time interval size should be left to the user, since it is necessary to find a balance between the accuracy of the model, its computational complexity and interpretability.

Figure 9 shows the graphs for the 1st test anomaly period with a comparison of the methods described in the introduction for the model with TF-IDF calculation only for log patterns (Figure 5a). It is worth noting that the k-nearest neighbors method, despite its simplicity, showed itself to be no worse than the other methods. All graphs clearly show the occurrence of anomalies in the middle of the test period, which led to failure.

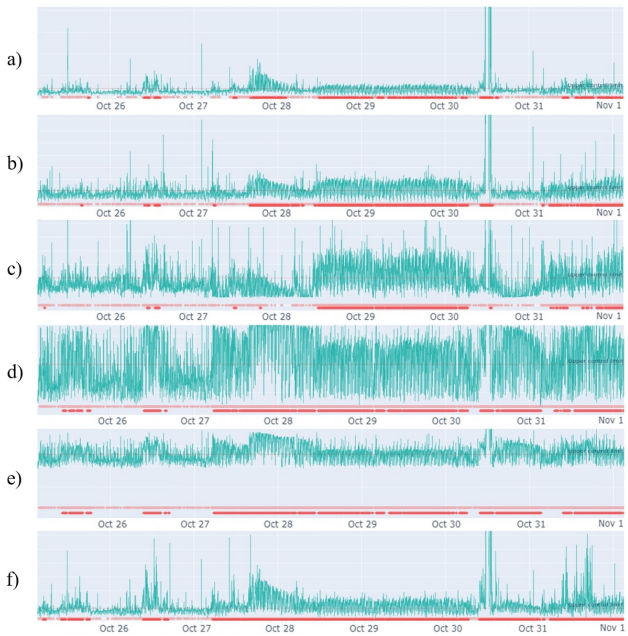

Figure 9. Anomaly plots constructed by the following methods: a) k-nearest neighbors, b) Elliptic Envelope, c) Local Outlier Factor, d) One Class SVM, d) Isolation Forest, e) Autoencoder

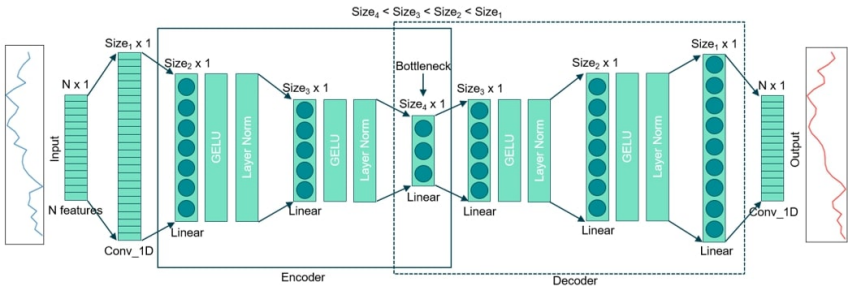

Figure 10. Schematic diagram of the Autoencoder used

The schematic of the Autoencoder used to calculate the anomaly of the test period shown in Figure 9f is shown in Figure 10. At the beginning of this neural network there are three consecutive encoder layers, transforming the original 100-dimensional vector space into a latent 10-dimensional one, called bottleneck. Then there are 3 decoder layers, returning to the original 100-dimensional vector space. The activation function was used by GeLU. The loss function during training of the autoencoder is set as Huber, combining the advantages of MSE and MAE.

As can be seen from Figure 9, the graphs turned out to be quite different, but they all highlight the period of database failure in the middle of the day on October 30, as well as the abnormal period preceding the failure. However, the correlation with the error graph (Figure 7b) is most clearly traced for the k-nearest neighbors method, so this method was chosen as the base one, and it is also easier to interpret and calculate.

Table 1

The importance of patterns in anomalies for one time interval

Log pattern	Pattern importance	Log tag	Number of patterns in interval
`Init Session ##any:0x7ea7e976a#:0x7f8eab7f0 [Basics] <WARNING> AuditedMemPool system tables deparse expression cannot reserve 100 MB of memory for planning [a0000043031afd,1]`	0.68719	`<WARNING>`	8122
`Init Session ##any:0x7ea7e976a#:0x7f8eab7f0 [Basics] <WARNING> AuditedMemPool system tables deparse expression Cannot release memory for [a0000043031afd,1], ResourceManager claims ...`	0.31267	`<WARNING>`	8122
`Init Session ##any:0x7ea7e976a#:0x7f8eab7f0 [Basics] <WARNING> MemoryPool static OPT::Plan* OPT::OptimizerInterface::makePlan (CAT::VQuery*, OPT::OptimizerConfig&) is using more mem...`	7.7252e-06	`<WARNING>`	103
`Init Session ##any:0x7ea7e976a#:0x7f8eab7f0 <NOTICE> @v_dwh_node0004: 00000/2001: NOTICE OF LICENSE NON-COMPLIANCE Continued use of this database is in violation of the...`	4.7842e-06	`<NOTICE>`	43
`Init Session ##any:0x7ea7e976a#:0x7f8eab7f0 <LOG> @v_dwh_node0004: 00000/6433: TLS session started for server`	3.8770e-06	`<LOG>`	40

In the future, it is planned to collect datasets for calculating metrics to determine the quality of the model and select optimal hyperparameters. At the moment, based on the results obtained from the two considered real cases of failure, further research has optimistic forecasts.

Let us consider one anomalous minute interval separately in more detail. As noted earlier, the vector of one interval can be decomposed into the sum of the vectors of its components with an assessment of their contribution. Table 1 shows an example of such a decomposition for the anomalous period from `2023-10-30 11:27` to `2023-10-30 11:28`. As can be seen from the table, the greatest contribution to the total vector was made by patterns with the `<WARNING>` tag, which indicate previously unseen events.

4. Conclusion

In this study, we demonstrated the results of applying semi-supervised learning techniques to solve the problem of anomaly detection in computer systems using the Vertica database as an example. The work was aimed at studying the possibility of using predictive maintenance approaches typical for technical equipment in relation to computer systems. It should be noted that even relatively simple algorithms demonstrated satisfactory efficiency in solving the problem. However, in future studies, we plan to test more complex anomaly detection techniques, as well as improved text data vectorization

algorithms. In addition, we plan to expand the experimental base by collecting additional database failure incidents, conduct a comprehensive quality assessment using relevant metrics, and select the most optimal algorithm and its hyperparameters.

Author Contributions: Conceptualization, Vladislav A. Kiriachek and Soltan I. Salpagarov; methodology, Soltan I. Salpagarov; software, Vladislav A. Kiriachek; validation, Vladislav A. Kiriachek and Soltan I. Salpagarov; formal analysis, Soltan I. Salpagarov; investigation, Vladislav A. Kiriachek; resources, Vladislav A. Kiriachek; data curation, Vladislav A. Kiriachek; writing—original draft preparation, Vladislav A. Kiriachek; writing—review and editing, Vladislav A. Kiriachek and Soltan I. Salpagarov; visualization, Vladislav A. Kiriachek; supervision, Soltan I. Salpagarov; project administration, Soltan I. Salpagarov. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data sharing is not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, P., Zhu, J., Zheng, Z. & Lyu, M. R. Drain: An online log parsing approach with fixed depth tree. *IEEE International Conference on Web Services (ICWS),* 33–40. doi:10.1109/ICWS.2017.13 (2017).
2. Du, M. & Li, F. Spell: Streaming parsing of system event logs. *2016 IEEE 16th International Conference on Data Mining (ICDM),* 859–864. doi:10.1109/ICDM.2016.0103 (2016).
3. Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* **5,** 135–146. doi:10.1162/tacl_a_00051 (2017).
4. Zhang, X. *et al.* Robust log-based anomaly detection on unstable log data. *ESEC/FSE,* 807–817. doi:10.1145/3338906.3338931 (2019).
5. Lu, S., Wei, X., Li, Y. & Wang, L. Detecting anomaly in big data system logs using convolutional neural network. *In 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech),* 151–158. doi:10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037 (2018).
6. Du, M., Li, F., Zheng, G. & Srikumar, V. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security,* 1285–1298. doi:10.1145/3133956.3134015 (2017).
7. Meng, W. *et al.* LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. *In IJCAI* **7,** 4739–4745. doi:10.24963/ijcai.2019/658 (2019).
8. Guo, H., Yuan, S. & Wu, X. LogBERT: Log Anomaly Detection via BERT. *In 2021 international joint conference on neural networks,* 1–8. doi:10.48550/arXiv.2103.04475 (Mar. 2021).
9. Yang, L., Chen, J., Wang, Z., Wang, W., Jiang, J., Dong, X. & Zhang, W. Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE),* 1448–1460. doi:10.1109/ICSE43902.2021.00130 (2021).
10. Nedelkoski, S., Bogatinovski, J., Acke, A., Cardoso, J. & Kao, O. Self-attentive classification-based anomaly detection in unstructured logs. *In 2020 IEEE international conference on data mining,* 1196–1201. doi:10.1109/ICDM50108.2020.00148 (2020).
11. Farzad, A. & Gulliver, T. A. Unsupervised log message anomaly detection. *ICT Express,* 229–237. doi:10.1016/j.icte.2020.06.003 (2020).

12. Wang, Q., Zhang, X., Wang, X. & Cao, Z. Log Sequence Anomaly Detection Method Based on Contrastive Adversarial Training and Dual Feature Extraction. *Entropy* **24,** 69. doi:10.3390/e24010069 (Dec. 2021).
13. Wan, Y., Liu, Y., Wang, D. & Wen, Y. *GLAD-PAW: Graph-Based Log Anomaly Detection by Position Aware Weighted Graph Attention Network* in (May 2021). doi:10.1007/978-3-030-75762-5_6.
14. Catillo, M., Pecchia, A. & Villano, U. AutoLog: Anomaly detection by deep autoencoding of system logs. *Expert Systems with Applications* **191.** doi:10.1016/j.eswa.2021.116263 (2022).
15. Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J. & Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation* **13(7),** 1443–1471. doi:10.1162/089976601750264965 (2001).
16. Liu, F. T., Ting, K. M. & Zhou, Z. H. Isolation Forest. *2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy,* 413–422. doi:10.1109/ICDM.2008.17 (2008).
17. Breunig, M., Kröger, P., Ng, R. & Sander, J. LOF: Identifying Density-Based Local Outliers. *ACM Sigmod Record* **29,** 93–104. doi:10.1145/342009.335388 (June 2000).
18. Rousseeuw, P. J. & Van Driessen, K. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* **41(3),** 212. doi:10.1080/00401706.1999.10485670 (1999).
19. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient estimation of word representations in vector space. doi:10.48550/arXiv.1301.3781 (2013).
20. Pennington, J., Socher, R. & Manning, C. D. Glove: Global vectors for word representation. *In Proceedings of the 2014 conference on empirical methods in natural language processing,* 1532–1543. doi:10.3115/v1/D14-1162 (2014).

Information about the authors

Kiriachek, Vladislav A.—PhD student of Department of Computational Mathematics and Artificial Intelligence of Peoples' Friendship University of Russia (RUDN University) (e-mail: w.a.kiryachok@mail.ru, phone: +7(968)8521346, ORCID: 0009-0002-9692-0225, Scopus Author ID: 57220041155)

Salpagarov, Soltan I.—Candidate of Physical and Mathematical Sciences, associate Professor of Department of Computational Mathematics and Artificial Intelligence of Peoples' Friendship University of Russia (RUDN University) (e-mail: salpagarov-si@rudn.ru, phone: +7(903)2716575, ORCID: 0000-0002-5321-9650, Scopus Author ID: 57201380251)

УДК 004.62:004.822
PACS 07.05.Mh, 07.05.Kf
DOI: 10.22363/2658-4670-2025-33-2-172-183 EDN: MGCVKV

Предиктивная диагностика логов компьютерных систем с помощью методов обработки естественного языка

В. А. Кирячёк, С. И. Салпагаров

Российский университет дружбы народов, ул. Миклухо-Маклая, д. 6, Москва, 117198, Российская Федерация

Аннотация. Данное исследование направлено на разработку и валидацию метода предиктивной диагностики и детекции аномалий в логах компьютерных систем, используя в качестве примера базу данных Vertica. Предложенный подход основан на обучении с частичным привлечением учителя в сочетании с методами обработки естественного языка. Для предварительной обработки данных разработан специализированный парсер, использующий семантический граф. Векторизация осуществлялась с применением NLP-библиотеки fastText и взвешивания TF-IDF. Эмпирическая валидация проводилась на реальных лог-файлах Vertica крупной IT-компании, содержащих как периоды нормального функционирования, так и аномалии, приведшие к сбоям. Проведена сравнительная оценка эффективности различных алгоритмов обнаружения аномалий, включая метод k-ближайших соседей, автоэнкодеры, One Class SVM, Isolation Forest, Local Outlier Factor и Elliptic Envelope. Результаты визуализированы посредством графиков аномальности, отражающих временные интервалы с превышением порогового уровня. Полученные результаты демонстрируют высокую эффективность предложенного подхода в идентификации предшествующих сбоям аномалий и определяют перспективные направления дальнейших исследований.

Ключевые слова: машинное обучение, методы обработки естественного языка, анализ логов, детекция аномалий, предиктивная диагностика