




Leveraging Anomaly Detection for Proactive Application Monitoring

Shyam Zacharia^(✉) 

British Telecom, Bengaluru, India
shyam.zacharia@bt.com

Abstract. Anomaly detection is one of the popular research fields in Machine Learning. Also, this is one of the key techniques in system and application monitoring in Industry. Anomaly detection comprises of outlier detection and identifying novelty from the data - it is a process to understand the deviation of an observation from existing observations [12] and identifying the new observations. Carrying out anomaly detection in an enterprise application is a challenge as there are complex processes to gather and analyze functional and non-functional logs of unlabeled data. In this paper we are proposing an unsupervised learning process with log featurization incorporating time window to detect outliers and novel errors from enterprise application logs.

Keywords: Anomaly detection · Outlier detection · Novelty detection

1 Introduction

Application monitoring with a real-world dataset is a complex task as huge quantities and a variety of log information are getting generated in every minute. The log information can be functional (application functionality logs) and non-functional (system operation logs, performance logs, security logs etc.), and these are unlabeled information. There exist plenty of anomaly detection algorithms in different categories for addressing the need. The log information is dynamic in nature as it is generated during a process journey and depends upon the data values in each step. And new logs will also be added during each cycle of application development. Rule based log capturing and anomaly detection methods will have enormous issues in these situations, as rules cannot cope with these frequent and dynamic changes. Our proposed method can capture the log in the same format and can also find the outliers and novel errors in the logs. We consider outliers as the errors which are deviant from the normal concentrated observation of data. We also define novel errors as errors which are not seen in the training data.

In this paper, we address the problem of anomaly detection from the log dataset by converting the log information to appropriate featurization and applying different unsupervised anomaly detection algorithms. The end-to-end strategic approach to tackle outlier and novelty detection within enterprise application logs is introduced in this paper.

The paper is organized as follows: In Sect. 2, we will highlight the related work in this area. Section 3 describes the proposed approach to the problem, and Sect. 4 contains

experiment results and benefits. Section 5 concludes the paper and presents future work. Acknowledgements for the paper follow.

2 Related Works

The mechanism to capture operational logs has been in existence since a long time. But most of the techniques were using a regex expression to parse the log file [8–10] to build the features from the logs and those patterns were used for detecting the outliers. Here the outlier detection is subjective to specific functionality. There are ways to incorporate the operational procedure along with the log information to identify anomalies [3, 11]. As mentioned, there exist plenty of Anomaly detection algorithms to identify the outlier and novelty. Some of the example algorithms are mentioned in [1, 2, 4, 5]. In the proposed approach we are portraying the anomaly detection by considering only the text information of functional or non-functional process logs.

3 Proposed Approach

For most of the addressed enterprise applications, the logs are generated in text format. This text format will be in accordance with the style of each application's logging procedures. In order to debug the errors, the entities such as time stamps, error codes, error messages and other related details are present in the log file. Some of these errors can be ignored as they will not cause any harm to the functionality of the application. But there are threatening errors which need to be addressed quickly to ensure application functionality.

We have considered three different types of logs generated by three different types of application/servers for our experiments, namely Siebel CRM (Customer Relationship Management) logs, Apache server logs and, WebLogic server logs. In each of these applications, we have followed the same approach and obtained consistent results at the end. The detailed overview will be mentioned based on the process we followed with the Siebel Application, but the same methods can be applied for all kinds of application logs.

3.1 Outlier Detection

An outlier is an observation which deviates from the course of other observations [12] or appears inconsistent with the remainder of the dataset [13]. As per [1–3] there are multiple ways to identify outliers. In the proposed method, we are using unsupervised outlier detection methods as we have the application log as input, which is not labelled. In the log details, we have functionality related errors which can originate from system issues in addition to data issues, programming issues, etc. In the non-functional logs there are issues such as server related - both hardware and software issues, network related issues, operational issues, performance issues, etc. The log files contain informational logging, warning contents and error contents. Our aim is to identify and isolate errors which appear inconsistent with previous appearances.

3.2 Novelty Detection

Novelty detection is an unsupervised learning technique to identify novel errors from the logs. The errors which have not occurred in training data are captured by this method. These novel errors will arise in case of addition of new functionality in the application, during system revamp, or in application upgrade scenarios.

These new errors in test data can be captured during our feature vector conversion, which is done using a Word2Vec [7] model. The log entry tokens which are not part of Word2Vec model are the actual novel errors. The words in these errors, unknown to the already formed Word2Vec model, are mapped to default vector values and concatenated to form the sentence feature vector which is then passed to the anomaly detection algorithm.

3.3 Implementation and Model Training

The approach in Fig. 1. has been taken for model training and prediction.

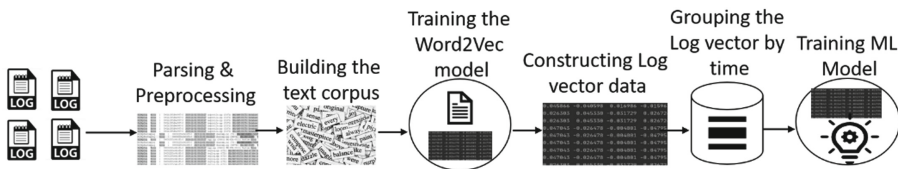


Fig. 1. Machine learning model training steps

- Removing the log information for the known issues:** All the log files generated in certain time period, for example in past one month, must be collected for building the training data. Only those log files generated when the application was working normally are utilized. In addition, only the logs which are captured with minimum log level settings are used for training purposes.
- Parsing and preprocessing the log files:** The relevant information such as event type, event subtype, time stamp and log message need to be parsed from each log file. Only the logs which are of the type ‘Error’ are then extracted for further processing, excluding information and warning related entries. Stop words and punctuation are also removed as part of the text cleaning process.
- Building the text corpus:** With the cleaned data, a new text corpus is created by tokenizing each word from the sentence list from the log files.
- Training the Word2Vec model:** The Word2Vec [7] model is built using the previously constructed corpus text data. The Gensim [6] package has been used for the purpose.
- Constructing the feature vector data:** In order to construct the feature vectors in each log entry, the vector values equivalent to the preprocessed event type, event subtype and log message are concatenated. Each of these sub item vectors are the average of word vectors in each item.

- **Bucketizing the data with time window:** The records need to be bucketized based on the time window, in order to calculate the error count in each time window. With the timestamp in the log data, the log record count is calculated within each time window.
- **Training the Machine Learning Model:** The bucketized data is used as an input to build the machine learning model. We have tried different unsupervised algorithms to get the best results. The experiment results are shown in the next section.

4 Experiment Results and Application

Siebel Application component logs are used as training data to detect outlier and novel errors. From the component logs, event type, event sub type and error log are extracted and use to form the feature vector. Each of this individual concatenated feature vector has dimension of size 10. After grouping based on time window, the error count is taken and concatenated with the feature vector. The model is then trained to predict whether an error is present for each feature vector created. Training Data details are given in Table 1.

Table 1. Data details for building the model

#Samples	#Dimension	Outlier perc
459	31	27.27

We have run the experiment with ten different algorithms [Angle-based Outlier Detector (ABOD) [16], Cluster-based Local Outlier Factor(CBLOF) [17], Feature Bagging(FB) [18], Histogram-base Outlier Detection (HBOS) [19], Isolation Forest(IF) [20], K Nearest Neighbors (KNN) [21], Local Outlier Factor (LOF) [5], Minimum Covariance Determinant (MCD) [22], One-class SVM (OCSVM) [15], Principal Component Analysis (PCA) [23]. We have used area under receiver operating characteristic (ROC) curve [24] and Precision @ rank n (P@N) [25] as evaluation metrics. The ROC performances and P@N performances (average of 10 independent trials) of each algorithm are shown as results in Table 2. The experiment conducted with the help of PyOD [14] package.

Table 2. ROC Performance and Precision @N Performance (average of 10 independent trials)

Algorithms	ABOD	CBLOF	FB	HBOS	IForest	KNN	LOF	MCD	OCSVM	PCA
ROC	0.12	0.916	0.833	0.708	0.733	0.9583	0.917	0.717	1	0.625
P@N	0.625	0.875	0.875	0.857	0.875	0.875	0.875	0.875	1	0.875

From Table 2 it can be seen that the OneClass SVM is doing a better job in predicting both outlier and novel errors from the input data. As per [15] OneClass SVM is able to

detect the error distribution from existing distribution, and the errors which are novel to the underlying error distribution. The error message tokens which did not exist in the Word2Vec corpus model, are the actual novel errors which are detected in the OneClass SVM.

Real World Application: We have implemented the OneClass SVM based anomaly detection in our enterprise application such as Siebel CRM, and other various applications using WebLogic servers. The implementation allows us to detect the outlier errors in the application, which help in proactively identifying the issues and resolving it in quicker way. Also, during the release cycle or in system upgrade time, we could easily identify novel errors and fix them without impacting the customer.

5 Conclusion

Enterprise level application log anomaly detection was attempted with the different algorithms and it was found that OneClass SVM yielded promising results in the feature vector approach. The outlier data and novel data together were difficult to capture in most of the anomaly detection algorithms as per their design. OneClass SVM was the most successful in detecting both types of errors, scoring the highest in both evaluation metrics. Additionally, this process helps in finding application related problems in a systematic way. In future there is scope of reducing the training time of the process in a more optimized approach.

Acknowledgement. This work has been conducted using the resources and data in British Telecom. I thank all the reviewers and supporters who helped me in accomplishing this task. I also express my sincere gratitude to BT Research and to my Managers for their extended help.

References

1. Wang, H., Bah, M., Hammad, M.: Progress in outlier detection techniques: a survey. *IEEE Access*, 7, 107964–108000 (2019). <https://doi.org/10.1109/access.2019.2932769>
2. Akoglu, L., Tong, H., Vreeken, J., Faloutsos, C.: Fast and reliable anomaly detection in categorical data. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management - CIKM '12* (2012). <https://doi.org/10.1145/2396761.2396816>
3. He, S., Zhu, J., He, P., Lyu, M.: Experience report: system log analysis for anomaly detection. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (2016). <https://doi.org/10.1109/issre.2016.21>
4. Baur, C., Wiestler, B., Albarqouni, S., Navab, N.: Deep autoencoding models for unsupervised anomaly segmentation in brain MR images. In: Crimi, A., Bakas, S., Kuijff, H., Keyvan, F., Reyes, M., van Walsum, T. (eds.) *BrainLes 2018*. LNCS, vol. 11383, pp. 161–169. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11723-8_16
5. Breunig, M., Kriegel, H., Ng, R., Sander, J.: LOF. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data - SIGMOD '00* (2000)
6. Gensim Home Page. <https://radimrehurek.com/gensim/models/word2vec.html>. Accessed 25 Jun 2020

7. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: *Advances in Neural Information Processing Systems*, pp. 3111–3119 (2013)
8. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Detecting large-scale system problems by mining console logs. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09* (2009)
9. Yamanishi, K., Maruyama, Y.: Dynamic syslog mining for network failure monitoring. In: *Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining - KDD '05* (2005)
10. Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. *NSDI*, pp. 353–366 (2012)
11. Ghanbari, S., Hashemi, A.B., Amza, C.: Stage-aware anomaly detection through tracking log points. In: *Proceedings of the 15th International Middleware Conference*, pp. 253–264 (2014). <https://doi.org/10.1145/2663165.2663319>
12. Hawkins, D.: *Identification of Outliers*. Chapman and Hall, London (1980)
13. Barnett, V., Lewis, T.: *Outliers in Statistical Data*. Wiley, New York (1994)
14. Zhao, Y., Nasrullah, Z., Li, Z.: PyOD: a python toolbox for scalable outlier detection. *J. Mach. Learn. Res.* **20**(96), 1–7 (2019)
15. Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., Williamson, R.: Estimating the support of a high-dimensional distribution. *Neural Comput.* **13**, 1443–1471 (2001)
16. Kriegel, H.P., Schubert, M., Zimek, A.: Angle-based outlier detection in high-dimensional data. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 444–452 (2008)
17. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. *Pattern Recogn. Lett.* **24**, 1641–1650 (2003)
18. Lazarevic, A., Kumar, V.: Feature bagging for outlier detection. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 157–166 (2005)
19. Goldstein, M., Dengel, A.: Histogram-based outlier score (hbos): a fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pp. 59–63 (2012)
20. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422 (2008). <https://doi.org/10.1109/icdm.2008.17>
21. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 15–27 (2002)
22. Hardin, J., Rocke, D.: Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Comput. Stat. Data Anal.* **44**, 625–638 (2004)
23. Shyu, M.L., Chen, S.C., Sarinnapakorn, K. and Chang, L.: A novel anomaly detection scheme based on principal component classifier. *Miami Univ Coral Gables FL Dept of Electrical and Computer Engineering* (2003)
24. Hanley, J., McNeil, B.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* **143**, 29–36 (1982)
25. Craswell, N.: Precision at n. *Encyclopedia of Database Systems*, pp. 2127–2128 (2009)