Fall 11-5-2022

A Simple Approach to Detect Anomalies in Microservices-Based Systems Using PyOD

Lauriana Tavares Landim
Instituto Politécnico de Castelo Branco, lauriana.landim@ipcbcampus.pt

Luís Barata
Instituto Politécnico de Castelo Branco, luis.barata@ipcb.pt

Eurico Lopes
Instituto Politécnico de Castelo Branco, eurico@ipcb.pt

A Simple Approach to Detect Anomalies in Microservices-Based Systems Using PyOD

Lauriana Tavares Landim, Instituto Politécnico de Castelo Branco, Portugal, lauriana.landim@ipcbcampus.pt

Luís Barata, Instituto Politécnico de Castelo Branco, Portugal, luis.barata@ipcb.pt

Eurico Lopes, Instituto Politécnico de Castelo Branco, Portugal, eurico@ipcb.pt

Abstract

Ease of scale is one of the defining characteristics of microservices. However, with scalability comes the problem of diversity of services, making it very important to detect anomalies the soonest possible. Because it is recent, there are still few studies on the best approaches to detecting anomalies in microservices. This paper proposes the Python toolkit, PyOD, as an approach for microservice anomaly detection. This toolkit is composed of a set of anomaly detection algorithms, including classical LOF (SIGMOD2000) to the latest ECOD (TKDE2022). To evaluate the approach, we used two of its algorithms, k Nearest Neighbors (kNN) and Histogram-based Outlier Score (HBOS) to detect anomalies such as application bugs, CPU exhausted, and network jam on the TraceRCA dataset. This dataset contains logs from a real microservices system. The preliminary results show that HBOS algorithm performs better than kNN, with Recall and F1-Score of 93% and 89%, respectively, while for kNN these metrics were 92% and 85%, respectively.

Keywords: anomaly detection; PyOD; outliers algorithms; microservices

1. INTRODUCTION

Microservices architecture has been gaining popularity, so much so that large enterprises have adopted this new architecture by migrating to microservices from a monolithic application or developing microservices-based applications from scratch. The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services (Fowler & Lewis, 2014). Instead of building a single hudge, monolithic application, the idea is to split an application into a set of smaller and interconnected services (Richardson, 2016). Meanwhile, services tend to increase quickly and become more complex, so it is essential to detect and diagnose possible failures promptly.

Accordingly to Cao, Cao and Zhang (2019), although there are a lot of anomalies types in microservice, these can be classified in Request Exception, Runtime Exception, Timeout Exception and Other Exception. Other Exception include Security exception and Version Exception. Within the Runtime Exception category there are more anomalies than in the other categories.

ISSN 2183-489X
DOI http://dx.doi.org/10.18803/capsi.v22.177-186

Some investigations to detect anomalies in microservices have already been conducted, primarily using techniques such as log analysis and monitoring, trace comparison, statistical methods, and clustering-based techniques. However, such approaches, are not directed to using dedicated programming languages and libraries for anomaly detection. This paper uses PyOD, an open-source Python toolbox for performing scalable outlier detection on multivariate data Zhao, Nasrullah and Li (2019). PyOD provides access to more than 40 detection algorithms, including probabilistic and proximity-based models. Through its wide range of algorithms, we choose k Nearest Neighbors (kNN) and Histogram-based Outlier Score (HBOS) to predict anomalies in our dataset and compare the results.

2. MICROSERVICE ARCHITECTURE

(Fowler & Lewis, 2014), define this architecture as an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. According to IBM Cloud Education (2021), microservices are widely used and benefit many industries worldwide. These benefits include faster delivery, improved scalability, more flexibility in technology choice, and greater autonomy. Twitter, Netflix, Apple, and eBay are examples of organizations from various domains that take advantage of these benefits to adopt microservices-based approaches.

Instead of a single monolithic unit, applications built using microservices are made up of autonomous, loosely coupled services, in other words, loosely dependent on each other. An example of this architectural style is shown in figure 1. As pointed out by Newman (2015), when services are loosely coupled, a change to one service should not require a modification of another. The whole point of a microservice is to make a change to one service and deploy it without needing to change any other part of the system. The concept of microservices is also often accompanied by the definition of the Single Responsibility Principle, coined by Robert C. Martin. Furthermore, according to Newman (2015), the principle states "*Gather together those things that change for the same reason and separate those things that change for different reasons*.", reinforcing the idea of autonomous and independent services.

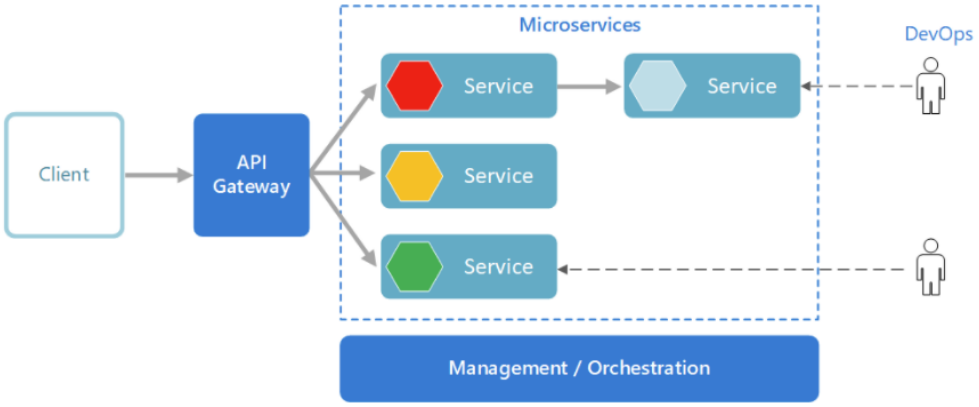

Figure 1 - Microservices architecture style. Source (Anita Bueherle, 2016)

Although different authors have presented different definitions for Microservices Architecture, some characteristics predominate in all of them. Based on the insights of Newman (2015), Bruce and Pereira (2019), and Fowler and Lewis (2014), we can conclude that these characteristics could be summarized in the following:

- Componentization via services - services are independent and focused on doing a task well, following the Single Responsibility Principle.
- Autonomy - services are developed around business resources; each service is a separate entity that operates and changes independently of the others.
- Resilience and Failure Isolation - applications need to be designed in such a way that they can tolerate service failure.
- Decentralized Database Management - each service manages its database, either in different instances of the same database technology or in entirely different database systems.
- Many languages, many options - each service can choose the best combination of technology for its use cases.

3. RELATED WORK

As a background for this work, we considered some academic studies in the area to understand each method used.

Meng et al. (2021) proposed an anomaly detection approach for microservices applications by comparing traces. As a dataset, they used Bench4Q to demonstrate the proposed technique and validate the approach; they used a microservices-based application called Social Network Yashchenko (2016). Cao et al. (2019) used the Conditional Random Field (CRF) based anomaly detection method and proved through experiments that the process could accurately find the faults in microservice system. The accuracy and the recall rate are relatively high. Barakat (2017) used the Kieker framework for performance monitoring and analysis of a microservices-based application and determined that some parameters about the performance of services can be visualized. Sun et al.

(2021) used the MonitorRank and FSB-RWRANK algorithms to develop service dependency graph automatically and thereby monitor microservices indicators and detect possible anomalies. Li et al. (2021) detected abnormal traces in microservices using a trace analysis-based approach called TraceRCA. They used the Train-Ticket benchmark to perform the experiments and deployed with Kubernetes. Wang et al. (2020) proposed an automatic anomaly diagnosis approach for microservices-based applications with statistics. They built baselines using Call Trees, and the results showed that the approach could accurately locate microservices causing anomalies.

4. DETECTION OF ANOMALIES IN MICROSERVICES

In microservices-based systems, anomaly detection is crucial because, as they grow larger, these types of systems sometimes become more prone to failure. For Chandola et al. (2009), anomaly detection refers to the problem of finding patterns in data that do not conform the expected behavior. These nonconforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains. Anomalies in a system can be defined as deviations from normal behavior. Anomalies often indicate that a component is failing or is about to fail and should hence be detected as soon as possible (Forsberg, 2019).

Though the focus of this paper is on anomaly detection in microservices, anomaly detection has been widely applied in countless application domains such as medical and public health, fraud detection, intrusion detection, industrial damage, image processing, sensor networks, robots behavior and astronomical data, as stated by Ahmed et al. (2016). The application of anomaly detection in the metrics emitted by microservices like CPU and memory utilization, error rate and response time, improves its reliability because these metrics provide important information about the components' performance, thus allowing to act in time in case some abnormal behavior is identified. To do this, it is necessary to know the normal state of the microservice. As per Düllmann (2017), this means that the existing data (e.g., response times) is used to learn the normal behavior of an application. To decide whether the current situation is an anomaly or not, thresholds can be set. This can happen either manually by setting the threshold to a fixed value or can be determined automatically using the normal behavior as an indicator.

5. DATASET

To train our model with PyOD, it was used TraceRCA dataset Li et al. (2021), a study data, from another research using trace analysis approach. This dataset contains two sets of data: A) and B). For this work, we used set B, which contains traces of a large Internet service provider's production microservice system. In addition, set B also includes a file with the faults injected into the application. Although our dataset is unsupervised, this fault file will be used only for the purpose of

evaluating the solution and not as part of training the model. The dataset file is in csv format and its 5 features include "timestamp", "latency", "succ", "source" and "target". These features are described in Table 1, along with their respective data types. Although the logs exceed 1,000,000 records, for testing purposes, random two-day records were used, corresponding to a total of 401.245. It should also be noted that the faults injected are of type CPU exhaustion, memory exhaustion, host network error, container network error, and database failures.

Feature	Description	Type
trace_id	Id of each single trace	String
timestamp	Precise indication of date and time when an request was made	Number
latency	The time in seconds it took for the request to be completed.	Number
Succ	Indicates whether the request was successful or not	Boolean
source	Indicates the source where the request came from	String
target	Indicates the target of each request	String

Table 1 - Description of the dataset features

6. APPROACH

There are many attempts to solve the anomaly detection problem. The more widely applicable approaches are unsupervised algorithms as they do not need labeled training data meeting the requirements of practical systems (Amer & Goldstein, 2012). PyOD is an open-source Python toolbox for performing scalable outlier detection on multivariate data (Zhao, Nasrullah, & Li, 2019). Some benefits of PyOD are that it is scalable, includes more than 40 algorithms, and can detect anomalies in multivariate data. As per the authors, since 2017, PyOD has been successfully used in various academic research and commercial products, such as (Zhao et al., 2021) and (Zhao, Nasrullah, Hryniewicki, et al., 2019).

A few steps were taken to detect anomalies in the dataset, using PyOD. First, the dataset were cleaned and the features, “latency” and “target”, that we consider to be more preditive were prepared and converted to a format that the algorithms could understand. Then we predict anomalies using detectors and PyOD functions, fit() and predict() to obtain the total number of outliers found in each one. After that, a function was developed specifically for calculating the confusion matrix. The purpose of this function is, based on the values predicted by the algorithm, to calculate True Positive, False Positive, True Negative and False Negative. For that, considering our dataset, the validation dataset (fault list) and the confusion matrix concepts, we established that to be :

True Positive - the "timestamp" of the prediction dataset must be between the time interval in which the fault was injected; the value returned by the predict() function must be equal to 1, which corresponds to outlier; and the " target " must be equal to the "ground_truth".

True Negative - the timestamp of the prediction dataset must be between the time interval in which the fault was injected; the value returned by the predict() function must be equal to 0; and the "target" must be equal to ground_truth.

False Positive - the timestamp of the prediction dataset must be outside the time range in which the fault was injected; and the value returned by the predict() function must be equal to 1.

False Negative - the timestamp of the prediction dataset must be outside the time range in which the fault was injected; and the value returned by the predict() function must be 0.

The pseudocode for this function is represented in table 1.

```
DEFINE FUNCTION getConfusionMatrix(faults, predicted):

    SET truePositive TO []
    SET trueNegative TO []
    SET falsePositive TO []
    SET falseNegative TO []
    SET predicted TO predicted.assign(date_time TO lambda x: df['timestamp'])

    FOR i, act IN faults.iterrows():

        SET minTime TO datetime.datetime.strptime(act.time_preliminary,'%Y-%m-%d %H:%M:%S+08:00')
        SET maxTime TO minTime + timedelta(minutes=5)



        truePositive.append(predicted.loc[(predicted.date_time  >= minTime) & (predicted.date_time <= maxTime)
& (predicted.outliers EQUALS 1) & (predicted.target EQUALS act.ground_truth)])

        trueNegative.append(predicted.loc[predicted.outliers EQUALS 0])

        falsePositive.append(predicted.loc[(predicted.date_time < minTime) | (predicted.date_time > maxTime) &
(predicted.outliers EQUALS 1)])

        falseNegative.append(predicted.loc[(predicted.date_time < minTime) | (predicted.date_time > maxTime) &
(predicted.outliers EQUALS 0)])




    ELSE:

        SET truePositive TO list(filter(lambda dfTP: not dfTP.empty, truePositive))
        SET trueNegative TO list(filter(lambda dfTN: not dfTN.empty, trueNegative))
        SET falsePositive TO list(filter(lambda dfFP: not dfFP.empty, falsePositive))
        SET falseNegative TO list(filter(lambda dfFN: not dfFN.empty, falseNegative))

    RETURN
```

Table 2 - Function to get Confusion Matrix

Ultimately, the result returned by the function, made it possible to calculate recall and F1 Score and determine which of the 2 algorithms had the best performance.

6.1. ALGORITHMS USED

The choice of algorithm to detect anomalies depends on a few factors. Some of these factors may be the type of dataset to be used, the number of features in the dataset, and whether the data is supervised or unsupervised. We evaluate our dataset with the following 2 algorithms, considering the characteristics of our dataset, which is considerable, multivariate, and unsupervised:

kth Nearest Neighbor (kNN) - The k-nearest neighbors algorithm, or kNN, is one of the simplest machine learning algorithms. Usually, k is a small, odd number - sometimes only 1. The larger k is, the more accurate the classification will be, but the longer it takes to perform the classification (*KNN*, 2019). According to Chandola et al. (2009), in this algorithm, the anomaly score of a data instance is defined as its distance to its kth nearest neighbor in a given data set.

Histogram-based Outlier Score (HBOS) – According to Goldstein & Dengel (2012), although this algorithm is only a combination of univariate methods unable to model dependencies between features, its fast computation is charming for large data sets. This is perfectly fitting for detecting anomalies in microservices that usually produce large numbers of logs and have many parameters to consider. Additionally, for every single feature (dimension), a univariate histogram is constructed first. If the feature comprises categorical data, a simple counting of the values of each category is performed, and the relative frequency (height of the histogram) is computed. For numerical features, two different methods can be used: (1) Static bin-width histograms or (2) dynamic bin-width histograms.

7. PRELIMINARY RESULTS

To compare the performance of our algorithms, it was necessary the use some metrics. From the literature review, it was possible to verify that recall and F1-score are among the most used metrics, to evaluate the performance of algorithms. Thus, we determined the confusion matrix and calculated those metrics to find the best algorithm.

Confusion Matrix

The confusion matrix provides more knowledge about the performance of our model by providing the information on correctly, or incorrectly classified classes through which we can identify errors (Tanouz et al., 2021). As mentioned in chapter 6, a function was developed to return the confusion matrix. The result is essentially the values of True Positive, False Positive, True Negative and False Negative, which was then used to calculate the following metrics:

Precision: number of classified correct outputs, we can say exactness of model.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall: the measure of our model correctly identifying True Positives

$$Recall\ = \frac{TruePositives}{TruePositives\ +\ FalseNegatives}$$

F1 score: Average of Precision and Recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The results, as presented in table 5 show that HBOS performs slightly better than KNN, with Recall and F1-Score of 93% and 89%, respectively, while for kNN these metrics were 92% and 85%, respectively.

ALGORITHMS	RECALL	F1 - SCORE
kNN	92%	85%
HBOS	93%	89%

Table 2 - Performance of the algorithms

8. CONCLUSION AND FUTURE WORK

The PyOD library, which provides access to over 40 algorithms for both supervised and unsupervised data, was used in TraceRCA dataset for anomaly detection. This work contributes significantly to the research community in an area that is still new - anomaly detection in microservices, This area, enables researchers to make further comparisons with other algorithms based on the results obtained using this library. The main point of PyOD is that, regardless of the dataset's characteristics, it is possible to choose among the various algorithms available to see which one fits best. After the prediction process, with "*latency*" and "*target*" as a feature, metrics were calculated to see which algorithm performed best.

For future works, we aim to expand the application of the algorithms to the entire dataset since usually, thousands of logs are produced per second in microservice systems. Furthermore, to achieve more precise results, we intend to evaluate other microservices dataset and aggregate more metrics. We also intend to do further investigations with new developments aimed at identifying the root cause of the anomalies.

REFERENCES

Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, *60*, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016

Amer, M., & Goldstein, M. (2012). *Nearest-Neighbor and Clustering based Anomaly Detection Algorithms for RapidMiner*. 12.

Anita Bueherle. (2016, October). *What are Microservices?* https://www.weave.works

Barakat, S. (2017). Monitoring and Analysis of Microservices Performance. *Journal of Computer Science and Control Systems*, *10*, 19–22.

Bruce, M., & Pereira, P. (2019). *Microservices in Action*.

Cao, W., Cao, Z., & Zhang, X. (2019). *Research on Microservice Anomaly Detection Technology Based on Conditional Random Field*.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 15:1-15:58. https://doi.org/10.1145/1541880.1541882

Düllmann, T. F. (2017). *Performance Anomaly Detection in Microservice Architectures Under Continuous Change*. 89.

Forsberg, V. (2019). *AUTOMATIC ANOMALY DETECTION AND ROOT CAUSE ANALYSIS FOR MICROSERVICE CLUSTERS*. 46.

Fowler, M., & Lewis, J. (2014, March 25). *Microservices a definition of this new architectural term*. https://martinfowler.com/articles/microservices.html

Goldstein, M., & Dengel, A. (2012, September 26). *Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm*.

IBM Cloud Education. (2021, June 23). *Microservices*. https://www.ibm.com/cloud/learn/microservices

KNN. (2019, May 17). DeepAI. https://deepai.org/machine-learning-glossary-and-terms/kNN

Li, Z., Chen, J., Jiao, R., Zhao, N., Wang, Z., Zhang, S., Wu, Y., Jiang, L., Yan, L., Wang, Z., Chen, Z., Zhang, W., Nie, X., Sui, K., & Pei, D. (2021). Practical Root Cause Localization for Microservice Systems via Trace Analysis. *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

Meng, L., Ji, F., Sun, Y., & Wang, T. (2021). Detecting anomalies in microservices with execution trace comparison. *Future Generation Computer Systems*, *116*, 291–301. https://doi.org/10.1016/j.future.2020.10.040

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Richardson, C. (2016). *From Design to Deployment*. NGINX, Inc.

Sun, Y., Zhao, L., Wang, Z., Cui, D., Yang, Y., & Gao, Z. (2021). Fault Root Rank Algorithm Based on Random Walk Mechanism in Fault Knowledge Graph. *2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 1–6. https://doi.org/10.1109/BMSB53066.2021.9547194

Tanouz, D., Subramanian, R. R., Eswar, D., Reddy, G. V. P., Kumar, A. R., & Praneeth, C. V. N. M. (2021). Credit Card Fraud Detection Using Machine Learning. *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 967–972. https://doi.org/10.1109/ICICCS51141.2021.9432308

Wang, T., Zhang, W., Xu, J., & Gu, Z. (2020). Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications With Statistics. *IEEE Transactions on Network and Service Management*, *17*(4), 2350–2363. https://doi.org/10.1109/TNSM.2020.3022028

Yashchenko, M. (2016). *Social network* [Java]. https://github.com/YashchenkoN/social-network

Zhao, Y., Hu, X., Cheng, C., Wang, C., Wan, C., Wang, W., Yang, J., Bai, H., Li, Z., Xiao, C., Wang, Y., Qiao, Z., Sun, J., & Akoglu, L. (2021). SUOD: Accelerating Large-Scale Unsupervised Heterogeneous Outlier Detection. *ArXiv:2003.05731 [Cs, Stat]*. http://arxiv.org/abs/2003.05731

Zhao, Y., Nasrullah, Z., Hryniewicki, M. K., & Li, Z. (2019). LSCP: Locally Selective Combination in Parallel Outlier Ensembles. *ArXiv:1812.01528 [Cs, Stat]*. https://doi.org/10.1137/1.9781611975673.66

Zhao, Y., Nasrullah, Z., & Li, Z. (2019). *PyOD: A Python Toolbox for Scalable Outlier Detection*. 7.