DOI: http://dx.doi.org/10.26483/ijarcs.v14i6.7036

Volume 14, No. 6, November - December 2023

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A COMPARATIVE ANALYSIS OF ANOMALY DETECTION FROM MICROSERVICE GENERATED UNSTRUCTURED LOGS

Anukampa Behera, Chhabi Rani Panigrahi
Department Computer Science
Rama Devi Women's University
Bhubaneswar, Odisha, India

Rohit Patel
Department of Computer Science and Information Technology, ITER, SOA Deemed to be University
Bhubaneswar, Odisha, India

Abstract: In a process, to ensure increased reliability and better availability, it is very important to detect any anomalies that refer to any abnormality observed in the behaviour of a standard process. The breakdown of service(s) eventually leads to production loss, and at the same time, a system that is unreliable brings lots of challenges to the operations team. Anomaly detection plays a significant role to ensure that an application is reliable, secured and available for user requests. For the overall performance optimization of a cloud microservice based application without any disruption in service, and identification of possible security threat, it is much essential that the anomalies must be detected and responded to in time. In real life large microservice based production infrastructures environments, even though ample instance of normal activities is available, it is not possible to predict and create a dataset of anomalies. So these kind of data are not suitable for a supervised two-class classification. In this work, unsupervised one-class approaches such as Local Outlier Factor, Isolation Forest, and One Class SVM are used to find anomalies. On experimentation these models have obtained a high accuracy of 98% to 99%. On comparing the performance of the models, One-Class SVM is found to produce significantly higher number of False Positives in comparison to other two considered models.

Keywords: Anomaly detection; One class algorithm; Local Outlier Factor; Isolation Forest; One Class SVM; Unstructured log analysis

I. INTRODUCTION

In the last decade, a paradigm shift from monolithic to microservice architecture has been observed in the IT industry. In a microservice architecture, the application is built up of several loosely coupled discrete units of functionality called services, each offering a particular functionality. Application programming interfaces (APIs) are used over the network to establish an interaction between the services. As the development, deployment, and scaling of services can be done independently, it offers much flexibility and contributes significantly to the implementation of continuous integration/continuous deployment (CI/CD). Despite its numerous benefits, microservice-based systems are quite complicated, and these complexities of this modern-day architecture have grown to such an extent that it is not feasible to detect any faults, like malfunction of a specific service or hardware or any possible threat through manual inspection [10]. Monitoring and detecting anomalies has become really challenging due to the distributive and dynamic nature of the micro-services.

Detection of any pattern that is unusual or any behaviour that is abnormal within a cloud-based microservices architecture is referred to as anomaly detection on cloud microservices. Within a micro-service ecosystem, anomalies are raised when a deviation from the regular pattern is observed, indicating either a performance problem, any kind of security breach, or any other potential issues. In a cloud-based microservice environment, data can be collected from several sources like network traffic, traces, metrics, system-generated logs, etc., to find out the occurrence of any atypical or uncommon activities or behaviour exhibited by the system. If the DevOps team and the system administrators can be notified immediately regarding anomalous system behaviour, it will help them to mitigate potential risks and threats by taking appropriate and immediate action [13].

Some of the types of anomalies those are commonly found in cloud microservices environments are discussed below to get an idea of the vast range of things that can go wrong. An exceptional drop or spike is noticed in disk, memory, CPU or network like resource consumption, degradation in handling service request in terms of high response time or latency are known as performance anomaly. Due to some misconfiguration or bugs in code, sometimes repetitive error patterns are logged, and 404, 500 like HTTP error codes are significantly increased, called error anomalies. Under security anomalies, mostly the indication of attacks or security breaches are lodged. For example, spike in network traffic, multiple failed login attempts, attempt to access sensitive data without authorisation, abnormal user behaviour with unauthorized API call, unexpected access attempts, etc. Indication of service interaction-related issues or introduction of a new dependency can lead to a change in the dependency graph; similarly, the number of requests between services can unusually change due to some missing dependency leading to dependency anomaly. Degradation in performance noticed due to changes made in the deployment pattern, configuration settings or sometimes due to any kind of misconfiguration in cloud services or microservices comes under configuration anomalies.

Scalability anomalies are raised when the service seizes to perform under high load or when system efficiency is affected due to under-provisioning or over-provisioning of resources. Impact on User experience is observed due to unavailability of service caused by critical component failure leading to cascading failures across the system. This type of situation raises outage anomaly leading to unprecedented and prolonged service outages. Lastly a degradation in performance can also happen due to resource leak anomalies where there is an incessant resource consumption even when the microservice is not active [13].

In the conventional model-based method for anomaly detection, one must have a clear understanding of all the

technicalities used in the entire process, which is very difficult to implement in case of the modern complicated technology and infrastructures. Data-driven techniques have emerged and gathered considerable attention recently to cope with the complexities of the current systems [6].

In this work, we have implemented three anomaly detection models based on unsupervised methods – local outlier factor, isolation forest and one class SVM respectively to detect anomalies from logs generated from microservices and made a comparative analysis of their performance. The remaining of the paper is organized as follows: in section II, related work in the field of anomaly detection using unstructured log analysis are discussed. A brief insight on the Local Outlier Factor, Isolation Forest and One Class SVM model's working principle is discussed in section III. In section IV, our proposed model is discussed followed by the experimentation, results and comparative analysis in section V. The conclusion is stated in section VI with future scope of work.

II. RELATED WORK

In this section, the works related to anomaly detection mostly using unsupervised approach is discussed.

Kun Lun Li [1] applied One Class SVM for the purpose of Intrusion Detection on the abstracted user audit logs, 1999 DARPA. As per their work, when clustering, KNN, Naïve Bias, SVM-light and One-Class SVM like algorithms were applied, One-Class SVM has shown the best result. Das *et al*., [9] proposed a distributed algorithm for detecting outliers in the data collected from various sites without moving them physically into a single location. This algorithm was first of its kind for anomaly detection for vertically partitioned data. They demonstrated the performance of their proposed methods experimenting on CMAPSS and NASA MODIS satellite image dataset respectively. They claim to identify 99% of the outliers by only using 1% communication towards data centralization in comparison to centralized method.

Jiang *et al.* [7] used One Class SVM for risk analysis and anomaly detection of equipment in Modern Supervisory Control and Data Acquisition (SCADA) systems by monitoring the performance of communication among them. Once trade-off parameters and slack variables were used to solve an optimal problem, most of the normal data were captured by One-Class SVM in a "small region" and a very small portion of data was flagged as anomalies. They created different classes to generate alarms at different levels by clustering these anomalies. Yin. S. *et al.* [6] proposed a modified version of One-Class SVM that they called robust 1-class SVM to suppress the effect of outliers. After introducing appropriate distance metrics and respective threshold, robust 1-class SVM was applied which the researchers have claimed to give satisfactory results. They stressed on the fact that if the training dataset has outliers, in that case robust 1-class SVM performs better for fault detection. Maglaras *et al.* [5] proposed an IT-OCSVM mechanism in a distributed SCADA network to provide accurate data regarding the time and origin of an intrusion as part of a distributed intrusion detection system. They embedded an aggregation procedure to decrease the overhead of IT-OCSVM so that it becomes suitable to be incorporated into a soft real-time system. They claim the proposed system detects all the induced attacks simulated while producing the minimum number of final alerts.

Xiao *et al.* [4] proposed vnuOCSVM to deal with the outliers available in the training dataset. They used the UCI benchmark dataset for experimentation. In comparison to other similar methods, the researchers claim vnuOCSVM to obtain higher g-mean and AUC values and give a better description of the target class, hence achieving a higher fault detection rate with lower false alarms. Khreich *et al.* [3] used one-class SVM with data extracted from system call traces combined with frequency. They obtained several n-grams of variable length by segmentation of the system call traces which were further mapped into sparse feature vectors of fixed size. To reduce the number of False Positives in class decomposed data while detecting anomalies, Haidar *et al*. [8] proposed an ensemble-based adaptive one-class and isolation forest framework with progressive artificial oversampling method.

To detect anomalies for microservices Cao *et al.* [14] proposed Conditional Random Field (CRF) based method where they collected several system parameters such as bandwidth occupancy, memory and CPU utilization etc., as the characteristic values for the sequence of observations. They generated the microservice fault matrix by labelling the abnormal types of the sequences occurred for their corresponding feature values. Nguyen *et al.* [2] used log data extracted from Juniper router devices and to detect anomaly, they applied the One-Class SVM model with different kernels. An unsupervised anomaly detection system was proposed by Farzad *et al.* [11], where two deep Autoencoder networks were used for feature extraction and Isolation Forest was implemented for prediction of positive data. The researchers have used Thunderbird, Openstack and BGL datasets for experimentation. Nobre *et al.,* [12] investigated the performance of Multi-Layer Perceptron (MLP) from detecting anomalies in a microservice environment both at the application and service level.

III. PREREQUISITES

This section describes the background of our problem statement and working principles of the models used.

A. Background

In this work, we have proposed methods to detect any deviation if occurred in a real life production scenario where all activities are periodical in nature called scheduled cron jobs. Here activities are tracked by their foot prints. For instance, "Source 'A' makes SSH on PORT 80 on example.com" is an example of an activity that occurs once at 10 AM every day. If for any day this activity happens thrice at different times, it can be treated as an anomaly. So, an anomaly can be raised of the observed frequency deviates from the set threshold. If any event has occurred for the first time, as there is no past record of the same, it will also be marked as an anomaly.

B. Local Outlier Factor(LOF)

Local Outlier Factor is a score computed by the LOF algorithm indicating the degree of abnormality amongst the recorded observations [15]. Here, samples with significantly lower density than its neighbours is detected as anomalies for which, deviation from local density of the data points are measured with reference to their neighbours. First average local density for the k-nearest neighbors is calculated which is compared with the instance's own local density to find out the LOF score. In case of a 'normal' instance, both the densities should match whereas for 'anomalies', a much smaller local density is observed for the test instance. The number of neighbours chosen is based on two factors: 1) it must be more than the threshold of objects a cluster must contain 2) an

assumption of maximum objects that can be considered as potential local outliers. In general practice the number of neighbours is by default fixed at twenty as it is difficult to get an exact value for the above mentioned two factors. This algorithm is best suitable for cases where different samples have varying underlying densities, as instead of calculating the degree of isolation of the sample itself, the isolation factor is calculated with respect to the nearest neighbours.

The following steps that can be adopted while applying an LOF model are [18]:

1) The distance between the observation point P and all neighbouring points are calculated using Euclidean or Manhattan like distance function.
2) Based on the value decided for k-nearest neighbour, the 'k' closest points are found.
3) Using the following equation (1) the local reachability density can be found.

$$lrd_k(o) = \frac{\|N_k(o)\|}{\sum_{o' \in N_k(o)} reachdist_k(o' \leftarrow o)} \quad (1)$$

where, for calculation of reachable distance the following equation (2) can be used.

$$reachdist_k(o' \leftarrow o) = \max\{dist_k(o),\ dist(o,\ o')\} \quad (2)$$

where, $N_K(O)$ denotes the number of neighbours

4) The LOF can be calculated as depicted in equation (3).

$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd_k(o')}{lrd_k(o)}}{\|N_k(o)\|} \quad (3)$$

A sample anomaly detection based on LOF is shown in Fig. 1.

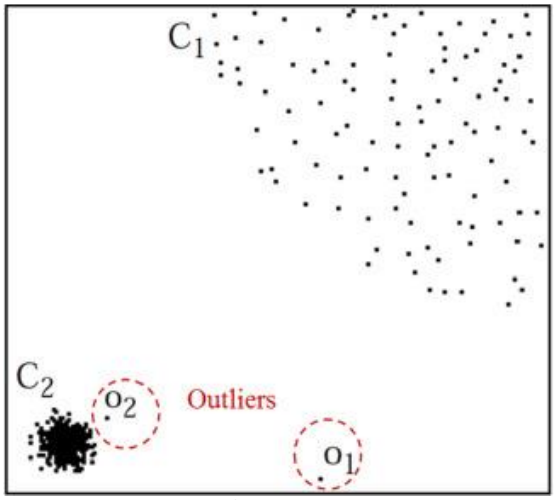

Figure 1. Detection of Local Outliers based on density

C. *Isolation Forest (IF)*

Random Forest algorithm is considered one of the efficient ways to perform anomaly detection on high-dimensional dataset. IF is a specific implementation [16] of Random forest that 'isolates' observations. IF selects a random split value within the minimum and maximum values of the randomly selected feature from the observation instance. As a tree-structure can be used to represent a recursive partitioning, the length of path from root to terminating node represents the number of splits needed for isolating a sample. The average path length of a forest constituting such random trees acts as the decision function and can be used as a measure of normality. In case of anomalies, the resultant paths after random partitioning, are considerably short. Thus, in a forest of random tree, if for particular samples shorter path lengths are produced, they can be considered as anomalies.

The following algorithm can be used to build an Isolation Forest [20].

Let a set of d-dimensional points be represented as $X = \{x_1, \dots, x_n\}$ and $X' \subset X$. A data structure with following properties can be defined as an iTree (Isolation Tree).

1) 'T' must be either an external leaf node or it can be any internal node with exact two children T_l , T_r and one "test".
2) The components of a "test" node are b (attribute) and a (split value) where $b < a$ determines path to either T_l or T_r .

For building an iTree, X' is recursively divided by a selected attribute b and a split value a until any of the following conditions occur:

1) The node is remained with only one instance.
2) Identical values occur for all data in the node.

Fig. 2 shows anomaly detection using IF [19].

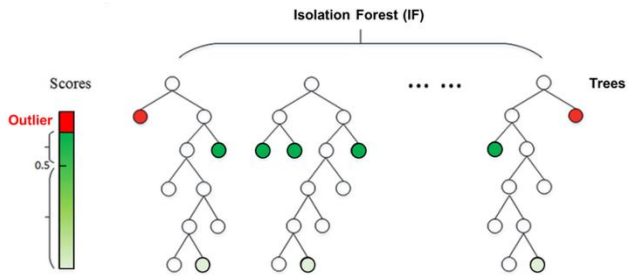

Figure 2. Anomaly detection using IF

D. *One Class SVM (OC-SVM)*

Support Vector Machines normally find outliers by tracing the presence of any additional data outside the different classes formed from the input data. In OC-SVM, all the input data belong to only a single class. Thus, for anomaly detection for the available data a decision boundary is decided based on Schölkopf's hyperplane method or Tax and Duin's hypersphere method. Any data that remains beyond the boundary is tagged as anomaly. The hypersphere method is also known as Support Vector Data Description (SVDD) [17]. For better capture of anomalies, SVDD forms a spherical boundary with a minimized volume in the feature space 'F' using the input data. (c, r) represent the hypersphere; 'c' denoting the center of the sphere and 'r' being the radius denoting the distance of support vector placed on the boundary and the center with a precondition as r>0. The volume minimization is done using the equation (4) using constraints stated in equation (5). All support vectors are combined linearly to form the center. Even though for any point 'a_i', its distance from the center must be less than 'r', ξ_i is introduced as a slack variable (depicted in equation (6)) with the penalty parameter as C for creating a soft margin.

$$\min_{r,c} r^2 + C\sum_{i=1}^{n} \xi_i \quad (4)$$

$$Subject\ to\ :$$

$$\|a_i - c\|^2 \le r^2 + \xi_i \quad \forall\ i = 1, \dots, n \quad (5)$$

$$\xi_i \ge 0\ ,\ \forall\ i = 1, \dots, n \quad (6)$$

Testing for the detection of outlier of a new data point,χ can be done after equation (7) is introduced with Lagrange multipliers α_i.

$$\|\chi - A\|^2 = \sum_{i=1}^{n} \alpha_i exp\left(\frac{-\|\chi - a_i\|^2}{\delta^2}\right) \geq -r^2/2 + C_r \qquad (7)$$

Here, $\delta \in R$, is a kernel parameter.

Fig, 3. depicts the separation of data points using OC_SVM method [21].

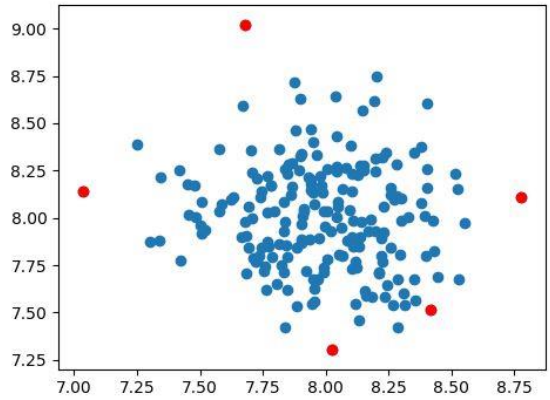

Figure 3. Detection of anomalies using OC-SVM

IV. Proposed Methodology

The steps followed to create the dataset from real life loglines and the proposed models are discussed in this section.

A. *Dataset Preparation*

For dataset preparation, we have used the steps proposed by Behera *et al.* [22]. A summary of the steps is given below.

Step 1: Initial data collection is done from a HIDS server to gather structured and tagged log data and an N-dimensional dataset is created. The logs generated are basically for any deflection that is found on the rule id defined by HIDS server.

Step 2: Without compromising with the data quality, the created dataset is reduced to a 4-dimensional feature-set, source host, time slot, frequency and unique profile identifier using the following method.

The first feature extracted is the source host (the source from which the event has generated). For handling the high volume data, a temporal division is made where the 24 hours logs are divided into smaller time buckets. For a specific user, the complete instance of the dataset is extracted for a single time bucket that acts as the second feature. Next a feature selection method is applied to get the most contributing features. A common format is created by taking the union of all unique components across rule ids. A set of all instances under a single rule ID is merged, and a new feature is added as frequency - the third feature. The footprint of a user starting from the log-in till the time he logs out is stored and represented as a profile identifier serving as the fourth feature.

B. *Models Used*

In an archetypal production infrastructure, plenty of instance can be found where all activities are done as per scheduled cron activities labelled under normal activities. But in order to train a system, instances of deviation from regular observed pattern must be made available in the dataset as abnormal activities; which is not only impossible to collect in many cases, it is very expensive as well. A simulated environment for faulty systems is not advisable as gathering all indications and causes leading to system's anomalous behaviour is not possible to forecast and anticipate; so it does not guarantee about the completeness of the dataset. Thus in this type of a situation, one class unsupervised approaches are highly suitable. In this regard, we propose to use one class algorithms - Local Outlier Factor, Isolation Forest, and One-Class SVM for the detection of anomalies in a real-life microservice-based production environment.

V. Experimentations, Results and Comparative Analysis

A. *Dataset Acquisition*

The experimentation was performed in a production environment where several microservices were deployed. Initially, the dataset was collected from the logs produced by the Open Source HIDS Security server (OSSEC). Next, the 24 Hours of data were grouped into buckets of 15-minute duration. This 15 minutes will be fixed and serve as our bucket size. All the data collected are cron jobs i.e., periodical in nature. A test dataset that is a subset of the original dataset is prepared to test the different methods. The subset contains the data of a particular `bucket` and a particular `weekday` for the past 30 days. For evaluating purposes, each instance is assigned a value of '1' as the label to identify them as non-anomaly jobs. Any job which is scheduled to be in different buckets, if falls into another bucket, is marked as `Anomaly`. We set the threshold value as three to accommodate weekly occurrence of events in the classification process. Hence, we selected a fraction of data belonging to different bucket and weekday and introduced them into our test dataset. These instances as given a value of '-1' as the label.

The required four features ['agent_id,' 'profile_id,' 'bucket_id,' 'frequency'] were derived and extracted from the test dataset, which will be used for the model training.

B. *Parameter Tuning*

For implementation and best results, the hyper-parameters were tuned as follows.

The 'Contamination' Value for Isolation Forest and Local Outlier Factor and 'nu' for OCSVM was set to be 25/7000. For LOF, the value for 'n_neighbour' is set within the range of 3 to 15 with a step value of 1. For IF, the value for 'n_estimators' was passed within the range of 100 to 1000 with a step value of 50. For OCSVM, before fitting the data frame to the model, the gamma value varied within the 0.05 to 1 range, and the step value was set at 0.01. For the rest of the parameters, a Grid search was used to test all the combinations for all three models.

C. *Metrics Used*

The metrics used in this work for evaluation of the results are as follows.

$$Class\ wise\ Precision = \frac{TP}{TP + FP}$$

$$Class\ wise\ Recall = \frac{TP}{TP + FN}$$

where,
$TP : True\ Positive$
$FP : False\ Positive$
$TN : True\ Negative$
$FN : False\ Negative$

$$Class\ wise\ F1-Score\ =\ \frac{2*Precision*Recall}{Precision+Recall}$$

$$Weighted\ Average\ Precision\ =\ \frac{\sum P_i*S_i}{\sum S_i}$$

where Pi is the Precision of class i
and S_i is the support of class i
i.e, the total number of samples

$$Weighted\ Average\ Recall\ =\ \frac{\sum R_i*S_i}{\sum S_i}$$

where Ri is the Recall of class i
and S_i is the support of class i
i.e, the total number of samples

$$Weighted\ Average\ F1\ =\ \frac{\sum F1_i*S_i}{\sum S_i}$$

where $F1i$ is the $F1$ score of class i
and S_i is the support of class i
i.e, the total number of samples

$$Macro\ Average\ Precision\ =\ \frac{\sum P_i}{n}$$

where Pi is the Precision of class i
and n is the total number of classes

$$Macro\ Average\ Recall\ =\ \frac{\sum R_i}{n}$$

where Ri is the Precision of class i
and n is the total number of classes

$$Macro\ Average\ F1\ Score\ =\ \frac{\sum F1_i}{n}$$

where $F1i$ is the $F1$ Score of class i
and n is the total number of classes

D. *Results Obtained*

The obtained results after passing the dataset to each three models are depicted in Table I, and the obtained classification reports for all three models along with their calculated accuracy is shown in Table II respectively.

A comparison of results obtained by all the three models, LOF, IF and OC-SVM are shown in Fig. 4.

Table I. Results Obtained from the LOF, IF and OC-SVM Models

		LOF		***IF***		***OC-SVM***	
		Predicted		**Predicted**		**Predicted**	
		-1	**1**	**-1**	**1**	**-1**	**1**
Actual	**-1**	13	7	18	3	14	6
	1	5	7041	10	7036	104	6942

Table II. Model Wise Classification Report

	LOF			***IF***			***OC-SVM***			
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Support**
-1	0.70	0.65	0.68	0.64	0.90	0.75	0.12	0.70	0.20	20
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	7046
Macro Average	0.85	0.82	0.84	0.82	0.95	0.87	0.56	0.84	0.60	7066
Weighted Average	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	7066
Accuracy	0.99			0.99			0.98			7066

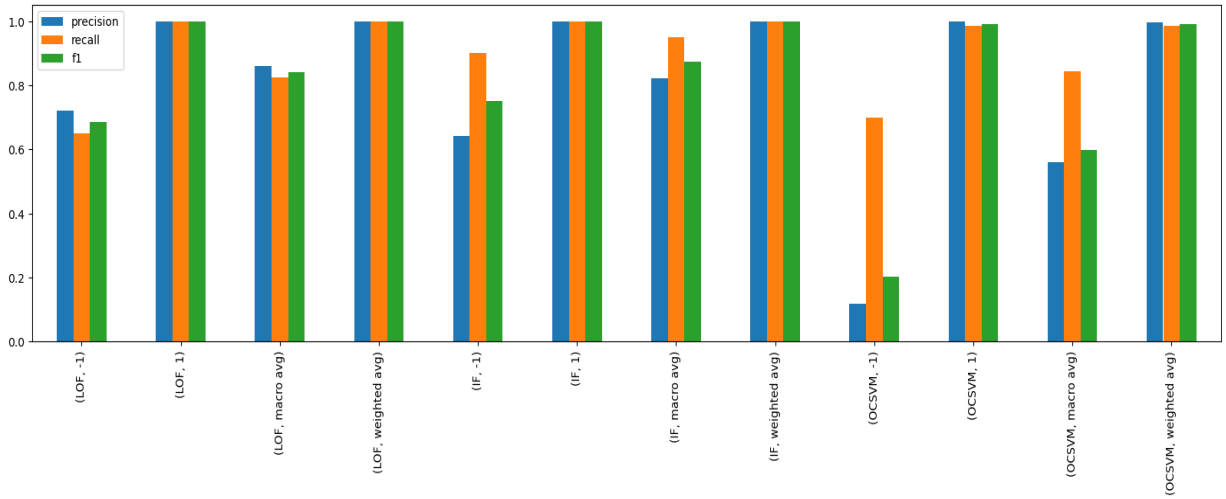

Figure 4: A comparative results obtained from LOF, IF and OC-SVM

The obtained results clearly state that even though all the three models are obtaining a high accuracy, there is a significant difference between the precessions obtained because of high rate of 'False Positives' getting generated by the OC-SVM model. The LOF and IF models the number of 'False Positives' generate are quite less. At the same time, when a single instance is anomalous, OC-SVM is able to catch it due to its lower tolerance towards deviations.

VI. Conclusion and Future Work

Due to the multi-variant and multi-dimensional data obtained from microservice based production infrastructures, it is yet to devise any standard method for anomaly detection. As in the real life scenarios obtaining sample instance for anomalies is not only costly but it may not be fitting all cases of fault observations, in this work, we have used unsupervised one class methods such as LOF, IF, and OC_SVM for detecting anomalies. With experimentations, the accuracies obtained by the models are 99%, 99% and 98% respectively. The OC-SVM model is found to produce a notable amount of 'False positives' whereas it is negligible in case of other two model LOF and IF. But due to its higher intolerance towards anomalies, the OC-SVM is best suitable to catch an anomaly in its first instance itself in case there is a possibility of only a single instance of deviation leading to larger problems in course of time such as planting of malwares in server. In future, the models can be tested with customized datasets obtained from different types of servers for a generic conclusion.

VII. References

[1] Kun-Lun Li, Hou-Kuan Huang, Sheng-Feng Tian and Wei Xu, "Improving one-class SVM for anomaly detection," Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693), Xi'an, 2003, pp. 3077-3081 Vol.5, doi: 10.1109/ICMLC.2003.1260106.

[2] Nguyen, TBT., Liao, TL., Vu, TA. (2019). "Anomaly Detection Using One-Class SVM for Logs of Juniper Router Devices." In: Duong, T., Vo, NS., Nguyen, L., Vien, QT., Nguyen, VD. (eds) Industrial Networks and Intelligent Systems. INISCOM 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 293. Springer, Cham. https://doi.org/10.1007/978-3-030-30149-1_24.

[3] Wael Khreich, Babak Khosravifar, Abdelwahab Hamou-Lhadj, Chamseddine Talhi, "An anomaly detection system based on variable N-gram features and one-class SVM", Information and Software Technology, Volume 91, 2017, Pages 186-197, ISSN 0950-5849, doi: 10.1016/j.infsof.2017.07.009..

[4] Yingchao Xiao, Huangang Wang, Wenli Xu, Junwu Zhou,"Robust one-class SVM for fault detection", Chemometrics and Intelligent Laboratory Systems, Volume 151, 2016, Pages 15-25, ISSN 0169-7439, doi: 10.1016/j.chemolab.2015.11.010.

[5] Leandros A. Maglaras, Jianmin Jiang, Tiago J. Cruz,"Combining ensemble methods and social network metrics for improving accuracy of OCSVM on intrusion detection in SCADA systems", Journal of Information Security and Applications,Volume 30, 2016, Pages 15-26, doi: 10.1016/j.jisa.2016.04.002.

[6] Shen Yin, Xiangping Zhu, Chen Jing, "Fault detection based on a robust one class support vector machine", Neurocomputing, Volume 145, 2014, Pages 263-268, doi: 10.1016/j.neucom.2014.05.035.

[7] J. Jiang and L. Yasakethu, "Anomaly Detection via One Class SVM for Protection of SCADA Systems," 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Beijing, China, 2013, pp. 82-88, doi: 10.1109/CyberC.2013.22.

[8] D. Haidar and M. M. Gaber, "Adaptive One-Class Ensemble-based Anomaly Detection: An Application to Insider Threats," 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, pp. 1-9, doi: 10.1109/IJCNN.2018.8489107.

[9] Das, K., Bhaduri, K., & Votava, P. "Distributed anomaly detection using 1-class SVM for vertically partitioned data". Statistical Analysis and Data Mining, 2011, 4(4), 393–406. doi:10.1002/sam.10125

[10] Bursic, S., Cuculo, V., D'Amelio, A. (2020). Anomaly Detection from Log Files Using Unsupervised Deep Learning. In: Sekerinski, E., et al. Formal Methods. FM 2019 International Workshops. FM 2019. Lecture Notes in Computer Science(), vol 12232. Springer, Cham. doi: 10.1007/978-3-030-54994-7_15

[11] Farzad A., T. Aaron Gulliver,"Unsupervised log message anomaly detection", ICT Express,Volume 6, Issue 3,2020,Pages 229-237, doi:10.1016/j.icte.2020.06.003.

[12] Nobre, João, E. J. Solteiro Pires, and Arsénio Reis. 2023. "Anomaly Detection in Microservice-Based Systems" Applied Sciences 13, no. 13: 7891, doi: 10.3390/app13137891

[13] [Online] Available at: https://www.linkedin.com/pulse/1-microservices-anomaly-detection-pushkar-pushp/

[14] Cao, W., Cao, Z., & Zhang, X. Research on Microservice Anomaly Detection Technology Based on Conditional Random Field. Journal of Physics: Conference Series, 1213., 2019

[15] [Online] Available at: https://scikit-learn.org/stable/modules/outlier_detection.html#local-outlier-factor

[16] [Online] Available at:https://scikit-learn.org/stable/modules/outlier_detection.html#isolation-forest

[17] [Online] Available at:https://www.baeldung.com/cs/one-class-svm

[18] [Online] Available at: https://www.sciencedirect.com/topics/computer-science/local-outlier-factor

[19] [Online] Available at: https://wiki.datrics.ai/isolation-forest-model

[20] [Online] Available at: https://en.wikipedia.org/wiki/Isolation_forest

[21] [Online] Available at: https://www.datatechnotes.com/2020/04/anomaly-detection-with-one-class-svm.html

[22] Behera A, Behera S, Panigrahi C R & Weng Tien-Hsiung, Using unstructured logs generated in complex large scale micro-service-based architecture for data analysis, Int. J. of Business Intelligence and Data Mining, Vol 1(1) , 248-263. 2022 doi: 10.1504/ijbidm.2022.10043252