Anomaly Detection

Soumyo Dey

Khoury College of Computer Science, Northeastern University, Boston, USA
dey.soum@northeastern.edu

Abstract

The demand for cloud computing is steadily increasing. Many companies, that create and work on virtual machines in the cloud are bound to encounter anomalies. To determine whether cloud services are functioning properly, someone must monitor them, detect anomalies, and address them as early detection of anomalies is critical. In order to leave enough room for maneuvering safeguards, as well as ensuring product quality. A critical technique for managing such complex cloud resources is automatic anomaly detection. In this paper, we evaluate the performance of machine learning algorithms for anomaly detection in EC2 CPU utilization data. We use four algorithms in total: One Class SVM, Isolation Forest, Local Outlier Factor, and GMM, and they are evaluated using EM values.

Introduction

Cloud computing is becoming more common, but it is also becoming more sophisticated. Due to heterogeneity, dynamicity, scalability, hidden complexity, and temporal constraints, large systems are extremely complicated. To operate efficiently and cost-effectively, data centers use hundreds of virtual computers that require dynamic resource scheduling. These data centers need accommodate to increasing demand for resources such as CPU and memory. A scheduling algorithm must dynamically allocate or re-allocate these resources. It is vital to keep an eye on the server metrics (for example, latency, CPU, RAM, and disk I/O), which are represented by a time series, for any odd behavior. Early detection of anomalous performance metric behavior is crucial for taking preventive action to secure consumers and enhance the user experience. To monitor and provide reliability to such a huge system, automation is required. These demands monitoring in order to get insight into the operation of the cloud's hardware, systems, and applications. The key to analyzing system behavior and identifying anomalous behavior is to monitor the system. Anomalies are defined as anomalous behavior caused by performance concerns, hardware or software malfunctions, or configuration issues.

Anomaly detection has been studied in a variety of application domains, including credit card fraud detection, intrusion detection in cybersecurity, and fault diagnosis in industry. Outliers in time series data, in particular, are examined for anomalous patterns over time. The primary goal of this study is to investigate the techniques used for anomaly detection on performance data to determine when a virtual server is experiencing problems. The goal is to compare and contrast various approaches and methodologies for detecting anomalies. In this project, we focused on unsupervised methods for detecting anomalous behavior. A variety of unsupervised algorithms have been used for anomaly detection and have been classified based on their functionality. Clustering-based, angle-based, statistical, neighbor-based, density-based, and classification-based are some of the categories. In this report, we discussed density-based algorithms (Local Outlier Factor and Gaussian Mixture Model) and classification-based algorithms (One Class SVM and Isolation Forest). To assess the performance of these algorithms, we used the EM value, a metric developed by Nicolas Goix [12].

Related Work

Remarkable amount of work has been done on the topic of unsupervised anomaly detection over the years. In this section, some of the related work has been discussed. For anomaly detection in compute cloud systems, the authors of [9] suggested an autonomous approach. In the process of building the feature space, the author uses dimensionality reduction.

When no labels are provided, OCSVM is sensitive to outliers. To address this shortcoming, Amer et al. (2013) [2] improved OCSVM for unsupervised anomaly detection by proposing two modifications that reduce the contribution of outliers to the decision boundary when compared to normal instances. In [3] the sensor data was analyzed, and it was discovered that pattern anomaly plays a dominant role for a single sensor. Furthermore, a sudden change in data is discovered to be a type of pattern anomaly. They converted the

problem to pattern anomaly detection of time series because the data collected by a single sensor was ordered by time.

Isolation forest (IForest) is the first method proposed in the isolation-based category [4, 5]. One of IForest's limitations is that it was built for static datasets rather than data streams. The authors of [6] propose an improvement to IForest that uses sliding windows to adapt to data stream context. Isolation Forest Algorithm for Stream Data is the name given to the proposed method (IForestASD). Other Isolation Forest improvements and adaptations exist, such as Extended Isolation Forest [7] and Functional Isolation Forest [8], but they are designed for batch settings and not data streaming.

Mishra *et al*. [10] use it to detect anomalies in data streams. The research focuses on density-based outliers between data and considers performance based on certain parameters. To identify unusual behavior in ships, Huang [11] conducted research using LOF and K Nearest Neighbor (K-NN). This study filters existing vessel datasets using KNN, calculates the local deviation index using the LOF algorithm, and compares the index findings to a predetermined threshold to decide whether the data is normal or not.

Project Description

Data

The Numenta Anomaly Benchmark (NAB) is a novel benchmark for evaluating algorithms for anomaly detection in streaming, online applications. It is comprised of over 50 real-world and artificial timeseries data files plus a novel scoring mechanism designed for real-time applications. Out of these 50 we are using EC2 CPU utilization data. It has two columns – 'timestamp' and 'value'. 'timestamp' has timestamp generated by AWS cloud watch while recording the CPU utilization. The CPU utilization value is recorded after every 5 seconds. 'value' contains the CPU utilization value at that instant.

Dataflow

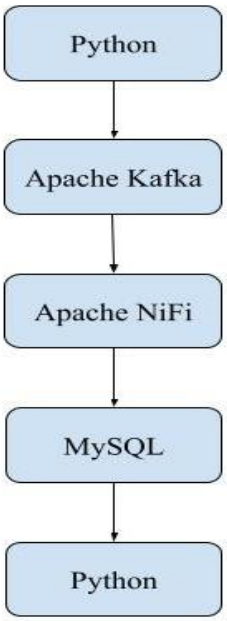
Python

As our current data is static, we are creating a continuous streaming data using a Python Jupyter notebook to Apache Kafka. The need for making the data continuous is to create a data pipeline which would be able to handle data that is collected by the CPU utilization monitoring system.

We also used Python to create a connection between MySQL Workbench and Python Jupyter notebook to access the data. Then the data was queried from the database for anomaly detection.

Apache Kafka

Kafka is a streaming platform capable of handling trillions of events a day. At its core, it is distributed, horizontally-scalable (because of built-in partitioning), fault-tolerant (because of replications), low latency (partially because reads and writes are done at the constant time due to ordered immutable sequence data-structure), commit log (records can be created, but not updated). Kafka allows us to decouple data streams and systems: The Source System pushes data to Kafka, and then the Target System sources the data from Kafka. In this way, Kafka is used as data transportation mechanism. In our case, the source system is currently a python notebook and the target system is Apache NiFi.



Apache NiFi

Apache NiFi is an ETL tool with flow-based programming that comes with a web UI built to provide an easy way (drag & drop) to handle data flow in real-time. It also supports powerful and scalable means of data routing and transformation, which can be run on a single server or in a clustered mode across many servers.

Processors, the rectangular parts of the NiFi workflow, can process, validate, filter, connect, split, or change data. The FlowFile Controller assists in managing the resources between those components as they communicate data in the form of FlowFiles over connected queues.

MySQL

MySQL is an open-source SQL relational database management system. It stores the information in the form of related tables. We have used MySQL as database from where it is easy to access data using Python.

Algorithms

This study includes the following ML-based anomaly detection techniques:

One-Class Support Vector Machines (OCSVM)

By projecting time-series data vectors onto phase spaces, OCSVM detects anomalies in time-series data. A prior knowledge of the underlying data distribution is not required for such classification-based techniques. This machine learning-based approach's basic idea is to learn a decision boundary that achieves the greatest separation between the points and the origin. When no labels are provided, OCSVM is sensitive to outliers.

Isolation Forest [4]

This ML approach developed by Liu et al. [4] to detect anomalies in timeseries using a sliding window is based solely on the concept of binary trees isolating data points and does not use any distance or density measure. In contrast to the widely used distance and density measures, this anomaly detection method is based on the concept of 'isolation.' Anomalies are 'isolated' from normal instances in this approach. The data instances that are few in number and have attribute-values that differ greatly from the rest of the data instances are more vulnerable to isolation. To isolate such instances, this method employs a binary tree structure known as the isolation tree (iTree).

Local Outlier Factor (LOF)

The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method that computes a given data point's local density deviation with respect to its neighbors. It considers samples with a significantly lower density than their neighbors to be outliers. The LOF is another method for detecting anomalies based on distance. It is used to detect local anomalies using local densities. The k-nearest-neighbors of each data point in a given streaming data set must be found using this method. The local density of each data point is estimated using k-nearest-neighbors by computing the local reachability density (LRD). Finally, the LOF of a point indicates its density in relation to the density of its neighbors. If a point's density is much lower than the densities of its neighbors, the point is far from dense areas and thus an outlier.

Gaussian Mixture Model

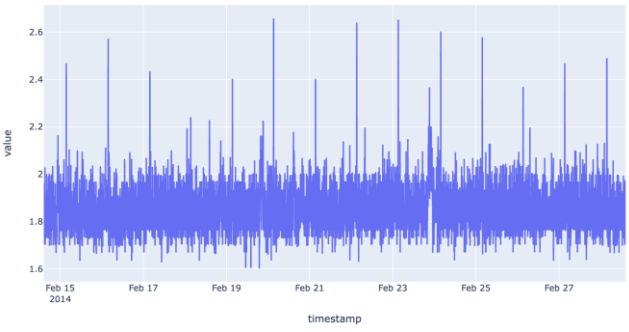
The Gaussian Mixture Model (GMM) is a probabilistic clustering model that assumes each data point belongs to a Gaussian distribution. Anomaly detection is the technique of recognizing anomalous data points. The Gaussian Mixture Model (GMM) detects outliers by finding data points in low-density regions [1]. We compute the probability of belonging to each cluster for each data point. From packets, this approach learns statistical regularities. GMM is a model for probabilistic learning. In this method, each of the input sets is modeled independently of the others, and the best feasible probability distribution for each group is attempted using a series of Gaussian probability distribution functions known as Gaussian Mixtures. The number of mixtures in each model, as well as their means and variances, all have varying implications on performance.

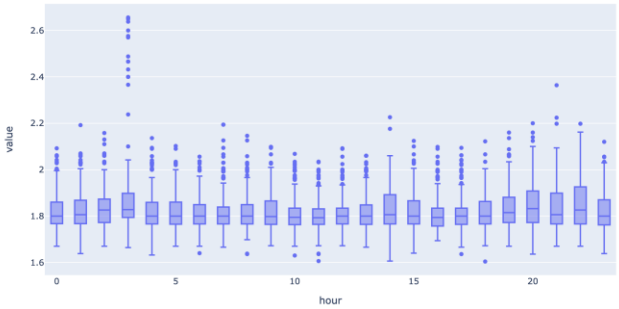
Empirical results

1. Exploratory Data Analysis (EDA)

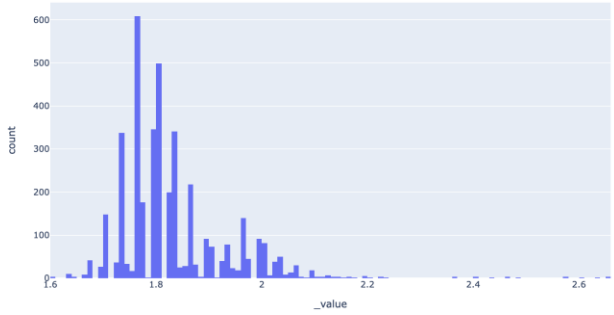
Understanding the data is one of the most crucial phases before implementing the algorithms. EDA is used so that we can make some deductions by visualizing our data if there are to make any.

We start of by simply plotting the data with ‘timestamps’ on x-axis and ‘value’ of y-axis. We can see that there are outliers present in the data but we can’t really pin point the timestamps.





Next up we make boxplot to get a better view of the outliers. Some obvious outliers appear. Finally, we make a histogram of count of values.



We can see that the data are mostly distributed within 1.616-2.016. Out of 4032 values 3821 values lie in the range. Thus 5% of our data is anomaly.

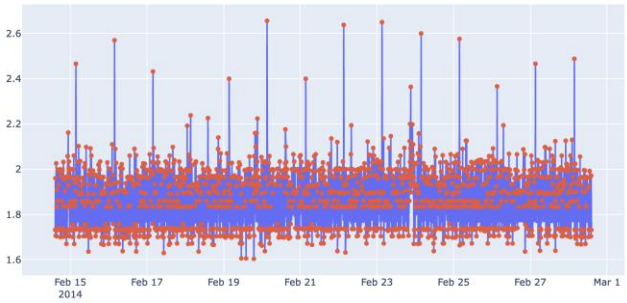
Implementation of Algorithms

One-Class Support Vector Machines (OCSVM)

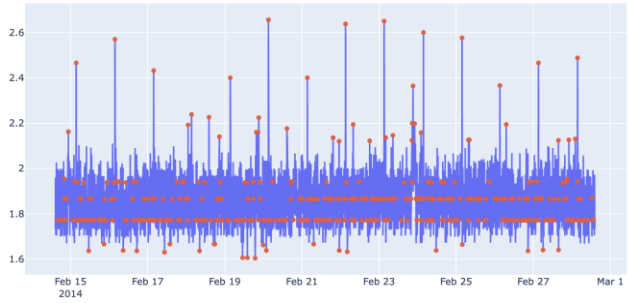
First to get the idea of how the algorithm is performing on the dataset we applied using the default parameters.

Parameters:

1. kernel = 'rbf'
2. degree = 3
3. gamma = 'scale'
4. coef0 = 0.0.
5. tol = 1e-3
6. nu = 0.5
7. shrinking = True
8. cache_size = 200
9. verbose = False
10. max_iter = -1

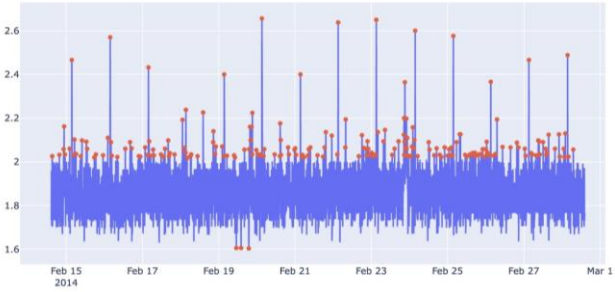


It can be seen that there are many outliers, which are not what we expected, so we adjust the nu value and modify the parameters to fit the results.



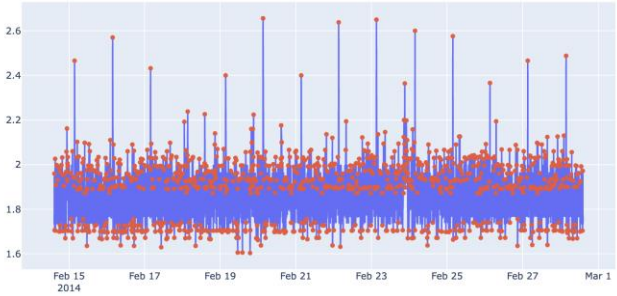
After reducing the nu value, we can clearly see that we are getting close to the desired outliers. But to get the exact value of nu we use IQR which is a statistical method to calculate outliers. Once we get the best value of nu, we simply use that for one class SVM.

Finally, we get the best output for one class SVM.



Isolation Forest

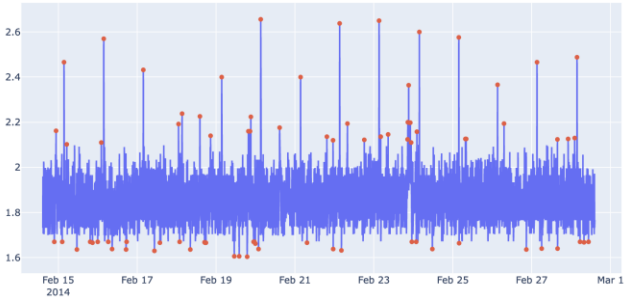
First to get the idea of how the algorithm is performing on the dataset we applied using the default parameters.



Parameters:

1. n_estimators = 100
2. max_samples = ‘auto’
3. contamination = 0.1
4. max_feature = ‘1.0’
5. bootstrap = False
6. n_jobs = None
7. random_state = None
8. verbose = 0
9. warm_start = False

We can see that some excessive outliers appear. To correct this and get a better result we reduce the contamination factor.



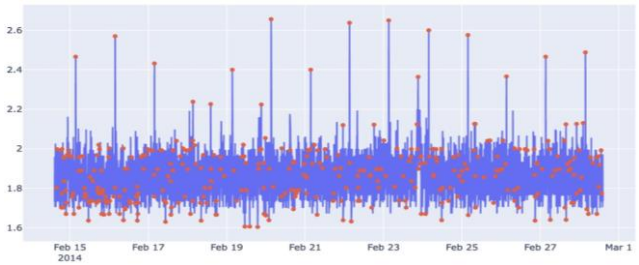
We can see we get a way better output with proper outliers identified.

Local Outlier Factor (LOF)

For LOF we simply use the default parameters.

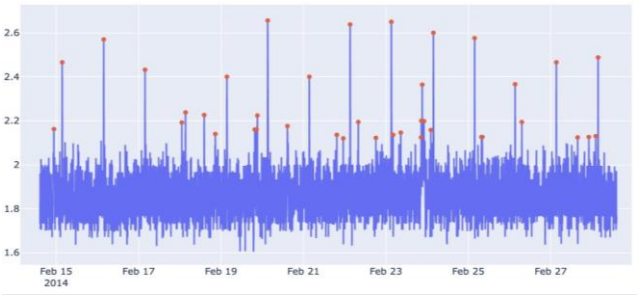
Parameters:

1. n_neighbors = 20
2. Algorithm = ‘auto’
3. leaf_size = 30
4. Metric = ‘minkowski’
5. p = 2
6. metric_params = None
7. Contamination = ‘auto’
8. Novelty = False
9. n_jobs = None



Gaussian Mixture Model (GMM)

For GMM also we simply use the default parameters.



Parameters:

1. n_components = 1
2. covariance_type = ‘diag’
3. random_state = None
4. min_covar = 1e-3
5. tol = 1e-3
6. n_iter = optional
7. n_init = optional
8. params = ‘wmc’
9. init_params = ‘wmc’

Algorithm Evaluation

Once we have applied all the algorithms, it was time to know which algorithms worked the best.

Algorithm	EM - Value
One Class SVM	0.011641
Isolation Forest	0.101893
Local Outlier Factor (LOF)	0.072206
Gaussian Mixture Model (GMM)	0.020273

In our situation as there are no labelled data available with us, we would be using EM values as the metric to evaluate our algorithms. As stated by Nicolas Goix [12] in his paper, 80 percent of the time EM values are able to tell which algorithm is better than the other specially on low dimension dataset and also explained how the algorithm works. So, it fit perfectly for our case and we used the algorithm as stated in the paper.

Conclusion/ Future directions

Higher the EM value, indicates the better performance of the algorithm. In our case Isolation Forest worked the best followed by GMM, One Class SVM is the worst performing algorithm.

In future we can automate this whole process as we now know the best algorithm that works best on this type of data. The data would be live streaming data that we can directly access through the database. This model won't just be used for CPU utilization but for all kinds of AWS metrics.

References

1. M. Junshui and S. Perkins, "Time-series novelty detection using one-class support vector machines," in Proc. of the International Joint Conference on Neural Networks, 2003, August 26, Portland, OR, USA.
2. M. Amer, M. Goldstein and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection", ACM SIGKDD Workshop on Outlier Detection and Description, 2013.
3. Su, J., Long, Y., Qiu, X., Li, S., Liu, D. (2015). Anomaly Detection of Single Sensors Using OCSVM_KNN. In: Wang, Y., Xiong, H., Argamon, S., Li, X., Li, J. (eds) Big Data Computing and Communications. BigCom 2015. Lecture Notes in Computer Science, vol 9196. Springer, Cham. https://doi.org/10.1007/978-3-319-22047-5_18
4. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation Forest. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422. IEEE (2008)
5. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. ACM Trans. Knowl. Discov. Data (TKDD) 6(1), 3 (2012)
6. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proc. Vol. 46(20), 12–17 (2013). https://doi.org/10.3182/20130902-3-CN-3020.00044
7. Hariri, S., Kind, M.C., Brunner, R.J.: Extended isolation forest. arXiv preprint arXiv:1811.02141 (2018)
8. Staerman, G., Mozharovskyi, P., Clémençon, S., d'Alché Buc, F.: Functional isolation forest (2019)
9. Smith D. , Qiang Guan, Song Fu, An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems, 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, 2010, pp: 376-381.
10. S. Mishra and M. Chawla, "A Comparative Study of Local Outlier Factor Algorithms for Outliers Detection in Data Streams" in Emerging Technologies in Data Mining and Information Security. Advances in Intelligent Systems and Computing, 2019
11. Yingfu Huang and Qiurong Zhang, "Identification of anomaly behavior of ships based on KNN and LOF combination algorithm", *AIP Conference Proceedings*, vol. 2073, no. 1, 2019.
12. Nicolas Goix, "How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms?" (2016), https://arxiv.org/abs/1607.01152