International Journal of Sciences

Research Article Volume 13 – July 2024 (07)

A Weakly Supervised Learning Approach to Anomaly Detection on Cloud Server Configuration

Qiuyu Tian[1], Hongwei Tang[1], Xiaohong Wang[1]

[1]Institute of Computing Technology Chinese Academy of Sciences, Beijing, China

Abstract: Cloud computing platforms have become increasingly popular across various industries, offering publicly accessible computing, storage, and network solutions to meet the demands of building, scaling, and managing applications. A critical component of these platforms is the recommendation system, which significantly influences customer experience and platform revenue. However, variations in customer behavior and product attributes result in different recommendation scenarios across platforms. One key scenario faced by customers of cloud computing platforms is configuration selection. In this paper, we present an innovative approach to detect potentially misconfigured cloud servers in such scenarios. Our method utilizes weakly supervised learning, using server lifetime as a weak signal to guide the configuration anomaly detection model. By implementing this configuration check, we can prevent customers from purchasing misconfigured products, thus promoting a stable and satisfactory relationship between cloud computing platforms and their customers.

Keywords: Weakly Supervised Learning, Anomaly Detection, Cloud Computing

1 Introduction

Cloud computing platforms encompass a variety of services, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), each providing complementary functionalities. Similar to online shopping platforms, cloud computing platforms have three key contributors: customers, products, and providers. On large platforms like AWS, the provider is the same for all cloud services. Online shopping platforms typically rely on multi-objective recall strategies, along with embedding learning and knowledge graph techniques, to generate customized and intelligent recommendations. However, the scenarios on cloud computing platforms differ significantly.

Users of cloud computing services often stick to a single type of cloud product and show little interest in products from other categories. The main products on cloud platforms are servers and storage solutions, which generally require users to select configurations during the purchasing process. When a customer attempts to purchase (or rent) a cloud server, they must configure various parameters, such as the number of CPU units, memory size, disk storage size, operating system, and GPU type. However, customers often possess varying levels of technical knowledge, and sometimes they fail to select appropriate configurations. For instance, if a student intends to rent a GPU cloud server to train a machine learning model and selects 8 CPU units, 32GB of memory, a Tesla T4 GPU, but only 30GB of disk storage, numerous errors due to insufficient disk space are likely to occur. Once a customer purchases a misconfigured cloud computing product, they may need to shut down the current server instance and start a new one soon after, leading to unsatisfactory outcomes such as additional charges and wasted effort in redoing work saved on the old server.

In this paper, we highlight the limitations of existing methodologies for anomaly detection, specifically statistical and unsupervised learning approaches, when applied to the scenario of configuration selection. Our data is not time-sensitive, thus time series approaches are not considered. Statistical approaches such as Z-score and Inter-Quartile Range (IQR) are limited to single continuous features, with Z-score further assuming a normal distribution of the feature. Existing unsupervised learning methods, including DBSCAN [1], LOF [2], and One-class SVM [3], detect anomalies based on distance and density measures, which are not applicable to categorical features. Additionally, these methods suffer from the curse of dimensionality, leading to computational inefficiency. These limitations are discussed in detail in Section 2.

We propose a weakly supervised approach to anomaly detection, using server lifetime as a weak signal. Server lifetime is defined as the duration between server creation and shutdown times. To achieve this, we first employ a CatBoost [4] regression model to predict server lifetime based on basic server configurations and user group information, with differences among user groups outlined in Section 4. We then apply the Isolation

Published at: http://www.ijsciences.com/pub/issue/2024-07/
DOI: 10.18483/ijSci.2779; Online ISSN: 2305-3925; Print ISSN: 2410-4477

Qiuyu Tian (Correspondence)
+

Forest [5] unsupervised anomaly detection model, incorporating predicted server lifetime as one of its descriptive features, to identify misconfigured servers. The Isolation Forest algorithm relies on predicted anomaly scores to detect anomalies, and we introduce an additional constraint on predicted server lifetime to filter anomalies and mitigate minority bias. Our approach offers several advantages: (i) it incorporates both continuous and categorical features, considering differences among user groups, (ii) it leverages server lifetime as a weak signal to enhance model performance, and (iii) it is scalable and computationally efficient. Detailed methodology and results are presented in Section 3.

2 Related Work

2.1 Statistical Approach for Anomaly Detection

The Z-score measures how far a data point is away from the mean as a signed multiple of the standard deviation; large absolute values of the Z-score suggest an anomaly. Let x be a descriptive feature, assume x follows Normal Distribution, $\bar{x}$ represents its mean and σ_x represents its standard deviation, then we have Z-score defined as:

$$z_i = \frac{x_i - \bar{x}}{\sigma_x} \qquad (1)$$

A Z-score of 3 is commonly used as the threshold above which it indicates outliers, since according to Normal Distribution, 99.7% of the data points lie between +/- 3 standard deviations. As we can see, using Z-score is a very naïve approach, since it assumes normal distribution, which is not always true.

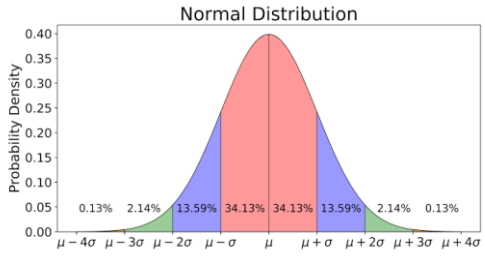


Figure 1: Probability Density Function of Normal Distribution Marked with Standard Deviations.

The Inter-Quartile Range is a measure of statistical dispersion[1]. It is defined as the difference between the 75th and 25th percentiles of the data. To calculate IQR, the data is divided into quartiles, denoted by Q_1 (the lower quartile, 25th percentile), Q_2 (the median, 50th percentile), Q_3 (the upper quartile, 75th percentile).

$$IQR = Q_3 - Q_1 \qquad (2)$$

The interquartile range is often used to find outliers in data. Outliers here are defined as observations that fall below $Q_1 - 1.5\,IQR$ or above $Q_3 + 1.5\,IQR$. In a boxplot, the highest and lowest occurring value within this limit are indicated by whiskers of the box (frequently with an additional bar at the end of the whisker) and any outliers as individual points.

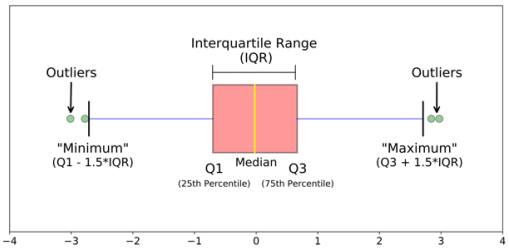


Figure 2: Illustration of IQR on Outlier Detection in Boxplot.

Compared to Z-score, the IQR method does not need to make assumption on statistical distribution of the feature. Here, 1.5 is a threshold that we can adjust. However, it still can deal with only one single feature at a time.

2.2 Unsupervised Machine Learning Approaches for Anomaly Detection

2.2.1 Distance-based Method: One-class SVM

Supervised anomaly detection requires labelled dataset that indicates whether a record is "normal" or "abnormal", while unsupervised approach deals with unlabeled dataset which assumes the majority of data are "normal" and tries to find data which are very different from normal ones as anomalies. Statistically, we use distance to describe how different one sample is from the other – similar points should be close in distance. Euclidean distance [7] and Manhattan distance [8] are two most commonly-used distance measures. Let $x_i = (x_{i,1}, \ldots, x_{i,m})$ denotes the i-th datapoint, and $x_{i,j}$ denotes the j-th feature of i-th datapoint (totally m features), we have:

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{m} (x_{1,j} - x_{2,j})^2} \text{ (Euclidean)} \qquad (3)$$

$$d(x_1, x_2) = \sum_{j=1}^{m} |x_{1,j} - x_{2,j}| \text{ (Manhattan)} \qquad (4)$$

One-class Support Vector Machine (SVM) is one of the unsupervised anomaly detection algorithms based on distance. SVM [9] algorithm was primarily used for binary classification problems. Let

[1] **Statistical Dispersion**: In statistics, dispersion (also called variability, scatter, or spread) is the extent to which a distribution is stretched or squeezed.

$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$ be the dataset, where $x_i \in R^m$ is the i-th input data and $y_i \in -1,1$ is the label of x_i. SVM algorithm tries to create a hyperplane (decision boundary) to separate class -1 and 1 by making projections on data, and support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Let w be the weights (coefficients) and b be the bias, the hyperplane is defined by:

$$w^T x + b = 0 \qquad (5)$$

The hyperplane determines the margin between the classes, class -1 and 1 are on two different sides of the hyperplane, and the distance from the closest point from each class to the hyperplane is equal. That is, with proper scaling on such distance, we can have $w^T x + b = 1$ for $y = 1$ and $w^T x + b = -1$ for $y = -1$. The distance from a point x^* to the hyperplane $w^T x + b = 0$ is given by:

$$\delta = \frac{|w^T x^* + b|}{||w||} \qquad (6)$$

If x^* is one of the support vectors, that is $y^* = 1$ or $y^* = -1$, we have $w^T x^* + b = \pm 1$, and then $\delta = 1/||w||$. In SVM algorithm, we want to maximize $1/||w||$ while subject to two constraints: 1) $w^T x + b \geq 1$ for $y = 1$ and 2) $w^T x + b \leq -1$ for $y = -1$, which can be written together as $y_i(w^T x_i + b) \geq 1$. With Lagrange Multiplier [10], the optimization problem can be described as:

$$w = \arg\min_w \frac{1}{2}\|w\|^2 + \sum_i \lambda_i (y_i (w^T x_i + b) - 1) \qquad (7)$$

In order to prevent the SVM classifier from over-fitting with noisy data, slack variables ε_i are introduced to generate soft margins, the idea of which is to allow some points sit on the wrong side of hyperplane.

$$w = \arg\min_w \frac{1}{2}\|w\|^2 + \sum_i \lambda_i \big(y_i (w^T x_i + b) - (1 - \varepsilon_i)\big) + C \sum_i \varepsilon_i \qquad (8)$$

One-class SVM [3] is similar, but instead of using a hyperplane to separate two classes of datapoints, it treats every datapoint as normal and uses a hyperplane to encompass all of them. Any datapoint that sit on the other side of hyperplane will be detected as anomalies.

2.2.2 Density-based Methods: DBSCAN and LOF

Density-based anomaly detection algorithms use the concept of "density" to define "abnormality" in a relative sense, starting from distance measures. One prominent method is Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [1], which determines density by counting the number of points within a specified radius. In DBSCAN, each point is classified according to the following criteria:

1) **Core Point:** A point is designated as a core point if it has at least a minimum number of neighboring points (MinPts) within the radius ε.
2) **Border Point:** A border point has fewer neighbors than MinPts within ε, but is within the εradius of a core point.
3) **Noise Point:** Points that are neither core points nor border points are classified as noise points.

The DBSCAN algorithm proceeds in two main steps after labeling the points:

1) Create a separate cluster for each core point or group of connected core points (core points are considered connected if they are within ε of each other).
2) Assign each border point to the cluster associated with its nearest core point.

While DBSCAN is widely used for clustering, it also effectively identifies anomalies by classifying noise points as anomalies, and treating all other points as normal. This dual functionality makes DBSCAN a versatile tool for both clustering and anomaly detection tasks.

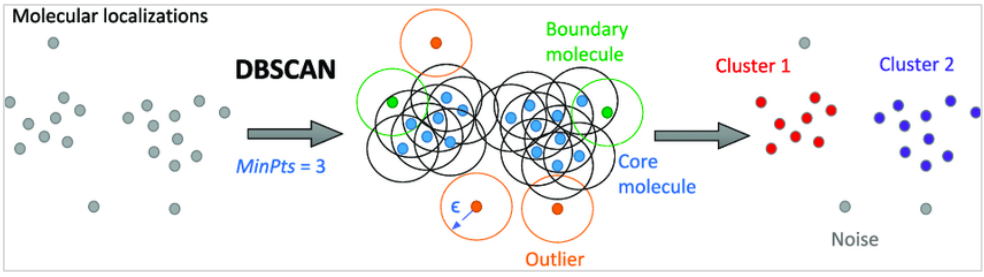


Figure 3: An Example Illustrating the Density-Based DBSCAN Clustering Method Applied to SMLM[2] Data.

[2] **SMLM:** Single-molecule localization microscopy describes a family of powerful imaging techniques that dramatically improve

Local Outlier Factor (LOF) [2] algorithm uses a different measure of density. Below are some important terminologies associated with LOF:

1) **k-distance:** is the distance between the point and its k-th nearest neighbor.

2) **k-neighbors:** the set of points that lie within the circle of radius of k-distance of x, denoted by $N_k(x)$.

3) **Reachability Distance**: the maximum of k-distance of x_j and the distance between x_i and x_j ; reachability distance is not symmetric.

$$rd_k(x_i, x_j) = \max\{\text{k\_distance}(x_j), d(x_i, x_j)\} \quad (9)$$

4) **Local Reachability Density:**

$$lrd\_k(x_i) = \frac{1}{\frac{1}{|N_k(x_i)|}\sum_{x_j \in N_k(x_i)} rd_k(x_i, x_j)} \quad (10)$$

5) **Local Outlier Factor:**

$$LOF_k(x_i) = \frac{\frac{1}{|N_k(x_i)|}\sum_{x_j \in N_k(x_i)} lrd_k(x_j)}{lrd_k(x_i)} \quad (11)$$

If the point is not an outlier (inlier), the ratio of average local reachability density (LRD) of neighbors is approximately equal to the local reachability density of a point, because the density of a point and its neighbors are roughly equal. In that case, LOF value is nearly equal to 1. On the other hand, if the point is an outlier, the LRD of a point is less than the average LRD of neighbors, then LOF value will be high.

The main issues related to distance or density-based algorithm is curse of dimensionality, especially those involve in nearest neighbor search. Furthermore, distance measures cannot be computed on feature space with (non-ordinal) categorical features. For example, even if we use one-hot encoding [11] to convert season into four indicator variables, the differences (e.g. $[0,1,0,0] - [1,0,0,0] = [-1,1,0,0]$) among them do not make sense.

2.2.3 New Modeling Concept: Isolation Forest

Unlike the distance and density-based approaches [1, 2, 3] we covered in Section 2.2.1 and Section 2.2.2 which try to build a model of normal datapoints, the Isolation Forest explicitly isolates anomalous datapoints. The Isolation Forest algorithm takes advantage of two quantitative properties of anomalous data points in a sample:

1) **Few**: anomalies are the minority consisting fewer instances;

2) **Different**: anomalies have attribute values that are very different from those of normal instances.

According to those two properties, there is the tendency of anomalies in a dataset to be easier to separate from the rest of the sample, compared to normal points. In order to isolate a data point, the algorithm recursively generates partitions on the sample by randomly selecting an attribute and then randomly selecting a split value for the attribute, between the minimum and maximum values allowed for that attribute. Recursive partitioning can be represented by a tree structure named Isolation Tree.

Given a dataset $X = \{x_1, \dots, x_n\} \in R^{n \times d}$, where d is the dimension of feature space. For subsample $X' \subset X$, the Isolation Tree (iTree) is defined as data structure with the following properties:

1) For each node T in the tree, T is either an external-node without child or an internal-node with one "test" and exactly two daughter nodes T_l and T_r.

2) A test at node T consists of an attribute q and a split value p such that the test $q < p$ determines the traversal of a data point to either T_l or T_r.

In order to build an iTree, the algorithm recursively divides X' by randomly selecting attribute q and a split value p, until either the node has only one instance or all data at the node have the same values. When the iTree is fully grown, each point in X is isolated at one of the external nodes. Intuitively, anomalies are those with the smaller path length in the tree, where the path length $h(x_i)$ of point $x_i \in X$ is defined as the number of edges x_i traverses from the root node to get to an external node.

The algorithm for computing the anomaly score of a data point is based on the observation that the structure of iTrees is equivalent to that of Binary Search Trees (BST) [12]: a termination to an external node of the iTree corresponds to an unsuccessful search in the BST. As a consequence, the estimation of average $h(x)$ for external node terminations is the same as that of the unsuccessful searches in BST, that is:

$$c(m) = \begin{cases} 2H(m-1) - \frac{2(m-1)}{n}, & m > 2 \\ 1, & m = 2 \\ 0, & m < 2 \end{cases} \quad (12)$$

spatial resolution over standard, diffraction-limited microscopy techniques and can image biological structures at the molecular scale.

where n is the testing data size, m is the size of the sample set, and H is the harmonic number, which can be estimated by $H(i) = \ln(i) + \gamma$, $\gamma = 0.5772156649$ is the Euler-Mascheroni constant. The value $c(m)$ represents the average path length of a BST built on m datapoints, which is used to normalize $h(x)$ and get an estimation of the anomaly score for a given instance x:

$$s(x, m) = 2^{\frac{-E[h(x)]}{c(m)}} \qquad (13)$$

where $E[h(x)]$ is average value of $h(x)$ from a collection of iTrees.

Anomaly detection with Isolation Forest is a process composed of two main steps:

1) Use training data to build iTrees;
2) Pass each instance in test set through all the iTrees and calculate the anomaly score

If anomaly score is close to 1, then the instance is very likely to be an anomaly; if anomaly score is smaller than 0.5, then the instance is likely to be a normal point. Furthermore, if all the instances have an anomaly score of around 0.5, then it is safe to assume that the sample does not have any anomalies.

Compared with distance and density-based methods, Isolation Forest has the following advantages:

1) Isolation Forest has a lower linear time complexity and small memory requirement (no need to store the training dataset in order to compute nearest neighbors);
2) All the iTrees in Isolation Forest are independent, which can be built on parallel or distributed environment to enhance training speed;
3) Isolation Forest is able to deal with high dimensional data with irrelevant features including one-hot encoded categorical features;
4) Isolation Forest can be trained with or without anomalies in the training set;
5) Isolation Forest can provide detection results with different levels of granularity without re-training.

2.3 An Extension to GBDT with Categorical Features: CatBoost

CatBoost is an improvement over GBDT (Gradient Boosting [13] Decision Tree), which applies special target statistics [15] (TS) technique to handle categorical features. The original GBDT method can be summarized like below:

Given a dataset $D = \{(x_i, y_i)\}_{i=1\dots n}$, each x_i consists of m features, and y_i is the target, which can be either discrete labels or continuous response. Let $F^{(t)}$ be the prediction function obtained from t-th iteration, which is updated in an additive manner: $F^{(t)} = F^{(t-1)} + \eta \cdot T^{(t)}$, where η is the step size (or learning rate), function $T^{(t)}$ represents the base predictor (decision tree) chosen to minimize the expected loss L:

$$T^{(t)} = \arg\min_T L\left(F^{(t-1)} + T\right) = \arg\min_T \mathbb{E}\left[L\left(y, F^{(t-1)}(x) + T(x)\right)\right] \qquad (14)$$

If the loss function L is chosen to be least square loss, then gradient is given by: $\frac{\partial L}{\partial F} = -2\left(y_i - F(x)\right)$. Ignoring the constant part, the base decision tree $T^{(t)}$ is actually fitting the residuals (gradient) $r_i^{(t)} = -\partial L/\partial F|_{F(x)=F^{(t-1)}(x)}$, this is where the name "Gradient" Boosting [14] comes from.

Target statistics method is commonly used in boosting tree to deal with categorical features. Let $x_{i,k}$ be the k-th feature of i-th sample, according to target statistics, it will be encoded as:

$$x_{i,k} = \frac{\sum_{j=1}^{n} 1_{[x_{j,k}=x_{i,k}]} \cdot y_j}{\sum_{j=1}^{n} 1_{[x_{j,k}=x_{i,k}]}} \qquad (15)$$

which is just the weighted average of target responses with regards to each level in feature k. However, using TS method directly may result in prediction shift or target leakage issue. Therefore, CatBoost employed Ordered TS [4] method. For each sample x_i, we randomly shuffle all the samples, and use the samples before x_i to compute the TS. In order to reduce the variance, we perform the random sampling and TS computation at each iteration, and thus different random sequences of samples were used throughout the training process.

In addition to ordered TS, CatBoost uses combinations of categorical features as additional categorical features which is capable of capturing high-order dependencies. CatBoost constructs such combination in a greedy way: for each split of a tree, CatBoost concatenates all the categorical features (including their combinations) already used for previous split in the current tree with all the remaining categorical features in the dataset, meanwhile those combinations are transferred into target statistics.

2.4 Concept of Weakly Supervised Learning

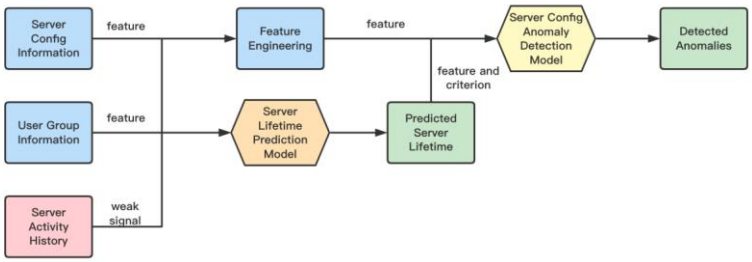
Weakly Supervised Learning encompasses scenarios where the quantity or quality of the supervision signal is significantly limited. According to Zhi-Hua Zhou [6], weak supervision can be categorized into three types. The first type is incomplete supervision, where only a limited portion of the training data is labeled, akin to the semi-supervised learning setting.

The second type is inexact supervision, which involves coarse-grained labels that lack sufficient detail. This typically occurs when the provided labels do not offer the necessary granularity. The third type is inaccurate supervision, which arises when the provided labels are not always accurate or true representations of the ground truth.

In our context, we lack human-generated labels to definitively determine whether a server configuration is appropriate. However, we have observed that customers tend to shut down servers with inadequate configurations and replace them with new ones, leading to a short server lifetime. Therefore, server lifetime can serve as an indirect indicator of whether a server configuration is effective or not. By utilizing server lifetime as a weak supervision signal, we align with the scenario of inaccurate supervision, where the signal is not always perfectly aligned with the ground truth but still provides valuable guidance for anomaly detection.

3 Methods

Figure 4: Flowchart of Server Configuration Anomaly Detection.



In this section, we describe the modeling details of the server configuration anomaly detection problem, including feature selection and model hyperparameter settings. Some experiments are done during this stage in order to validate the feature selection part, and we will show their results in Section 4. The process flowchart is given below:

3.1 Server Lifetime Prediction

The raw data comes from OneITLab cloud service platform, which contains 3147 servers, and 2904 of them is already shut down. We remove the servers that is marked as test server and only focus on those with clear record on creation and shutdown timestamp, which results in 1170 servers including common servers (CPU-only) and GPU servers. OneITLab is mainly used by students, professors, and researchers from Chinese Academy of Sciences. It offers cloud services like virtual server, storage, and clusters.

We first extract the server configuration, user group, and server activity history information from the backend database, and selected the following descriptive features to train CatBoost regression model to predict the server lifetime:

Table 1: Descriptive Features for Server Lifetime Prediction Model.

Feature Name	Variable Name	Examples
Number of CPUs	cpu_num	4, 8
Memory Size	memory_num	16, 32 (GB)
Disk Size	storage_size	30, 64, 128 (GB)
Internet Bandwidth	bandwidth	1, 100 (M)
Operating System	os_name	Ubuntu, CentOS
Number of GPUs	gpu_num	0, 1, 2
User Group	group	teacher, student, testbed user
Architecture Type	arch_type	X86, ARM

The model hyperparameters for CatBoost are configured as below:

Table 2: Hyperparameter Configuration of CatBoost Regressor.

Hyperparameter Name	Value
Categorical Feature Set	group, os_name, arch_type
Number of Iterations	1000
Decision Tree Structure	Symmetric
L2 Regularization Strength	3
Max Tree Depth	5
Learning Rate	0.05
Max Number of Leaves	64

3.2 Server Configuration Anomaly Detection Model

Minority Bias in anomaly detection models refers to the situation when samples from minority groups are more likely to be detected as anomalies, which results in high false positive rate. In our scenario, few users ordered servers with very high configurations, e.g. 10 CPU, 64GB RAM, 2048GB Disk. Since those server configurations appear to be rare in training data, unsupervised anomaly detection algorithms often detect them as anomalies, even though those server

configurations are reasonable for users with needs for large project or computationally intensive tasks. We employ two methods to handle the Minority Bias issue: 1) perform feature engineer to generate more representative descriptive features, 2) use predicted server lifetime as weak signal to filter the anomaly detection outcomes.

1) Feature Engineering

We use Pearson Correlation Coefficients [16] to analyze the correlation between continuous server configuration features described in Section 3.1. The formula of Pearson Correlation Coefficient is shown below:

$$\rho_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (16)$$

which has a range of -1 to 1. The absolute value of Pearson Correlation Coefficient indicates the strength of linear correlation between variable x and variable y, and its sign represents if they are positively or negatively correlated.

Table 3: Pearson Correlation Coefficients of Server Configurations

	cpu_num	memory_num	storage_num	bandwidth	gpu_num
cpu_num	1.00	0.97	0.49	0.23	0.67
memory_num	0.97	1.00	0.48	0.18	0.80
storage_num	0.49	0.48	1.00	0.15	0.27
bandwidth	0.23	0.18	0.15	1.00	0.00
gpu_num	0.67	0.80	0.27	0.00	1.00

Based on Pearson Correlation Coefficients, we find that those features are all positively correlated, which is consistent with that a computationally intensive task or large project usually requires all the configurations to be high. Intuitively, if someone selects a server with disk size smaller than memory size, it will be problematic. Therefore, as one of number of CPUs, memory size, and disk size increases, we expect the other to increase synchronically. One way to quantify this phenomenon is to use ratios, e.g. “disk size: memory size”, “memory size: number of CPUs”, and “disk size: number of CPUs”. If those kinds of ratios appear to be too high or too low, it is likely that corresponding configurations are abnormal.

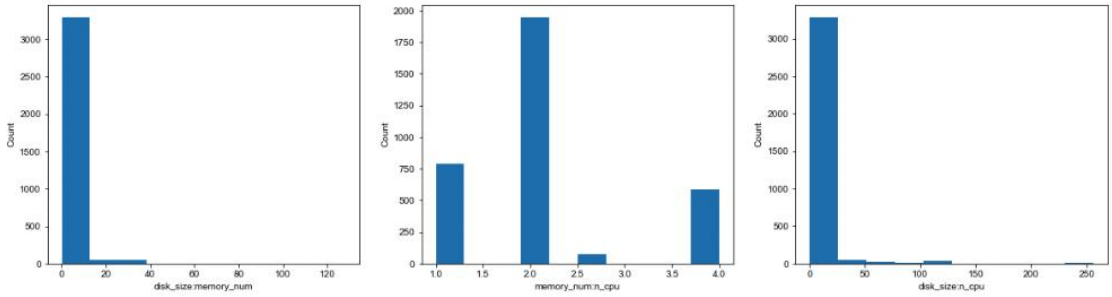


Figure 5: Histogram of “Disk Size: Memory Size”, “Memory Size: Number CPUs”, and “Disk Size: Number of CPUs” (from left to right)

However, if we check the histogram of those ratios, we find that “disk size: memory size” and “disk size: number of CPUs” follow long-tail distribution, and it is hard to detect low ratios. This is because number of CPUs usually increases exponentially, e.g. 2, 4, 8, 16, and etc., and the selection of memory size and number of CPU are usually bundled together. However, we can perform log transformations on those distribution to make them close to Gaussian distribution:

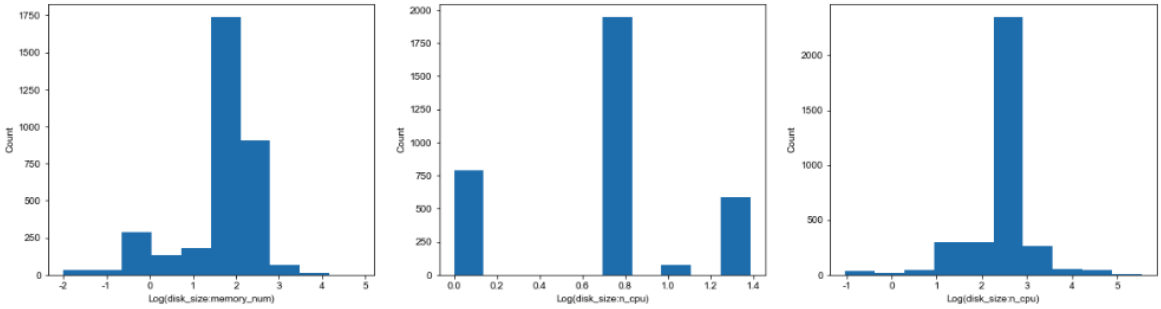


Figure 6: Histogram of “Log(Disk Size: Memory Size)”, “Log(Memory Size: Number CPUs)”, and “Log(Disk Size: Number of CPUs)” (from left to right)

With log transformations, it is easier to detect both high and low ratios. Moreover, in log transformation, the order in ratios does not matter, that is “disk size: memory size” and “memory size: disk size” look similar, and thus we only need to include one of them as descriptive features. Therefore, we construct three

log-transformed ratio features using disk_num, memory_num, and n_cpu, together with bandwidth and predicted server lifetime for the anomaly detection model.

Table 5: Descriptive Features for Anomaly Detection Model

Feature Name	Variable Name
Predicted Server Lifetime	pred_lifetime
Log(Memory Size: Number of CPUs)	log_memory2cpu
Log(Disk Size: Number of CPUs)	log_disk2cpu
Log(Disk Size: Memory Size)	log_disk2memory
Internet Bandwidth	bandwidth

2) Predicted Server Lifetime as Weak Signal

We use the features in table to build the Isolation Forest model, with the following hyperparameters:

Table 6: Hyperparameter Configuration of Isolation Forest Anomaly Detection Model

Hyperparameter Name	Value
Use Bootstrap Sample or Not	Yes
Contamination Ratio	0.01
Max Number of Features	1.0
Number of iTrees	1000
Max Server Lifetime	168

Here, the Max Server Lifetime is an additional hyperparameter that we add to this model, which serves as a threshold for us to select the final anomalies. 168 is in the unit of hours, which indicates one week's time ($7 \times 24 = 168$).

The original Isolation Forest model outputs an anomaly score ranged from 0 to 1for each data point, and those with a score close to 1 are selected as anomalies. However, in order to avoid Minority Bias issue, we won't select the server configurations with a predicted server lifetime more than one week as anomalies. This is because there should be a positive relation between server lifetime and customer's satisfaction, and intuitively, the configurations that users get used to should be reasonable and not selected as anomalies.

4 Experiments and Results

4.1 The Difference in User Groups

There are 6 user types in OneITLab's dataset: (1) student, (2) testbed, (3) teacher, (4) research project leader, (5) admin, and (6) super admin, and those roles are not mutually exclusive – one user can have multiple roles. Table 7 shows the average configuration selected by users from different groups.

Table 7: Average Configurations for Different User Group

User Groups	Average n_cpu	Average memory_num	Average bandwidth	Average storage_num	Number of Servers
Student	6.87	15.80	1.41	95.98	717
Testbed User	4.15	8.95	1.09	55.46	224
Teacher	5.49	13.11	1.46	79.96	1012
Research Project Leader	7.21	17.19	10.85	130.24	1013
Admin	5.00	12.98	0.95	51.85	324
Super Admin	7.94	23.21	1.78	60.16	107

As we can see from the table, research project leaders tend to choose servers with higher configurations than others while testbed user and admin usually select lower configurations.

In addition, we use entropy to quantify the diversity of configuration selections for each user groups. Entropy measures the impurity of a set of elements, which is given by the formula:

$$H(X) = \sum_{i=1}^{n} -p_i \cdot \log p_i \qquad (17)$$

Assume X is a set of elements taking up to n values, $v_1, \ldots, v_n$, then p_i is the probability that a randomly chosen element has a value of v_i, $p_i = \Pr(x = v_i)$. The higher the entropy, the higher diversity (impurity) of the list. Table shows the entropy in configurations of each user group.

Table 8: Entropy of Configurations for Different User Group

User Groups	n_cpu Entropy	memory_num Entropy	bandwidth Entropy	storage_num Entropy
Student	1.27	1.18	0.59	1.10
Testbed User	0.74	1.11	0.55	1.35
Teacher	1.15	1.21	0.49	1.26
Research Project Leader	1.34	1.29	0.86	1.44
Admin	0.89	0.78	0.45	0.31
Super Admin	1.31	1.30	0.54	0.59

We observed high diversity of configuration selection among research project leader and low diversity among admin and testbed users.

4.2 Impact of Log Ratios

In Section 3.2 we perform feature engineering, using the log transformations on configuration ratios to make it appear to follow Gaussian distribution. We argue that log transformation will make the patterns clear, such that it is easier for us to detect anomalies resulted from very high or very low configuration ratios. Taking server lifetime as weak supervision signal, we expect the log ratios can easily distinguish servers with very short lifetime.

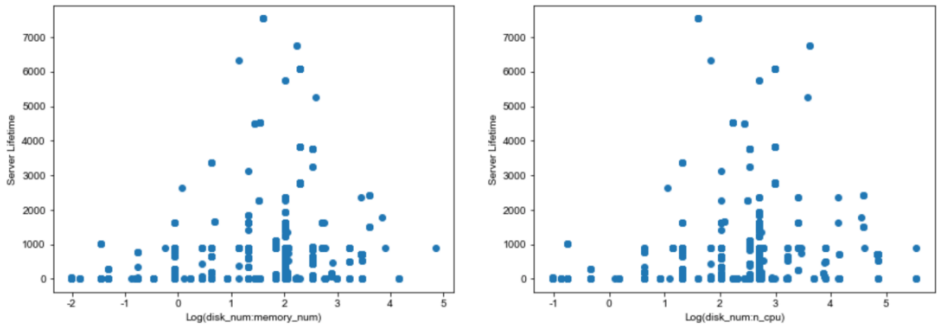


Figure 7: Scatterplot of Server Lifetime against Log(disk size: memory size) and Log(disk size: number of CPUs)

Figure 7 shows the scatter plot of server lifetime against two log ratio features, Log(disk size: memory size) and Log(disk size: number of CPUs). We observe that server lifetimes tend to be higher around the middle region of log ratios, which is consistent with our assumption.

4.3 Model Performance

Since we use CatBoost regressor to predict the server lifetime which is not a linear model, R Squared (Coefficient of Determination) is not a good evaluation metric in this case. Therefore, we use WMAPE (Weighted Mean Absolute Percent Error) to evaluate our model in addition to R Squared.

$$r^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (18)$$

$$WMAPE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{\sum_{i=1}^{n}|y_i|} \quad (19)$$

WMAPE is a measure of the absolute error relative to the true value, which ranges from 0 to infinity. WMAPE is a metric for non-negative target values, and low WMAPE value indicates good model performance. R Squared ranges from 0 to 1, and a value close to 1 indicates the model performance is good.

Table 9: Server Lifetime Prediction Model Performance

Metrics	Value
R Square	0.831
WMAPE	0.323

We implemented 10-Fold Cross Validation [3] to compute those metric values.

4.4 Selected Anomalies

Due to lack of labelled anomalies, we don't have ground truth to evaluate the performance of our anomaly detection model. However, as we can observe from Table 10, the detected anomalies all display mismatches of certain server configurations, especially those with too large or too small disk space compared to the CPU and memory selection.

[3] **K-Fold Cross Validation**: In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

Table 10: Detected Server Configuration Anomalies

Number of CPUs	Memory Size (GB)	Disk Storage Size (GB)	Network Bandwidth (M)	OS	Arch Type	User Group	Anomaly Scores
84	224	30	1	CentOS	arm	Teacher	0.68
84	224	30	1	CentOS	arm	Student	0.68
84	224	30	1	CentOS	arm	Research Project Leader	0.68
84	224	30	1	CentOS	arm	Super Admin	0.67
84	224	30	1	CentOS	x86	Teacher	0.68
84	224	30	1	CentOS	x86	Student	0.68
84	224	30	1	CentOS	x86	Research Project Leader	0.68
84	224	30	1	CentOS	x86	Admin	0.68
84	224	30	1	CentOS	x86	Teacher	0.68
84	224	30	1	CentOS	x86	Student	0.68
84	224	30	1	CentOS	x86	Research Project Leader	0.68
84	224	30	1	CentOS	x86	Admin	0.68
84	224	30	1	CentOS	x86	Teacher	0.68
84	224	30	1	CentOS	x86	Student	0.68
84	224	30	1	CentOS	x86	Research Project Leader	0.68
16	64	4096	50	Ubuntu	x86	Teacher	0.73
16	64	4096	50	Ubuntu	x86	Student	0.73
16	64	4096	50	Ubuntu	x86	Research Project Leader	0.73
90	240	100	50	CentOS	x86	Research Project Leader	0.71

5 Conclusions

In this paper, we have summarized and analyzed the limitations of classical anomaly detection algorithms, specifically highlighting the issue of Minority Bias. Traditional methods, including statistical approaches and unsupervised learning techniques, often struggle with incorporating both continuous and categorical features effectively, leading to suboptimal performance in complex scenarios such as server configuration selection in cloud computing platforms.

To address these limitations, we proposed a novel weakly supervised method that leverages predicted server lifetime as a weak signal to guide the anomaly detection process. Our approach utilizes the CatBoost Regressor, which is adept at handling both categorical and continuous features, thereby considering the diverse customer behaviors across different user groups. By predicting server lifetime based on server configurations and user group information, we effectively integrate the nuances of user interactions and preferences into the anomaly detection framework.

The predicted server lifetime serves a dual purpose: it acts as a descriptive feature in the Isolation Forest anomaly detection model and as an outcome threshold to filter detected anomalies. This dual use allows the model to incorporate the impact of categorical features such as user groups, operating systems, and architecture types, thereby mitigating the risk of Minority Bias. The inclusion of these features enhances the model's ability to accurately detect misconfigured servers, ensuring a more robust and reliable anomaly detection process.

Our method demonstrates several advantages. Firstly, it integrates both continuous and categorical data, enabling a comprehensive analysis of server configurations. Secondly, by using server lifetime as a weak signal, we enhance the model's performance and reliability. Lastly, our approach is scalable and computationally efficient, making it suitable for deployment in large-scale cloud computing platforms.

Future work will involve further refining the model to handle more complex configurations and expanding the scope to include additional types of cloud services. Additionally, exploring other weakly supervised learning techniques and incorporating more diverse data sources could further improve the robustness and accuracy of the anomaly detection process. By continuing to enhance these methods, we aim to provide cloud computing platforms with more effective tools for ensuring optimal server configurations and enhancing overall customer satisfaction.

References

1. Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, & U. M. Fayyad (Eds.), Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96) (pp. 226–231). AAAI Press. https://www2.cs.uh.edu/~ceick/7363/Papers/dbscan.pdf
2. Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying Density-based Local Outliers. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (pp. 93–104). ACM. https://doi.org/10.1145/335191.335388
3. Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (1999). Support Vector Method for Novelty Detection. In Advances in Neural Information Processing Systems (Vol. 12, pp. 582–588). MIT Press.
4. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased Boosting with Categorical Features. In Advances in Neural Information Processing Systems (Vol. 31). https://arxiv.org/abs/1706.09516
5. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. In Proceedings of the 2008 IEEE International Conference on Data Mining (pp. 413–422). IEEE. https://doi.org/10.1109/ICDM.2008.17
6. Zhou, Z.-H. (2018). A Brief Introduction to Weakly Supervised Learning. National Science Review, 5(1), 44–53. https://doi.org/10.1093/nsr/nwx106
7. Smith, K. (2013). Precalculus: A Functional Approach to Graphing and Problem Solving. Jones & Bartlett Publishers.
8. Black, P. E. (2019). Manhattan distance. In Dictionary of Algorithms and Data Structures. National Institute of Standards and Technology.
9. Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297. https://doi.org/10.1007/BF00994018
10. Beveridge, G. S. G., & Schechter, R. S. (1970). Lagrangian Multipliers. In Optimization: Theory and Practice (pp. 244–259). New York: McGraw-Hill
11. Brownlee, J. (2020). Ordinal and One-Hot Encodings for Categorical Data. Machine Learning Mastery. https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/
12. Culberson, J., & Munro, J. I. (1989). Explaining the Behaviour of Binary Search Trees Under Prolonged Updates: A Model and Simulations. The Computer Journal, 32(1), 68–69. https://doi.org/10.1093/comjnl/32.1.68
13. Roe, B. P., Yang, H.-J., Zhu, J., Liu, Y., Stancu, I., & McGregor, G. (2005). Boosted decision trees as an alternative to artificial neural networks for particle identification. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 543(2), 577–584. https://doi.org/10.1016/j.nima.2004.12.018
14. Friedman, J. H. (1999). Greedy Function Approximation: A Gradient Boosting Machine. In Annals of Statistics (Vol. 29, pp. 1189–1232). https://doi.org/10.1214/aos/1013203451
15. Micci-Barreca, D. (2001). A Preprocessing Scheme for High-cardinality Categorical Attributes in Classification and Prediction Problems. ACM SIGKDD Explorations Newsletter, 3(1), 27–32. https://doi.org/10.1145/507533.507538
16. Wright, S. (1921). Correlation and causation. Journal of Agricultural Research, 20(7), 557–585.