sciendo

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **23**, No 4
Sofia • 2023 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2023-0034

Fault Tolerance of Cloud Infrastructure with Machine Learning

Chetankumar Kalaskar, Thangam S.

Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwavidyapeetam, Bangalore, 560035 Karnataka, India
E-mails: k_chetankalaskar@blr.amrita.edu s_thangam@blr.amrita.edu

***Abstract*:** *Enhancing the fault tolerance of cloud systems and accurately forecasting cloud performance are pivotal concerns in cloud computing research. This research addresses critical concerns in cloud computing by enhancing fault tolerance and forecasting cloud performance using machine learning models. Leveraging the Google trace dataset with 10000 cloud environment records encompassing diverse metrics, we systematically have employed machine learning algorithms, including linear regression, decision trees, and gradient boosting, to construct predictive models. These models have outperformed baseline methods, with C5.0 and XGBoost showing exceptional accuracy, precision, and reliability in forecasting cloud behavior. Feature importance analysis has identified the ten most influential factors affecting cloud system performance. This work significantly advances cloud optimization and reliability, enabling proactive monitoring, early performance issue detection, and improved fault tolerance. Future research can further refine these predictive models, enhancing cloud resource management and ultimately improving service delivery in cloud computing.*

***Keywords*:** *Cloud computing, Fault tolerance, Machine learning, Reliability of cloud.*

1. Introduction

Cloud computing represents a transformative computing service paradigm that offers significant advantages to both small-scale users and large-scale commercial and scientific applications. This paradigm is defined as a model that facilitates ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. These resources encompass networks, servers, storage, applications, and services, all of which can be rapidly provisioned and deprovisioned with minimal management effort or interaction with service providers [1]. The core attributes of cloud computing include on-demand access, resource autonomy, rapid elasticity, and continuous availability [2]. Cloud resources are allocated using standard protocols, such as IAM, OAuth, OpenID for authentication, and AMI, OVF, SOAP, and REST for data and workload migration [3-4]. The adoption of these standards fosters the

widespread acceptance of cloud services. Furthermore, the cloud delivers enhanced business agility while simultaneously reducing costs, making it an appealing choice for a diverse user base. This combination of features positions cloud computing as a driving force in the contemporary technology landscape.

Cloud computing, while garnering significant attention is still considered in its adolescence concerning its fault-handling capabilities [5]. The architecture of cloud computing is characterized by its dynamic and increasingly complex nature [6-7]. Unlike traditional systems, cloud deployments rely on millions of commodity components, making them more susceptible to faults and failures [8]. In this context, a fault is identified as an abnormal condition or defect in one or more parts of a system, potentially rendering the system incapable of performing its intended functions [9]. The occurrence of a fault within the system leads to errors, defined as deteriorations in one or more system components that deviate the system from its normal state [10]. These errors, left unaddressed, progress into system failures, disrupting the regular service delivery and degrading system performance. Inadequate handling of these system failures can render the system inoperable. The consequences of such failures can be severe, to the extent that they may have a detrimental impact on the economic stability of the service provider. It is therefore imperative to address and enhance fault tolerance within cloud computing to ensure the uninterrupted delivery of services and safeguard the economic health of the provider. Fault tolerance is a critical concept denoting a system's ability to maintain its intended functionality in the presence of faults [11-12]. In the absence of fault tolerance, even a well-designed system composed of the finest components and services cannot be considered reliable [13-15]. The importance of reliability is particularly pronounced in the realm of cloud computing, given the execution of a substantial number of delay-sensitive, real-time applications. Additionally, the reliability of services plays a pivotal role in ensuring the widespread acceptance of cloud solutions. As a result, the issue of fault tolerance has garnered significant attention in research, leading to the development of numerous fault tolerance frameworks documented in the literature. This paper aims to contribute by presenting a comprehensive survey of fault tolerance within the cloud computing environment.

1.1. Motivation of the study

The motivation for this study is deeply rooted in the evolution of cloud computing, which has emerged as a transformative force in information technology. As cloud services become increasingly integral to both small-scale users and large-scale enterprises, ensuring the fault tolerance and performance predictability of these systems has become paramount. The distributed and dynamic nature of cloud environments, coupled with the reliance on millions of commodity components, has made them susceptible to faults and failures. These faults can lead to errors and, if not properly handled, result in system failures, disrupting service delivery and potentially causing economic strain for service providers. Therefore, addressing these challenges is essential to enhance the reliability and performance of cloud computing. Machine learning stands out as a promising avenue to bolster fault tolerance and improve the prediction of cloud system performance. Its adaptability and data-driven

capabilities make it an ideal candidate for mitigating the limitations of current fault tolerance approaches. By systematically leveraging machine learning models, the study aims to empower cloud systems with proactive monitoring and early detection of performance issues, ultimately ensuring fault tolerance and improved resource management. The use of the Google trace dataset, a rich source of real-world cloud environment data, adds a practical dimension to the research, offering a foundation for the development and evaluation of these machine learning models. The motivation behind this study is to address the critical concerns within cloud computing by harnessing the potential of machine learning. By advancing fault tolerance and enhancing cloud performance prediction, the research aims to contribute significantly to the optimization and reliability of cloud computing. It not only offers practical solutions for improved fault tolerance but also aligns with the dynamic nature of cloud systems, ensuring their continuous and dependable service delivery. The ultimate goal is to pave the way for more resilient and efficient cloud computing, benefitting both users and service providers.

1.2. Organization of the paper

The paper's structure is as follows. Section 2 conducts an in-depth review of existing literature, exploring prior research on fault tolerance, cloud computing, and machine learning in the context of this study. It identifies gaps and opportunities, emphasizing the need for innovative approaches like machine learning to enhance fault tolerance in cloud environments. Section 3 explains the proposed methodology, data sets used, and machine learning models used, including linear regression, decision trees, and gradient boosting, focusing on their application for improving fault tolerance and cloud system performance prediction. Section 4 of the paper presents research findings, including the performance of machine learning models and a feature importance analysis highlighting the most influential factors impacting cloud system performance. Section 5 summarizes key findings and their broader significance in fault tolerance and cloud performance prediction. It addresses study motivations, acknowledges limitations, and suggests future research directions, offering a view of the work's contributions and potential impact.

2. Related works

The roots of fault tolerance research stretch back to the 1940s with the inception of reliability engineering. The primary objective of reliability engineering has been to design systems capable of enduring faults and maintaining operational effectiveness, even in the face of failures. As computer systems gained prominence, the focus of fault tolerance research shifted towards developing methods to ensure the dependability and continuous availability of these systems. In the specific context of cloud computing, fault tolerance research has gained paramount significance, largely due to the distributed and dynamically evolving nature of cloud environments. Cloud computing traces its lineage to the domain of distributed systems research, which centers on constructing systems that can function seamlessly, even when individual

components encounter failures. In recent times, fault-tolerant cloud architectures, algorithms for placing fault-tolerant virtual machines, and advanced fault detection and recovery techniques have taken center stage in research endeavors.

In [16], a survey has provided an extensive examination of intelligent resource management techniques in the context of fault tolerance within cloud computing. This research discussed a range of methodologies, including machine learning, fuzzy logic, and swarm intelligence, while also presenting a comparative analysis of their strengths and weaknesses. Additionally, the study emphasizes the imperative for further exploration in this area to address the intricate challenges related to fault tolerance in complex cloud systems. In [17] is introduced a robust framework for data-intensive computing in cloud environments with a focus on fault tolerance. This framework has integrated key components, including data replication, workload balancing, and fault detection and recovery mechanisms, with the overall aim of ensuring high availability and optimal performance for data-intensive computing tasks in cloud settings. Furthermore, an additional research effort introduced a reliability-aware approach to fault tolerance in parallel processing within cloud computing. This approach incorporates a fault model rooted in reliability awareness to identify critical system components and allocate additional resources to enhance fault tolerance. It has also implemented a sophisticated load-balancing mechanism to optimize cloud system performance and efficiency, marking a substantial advancement in this field. In [18], a significant research effort focused on enhancing fault tolerance for containerized applications in cloud environments. This work introduces a technique that combines checkpointing and container migration to ensure fault tolerance and minimize system downtime in the event of failures. Additionally, the research includes a comprehensive performance evaluation of this technique using real-world applications, showcasing its effectiveness and efficiency. Another influential study addresses [19] dynamic replication schemes for fault tolerance. This research proposes a dynamic replication scheme for cloud computing that harnesses a prediction model to determine the optimal number of replicas needed to ensure fault tolerance. Furthermore, the scheme has incorporated a load-balancing mechanism to ensure high system performance and efficiency. In addition, a comprehensive survey paper [20] explores various techniques related to fault tolerance resource management in distributed cloud systems. This survey provides an in-depth analysis of concepts such as redundancy, replication, and checkpointing, offering a comparative assessment of their strengths and weaknesses. The paper emphasizes the ongoing need for further research in this domain to effectively address the intricate challenges associated with fault tolerance in distributed cloud systems. In [21], a research article introduces an extensive framework designed to ensure fault tolerance and high availability of containerized microservices within the cloud. This framework employs a combination of checkpointing, replication, and load-balancing techniques to achieve these objectives. Additionally, the paper includes a thorough performance assessment of the proposed framework using a real-world use case. Paper [22] offers a systematic review of fault-tolerant resource management techniques within the context of cloud computing. The paper delves into various methods, including redundancy, replication, checkpointing, and load balancing, and

conducts a comparative analysis of their strengths and weaknesses. It introduces a machine learning-based approach for predicting service availability in the cloud, utilizing historical data and machine learning techniques to achieve highly accurate predictions. The paper also provides a performance evaluation of this approach using real-world datasets, demonstrating its effectiveness and efficiency as presented in [23]. Paper [24] presents an innovative cloud-based architecture designed for Internet of Things (IoT) applications. This architecture employs a combination of redundancy, replication, and load balancing to ensure fault tolerance and the continuous availability of IoT applications. The paper also includes a comprehensive performance evaluation based on a real-world IoT use case. Additionally, [25] introduces an efficient technique for ensuring fault tolerance in edge computing environments. This method combines redundancy and replication to guarantee high resource availability in edge computing scenarios. The paper also offers a performance assessment using a real-world edge computing use case. Paper [26] proposes an architecture for service selection based on consumer feedback in service-oriented environments. Furthermore, [27] introduces dynamic resource provisioning for cloud applications through Bayesian learning. The study presented in [28] explores smart city video surveillance using fog computing, while [29] delves into the issues and future directions of fog computing. According to [30], a hardware setup for vehicle-to-vehicle communication under foggy conditions is presented. In [31], the focus is on power consumption prediction in cloud data centers using machine learning techniques. Paper [32] proposes a hybrid cloud approach for efficient data storage and security. Paper [33] provides a comprehensive study on evolutionary games in cloud, fog, and edge computing. Moreover, an exploration of the potential applications of integrating blockchain technology with fog computing is found in [34], revealing diverse practical implications. Lastly, [35] focuses on serverless High-Performance Computing over the Cloud, discussing the concept of serverless computing and its role in achieving high-performance computing capabilities within a cloud environment. These collective studies significantly contribute to a deeper understanding of modern computing paradigms and their wide-ranging applications.

In this comprehensive survey of the literature, a number of noteworthy research gaps and opportunities in the realm of fault tolerance within cloud computing and its associated fields have been discerned. A notable trend in the surveyed literature is a significant reliance on reactive fault tolerance methods, a preference that can be attributed to concerns regarding increased overhead and the complexities associated with proactive fault tolerance approaches. Notably, replication emerges as the most frequently employed fault tolerance technique, with checkpoint restart and job migration being other prevalent methods. To address these identified gaps and opportunities, the focus of this research endeavor shifts towards the exploration of the potential of the machine and deep learning techniques as promising solutions for mitigating the limitations of existing fault tolerance strategies. These advanced methodologies, characterized by their adaptability and data-driven capabilities, offer a compelling avenue for enhancing fault tolerance mechanisms, with the ultimate goal of advancing the reliability and performance of cloud computing and its associated fields.

The research initiative is laser-focused on addressing these pressing concerns. Dedication to developing and meticulously evaluating machine learning models that bolster fault tolerance and empower the precise prediction of cloud system performance is paramount. To support this investigation, we leverage the Google Trace dataset, a rich source of information containing 10000 records from an actual cloud environment. This dataset encompasses an array of diverse system metrics and performance indicators, providing a solid foundation for these research efforts.

The investigation encompasses the following key aspects:

- A diverse range of machine learning algorithms, such as linear regression, decision trees, and gradient boosting, have been systematically harnessed to build predictive models with the objectives of improving fault tolerance and predicting cloud system performance.
- A central aspect of the research encompassed a comparative analysis of the machine learning models. Performance evaluation has been conducted with a focus on metrics such as accuracy, precision, recall, and other pertinent indicators, ultimately leading to the identification of the most effective models.
- An in-depth examination was conducted to identify the most influential factors affecting cloud system performance and fault tolerance. This feature importance analysis provides insights into the elements that have the most significant impact on system behavior.

The research offers several notable advantages, presenting practical solutions for enhancing fault tolerance and predicting cloud system performance. The application of machine learning facilitates proactive monitoring and early performance issue detection, ultimately resulting in heightened reliability and improved resource management within cloud environments. Nonetheless, it is crucial to recognize specific limitations. The study is predominantly centered on a particular dataset, and the suitability of the developed models may exhibit variability in distinct cloud environments. The efficacy of these models could further hinge on the selection of machine learning algorithms and feature subsets. Moreover, the study does not provide a comprehensive exploration of the real-time intricacies and complexities inherent in cloud systems. In future research endeavors, several promising directions emerge. First, the exploration of a broader array of cloud datasets is warranted to assess the adaptability and robustness of machine learning models across diverse cloud environments. Additionally, there is a need for continual refinement of machine learning algorithms and the exploration of advanced techniques to enhance the accuracy of cloud system performance predictions. Real-time monitoring methods, integrated with machine learning, should be developed to enable immediate fault detection and mitigation, aligning with the dynamic characteristics of cloud systems. Investigating the effectiveness of ensemble methods, which harness the collective strength of multiple machine learning models, holds the potential to further fortify fault tolerance and performance prediction. Furthermore, the exploration of hybrid approaches that amalgamate conventional fault tolerance methods with machine learning offers the promise of maximizing system reliability in the face of the ever-evolving cloud environment.

3. The proposed methodology

In the proposed methodology for fault detection using the Google Trace dataset, the process begins with meticulous data preprocessing, including data acquisition and labeling. The dataset is split into a training set and a testing set and the issue of class imbalance is addressed by creating under-sampled and oversampled subsets. Multiple machine learning models are selected, trained, and evaluated on both types of datasets, allowing for a comprehensive understanding of their performance under different data sampling conditions. Performance metrics, such as accuracy, sensitivity, and specificity, are employed to assess model performance, and the top models are selected for further analysis. Crucially, feature importance analysis is conducted to identify the most influential factors in fault detection, with an emphasis on common features among the top-performing models.

If no common features are initially found, an iterative approach is adopted to refine the selection process. The final step of the methodology involves proposing an algorithm for fault detection based on the selected common features. This algorithm is then rigorously tested on a separate testing dataset to evaluate its effectiveness in identifying faults within cluster computing environments. Throughout this process, comprehensive documentation and reporting are maintained to capture the results and insights obtained, providing a foundation for iterative improvement and further research in the field of fault tolerance in cluster computing systems.

3.1. Data set

The Google Trace dataset, a publicly available resource, offers insights into real-world events within Google's production clusters. Researchers have extensively employed this dataset to explore diverse facets of cluster computing, including fault tolerance, energy consumption, and job scheduling. A prevalent application of the Google Trace dataset is its role in modeling fault tolerance within cluster computing systems. Researchers leverage this dataset to gain a deeper understanding of cluster behavior in the presence of faults, ultimately leading to the development of more robust fault tolerance mechanisms. The dataset has been instrumental in investigating various aspects of fault tolerance, encompassing checkpointing, replication, and recovery. For instance, researchers have assessed the efficacy of different checkpointing strategies by analyzing fault frequency and the time required for job checkpointing. Additionally, the dataset has been utilized to examine the impact of replication on fault tolerance, gauging the performance of jobs with and without replication. Constructed from traces originating from multiple Google data centers, the Google Trace dataset draws from anonymized job-level traces found in Google's workload traces repository. These traces span several years, commencing in 2011, and encompass both single-tenant and multi-tenant data centers. The dataset comprises aggregated data on resource utilization, including CPU, memory, and disk I/O for each job, complemented by metadata such as job ID, submission time, and job duration. Moreover, the dataset incorporates information regarding the failure characteristics of each job, shedding light on the number and types of failures encountered during job execution. Google has periodically released updated versions of the dataset, with the latest iteration being the Google Cluster-Usage Trace v2. The

dataset serves as a valuable resource for academic research, finding applications in numerous studies related to resource management, job scheduling, and fault tolerance in data centers. Its extensive citation in over 1000 research papers underscores its pivotal role in the domain of data center management and optimization. The Google Trace dataset is not limited to one cluster but encompasses traces from two distinct clusters: a production cluster and a cluster designated for research purposes. The production cluster trace, sourced from Google's Borg cluster management system, spans a one-month period in 2011. It encompasses an impressive volume of data, including approximately 13.25 million job submissions, 110 million task submissions, and a staggering 1.5 billion task events. The dataset provides insights into job and task details, submission times, resource requirements, and runtime statistics like CPU utilization and memory consumption. Complementing this, the research cluster trace offers data collected over a one-year duration, ranging from 2011 to 2012. It encompasses around 12,000 job submissions and 63,000 task submissions, offering similar information as the production cluster trace. Notably, both traces feature data pertaining to machine failures and job/task migration events, rendering them invaluable for research related to fault tolerance and recovery within cluster computing. Researchers have harnessed the Google Trace dataset to advance scheduling and resource management algorithms tailored for large-scale computing systems.

Variables used. The Google trace dataset encompasses a comprehensive set of variables, each serving a unique purpose in describing the dynamics of workload and resource utilization within a data center. These variables include essential identifiers like Job ID, Task Index, and Machine ID, temporal information such as Time and Duration, as well as resource-specific metrics like CPU Usage, Memory Usage, Input/output Usage, Network Usage, and Disk Usage.

Table 1. Variable featured in the Google trace dataset

Variable name	Description
Job ID	A unique identifier for each job submitted to the data center
Task index	A unique identifier for each task within a job
Machine ID	A unique identifier for each machine in the data center
Time	The timestamp at which a task was submitted to the data center
Duration	The time taken by a task to complete
CPU usage	The amount of CPU utilized by a task
Memory usage	The amount of memory utilized by a task
Input/output usage	The number of input/output operations performed by a task
Network usage	The amount of network bandwidth utilized by a task
Disk usage	The amount of disk space utilized by a task
Task type	The type of task performed (e.g., map, reduce, shuffle)
Job type	The type of job submitted to the data center (e.g., batch, interactive)
Priority	The priority assigned to a task within a job
Machine type	The type of machine on which a task was executed (e.g., small, medium, large)
Zone	The geographical zone in which a machine was located
CPU requested	The amount of CPU requested by a task
Memory requested	The amount of memory requested by a task
Disk requested	The amount of disk space requested by a task
Network requested	The amount of network bandwidth requested by a task
Input/output requested	The number of input/output operations requested by a task

Moreover, task-related characteristics, including Task type and Job type, are featured, along with priority assignments. Information about the hardware environment is also available through Machine type, Zone, and resource allocation specifications, including CPU requested, Memory requested, Disk requested, Network requested, and Input/output requested. This diverse range of variables equips researchers with a rich dataset to analyze and model data center operations effectively. Table 1 provides a comprehensive list of the variables featured in the dataset.

3.2. Addressing class imbalance

The Google trace dataset exhibits class imbalance, with a small fraction of jobs labeled as "failed" (1.5%) and the majority as "successful" (98.5%). This class imbalance can challenge machine learning models, potentially leading to biases and difficulties in accurately predicting the minority class. To mitigate this, several techniques can be applied, such as oversampling the minority class or adopting cost-sensitive learning strategies.

3.3. Model training and testing

Ten machine-learning models have been employed to analyze and enhance the Google Trace dataset for fault detection. These models have been systematically trained and tested using two distinct data sampling methods: under-sampling and oversampling. The selected models are as follows:
Models trained with the under-sampling dataset:

- M1U – C5.0;
- M2U – eXtreme Gradient Boosting – TREE;
- M3U – Model Averaged Neural Network;
- M4U – AdaBoost.M1;
- M5U – Bayesian Generalized Linear Model.

Models trained with the oversampling dataset:

- M1O – C5.0;
- M2O – eXtreme Gradient Boosting – TREE;
- M3O – Model Averaged Neural Network;
- M4O – AdaBoost.M1;
- M5O – Bayesian Generalized Linear Model.

3.4. Performance evaluation

The performance evaluation of the machine learning models using the Google trace dataset has been a rigorous and multifaceted process aimed at assessing their effectiveness in fault detection and management. To ensure a comprehensive understanding of the models' performance, a wide array of performance metrics has been employed, encompassing key aspects of fault detection and prediction. These metrics included True positives, False negatives, True negatives, False positives, Accuracy, Sensitivity, Specificity, Positive predictive value, Negative predictive

value, Prevalence, Detection rate, Detection prevalence, and Balanced accuracy. By evaluating these metrics, researchers have been able to gain profound insights into the models' strengths and weaknesses in identifying both actual faults and non-fault scenarios.

Of particular significance were the Sensitivity and Specificity metrics, which measure the models' abilities to avoid false negatives (missing actual faults) and false positives (incorrectly identifying faults in non-fault situations). The consideration of these metrics was especially vital in the context of an imbalanced dataset, where a small percentage of jobs have been labeled as "failed" while the majority were labeled as "successful". The imbalanced nature of the dataset has posed unique challenges, as machine-learning models might develop biases towards the majority class, potentially hindering their ability to predict accurately the minority class of faults. The 95% Confidence interval for accuracy has been also employed to estimate the range within which the true accuracy of the models was likely to fall, accounting for potential uncertainty in the accuracy measurement.

Furthermore, the amalgamation of these metrics has provided a comprehensive gauge of model performance, with Balanced Accuracy serving as a crucial composite indicator that considered the equilibrium between Sensitivity and Specificity. This evaluation process has been designed with the primary objective of singling out the top-performing models that possess the capability to adeptly identify and manage faults within intricate systems. Essentially, it afforded researchers a holistic panorama of the models' fault tolerance capacities, empowering them to make judicious choices concerning model selection and the identification of the pivotal features that have exerted influence over fault detection. These discoveries have played a pivotal role in shaping the formulation of an algorithm proposed for fault detection, intended to harness the collective potential of these models in real-world applications, thereby elevating system reliability and fortifying fault tolerance.

Through a meticulous examination of these metrics, it becomes feasible to discern the most exemplary models. To ensure an equitable and comprehensive assessment, it is prudent to opt for at least one model from each category of datasets, namely the undersampled and oversampled sets, in the quest to pinpoint the model that excels in performance. Once the models have been cherry-picked, the subsequent step involves the precise identification of the salient features that exert a significant impact on fraud detection.

3.5. Proposed algorithm for fault detection

By selecting the most common and significant features from all chosen models, an algorithm can be formulated to detect faults based on these important features. The proposed algorithm follows a structured approach to ensure robust fault detection. This includes iterative steps to identify the most influential features in the context of fault detection.

Step 1. Commence by labeling the dataset with appropriate class labels, distinguishing between "failed" and "successful" cases. To further enhance the fault detection process, assign M-scores or weights to relevant data points based on feature importance.

Step 2. Divide the dataset into two distinct sets, namely training and testing data, allowing for model training and evaluation.

Step 3. Subdivide the training set into two subsets: one for under-sampled data and the other for oversampled data. This approach aims to address the class imbalance challenge.

Step 4. Randomly select multiple machine learning models to generate trained models for classification. These models will be trained on both the under-sampled and oversampled datasets.

Step 5. Choose a performance metric that aligns with the specific objectives of the fault detection process. For instance, ROC (Receiver Operating Characteristic) can be employed for optimizing the training process and evaluating model performance.

Step 6. Extract relevant performance metrics for each trained model. These metrics should encompass key aspects such as Accuracy, Sensitivity, and Precision for both the positive and negative classes and other metrics that capture the model's effectiveness.

Step 7. Select the top N models based on essential performance metrics. These metrics guide the identification of the most proficient models in fault detection. For instance, N can be set to 2 for this purpose.

Step 8. Ensure a balance between the selected models from both the under-sampled and oversampled datasets to maintain comprehensive fault detection capabilities.

Step 9. Examine and evaluate the significance of individual features on a scale of 0 to 100 for the selected models. These scores are essential in identifying influential features.

Step 10. Identify common features among the top N features from all the selected models. This is a crucial step in pinpointing the key variables contributing to effective fault detection.

Step 11. If there are no common features among the top N features, consider reducing the number of models to $N-2$ for a more specific common feature selection. This process ensures the most relevant features are identified.

Step 12. In the event that common features remain elusive, reselect the N models from the full set of trained models, possibly by altering the metrics chosen for selecting the top models.

Step 13. Iterate through Steps 8-12 until at least three top features are consistently identified. This iterative approach hones the selection process and identifies robust fault detection features.

By following this structured algorithm given in Table 2, fault detection can be enhanced through the identification and utilization of essential features and the selection of top-performing models, ultimately contributing to more effective fault tolerance and system reliability in cloud computing and data center management.

Table 2. Pseudocode of the proposed method

1. Initialize the dataset and labels
2. Calculate M-scores for dataset entries based on feature importance
3. Split the dataset into training and testing sets
4. Create two subsets: one for undersampled data and one for oversampled data
5. Randomly select machine learning models:
- M1U (C5.0 – Undersampled)
- M2U (eXtreme gradient boosting – TREE – undersampled)
- M3U (Model averaged neural network – undersampled)
- M4U (AdaBoost.M1 – undersampled)
- M5U (Bayesian generalized linear model – undersampled)
- M1O (C5.0 – oversampled)
- M2O (eXtreme gradient boosting – TREE – oversampled)
- M3O (Model averaged neural network – oversampled)
- M4O (AdaBoost.M1 – oversampled)
- M5O (Bayesian generalized linear model – oversampled)
6. For each model:
a. Train the model using the undersampled training dataset
b. Train the model using the oversampled training dataset
7. Select a performance metric, e.g., ROC, for model training optimization
8. For each trained model:
a. Extract performance metrics including Accuracy, Sensitivity, Precision for positive class, and Precision for negative class
9. Select the top N models based on performance metrics, where $N = 2$
10. Ensure a balanced selection of models from both undersampled and oversampled datasets
11. Assess feature significance on a scale of 0 to 100 for selected models
12. Identify common features among the top N features from all selected models
13. If there are no common features:
a. Reduce the number of models to $N – 2$ for common feature selection
b. If still no common features, reselect N models with adjusted metrics
14. Repeat steps 8 to 13 until at least three top features are identified
15. The final selected models and top features contribute to a robust fault detection algorithm

4. Simulation results and discussion

- A Reduced dataset for Local machine processing

In order to facilitate the analysis of the Google trace dataset, which proved too extensive to be accommodated by a local machine due to its sheer size and the local machine's constrained processing capabilities, a scaled-down version of the dataset was meticulously crafted. This downsized dataset comprises a total of 10,000 records, designed to be more manageable for the local machine's limited resources.

- Imbalance in failure detection

Within this reduced dataset, there is a variable labeled "fault", which can take on the value "yes" or "no", indicating the presence or absence of a fault. Among the 10000 records, 2299 instances are classified as positive cases, signifying the presence of a fault (where "fault" is set to "yes"). Conversely, the remaining records belong to the negative class, denoting the absence of a fault (where "fault" is set to "no"). It's important to note that this dataset exhibits an imbalance in failure detection, with a considerable number of non-faulty instances compared to the faulty ones.

This downsized dataset serves as a practical solution for local machine processing, enabling efficient analysis and modeling while accounting for the inherent class imbalance in the context of failure detection.

Table 3. Output from models trained with under-sampled datasets and tested on training dataset

Model	M1U	M2U	M3U	M4U	M5U
Algorithm	c5.o	Ada – Boosted classification trees	xgboost extreme – Gradient boosting	avnNet – Model averaged neural network	bayesglm – Generalized bayesian linear model
Program name	Upsampling	Upsampling	Upsampling	Upsampling	Upsampling
Data	Training set	Training set	Training set	Training set	Training set
TP	1840	209	1828	11	1304
FN	0	1631	12	1829	536
TN	6161	1249	6068	6156	4257
FP	0	4912	93	5	1904
Accuracy	1	0.1822	0.9869	0.7708	0.695
"95% CI"	(0.9995, 1)	(0.1738, 0.1.909)	(0.9841, 0.9893)	(0.7614, 0.78)	(0.6848, 0.7051)
No information rate	0.77	0.77	0.77	0.77	0.77
P-value [Ace > NIR]	Less than 2.2×10^{-16}	1	Less than 2.2×10^{-16}	0.4428	1
Kappa	1	–0.4207	0.9635	0.0079	0.317
Macnemar's test P-value	NA	Less than 2.2×10^{-16}	5.85×10^{15}	Less than 2.2×10^{-16}	Less than 2.2×10^{-16}
Sensitivity	1	–0.11359	0.9935	0.005978	0.7087
Specificity	1	0.20273	0.9849	0.999188	0.691
Positive predictive value	1	0.04081	0.9516	0.6875	0.4065
Negative predictive value	1	0.43368	0.998	0.7709	0.8882
Prevalence	0.23	0.22997	0.23	0.229971	0.23
Detection rate	0.23	0.02612	0.2285	0.001375	0.163
Detection Prevalence	0.23	0.64004	0.240	0.002	0.4009
Balanced Accuracy	1	0.15816	0.9892	0.502583	0.6998

The presented Tables 3 and 4 offer a comprehensive evaluation of machine learning models trained on undersampled and oversampled datasets, subsequently tested on the training data. In Table 3, where undersampled data is employed, the five models (M1U-M5U) exhibit notable variations in their performance metrics. Model M1U stands out with an exceptional accuracy of 1, reflecting an impeccable ability to correctly classify data. Conversely, M2U underperforms significantly, with an accuracy of only 0.1822, indicating challenges in distinguishing between classes. The other models fall within this spectrum. These outcomes underscore the importance of the choice of machine learning algorithm and data preprocessing techniques in addressing imbalanced datasets.

Table 4. Output from models trained with over-sampled datasets and tested on training dataset

Model	M1O	M2UO	M3O	M4O	M5O
Algorithm	c5.0	Ada – Boosted classification trees	xgboost extreme – Gradient boosting	avnNet – Model averaged neural network	bayesglm – Generalized bayesian linear model
Program name	down-sampling	down-sampling	down-sampling	down-sampling	down-sampling
Data	Training set	Training set	Training set	Training set	Training set
TP	1840	193	1833	1695	1288
FN	0	1647	7	145	552
TN	5942	1244	5836	5530	4246
FP	219	4197	325	631	1915
Accuracy	0.9726	0.1796	0.9585	0.2907	0.6917
"95% CI"	(0.9688,0.976)	(0.1712, 0.1882)	(0.9539, 0.9628)	(0.2808, 0.3008)	(0.6814, 0.7018)
No information rate	0.77	0.77	0.77	0.77	0.77
P-value [Ace > NIR]	Less than 2.2×10^{-16}	1	Less than 2.2×10^{-16}	1	1
Kappa	0.9258	–0.4271	0.8895	0.0117	0.3089
Macnemar's Test P-value	Less than 2.2×10^{-16}	Less than 2.2×10^{-16}	Less than 2.2×10^{-16}	Less than 2.2×10^{-16}	Less than 2.2×10^{-16}
Sensitivity	1	0.10489	0.9962	0.9212	0.7
Specificity	0.9645	0.20192	0.9472	0.1024	0.6892
Positive predictive value	0.8936	0.03777	0.8494	0.2346	0.4021
Negative predictive value	1	0.4303	0.9988	0.8131	0.885
Prevalence	0.23	0.22997	0.23	0.23	0.23
Detection rate	0.23	0.02412	0.2291	0.2118	0.161
Detection prevalence	0.2573	0.63867	0.2697	0.903	0.4003
Balanced accuracy	0.9822	0.1534	0.9717	0.5118	0.6946

Table 4, in contrast, reveals results when models are trained with oversampled data, aimed at enhancing class balance by inflating minority class instances. The general trend indicates improved accuracy across all models compared to Table 3. Model M1O impressively attains an accuracy of 0.9726, signifying improved classification capabilities following oversampling. However, a potential trade-off becomes apparent as sensitivity and specificity metrics are somewhat compromised when compared to M1U. This suggests that oversampling may influence the models' ability to distinguish between positive and negative cases. The positive Kappa values indicate that the models consistently outperform random chance in their predictions. It is imperative to recognize that the evaluation in both tables occurs on the training dataset, a situation that can lead to over fitting. Consequently, it is vital to conduct further testing on independent datasets to assess the models' generalization capabilities. Moreover, while these tables provide valuable insights into model performance, the choice of the most suitable model should be contingent on the specific objectives and considerations of the task, as well as the implications of false positives and false negatives in the context of the application.

The provided figures, namely Figs 1 to 8, serve to elucidate the performance of various machine learning models in the context of training and testing on both under sampled and oversampled training datasets. These figures delve into multiple critical performance metrics, notably accuracy, sensitivity, positive precision, and negative

precision, shedding light on how these models handle imbalanced data. Let us embark on an in-depth elucidation of the findings:

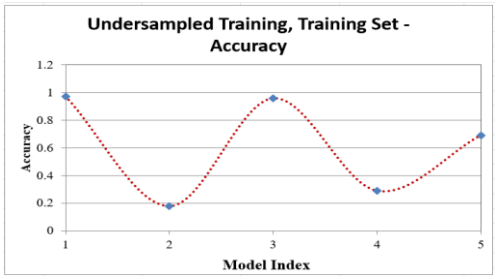


Fig. 1. Accuracy of machine learning models trained and tested with under sampled training subset

Fig. 1 showcases the accuracy of machine learning models when trained and tested on under sampled training data. It is discerned that models M1U (utilizing the C5.0 Algorithm) and M3U (leveraging the Xgboost technique) exhibit a conspicuously superior accuracy. M1U attains an extraordinary 97.26%, while M3U delivers a commendable 95.85%. These results suggest that these two models are particularly adept at handling the intricacies of the under sampled dataset. Conversely, M2U (employing Ada-boost), M4U (based on Averaged Neural Network), and M5U (utilizing Bayesian GLM) face challenges in effectively fitting the data.



Fig. 2. Accuracy of machine learning models trained and tested with over sampled training subset

Fig. 2 provides insight into the accuracy of models trained and tested on oversampled training data. Notably, M1O and M3O demonstrate superior performance, mirroring the findings in Fig. 1. M1O attains a flawless accuracy of 100%, while M3O reaches a commendable 98.69%. This implies that both undersampled and oversampled training data are conducive to the successful performance of M1 and M3 models, outshining their counterparts.

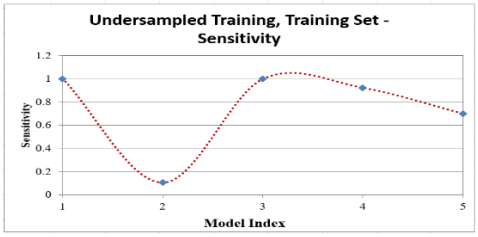


Fig. 3. Sensitivity of machine learning models trained and tested with under sampled training subset

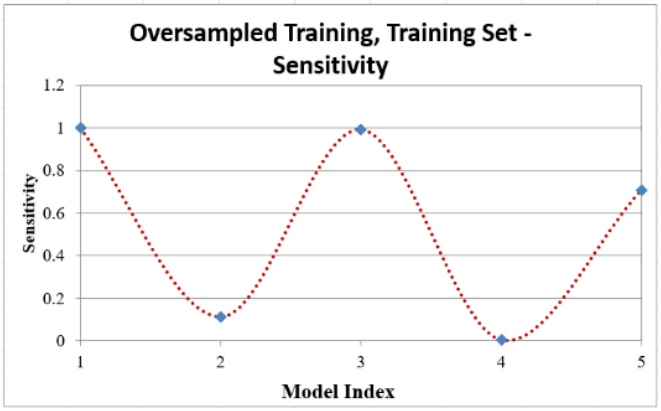


Fig. 4. Sensitivity of machine learning models trained and tested with over sampled training subset

Figs 3 and 4 scrutinize sensitivity, revealing that M1U, M3U, and M4U consistently achieve sensitivity levels exceeding 90%. It is evident that M1U and M3U outshine the other models, particularly in the context of the oversampled dataset. However, it is noteworthy that M4U experiences a decline in sensitivity when handling the oversampled dataset. In contrast, the specificity metrics presented in Fig. 3 indicate that M4U fares poorly, underscoring its proficiency in handling the positive class while grappling with the negative class.

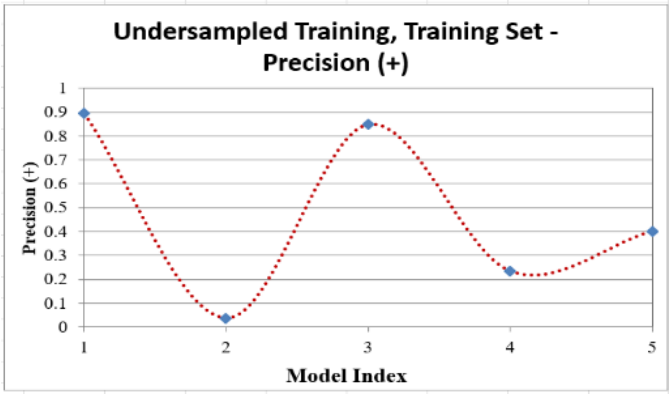


Fig. 5. Positive precision of machine learning models trained and tested with under sampled training subset

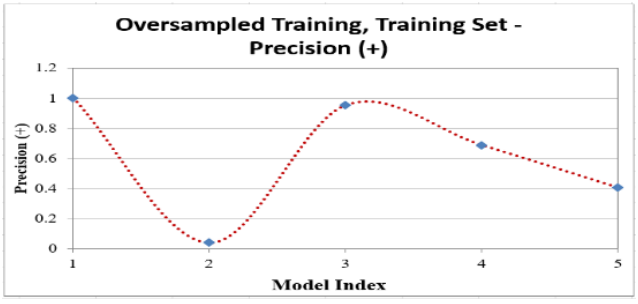


Fig. 6. Positive precision of machine learning models trained and tested with over sampled training subset

Figs 5 and 6 are devoted to positive precision for both undersampled and oversampled datasets. Models M1U and M3U consistently exhibit elevated positive precision levels. M1U attains 89.36% and a perfect 100% for undersampled and oversampled datasets, respectively. These high precision levels are partly attributed to the minimal occurrence of false positives.

Figs 7 and 8 delve into negative precision concerning undersampled and oversampled datasets. A striking observation is the exceedingly high negative precision levels, reaching 100%, for M1U, M3U, M1O, and M3O. These models excel in the accurate identification of the majority class, which is the negative class, underscoring their efficacy in navigating the intricacies of imbalanced datasets. In summation, grounded in the evaluation encompassing accuracy, sensitivity, positive precision, and negative precision, it is discerned that models M1 and M3 consistently outperform their peers across both under sampled and oversampled datasets. These models illustrate their aptitude in managing imbalanced data, particularly excelling in the accurate identification of the majority class. Nonetheless, the selection of a model for a specific task should be contingent on the precise objectives and inherent trade-offs, and it is imperative to ensure that the results generalize effectively to unseen data through rigorous validation on independent test datasets.

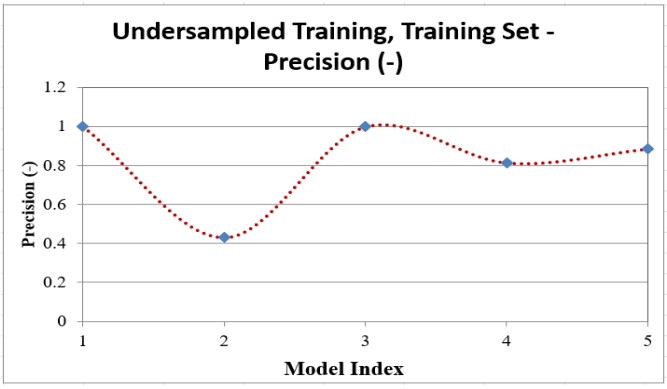


Fig. 7. Negative precision of machine learning models trained and tested with under sampled training subset

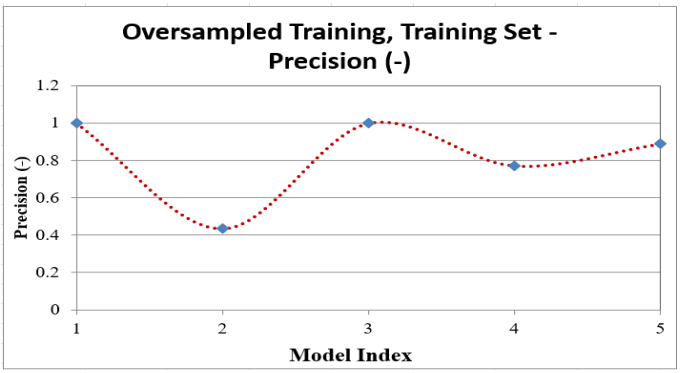


Fig. 8. Negative precision of machine learning models trained and tested with over sampled training subset

Table 5. Output from models trained with under sampled datasets and tested on testing dataset

Model	M1U	M2U	M3U	M4U	M5U
Algorithm	c5.o	Ada – Boosted classification trees	xgboost extreme – Gradient boosting	avnNet – Model averaged neural network	Bayesglm – Generalized bayesian linear model
Program name	Up sampling	Up sampling	Up sampling	Up sampling	Up sampling
Data	Testing set	Testing set	Testing set	Testing set	Testing set
TP	440	48	437	6	324
FN	19	411	22	453	135
TN	1450	313	1430	1540	1066
FP	90	1227	110	0	474
Accuracy	0.9455	0.1806	0.934	0.7734	0.6953
"95% CI"	(0.9346,0.955)	(0.164,0.1982)	(0.9222, 0.9445)	(0.7544, 0.7916)	(0.6746, 0.7155)
No information rate	0.7704	0.7704	0.7704	0.7704	0.7704
P-value [Ace > NIR]	2.20×10^{-16}	1	2.20×10^{-16}	0.3866	1
Kappa	0.8538	–0.4262	0.8251	0.02	0.3161
Macnemar's test P-Value	2.20×10^{-11}	Less than 2.20×10^{-16}	3.67×10^{-14}	Less than 2.20×10^{-16}	Less than 2.20×10^{-16}
Sensitivity	0.958	0.10458	0.9521	0.01307	0.7059
Specificity	0.9416	0.20325	0.9286	1	0.6922
Positive predictive value	0.8302	0.03765	0.7989	1	0.406
Negative predictive value	0.9871	0.43232	0.9848	0.772704	0.8876
Prevalence	0.2296	0.2296	0.2296	0.2296	0.2296
Detection Rate	0.2201	0.02401	0.2186	0.00300	0.162
Detection Prevalence	0.2651	0.63782	0.2736	0.00300	0.3992
Balanced Accuracy	0.9501	0.15391	0.4403	0.506536	0.699

Table 6. Output from models trained with over sampled datasets and tested on testing dataset

Model	M1O	M2O	M3O	M4O	M5O
Algorithm	c5.o	Ada – Boosted classification trees	xgboost extreme – Gradient boosting	avnNet – Model averaged neural network	Bayesglm – Generalized bayesian linear model
Program name	Up sampling	Up sampling	Up sampling	Up sampling	Up sampling
Data	Testing set	Testing set	Testing set	Testing set	Testing set
TP	432	61	430	6	326
FN	27	398	29	453	133
TN	1516	321	1480	1540	1073
FP	30	1219	60	0	467
Accuracy	0.9713	0.1911	0.9555	0.7736	0.6998
"95% CI"	(0.9632,0.9783)	(0.1741,0.209)	(0.9445, 0.9641)	(0.7544, 0.7916)	(0.6746, 0.7199)
No information rate	0.7704	0.7704	0.7704	0.7704	0.7704
P-value [Ace > NIR]	2.20×10^{-16}	1	2.20×10^{-16}	0.3866	1
Kappa	0.9196	–0.4262		0.02	0.3866
Macnemar's test P-value	0.7911	Less than 2.20×10^{-16}	0.001473	Less than 2.20×10^{-16}	Less than 2.20×10^{-16}
Sensitivity	0.9412	0.1329	0.9368	0.013072	0.7102
Specificity	0.9805	0.2084	0.961	1	0.6968
Positive predictive value	0.935	0.04766	0.8776	1	0.4111
Negative predictive value	0.9824	0.44645	0.9801	0.7727	0.8197
Prevalence	0.2296	0.22961	0.2295	0.229615	0.2296
Detection Rate	0.2161	0.03052	0.2151	0.003002	0.3967
Detection prevalence	0.2311	0.64032	0.245	0.003002	0.3967
Balanced accuracy	0.9605	0.1707	0.9489	0.506536	0.7035

Tables 5 and 6 provide a comprehensive evaluation of machine learning models that have undergone training on both under sampled and oversampled datasets, subsequently undergoing testing on dedicated evaluation datasets. These tables present a diverse range of performance metrics, encompassing parameters such as accuracy, sensitivity, positive precision, and negative precision. These metrics allow for an exhaustive appraisal of how these models navigate complex issues of class imbalance.

In Table 5, it is evident that models M1U and M3U emerge as the top performers, achieving notable accuracy rates of 94.55% and 93.40%, respectively. This underscores their proficiency in correctly classifying the evaluation dataset, particularly when dealing with imbalanced data. M1U displays remarkable sensitivity, highlighting its capacity to correctly identify positive cases, while M4U showcases exceptional specificity in effectively distinguishing negative cases. Noteworthy positive precision values for M1U and M3U emphasize their precision in positive predictions, complemented by consistently high negative precision across models, indicating their adeptness in identifying the majority class, typically representing the negative class.

Turning to Table 6, where models have been trained on oversampled data, M1O and M3O exhibit exceptional accuracy rates of 97.13% and 95.55%, respectively. These models shine in their ability to accurately classify the evaluation dataset, particularly in scenarios characterized by class imbalance. M1O and M3O maintain high sensitivity levels, underscoring their effectiveness in accurately identifying positive cases. Notably, M4O excels in specificity, achieving a 100% accuracy in distinguishing negative cases. Positive precision values for M1O and M3O are commendable, further attesting to their precision in positive predictions. The persistent high negative precision values in these models demonstrate their proficiency in identifying the majority class. This comprehensive array of performance metrics reinforces the necessity of selecting the most appropriate model contingent upon the specific task, while underscoring the importance of rigorous validation on independent test datasets to ascertain the models' capacity for generalization.

Figs 9 to 16 provide a detailed analysis of the performance of machine learning models denoted as M1U, M2U, M3U, M4U, and M5U for the undersampled dataset, as well as M1O, M2O, M3O, M4O, and M5O for the oversampled dataset. These models were tested on a separate testing subset, and its worth noting that this testing set was not utilized during the training of any models, serving as a hold-out set for unbiased evaluation.

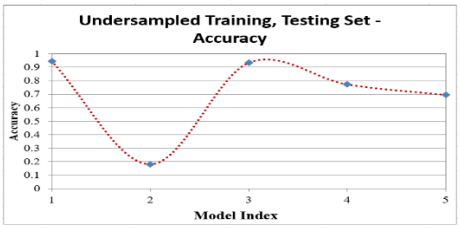


Fig. 9. Accuracy of machine learning models trained with under sampled dataset and tested with testing subset

Fig. 9 centers on the accuracy of machine learning models trained with the undersampled dataset and subsequently tested with the testing subset. It becomes evident that M1U and M3U outshine their counterparts, displaying significantly higher accuracy, with M1U achieving 94.55% and M3U achieving 93.40%. Notably, M4U also delivers good performance, with an accuracy of approximately 78%, while the other models demonstrate a less robust fit to the undersampled data.

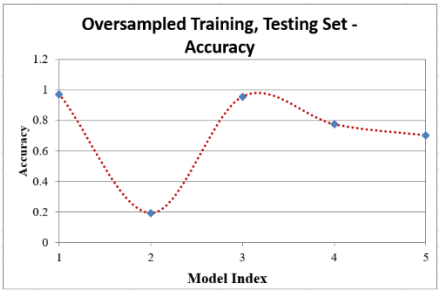


Fig. 10. Accuracy of machine learning models trained with over sampled dataset and tested with testing subset

Fig. 10 extends the evaluation to models trained with the oversampled dataset and tested with the testing subset. Here, M1O takes the lead with an accuracy of 97.15%, closely followed by M3O at 95.55%. M4O also demonstrates commendable performance, achieving approximately 78% accuracy. These findings highlight the consistent accuracy of models M1 and M3 when confronted with both undersampled and oversampled datasets and when tested with both training and testing datasets.

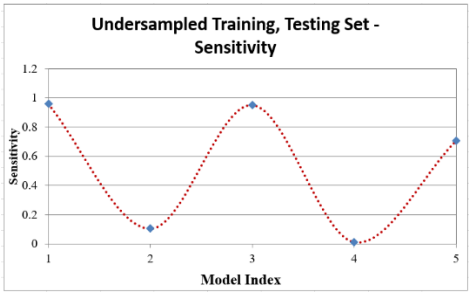


Fig. 11. Sensitivity of machine learning models trained with under sampled dataset and tested with testing subset

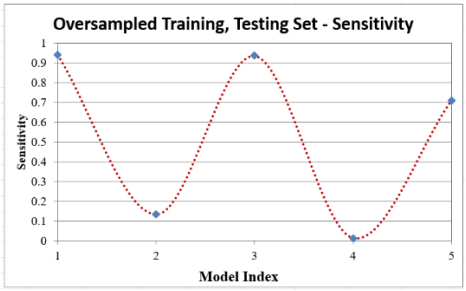


Fig. 12. Sensitivity of machine learning models trained with over sampled dataset and tested with testing subset

Figs 11 and 12 delve into sensitivity, revealing that M1U, M3U, M1O, and M3O consistently maintain sensitivity levels exceeding 90%, indicating their effectiveness in correctly identifying positive cases. However, M4U and M4O show less robust sensitivity performance, particularly when tested with the undersampled dataset.

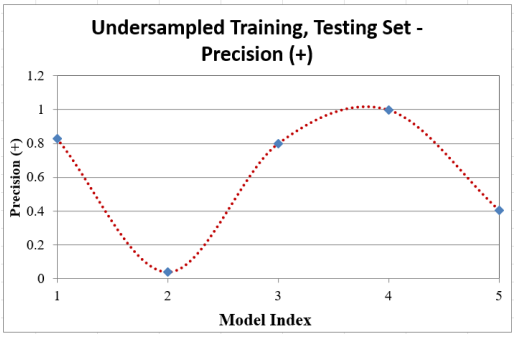


Fig. 13. Positive precision of machine learning models trained with under sampled dataset and tested with testing subset

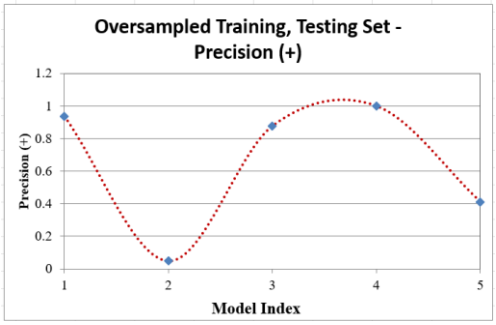


Fig. 14. Positive precision of machine learning models trained with over sampled dataset and tested with testing subset

Figs 13 and 14 depict positive precision values for models trained and tested with both undersampled and oversampled training subsets. Notably, M4U and M4O reach 100% positive precision. Meanwhile, sensitivity for M1U and M3U shows a slight dip to around 80%, while M1O and M3O maintain sensitivity above 90%.

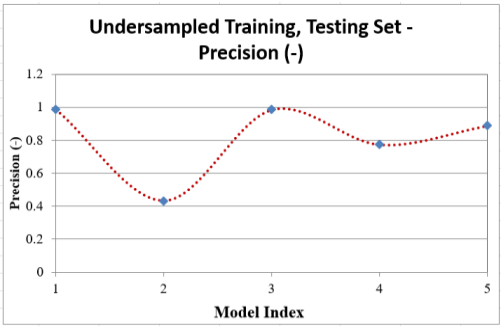


Fig. 15. Negative precision of machine learning models trained with under sampled dataset and tested with testing subset

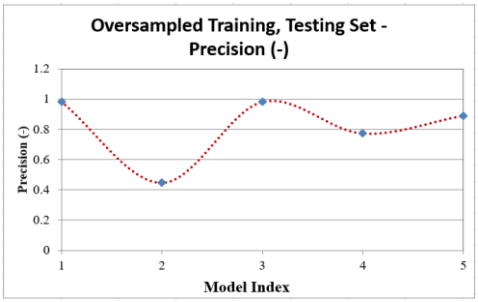


Fig. 16. Negative precision of machine learning models trained with over sampled dataset and tested with testing subset

Figs. 15 and 16 illustrate negative precision values for models trained and tested with both under sampled and oversampled training subsets. Interestingly, positive precision reaches 100% for M4U and M4O. Sensitivity for M1U and M3U remains near 100%, while M1O and M3O display sensitivity levels above 80%.

These comprehensive results collectively demonstrate the robustness of models M1 and M3 across both undersampled and oversampled datasets, further emphasizing their consistent performance. However, the specific choice of model should align with the specific task requirements, and rigorous validation on independent test datasets is crucial to ascertain the generalizability of the findings.

The top 10 features extracted from models M1 and M3 shed light on the crucial factors influencing their predictions. Interestingly, a substantial overlap exists between these two sets of top features. Specifically, seven features are common to both models, underscoring their shared significance in predicting the performance of cloud systems and enhancing fault tolerance.These common features encompass aspects such as “cycles_per_instruction”, which relates to the efficiency of resource utilization, “cpu_usage_distribution1”, reflecting the distribution of CPU usage, “start_time” and “average_usage_memory”, which are essential in understanding the temporal and memory-related dynamics, “random_sample_usage_cpus”, indicating unpredictability in CPU utilization, “vertical_scaling”, a vital scalability metric, and “priority”, denoting the relative importance of tasks or processes within the system. The presence of these shared features suggests that either the C5.0 or XGBoost model can be effectively employed to predict cloud performance and bolster its reliability. By closely monitoring these seven parameters and implementing timely interventions, cloud administrators can proactively address issues, optimize resource allocation, and enhance the overall dependability of cloud systems, ensuring smooth operations and robust fault tolerance.

5. Conclusion

In conclusion, this research comprehensively evaluates the performance of machine learning models using a scaled-down version of Google trace data, addressing the computational limitations of local machines. Two key datasets, undersampled and oversampled, have been created to mitigate label imbalance. Five distinct algorithms, namely C5.0, Ada-Boost, XGBoost, Average Neural Network, and Bayesian GLM,

have been trained with these datasets, resulting in ten different models. Rigorous testing on both training and testing datasets has been conducted to gauge the models' performance across various metrics. The findings highlight the consistent and superior performance of models M1U (C5.0 Algorithm) and M3U (XGBoost) in terms of accuracy, sensitivity, positive precision, and negative precision. Notably, these models exhibited performance exceeding 90% in most cases, making them stand out. Furthermore, seven common features, including "cycles_per_instruction", "cpu_usage_distribution1", "start_time", "average_usage_memory", "random_sample_usage_cpus", "vertical_scaling", and "priority", emerged as critical indicators for predicting cloud performance and enhancing fault tolerance.

As future avenues, this research suggests evaluating these models on larger datasets, exploring alternative algorithms, enhancing feature engineering and selection, and validating the models externally. Real-time applications in cloud monitoring and dynamic model updates to accommodate evolving cloud environments are also recommended. Ensemble methods could be explored to enhance predictive accuracy. In summary, this study offers insights into cloud system performance prediction and fault tolerance, with potential real-world applications and avenues for continued research in cloud computing.

5. References

1. Abd Elfattah, E., M. Elkawkagy, A. El Sisi. A Reactive Fault Tolerance Approach for Cloud Computing. – In: Proc. of 13th International IEEE Computer Engineering Conference (ICENCO'17), 2017, pp. 190-194.
2. Hasan, M., M. S. Goraya. Priority Based Cooperative Computing in Cloud Using Task Backfilling. – Lect. Notes Software. Eng., Vol. **4**, 2016, pp. 229-233. **http://dx.doi.org/10.18178/nse.2016.4.3.255**
3. Kochhar, D., A. K. J. Hilda. An Approach for Fault Tolerance in Cloud Computing Using Machine Learning Technique. – Int. J. Pure Appl. Math., Vol. **117**, 2017, No 22, pp. 345-351.
4. Gupta, S., B. B. Gupta. XSS-Secure as a Service for the Platforms of Online Social Network-Based Multimedia Web Applications in the Cloud. – Multimedia Tools Appl., Vol. **77**, 2018, No 4, pp. 4829-4861.
5. Tebaa, M., S. El Hajji. From Single to Multi-Clouds Computing Privacy and Fault Tolerance. – In: Proc. of International Conference on Future Information Engineering, Elsevier B. V., 2014, pp. 112-118. **http://dx.doi.org/10.1016/j.ieri.2014.09.099**
6. Abid, A., M. T. Khemakhem, S. Marzouk, M. Bem Jemaa, T. Monteil, K. Drira. Toward Ant Fragile Cloud Computing Infrastructures. – Procedia Compute. Sci., Vol. **32**, 2014, pp. 850-855. **http://dx.doi.org/10.1016/j.procs.2014.05.501**
7. Lin, X., A. Mamat, Y. Lu, J. Deogun, S. Goddard. Real-Time Scheduling of Divisible Loads in Cluster Computing Environments. – Parallel Distributed. Computing, Vol. **70**, 2010, pp. 296-308. **http://dx.doi.org/10.1016/j.jpdc.2009.11.009**
8. Jhawar, R., V. Piuri. Fault Tolerance and Resilience in Cloud Computing Environments. – In: J. Vacca, Ed. Computer and Information Security Handbook. 2013, pp. 1-29. **http://dx.doi.org/10.1109/CLOUD.2011.16**
9. Sun, D., G. Chang, C. Miao, X. Wang. Modelling and Evaluating a High Serviceability Fault Tolerance Strategy in Cloud Computing Environments. – Int. J. Security Network, Vol. **7**, 2012, pp. 196-210. **http://dx.doi.org/10.1504/IJSN.2012.053458**

10. T c h e r n y k h, A., U. S c h w i e g e l s o h n, V. A l e x a n d r o v, E. T a l b i. Towards Understanding Uncertainty in Cloud Computing Resource Provisioning. – In: Proc. of International Conference on Computational Science, 2015, pp. 1772-1781.
http://dx.doi.org/10.1016/j.procs.2015.05.387
11. W a n g, T., W. Z h a n g, C. Y e, J. W e i, H. Z h o n g, T. H u a n g. FD4C: Automatic Fault Diagnosis Framework for Web Applications in Cloud Computing. – IEEE Trans. Syst. Man Cyber Network. Syst., Vol. **46**, 2016, pp. 61-75.
http://dx.doi.org/10.1109/TSMC.2015.2430834
12. A h m e d, W., Y. W. W u. A Survey on Reliability in Distributed Systems. – J. Computer and Syst. Sci., Vol. **79**, 2013, pp. 1243-1255.
http://dx.doi.org/10.1016/j.jcss.2013.02.006
13. H e r n á n d e z, S., J. F a b r a, P. Á l v a r e z, J. E z p e l e t a. Using Cloud-Based Resources to Improve Availability and Reliability in a Scientific Workflow Execution Framework. – In: Proc. of 4th International Conference on Cloud Computing, GRIDs and Virtualization, 2013, pp. 230-237.
14. C h e r a g h l o u, M. N., A. K h a d e m-Z a d e h, M. H a g h p a r a s t. A Survey of Fault Tolerance Architecture in Cloud Computing. – J. Network. Compute. Appl., Vol. **61**, 2016, pp. 81-92.
http:// dx.doi.org/10.1016/j.jnca.2015.10.004
15. P r a t h i b a, S., S. S o w v a r n i c a. Survey of Failures and Fault Tolerance in Cloud. – In: Proc. of 2nd International Conference on Computer Communications Technologies (ICCCT'17), 2017, pp. 169-172.
16. Z h a n g, J., Y. J i a, Y. Y u. Intelligent Resource Management for Fault Tolerance in Cloud Computing: A Survey. – Journal of Network and Computer Applications, Vol. **132**, 2019, pp. 38-52.
17. G a o, J., H. W a n g, H. S h e n. Machine Learning Based Workload Prediction in Cloud Computing. – In: Proc. of 29th International Conference on Computer Communications and Networks (ICCCN'20). IEEE, 2020, Los Alamitos, pp. 1-9.
18. R o d r i g u e z, G. G., J. M o r r i s o n. A Fault Tolerance Technique for Containers in the Cloud. – Journal of Cloud Computing, Vol. **9**, 2020, No 1, pp. 1-18.
19. A b d u l l a h, S. M., M. M. H a s a n, A. A l z a h r n i. A Dynamic Replication Scheme for Fault Tolerance in Cloud Computing. – International Journal of Grid and High Performance Computing, Vol. **12**, 2020, No 1, pp. 1-21.
20. A l m u k h a i z i m, S. H. S., M. O t h m a n. Fault-Tolerant Resource Management in Distributed Cloud Systems: A Survey. – Journal of Grid Computing, Vol. **18**, 2020, No 1, pp. 71-98.
21. N i g a m, S. S., P. P a t n a i k, A. K. M a n d a l. Towards a Comprehensive Framework for Fault-Tolerant Containerized – Micro Services in the Cloud. – Journal of Cloud Computing: Advances, Systems and Applications, Vol. **9**, 2020, No 1, pp. 1-26.
22. A l o m a r i, F., M. Z. I s l a m. Fault-Tolerant Resource Management in Cloud Computing: A Systematic Review. – International Journal of Distributed Systems and Technologies, Vol. **12**, 2021, No 1, pp. 44-62.
23. A l h a d d a d, S., M. Z. I s l a m. Cloud-Based Service Availability Prediction Using Machine Learning Techniques. – Journal of Cloud Computing, Vol. **9**, 2020, No 1, p. 17.
24. G a n i, M. A., S. U l l a h, S. U. K h a n. A Fault-Tolerant Cloud-Based Architecture for IoT Applications. – Journal of Grid Computing, Vol. **18**, 2020, No 2, pp. 213-227.
25. Q u a m a r, N., A. B. M. A. A. I s l a m. Efficient Fault-Tolerant Resource Allocation in Edge Computing. – International Journal of Computer Networks and Communications Security, Vol. **8**, 2020, No 3, pp. 44-52.
26. T h a n g a m, S., E. K i r u b a k a r a n, J. W i l l i a m. Architecture for Service Selection Based on Consumer Feedback (FBSR) in Service Oriented Architecture Environment. – International Information Institute (Tokyo). Information, 2014, pp. 282-286.
27. P a n w a r, R., M. S u p r i y a. Dynamic Resource Provisioning for Service-Based Cloud Applications: A Bayesian Learning Approach. – Journal of Parallel and Distributed Computing, Vol. **168**, 2022, Issue October 2022, pp. 90-107.
https://doi.org/10.1016/j.jpdc.2022.06.001

28. P r a k a s h, P., R. S u r e s h, P. N. D h i n e s h K u m a r. Smart City Video Surveillance Using Fog Computing. – International Journal of Enterprise Network Management, Vol. **10**, March 2019, pp. 389-399. DOI: 10.1504/IJENM(2019).
29. P r a k a s h, P., K. G. D a r s h a u n, P. Y a a z h l e n e, M. V. G a n e s h, V. V a s u d a. Fog Computing: Issues, Challenges and Future Directions. – International Journal of Electrical and Computer Engineering (IJECE), Vol. **7**, December 2017, No 6, pp 3669-3673. **https://DOI:10.11591/ijece.v7i6.pp3669-3673**
30. S i n g h, B. S., M. P r a t a p, D. K. S a n g e e t a. Hardware Setup for VLC Based Vehicle to Vehicle Communication under Fog Weather Condition. – International Journal of Advanced Science and Technology, Vol. **29**, 2020, No 3s.
31. D e e p i k a, T., P. P r a k a s h. Power Consumption Prediction in Cloud Data Center Using Machine Learning. – International Journal of Electrical and Computer Engineering, 2020, pp. 1524-1532. **http://doi.org/10.11591/ijece.v10i2**
32. S a n d e e p, H. R., S. T h a n g a m. A Hybrid Cloud Approach for Efficient Data Storage and Security. – In: Proc. of 6th International Conference on Communication and Electronics Systems (ICCES'22), 2022.
33. I y e r, G. N. Evolutionary Games for Cloud, Fog and Edge Computing – A Comprehensive Study. – Advances in Intelligent Systems and Computing, Vol. **990**, 2020, pp. 299-309. **http://doi.org/:10.1007/978-981-13-8676-3_27**
34. Y e h i a, I., A. A. A l j a a f r e h. Block Chain-Fog Computing Integration Applications. – Cybernetics and Information Technologies, Vol. **23**, 2023, No 1, pp. 3-37.
35. P e t r o s y a n, D., H. A s t s a t r y a n. Serverless High-Performance Computing over Cloud. – Cybernetics and Information Technologies, Vol. **22**, 2022, No 3, pp. 82-92.

Received: 13.07.2023; Second Version:15.10.2023; Third Verion: 26.10.2023; Accepted: 07.11.2023