CSC8199 Project and Dissertation: Anomaly Prediction and Diagnosis for Microservices Using Machine Learning (ML) on Cloud Monitoring Data

Cynthia Dorothy George SandhanaKumar

MSc in Cloud Computing,
School of Computing Science, Newcastle University.
C.D.George-Sandhanakumar2@newcastle.ac.uk

Abstract. Microservices-based applications can be developed and deployed in a cloud environment much more quickly using emerging container technologies such as Docker. Application providers are increasingly concerned about the dependability of these microservices. Anomaly detection techniques can detect anomalous behaviours that may lead to unexpected failures. To meet the high service quality requirements, critical services in the field of Virtualization require elaborate reliability and high availability mechanisms. Traditional monitoring systems detect overload and outages and automatically scale out services or mask faults. However, faults are frequently preceded by anomalies and failures of the services, which are missed when only outages are detected. By analysing metrics collected from all layers and components of the cloud infrastructure, we propose to use machine learning techniques to detect abnormal behaviour of services and hosts. An offline evaluation of data from anomaly injection experiments reveals that the different models have very high accuracy, precision and recall values.

Declaration: I declare that this dissertation represents my own work except where otherwise explicitly stated.

1. Introduction

Cloud Computing [1] has significantly grown in popularity as a model for delivering services through the Internet. This is due to various of technical factors, such as increased energy efficiency, better usage of hardware and software resources, elasticity, performance isolation, flexibility, and an on-demand service architecture [2]. For both technical and financial reasons, Cloud Computing is now commonly employed to deliver services over the Internet. In recent years, the number of Cloud-based services has expanded swiftly and significantly, as has the complexity of the infrastructures supporting these services. As it is a vitally growing paradigm effective and efficient monitoring is required to properly operate and manage such complex infrastructures through which the complete use of cloud can be experienced. To properly administer these systems and manage their rising complexity, precise and fine-grained monitoring operations are required [3]. Commercial datacentres and colocation facilities, as well as hyperscale datacentres run by firms like Amazon, Apple, Google, Microsoft, and Facebook, house cloud computing resources. To maintain stable operation, these datacentres must be constantly monitored. Various monitoring solutions are available to monitor various layers of a corporate IT system, including business operations, applications, and infrastructure. The system administrators, on the other hand are in charge of deciding what to monitor, when to monitor it, and where to monitor it. As a result, the intuition-driven monitoring technique is informal, of varying quality, and not scalable to big systems. Furthermore, this strategy falls behind the rapid evolution of enterprise systems, particularly in hybrid cloud computing settings that combine public and private datacentres [4]. When talking about cloud platforms, Microservices have risen to prominence as a new strategy to designing and delivering cloud applications that require greater agility, scale, and reliability.

Microservices are being used in the development of more and more Web applications to improve scalability, flexibility, and reliability. A microservice application is made up of a collection of services that are isolated, scalable, and fail-safe. Each service can be thought of as its own application, with endpoints exposed for communication with other services. There are numerous advantages to using a microservice architecture. Software, for example, can be released more quickly, and teams can be smaller and more focused on their own work. The following virtualization techniques are widely used to generate enough isolated resources for such a large number of services. Traditional methods of virtualization include virtual machines (VMs). Each VM that is created has its own operating system (OS). Containers are another emerging virtualization technology that is gaining popularity over VMs due to their lightweight, high performance, and scalability. And the containers that are created share the same host operating system. [6]. A microservice-based cloud application architecture promotes the deconstruction of monolithic application components into independent software components known as "microservices" to achieve this goal. Because the microservices can be designed, deployed, and updated independently of one another, complicated run-time performance monitoring and management issues arise [5]. The discussion above on cloud computing, hybrid cloud, microservices paved a way to dig more into cloud monitoring on mentioned areas especially in hybrid cloud as it has number of challenges taken into consideration. In recent years, the advancement of virtualization technologies, particularly container technology, has contributed to the widespread adoption of microservice architecture. As a result, service providers begin to place greater emphasis on the dependability of these microservices. Service Level Agreements (SLAs) are typically entered into between service providers and users to specify the quality of the services provided. They could include things like performance requirements and dependability properties. A breach of such SLAs may result in severe consequences [7].

Traditional reliability approaches use active or passive monitoring techniques such as heartbeats or status requests to monitor system components for outages. These mechanisms detect flaws only after they have occurred. Faults, on the other hand, are frequently preceded by abnormal system behaviour or degraded system states, which can only be seen in fine-grained monitoring data obtained within the affected host. Detecting and handling such anomalies can help to prevent faults from occurring and, in general, increase system throughput. Machine learning algorithms enable us to learn from past observations and apply this knowledge to make predictions about the future. When applied to anomaly detection, this means first observing a host while it is experiencing anomalies as well as during normal operation. The observed data is then converted to a model representing the host's normal and abnormal behaviour using a machine learning algorithm. By observing the host, the resulting model can now estimate whether it is acting normally [8].

Anomaly detection can aid in the identification of unusual patterns that do not conform to expected patterns. Because anomaly detection necessitates a large amount of historical data, service providers must install numerous monitoring tools on their infrastructure to collect real-time service performance data. [9].

These microservice providers are currently facing two major challenges. To begin, what metrics should be monitored for container-based microservices? Second, even if all of the metrics are collected, how do you determine whether the application's behaviours are abnormal or not? In this paper, an anomaly detection system is proposed that can effectively address these two major challenges.

In this research, classification techniques are used to identify abnormalities on a set of hosts are evaluated. A cloud testbed running a Book application scenario with three separate microservices is the foundation for the evaluation. Data monitoring and experimentation with controlled fault injection provide the information needed to train and assess machine learning systems. Some algorithms obtain an extraordinarily high average F1-measure, according to the validation. The concept of applying machine learning for anomaly detection can be applied to productive systems because these results imply that the monitoring data does, in fact, include insights about anomalous host behaviour.

1.1 Aim

The aim of the project is to develop a various machine learning models which analyse different parameters such as CPU, memory, response time and throughput from all microservices available in different clouds such as Hybrid and multi-Cloud. The proposed outcome of this project is to alert when parameters in web, application and DB level is not right in hybrid cloud here multiple microservices are deployed in different virtual machines.

1.2 Objectives

The goal has been divided into many objectives to adequately qualify that aim has been accomplished.

Research into Cloud monitoring system, machine learning based anomaly detection and identify models:
A thorough research on existing cloud monitoring system and understanding on the dataset from the monitoring agent before the implementation and application. An essential part of this research is to identify a range of machine learning models.
Identify functional requirements for the implementation:
A list of functional requirements will be identified for achieving the objectives of the project.
Develop models for training the monitoring data to detect the anomalies:
The models aim to train and test the dataset dividing them to 80% and 20% respectively.
Evaluate the machine learning model:
The model will be evaluated against the accuracy, precision, recall and F1 score of the actual and test data.
Determine the system's success and suggest how it might be improved

1.3 Structure of Dissertation

The structure of the rest of this dissertation is as follows:

• **Section 2. Background and Related Work**- This section dives deep into key aspects of the system. The goal is to identify problems that the system can solve, to investigate best practices, and to analyse existing systems to help guide development.
• **Section 3. Experimental Setup for Data Collection-** This section talks about how the experiments conducted to collect the monitoring data of various microservices under different scenarios for the anomaly detection
• **Section 4. Implementation-** This is a phased section, with each phase representing a different design period. Each phase builds on the previous one, forming a chronological development story that highlights key challenges and changes along the way.
• **Section 5. Evaluation and Results-** This section examines how the final system performed, the findings of usability studies, and how components could be improved.
• **Section 6. Conclusion and Future Work-** This section highlights the completed work, revisits the goal and objectives, and suggests future work that could be used to improve the situation.

2. Background and Related Work

This section's goal is to introduce the reader to the key research areas for implementing the anomaly detection methods. It first introduces Microservices, Containers, hybrid cloud before delving into

Machine learning definition, types, algorithms, and models, anomaly detection. Finally, it goes over existing anomaly detection work.

2.1 Background

Microservice architecture is a cloud application design pattern that moves complexity away from traditional monolithic applications and into the infrastructure [10]. In contrast to a monolithic system, microservices architecture builds a system out of a collection of small services, each of which is isolated, scalable, and resilient to failure. Language-independent application programming interfaces are used by services to communicate across a network (API)

Containers are lightweight OS-level virtualizations that allow us to run an application and its dependencies in a separate process with limited resources. Each component runs in its own environment and does not share the host operating system's (OS) memory, CPU, or disc. With more and more applications and services being deployed on cloud-hosted environments, microservice architecture is becoming increasingly reliant on container technology [11][9].

The use of cloud services from more than one cloud vendor is referred to as Multicloud (as in Figure 1). It could be as simple as utilising software-as-a-service (SaaS) from various cloud vendors, such as Salesforce and Workday. However, in the enterprise, multicloud typically refers to running enterprise applications on multiple cloud service providers' platform-as-a-service (PaaS) or infrastructure-as-a-service (IaaS), such as Amazon Web Services (AWS), Google Cloud Platform, IBM Cloud, and Microsoft Azure. A multicloud solution is a cloud computing solution that is portable across the cloud infrastructures of multiple cloud providers. Multicloud solutions are usually built on open-source, cloud-native technologies like Kubernetes, which are supported by all public cloud providers [23].

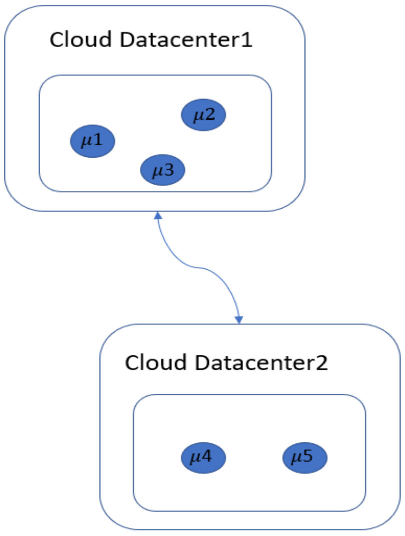


Fig. 1. This shows a figure consisting of different microservices deployed in different clouds. The arrow represents the connection between the microservices which belongs to same application.

2.1.1 Machine Learning

Machine learning (ML) is a component of artificial intelligence (AI) that permit software applications to become more accurate at predicting outcomes without explicitly programming them to do so. Machine learning algorithms predict new output values using this historical data as input. Machine learning is important because it provides businesses with insights into trends in customer behaviour and business operational patterns, as well as assisting in the development of new products. Machine learning is central to the operations of many of today's leading companies, including Facebook, Google, and Uber. For many businesses, machine learning has become a considerable competitive differentiator.

The study of the analysis and creation of algorithms that can learn from and predict data is known as machine learning. ML has proven useful because it can solve problems at a speed and scale that the human mind cannot match. Machines can be trained to identify patterns in and relationships between input data and automate routine processes using massive amounts of computational power behind a single task or multiple specific tasks.

Data Is Key: Machine learning success is dependent on the algorithms that power it. Without being explicitly programmed, ML algorithms create a mathematical model based on sample data, known as "training data," to make predictions or decisions. This can reveal data trends that businesses can use to improve decision-making, optimise efficiency, and collect actionable data at scale.

*AI is the Goal***:** ML lays the groundwork for AI systems that automate processes and solve data-driven business problems autonomously. It allows businesses to replace or supplement certain human capabilities. Chatbots, self-driving cars, and speech recognition are examples of common machine learning applications.

Algorithms, like AI, are fundamental tools for solving ML problems. To put it another way, ML algorithms are at the heart of every ML system. ML systems can learn and perform a task without the need for human intervention. Figure 2 depicts how a prediction task in ML is carried out [12]. The process depicted clearly shows that a desired prediction is realised by analysing inputs and corresponding observations.

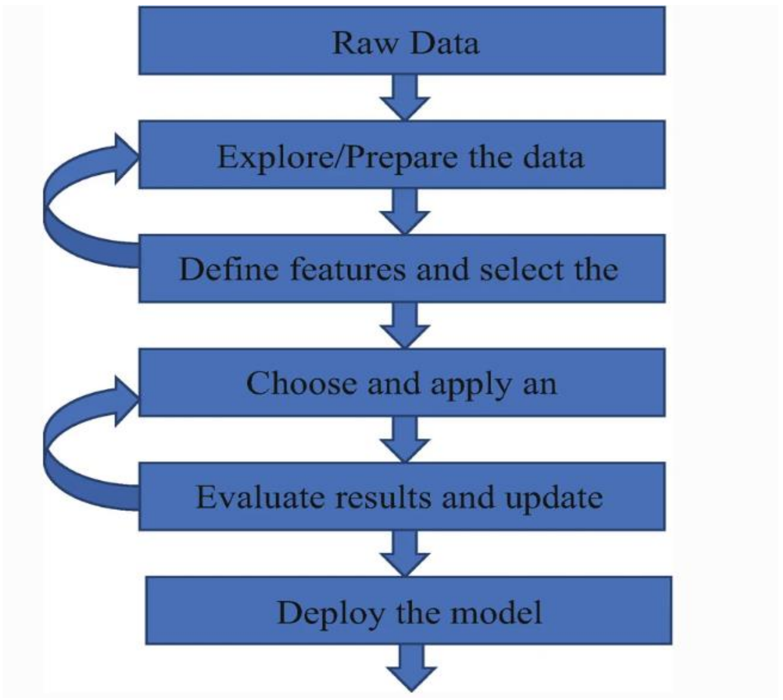


Fig. 2. This figure shows the prediction process in machine learning.

There are two types of ML algorithms: supervised and unsupervised. details about supervised and unsupervised machine learning algorithms for developing ML models and systems are discussed in the subsequent sessions. These ML algorithms are ideal for creating classification and prediction models.

Supervised Machine Learning Algorithms: In this method, a model is trained to predict the results of a new set of user inputs by using an old set of inputs and its known set of outputs. The sought-after output is also known as a target or a label. Essentially, the system learns to predict from prior experience. A set of input and target pairs constitutes training data. A data scientist teaches the system by identifying the features and variables that should be examined. Following training, these models compare the new results to the old ones and adjust their weight data to improve future prediction accuracy.

- Regression
- Classification

Regression: A numerical value or continuous variable is predicted using regression based on the relationship between predictors (input variables) and output variables. Predicting the price of a house based on current prices in the neighbourhood, school district, total area, number of rooms, locality, and crime rate is one example.

Categorization: A categorical variable is predicted in classification, i.e., input data can be categorised based on labels. An email classification problem, for example, is determining whether an email is spam or not.

In summary, the regression technique should be used when quantifying predictable data, and the classification technique should be used when predicting a label from predictable data.

Unsupervised Machine Learning Algorithms: This method does not involve training the model with old data, so there is no "teacher" or "supervisor" to assist in creating a model with previous examples. The system is not trained by supplying a set of inputs and outputs. Instead, based on its own observations, the model will learn and predict the output. Consider a basket of apples and bananas that have not been labelled or given any specifications at this time. Only by comparing colour, size, and shape will the model learn and organise them. This is accomplished by looking for specific features and similarities between them.

The following are the techniques used in unsupervised learning:

- Clustering
- Dimensionality reduction
- Anomaly detection

Clustering: Clustering is a method of dividing or grouping data into clusters based on similarities observed. Data is examined in order to form meaningful groups or subsets. For example, books in a library are assigned to a specific cluster based on classification indices.

Dimension Reduction: When a dataset contains an excessive number of features, the data separation process becomes more complicated. Dimensionality reduction is a technique used to solve complex scenarios. The basic idea is to reduce the number of variables or features in a dataset without sacrificing its basic classification or prediction capability. Image classification is the best example of where this technique is commonly used, focusing on key attributes such as eyes, nose, and face shape.

Anomaly Detection: Anomaly is defined as a deviation from the norm, such as bank fraud versus regular transactions. Anomaly detection is the detection of such problems by using events or observations that differ significantly from the majority of the data. Examples of applications include detecting structural defects in manufacturing and medical problems such as cancer detection [14].

2.1.2 Clustering-Based Discretization for Supervised Learning

Discretization is the mapping of a continuous variable into discrete space, and it is frequently used in sociological data analysis to aid comprehension by grouping multiple values of a continuous variable and partitioning the continuous domain into non-overlapping intervals. The transformation of a continuous variable, such as height (of a human), from a numerical measure (for example, 6′3″), into tall/short/medium categories is one example (as in Figure 3).

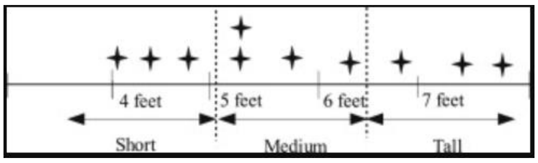


Fig. 3. This figure shows the example of discretization process where the continuous values got discretised.

Discretization facilitates the application of several machine learning algorithms to problems with continuous space attributes, such as frequent item-set discovery applications. Furthermore, the need for continuous variable discretization in machine learning arises because some machine learning algorithms are designed to work best with discrete variables. Discretization methods are classified in several ways, including splitting versus merging, global versus local, supervised versus unsupervised, and static versus dynamic [13].

2.1.3 Clustering for Unsupervised Learning

K-Means Clustering Algorithm: Sometimes the objective is to assign labels based on the features without any prior knowledge of the labels. By establishing clusters and giving each cluster a label, this objective is accomplished. When it's necessary to separate a huge set of users into groups according to recognisable characteristics, clustering might be utilised as an example. In other words, we look for homogenous subgroups within the data, where each cluster has as similar a set of data points as possible. The most common similarity metric is the Euclidean-based distance or correlation-based distance. Choosing the similarity metric to employ depends on the application.

One of the more well-known and widely used algorithms is definitely K-means. Based on the dataset and the number of clusters the user defines (expressed by the variable K), it employs an iterative refinement process to achieve its final grouping. For instance, the dataset will be divided into three clusters if K is set to 3.

A general-purpose algorithm called K-means creates clusters based on the geometric separations between points. Because it is a centroid-based technique, points are classified into clusters based on their distances (which are primarily Euclidean distances) from the centroid. The most popular centroid-based clustering algorithm is this one. Centroid-based algorithms are effective but sensitive to initial conditions and outliers. An effective, efficient, and straightforward clustering algorithm is K-means.

A non-deterministic iterative approach is K-means. In order to arrive at a final clustering of the data points, K-means repeatedly calculates new means starting with arbitrarily selected data points as suggested means of the data groups. The clusters are globular and of comparable size because they are organised around centroids. A centroid reflecting the cluster's centre is often defined as the mean. The centroid may or may not belong to the dataset.

Rerunning this method with different randomised starting centroids might result in a better performance because the original result of running it might not be the best possible output (different initial objects may produce different clustering results). Due to this, it is customary to run the algorithm numerous times, using various starting positions, and to test various initiation strategies. Then again, how does one determine the right value for K or the number of centroids to make? There isn't a single solution. Different methods exist to attempt to estimate the ideal number of centroids or clusters, despite the fact that it is unknown from the start. Testing various cluster counts and measuring the resulting sum of squared errors is a frequent strategy. The K value is then selected so that an increase in the number of clusters causes a very little decrease in the error total while a decrease causes a significant increase in the error sum. The elbow point, which designates the ideal number of clusters, can be utilised as a visual indicator to determine the ideal choice for the value of K.

Data visualisation is typically employed to assess results because clustering is unsupervised (i.e., there is no "correct answer"). Classification algorithms are often more suited if there is a "correct answer" (i.e., we have pre-labelled groups in the training data).

K-means can be used to analyse continuous, quantitative data that has fewer dimensions. It has been effectively used, for instance, to cluster documents, identify crime hotspots, segment customers, find insurance fraud, analyse data from public transportation, cluster IT alerts, and more.

Assumptions:

- The spherical variance of each attribute's (variable) distribution is a presumption made by K-means.
- The variance is the same for each component variable.
- All K clusters have an identical prior probability, meaning that there are nearly an equal number of observations in each cluster.

K-Means Algorithm: The K-means algorithm operates as follows given K (the number of clusters)

1. Select K data points at random to serve as the initial centroids and cluster centres.
2. Add the squared distances between each data point and each centroid.
3. Assign the nearest centroid to each data point.
4. Take the average of all the data points in each cluster to recalculate the centroids for the clusters.
5. Move on to 3 if a convergence requirement is not satisfied.

Convergence Criterion: In the initial few repetitions, the majority of the convergence occurs. Basically, the K-means algorithm can be stopped by adopting one of three stopping criteria:

- Newly created clusters' centroids remain unchanged.
- The same cluster of points holds true.
- There are no more iterations possible.

If the centroids of freshly generated clusters do not change, the process can be stopped. The algorithm is not picking up any new patterns even after many iterations, and it is time to cease training if we are still obtaining the same centroids for every cluster. If the points remain in the same cluster even after training the algorithm for several iterations, this is another indication that we should quit. In the event that the maximum number of iterations is achieved, we can finally halt the training. Let's say we chose 100 iterations as the number of iterations. Before coming to an end, the process will repeat for 100 times [14].

Gaussian Mixture Model: The Gaussian distribution, also known as the normal distribution, is a popular model for continuous variable distribution. The Gaussian distribution can be written as follows for a single variable x:

$$\mathcal{N}(x|m,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left\{-\frac{1}{2\sigma^2}(x-m)^2\right\}$$

where *m* denotes the mean and σ^2 denotes the variance.

The multivariate Gaussian distribution for a D-dimensional vector X looks like this:

$$\mathcal{N}(X|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\left\{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right\}$$

where μ is a D-dimensional mean vector, Σ is a D×D covariance matrix, and |Σ| denotes the determinant of Σ.

The Gaussian distribution appears in a variety of contexts and can be motivated from a variety of perspectives. For instance, consider the sum of multiple random variables. The central limit theorem (due to Laplace) states that, under certain mild conditions, the sum of a set of random variables, which is itself a random variable, has a Gaussian distribution as the number of terms in the sum increases.

As shown in Figure 4 how a linear combination of Gaussian densities can result in very complex densities. Almost any continuous density can be approximated to arbitrary accuracy by using a sufficient number of Gaussians and adjusting their means and covariances as well as the coefficients in the linear combination [25].

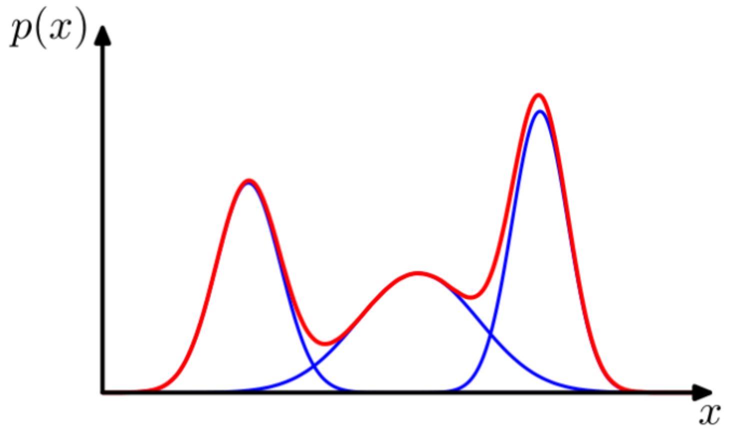


Fig. 4. This figure shows complex Gaussian densities which is formed from a linear combination of Gaussian densities.

As a result, we consider the superposition of K Gaussian densities of the form

which is called a *mixture of Gaussians*. Each Gaussian density $N(x|\mu_{k,}\Sigma_k)$ is called a *component* of the mixture and has its own mean $\mu_{k,}$and covariance Σ_k.

The parameters π_k are called *mixing coefficients*. They verify the conditions

$$\sum_{k=1}^{K}\pi_k = 1 \quad \text{and} \quad 0 \leq \pi_k \leq 1$$

The parameters π, μ and Σ govern the shape of the Gaussian mixture distribution, where we have used the notation π = { π_1,...,π_k} and $\mu = \{\mu_1, \dots\dots, \mu_k\}$ and $\sum = \{\sum_1, \dots\dots, \sum_k\}$.. Maximum likelihood is one method for determining the values of these parameters. The likelihood function's log is given by

$$\ln p(X|\pi,\mu,\Sigma) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(x_n|\mu_k,\Sigma_k)\right\}$$

Because of the presence of the summation over k inside the logarithm, the situation is much more complex than with a single Gaussian. As a result, there is no longer a closed-form analytical solution for the maximum likelihood solution for the parameters. Iterative numerical optimization techniques are one method for maximising the likelihood function. We can also use a powerful framework known as Expectation Maximization.

Expectation Maximization:

Expectation: The expectation step, or E step, consists in calculating the expectation of the component assignments C_k for each data point $x_i \epsilon x$ given the model parameters $\pi_k, \ \mu_{k,} \ \sigma_k$.

Maximization: The second step, known as the maximisation step or M step, consists of maximising the model parameters-based expectations calculated in the E step. This step involves updating the values $\pi_k, \ \mu_{k,} \ \sigma_k$.

The iterative process is repeated until the algorithm converges and provides a maximum likelihood estimate. Intuitively, the algorithm works because knowing the component assignment C_k for each x_i simplifies solving for $\pi_k, \ \mu_{k,} \ \sigma_k$. while knowing $\pi_k, \ \mu_{k,} \ \sigma_k$ simplifies inferring $p(C_k|x_i)$. The expectation step is for the latter case, while the maximisation step is for the former. Thus, maximum likelihood estimates of non-fixed values can be calculated efficiently by alternating between which values are assumed fixed or known.

GMM Algorithm:

1. Set some random values for the mean $\mu_{k,}$, covariance matrix $\sum_k$, and mixing coefficients π_k(or other values).
2. Calculate the $C_k$values for all k.
3. Again Using the current \C_k values, estimate all of the parameters.
4. Calculate the log-likelihood function.
5. Include a convergence criterion.
6. If the log-likelihood value converges to a value (or if all of the parameters converge to a value), stop; otherwise, proceed to Step 2.

This algorithm only guarantees that we will arrive at a local optimal point; it does not guarantee that this local optimal point will also be the global optimal point. As a result, if the algorithm begins at different points, it will generally end up in different configurations [24].

2.1.4 Typical Conventional Machine Learning Models

Naïve Bayes Classifier Algorithm: Because it presumes that the occurrence of one trait is unrelated to the occurrence of other features, it is referred to as naive. For instance, if the fruit is to be identified based on colour, shape, and flavour, an orange is recognised as a yellow, spherical, and sweet fruit. Therefore, without relying on one another, each characteristic helps to identify it as an orange. As a result of its reliance on the Bayes' Theorem principle, it is known as Bayes'.

The Bayes Theorem: It is commonly referred to as Bayes' rule or Bayes' law, is used to calculate the likelihood of a hypothesis given some prior information. The conditional probability determines this.

The Bayes theorem's formula is as follows:

$$\mathbf{P(A|B) = P(B|A)\ P(A)/P(B)}$$

were

- The posterior probability is P(A|B). Hypothesis A's likelihood of occurring in the observed occurrence B
- The likelihood probability is P(B|A). Probability of the information provided that a hypothesis is likely to be correct.
- The prior probability is P(A). hypothesis' likelihood before looking at the evidence.
- The marginal probability is P(B). likelihood of the evidence.

The Bayes theorem can be used to create a learning system that, given a fresh set of qualities, predicts the likelihood that the response variable will belong to a particular class. Two different kinds of probabilities can be calculated from the training data and are included in an NB model:

1. The likelihood of each class, first
2. The conditional probability for every class considering every input value.

A Naive Bayes model identifies the posterior probability for each class and chooses the class with the highest probability by multiplying several separate estimated probabilities together. The maximal a posteriori probability refers to this. The complexity of the model is decreased by the conditional independence of the features.

Linear classifiers called naive Bayes classifiers are renowned for being straightforward but extremely effective. The Naive Bayes (NB) classifier is one of the most often used learning techniques for creating machine learning (ML) models. It is based on the well-known Bayes Theorem of probability and assumes that predictors are independent. It is a probabilistic classifier that makes predictions based on conditional probabilities for the classifications that are most likely to occur. Along with being straightforward, Nave Bayes is known to perform better than even more complex classification techniques. The method makes a significant assumption that the input variables are independent, which is not necessarily true for actual data. Nevertheless, the method is highly helpful for a variety of complex issues. A Nave Bayes classifier converges more quickly than discriminative models like logistic regression if the NB conditional independence requirement is valid. Additionally, less data is required to train the model. Even if the NB assumption of feature independence is false, it frequently performs well in practise. Text classification, spam filtering, sentiment analysis, recommendation systems, and other applications use the NB algorithm. However, Nave Bayes classifiers may perform poorly due to serious breaches of the independent assumptions and nonlinear classification issues.

Types of Naïve Bayes Model:
Gaussian: The Gaussian model presupposes that features are distributed normally. This indicates that the model thinks that predictor values are samples from the Gaussian distribution if they take continuous values rather than discrete ones.

Multinomial: When the data is multinomially distributed, the multinomial Naive Bayes classifier is employed. Problems with document classification are its main application. It denotes the category—such as politics, sports, or entertainment—a specific document falls into. The frequency of the terms included in the document is one of the features/predictors that the classifier uses.

Bernoulli: In contrast to multinomial classifiers, Bernoulli classifiers use independent Boolean variables as predictor variables, such as whether a given word is included in a document or not [14].

Naïve Bayes Algorithm:
When using this algorithm, the following steps are taken:

1. Determine the prior probabilities for the class labels provided.
2. Compute conditional probabilities for each class using each attribute.
3. Add the conditional probability for the same class.

4. Multiply the probability obtained in step 3 by the prior probability.
5. Ascertain which class has a greater likelihood. The provided input set has higher probability classes.

Support Vector Machine Algorithm: Support vector machines (SVM) are supervised machine learning algorithms that can be used for regression and classification issues. They can be used for regression as well as support vector classification (SVC) and support vector regression (SVR). SVM is frequently employed in classification. For example, text categorization, image classification, and face detection may all be done using the SVM algorithm. When they were first created in the 1990s, they were hugely popular at the time and, with a little tweaking, they are now the preferred techniques for high-performing algorithms. While it may seem straightforward, not all datasets can be separated linearly. In reality, almost all data in the actual world is randomly distributed, making it challenging to distinguish between distinct classes linearly. Such issues are resolved by SVM using a kernel technique. Kernel technique carries out a few data transformations to choose the best line to use to divide data based on the labels or outputs specified. Kernel trick is a technique for solving nonlinear problems with a linear classifier. Data that appear to be indivisible along a linear path must be transformed. As seen in Figure 5 the kernel function transforms the initial nonlinear data into a higher-dimensional space where they can be separated [14].

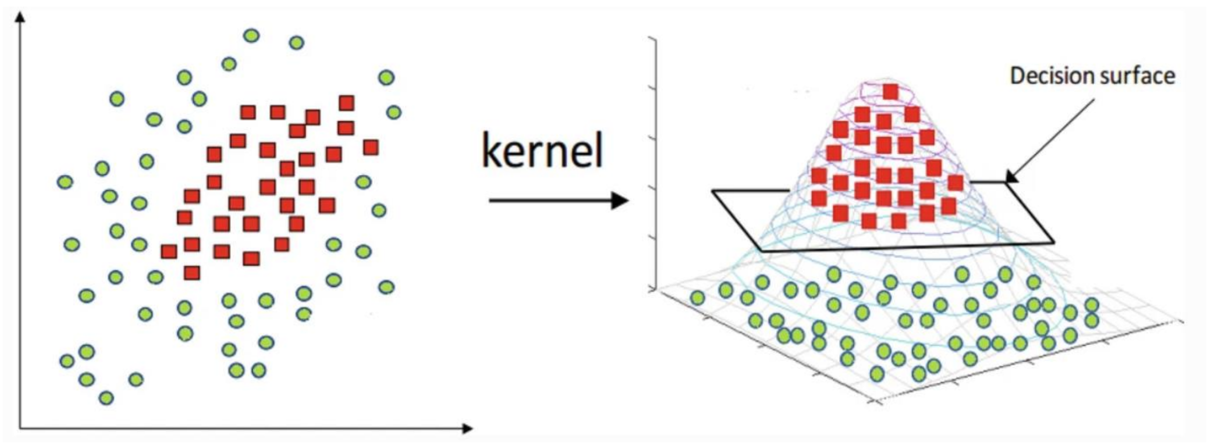


Fig. 5. Behaviour of kernel function.

The data are mapped from two-dimensional to three-dimensional space in this picture, and a decision boundary that distinguishes between the various classes is also identified. By using the kernel method, we are able to work in the original feature space without having to calculate the coordinates of the data in a higher dimensional space [26] There are numerous kernel functions, including the sigmoid, polynomial, and Gaussian/RBF kernels. To prevent the model from being overfit, care must be used when selecting the kernel function. Thus, regularisation (covered in another section of the book) and selecting the appropriate kernel function (with the appropriate parameters) are crucial. RBF, a kernel function with a localised and finite response, is the most popular type [27].

Finding a hyper-plane that best divides the characteristics into distinct domains is the foundation of SVM. In essence, SVM determines the separation between two observations. Finding a hyper-plane in N-dimensional space—where N is the number of features—that clearly classifies the data is the goal of the SVM. To put it another way, the goal of SVM is to identify decision planes that indicate decision boundaries. A decision plane is a structure that divides a collection of objects with various class memberships. The SVM algorithm, also known as margin maximisation, attempts to maximise the distance between the various classes involved because there are numerous such linear hyper-planes. The likelihood that fresh data will generalise well is increased if the line that maximises the distance between the classes is found. Figure 5 depicts a binary class as an example. Support vectors (SV) are the points that are closest to the hyper-plane, and margin is the separation between the vectors and the hyper-plane. The fundamental assumption to make in this situation is that the chance of accurately categorising points in their respective regions or classes increases with the distance of the support vector points from the hyper-plane. In order to determine the hyper-plane, SV points are crucial because if vector positions change, the hyper-location planes will also change. This hyper-plane is also referred to as a maximising margin hyper-plane technically.

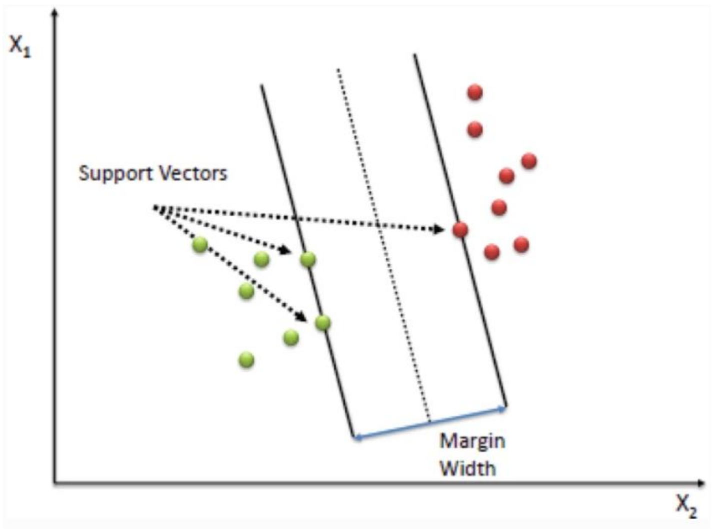


Fig. 6. Example of Binary Class – SVM.

A nonlinearly separable function is mapped into a higher dimension linearly separable function using kernel methods. The classifier represented by the normal vector and bias of the hyper-plane is discovered via a support vector machine training technique. This hyper-plane (boundary) creates the greatest feasible distance between the various classes. A SVM identifies the classifier represented by the bias and normal vector.

Decision Trees: Algorithms for supervised machine learning include decision trees (DTs). Decision nodes and leaves are the two components that can be used to explain the tree. Each node (decision node or internal node) within the tree represents a test on a particular feature, and the leaves are the final results.

Both classification and regression issues can be solved using these techniques. One can map the many outcomes brought on by the decisions or consequences using the DT and a certain set of inputs. These trees are employed to give the user graphical outputs based on a number of independent variables. Both missing and heterogeneous data can be handled with DT. Additionally, DT algorithms are capable of generating understandable rules. It is possible to do classification with few calculations. DTs are employed in advertising, sales, churn reduction, fraud and anomaly detection, medical diagnosis, etc.

A tree-like graph is a DT. An edge is produced at each node based on certain particular traits of one of the attributes. We may say that edges represent the answers to the questions, and nodes indicate a point where we choose an attribute and ask a question. Last but not least, the leaves stand in for actual output or class labels. They work with a straightforward linear decision surface when making nonlinear decisions. For non-parametric machine learning modelling of regression and classification issues, DTs are frequently utilised. A DT uses the predictor data to make sequential, hierarchical decisions on the outcome variable in search of answers. Hierarchical implies that the model is established by a set of queries that, when used to categorise any observation, result in a class label or a value. Once configured, the model behaves like a protocol, producing a certain outcome from the input data through a series of "if this happens, then that happens" conditions.

The absence of underlying assumptions on the distribution of the mistakes or the data characterises a non-parametric approach. In essence, it indicates that the model was built using the data that had been observed.

When compared to other classification algorithms, the DT algorithm's comprehension level is rather straightforward. Regression and classification can both be done using the DT method. The DT technique can handle missing data as well as diverse data. Because DTs frequently imitate human-level reasoning, it is easier to comprehend the data and arrive at sound conclusions. Using DTs, it is simple to understand the reasoning behind the data (unlike SVM, ANN, etc.)

This algorithm, which is greedy and recursive, is known as Hunt's algorithm. Greedy refers to the process of evaluating an objective function locally to maximise (minimise) at each step. Recursive in the same way means that it breaks the main question into smaller ones and answers them all with the same justification. Purity is a statistic that is used to determine whether to divide at each node. When a node's data is evenly split 50/50, it is 100% impure, and when all of its data is from a single class, it is 100% pure. Maximum purity should be achieved and keep impurities to a minimum in order to maximise the model.

To decide whether to divide a node into two or more sub-nodes, DT may consider a number of factors. The purity of the node in relation to the target variable improves with each split. The most homogenous sub nodes are produced by splitting the nodes based on all of the available qualities. We can gauge the homogeneity (or heterogeneity) of a data table based on its classes if it has attributes and classes for the attributes. If a table only contains one class, it is said to be pure or homogenous. A data table is considered impure or heterogeneous if it contains many classes. The level of impurity can be quantified using a variety of metrics. Entropy, Gini index, and classification error are a few well-known indices to gauge the level of impurity [14].

Decision Tree Algorithm: The algorithm provided below can help you better understand the entire procedure:

1. A top-down recursive construction method is used to build the tree.
2. All of the training examples are at the root at the beginning.
3. Examples are recursively partitioned according to chosen attributes.
4. An impurity function is used to select attributes (e.g., information gain).

To halt partitioning, there must be:

- A given node's examples all belong to the same class.
- The leaf is the majority class, hence there are no further attributes for partitioning.
- There are no longer any examples.

Random Forest: DTs are sensitive to the particular data that was used to train them. The resulting DT and, thus, the predictions will be very different if the training data are altered. Additionally, because they cannot trace back, i.e., there is no backtracking once they have split and advanced, DTs prefer to discover local optima and are computationally expensive to train. They also carry a significant risk of over-fitting. We use Random Forest, which demonstrates the power of fusing numerous DTs into one model, to address these flaws. One of the most adaptable and well-known machine learning algorithms is Random Forest. It can be applied to classification and regression issues in machine learning. It is built on the idea of ensemble learning, which is a method of integrating various classifiers to address difficult issues and enhance model performance. As its name implies, Random Forest is a classifier that uses a number of DTs on different subsets of the provided dataset and averages the results to increase the dataset's predictive accuracy. Instead of depending on a single DT, the Random Forest uses guesses from each tree and forecasts the result based on the votes of the majority of predictions. Higher precision and over-fitting are prevented by the larger number of trees in the forest.

It is significantly simpler to create a good, generalised model (on a dataset) thanks to its integrated ensemble capacity. The ensemble learning technique Random Forests, also known as "random decision forests," combines many algorithms to produce superior results for classification, regression, and other tasks. Though each model works best when coupled with others, they are all weak individually. This machine learning algorithm builds a forest and makes it somewhat random, as suggested by the algorithm's name. Each tree is planted and grows as follows: if the training set has N instances, then N cases are randomly selected but with replacement for the sample of N cases. for growing the tree this sample will serve as the tree's training set. In contrast, if there are M input features or variables, a number m < M is supplied, and the best split on this m is used to split each node. At each node, m variables are randomly chosen from the M. Throughout the growth of the forest, m is maintained at a fixed value. Each tree is grown to its best ability. Additionally, each tree is grown on a unique sample of unique data [14].

Logistic Regression:

Logistic regression is a classic predictive modelling technique that is still widely used to model binary categorical variables [28]. Logistic regression is linear regression's classification counterpart. It is also among the most widely used supervised machine learning algorithms. It is used to forecast the categorical dependent variable from a set of independent variables. By quantifying each independent variable's unique contribution to predicting the output of a categorical dependent variable, logistic regression is an efficient and powerful way to analyse the effects of a group of independent variables with binary outcomes. As a result, the outcomes must be categorical or have discrete values. Each can be yes or no, 0 or 1, true or false, and so on, but instead of providing the exact value as 0 or 1, it provides probabilistic values that fall between 0 and 1. This method can be used to detect spam, credit card fraud, and predict whether a given mass of tissue is benign or malignant, among other things.

In the sense that the logistic regression algorithm is used for classification tasks, the name of this algorithm can be a little confusing. In fact, logistic regression is very similar to linear regression, with the exception of how it is used. For regression problems, linear regression is used, whereas logistic regression is used for classification problems. The term "regression" implies that a linear model should fit into a linear space. Predictions are mapped to a linear combination of features [29] by the logistic function shown in Figure 6, which means that predictions can be interpreted as class probabilities. Explanatory variables are used to model the odds or probabilities that describe the outcome of a single trail. Because the models are still "linear," they work well when the classes are separable linearly (i.e., they can be separated by a single decision surface). Instead of fitting a regression line, we fit a "S"-shaped logistic function, as shown in Figure 6, that predicts two maximum values (0 or 1). The logistic function's curve indicates the likelihood of something, such as whether the email is spam or not.

Logistic regression iteratively identifies the strongest linear combination of variables with the highest probability of detecting the observed outcome using the components of linear regression reflected in logit scale. A logarithmic transformation of the input values produces the output, which is defined by the logistic function $f(x) = 1/(1 + ex)$. The probability is then forced into a binary classification using a threshold. The goal of logistic regression is to train the model to find the coefficient values that will minimise the error between the predicted and actual outcomes. Maximum likelihood estimation is used to calculate these coefficients.

Types of Logistic Regression: Logistic regression is classified into three types based on categories:

Binomial: The dependent variables in binomial logistic regression can only be of two types: 0 or 1, yes or no.
Multinomial: It allows for three or more types of dependent (categorical) variables. Multinomial logistic regression can be used to differentiate between cats and dogs or sheep.
Ordinal: It allows for three or more ordered types of dependent variables, such as small, medium, or large [14].

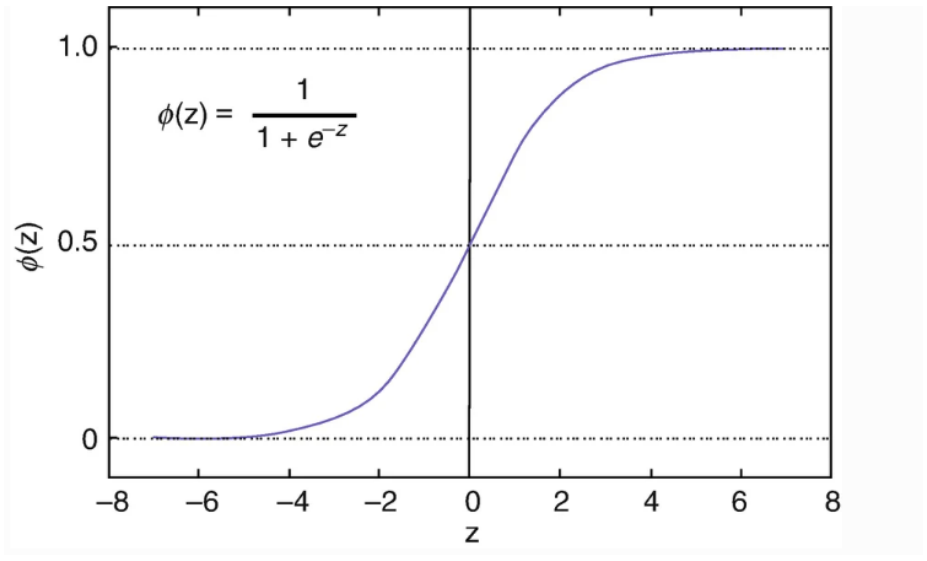


Fig. 7. Linear Combination of Features.

2.1.5 Performance Metrics for Classification Models

Various classification algorithms were discussed earlier. Classification accuracy is frequently used as a metric to assess a classifier's effectiveness. We use a confusion matrix to start off the talk.
Confusion Matrix: The confusion matrix is used to provide a fuller view while evaluating a model's performance. Confusion matrix provides a matrix output that sums up the model's overall performance. It is the simplest approach to gauge how well a classification problem is performing when the output can include two or more different types of classes. An N *x* N matrix, where N is the number of expected classes, is a confusion matrix. A confusion matrix, as shown in Figure 7 is a table having two dimensions, "Actual" and "Predicted," as well as "True Positives (TP), "True Negatives (TN)," "False Positives (FP)," and "False Negatives (FN)" for the situation at hand when N = 2.

True Positives (TP): When a data point's actual class and anticipated class are both 1, this is the situation.

True Negatives (TN): A data point is considered to be true negative when both its actual class and anticipated class are 0.

False Positives (FP): when a data point's anticipated class is 1 but the actual class is 0. It also goes by the name type I mistake.

False Negatives (FN): These occur when a data point's projected class and actual class are both 1. Type II mistake is another name for it.

The foundation for measuring accuracy, recall, precision, specificity, etc. is the confusion matrix given in Figure 8 below.

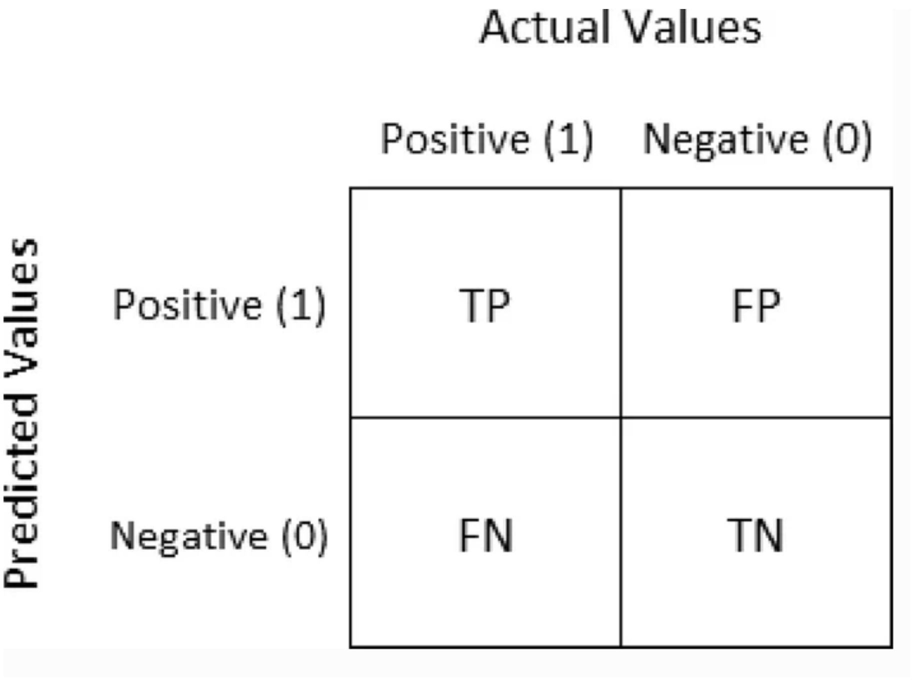


Fig. 8. Details of Confusion Matrix.

Accuracy: The most typical performance statistic is this one. It is described as the proportion of all forecasts made to those that were true. From the confusion matrix, the following may be calculated:

Accuracy=TP+TN/TP+FP+FN+TN

Only when there are about equal numbers of samples from each class does it perform well. When the costs of misclassifying the minor class samples are very large, a serious issue arises. The cost of failing to diagnose a patient with a rare but fatal disease is far higher than the expense of subjecting a healthy person to additional tests.

Classification Report: This report includes the Precisions, Recall, F1, and Support scores. These are described in detail below:

Precision: It is calculated by dividing the total number of correctly positive results by the total number of positive results that the classifier anticipated.

Precision=TP/TP+FP

Recall or Sensitivity: This is calculated as the proportion of valid positive outcomes to all relevant samples (all samples that should have been identified as positive).

Recall=TP/TP+FN

Specificity: The quantity of negative results that our ML model returned is known as specificity.

Specificity=TN/TN+FP

F1 Score: Using this score, we can determine the harmonic mean of recall and precision. The F1 score's mathematical range is [0.0.1]. It provides information on the classifier's accuracy (the percentage of correctly classified cases) and robustness (it does not miss significant number of instances). Although a forecast made with high precision but poor recall is incredibly accurate, a significant number of difficult-to-classify cases are missed as a result. The model performs better the higher the F1 score. The F1 score seeks to strike a balance between recall and precision. We can use the formula below to determine the F1 score:

$$F1=2*Precision*Recall / (Precision+Recall)$$

Comparing two models that have a high recall but a low precision is challenging. F-Score is used to compare them as a result. F-score aids in measuring both recall and precision simultaneously. By punishing the extreme values more harshly, it substitutes the harmonic mean for the arithmetic mean. This is the case because it equally weighs the respective contributions of recall and precision [14].

2.1.6 Different Aspects of An Anomaly Detection Problem

Anomaly detection is the identification of items, events, or observations in a dataset that do not conform to an expected pattern or other items [15]. The correlation between workloads and application performance should be stable in normal circumstances, but it fluctuates significantly when faults occur [16]. The various facets of anomaly detection are identified and covered in this section. As was previously said, a particular formulation of the problem depends on a variety of variables, including the type of the input data, the availability or lack thereof of labels, as well as the limitations and restrictions imposed by the application domain. This section highlights the complexity of the problem domain and demonstrates why a wide range of anomaly detection methods are required.

Nature of the Input: The type of the incoming data is a crucial component of any technique for detecting anomalies. According to Tan et al. (2005), Chapter 2, input is typically a group of data instances (also known as an object, record, point, vector, pattern, event, case, sample, observation, or entity). A set of attributes (also known as variables, characteristics, features, fields, or dimensions) can be used to characterise every data instance. The qualities may be categorised, binary, or continuous, among other sorts. One attribute (univariate) or several attributes could be present in each data instance (multivariate). All attributes in multivariate data instances may be of the same type or may be a combination of distinct data types.

The suitability of anomaly detection approaches depends on the attributes' nature. For instance, separate statistical models must be employed for continuous and categorical data when using statistical approaches. The qualities' nature would also dictate the distance to be employed for nearest-neighbor-based approaches. The pairwise distance between instances is frequently presented as a distance or similarity matrix in place of the real data. Techniques that need unique data instances, like many statistical and classification-based techniques, are inapplicable in such circumstances.

Type of Anomalies: The type of intended anomaly is a key factor in an anomaly detection technique. The three categories of anomalies are as follows:

Point Anomalies: An instance of data is referred to be a point anomaly if it may be said to be abnormal relative to the rest of the data. The majority of research on anomaly detection is centred on this sort of anomaly because it is the most straightforward.

Contextual Anomalies: A contextual anomaly (also known as a conditional anomaly; Song et al. 2007) is when a data instance is anomalous in one context but not in another. The data set's structure infers the concept of a context, which must be provided as part of the problem formulation. Two sets of attributes are used to define each data instance: Contextual and Behavioural attributes.

Collective Anomalies: A collective anomaly is when several connected data examples are aberrant when compared to the total data set. Although each data instance in a collective anomaly may not be an anomaly in and of itself, their existence as a collection is.

Data Labels*:* A data instance's labels indicate whether it is a normal or anomalous instance. It should be highlighted that it is frequently unaffordable to get labelled data that is accurate and representative of all behavioural kinds. Since labelling is frequently done manually by a human expert, obtaining the labelled training data set necessitates a significant amount of work. Obtaining labels for normal behaviour is typically easier than obtaining a labelled set of anomalous data examples that covers all potential types of anomalous behaviour. Additionally, the anomalous behaviour is frequently dynamic in character; for instance, new kinds of anomalies may appear for which there is no labelled training data. Atypical occurrences are extremely uncommon in some situations, such as air traffic safety, where they could result in catastrophic disasters.

Anomaly detection methods can function in one of the following three modes, depending on the availability of the labels: Supervised, Semi-Supervised, Unsupervised anomaly detection.

Output of Anomaly Detection: The way that abnormalities are reported is crucial for any technique for anomaly identification. The outputs generated by anomaly detection methods typically fall into one of the two categories below: Scores and Labels.

2.2 Related Work

The detection of anomalies is a well-established research field with numerous theoretical and practical application areas. This section focuses on more recent work aimed specifically at anomaly detection in cloud computing.

Machine learning techniques have received a lot of attention from intrusion detection researchers in order to address the shortcomings of knowledge base detection techniques. Anomaly detection techniques include both supervised and unsupervised techniques. To achieve good results for these techniques, many algorithms were used. Salima et al. provides an overview of machine learning techniques for detecting anomalies. The experiments showed that supervised learning methods outperform unsupervised learning methods when the test data contains no unknown attacks. Non-linear methods, such as SVM, multi-layer perceptron, and rule-based methods, achieve the best performance among supervised methods. Although they differ in their ability to detect all attack classes efficiently, unsupervised techniques such as K-Means, SOM, and one class SVM outperformed the other techniques [17].

H. Won et al. says that multi-layer data is collected in a cloud infrastructure that provides virtual machines and containers as a service, and this data is used for supervised learning-based machine learning to detect fault. The accuracy of fault detection was compared and analysed using various models, and various methods and feature elimination models for removing unnecessary features were introduced. A feature analysis model was used to test the effect of removing features that are less important for fault detection on fault detection accuracy, and a method to increase fault detection accuracy through a combination of feature analysis models was proposed [18]

An important area of research is failure prediction based on runtime system metrics. Failure prediction can aid in improving overall system performance. There have been numerous models and methods that use the current state of a system as well as, in many cases, previous experience. Fu et al. [19] present an anomaly detection method based on runtime monitoring, current system state, and previous experience. This method monitors system execution and collects runtime performance data that is typically unlabelled. It is a self-evolving anomaly detection framework that combines classification and clustering techniques. It does not require any prior failure history and can self-adapt by recursively exploring newly generated verified detection results for future anomaly identification, unlike other failure detection approaches.

Du et al. analysed the performance metrics for container-based microservices in this paper, introduced two phases for detecting anomalies with machine learning techniques, and then proposed an anomaly detection system for container-based microservices. This ADS is based on service and container performance monitoring data, machine learning algorithms for classifying abnormal and normal behaviour, and the fault injection module for collecting performance data in various system conditions [9].

3. Experimental Setup for Data Collection

A dedicated testbed is used to collect the training and test data used to evaluate the above Machine Learning algorithms discussed. The testbed environment is described in this section, including the components for collecting data and injecting anomalies into monitored hosts.

The data for diagnosis has been taken from a Multi-microservices Multi-virtualization Multi-cloud (M3) Monitoring framework that checks the performance of microservices deployed across heterogeneous virtualization platforms. This monitoring solution employs a Book-Shop application with Docker containers and virtual machines deployed in Amazon and Azure cloud environments it was tested in a variety of scenarios [5].

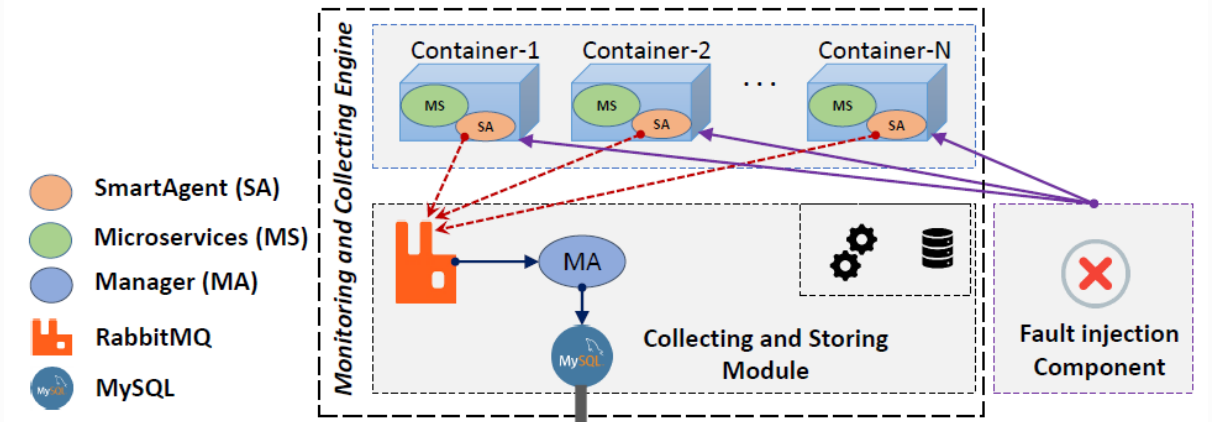


Fig. 9. M3 Monitoring framework.

An experiment was conducted for the evaluation of the proposed monitoring system M3 using the defined setup. The test application is deployed in container and VM environments on Amazon EC2 and Microsoft Azure. A large number of experiments were carried out by varying the workload configurations in order to measure various system parameters such as CPU, Memory, Response, and Throughput.

Both Amazon EC2 and Microsoft Azure machines are running the Linux operating system Ubuntu:16.04 (https://www.ubuntu.com/) and a Docker platform (version 17.06.1 ee 1) (https://www.docker.com/) to execute the microservices. Azure's VM configuration is StandardA1v2 with 1 vCPU and 2 GB of memory. We considered four such virtual machines. Amazon's VMs were of the t2.micro type, with 1 VCPU and 1 GB of memory per machine. For our experiment, we also considered two VMs.

Different software was installed on VMs and containers. We chose Tomcat (Version 7) (http://tomcat.apache.org/) and Nginx (Version 1.13.7) (https://nginx.org/en/) for the web server and MySQL (Version 5.7) (https://www.mysql.com/) for the database. Docker Hub (https://hub.docker.com/) was used to obtain all container images. The machine configurations used for the experiments are as follows: The first machine used Java (Version 8) on the virtual machine guest OS, the second machine had Docker platform installed and used Docker Compose file (version 1.18.0) for which one image (https://hub.docker.com/-/tomcat/) for Tomcat and (https://hub.docker.com/-/mysql/) for MySQL was used, The third machine had the same configuration as the second machine but with different services, and the final machine had the Docker platform installed and used a Docker-Compose file with two images: one for Tomcat and one for MySQL. In Amazon, two machines are used; one uses a Java virtual machine, while the other instals the Docker platform and applications via a Docker-Compose file that includes one image for Tomcat and another for Nginx. The proposed system has been tested in three different scenarios:

• **Scenario 1:** Deploying two microservices (Tomcat and MySQL) for Book and Purchase services in a single Microsoft Azure VM. In addition, one VM running two microservices (Nginx and Tomcat) for the Amazon Web Services-deployed User Interface service.

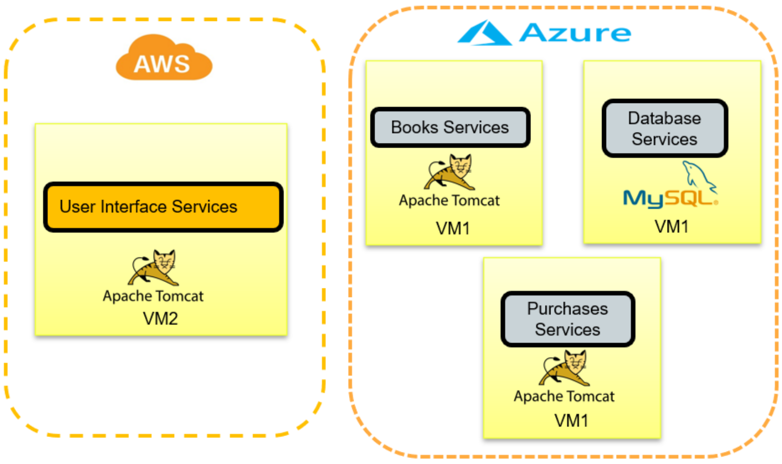


Fig. 10. Scenario1 Deployment.

• **Scenario 02:** Deployed two microservices (Tomcat and MySQL) for the Book service, which is running in the first container, and two microservices (Tomcat and MySQL) for the Purchase service, which is running in another container; all containers are running in Azure. In addition, for the User Interface service, we deployed two microservices (Tomcat and Nginx) in a single container hosted by Amazon Web Services.

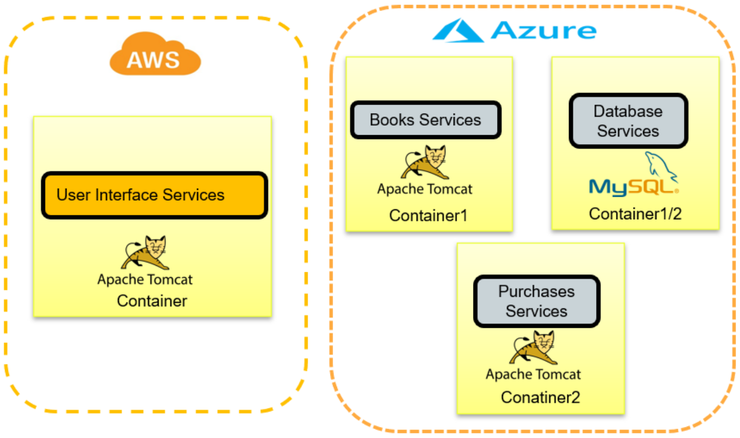


Fig. 11. Scenario 2 Deployment.

• **Scenario 03:** Deployed one microservice (Tomcat) for Book and Purchase services in one container and one microservice (MySQL) for Database in another; all containers are deployed in Azure. In addition, for the User Interface service, one VM running two microservices (Nginx and Tomcat) is deployed in Amazon Web Services.

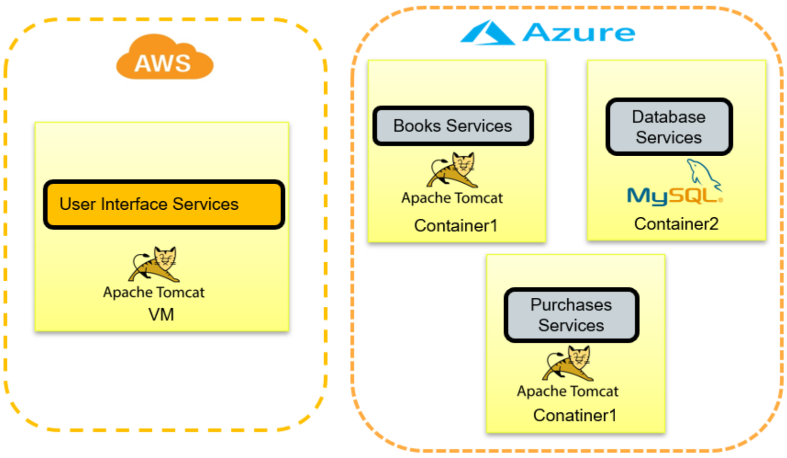


Fig. 12. Scenario 3 Deployment.

Experiments have been carried out in which the manager pushed system and process level statistics about services running on two public clouds. The metrics obtained for manager were related to all JMeter tests for results analysis. As previously stated, the JMeter tests generate 900 requests in order to simulate the workload and validate the agents' ability to capture performance metrics in all three scenarios and collected data of parameters such as CPU, Memory, Response, and Throughput of all three services – Book, Purchase and User interface [5].

4. Implementation

This section provides an overview of the development approach used, specifically how the work was divided into a series of prototype steps. It then discusses each step, in detail that are used in the development.

4.1 Implementation Technologies

The Machine Learning code has been written on a web IDE, called Google Colab notebook which is a Jupyter notebooks that are well integrated with Google Drive and operate in the cloud, it is simple to set up and access.

Programming language used for this project is Python - The advantages that make Python the best choice for projects based on machine learning and AI include flexibility, platform freedom, access to excellent libraries and frameworks for ML, and a large community. These increase the language's general appeal.

Various python libraries have been used in the coding for classification, visualization, and clustering such as NumPy, matplotlib, scikit-learn, pandas.

NumPy: In essence, NumPy offers n-dimensional array objects. Additionally, NumPy offers mathematical functions that can be applied to several calculations.

matplotlib: To visualise data, the scientific plotting library matplotlib is typically used. Importantly, data analysis requires visualisation. Histograms, scatter plots, lines, etc. can all be plotted.

scikit-learn: Data analysis and data mining tools are available through scikit-learn, which is based on NumPy, SciPy, and matplotlib. It comes with built-in classification and clustering algorithms as well as practise datasets including the iris dataset, the Boston housing market dataset, the diabetes dataset, etc.

pandas: Data analysis is done with pandas. It can output charts and graphs from multi-dimensional arrays as an input. A table containing columns of several data kinds may be given to pandas. It may be able to import data from different databases and data formats, including SQL, Excel, and CSV.

4.2 Methodology

Below are the steps and flow that shows how the implementation has been done to achieve the goal of this project. |The raw data has been collected and through techniques like value imputation (replacing missing data with default values) and feature transformation (generating new features from existing ones), data preparation tries to increase the quality of a dataset. The next step in the feature engineering process is to choose and create features from the original data so that the model may be learned by the selected algorithm. Then choose and configure the algorithm that best fits the data structure for clustering the data to detect normal and abnormal data. After that the model is trained and assessed in the last step. This pipeline is not always run sequentially in a single direction. Retrying a task to improve the end outcome is a typical practise. This happens more frequently in the early stages of development while all of the algorithms' precise parameters are still being tweaked, but it can still happen even after the system has been put into operation.

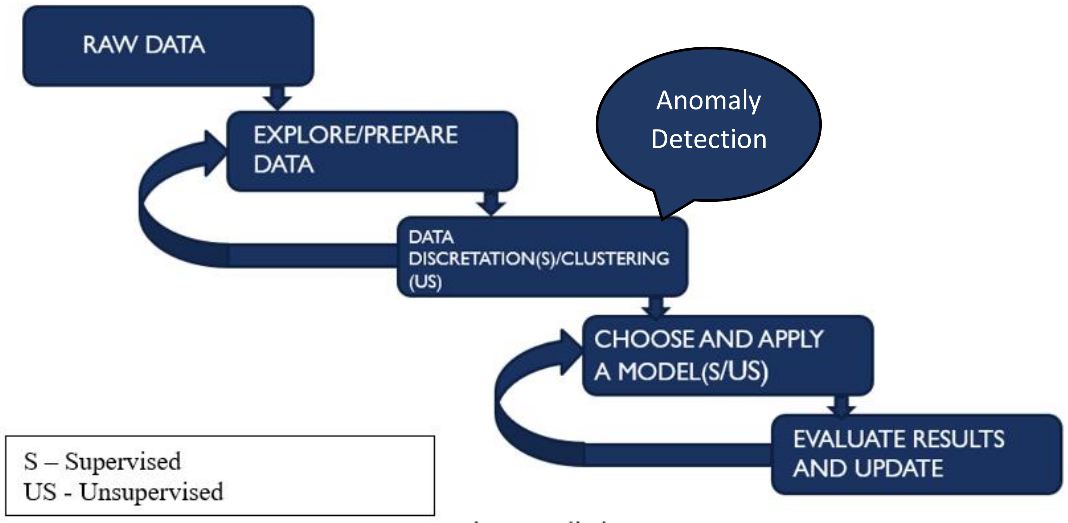

RAW DATA
EXPLORE/PREPARE DATA
Anomaly Detection
DATA DISCRETATION(S)/CLUSTERING (US)
CHOOSE AND APPLY A MODEL(S/US)
EVALUATE RESULTS AND UPDATE
S – Supervised
US - Unsupervised

Fig. 13. Methodology for the Anomaly detection.

4.3 Data Collection

A Monitoring Agent is a piece of software that gathers data from microservices that are executing in containers or virtual machines. And then the Monitoring Manager, a software that collects monitoring data from agents placed within containers or virtual machines dispersed over diverse cloud environments. The manager keeps the outcomes of each data collection by a monitor agent in a MySQL database for later analysis as shown in figure 14 below [5].

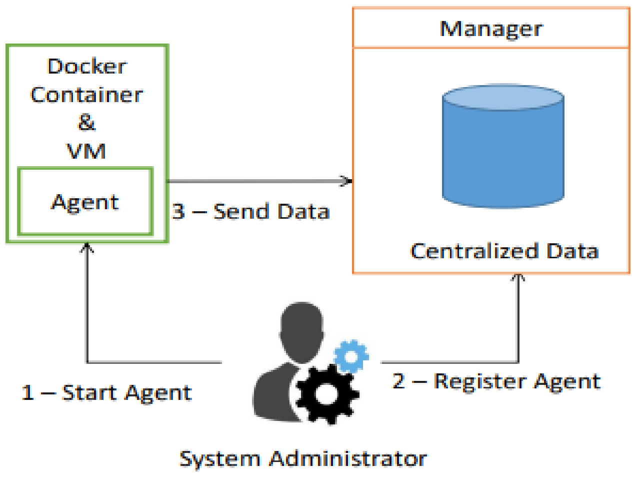


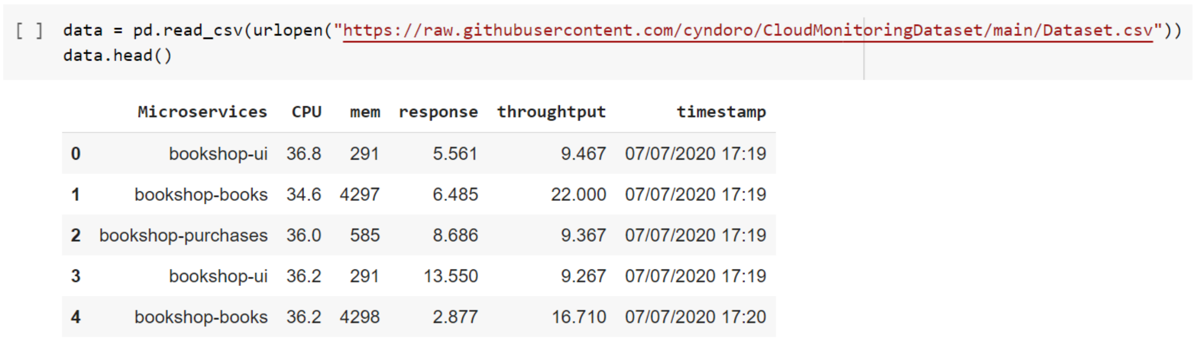
Fig. 14. Flow of Data Collection in M3 Monitoring Framework.

The primary observations that have been taken into account when carrying out anomaly detection process are CPU (%), Memory (MB), Response (ms) and Throughput (mbps). The structure of the data stored in the database is shown as below.

CPU Usage: Finding out how much of these resources are being used by a given ML task is necessary to measure CPU consumption throughout that work. The values range from 0% to 100%, yet when using several cores, some operating systems may show values as high as 100%. Usually, the length of time that the CPU is being used at near to 100% is more interesting than the percentage by itself. This suggests that work completion time might be sped up with increased computing power. With accurate measurements, it may be feasible to determine the precise moment when the CPU is overloaded, enabling the team to concentrate its efforts on perfecting that one microservice rather than the pipeline as a whole.

Memory Usage*:* Memory issues are very similar to CPU/GPU ones, as computer memory is another resource that is being consumed by various tasks running and must be monitored for excessive loads during particular tasks. Monitoring memory can help to detect code defects as well. So, everything that was discussed for CPU would be applicable here.

Response: The interval between a terminal operator finishing an enquiry and getting a reply. The time it takes to respond to an enquiry includes the time it takes for the computer to process it and send back the answer to the terminal. A common metric for evaluating the effectiveness of an interactive system is response time.



```
[ ]  data = pd.read_csv(urlopen("https://raw.githubusercontent.com/cyndoro/CloudMonitoringDataset/main/Dataset.csv"))
     data.head()
```

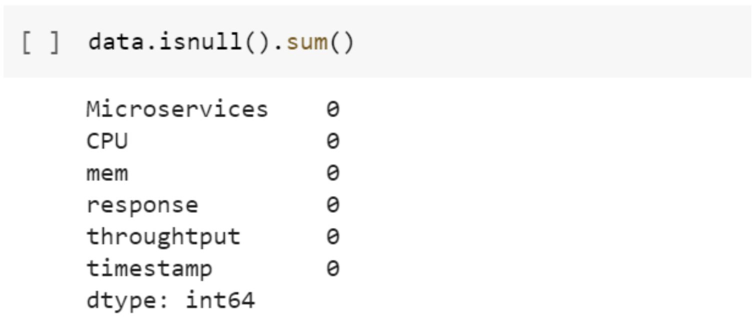
	Microservices	CPU	mem	response	throughtput	timestamp
0	bookshop-ui	36.8	291	5.561	9.467	07/07/2020 17:19
1	bookshop-books	34.6	4297	6.485	22.000	07/07/2020 17:19
2	bookshop-purchases	36.0	585	8.686	9.367	07/07/2020 17:19
3	bookshop-ui	36.2	291	13.550	9.267	07/07/2020 17:19
4	bookshop-books	36.2	4298	2.877	16.710	07/07/2020 17:20

Fig. 15. Dataset obtained from the monitoring setup.

Throughput: The number of requests made per second is the rate at which the system can handle its maximum load.

4.4 Data Pre-Processing

Missing Feature values*:* In general, there are a few crucial considerations to make while processing unknown feature values. The source of "unknownness" is one of the most crucial ones: I a value is absent due to forgetfulness or loss; (ii) a feature is inapplicable or non-existent for a specific instance; and (iii) for a given observation, the creator of a training set does not care about the value of a certain feature (so-called don't-care value). Similar to the case, the expert must select from a variety of approaches to deal with missing data [21]. In the dataset given, it doesn't have any null values.



```
[ ]  data.isnull().sum()

     Microservices    0
     CPU              0
     mem              0
     response         0
     throughtput      0
     timestamp        0
     dtype: int64
```

Fig. 16. Checking for null values in the dataset.

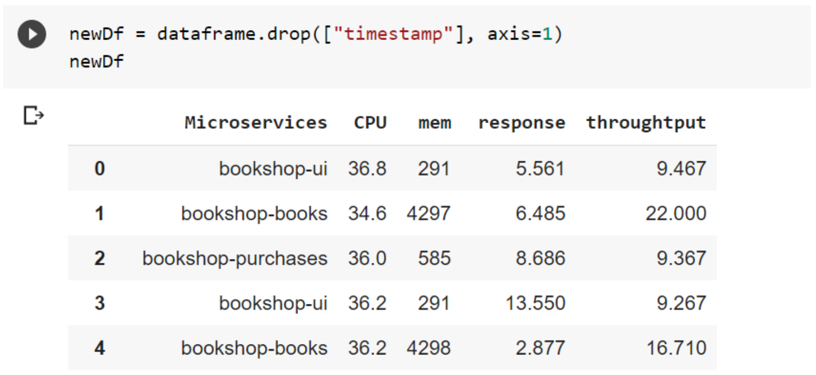
Feature Selection: The act of detecting and eliminating as many unnecessary and redundant features as feasible is known as feature subset selection. This makes the data less dimensional and makes learning algorithms work more quickly and efficiently. Features are typically described as:

• *Relevant:* These characteristics have an impact on the final product, and the others cannot fully fill their place.

• *Irrelevant:* Features that don't affect the result and whose values are produced at random for each sample are referred to as irrelevant features.

• *Redundant:* The simplest method to model redundancy is when one feature can do the function of another.

As just CPU, Memory, Response, and Throughput are used for analysis, timestamp, one of the features, has been deleted from the data. Additionally, the Microservices capability was eliminated in a couple situations when the process couldn't handle strings.



```
newDf = dataframe.drop(["timestamp"], axis=1)
newDf
```

	Microservices	CPU	mem	response	throughtput
0	bookshop-ui	36.8	291	5.561	9.467
1	bookshop-books	34.6	4297	6.485	22.000
2	bookshop-purchases	36.0	585	8.686	9.367
3	bookshop-ui	36.2	291	13.550	9.267
4	bookshop-books	36.2	4298	2.877	16.710

Fig. 17. Dataset after removal of a feature – Timestamp.

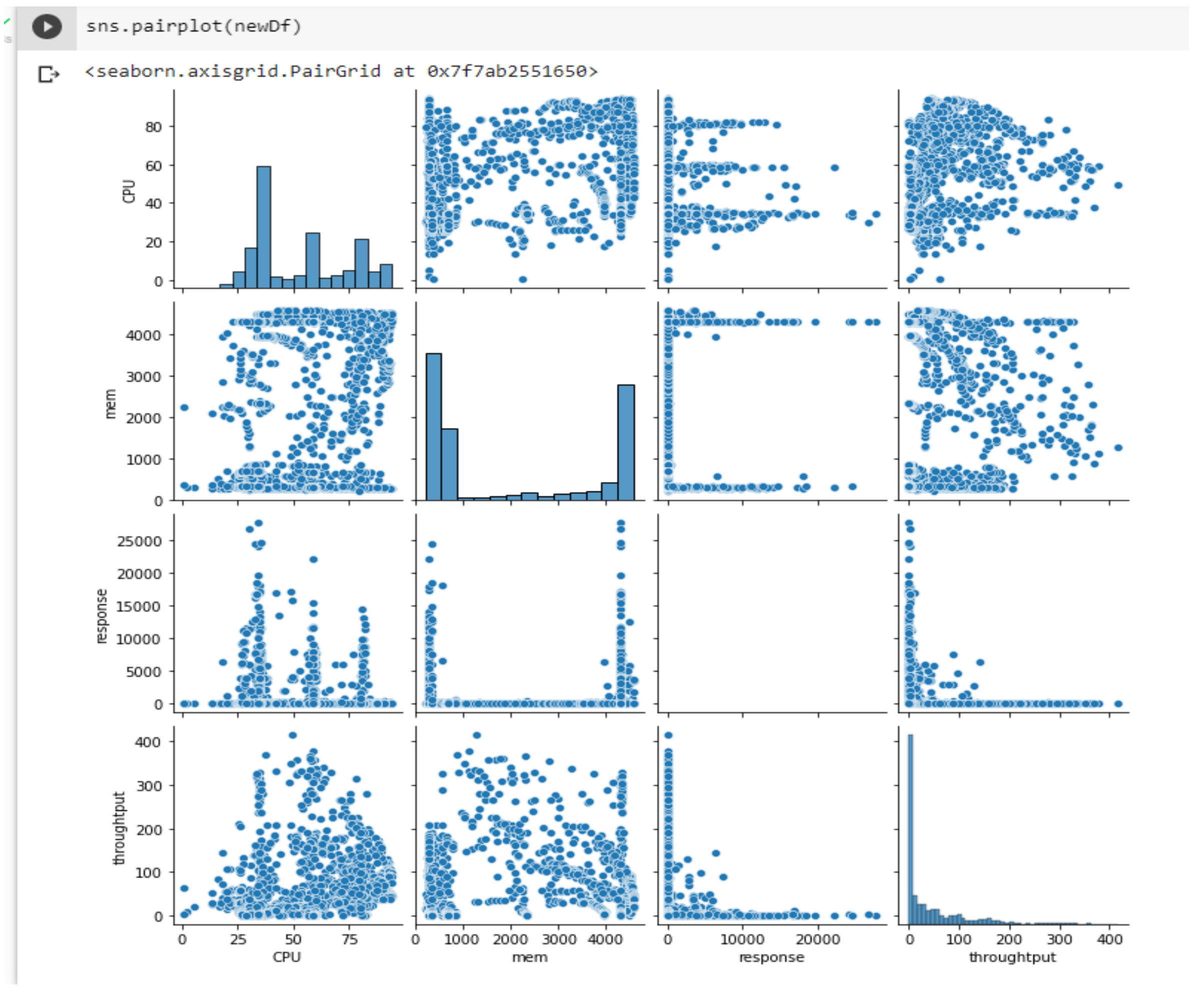
Data Visualization: In order to construct a range of plot styles useful for statistical data analysis and even some statistical model fitting, Seaborn offers high-level commands. The seaborn for the provided dataset is shown below. The following might be accomplished with standard Matplotlib commands (in fact, this is what Seaborn actually does), but the Seaborn API is far more practical.

A Matplotlib-based Python data visualisation library is called Seaborn. It offers a sophisticated drawing tool for creating eye-catching and educational statistical visuals. A well-designed visualisation simply has an exceptional quality. The total package has a pleasing aesthetic quality, stands out in terms of colour and layering, and flows naturally in terms of shapes while also giving us useful insights.

seaborn.barplot() method

In essence, a bar plot is used to aggregate categorical data using some methods, the mean being the default. It may also be seen as a representation of the group in motion. In order to apply this plot, we select a numerical column for the y-axis and a category column for the x-axis. We observe that this creates a plot taking a mean per categorical column.

Histograms of subsets can be used to view data in some cases. Since the project uses supervised machine learning which focuses on independent feature anomalies, histogram is employed as a visualisation tool to help users comprehend the data on their own, this has been shown in next session.


```
sns.pairplot(newDf)
```

```
<seaborn.axisgrid.PairGrid at 0x7f7ab2551650>
```

Fig. 18. Seaborn Plot for the dataset.

4.4.1 *Labelling the data (Normal/Anomaly)*

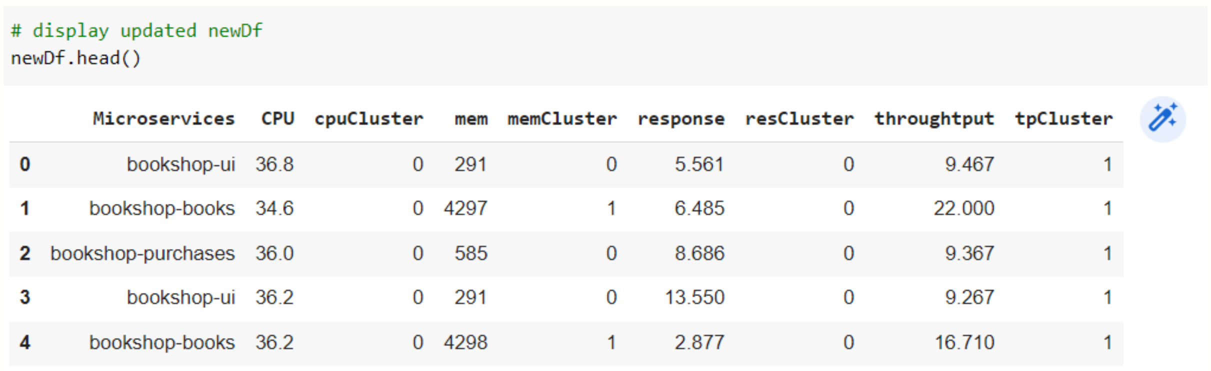
This project used two ways of approach to label the data as normal or anomaly. One is Discretization for supervised learning and another one is clustering for unsupervised learning.

Discretization: In here threshold has been set for each parameter and based on the threshold data has been labelled accordingly. Thresholds has been set based on the metrics of the parameters (as given in the section 3) of the microservices along with the research how the parameters.

It involves organising continuous data, such as CPU, memory, response, and throughput values, into discrete buckets. We can easily maintain the data labelling in this way. When training a model with discrete data, it goes more quickly and efficiently than when training it with continuous data. Large volumes of data can make the model run slowly even when continuous-valued data offers more information. Discretization can assist us in finding a balance between the two in this situation. Binding and using a histogram are two well-known techniques for data discretization. Although data discretization is helpful, it can be difficult to choose the range for each bucket. The fundamental difficulty in discretization is determining how many intervals or bins to use and how wide to make them.

In this case, we utilise a function called pandas.cut (). The segmented data can be bucketed and sorted using this function. Below is the code snippet used for CPU discretization, where threshold has been set to 70 % of the CPU utilization, labelled normal and anomaly as 0 and 1 respectively. In the same way remaining 3 parameters has been discretized, the threshold set for memory, response and throughput are 2000 MB, 2000 ms and 25 mbps respectively.

```
#Data discretization
label = pd.cut(newDf.CPU,bins=[0,70,99],labels=['0','1'])
newDf.insert(2,'cpuCluster',label)
```



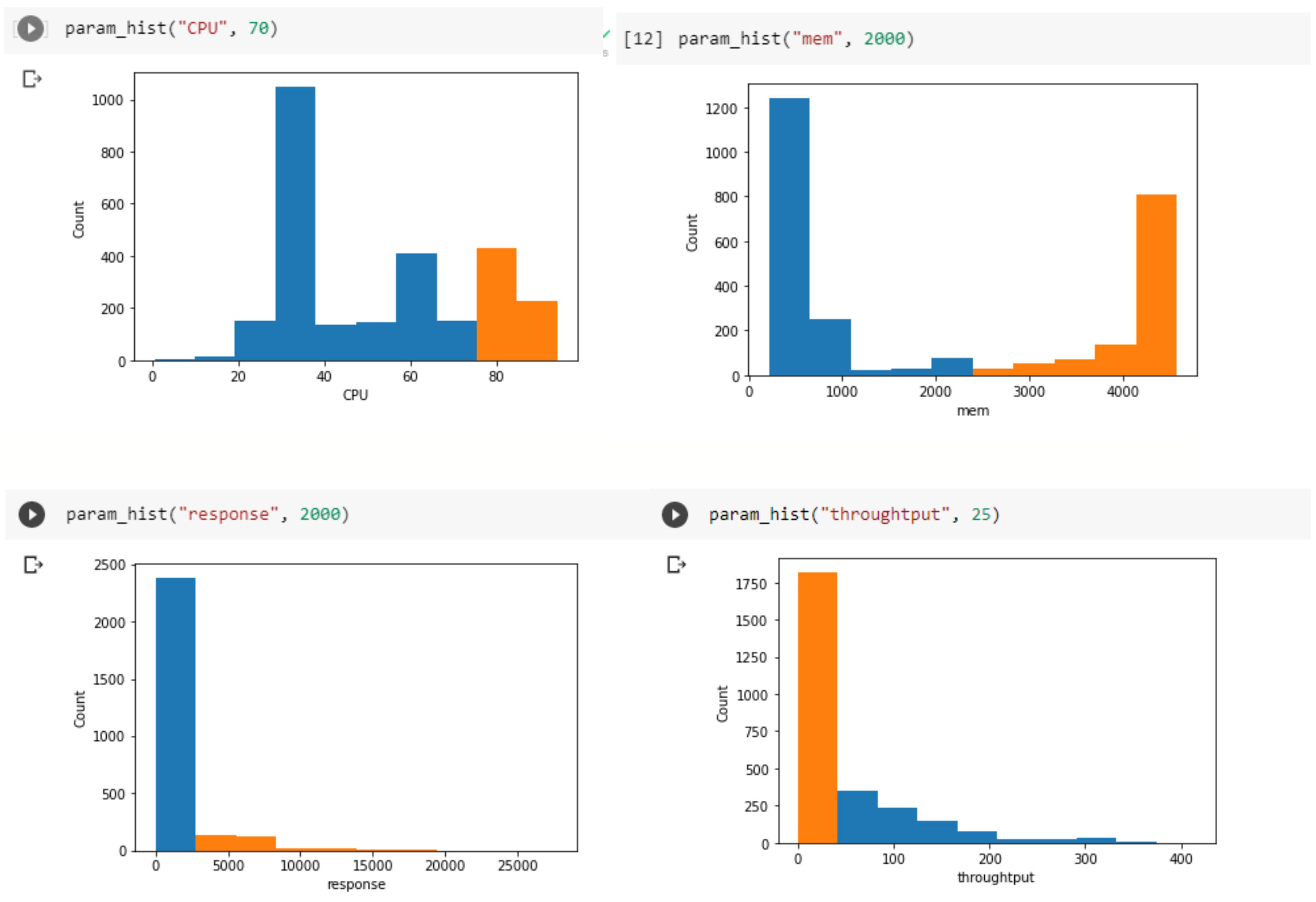
```
# display updated newDf
newDf.head()
```

	Microservices	CPU	cpuCluster	mem	memCluster	response	resCluster	throughtput	tpCluster
0	bookshop-ui	36.8	0	291	0	5.561	0	9.467	1
1	bookshop-books	34.6	0	4297	1	6.485	0	22.000	1
2	bookshop-purchases	36.0	0	585	0	8.686	0	9.367	1
3	bookshop-ui	36.2	0	291	0	13.550	0	9.267	1
4	bookshop-books	36.2	0	4298	1	2.877	0	16.710	1

Fig. 19. Dataframe after discretization.

Below is the user defined function created to view all 4 parameters histogram with respect to its threshold. The graph below shows the normal behaviour in the blue part and the abnormalities in the orange section.

```
def param_hist(param, thresh):
    x = newDf[param]
    _, _, bars = plt.hist(x, bins = 10, color="C0")
    for bar in bars:
      if param != "throughtput":
        if bar.get_x() > thresh:
            bar.set_facecolor("C1")
      else:
        if bar.get_x() < thresh:
          bar.set_facecolor("C1")
    plt.xlabel(param)
    plt.ylabel("Count")
    plt.show()
```



```
param_hist("CPU", 70)
```

```
[12] param_hist("mem", 2000)
```

```
param_hist("response", 2000)
```

```
param_hist("throughtput", 25)
```

Fig. 20. Histogram for all 4 parameters differentiation the threshold for anomalies. Blue shows normal value and orange shows the anomalies.

Clustering: K- means and GMM (Gaussian Mixture model) are the two Clustering used in this project for labelling the parameters. On comparison of these two clustering, k means clusters has been considered for training the dataset.

K-Means Clustering: In the r-dimensional Euclidean space R^r, where the distance d between any two points is defined as usual, let x_i, i = 1, 2, n be points. The goal of the k-mean clustering issue is to identify a set of k points m_1, m_2, ..., m_k that belong to $R^r$and minimise the expression for a given k.

$$\sum_{i=1}^{n} \sum_{j=1}^{k} (d(x_i, m_j))^2$$

Where d (x_j, m_j) denotes the distance of Euclidean in between two points x_j and m_j.

Let S_j = { x_i, i € {1, 2…., n}: d (x_j, m_j) <= d (x_j, m_r) where r is 1, 2……., k}. Sj is thus referred to as the jth cluster, while m_j is referred to as its centroid.

As a result, the goal of the k-mean clustering issue is to divide the multiset (data set) X= $(x_1, x_2, ..., x_n)$ of data points x_i into subsets (referred to as clusters) S_1, S_2, ..., S_k described above [22].

Below are the steps taken place in K-means clustering:

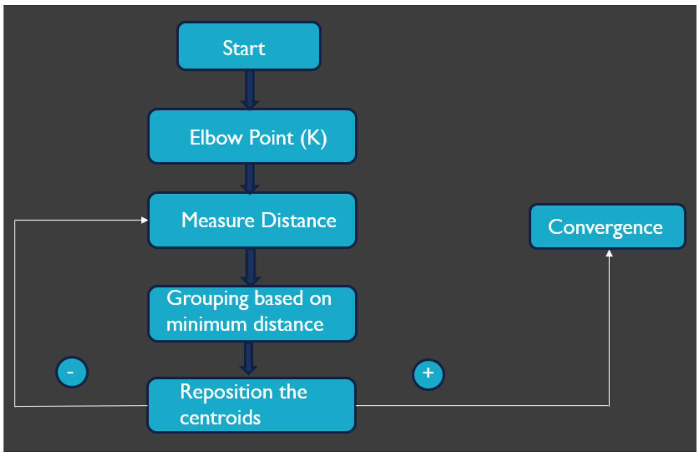


Fig. 21. Flow of KMeans Algorithm.

Here value of K is taken as 2 for the number of clusters using elbow point method, which will allow to separate the dataset into its many clusters.

```
Sumfsqrd_dis = []
K = range(1,10)
for num_clusters in K :
 kmeans = KMeans(n_clusters=num_clusters)
 kmeans.fit(dataframe)
 Sumfsqrd_dis.append(kmeans.inertia_)
plt.plot(K,Sumfsqrd_dis,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

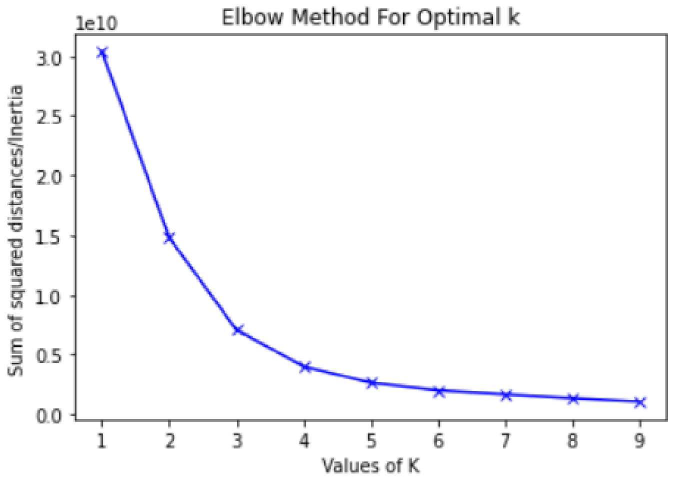


Fig. 22. Output of the Elbow method.

To build the cluster's centroid, select two random points at random as K value is chosen as 2. Now, each data point will be assigned to a scatter plot based on how far it is from the nearest K-point or centroid. To do this, a median between the two centroids will be drawn. By selecting a different centroid, repeat the procedure. The new centre of gravity of these centroids, will help us choose the new centroids. Each data point will then be reassigned to the new centroid. Then it carries out the same steps until convergence.

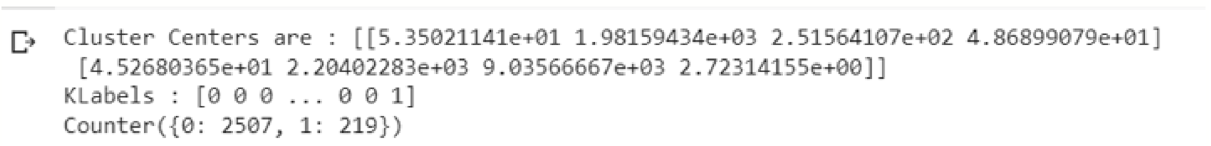
Below is the code for the KMeans implementation:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2)
new = dataframe.filter(['CPU','mem', 'response', 'throughtput'], axis=1)
kmeans.fit(new)


centroids = kmeans.cluster_centers_
Klabel = kmeans.labels_
print ("Cluster Centers are :", centroids)
print ("KLabels :", Klabel)
from collections import Counter
Counter(Klabel)
```

Centroids clusters and count of the clusters are shown below which is the output of the above code. Hypothetically, assumed that cluster that has a greater number of datapoints are the normal behaviour and another is anomaly. So as per below count 0's is normal and 1's are anomalies.



```
Cluster Centers are : [[5.35021141e+01 1.98159434e+03 2.51564107e+02 4.86899079e+01]
 [4.52680365e+01 2.20402283e+03 9.03566667e+03 2.72314155e+00]]
KLabels : [0 0 0 ... 0 0 1]
Counter({0: 2507, 1: 219})
```

Fig. 23. Output of KMeans clustering. It shows the Centroids, Labels, and the no. of each label.

Below is the relation between the response and throughput for the KMeans clustering done above.

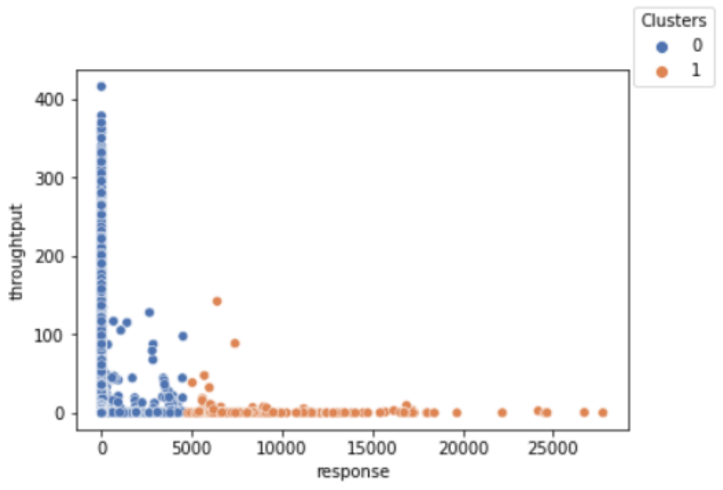


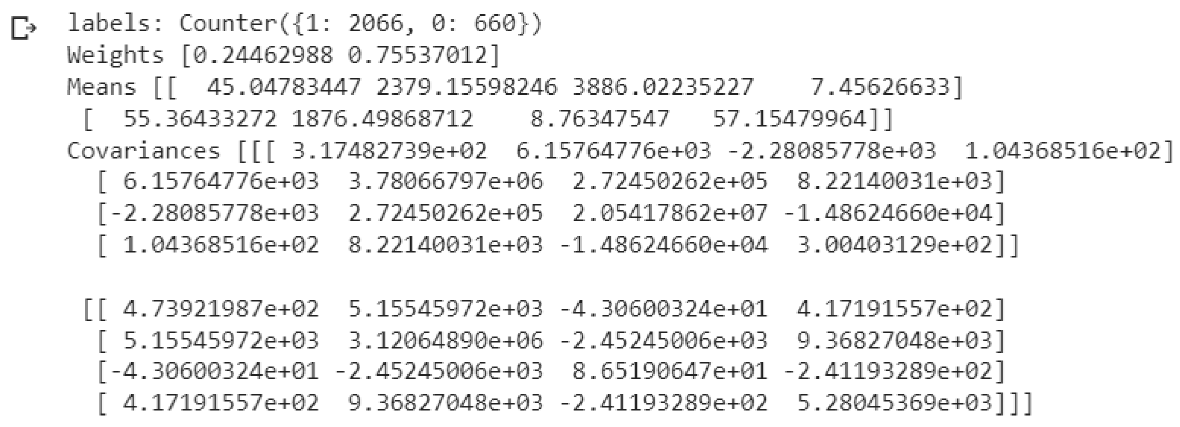
Fig. 24. Graph of KMeans clustering. It shows how it has clustered between throughput and response in Y and X axis respectively. 0's represents normal behaviour, 1's says abnormal behaviour.

GMM Clustering: As discussed in section 2.1.3 GMM clustering has been done using EM algorithm E and M stands for expectation and minimization. Below is the code for the implementation of GMM clustering. n_components value can be obtained as same way used for KMeans algorithm. Here the value used is 2 as we needed normal and abnormal clusters.

```
gmm = GaussianMixture(n_components = 2, init_params='random')

# Fit the GMM model for the dataset
# which expresses the dataset as a
# mixture of 3 Gaussian Distribution
gmm.fit(new)
cluster = gmm.predict(new)
cluster
print ('labels:', Counter(cluster))
print('Weights', gmm.weights_)
print('Means', gmm.means_)
print('Covariances', gmm.covariances_)
```

The out of the implementation above has been given below which shows the label count, component weights $N(x|\mu_{k,}\Sigma_k)$, Mean $\mu_{k,}$ and Covariance Σ_k of the data points.



```
labels: Counter({1: 2066, 0: 660})
Weights [0.24462988 0.75537012]
Means [[  45.04783447 2379.15598246 3886.02235227    7.45626633]
 [  55.36433272 1876.49868712    8.76347547   57.15479964]]
Covariances [[[ 3.17482739e+02  6.15764776e+03 -2.28085778e+03  1.04368516e+02]
  [ 6.15764776e+03  3.78066797e+06  2.72450262e+05  8.22140031e+03]
  [-2.28085778e+03  2.72450262e+05  2.05417862e+07 -1.48624660e+04]
  [ 1.04368516e+02  8.22140031e+03 -1.48624660e+04  3.00403129e+02]]

 [[ 4.73921987e+02  5.15545972e+03 -4.30600324e+01  4.17191557e+02]
  [ 5.15545972e+03  3.12064890e+06 -2.45245006e+03  9.36827048e+03]
  [-4.30600324e+01 -2.45245006e+03  8.65190647e+01 -2.41193289e+02]
  [ 4.17191557e+02  9.36827048e+03 -2.41193289e+02  5.28045369e+03]]]
```

Fig. 25. Output of KMeans clustering. It shows the Labels, weights, Means and Covariances.

Below is the relation between the response and throughput for the GMM clustering done above.

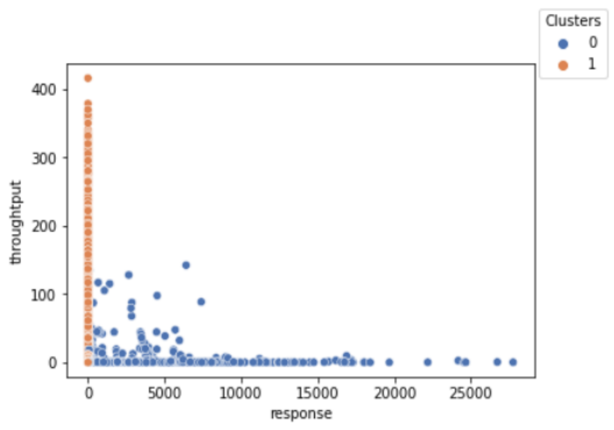


Fig. 26. Graph of GMM clustering. It shows how it has clustered between throughput and response in Y and X axis respectively. 1's represents normal behaviour, 0's says abnormal behaviour.

Comparing the output of both the clustering methods discussed above, it looks like k-means has the output which make sense up to a point based on the hypothesis. So further modelling is done using k-means clustering output only.

4.5 ML Modelling

Classification algorithms, such as decision trees or support vector machines (SVM), frequently share an interface for training and testing models. A function $train(S_1 \ldots S_n)$ is used for training, with each $S_i \epsilon \mathbb{R}^m \times \mathcal{L}$ representing a labelled sample of *m* metrics derived from the observed host. The set L contains all possible labels. We use $\mathcal{L} = \{normal, abnormal\}$ in the context of anomaly detection. The *train* function generates a classification model in the form of a *classify function*: $\mathbb{R}^m \rightarrow \mathcal{L}$. In other words, the *classify* function takes an unlabelled sample and predicts a label from $\mathcal{L}$.

Training is usually done offline and is computationally expensive, whereas the classify function evaluates quickly enough to be used in real-time. A sufficient number of S_i samples must be recorded in advance in order to perform the training. While the host is running normally, samples with the label *normal* can be easily recorded. This necessitates the intervention of a human administrator to manually restore the host's normal operation. Obtaining *abnormal* samples is more difficult because it necessitates the artificial injection of anomalies into the host.

On the host, a framework for controlled anomaly injection must be implemented and executed, while all observed samples must be labelled as *abnormal*. It is critical to place the host in different load situations for both *normal* and *abnormal* samples in order to capture all behaviour patterns that can be expected later, during runtime. This necessitates the development of a second framework for generating realistic workloads that change throughout the data collection process.

The overviews of the machine learning algorithms that are included in the evaluation are discussed below. To obtain the necessary training and test data, a prototype implementation of the above frameworks and data collection procedures is used.

In this research, widely used algorithms for training classification models on the datasets are compared: Naive Bayes (NB), Logistic Regression, Decision Tree, Random Forest (RF) and Support Vector Machine (SVM). The goal of these classifiers is to identify anomalous services using monitored performance data.

Dataset contains 2676 records. Dataset has been divided like, 80% of the records are used to train the classification model and the remaining 20% are used to validate the model.

Five of the methods in the library scikit-learn are used to train the classification models. Section 5 displays the outcomes.

- Naive Bayes
- Support Vector Machines (configured with kernel = rbf)
- Random Forests (configured witih max_depth = 20, random_state = 42, verbose = 1 and n_estimators = 50)
- Decision Tree (configured with criterion = 'entropy’, random_state=42 and max_depth = 6)
- Logistic Regression (configured with max_iter=10000)

5. Evaluation and Results

Below is the model comparison for all the parameters i.e., CPU, memory, response and throughput in the dataset for supervised clustering. The confusion matrix, precision, recall, and F1 score (refer section 2.1.5) were computed in this evaluation to assess the effectiveness of the model. The precision and recall together make up the F1 score, which is calculated using the following formula:

F1 = 2 (recall * precision / recall + precision)

ROC curve shows the true and false positive rates for all the models.

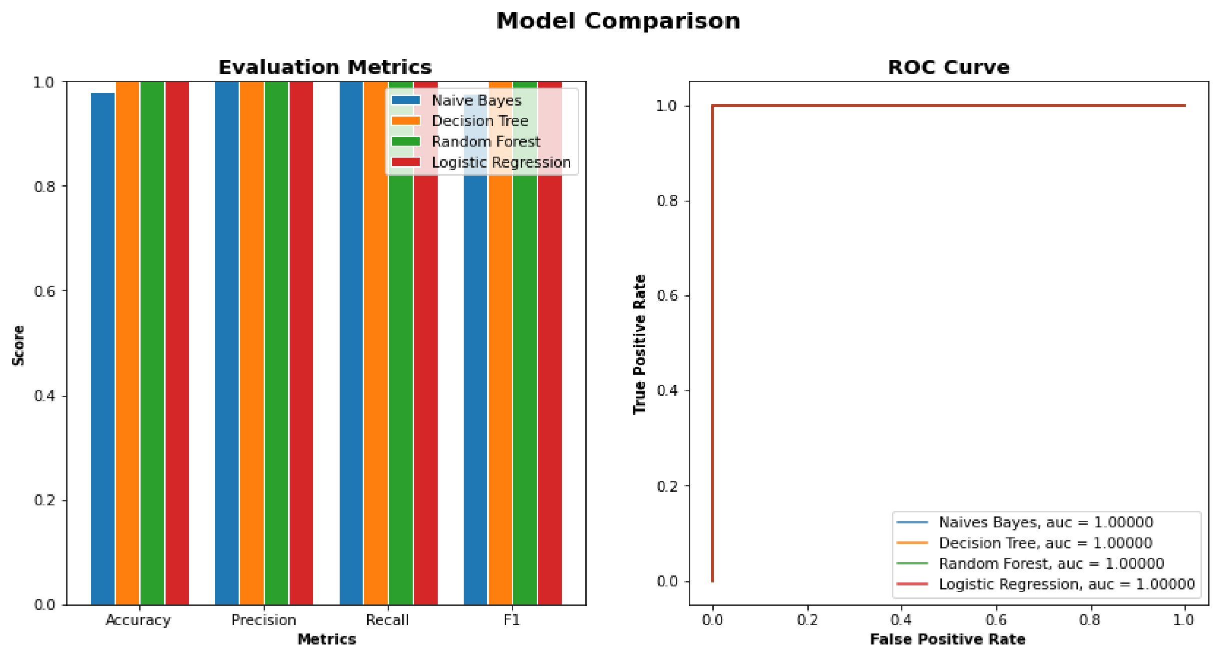
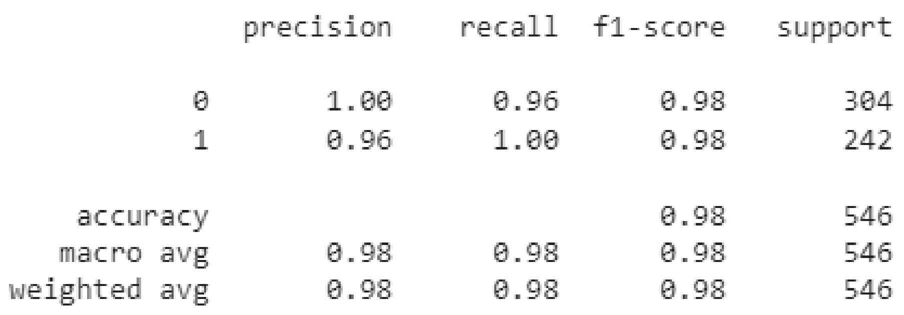


Fig. 27. Evaluation Metrics – Model Comparison.

Below are the classification reports for all the models for CPU, to note is that is it same for all the parameters as well. Macro average returns the average without taking into account the proportion for each label in the dataset and specifies the function to compute f1 for each label. The function weighted average returns the average while taking the proportion for each label in the dataset into account. It computes f1 for each label.

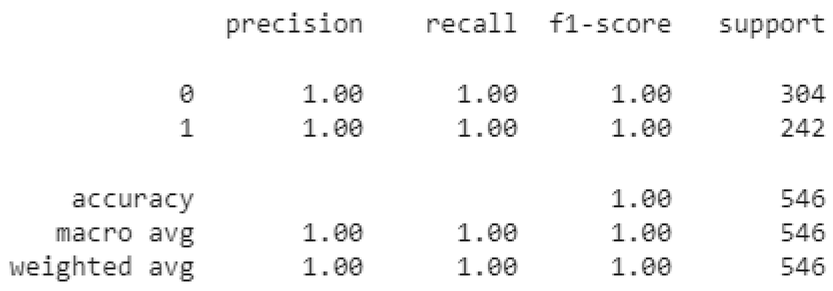


```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       304
           1       0.96      1.00      0.98       242

    accuracy                           0.98       546
   macro avg       0.98      0.98      0.98       546
weighted avg       0.98      0.98      0.98       546
```

Fig. 28. Naïve Bayes Classification Report for supervised labelling

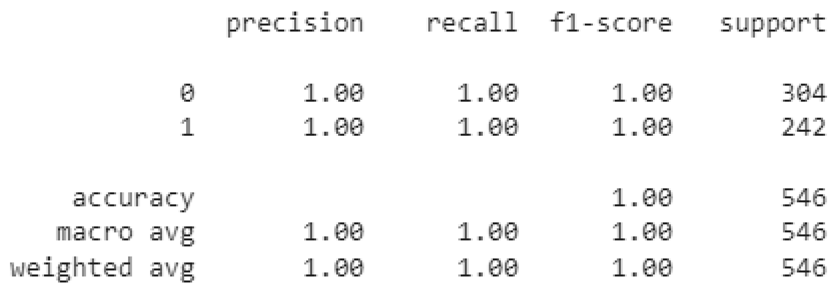


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       304
           1       1.00      1.00      1.00       242

    accuracy                           1.00       546
   macro avg       1.00      1.00      1.00       546
weighted avg       1.00      1.00      1.00       546
```

Fig. 29. Random Forest Classification Report for supervised labelling

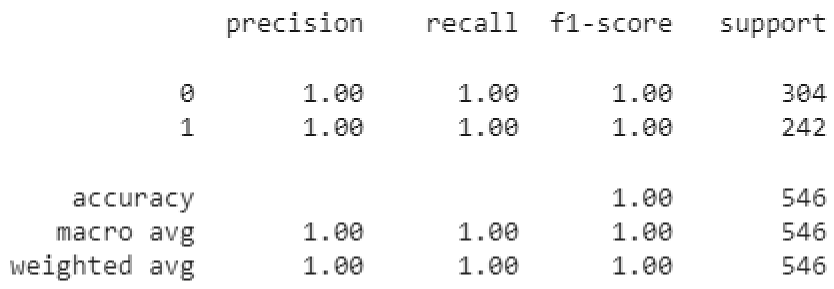


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       304
           1       1.00      1.00      1.00       242

    accuracy                           1.00       546
   macro avg       1.00      1.00      1.00       546
weighted avg       1.00      1.00      1.00       546
```

Fig. 30. Decision Tree Classification Report for supervised labelling



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       304
           1       1.00      1.00      1.00       242

    accuracy                           1.00       546
   macro avg       1.00      1.00      1.00       546
weighted avg       1.00      1.00      1.00       546
```

Fig. 31. Logistic Regression Classification Report for supervised labelling

For unsupervised clustering below are the evaluation metrics, where Naïve bayes shows 95.23% of accuracy and Support vector machine shows 99.81% of accuracy. Below are the confusion matrices for each classifier respectively.

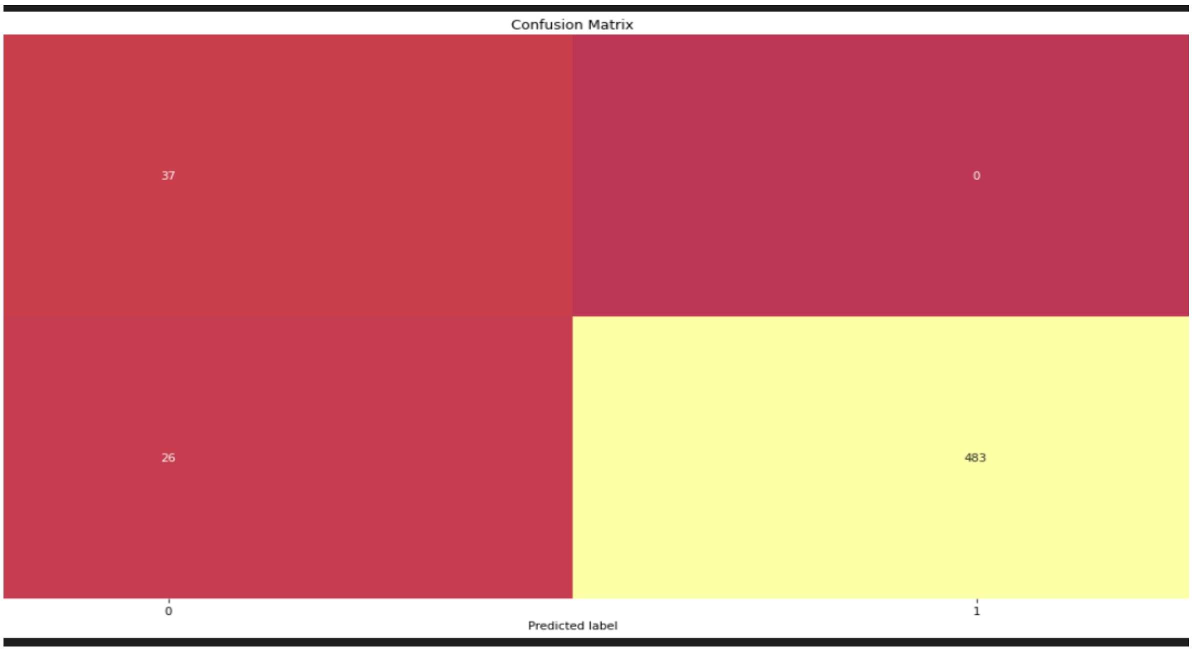


Fig. 32. Confusion matrix for Naïve Bayes Classifier

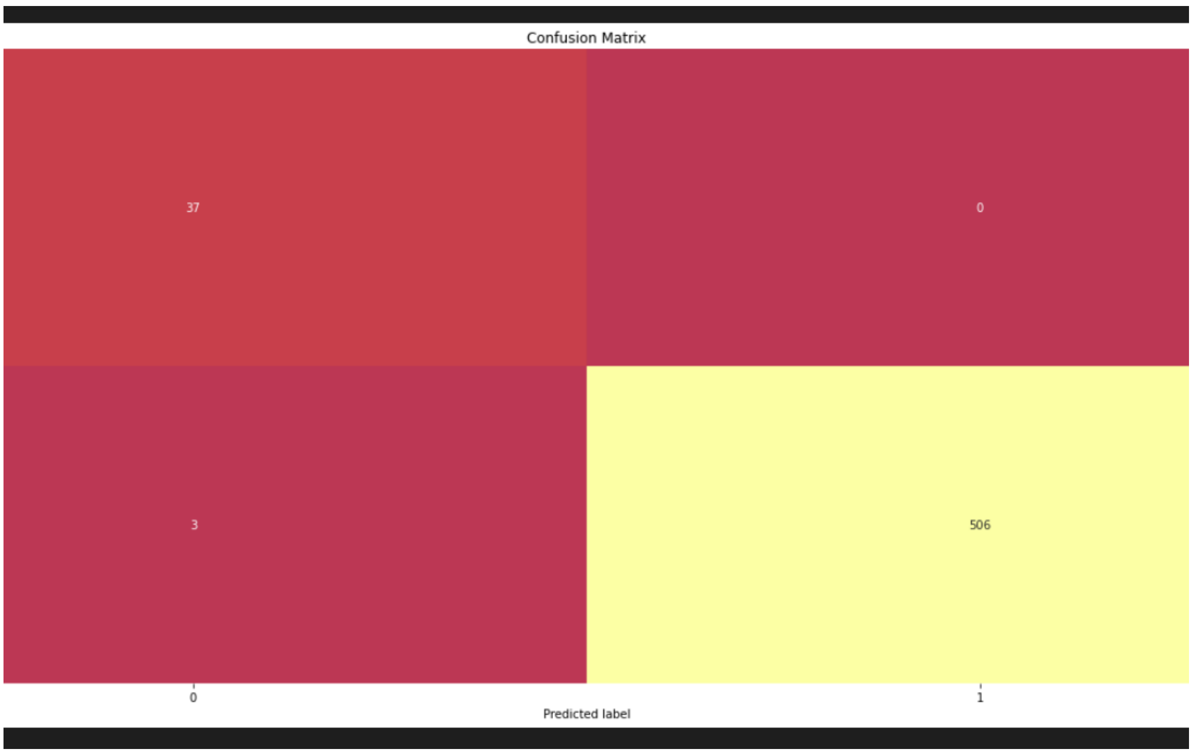
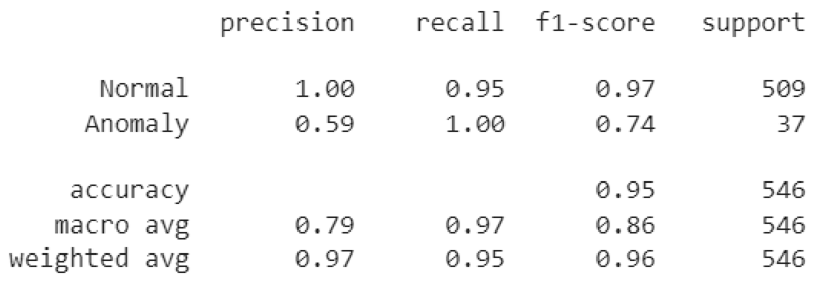


Fig. 33. Confusion matrix for Support vector machine Classifier

The classification reports are as follows for unsupervised clustering.

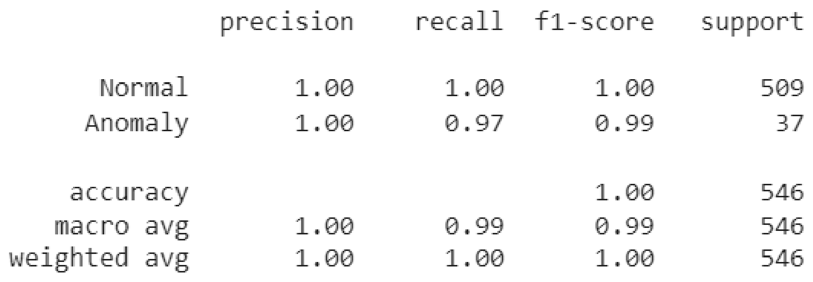


```
              precision    recall  f1-score   support

      Normal       1.00      0.95      0.97       509
     Anomaly       0.59      1.00      0.74        37

    accuracy                           0.95       546
   macro avg       0.79      0.97      0.86       546
weighted avg       0.97      0.95      0.96       546
```

Fig. 34. Naïve Bayes Classification Report for Unsupervised labelling



```
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00       509
     Anomaly       1.00      0.97      0.99        37

    accuracy                           1.00       546
   macro avg       1.00      0.99      0.99       546
weighted avg       1.00      1.00      1.00       546
```

Fig. 35. SVM Classification Report for Unsupervised labelling

6. Conclusion and Future Work

The cloud monitoring community paradigm has given a lot of attention to machine learning approaches. Using machine learning techniques, this research presented how to identify anomalies in this work, studied the performance indicators for container-based microservices, and finally proposed methods to detect anomalies using machine learning for microservices. It uses various algorithms of machine learning to categorise abnormal and expected behaviours, and performance monitoring data from services and containers. This work explored detecting anomalies using both supervised and unsupervised techniques. The results of the trials showed that supervised learning techniques performed noticeably better than unsupervised ones. Also Compared to Naïve Bayes classifier remaining all classifiers used showed good accuracy for both supervised and unsupervised models.

Future research will focus on a more typical case study of microservice architecture. It focuses deeper on faults, and it types and also how to rectify them. This work lacks to perform real time online. Future work targets to achieve this work in online phase.

Acknowledgments First and foremost, I thank God Almighty for everything within my master's programme. Secondly, I am grateful to Prof. Rajiv Ranjan for his supervision and the insights which he gave me throughout the course of the project. Indeed, working on this project has helped me learn a tonne and provided me valuable skills. I have thoroughly enjoyed the project and look forward to continuing to do more research in the field of machine learning in the future, especially for Cloud Monitoring/Anomaly detection.

Lastly (but definitely not least), I must thank my family, friends, especially my fiancé for putting up with me during another year of study (after a long gap). Their help has been greatly valued and appreciated from start to end during this year.

References

1. P. Mell, T. Grance, The NIST Definition of Cloud Computing, NIST Special Publication 800-145, 2011.
2. M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. R abkin, I. Stoica, M. Zaharia, A view of cloud computing Commun. ACM, 53 (4) (2010), pp. 50-58
3. Aceto G, Botta A, De Donato W, Pescapè A (2013) Cloud monitoring: a survey. Comput Netw 57(9):2093–2115
4. M. Natu, R. K. Ghosh, R. K. Shyamsundar and R. Ranjan, "Holistic Performance Monitoring of Hybrid Clouds: Complexities and Future Directions," in IEEE Cloud Computing, vol. 3, no. 1, pp. 72-81, Jan.-Feb. 2016, doi: 10.1109/MCC.2016.13.
5. A. Noor et al., "A Framework for Monitoring Microservice-Oriented Cloud Applications in Heterogeneous Virtualization Environments," 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 2019, pp. 156-163, doi: 10.1109/CLOUD.2019.00035.
6. Singh, V., et al.: Container-based microservice architecture for cloud applications. In: Computing, Communication and Automation (ICCCA) (2017)
7. Sauvanaud, C., et al.: Anomaly detection and diagnosis for cloud services: practical experiments and lessons learned. J. Syst. Softw. 139, 84–106 (2018)
8. A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao and F. Liu, "Evaluating machine learning algorithms for anomaly detection in clouds," 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 2716-2721, doi: 10.1109/BigData.2016.7840917.
9. Du, Q., Xie, T., He, Y. (2018). Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring. In: Vaidya, J., Li, J. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2018. Lecture Notes in Computer Science (), vol 11337. Springer, Cham. https://doi.org/10.1007/978-3-030-05063-4_42
10. Rusek, M., Dwornicki, G. and Orłowski, A., 2016, September. A decentralized system for load balancing of containerized microservices in the cloud. In International Conference on Systems Science (pp. 142-152). Springer, Cham.
11. Kratzke, N.: About microservices, containers and their underestimated impact on network performance. arXiv preprint arXiv:1710.04049(2017) (2017)
12. Chappell, D. (2015). *Introducing azure machine learning: A guide for technical professionals*. San Francisco, CA: Chappell & Associates.
13. Gupta, A., Mehrotra, K.G. and Mohan, C., 2010. A clustering-based discretization for supervised learning. Statistics & probability letters, 80(9-10), pp.816-824.
14. Gupta, P., Sehgal, N.K. (2021). Machine Learning Algorithms. In: Introduction to Machine Learning in the Cloud with Python. Springer, Cham. https://doi.org/10.1007/978-3-030-71270-9_2
15. Chandola, V., Banerjee, A. and Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), pp.1-58.
16. Wang, T., Zhang, W., Ye, C., Wei, J., Zhong, H. and Huang, T., 2015. FD4C: Automatic fault diagnosis framework for Web applications in cloud computing. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46(1), pp.61-75.
17. Omar, Salima, Asri Ngadi, and Hamid H. Jebur. "Machine learning techniques for anomaly detection: an overview." *International Journal of Computer Applications* 79.2 (2013).
18. H. Won and Y. Kim, "Performance Analysis of Machine Learning Based Fault Detection for Cloud Infrastructure," 2021 International Conference on Information Networking (ICOIN), 2021, pp. 877-880, doi: 10.1109/ICOIN50884.2021.9333875.
19. S. Fu, J. Liu, and H. Pannu, "A Hybrid Anomaly Detection Framework in Cloud Computing Using One-Class and Two-Class Support Vector Machines," in Proc. of the 8th IEEE International Conference on Advanced Data Mining and Applications, Dec. 2012, pp. 726–738.

20. L. Cardoso Silva et al., "Benchmarking Machine Learning Solutions in Production," 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), 2020, pp. 626-633, doi: 10.1109/ICMLA51294.2020.00104.
21. Kotsiantis, S.B., Kanellopoulos, D. and Pintelas, P.E., 2006. Data preprocessing for supervised leaning. International journal of computer science, 1(2), pp.111-117.
22. Kumari, R., Singh, M.K., Jha, R. and Singh, N.K., 2016, March. Anomaly detection in network traffic using K-mean clustering. In 2016 3rd international conference on recent advances in information technology (RAIT) (pp. 387-393). IEEE.
23. https://www.ibm.com/cloud/learn/multicloud
24. https://www.kaggle.com/code/vipulgandhi/gaussian-mixture-models-clustering-explained/notebook
25. https://www.mghassany.com/MLcourse/gaussian-mixture-models-em.html
26. https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-it-is-important-98a98db0961d.
27. https://data-flair.training/blogs/svm-kernel-functions/.
28. https://medium.com/@aravanshad/how-to-choose-machine-learning-algorithms-9a92a448e0df.
29. https://en.wikipedia.org/wiki/Feature_scaling.