INTERNSHIP REPORT

A report submitted in partial fulfilment of the requirements for the Award of Degree of

BACHELOR OF ENGINEERING
in
COMPUTER ENGINEERING

by Student
Kalpesh Patil
(72147797G)
Jayesh Chaudhari
(72147805J)
Atharv Khamkar
(72147706C)
Vivek Rodge
(72147830B)

Under Supervision of
Prof. Deepali. D. Ahir
(Duration: 1 month)

DEPARTMENT OF COMPUTER ENGINEERING

Modern Education Society's College of Engineering, Pune
Approved by AICTE,

Affiliated to SPPU, Pune

Maharashtra

DEPARTMENT OF COMPUTER ENGINEERING

MODERN EDUCATION SOCIETY'S COLLEGE

OF

ENGINEERING, PUNE

CERTIFICATE

This is to certify that the **Internship report** submitted by **Kalpesh Ananda Patil (University PRN Number: 72147797G)** is work done by him and submitted during 2022-23 academic year, in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF ENGINEERING in COMPUTER ENGINEERING,** at MES College of Engineering, Pune**.**

Dr.(Mrs.) Sharmila Wagh

College Internship Coordinator

Dr.(Mrs.) S.P.Deore

Department Internship Coordinator

Dr.(Mrs.) N.F.Shaikh

Head of theDepartment
Department of Computer Engineering

DEPARTMENT OF COMPUTER ENGINEERING

MODERN EDUCATION SOCIETY'S COLLEGE

OF

ENGINEERING, PUNE

CERTIFICATE

This is to certify that the **Internship report** submitted by **Jayesh Nandu Chaudhari (University PRN Number: 7214777605J)** is work done by him and submitted during 2021-22 academic year, in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF ENGINEERING in COMPUTER ENGINEERING,** at MES College of Engineering, Pune**.**

Dr.(Mrs.) Sharmila Wagh

College Internship Coordinator

Dr.(Mrs.) S.P.Deore

Department Internship Coordinator

Dr.(Mrs.) N.F.Shaikh

Head of the Department
Department of Computer Engineering

DEPARTMENT OF COMPUTER ENGINEERING
MODERN EDUCATION SOCIETY'S COLLEGE
OF
ENGINEERING, PUNE

CERTIFICATE

This is to certify that the **Internship report** submitted by **Atharv Gurudas Khamkar (University PRN Number: 72147706C)** is work done by him and submitted during 2022-23 academic year, in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF ENGINEERING in COMPUTER ENGINEERING,** at MES College of Engineering, Pune**.**

Dr.(Mrs.) Sharmila Wagh
College Internship Coordinator

Dr.(Mrs.) S.P.Deore
Department Internship Coordinator

Dr.(Mrs.) N.F.Shaikh
Head of the Department
Department of Computer Engineering

DEPARTMENT OF COMPUTER ENGINEERING
MODERN EDUCATION SOCIETY'S COLLEGE
OF
ENGINEERING, PUNE

CERTIFICATE

This is to certify that the **Internship report** submitted by **Vivek Sudamrao Rodge (University PRN Number: 72147830B)** is work done by him and submitted during 2022-23 academic year, in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF ENGINEERING in COMPUTER ENGINEERING,** at MES College of Engineering, Pune**.**

Dr.(Mrs.) Sharmila Wagh
College Internship Coordinator

Dr.(Mrs.) S.P.Deore
Department Internship Coordinator

Dr.(Mrs.) N.F.Shaikh
Head of theDepartment
Department of Computer Engineering

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude towards my internship guide for her support, continuous guidance, and for being so understanding and helpful throughout the internship. I have furthermore to thank Computer Department HOD Dr. (Mrs.) N. F. Shaikh to encourage me to go ahead and for continuous guidance. I also want to thank Dr. (Mrs.) S.K. Wagh and Prof. Deepali. D. Ahir for all her assistance and guidance in preparing the report. I would like to thank all those who have directly or indirectly helped me with the completion of the work during this internship.

Student Name
Kalpesh Ananda Patil
Jayesh Nandu Chaudhari
Atharv Gurudas Khamkar
Vivek Sudamrao Rodge

INDEX

DIAGRAM

General guidelines and instructions:

Internships are educational and career development opportunities, providing practical experience in a field or discipline. Internships are far more important as the employers are looking for employees who are properly skilled and having awareness about industry environment, practices and culture. Internship is structured, short-term, supervised training often focused around particular tasks or projects with defined time scales.
Core objective is to expose technical students to the industrial environment, which cannot be simulated/experienced in the classroom and hence creating competent professionals in the industry and to understand the social, economic and administrative considerations that influence the working environment of industrial organizations.
Engineering internships are intended to provide students with an opportunity to apply conceptual knowledge from academics to the realities of the field work/training. The following guidelines are proposed to give academic credit for the internship undergone as a part of the Third Year Engineering curriculum.

Duration:

Internship is to be completed after semester 5 and before commencement of semester 6 of at least 4 to 6 weeks; and it is to be assessed and evaluated in semester 6.

Internship work Identification:

Student may choose to undergo Internship at Industry/Govt.
Organizations/NGO/MSME/Rural Internship/ Innovation/IPR/Entrepreneurship.
Student may choose either to work on innovation or entrepreneurial activities resulting in start-up or undergo internship with industry/NGO's/Government organizations/Micro/Small/ Medium enterprises to make themselves ready for the industry [1].
Students must register at Internshala [2].

Reference:

[1] https://www.aicte-india.org/sites/default/files/AICTE%20Internship%20Policy.pdf
[2] https://internship.aicte-india.org/

Internship Course Objectives:

Internship provides an excellent opportunity to learner to see how the conceptual aspects learned in classes are integrated into the practical world. Industry/on project experience provides much more professional experience as value addition to classroom teaching.

To encourage and provide opportunities for students to get professional/personal experience through internships.
To learn and understand real life/industrial situations.

To get familiar with various tools and technologies used in industries and their applications.
To nurture professional and societal ethics.
To create awareness of social, economic and administrative considerations in the working environment of industry organizations.
To highlight the talents, you already have in the field as well as your desire to learn more.

Course Outcomes:

On completion of the course, learners should be able to:

CO1: To demonstrate professional competence through industry internship.
CO2: To apply knowledge gained through internships to complete academic activities in a professional manner.
CO3: To choose appropriate technology and tools to solve given problem.
CO4: To demonstrate abilities of a responsible professional and use ethical practices in day to day life.
CO5: Creating network and social circle, and developing relationships with industry people.
CO6: To analyse various career opportunities and decide carrier goals.

WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
FIRST WEEK	24/04/2023	Monday	Topic finalization
	25/04/2023	Tuesday	Searched different research papers on topic
	26/04/2023	Wednesday	Searched different research papers on topic
	27/04/2023	Thursday	Studied different literature surveys or research papers. to compare algorithms and choose an appropriate algorithm for implementation.
	28/04/2023	Friday	Studied different literature surveys or research papers. to compare algorithms and choose an appropriate algorithm for implementation.
	29/04/2023	Saturday	Selection of algorithm on the basis of comparison and requirement

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
SECOND WEEK	01/05/2023	Monday	Finalize the algorithm
	02/05/2023	Tuesday	Understand the requirement and collected required information
	03/05/2023	Wednesday	Understand the requirement and collected required information
	04/05/2023	Thursday	Understand the working of isolation forest regression
	05/05/2023	Friday	Created dataset
	06/05/2023	Saturday	Created dataset

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
THIRD WEEK	08/05/2023	Monday	Pre-processed dataset
	09/05/2023	Tuesday	Implementation on small dataset of 20 rows to check accuracy and working
	10/05/2023	Wednesday	Implementation on another small dataset to check accuracy and working
	11/05/2023	Thursday	Implementation on real-time dataset of to check accuracy and working
	12/05/2023	Friday	Implementation on real-time dataset of to check accuracy and working
	13/05/2023	Saturday	Solving errors

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
FOURTH WEEK	15/05/2023	Monday	Solving errors to increase accuracy
	16/05/2023	Tuesday	Final implementation or testing
	17/05/2023	Wednesday	Final implementation
	18/05/2023	Thursday	Documentation and report writing
	19/05/2023	Friday	Documentation and report writing

ABSTRACT

Anomaly detection is an important problem that has been researched within diverse research areas and application domains. Many anomaly detection techniques have been specifically developed for certain application domains, while others are more generic. This report tries to provide a structured and comprehensive overview of the ANOMALY DETECTION on CPU UTILIZATION, MEMORY UTILIZATION. We have grouped existing techniques into different categories based on the underlying approach adopted by each technique. For each category we have identified key assumptions, which are used by the techniques to differentiate between normal and anomalous behaviour. When applying a given technique to a particular dataset, these assumptions can be used as guidelines to assess the effectiveness of the technique in that domain. For MEMORY and CPU usage dataset, we provide a isolation forest anomaly detection technique. This template provides an easier and succinct understanding of the isolation forest technique for utilization dataset. Further, for each category, we identify the advantages and disadvantages of the various techniques in that category. We also provide a discussion on the computational complexity of the techniques since it is an important issue in real application domains. We hope that this report will provide a better understanding of the different directions of anomaly detection and best technique for anomaly detection in which research has been done on this topic.

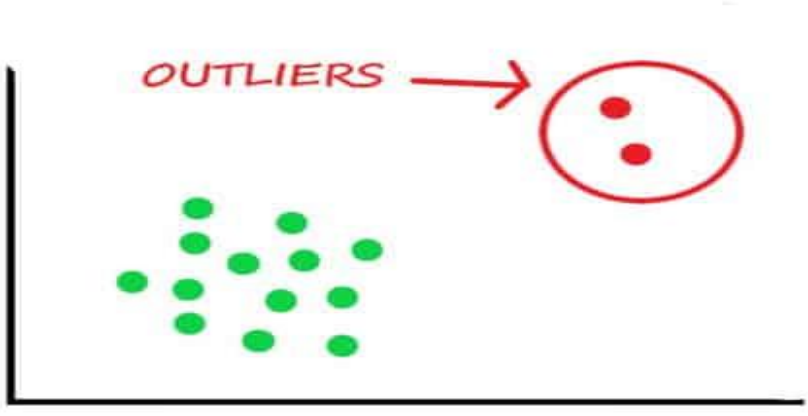
1.Introduction

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behaviour. These non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection; sometimes interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities. The importance of anomaly detection is due to the fact that anomalies in data translate to significant (and often critical) actionable information in a wide variety of application domains.

Detecting outliers or anomalies in data has been studied in the statistics community as early as the 19th century. Over time, a variety of anomaly detection techniques have been developed in several research communities. Many of these techniques have been specifically developed for certain application domains, while others are more generic. This REPORT tries to provide a structured and comprehensive overview of the research on anomaly detection. We hope that it facilitates a better understanding of the different directions in which research has been done on this topic, and how techniques developed in one area can be applied in domains for which they were not intended to begin with.

What are anomalies?

Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior. Figure 1 illustrates anomalies in a simple 2-dimensional data set. The data has two normal regions, N1 and N2, since most observations lie in these two regions. Points that are sufficiently far away from the regions, e.g., points o1 and o2, and points in region O3, are anomalies



Anomalies might be induced in the data for a variety of reasons, such as malicious activity, e.g., credit card fraud, cyber-intrusion, terrorist activity or breakdown of a system, but all of the reasons have a common characteristic that they are interesting to the analyst. The

“interestingness” or real-life relevance of anomalies is a key feature of anomaly detection.
Noise can be defined as a phenomenon in data which is not of interest to the analyst, but acts as a hindrance to data analysis. Noise removal is driven by the need to remove the unwanted objects before any data analysis is performed on the data. Noise accommodation refers to immunizing a statistical model estimation against anomalous observations

Challenges

At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behavior. A straightforward anomaly detection approach, therefore, is to define a region representing normal behavior and declare any observation in the data which does not belong to this normal region as an anomaly. But several factors make this apparently simple approach very challenging:

Defining a normal region which encompasses every possible normal behavior is very difficult. In addition, the boundary between normal and anomalous behavior is often not precise. Thus an anomalous observation which lies close to the boundary can actually be normal, and vice-versa.
When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the anomalous observations appear like normal, thereby making the task of defining normal behavior more difficult
In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future.
The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus, applying a technique developed in one domain to another is not straightforward.
Availability of labeled data for training/validation of models used by anomaly detection techniques is usually a major issue

Due to the above challenges, the anomaly detection problem, in its most general form, is not easy to solve. In fact, most of the existing anomaly detection techniques solve a specific formulation of the problem

Our Contributions

This report is an attempt to provide a structured and a broad overview of extensive research on anomaly detection techniques spanning multiple research areas and best suitable technique for anomaly detection on CPU usage dataset. This report we have analyse various technique such as knn,svm,lof,isolation forest. For each of the categories, we not only discuss the techniques, but also identify unique assumptions regarding the nature of anomalies made by the techniques in that category. These assumptions are critical for determining when the techniques in that category would be able to detect anomalies, in the CPU and MEMORY usage dataset.

2.Problem statement

Anomaly/outlier detection on CPU and MEMORY usage dataset using best suitable techniques.

2.1 Objectives

- Generate a CPU UTILIZATION AND MEMORY UTILIZATION dataset.
- Dataset consist of reading of CPU and memory utilization after every 5 seconds for around 30 minutes.
- Select best suitable technique for anomaly detection in above dataset.
- Calculate anomalies present in dataset.
- Give accuracy for used technique

3. Motivation

The motivation for anomaly detection using the most suitable model for CPU utilization and CPU usage in each timestamp lies in the need to ensure optimal system performance, resource allocation, and proactive maintenance. CPU utilization and usage are crucial metrics for evaluating system performance. By detecting anomalies in these metrics, it becomes possible to identify periods of excessive CPU load or underutilization. This information can be used to optimize system performance, allocate resources more efficiently, and identify potential bottlenecks or inefficiencies. Anomalies in CPU metrics can be early indicators of system issues or failures. By monitoring CPU utilization and usage patterns, it becomes possible to detect deviations from normal behaviour that may signify hardware malfunctions, software bugs, or other system-related problems. Timely detection of anomalies allows for proactive maintenance, preventing system failures and reducing downtime. Understanding the patterns and trends in CPU utilization and usage is essential for capacity planning. By analysing historical data and detecting anomalies, it becomes possible to forecast future resource requirements accurately.

Anomaly detection models can help identify sudden spikes or unusual patterns that may require additional computational resources or adjustments in the system's capacity. In cloud environments, where resources are dynamically allocated and shared among multiple users, anomaly detection in CPU metrics is crucial. By monitoring CPU utilization and usage, anomalies can be identified that may indicate resource contention, improper resource allocation, or malicious activities. Detecting anomalies in real-time allows for prompt action to ensure fair resource distribution, optimize cost-effectiveness, and maintain the desired performance levels. Anomalies in CPU metrics can also be indicative of security breaches or intrusion attempts. For example, sudden spikes in CPU usage may indicate malicious activities such as crypto-mining malware or denial-of-service (DoS) attacks. By employing anomaly detection models, such abnormal patterns can be identified, and appropriate security measures can be implemented to mitigate the risks. The most suitable model for CPU anomaly detection should be scalable and adaptable to handle large-scale systems with varying workloads. It should be able to process and analyse high-frequency data streams efficiently and provide real-time anomaly detection. The model should also be flexible enough to accommodate changes in the system environment, such as software updates, hardware upgrades, or changes in workload patterns.

4.TECHNIQUES

1.Support Vector Machine (SVM)

Support vector machines are a set of supervised learning methods used for classification, regression, and outliers' detection. All of these are common tasks in machine learning. You can use them to detect cancerous cells based on millions of images or you can use them to predict future driving routes with a well-fitted regression model. A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from.

1.1 Uses of Support Vector Machine:
SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data. Effective on datasets with multiple features, like financial or medical data.
Uses a subset of training points in the decision function called support vectors which makes it memory efficient.

1.2 Reasons why the SVM is not best algorithm:
It's a great algorithm when you are working with smaller datasets that have tens to hundreds of thousands of features.
They typically find more accurate results when compared to other algorithms because of their ability to handle small, complex datasets. SVM is not suitable for modules having large dataset.
It has high training time to train the modules, so it is not time efficient.
SVMs don't directly provide probability estimates. Those are calculated using an expensive five-fold cross-validation.

2.Local Outlier Factor (LOF)

Local Outlier Factor (LOF) is an unsupervised machine learning algorithm used for outlier detection. It measures the local density deviation of a data point compared to its neighbours. LOF provides a score that indicates the degree of abnormality of each data point. LOF assumes that outliers have a lower density compared to their neighbours. It measures the degree to which a data point deviates from the local neighbourhood density. Outliers are identified as data points with significantly lower density compared to their neighbours.
LOF uses a density-based approach to detect outliers. It focuses on the local density of data points rather than global properties. The algorithm considers each data point's neighbourhood and compares its density with that of its neighbours.

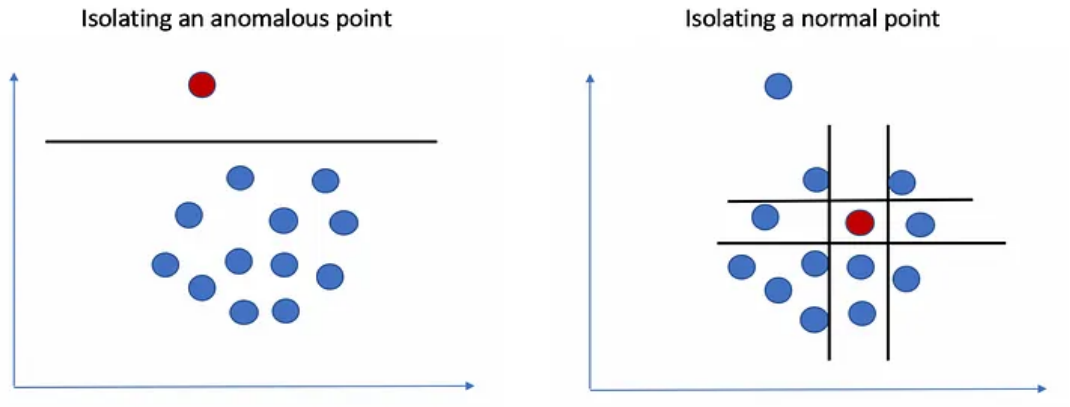
2.1 Uses and advantages of Local Outlier Factor:
LOF is a useful algorithm for detecting outliers in various domains such as fraud detection, network intrusion detection, and anomaly detection. It provides a local perspective on outlier detection, allowing for the identification of anomalies within specific contexts.
A point will be considered as an outlier if it is at a small distance to the extremely dense cluster. The global approach may not consider that point as an outlier. But the LOF can effectively identify the local outliers.

2.2 Reasons why the Local Outlier Factor is not best algorithm:
The algorithm's computational complexity increases with the size of the dataset. It is not suitable for modules having large size of dataset.

Local Outlier Factor may not perform well in datasets with uniform densities or when outliers are present within dense regions.
Since LOF is a ratio, it is tough to interpret. There is no specific threshold value above which a point is defined as an outlier. The identification of an outlier depends on the problem and the user.

3.Isolation Forest

Isolation Forest is an anomaly detection algorithm that isolates anomalies or outliers in a dataset. It is particularly effective in detecting anomalies in high-dimensional data and is based on the concept of random forests. Isolation Forest works by randomly selecting a feature and then randomly selecting a split value between the minimum and maximum values of that feature. This process is repeated recursively to create isolated trees. The depth of each tree is typically set to the average path length. The Isolation Forest Algorithm takes advantage of the following properties of anomalous samples (often referred to as outliers):
Fewness - anomalous samples are a minority and there will only be a few of them in any dataset.
Different - anomalous samples have values/attributes that are very different from those of normal samples
These two properties make it easier to isolate anomalous samples from the rest of the data in comparison to normal points.



we can isolate an anomalous point from the rest of the data with just one line, while the normal point on the right requires four lines to isolate completely.

3.1 Algorithm:

Given a sample of data points X, the Isolation Forest algorithm builds an Isolation Tree (iTree), T, using the following steps.
1.Randomly select an attribute q and a split value p.
2.Divide X into two subsets by using the rule q < p. The subsets will correspond to a left subtree and a right subtree in T.
3.Repeat steps 1–2 recursively until either the current node has only one sample or all the values at the current node have the same values.

The algorithm then repeats steps 1–3 multiple times to create several Isolation Trees, producing an Isolation Forest. Based on how Isolation Trees are produced and the properties of anomalous points, we can say that most anomalous points will be located closer to the root of the tree since they are easier to isolate when compared to normal points.

Why the Isolation Forest is best algorithm:

Utilizes no distance or density measures to detect anomalies. This eliminates major computational cost of distance calculation in all distance-based methods and density-based methods.
Capacity to scale up to handle extremely large data size and high-dimensional problems with a large number of irrelevant attributes.
It doesn't require assumptions about data distribution and works well for both global and local outlier detection.
The algorithm is robust against irrelevant features and can handle datasets with mixed variable types.
Isolation Forest provides an anomaly score that indicates the level of abnormality for each data point.
Isolation forest also has some disadvantages like It may struggle with detecting outliers in datasets with overlapping clusters or when the outliers are within the majority cluster. But the Isolation Forest algorithm is suitable for large sizes of dataset. It is Scalable and efficient, so we have used isolation forest regression for implementation.

5.Software requirements specifications

5.1. System configuration

Software Requirements:

Operating system : Windows
Coding Language : Python
Software : Jupyter Notebook

Hardware Requirement:

System : PC
Hard Disk : 500 GB
Ram : 8 GB

6. Methodology

6. 1: Dataset Generation:

This is the first step in implementation to collect data.

6.1.1. Function: 'collect_data ()' :

This function is designed to gather information about CPU and memory consumption. To obtain the required data, it makes use of the psutil library. The psutil.cpu_percent() method returns the percentage of the current CPU use. The psutil.virtual_memory().percent function also returns the percentage of current memory consumption. The function uses time.time() to produce a timestamp that displays the current time in Unix time format.

The gathered information is then arranged into a list with the elements [timestamp, cpu_usage, memory_usage], where cpu_usage denotes the proportion of CPU usage and memory_usage denotes the percentage of memory usage. This list is the output that the function finally returns.

6.1.2. Data Collection:

In order to generate a CSV file and store the collected CPU and memory usage data, the provided function incorporates additional steps. First, the open() function is utilized to create a new CSV file named "cpu_memory_data.csv" in write mode. This file will be used to store the data points. To facilitate writing data to the CSV file, a csv.writer object is created.

The column names, including "timestamp", "cpu_utilization", and "memory_usage", are written as the first row of the CSV file using the writer.writerow() method. This ensures that the column names are clearly labeled.

The function proceeds to collect data every 5 seconds for a duration of 1 hour, resulting in a total of 360 iterations. Within each iteration, the current timestamp, CPU utilization percentage, and memory usage percentage are gathered using relevant functions such as time.time(), psutil.cpu_percent(), and psutil.virtual_memory().percent, respectively. This information is then appended as a sublist to the data list.

To ensure a consistent interval between data collection intervals, the function introduces a 5-second delay using time.sleep(5).

Once the loop completes its iterations and the data is collected, the function writes the data to the CSV file using the writer.writerows(data) method. Each sublist within the data list represents a row in the CSV file, with each element in the sublist corresponding to a column value.

By executing this updated function, you will be able to automatically collect CPU and memory usage data every 5 seconds for a duration of 1 hour. The collected data will be stored in a CSV file named "cpu_memory_data.csv" with appropriate column names, facilitating further analysis or visualization of the data.

6.1.3. Data Storage:

To generate a CSV file and store the collected CPU and memory usage data, the function utilizes the open() function with the write mode to create a new file named "cpu_memory_data.csv". This file will serve as the destination for storing the data points. In order to facilitate the writing process, a csv.writer object is created.

The first row of the CSV file is reserved for column names. Therefore, the function writes the column names "timestamp", "cpu_utilization", and "memory_usage" to the CSV file using the writer.writerow() method. This ensures that the CSV file contains clear and identifiable column headers.

After writing the column names, the function proceeds to collect the data points. For each data collection interval, the current timestamp, CPU utilization percentage, and memory usage percentage are gathered. The data is appended as a sublist to the data list.

Once all the required data is collected, the function employs the writer.writerows(data) method to write the contents of the data list to the CSV file. Each sublist within the data list represents a row in the CSV file, with the elements within the sublist corresponding to the values of the respective columns.

By executing this updated function, you will generate a CSV file named "cpu_memory_data.csv" that contains the collected CPU and memory usage data. The file will include the appropriate column names and the corresponding data points, making it suitable for further analysis or processing using CSV-compatible tools and applications.

In [3]:
```
import psutil
import time
import csv

# Create a function to collect CPU and memory usage
def collect_data():
    cpu_usage = psutil.cpu_percent()
    memory_usage = psutil.virtual_memory().percent
    timestamp = int(time.time())
    return [timestamp, cpu_usage, memory_usage]

# Create an empty list to store the data
data = []

# Collect data every 5 seconds for 1 hour
for i in range(360):
    data.append(collect_data())
    time.sleep(5)

# Save the data to a CSV file
with open('cpu_memory_data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['timestamp', 'cpu_utilization', 'memory_usage'])
    writer.writerows(data)
```

In [1]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("cpu_memory_data.csv")
```

In [3]:
```
df.head(20)
```

Out[3]:

	timestamp	cpu_utilization	memory_usage
0	1683633162	2.40	81.8
1	1683633167	2.40	80.7
2	1683633172	1.60	81.2
3	1683633177	3.60	81.1
4	1683633182	5.70	80.7
5	1683633187	3.90	81.8
6	1683633192	0.90	97.0
7	1683633197	6.50	83.0
8	1683633202	2.40	73.8
9	1683633207	1.90	73.2
10	1683633212	8.80	73.2

6. 2. Implementation:

First, the required libraries are imported: pandas as pd and numpy as np. These libraries provide functionality for data manipulation and numerical operations, respectively.

Next, the code reads the CPU and memory data from the CSV file named "cpu_memory_data.csv" and stores it in a pandas DataFrame called df using the pd.read_csv() function.

Then, the Isolation Forest model is imported from the scikit-learn library with the line from sklearn.ensemble import IsolationForest. This model is used for anomaly detection based on the Isolation Forest algorithm.

A random number generator object is created using np.random.RandomState(10) to ensure reproducibility of random numbers generated later in the code. The seed value of 10 is used to initialize the random number generator.

An instance of the Isolation Forest model is created with specific parameters. The n_estimators parameter is set to 100, indicating the number of trees in the Isolation Forest. The max_samples parameter is set to 'auto', which means the entire dataset will be used for each tree. The contamination parameter is set to 0.035, representing the expected proportion of anomalies in the data. Finally, the random_state parameter is set to the random number generator object created earlier to ensure consistent results.

The Isolation Forest model is fitted to the columns 'timestamp', 'cpu_utilization', and 'memory_usage' of the DataFrame df using the model.fit() method. This step trains the model on the provided data to learn patterns and detect anomalies.

The model.get_params() function is called to retrieve and print the parameters of the trained Isolation Forest model. This provides information about the configuration of the model, including the number of trees, maximum samples, contamination level, and random state.

A new column named 'scores' is added to the DataFrame df, and the model.decision_function() method is used to calculate the anomaly scores for each data point in the specified columns. These scores represent the degree of anomaly, with higher values indicating a higher likelihood of being anomalous.

Another new column named 'anomaly_score' is added to the DataFrame df, and the model.predict() method is used to predict the label (anomaly or normal) for each data point in the specified columns. Anomalous data points are assigned a value of -1, while normal data points are assigned a value of 1.

The code then filters the DataFrame df to display the first 13 rows where the 'anomaly_score' column has a value of -1, indicating anomalies. This is achieved with the line df[df['anomaly_score']==-1].head(13). The displayed rows represent the data points that the Isolation Forest model identified as anomalous.

Finally, the code uses df[df['anomaly_score']==-1].count() to count the number of rows in the DataFrame df where the 'anomaly_score' column has a value of -1. This provides the total count of anomalous data points detected by the Isolation Forest model.

In [164]:
```
from sklearn.ensemble import IsolationForest
random_state = np.random.RandomState(10)
model=IsolationForest(n_estimators=100,max_samples='auto',contamination=float(0.035)
                      ,random_state=random_state)

model.fit(df[["timestamp","cpu_utilization","memory_usage"]])

print(model.get_params())
```

```
{'bootstrap': False, 'contamination': 0.035, 'max_features': 1.0, 'max_samples': 'auto', 'n_estimators': 100, 'n_jobs': None,
'random_state': RandomState(MT19937) at 0x1C3E298BA40, 'verbose': 0, 'warm_start': False}
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but Isolation
Forest was fitted with feature names
  warnings.warn(
```

In [168]:
```
df['scores'] = model.decision_function(df[["timestamp","cpu_utilization","memory_usage"]])

df['anomaly_score'] = model.predict(df[["timestamp","cpu_utilization","memory_usage"]])

df[df['anomaly_score']==-1].head(13)
```

7.Results / Analysis

7.1. Result:

The Isolation Forest algorithm is applied to the collected CPU and memory usage data. The algorithm detected 1 anomaly in the dataset of 360 rows, which has "cpu_utilization" = 63.4 and ‘memory_usage’ = 91.0 .and shows all those outliers in the anamoly_score column using ‘-1’ and normal points with ‘1," and it gives more accuracy over other anomaly detection algorithms like knn and lof. Isolation forest regression identifies anomalies or outliers in a dataset based on their dissimilarity from the majority of the data points.

```
In [79]:
df['scores'] = model.decision_function(df[["cpu_utilization","memory_usage"]])

df['anomaly_score'] = model.predict(df[["cpu_utilization","memory_usage"]])

df[df['anomaly_score']==-1].head(13)
```

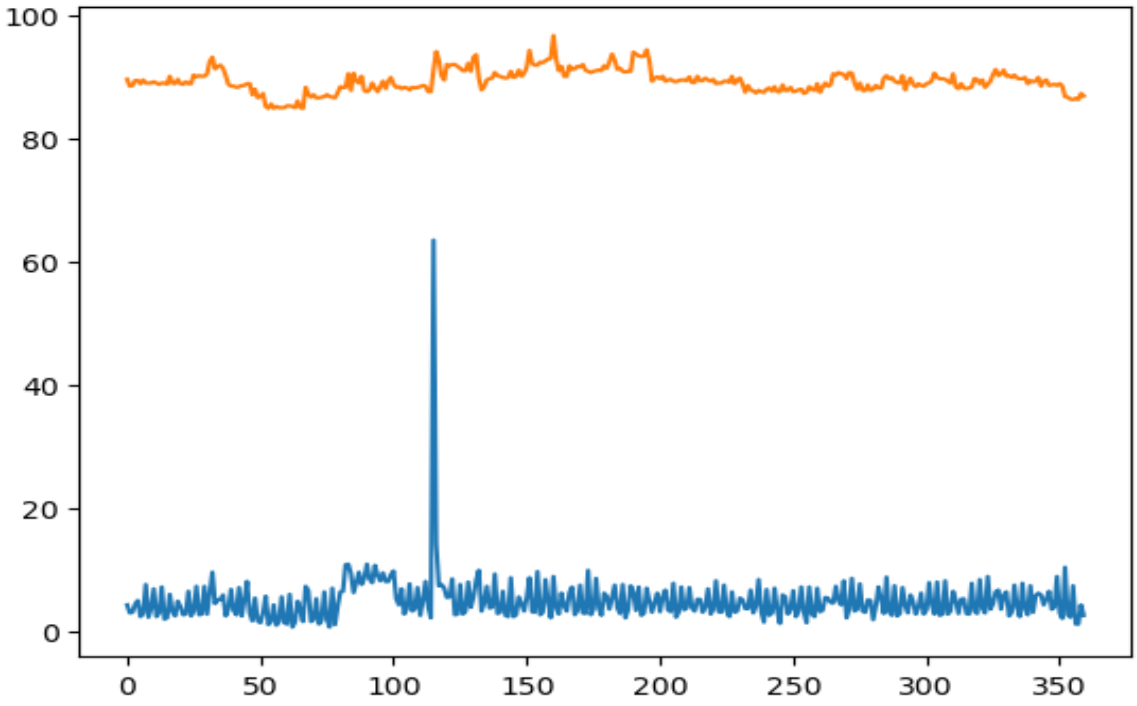
Out[79]:

	timestamp	cpu_utilization	memory_usage	scores	anomaly_score
115	1684482892	63.4	91.0	-0.027758	-1

```
In [80]: df[df['anomaly_score']==-1].count()
```

```
Out[80]: timestamp          1
         cpu_utilization    1
         memory_usage       1
         scores             1
         anomaly_score      1
         dtype: int64
```

Graphical representation:
Here we can see that at data point 115(row number in dataset) suddenly CPU utilization get increased upto 63.4



To check the accuracy of isolation forest regression, before implementing that algorithm on a large dataset, we test it on small datasets of 9 rows, in which we insert 3 outliers, and this model also calculates outliers to be 3. It has 100 percent accuracy. The output of the isolation forest model over that small dataset is as follows:

In [8]:
```
data
```

Out[8]:

	student_id	marks	oral_marks	scores	anomaly_score
0	1	90	90	0.152469	1
1	2	450	90	-0.043618	-1
2	3	398	80	-0.038945	-1
3	4	89	99	0.016691	1
4	5	95	85	0.114836	1
5	6	98	88	0.138991	1
6	7	99	89	0.138609	1
7	8	68	90	0.049725	1
8	9	88	78	0.025079	1
9	10	409	300	-0.190255	-1

Model Evaluation:

In [9]:
```
anomaly_count  = 3
accuracy = 100*list(data['anomaly_score']).count(-1)/(anomaly_count)
print("Accuracy of the model:", accuracy)
```

```
Accuracy of the model: 100.0
```

7.2. Analysis:

Anomaly detection using the Isolation Forest algorithm can be useful for identifying unusual patterns or outliers in the CPU and memory usage data.

The Isolation Forest algorithm works by constructing isolation trees and isolating anomalies based on shorter average path lengths in the tree structures.

The detected anomalies could represent instances where the CPU and memory usage deviated significantly from the expected behavior.

These anomalies might indicate potential issues such as abnormal resource consumption, performance bottlenecks, or system malfunctions.

By identifying and investigating these anomalies, system administrators or analysts can take appropriate actions to address any underlying problems.

The Isolation Forest algorithm is advantageous for anomaly detection as it is capable of handling high-dimensional datasets and does not require assumptions about the underlying data distribution.

However, it is essential to interpret the detected anomalies in the context of the specific system or application being monitored. Some anomalies may be expected behavior depending on the system's characteristics or workload patterns.

Further analysis and domain expertise are required to determine the significance and potential impact of the detected anomalies.

It is recommended to monitor the system continuously and apply anomaly detection algorithms regularly to ensure timely identification of any unusual behavior

8.Conclusion

In conclusion, the isolation forest algorithm is a powerful and effective method for detecting anomalies in datasets. It offers several advantages over traditional outlier detection techniques, such as its ability to handle high-dimensional data and its computational efficiency.

The isolation forest algorithm has been shown to be particularly effective in scenarios where anomalies are present and distinct from normal data points. It has been successfully applied in various domains, including fraud detection, cybersecurity, and system monitoring.

However, it is important to note that the isolation forest algorithm may have limitations in certain situations. It may struggle to detect anomalies that are surrounded by normal data points or anomalies that are very close to the majority of data points. Additionally, the algorithm's performance can be impacted by the choice of hyperparameters, such as the number of trees and the subsampling size.

In summary, the isolation forest algorithm is a valuable tool for anomaly detection, offering advantages in terms of its scalability, interpretability, and ability to handle high-dimensional data. While it may have limitations in certain scenarios, it remains a popular and effective technique for identifying anomalies in various

9 .Refrences

[1] A probabilistic generalization of isolation forest by Mikhail Tokovarov, Paweł Karczmarek published at science direct.

[2] High-Dimensional and Large-Scale Anomaly Detection using a Linear One-Class SVM with Deep Learning. By Sarah M. Erfani, Sutharshan Rajasegarar, Shanika Karunasekera,Christopher Leckie. Published by Science Direct.
URL- https://www.sciencedirect.com/science/article/pii/S0031320316300267.

[3] A Review of Local Outlier Factor Algorithms for Outlier Detection in Big Data Streams. By Omar Alghushairy , Raed Alsini , Terence Soule and Xiaogang MA. Published at Research Gate.
URL- https://www.researchgate.net/publication/348057855.

[4] A Deep Clustering Algorithm based on Gaussian Mixture Model by Xianghong Lin, Xiaofei Yang and Ying Li.
URL- https://iopscience.iop.org/article/10.1088/1742-6596/1302/3/032012.

[5] Machine Learning based Improved Gaussian Mixture Model for IoT Real-Time Data Analysis by Sivadi Balakrishna, Moorthy Thirumaran, Vijender Kumar Solanki published at Research Gate.
URL- https://www.researchgate.net/publication/339711667.

[6] A Genetic-Based Incremental Local Outlier Factor Algorithm for Efficient Data Stream Processing by
Omar Alghushairy, Raed Alsini, Xiaogang Ma, Terence Soule published at Research Gate.
URL- https://www.researchgate.net/publication/341042425.

[7] Anomaly detection and characterization in spatial time series data: A cluster-centric approach by H. Izakian, W. Pedrycz published at IEEE.

[8] Isolation-based anomaly detection, ACM Transactions on Knowledge Discovery from Data by F.T. Liu, K.M. Ting, Z.-H. Zhou published at IEEE International Conference on Data Mining 2008.

[9] isolation forest: An experimental comparative analysis by P. Karczmarek, A. Kiersztyn, W. Pedrycz.

[10] Long short-term memory networks for anomaly detection in time series by P. Malhotra, L. Vig, G. Shroff, G., P. Agarwal.