PROJECT
REPORT
ON

Web Server Log Analysis

Carried
Out at

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING
ELECTRONIC CITY, BANGALORE

UNDER THE SUPERVISION OF

Mr. Abhishek Chavan

C-DAC Bangalore

Submitted By:

Abhishek Bharti(220950125001)
Abhishek Mishra(220950125002)
Abhishekh Sawner(220950125003)
Akash Mishra(220950125009)
Alshone Jose Prasad(220950125010)

PG DIPLOMA IN ADVANCED COMPUTING
C-DAC, BANGALORE

Candidate's Declaration

We hereby certify that the work being presented in the report entitled **Web Server Log Analysis**, in partial fulfillment of the requirements for the award of PG Diploma Certificate and submitted in the department of PG-DBDA of the C-DAC Bangalore, is an authentic record of our work carried out during the period, 15th September 2022 to 15th March 2023 under the supervision of **Mr. Abhishek Chavan**, C-DAC Bangalore. The matter presented in the report has not been submitted by us for the award of any degree of this or any other Institute/University.

Abhishek Bharti (220950125001)

Abhishek Mishra (220950125002)

Abhishekh Sawner (220950125003)

Akash Mishra (220950125009)

Alshone Jose Prasad (220950125010)

Counter Signed by

- - - - - - - - - - - -

ACKNOWLEDGMENT

We take this opportunity to express our gratitude to all those people who have been directly and indirectly with us during the competition of this project.

We pay thanks to Mr. Abhishek Chavan who has given guidance and a light to us during this major project. His versatile knowledge about “title name “has eased us in the critical times during the span of this Final Project.

We acknowledge here our debt to those who contributed significantly to one or more steps. We take full responsibility for any remaining sins of omission and commission.

Students Names

Abhishek Bharti (220950125001)

Abhishek Mishra (220950125002)

Abhishekh Sawner (220950125003)

Akash Mishra (220950125009)

Alshone Jose Prasad (220950125010)

CERTIFICATE

This is to certify that the work titled Web Server Log Analysis is carried out by Abhishek Bharti (220950125001), Abhishek Mishra (220950125002), Abhishekh Sawner (220950125003), Akash Mishra (220950125009), Alshone Jose Prasad (220950125010) the bonafide students of Diploma in Big Data Analytics of Centre for Development of Advanced Computing, Electronic City, Bangalore from 2nd January 2023 - 2nd March 2023. The Course End Project work is carried out under my direct supervision and completed.

Mr.
Abhishek Chavan

C-DAC #68,
Electronic City,
Bangalore - 560100, India

Abstract

Web server log data has exponentially increased as a result of the exponential rise of web-based applications. These logs analysis can reveal important information about user activity, web server performance, and security problems. It is a difficult task, though, because to the sheer amount and intricacy of these data. The goal of this project is to provide a user-friendly web server log analysis system that automates log processing and produces useful insights. We have reviewed the available methods and tools for web server log analysis in order to design the system. According to the survey, the majority of current tools are either too difficult to use or do not offer useful information. Moreover, these technologies do not consider the distinctive qualities of many web apps, such as user behaviour patterns and application-specific log formats. By utilising machine learning techniques to automatically detect patterns in log data and deliver tailored insights based on the application type, our technology overcomes these restrictions.

Due to the shortcomings of current technologies and the significance of web server log analysis for organisations, our system is required. Web server logs hold a variety of data that may be used by organisations to improve the efficiency of their web servers, better understand their clients, spot security problems, and adhere to legislation. Nevertheless, manually evaluating these logs is a laborious and error-prone operation. By offering a user-friendly interface that enables users to rapidly spot performance bottlenecks, security concerns, and user behaviour trends, our technology attempts to meet this demand.

One of the primary requirements for web server log analysis is performance optimization. The performance of the web server and the apps it hosts can be learned from the web server logs. It is possible to spot performance bottlenecks and potential improvement areas by analysing the logs. The logs, for instance, may show slow-loading pages, server issues, or an influx of requests that is taxing the server.

Another crucial requirement for web server log analysis is user behaviour analysis. The most popular pages, user demographics, and the time of day when users are most active are just a few examples of the information that web server logs can provide about user behaviour trends. Businesses can improve their marketing strategy and better understand their customers by analysing these patterns. The logs may reveal the most popular items, the most popular pages, or the most successful marketing strategies, for instance.

A crucial requirement for web server log analysis is the discovery of security issues. Logs from web servers can reveal information regarding security problems, including attempts at unauthorised access, malware attacks, and suspicious behaviour. These logs can be examined to identify security flaws and stop upcoming attacks. For example, the logs can reveal failed login attempts, unusual traffic patterns, or suspicious IP addresses.

Another essential requirement for web server log analysis is compliance. Regulation adherence is required by several businesses, including HIPAA and GDPR. Businesses can make sure they are complying with these criteria and are not breaking any privacy laws by analysing web server logs. The logs, for instance, can show which pages have personal information on them, who has accessed it, and whether any unwanted access attempts have been made.

By offering a user-friendly interface and tailored insights based on the type of application, the proposed web server log analysis system seeks to overcome the shortcomings of existing tools. Web server speed optimization, user behaviour analysis, security issue detection, and compliance monitoring are some of the system's potential applications. Our technology may assist organisations in enhancing the performance of their web servers, better understanding their clients, and ensuring privacy requirements are followed by automating log processing and

offering useful insights.
Web server logs can also be used to detect security problems including malware assaults, unauthorised access attempts, and suspicious activities. These logs can be examined to identify security flaws and stop upcoming attacks.Compliance: Many laws, including the GDPR and the HIPAA, must be followed by many different industries. Businesses can make sure they are complying with these criteria and are not breaking any privacy laws by analysing web server logs.

Table of Content

Chapter-1

INTRODUCTION

1.1 Web Server

Using the Internet or a local network, a web server is a computer system that stores and transmits web pages and other web content to clients. It operates by listening for client requests and responding with the requested data when one is received. A web server normally replies to a request from a client by sending an HTTP (Hypertext Transfer Protocol) message that contains the requested web page or other material.

An key component of the World Wide Web's architecture are web servers. They support a wide range of applications, from straightforward static web pages to intricate dynamic web apps that produce content as-needed, and they allow websites to be viewed from anywhere in the globe, 24 hours a day.

1.2 Logs

The events that take place in a system are recorded in logs. Logs on web servers typically record information about HTTP requests and responses, such as the date and time of the request, the source and destination IP addresses, the requested URL, the HTTP method (GET, POST, etc.), the status code the server returned, and other specifics.

Logs are essential for maintaining and fixing web servers. Administrators can use them to find performance problems, security risks, and other issues that might affect the web server's availability or dependability. Furthermore, logs are frequently needed to comply with laws like the GDPR and HIPAA.

Organizations can make sure that their web apps are compliant with regulations and shield themselves from fines or legal action by analysing log data.

1.2.1 Types of Logs

Logs are produced in a wide variety of ways by various systems and programmes. The following are some of the most typical sorts of logs:

1] System logs: These logs store details about the operating system, such as warnings, failures, and system events.

2] Application logs: These logs record details on how certain applications behave, such as errors, exceptions, and other events.

3] Logs for security events, such as login attempts, modifications to system configurations, and other security-related events are recorded in these logs.

4] Network logs: These logs include details regarding network activity, such as traffic volume, traffic trends, and other activities connected to the network.

5] Server logs: These logs record details about web servers, such as web traffic, problems, and other server performance-related occurrences.

6] Database logs: These logs record details of database transactions, such as errors, warnings, and other occasions relating to the operation of the database.

7] Audit logs: These logs record details of user activity, such as login attempts, access requests, and other user behavior-related occurrences.

System events, such as startup and shutdown events, hardware events, and other events relating to system performance, are detailed in event logs.

Logs are a significant source of data for performance analysis, security monitoring, and troubleshooting in general. Organizations may improve performance, avoid security problems, and optimise their whole IT infrastructure by examining logs to get insights into the behaviour of their systems and applications.

1.3 Status Code

A status code is a three-digit code that the server returns to the client to indicate the status of the response to an HTTP request in the context of web servers and HTTP requests. It gives details on the outcome of the client's request and is a component of the HTTP protocol.

Common status codes include the following:

- 200 OK: The request has been completed successfully by the server.
- 301 Permanently Moved: The requested resource has been transferred permanently to another URL.
- 404 Not Found: The server was unable to locate the requested resource.
- 500 Internal Server Error: The server was unable to process the request due to an unforeseen circumstance.

Each status code has a distinct meaning that aids in resolving any problems that may arise with the web server or the client's request.

1.4 Web Server Logs

The information contained in web server logs is useful for understanding user behaviour and streamlining web applications. The performance, usability, and security of web applications can be studied using the data that web servers keep, including IP addresses, URLs, user agents, and response codes. The purpose of this project is to use web server logs to learn more about how a web application is used to spot any potential security risks or anomalies. In this project, the log data will be cleaned, preprocessed, and analysed using Python and other data analysis and machine learning packages.

Cleaning and preprocessing the raw log data is the initial phase in the analysis, and it involves extracting relevant information from the log files, such as IP addresses, timestamps, URLs, and user agents. After the data has been preprocessed, we will examine it using a variety of statistical and visualisation approaches to learn more about the web application's usage trends. In the next step, we will use machine learning algorithms to detect anomalies or security threats in the log data. Anomalies can be caused by various factors such as bot traffic, server errors, or malicious attacks, and can have significant impact on the performance and security of the web application. We will use unsupervised learning algorithms such as Isolation Forest and Local Outlier Factor to detect anomalies in the log data.

The volume of data produced by web servers has increased dramatically as a result of an increased reliance on web-based applications and services. A wealth of data may be found in web server logs that can be utilised to understand user behaviour, website performance, and security risks. Organizations that depend on web-based services must now analyse web server logs in order to get insight into their online applications, optimise their infrastructure, and guarantee the security of their systems.

The objective of this project is to use various data analysis and machine learning approaches to examine web server logs and get useful insights from them. Python will be used to do data analysis, alter data, and create machine learning models in order to find trends and

abnormalities in web server log data. The pre-processing of the log data, feature engineering, and application of machine learning models to the pre-processed data will all be part of the project's efforts to distinguish between regular and abnormal traffic.

Several well-known Python libraries, including Pandas, NumPy, Matplotlib, and Scikit-Learn, will be used in the project to carry out data analysis and machine learning tasks. We will also use PyCaret, a high-level machine learning library, to streamline the machine learning process and perform hyperparameter tuning to optimize our models' performance.

1.5 PROBLEM STATEMENT

The way we work, live, and conduct business has all been transformed by the internet. Yet as there are more and more online enterprises, cyber threats are a huge worry. In 2020 alone, there were over 5 billion cyberattacks, according to a recent Symantec analysis. In addition to risking the security of online enterprises, these attacks also result in financial losses, harm to brand reputation, and legal liability.

An efficient method to track and examine web traffic in order to spot irregularities and stop cyberattacks is web server log analysis. The user's IP address, the requested URL, the date and time of the request, the user agent, and the server response code are all recorded in web server logs for each request made to the server.Web server log analysis is problematic since it may be a time-consuming and difficult operation, especially for companies that get a lot of requests. It calls for proficiency in data analysis as well as a solid grasp of web server protocols. Also, because web server logs contain a lot of data and abnormalities may be subtle and hard to spot, it might be challenging to find anomalies in the logs.

To overcome these difficulties, we suggest a system for analysing web server logs that makes use of machine learning algorithms to spot anomalies. A model will be trained by the system using supervised learning methods on a labelled dataset of web server logs, where the labels denote whether a log entry is typical or abnormal. Based on the properties of a new log entry, the trained model will be used to forecast whether it is normal or abnormal.

Businesses will have a quick and easy way to check their web server logs for anomalies and potential cyberattacks according to the suggested solution. Businesses will be able to swiftly spot and respond to abnormal activity, lowering the likelihood of financial losses, reputational harm to their brands, and legal liability.

Chapter-2

Literary Survey

A literary survey is an important step in any research project, as it involves reviewing the existing literature on a topic to identify what has already been studied and what gaps exist in current knowledge. In the case of a web server log analysis project, a literary survey might involve reviewing existing research on topics related to web server logs, such as log file formats, data preprocessing, log analysis techniques, and so on.

Here are a few recent studies related to web server log analysis that you might consider including in your literary survey:

1. Web Server Log Analyzer by Nisar Nadaf, Pramod Patha :
   Large amount of useful operations data and warnings of failures are available in the server logs. The challenge is that such information gets increases with time, as a result of it number of entries in the logs, which can quickly grow to unmanageable size. Automating the analysis of server logs is essential to allow using the logs as a proactive administrator tool. Log analyzer is 3-tier architecture based software that parses through the log files generated by whichever web server follows the standard web server logging standards. Analyze parsed data and categorize them into meaningful reports that can be read by the user for administration or monitoring purpose. A software application designed to parse a log file, which typically contains raw collected data, and convert it to an easy-to-read and understand form.
2. ANALYSIS OF WEB LOGS AND WEB USER IN WEB MINING by L.K. Joshila Grace, V.Maheswari, Dhinaharan Nagamalai:
   Log files contain information about User Name, IP Address, Time Stamp, Access Request, number of Bytes Transferred, Result Status, URL that Referred and User Agent. The log files are maintained by the web servers. By analysing these log files gives a neat idea about the user. This paper gives a detailed discussion about these log files, their formats, their creation, access procedures, their uses, various algorithms used and the additional parameters that can be used in the log files which in turn gives way to an effective mining. It also provides the idea of creating an extended log file and learning the user behaviour.

3. "Deep Learning Based Web Server Log Analysis for Cybersecurity" by Taeyeon Ki and Seungyeob Choi (2020):
   This paper discloses a log analysis method based on deep learning for an intrusion detection system, which includes the following steps: preprocess the acquired logs of different types in the target system; perform log analysis on the preprocessed logs using a clustering-based method; then, encode the parsed log events into digital feature vectors; use LSTM-based neural network and log collect-based clustering methods to learn the encoded logs to form warning information; lastly, trace the source of the warning information to the corresponding component to determine the point of intrusion.

4. ANALYSIS OF WEB SERVER LOG FILES TO INCREASE THE EFFECTIVENESS OF THE WEBSITE USING WEB MINING TOOL by Arvind K. Sharma and P.C. Gupta:
   The fundamental role of Web usage mining is to capture, analyze, and model the Web server logs. Usually it automatically discovers the usage behaviour of the Website users. In this paper, we have been implemented a Web mining tool to analyze the Web server log files of the Website. It evaluates the important information about visitors, top errors, web browsers and different platforms used by the Website users mostly. The obtained information shall definitely increase the effectiveness of the Website
5. Comprehensive Analysis of Web Log Files for Mining by Vikas Verma , A. K. Verma , S. S. Bhatia:
   World Wide Web is a global village and rich source of information. Day by day number of web sites and its users are increasing rapidly. Information extracted from WWW may sometimes do not turn up to desired expectations of the user. A refined approach, referred as Web Mining, which is an area of Data Mining dealing with the extraction of interesting knowledge from the World Wide Web, can provide better result. While surfing the web sites, users' interactions with web sites are recorded in web log file. These Web Logs are abundant source of information. Such logs when mined properly can provide useful information for decision making. Mining of these Web Logs is referred to as Web Log Mining. This paper analyses web log data of NASA of the month of August 1995 of 15.8MB and depicts certain behavioral aspects of users using web log mining.
6. Discovering Web Server Logs Patterns Using Clustering and Association Rules Mining By Mohammed Hamed Ahmed Elhebir and Ajith Abraham :
   In this paper, the server log files of the Website " www.sust.edu" is considered for overall study and analysis. This paper presents the discovering patterns of Web Usage Mining (WUM) using Clustering and Association Rule from web log data. In the first stage, the data pre-processing phase was performed, and in the second stage k-means and density-based clustering algorithms are used for clustering the log file into groups. Then these clusters are plotted on a plane using WEKA tool, where different clusters are distinguished by distinct colors and distinct symbols. Performance and accuracy of the clustering algorithms are presented and compared.

These studies are just a few examples of the research that has been conducted in the area of web server log analysis. By conducting a comprehensive literary survey, you can identify the key findings and limitations of previous studies, and use this information to inform your own research project.

Chapter-3

Architecture and Flow Chart

3.1 Architecture of web server log analysis project

The architecture of a web server log analysis project typically involves several components, which may include data collection, storage, preprocessing, analysis, and visualization. Here is a general overview of the architecture of a web server log analysis project:

Data Collection: In this step, web server logs are collected from the web server and stored in a database or file system. The logs can be collected using various methods, such as using log files, web server access logs, or other tools.

Data Storage: The collected web server logs are stored in a database or file system. The database can be a relational database, NoSQL database, or a distributed file system like Hadoop Distributed File System (HDFS).

Data Preprocessing: In this step, the collected data is preprocessed to clean and filter the data. This step involves removing irrelevant or duplicate entries, and converting the data into a standardized format. Data preprocessing techniques can be applied to improve the quality and consistency of the data.

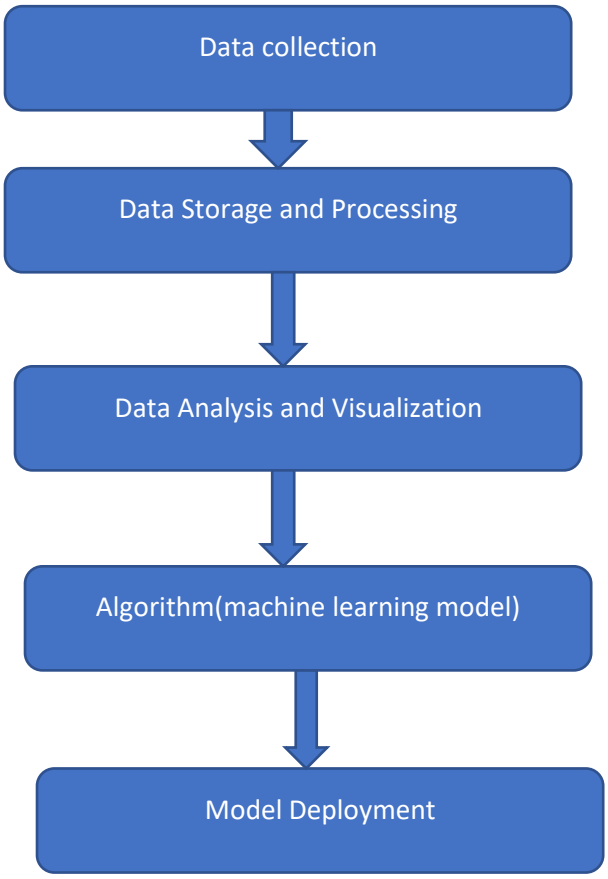
Data Analysis: Data analysis techniques are applied to the preprocessed data to extract insights and patterns. This step involves applying statistical or machine learning algorithms to the data. For instance, log data can be used to identify popular pages, determine user behavior patterns, and detect security breaches.

Data Visualization: Data visualization tools are used to present the insights and patterns in a clear and understandable format. This step involves creating charts, graphs, or other visualizations to display the results of the analysis. Data visualization can help in identifying trends and patterns that might be difficult to detect using other methods.

Deployment: The final step is to deploy the web server log analysis system. The system can be deployed in a cloud-based environment or an on-premise infrastructure, depending on the needs and resources of the organization. The system can be designed to receive data from multiple web servers and can be scalable to handle increasing data volumes.

3.2 Flow Chart For Web Server Log Analysis

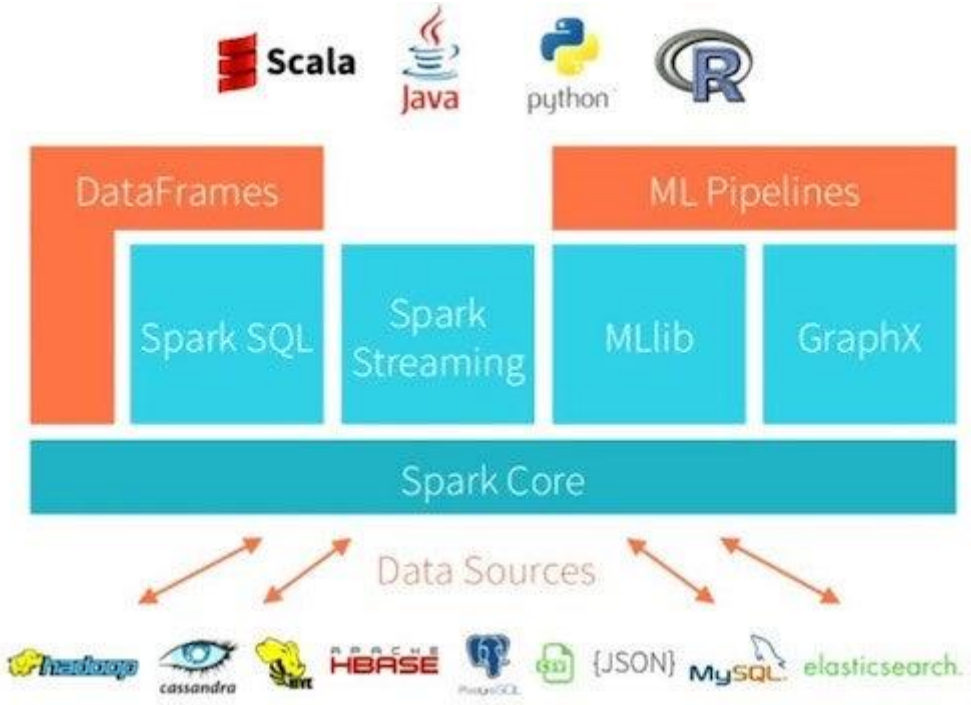
The flow chart for a web server log analysis project typically includes several steps. Here is a general flow chart that can be used as a starting point:



Chapter-4

Working

One of the most popular and effective enterprise use-cases which leverage analytics today is log analytics. Nearly every organization today has multiple systems and infrastructure running day in and day out. To effectively keep their business running, these organizations need to know if their infrastructure is performing to its maximum potential. Finding out involves analyzing system and application logs and maybe even applying predictive analytics on log data. The amount of log data involved is typically massive, depending on the type of organizational infrastructure involved and applications running on it.



The log data processing pipeline

Gone are the days when we were limited to analyzing a data sample on a single machine due to compute constraints. Powered by big data, better and distributed computing, and frameworks like Apache Spark for big data processing and open source analytics, we can perform scalable log analytics on potentially billions of log messages daily. The intent of this case study-oriented tutorial is to take a hands-on approach showcasing how we can leverage Spark to perform log analytics at scale on semi-structured log data.

Main Objective: Web Server Log Analytics

As we mentioned before, Apache Spark is an excellent and ideal open source framework for wrangling, analyzing and modeling structured and unstructured data. Our main objective is one of the most popular use-cases in the industry log analytics. Server logs are a common enterprise data source and often contain a gold mine of actionable insights and information. Log data comes from many sources in these conditions, such as the web, client and compute servers, applications, user-generated content, and flat files. These logs can be used for monitoring servers, improving business and customer intelligence, building recommendation systems, fraud detection, and much more.

Spark allows you to cheaply dump and store your logs into files on disk, while still providing rich APIs to perform data analysis at scale.

Setting Up Dependencies

The first step is to make sure you have access to a Spark session and cluster. For this step, you can use your own local Spark setup or a cloud-based setup. Typically, most cloud platforms provide a Spark cluster these days and you also have free options, including Databricks community edition.

Often pre-configured Spark setups already have the necessary environment variables or dependencies pre-loaded when you start your Jupyter Notebook server.

Now in case you don't have these variables pre-configured and get an error, you can load and configure them using the following code:

```
[ ]: from pyspark import SparkContext
     from pyspark.sql import SparkSession
     from pyspark.sql.context import SQLContext
     import matplotlib.pyplot as plt
```

Creating a Spark Session

```
[ ]: sc=SparkContext()
     sqlContext=SQLContext(sc)
     spark=SparkSession(sc)
```

We also need to load other libraries for working with DataFrames and regular expressions. Working with regular expressions is one of the major aspects of parsing log files. This tool offers a powerful pattern-matching technique which can be used to extract and find patterns in semi-structured and unstructured data.

Loading and Viewing the Log Dataset

Given that our data is stored in the following path (in the form of flat files), let's load it into a DataFrame. We'll do this in steps. The following code loads our disk's log data file names:

```
[5]: import re
     import pandas as pd
     import glob
```

Inserting every log file having a similar pattern with extension .gz

```
[6]: raw_files=glob.glob('/home/mav_27/Project/Data/nginx-feb2023/nginx/*.gz')
```

```
[7]: raw_files
```

Now, we'll use **sqlContext.read.text()** or **spark.read.text()** to read the text file. This code produces a DataFrame with a single string column called **value**:

```
[9]: base_df=spark.read.text(raw_files)
```

```
[10]: base_df.printSchema()

root
 |-- value: string (nullable = true)
```

Let's now take a peek at the actual log data in our DataFrame:

```
base_df.show(10,truncate=False)

+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
|value
|
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
|143.244.50.172 - - [16/Feb/2023:03:28:45 +0530] "GET /config/getuser?index=0 HTTP/1.1" 400 248 "-" "Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:76.0) Gecko/20100101 Firefox/76.0"
|
|164.90.235.116 - - [16/Feb/2023:04:11:34 +0530] "GET / HTTP/1.1" 200 5952 "http://14.139.152.12/" "Mozilla/5.0 (Windows NT 10.0; Wi
n64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"
|
|66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /robots.txt HTTP/1.1" 404 146 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +htt
p://www.google.com/bot.html)"
|
|66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /assets/img/favicon.png HTTP/1.1" 200 491 "-" "Googlebot-Image/1.0"
|
|185.221.219.172 - - [16/Feb/2023:05:06:47 +0530] "GET /.git/config HTTP/1.1" 404 548 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Window
s NT 6.2; Trident/5.0)"
```

Data Wrangling

In this section, we clean and parse our log dataset to extract structured attributes with meaningful information from each log message.

Log data understanding:
If you're familiar with web server logs, you'll recognize that the data displayed above is in Common Log Format:

The log file entries produced in CLF will look something like this:

127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

The entries give details about the client who had requested for the web site to the web server

• 127.0.0.1 - This is the IP address of the client which made the request to the server.

• - - The hyphen present in the log file entry next to the IP address indicates that the requested information is not available.

• frank - The user id of the person requesting the document as determined by HTTP authentication • [10/Oct/2000:13:55:36 -0700] (%t) -The time format resembles like [day/month/year: hour: minute: second zone]

• "GET /apache_pb.gif HTTP/1.0" - The request sent from the client is given in double quotes. GET is the method used. apache_pb.gif is the information requested by the client. The protocol used by the client is given as HTTP/1.0

• 200 - This is the status code sent by the server. The codes beginning with 2 for successful response, 3 for redirection, 4 for error caused by the client, 5 for error in the server International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.1, January 2011 103

• 2326 - The last entry indicates the size of the object returned to the client by the server, not including the response headers. If there is no content returned to the client, this value will be "-"

The log file entries produced in combined log format will look something like this:

127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"

The additional parameters that are present in the combined log format are discussed below

• "http://www.example.com/start.html" - This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache_pb.gif).

• "Mozilla/4.08 [en] (Win98; I ;Nav)" - This is the information that the client browser reports about itself to the server.

We now need techniques to parse, match, and extract these attributes from the log data.

Data Parsing and Extraction with Regular Expressions

Next, we must parse our semi-structured log data into individual columns. We'll use the special built-in regexp_extract() function to do the parsing. This function matches a column against a regular expression with one or more **capture groups**, and allows you to extract one of the matched groups. We'll use one regular expression for each field we wish to extract.

Let's extract and take a look at some sample log messages:

Taking a Sample data from Base Dataframe for data parsing

[202]:
```
sample_logs=[items['value'] for items in base_df.take(15)]
sample_logs
```

[202]:
```
['143.244.50.172 - - [16/Feb/2023:03:28:45 +0530] "GET /config/getuser?index=0 HTTP/1.1" 400 248 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0"',
 '164.90.235.116 - - [16/Feb/2023:04:11:34 +0530] "GET / HTTP/1.1" 200 5952 "http://14.139.152.12/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"',
 '66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /robots.txt HTTP/1.1" 404 146 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"',
 '66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /assets/img/favicon.png HTTP/1.1" 200 491 "-" "Googlebot-Image/1.0"',
 '185.221.219.172 - - [16/Feb/2023:05:06:47 +0530] "GET /.git/config HTTP/1.1" 404 548 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.2; Trident/5.0)"',
 '66.249.69.101 - - [16/Feb/2023:05:24:52 +0530] "GET /apim/devportal/site/public/images/logo.svg HTTP/1.1" 304 0 "-" "Googlebot-Image/1.0"',
 '52.167.144.90 - - [16/Feb/2023:06:26:16 +0530] "GET / HTTP/1.1" 200 5952 "-" "Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm) Chrome/103.0.5060.134 Safari/537.36"',
 '152.89.196.211 - - [16/Feb/2023:07:00:15 +0530] "GET /vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php HTTP/1.1" 404 548 "http://14.139.152.12:80/vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36"',
 '114.119.133.53 - - [16/Feb/2023:07:01:52 +0530] "GET /robots.txt HTTP/1.1" 404 146 "-" "Mozilla/5.0 (compatible;PetalBot;+https://webmaster.petalsearch.com/site/petalbot)"',
 '198.20.69.98 - - [16/Feb/2023:07:26:00 +0530] "GET / HTTP/1.1" 200 5952 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36"',
```

Extracting Hostnames

Let's write some regular expressions to extract the hostname from the logs:

[203]:
```
host_pattern=r'(^\S+\.[\S+\.]+\S+)\s'
hosts=[re.search(host_pattern,items).group(1)
       if re.search(host_pattern,items)
       else 'no match'
       for items in sample_logs]
```

[204]:
```
hosts
```

Extracting Timestamps

Let's use regular expressions to extract the timestamp fields from the logs:

```
date_pattern=r'\[(\d{2}/\w{3}/\d{4})'
date=[re.search(time_stamp_pattern,items).group(1)
            for items in sample_logs]
date
```

```
time_pattern=r':(\d{2}:\d{2}:\d{2})'
time=[re.search(time_pattern,items).group(1)
            for items in sample_logs]
time
```

Extracting HTTP Request Method, Uris, And Protocol

Let's now use regular expressions to extract the HTTP request methods, URIs, and Protocol patterns fields from the logs:

[207]:
```
method_pattern=r'\"(\S+)\s(\S+)\s*(\S*)\"'
method_uri=[re.search(method_pattern,items).group()
           for items in sample_logs]
```

[208]:
```
method_uri
```

Extracting HTTP Status Codes

Let's now use regular expressions to extract the HTTP status codes from the logs:

[209]:
```
status_pattern=r'\s(\d{3})\s'
status_code=[re.search(status_pattern,items).group(1)
            for items in sample_logs]
```

[210]:
```
status_code
```

Extracting Http Response Content Size

Let's now use regular expressions to extract the HTTP response content size from the logs:

```
211]: content_size_pattarn=r'\s(\d+) "'
      content_size=[re.search(content_size_pattarn,items).group(1)
                    for items in sample_logs]
```

```
212]: content_size
```

Putting It All Together

Let's now leverage all the regular expression patterns we previously built and use the regex_extract(…) method to build our DataFrame with all of the log attributes neatly extracted in their own separate columns.

Creating a clean dataframe with parsed data ¶

```
from pyspark.sql.functions import regexp_extract
```

```
logs_df=base_df.select(regexp_extract('value',host_pattern,1).alias('Host'),
                       regexp_extract('value',date_pattern,1).alias('Date'),
                       regexp_extract('value',time_pattern,0).alias('Time'),
                       regexp_extract('value',method_pattern,1).alias('Method'),
                       regexp_extract('value',method_pattern,2).alias('Endpoint'),
                       regexp_extract('value',method_pattern,3).alias('Protocol'),
                       regexp_extract('value',status_pattern,1).alias("Status Code"),
                       regexp_extract('value',content_size_pattarn,1).cast('integer').alias('Content Size'))
```

Removing Bad Rows

After putting it all together we will now treat the rows having null values.

```
bad_row_df=logs_df.filter(logs_df['Host'].isNull() |
                          logs_df['Date'].isNull() |
                          logs_df['Time'].isNull() |
                          logs_df['Method'].isNull() |
                          logs_df['Endpoint'].isNull() |
                          logs_df['Protocol'].isNull() |
                          logs_df['Status Code'].isNull() |
                          logs_df['Content Size'].isNull())
```

```
bad_row_df.count()
```

```
482
```

Converting the Spark DataFrame to Pandas DataFrame

For analysis purpose the spark DataFrame has to be converted into pandas DataFrame.

```
[221]: logs_df=logs_df.toPandas()
```

```
[222]: logs_df.dtypes
```

```
[222]: Host            object
       Date            object
       Time            object
       Method          object
       Endpoint        object
       Protocol        object
       Status Code     object
       Content Size   float64
       dtype: object
```

HTTP Status Code Analysis

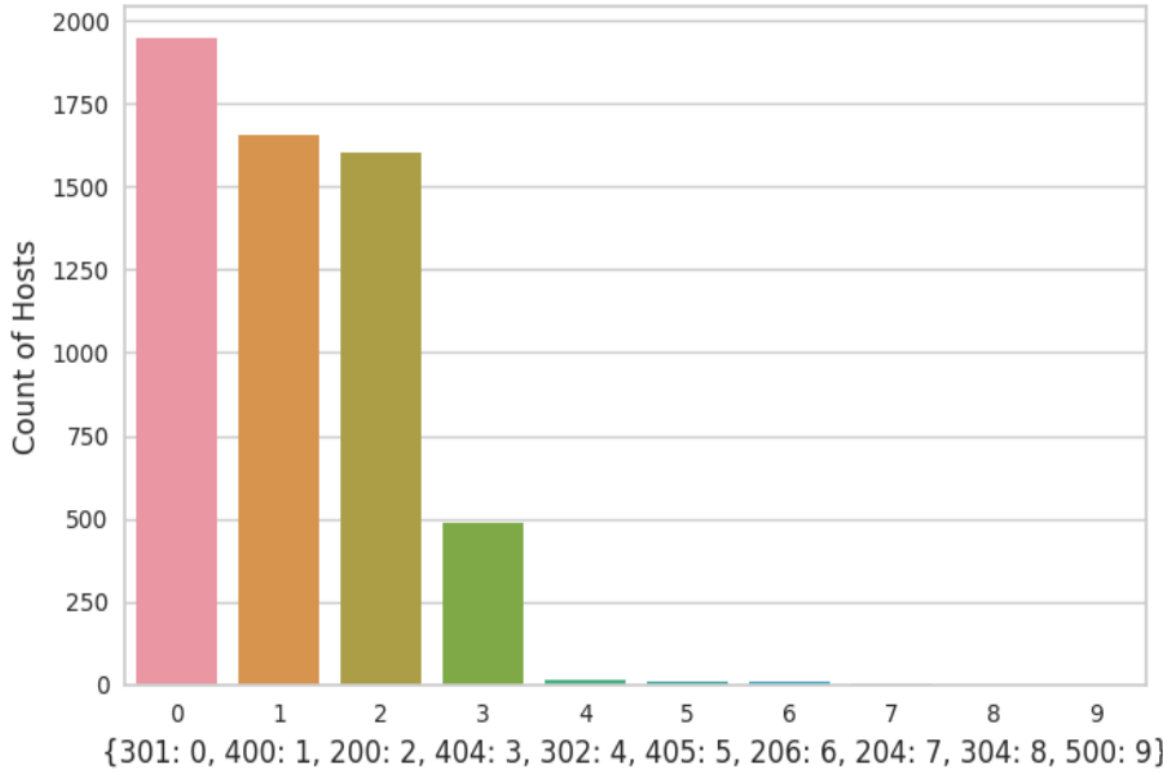
Let’s take a look at each status code's occurrences in the form of a frequency table:

```
232]: logs_df['Status Code'].value_counts()
```

```
232]: 301    1948
       400    1658
       200    1605
       404     486
       302      12
       405      11
       206      10
       204       4
       304       1
       500       1
       Name: Status Code, dtype: int64
```

Looks like the most frequent status code is 301—OK—301 redirects are designed for pages that have permanently moved. Accordingly, you should use them to redirect traffic that might have gone to your old and obsolete pages. Status code 400 also occur many times which signifies bad Request response status code indicates that the server cannot or will not process the request due to something that is perceived to be a client error. Let’s visualize this:

```
96]: plt.figure(figsize=(8,5))
     sns.barplot(y=df['Status Code'].value_counts(),x=df['Status Code'].sort_values().unique())
     plt.xlabel(error_code)
     plt.ylabel('Count of Hosts')
```



Let’s find out how many percentage of error code are present in our file.

```
52]: error=df['Host'][df['Status Code']<5].count()

53]: total=df['Status Code'].count()

54]: print('Percentage of Error Codes: ',(error/total)*100)

     Percentage of Error Codes:  62.39539748953975
```

HTTP Request Method Analysis

Let's find out how many percentage of request method are present in our file.

```
[55]: getMethod=df.Method[df.Method==4].count()
```

```
[56]: print('Percentage of GET Method: ',(getMethod/total)*100)
```

```
Percentage of GET Method:  60.442817294281724
```

```
[57]: postMethod=df.Method[df.Method==7].count()
```

```
[58]: print('Percentage of POST Method: ',(postMethod/total)*100)
```

```
Percentage of POST Method:  12.325662482566248
```

```
[59]: otherMethod=df.Method[(df.Method!=4) & (df.Method!=7)].count()
```

```
[60]: print('Percentage of others Method: ',(otherMethod/total)*100)
```

```
Percentage of others Method:  27.23152022315202
```

Analysing Number Of Request Made By Each Host

And adding a new feature in dataframe by the name 'No of Requests'.

```
logs_df['No of Requests']=logs_df.groupby(["Host"])['Endpoint'].transform('count')
```

```
/tmp/ipykernel_185/1435016481.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
a-copy
  logs_df['No of Requests']=logs_df.groupby(["Host"])['Endpoint'].transform('count')
```

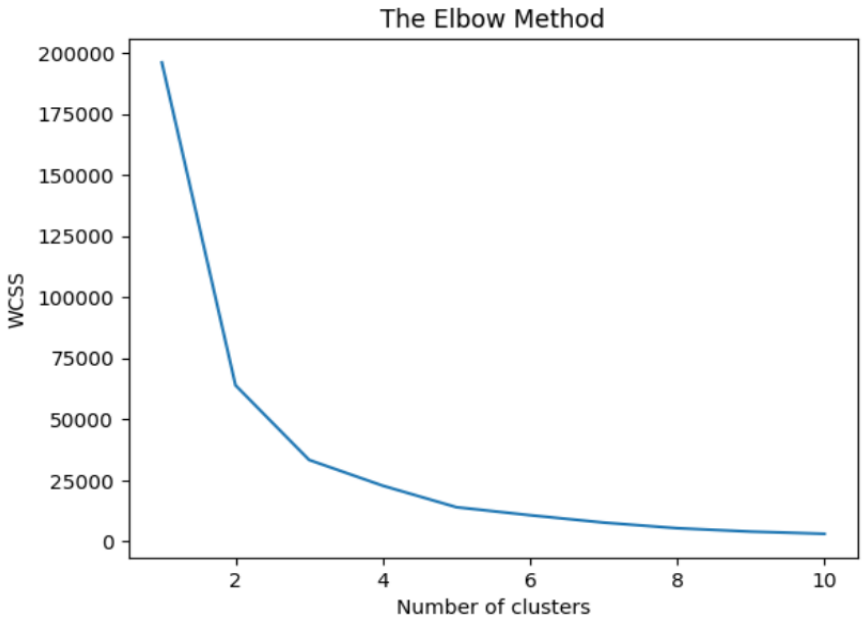
```
logs_df.head(10)
```

	Host	Date	Time	Method	Endpoint	Protocol	Status Code	Content Size	No of Requests
0	2415145644	16/Feb/2023	03:28:45	GET	/config/getuser?index=0	HTTP/1.1	400	248.0	62
1	2757421940	16/Feb/2023	04:11:34	GET	/	HTTP/1.1	200	5952.0	5
2	1123632510	16/Feb/2023	04:39:52	GET	/robots.txt	HTTP/1.1	404	146.0	20
3	1123632510	16/Feb/2023	04:39:52	GET	/assets/img/favicon.png	HTTP/1.1	200	491.0	20
4	3118324652	16/Feb/2023	05:06:47	GET	/.git/config	HTTP/1.1	404	548.0	1
5	1123632485	16/Feb/2023	05:24:52	GET	/apim/devportal/site/public/images/logo.svg	HTTP/1.1	304	0.0	3
6	883396698	16/Feb/2023	06:26:16	GET	/	HTTP/1.1	200	5952.0	19
7	2556019923	16/Feb/2023	07:00:15	GET	/vendor/phpunit/phpunit/src/Util/PHP/eval-stdi...	HTTP/1.1	404	548.0	84

Creating Clusters Using K-Means

First let's find out how many appropriate clusters can be made by elbow curve method.

```
[9]: from sklearn.cluster import KMeans
     import numpy as np
     wcss = []
     for i in range(1, 11):
         kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
         kmeans.fit(x)
         wcss.append(kmeans.inertia_)
     plt.plot(range(1, 11), wcss)
     plt.title('The Elbow Method')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
     plt.show()
```

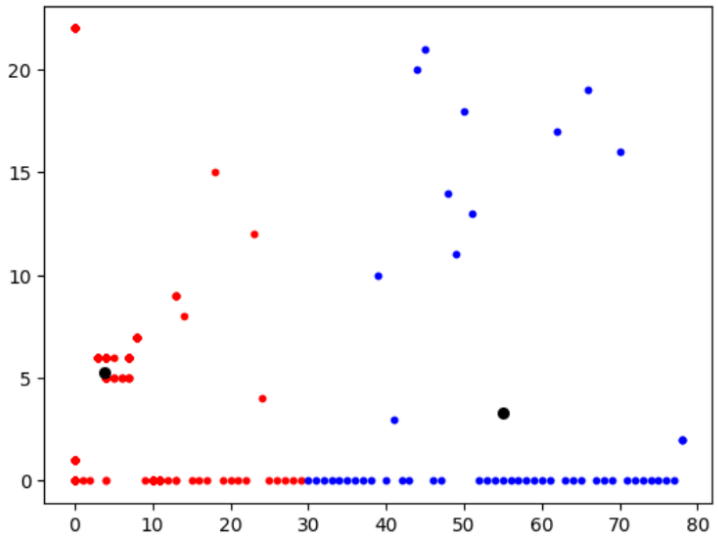


And now applying k-means.

```
kmeans = KMeans(n_clusters = 2, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(x)
```

```
21]: plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 10, c = 'red', label = 'Cluster 1')
     plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 10, c = 'blue', label = 'Cluster 2')
     # plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 10, c = 'green', label = 'Cluster 3')
     plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 30, c = 'black', label = 'Centroids')
```

```
21]: <matplotlib.collections.PathCollection at 0x7f13497c3400>
```



As we can see using K-Means, forming the clusters is difficult,hence we are going to use a package pycaret which is used to detect anomalies and form a 3D Cluster.

Creating Clusters Using Pycaret Package

PyCaret's anomaly detection module (pycaret.anomaly) is a an unsupervised machine learning module which performs the task of identifying rare items, events or observations which raise suspicions by differing significantly from the majority of the data.

PyCaret anomaly detection module provides several pre-processing features that can be configured when initializing the setup through setup() function. It has over 12 algorithms and few plots to analyze the results of anomaly detection. PyCaret's anomaly detection module also implements a unique function tune_model() that allows you to tune the hyperparameters of anomaly detection model to optimize the supervised learning objective such as AUC for classification or R2 for regression.

We will use isolation forest, cluster, KNN model for detecting anomaly.

[22]:
```
from pycaret.anomaly import *
```

[23]:
```
expo_ano=setup(data=df)
```

	Description	Value
0	Session id	8964
1	Original data shape	(5736, 9)
2	Transformed data shape	(5736, 3589)
3	Numeric features	7
4	Categorical features	2
5	Preprocess	True
6	Imputation type	simple
7	Numeric imputation	mean
8	Categorical imputation	mode
9	Maximum one-hot encoding	-1
10	Encoding method	None
11	CPU Jobs	-1
12	Use GPU	False
13	Log Experiment	False

Using model with Algo iForest

[24]:
```
iforest=create_model('iforest')
```

[25]:
```
plot_model(iforest)
```

26]:
```
result=assign_model(iforest)
```

27]:
```
result.Anomaly.value_counts()
```

27]:
```
0    5454
1     282
Name: Anomaly, dtype: int64
```

Using cluster model:

```
[31]: dt=create_model('cluster')
```

```
[32]: plot_model(dt)
```

```
result2=assign_model(dt)
```

```
result2
```

	Host	Date	Time	Method	Endpoint	Protocol	Status Code	Content Size	No of Requests	Anomaly	Anomaly_Score
0	2415145644	2023-02-16	03:28:45	4	236	6	6	248.0	62	0	1.726190e+08
1	2757421940	2023-02-16	04:11:34	4	10	6	0	5952.0	5	0	1.696573e+08
2	1123632510	2023-02-16	04:39:52	4	400	6	7	146.0	20	0	6.466448e+07
3	1123632510	2023-02-16	04:39:52	4	177	6	0	491.0	20	0	6.466448e+07
4	3118324652	2023-02-16	05:06:47	4	40	6	7	548.0	1	0	5.668946e+07
...	...	...	...	...	...	...	...	...	...	...	...
5731	2584535982	2023-02-19	02:48:08	7	240	6	7	146.0	6	0	3.228675e+06
5732	2584535982	2023-02-19	02:48:10	4	240	6	7	146.0	6	0	3.228675e+06
5733	2584535982	2023-02-19	02:48:12	7	10	6	8	150.0	6	0	3.228675e+06
5734	2584535982	2023-02-19	02:48:13	7	240	6	7	146.0	6	0	3.228675e+06
5735	3164952431	2023-02-19	03:06:29	4	10	6	0	5952.0	1	0	1.006168e+07

Using KNN model:

```
[39]: knn=create_model('knn')
```

```
[41]: plot_model(knn)
```

```
[42]: result3=assign_model(knn)
```

```
[43]: result3
```

[43]:

	Host	Date	Time	Method	Endpoint	Protocol	Status Code	Content Size	No of Requests	Anomaly	Anomaly_Score
0	2415145644	2023-02-16	03:28:45	4	236	6	6	248.0	62	0	0.000000
1	2757421940	2023-02-16	04:11:34	4	10	6	0	5952.0	5	0	14103.180918
2	1123632510	2023-02-16	04:39:52	4	400	6	7	146.0	20	0	281.709070
3	1123632510	2023-02-16	04:39:52	4	177	6	0	491.0	20	0	462.341865
4	3118324652	2023-02-16	05:06:47	4	40	6	7	548.0	1	0	224907.432834
...	...	...	...	...	...	...	...	...	...	...	...
5731	2584535982	2023-02-19	02:48:08	7	240	6	7	146.0	6	0	228.525710
5732	2584535982	2023-02-19	02:48:10	4	240	6	7	146.0	6	0	228.525710
5733	2584535982	2023-02-19	02:48:12	7	10	6	8	150.0	6	0	228.525710
5734	2584535982	2023-02-19	02:48:13	7	240	6	7	146.0	6	0	228.525710
5735	3164952431	2023-02-19	03:06:29	4	10	6	0	5952.0	1	0	87306.978782

After applying clustering and adding label we will applying different types of algorithm such as :

1. Catboost
2. Xgboost
3. Gradient boosting
4. Random forest
5. Ada boost

Down sampling The Dataset

Down sampling an imbalanced dataset is essential because many machine learning algorithms assume that the data is balanced and will perform poorly if this assumption is not met.

Down sampling can address this problem by reducing the size of the majority class and creating a new, balanced dataset. It allows the machine learning algorithm to learn from a more representative sample of the data and can improve its performance on the minority class.

Downsampling the dataset

```
[6]: from sklearn.utils import resample
```

```
[7]: anomaltnotdetected_downsample=resample(anomaltnotdetected,replace=True,n_samples=len(anomalydetected),random_state=94)
```

```
[8]: anomaltnotdetected_downsample.shape
```

```
[8]: (287, 11)
```

```
[9]: downsampled_df=pd.concat([anomaltnotdetected_downsample,anomalydetected])
```

```
[10]: downsampled_df=downsampled_df.sample(frac=1,random_state=94)
```

```
[14]: downsampled_df2=pd.concat([anotheranomalynotdetected_downsampling,anomalydetected])
```

```
[15]: downsampled_df2=downsampled_df2.sample(frac=1,random_state=94)
```

```
[11]: downsampled_df
```

[11]:

	Host	Date	Time	Method	Endpoint	Protocol	Status Code	Content Size	No of Requests	Anomaly	Anomaly_Score
1047	3005985266	2023-02-17	10:04:07	0	1	1	6	0.0	35	0	0.000000e+00
4745	824447248	2023-02-17	16:35:34	4	186	6	0	537868.0	74	0	1.185676e+05
906	1152894686	2023-02-21	19:23:54	4	10	6	3	169.0	4	1	2.033551e+07

Splitting The Data Into Train And Test

Splitting the data such that 70% data is for training and 30% for testing.

Spliting the Data for Training and Testing

```
[53]: (x_train,x_test,y_train,y_test)=train_test_split(X,Y,test_size=0.3,random_state=94)
```

```
[54]: x_train
```

[54]:

	Host	Method	Endpoint	Protocol	Status Code	Content Size	No of Requests
998	1713757092	7	10	6	3	169.0	2
14	3323217250	4	400	6	7	146.0	10
133	2050150420	4	164	6	0	305.0	282
370	585331374	4	10	6	0	5952.0	3
5206	2039100042	0	2	22	3	169.0	1
...	...	...	...	...	...	...	...
47	2050150420	4	197	6	0	20309.0	282
284	879889991	4	177	6	0	491.0	3
246	2637312005	4	188	6	0	35445.0	41
5345	872299377	4	10	6	3	169.0	2
5288	1099761322	4	16	6	3	169.0	2

401 rows × 7 columns

Using Randomized Search of Hyperparameter Optimization on every model:

RandomizedSearchCV is very useful when we have many parameters to try and the training time is very long

The first step is to write the parameters that we want to consider and from these parameters select the best ones.

```
[55]: param1={"random_state":[i for i in range (100)],"n_estimators":[i for i in range(200)],"n_jobs":[i for i in range(20)],"learning_rat
      param2={'max_depth': [3,4,5],'n_estimators':[100, 200, 300],"learning_rate": [0.02,0.1,0.01,0.2,0.001,0.002]}
      param3={"random_state":[i for i in range (100)],"n_estimators":[i for i in range(200)],"criterion":["squared_error","friedman_mse"],
      param4={"random_state":[i for i in range (100)],"n_estimators":[i for i in range(200)],"criterion":["entropy","gini"],"n_jobs":[i fo
      param5={"random_state":[i for i in range (100)],"n_estimators":[i for i in range(200)],"learning_rate":[0.001,0.002,0.2,0.1,0.01]}
```

Now we can create our **RandomizedSearchCV** object and fit the data. Finally, we can find the best parameters and the best scores.

```
[56]: model1=RandomizedSearchCV(XGBClassifier(),param_distributions=param1,cv=5,verbose=1,scoring="accuracy")
      model2=RandomizedSearchCV(XGBRFClassifier(),param_distributions=param1,cv=5,verbose=1,scoring='accuracy')
      model3=RandomizedSearchCV(CatBoostClassifier(),param_distributions=param2,cv=5,verbose=1,scoring='accuracy')
      model4=RandomizedSearchCV(GradientBoostingClassifier(),param_distributions=param3,verbose=1,cv=5,scoring='accuracy')
      model5=RandomizedSearchCV(RandomForestClassifier(),param_distributions=param4,verbose=1,cv=5,scoring='accuracy')
      model6=RandomizedSearchCV(AdaBoostClassifier(),param_distributions=param5,verbose=1,cv=5,scoring='accuracy')
```

[57]:
```
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
model4.fit(x_train,y_train)
model5.fit(x_train,y_train)
model6.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Best Parameter For Each Model:

[58]:
```
print(model1.best_params_)
print(model2.best_params_)
print(model3.best_params_)
print(model4.best_params_)
print(model5.best_params_)
print(model6.best_params_)
```

```
{'random_state': 97, 'n_jobs': 17, 'n_estimators': 38, 'learning_rate': 0.1}
{'random_state': 13, 'n_jobs': 4, 'n_estimators': 143, 'learning_rate': 0.01}
{'n_estimators': 300, 'max_depth': 5, 'learning_rate': 0.1}
{'random_state': 15, 'n_estimators': 184, 'learning_rate': 0.1, 'criterion': 'friedman_mse'}
{'random_state': 88, 'n_jobs': 10, 'n_estimators': 21, 'criterion': 'gini'}
{'random_state': 4, 'n_estimators': 173, 'learning_rate': 0.1}
```

Predicting The Anomalies:

59]:
```
p=model1.predict(x_test)
p2=model2.predict(x_test)
p3=model3.predict(x_test)
p4=model4.predict(x_test)
p5=model5.predict(x_test)
p6=model6.predict(x_test)
```

Checking performance of various models by applying various performance metrics such as:

- Precision
- Recall
- F1-score
- Confusion matrix
- Accuracy
- Matthews correlation coefficient

After checking performance of various model we will save the best model using joblib or pickle. Pickle or joblib is a useful Python tool that allows you to save your ML models, to minimise lengthy re-training and allow you to share, commit, and re-load pre-trained machine learning models. Most data scientists working in ML will use Pickle or Joblib to save their ML model for future use .

[60]:
```
from sklearn.metrics import classification_report,confusion_matrix,matthews_corrcoef
```

[61]:
```
print(classification_report(p,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        84
           1       0.95      0.98      0.96        89

    accuracy                           0.96       173
   macro avg       0.96      0.96      0.96       173
weighted avg       0.96      0.96      0.96       173
```

[62]:
```
print(matthews_corrcoef(y_test,p))
```

```
0.9194813598071867
```

[63]:
```
confusion_matrix(p,y_test)
```

[63]:
```
array([[79,  5],
       [ 2, 87]])
```

[64]:
```
import joblib
```

[65]:
```
joblib.dump(model1,'../Model/model_1')
```

[65]:
```
['../Model/model_1']
```

[72]:
```
print(classification_report(p4,y_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96        80
           1       0.97      0.96      0.96        93

    accuracy                           0.96       173
   macro avg       0.96      0.96      0.96       173
weighted avg       0.96      0.96      0.96       173
```

[73]:
```
print(matthews_corrcoef(y_test,p4))
```

```
0.9187486195256728
```

[74]:
```
confusion_matrix(p4,y_test)
```

[74]:
```
array([[77,  3],
       [ 4, 89]])
```

[75]:
```
joblib.dump(model4,'../Model/model_4')
```

[75]:
```
['../Model/model_4']
```

Among all these models, we have selected 2 models:

XGBoost and GradientBoost Models,

as both of these models have good precision , f1-score as well as matthews's Correlation Coefficient

KAFKA Connection

After the model has been saved we will now create a kafka producer. producers can Push Messages into the tail of these newly created logs while consumers Pull Messages off from a specific Kafka Topic.

By creating Kafka Topics, users can perform **Logical Segregation** between Messages and Events, which works the same as the concept of different tables having different types of data in a database.

Connecting to the server and Creating topic for Kafka

[8]:
```
import json
from kafka.admin import NewTopic,KafkaAdminClient
from kafka import KafkaProducer
from time import sleep
```

[9]:
```
topic=NewTopic(name='Topic3',num_partitions=3,replication_factor=1)
```

[10]:
```
admin=KafkaAdminClient(bootstrap_servers='localhost:9092')
```

[11]:
```
admin.create_topics([topic])
```

[12]:
```
def json_serializer(data):
    return json.dumps(data).encode('utf-8')
```

[13]:
```
producer=KafkaProducer(bootstrap_servers='localhost:9092',value_serializer=json_serializer)
```

[14]:
```
admin.describe_topics([topic])
```

[14]:
```
[{'error_code': 17,
  'topic': '<kafka.admin.new_topic.NewTopic object at 0x7f162f82d870>',
  'is_internal': False,
  'partitions': []}]
```

Sending the Data to the Consumer

[23]:
```
for index,row in df.iterrows():
    producer.send("topic3", row.to_json())
```

Then consumer sends the anomaly data (for which anomaly = 1) to mongo db

For that first we have to connect to mongo db database

[13]:
```
try:
   client = MongoClient('localhost',27017)
   db = client.AnomalyDetection
   print("Connected successfully!")
except:
   print("Could not connect to MongoDB")
```

```
Connected successfully!
```

[28]:
```
db.list_collection_names()
```

[28]:
```
['coba_info', 'Details']
```

[15]:
```
def json_deserializer(data):
    return json.loads(data)
```

[31]:
```
consumer = KafkaConsumer('topic3',bootstrap_servers=['localhost:9092'],auto_offset_reset="latest",group_id="group-1",
                         value_deserializer=json_deserializer,consumer_timeout_ms=15000)
```

Taking the results from Consumer and inserting the data to the MongoDB Database

[32]:
```
for msg in consumer:
    record = json.loads(msg.value)
    HostIP = record['Host']
    Date = record['Date']
    Time = record['Time']
    Method = record['Method']
    Endpoint = record['Endpoint']
    Protocol= record['Protocol']
    Status_Code= record['Status Code']
    Content_Size= record['Content Size']
    No_of_Requests= record['No of Requests']
    Anomaly= record['Anomaly']
    Anomaly_Score= record['Anomaly_Score']

    if record['Anomaly']==1:
    # Create dictionary and ingest data into MongoDB
        try:
           rec = {'Host':HostIP,'Date':Date,'Time':Time,'Method':Method,'Endpoint' :Endpoint,'Protocol': Protocol,'Status_Code':Stat
           rec_id1 = db.Details.insert_one(rec)
           print("Data inserted with record ids", rec_id1)
        except:
           print("Could not insert into MongoDB")
```

```
Data inserted with record ids <pymongo.results.InsertOneResult object at 0x7fa7b0201e10>
Data inserted with record ids <pymongo.results.InsertOneResult object at 0x7fa7b0202d70>
```

Here is the result:

Displaying A single Content of the MongoDB DataBase

]:
```
db.Details.find_one()
```

]:
```
{'_id': ObjectId('6409513ab980f37f6a19c81d'),
 'Host': 580338691,
 'Date': '2023-02-16',
 'Time': '13:48:43',
 'Method': 6,
 'Endpoint': 10,
 'Protocol': 5,
 'Status_Code': 8,
 'Content_Size': 150.0,
 'No_of_Requests': 1,
 'Anomaly': 1,
 'Anomaly_Score': 4799787.000047398}
```

Chapter-5

Applications

Web server log analysis has many applications across different industries and business domains. Here are some of the key applications of web server log analysis:

Website performance optimization: Web server log analysis can help businesses optimize the performance of their websites by identifying slow-loading pages, high traffic times, and other issues that may affect the user experience. By optimizing website performance, businesses can improve user engagement and increase conversions.

Marketing and advertising: Web server log analysis can help businesses identify the most popular pages on their website, as well as the sources of traffic and user behavior patterns. This information can be used to optimize marketing and advertising campaigns to better target the right audience and increase conversions.

Security and fraud prevention: Web server log analysis can help businesses detect and prevent security threats, such as DDoS attacks, hacking attempts, and phishing scams. By analyzing server logs, businesses can identify suspicious activity, block malicious IP addresses, and take other measures to enhance security.

E-commerce and online sales: Web server log analysis can help e-commerce businesses optimize their online stores by identifying customer behavior patterns, such as the most popular products, common search queries, and shopping cart abandonment rates. This information can be used to optimize product offerings, improve website design, and increase sales.

IT operations and infrastructure management: Web server log analysis can help IT teams monitor server and network performance, identify and resolve errors and issues, and optimize infrastructure to improve uptime and reduce downtime.

Conclusion

Web server log analysis is a powerful tool for detecting anomalies on your website. Anomalies can be defined as deviations from the expected or normal behavior, and they can be caused by various factors, including errors, security breaches, or unusual user behavior.

By analyzing server logs, you can detect anomalies that may indicate security threats, performance issues, or other problems with your website. For example, you can detect unusually high traffic or requests from a specific IP address, which may indicate a potential DDoS attack or other security threat. You can also identify errors or failures that are occurring more frequently than expected, indicating performance issues or bugs in your website code.

In conclusion, web server log analysis is a powerful tool for detecting anomalies and improving website performance and security. By analyzing server logs, you can identify potential threats and performance issues, optimize your website content and enhance user experience.