Using Machine Learning to detect and diagnose Anomalies in Microservices

Jacob Smith

180275670

May 2022

BSc Computer Science with Industrial Placement

Professor Raj Ranjan

Word Count: 13592

Abstract

Since their inception, computer systems have been plagued with anomalies and faults, with new and efficient software architectures emerging to combat this. Microservices, a widely used software architecture, offers many benefits to that of a monolithic system, but not without added complexities in identifying anomalies. This paper explores a dataset produced by monitoring system metrics of a microservice architecture, with machine learning techniques employed to detect anomalies. Four distinct models have been implemented, Random Forests, Naïve Bayes, Support Vector Machine, and k-Nearest Neighbour. Python and third-party libraries for data manipulation, visualisation, and machine learning have been used for this project. The performance of the models has been compared using calculations derived from the confusion matrix for both binary and multiclass classification. Random Forests and k-Nearest Neighbour performed the best in all settings, with the latter enduring significantly less time to compute. Future works for the project have been included to improve the performance, accuracy, and real-life application.

Declaration

"I declare that this document represents my own work, except where otherwise stated."

Acknowledgements

I want to thank my dissertation supervisor Professor Raj Ranjan and his team for support and guidance throughout the project and for introducing me to the idea of solving issues faced in microservices through machine learning.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1 Purpose

Modern-day computer systems have been evolving exponentially with technological advances in hardware, software architectures, and the introduction of innovative technologies such as quantum computing. With the increase in scale and diversity of computer systems, anomalies have also evolved, attributing to erratic behaviour and severe system failure. With this comes innovations to minimise this activity, such as Microservices and Machine Learning (ML).

Microservices have become a widespread software architecture used by companies like Amazon, Netflix, eBay, and Uber (Richardson, 2021). Microservices break down a monolithic application by functionality into smaller, loosely coupled services that communicate in lightweight standardised protocols such as REST. This design architecture brings many advantages such as scalability, resiliency, and faster time to market due to the system's modularity. These advantages make it a suitable software architecture for the Cloud as each service can be developed, tested, and deployed through continuous development and integration.

Companies such as Uber reportedly have used over 2200 microservices as of 2020 (Gluck, 2020) to ensure a smooth user experience of their application. Each service differs in configuration, hardware, location, and complexity of communications between services happening in real-time. One service experiencing a fault or failure could disable features or compromise users. This shift in software development has introduced new and unexpected anomalies which need to be detected and diagnosed through understanding the scope of a failure in the modern world of the Cloud.

The high number of faults led to the creation of Chaos engineering, which is an approach pioneered by Netflix. Netflix created multiple open-source tools such as Chaos Monkey and The Simian Army (Butow, 2021). The underlying idea is to run experiments in the form of fault injection into a running system to monitor the scale of the injected fault. Monitoring fault injection allows for a failure to be observed and identified when occurring outside of an experiment. With this approach, large-scale datasets can be produced by monitoring solutions with the involvement of fault injected anomalies to predict if real-time data points are anomalous. A system created to detect anomalies can then be deployed into the Cloud to alert developers at the time of a fault occurrence.

In the last decade, machine learning has been used for many anomaly detection problems allowing for data points that do not fit a typical pattern to be identified in large datasets. Previously, the primary use cases for this technique have been detecting financial fraud, irregularities in time series, and intrusion detection (Johnson, 2020). A similar approach can be used with metrics from monitoring microservices to localise different types of faults to produce a model that can detect anomalies in real-time data.

In this paper, I hope to apply previously proven methods to a data set produced by monitoring multiple microservices to identify anomalies and localise the fault to a specific service or hardware component. Multiple models will then be compared using the confusion matrix to distinguish the high-achieving approaches to determine the best solution for the monitoring service setup.

1.2 Document Map

This document is divided into the following chapters:

- **Chapter 1 – Introduction**
  This chapter will discuss the concept of microservices, where they are used, and methods to detect anomalies. The following section includes the motivation behind the project, a well-defined aim and objectives, and a schedule.

- **Chapter 2 – Background Research**
  This chapter contains prior research into the project's domain knowledge, such as the history of microservices, issues they face, and current monitoring solutions available. Multiple anomaly detection models, including how they relate to the project, are included in this section, backed by research from papers and online.

- **Chapter 3 – Implementation**
  This chapter covers the project's development phase alongside an in-depth analysis of the dataset. Every model discussed in the background research has been implemented with a training and testing strategy. The design decisions taken and optimisations to ensure a high-quality final product will be discussed in this section.

- **Chapter 4 – Results and Evaluation**
  This chapter puts forward the project's final product in the form of results displayed in graphical representations. The metrics in which the models have been compared are outlined with descriptions. An explanation is provided to give meaning to the results, with an overview of if the technical requirements put forward in the introduction have been successfully met.

- **Chapter 5 – Conclusion**
  Comprised in this chapter is a summary of the paper linking back to the main aim and objectives of the project alongside thoughts towards the solution solving the initial problem.

1.3 Project Aim

Investigate current issues with anomaly detection in a microservice architecture and implement appropriate Machine Learning techniques to improve how we currently detect anomalies.

1.4 Project Objectives

1. **Research and identify three existing machine learning models for detecting anomalies through research papers**

Develop an understanding of the different machine learning techniques in which anomalies can be detected. Comparisons between models will be made to select different techniques to produce a range of results. This objective will be achieved when I select three appropriate techniques that can be implemented with my given dataset.

2. **Explore the current limitations and difficulties of detecting anomalies in a microservice architecture**

Exploring the current limitations, will mitigate against repeating previous projects' mistakes and identify why an anomaly may have happened. This research will prove helpful when working with the datasets produced by the microservices to conclude how and why the model detects an anomaly.

3. **Utilise online resources to implement machine learning techniques on my given datasets**

The internet is full of resources outlining and explaining how numerous machine learning techniques work and how they can be implemented in various programming languages. The objective will be complete once resources have been explored and a given technique is applied to my dataset.

4. **Train and test the three machine learning models to improve the performance**

This objective follows the previous, as it is essential to implement the ML models and improve the models for the final comparison to use an optimised model version. Otherwise, the results may be misleading if one model is neglected in optimisation.

5. **Evaluate and compare the different machine learning techniques to produce a conclusion and consolidate my findings**

Once the techniques have been implemented, a comparison of the three models can be performed to produce metrics which will allow the models to be evaluated. The results will gauge the performance of each model and fulfil the aim once a conclusion is made.

1.5 Project Requirements

To achieve the above objectives, the following requirements have been outlined

1.5.1 Functional Requirements

1. Detect labelled anomalies for both Binary and Multiclass Classification using Machine learning
2. Diagnose the microservice in which an anomaly occurred
3. Produce graphical representations of the performance of a model
4. Create the product to be able to be deployable into any environment

1.5.2 Non-Functional Requirements

5. Obtain an accuracy, precision, recall and F1 score above 90% for at least one model
6. Ensure scalability with the number of observations in both training and predication phases
7. The model's time to train and predict must be suitable for the number of observations
8. Follow the implementation of a proven anomaly detection algorithm correctly

1.6 Project Schedule

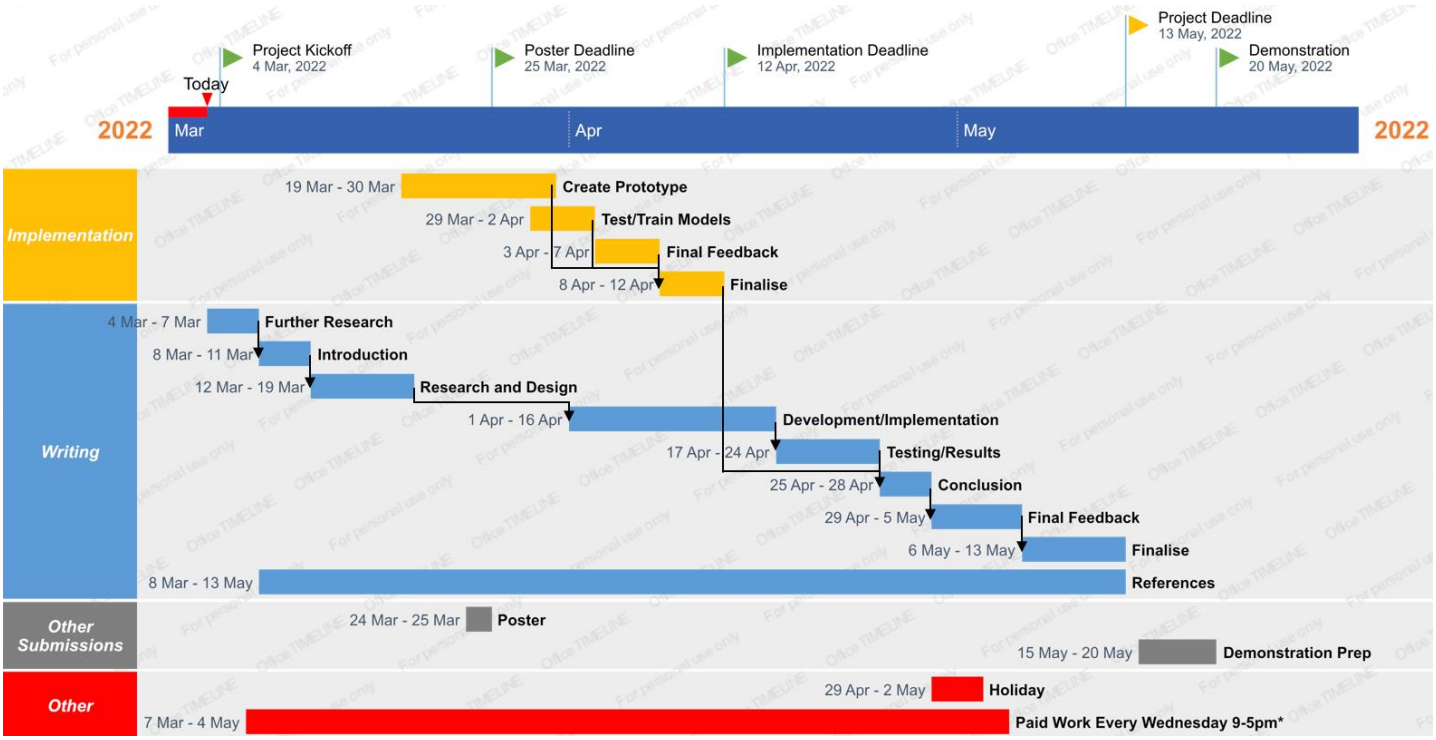

Figure 1: Schedule of the Project

1.7 Explanation of Schedule

Figure 1.1 displays an overview of the project, which will take place from March to May, alongside time allocations to specific tasks to ensure the project's goals are completed by the stated deadline.

The plan has been split into four parts Implementation, Writing, Other Submissions and Other. Writing and implementation have sub-tasks that include dependencies that must be completed to start the following task. Tasks happen in parallel to ensure time is not wasted when waiting for feedback. Along the top are milestones stating a deadline for when essential tasks must be completed.

Other Submissions include a poster and demonstration of the final product. The other section includes paid work I do and a planned holiday at the end of April. One thing to note is that the schedule was

delayed by one week extending the finalise section to the 20th of May with the demonstration on the 24th.

1.8 Summary of Results

Binary Classification and Multiclass Classification had the same models perform the best, with Random Forests achieving a high score of 0.99 in accuracy, precision, recall, and f1-score. K-Nearest Neighbor followed them with scores ranging from 0.93 to 0.91 and an impressive computation time of 3-5 milliseconds compared to Random Forests, 30-40 milliseconds. The third highest achieving model was Support Vector Machine (Polynomial), with scores ranging from 0.91-0.87 and a low 7-10 millisecond compute time. Naïve Bayes performed the worse, achieving results in the range of 0.40-0.85, with the lower scores found in the multiclass classification section.

Chapter 2: Background Research

The following section is the research conducted prior to the implementation of the project. 2.1 contains research solely on microservices to establish domain knowledge, whilst 2.2 focuses on the existing monitoring solutions and the solution used in this project. 2.3 dives into anomaly detection methods and how they are implemented.

2.1 Microservices

2.1.1 History of Microservices

A preliminary term “Micro-Web-Services” was used by Dr Peter Rodgers in 2005 during a conference, with the term we know today as “Microservices” being introduced in 2011 (*What is Microservices Architecture | Sumerge*, 2020). After numerous case studies presented at large conferences, corporations like Netflix and Amazon adopted microservices as their flagship software engineering approach. Specifically, Netflix decided to switch after experiencing massive database corruption in their monolithic application due to a missing semicolon highlighting the system's vulnerability through a single point of failure (Labs, 2016).

Netflix continued to experiment and evolve microservices which they still use today, with over 1000 microservices working together in a single ecosystem to contribute to an organised application for streaming online media (*The Story of Netflix and Microservices*, 2020). The Cloud quickly became synonymous with microservices due to similar advantages the Cloud offers, leading to compounded benefits.

The concept of Cloud computing was introduced much earlier, becoming mainstream in the mid-2000s with the emergence of the “Big 3”. The Big 3 include Google Cloud Platform (GCP), Microsoft Azure, and Amazon Web Services (AWS) (*Cloud Computing History and Evolution | Unboundsecurity*, 2021). Initially, cloud providers focused on reducing the on-premises expenditure of infrastructure for their clients by providing on-demand provisioning of resources for applications. For example, a small e-commerce website may need more bandwidth for servers at peak times (midday Sunday), while they would experience less traffic at 2 am on a Tuesday. Instead of the small company paying for expensive servers where they may only take advantage of the total bandwidth 10% of the time throughout the week, a cloud platform can provide resources when needed and only charge for what is used.

Microservices and the Cloud share similar benefits, such as scalability, resiliency, and faster time to market for applications. Because of this, it has become an industry standard to combine both technologies in-order for companies to ensure their application is modular, robust, and highly available (*What Are Microservices? And How Do They Work?* 2022).

To further this ideology, containerisation, an idea conceptualised decades before, emerged through the open-source Docker Engine in 2013 (*Containerization*, 2022). With Docker came an industry standard for lightweight containers, which packaged together a singular service including all the relevant dependencies, files, and settings into a single container. Having a service packaged together in this fashion offers benefits such as security, ease of management and fault isolation. Through cloud-specific features such as Continuous Integration and Continuous Delivery (CI/CD), containers can be easily and quickly distributed across servers anywhere in the world. Containerisation deployable through CI/CD is common today, where smaller specialised software engineering teams will work on a specific section of an application challenging the old mentality of a developer needing to know how an entire system works.

Multiple containers can then be deployed onto a single machine, all sharing the same operating system, which is opposed to the virtual machine ideology of each instance having an independent operating system. Notable advantages of containers are that they are typically smaller in capacity than a virtual machine and require less time to start up due to fewer operating systems needing to be installed. The smaller footprint of containers leads to more containers being able to be created and used in the same space than if they were virtual machines.

2.1.2 Issues cloud microservices face

As established, microservices and the Cloud have created a dynamic environment where systems can be drastically different from one another. Before this, most systems would be running off the same servers, using the same programming language and in the exact location. With the introduction of heterogeneous systems, unexpected anomalies have occurred, leading to the need to identify these issues and the need for a more comprehensive solution as traditional strategies become ineffective.

Most cloud providers offer a service for identifying issues, but as the competition of cloud products increases, so does the number of companies using a mixture of different providers. When a mixture of providers is used, a cloud-specific monitoring service becomes ineffective. For example, AWS may be the best solution for running servers, whilst GCP may have a more extensive machine learning library leading to Amazon's CloudWatch not being able to monitor resources in GCP.

The term "multi-cloud" has since been coined, describing a deployment model that spans its services across multiple cloud providers (*Multi-cloud vs. hybrid cloud: What's the difference?* 2022). Another term introduced, "hybrid clouds" describes resources that run over a combination of private and public environments. A reason for this may be to have complete control over a section of an application. For example, a database containing sensitive information such as customer banking data would be preferably stored in an on-premises data centre to provide greater security and comply with government laws on data protection. Many companies take a further step to blend together multiple public cloud providers and a hybrid cloud, which increases the complexity of the overall system.

Modern computer system ecosystems are incredibly heterogeneous, with one application being spread across thousands of microservices and the possibility of being hosted across both hybrid and multi-clouds. This heterogeneous nature of systems has made detecting anomalies extremely difficult as workloads will be executed on various hardware, networks, and software configurations.

Traditional strategies to mitigate, predict or recover from faults are not appropriate in a distributed system such as microservices as they can lead to further failures. If a system in a different cloud invokes a recovery mechanism, it could cause failures in other clouds that are not monitored or accounted for, as many monitoring services are not able to monitor multi-cloud systems (Garraghan et al., 2018).

It is paramount for a monitoring solution to have complete visibility of an application and the ecosystem in which it is running. Many solutions can only effectively monitor a few resources as providers are private companies and only allow support to specific monitoring companies, e.g., Datadog's Azure integration for Microsoft Azure environments. Whilst Datadog can effectively monitor Microsoft Azure, it may not have the same success when monitoring AWS (*Microsoft Azure*, 2022).

Virtual machines were once the best way to run applications, but with the introduction of containers, many monitoring solutions do not support monitoring both virtual machines and containers. Monitoring a container is not as trivial as a virtual machine as each cloud platform, and

containerisation software will use different strategies for dividing the underlying resources. For example, Docker uses namespaces and Control Groups (cgroups) to allocate resources such as CPU, RAM, and network bandwidth from the underlying OS. Developers can then set soft limits for memory usage while virtual machines impose strict (hard) limits (Gulati, 2022). As containers share an OS, it can be difficult to directly monitor the host with the chance of soft limits increasing past their allocated limit creating interference among containers on a single host.

2.2 Monitoring Microservices, Multi Virtualisation, and Multi-Cloud (M3)

The following section takes ideas from a paper presenting the framework's configuration and motivation for creation, named "A Framework for Monitoring Microservice-Oriented Cloud Applications in Heterogeneous Virtualization Environments" (Noor et al., 2019).

The M3 framework is a microservices monitoring system developed at Newcastle University with the help of other institutions to combat current limitations in cloud monitoring solutions. Data produced from this system is used in this paper to train and test the machine learning models in the implementation chapter.

2.2.1 Monitoring Components

M3 is a system consisting of two components, the monitoring manager and the monitoring agent. Monitoring agents are placed within each container/virtual machine (VM), tracking the underlying system's performance metrics and periodically sending them to the manager. The agent treats all systems and cloud/non-cloud infrastructure the same regardless of differences. The monitoring agent named the SmartAgent is responsible for agent registration, sending data and updating agent configuration parameters. SystemAgent and ProcessAgent extend SmartAgent, with the former monitoring the system as a whole and the latter monitoring a specific process. An agent is created using the SIGAR library, allowing access to multiple application programming interfaces (APIs) to identify system metrics, including Memory usage, Free Memory, Network usage etc.

The monitoring manager, deployed on an isolated server from other services, stores all the data sent via HTTP API requests from agents into a MySQL database. An API is provided for accessing data. Data can be transferred in a centralised or decentralised architecture. Centralised means all data is sent to the manager and has direct communication but can be the victim of a single point of failure. Decentralised means each Cloud has a local monitoring manager collecting information from all containers, with one global manager collecting data from the local containers.

2.2.2 System

The system is a three-tier architecture made up of four main Microservices:

- User Interface service (web tier)
- Books service (application tier)
- Purchase service (application tier)

- MySQL (data storage)

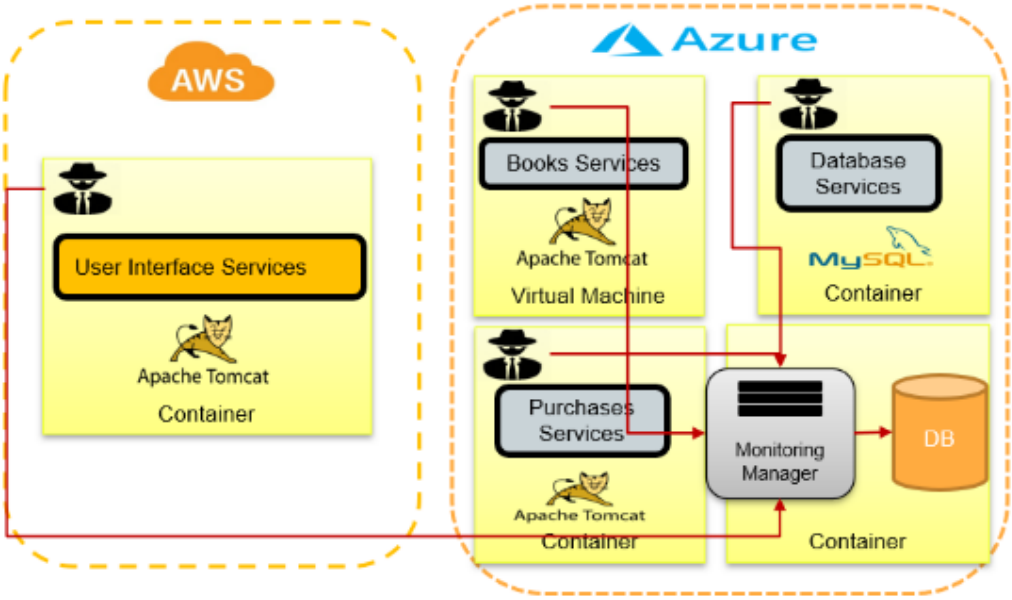

Figure 2: Architectural Diagram of M3

Above is an architectural diagram of M3 showing how it can monitor both AWS and Azure simultaneously in a multi-cloud setup. The services can be either a container or virtual machine, which many existing monitoring solutions fail to support, motivating the need for M3.

Experiments were conducted with varying workload configurations and measuring system parameters such as CPU, memory, and latency. Docker was used for the containerisation. Latency results show that when containers were used, the latency was greater than the virtual machines concluding that virtual machines make use of the system more efficiently. In conclusion, M3 monitored microservices in multiple scenarios with accurate metrics being recorded. Further testing of this system can be found in the paper, with more research needed to see how M3 compares to existing solutions.

2.3 Existing systems for monitoring and detecting anomalies

Multiple monitoring solutions can be deployed onto the Cloud to detect anomalies and alert developers. However, many solutions are either platform-specific or provide limited functionality when monitoring hybrid and multi-cloud architectures. This section will discuss the different solutions and compare them to the solution I use in this paper.

Before discussing available solutions on the market, it is essential to define parameters that create an excellent monitoring solution:

- Supporting both hybrid and multi-cloud
- Support both virtual machines and containers
- Provide deep analytics and metrics

Fulfilling the above criteria is vital in creating a solution which can monitor every aspect of a system and its components, compared to using a combination of technology which would not be optimal.

Two monitoring methods exist, which will be referred to below, white and black box. White box is the monitoring of applications running on a server, metrics include the number of HTTP requests, MySQL queries and the number of users using a web application in a specific period. Black Box is the

monitoring of servers with a focus on the host itself, such as CPU usage, memory, and disk space (Rogers, 2018).

2.3.1 Amazon CloudWatch

CloudWatch is a monitoring and management service launched in 2009, providing data, metrics, and insights for resources and infrastructure provisioned on AWS and can also monitor on-premises infrastructure. Amazon offers this in a single platform with a dashboard which can alert developers when an operational change is needed or automate changes through CloudWatch Events and auto-scaling.

The functionality allows for data collection, monitoring, analysing, and automatic response to change. However, CloudWatch does not allow multi-cloud monitoring as cloud providers cannot access other cloud providers' resources to gather metrics (*Amazon CloudWatch - Application and Infrastructure Monitoring*, 2022).

2.3.2 Microsoft Azure Monitor

Azure Monitor attempts to maximise performance and cost savings like CloudWatch by collecting and analysing data from Azure-specific and on-premises resources. Azure splits data into metrics and logs. Metrics are numeric data from monitoring resources into a time series database where logs are designed to collect and organise the metric data. Logs provide various data types, which can be queried for more specific insights into the microservice ecosystem. (Staff et al., 2021)

Both CloudWatch and Azure Monitor are highly similar in functionality and capabilities. The only reason for picking one over the other is dependent on which cloud service you are using. Because the cloud providers are competitors and developed in-house, the ability to monitor an Azure resource from Amazon CloudWatch is a long way away. A third-party option would be the only option to obtain a full-scale view of a microservice ecosystem using a multi-cloud configuration.

2.3.3 Datadog

Datadog is a solution which started development in 2010 by French CEO Oliver Pomel and President Alexis ê-Quôc (*The Incredible Rise of Datadog: An Entrepreneurial Success Story*, 2022). Datadog's most recent version is Agent 7, providing performance metrics and event monitoring for various infrastructures and cloud services, including databases, servers, and tools. DataDog offers several metric types: count, rate, gauge, histogram, and distribution, all serving a different purpose. The count calculates the sum by adding submitted values in a time interval suitable for tracking network traffic. The rate takes the count and divides it by the time interval's length for per second metrics. Gauge takes values reported in an interval, suitable for RAM and CPU usage. Histogram reports five different values summarising submitted values: count, average, median, 95^{th} percentile and max. Distribution summarises values submitted during a time interval across all hosts in the environment (*Metrics*, 2022).

Datadog runs on a host, collecting data and sending it to a database. It can be deployed to both on-premises or cloud infrastructure in a range of languages such as Java, Python, PHP, and .NET (Gillis, 2022).

2.3.4 Prometheus

Prometheus is a monitoring solution developed by SoundCloud in 2012 in response to previous solutions (Graphite and StatsD) not handling microservices architecture efficiently and concisely (Horovits, 2020). Prometheus performs primarily white box monitoring by scraping HTTP endpoints of services and uses the information gathered to identify anomalies in metrics such as throughput and response time. Data collected and produced can be stored on a local database or a remote storage system for further control. Metric types include summary, gauge, counter, and histogram. The summary shows the total count of observations and the sum of values after sampling observations. Hybrid and Multi-cloud compatible (Hailey and Sensu, 2021).

2.3.5 Summary of Solutions

Monitoring Solution	Hybrid Cloud	Multi-Cloud	Container Support	VM Support	Complex Querying	Paid
Amazon CloudWatch	✓	×	✓	✓	✓	×
Azure Monitor	✓	×	✓	✓	✓	×
Datadog	✓	✓	✓	×	✓	✓
Prometheus	✓	✓	✓	×	✓	×
M3	✓	✓	✓	✓	×	×

In conclusion, Datadog and Prometheus are reasonable solutions, but the system may not be monitored effectively in more complex ecosystems that use less supported providers. For example, Datadog can monitor an IBM Bluemix Container Service instance via Kubernetes but does not support the monitoring of an IBM Cloud Virtual Server (Rosen and Rosen, 2017). This is due to reliance on the cloud provider to provide an API or extension for monitoring virtual machines rather than the monitoring solution providing this functionality natively. M3 allows any system to be monitored but does not provide deep analytics found in a paid solution such as Datadog. A trade-off between product functionality and the number of systems that can be monitored needs to be considered. In this paper, I perform anomaly detection using the ML models in the next section, which makes M3 the most suitable solution as I only need barebone metrics as opposed to functionality such as alert systems that Amazon provides

2.4 Machine Learning

Multiple ML models which have previously been used for anomaly detection are discussed in this section alongside concepts such as different types of classifications. The relevance to this project is that the ML models will be able to be applied to my dataset, and understanding the inner workings will ease implementation.

2.4.1 Binary Classification vs Multiclass Classification

Binary classification is the task of allocating a label to a data point picked from a maximum of two classes based on a classification rule. In the context of this paper, normal and anomalous would be the two classes.

Multiclass classification is the idea of labelling data points but with an option of more than two classes. For example, a data point could belong to a CPU-specific anomalous class, an anomalous memory class, or a normal class. Multiclass classification is not precisely anomaly detection as when more than two classes are introduced, the task becomes a classification problem.

2.4.2 Naïve Bayes

Naïve Bayes is a family of algorithms based on Bayes Theorem and can be used for classification. Naïve Bayes uses probability to classify data points, assuming that all attributes of a data point are independent of each other. For example, an anomalous CPU data point may have a high CPU and throughput value. The high throughput value could explain the high CPU value, but with Naïve Bayes, this correlation would be ignored, and each value is treated independently.

Naïve Bayes performs better when using categorical features than numerical ones, which may hinder the algorithm's performance in this paper, as metrics such as CPU usage, memory, throughput, and response time are numerical. Bayes Theorem can be found below, which calculates a conditional probability.

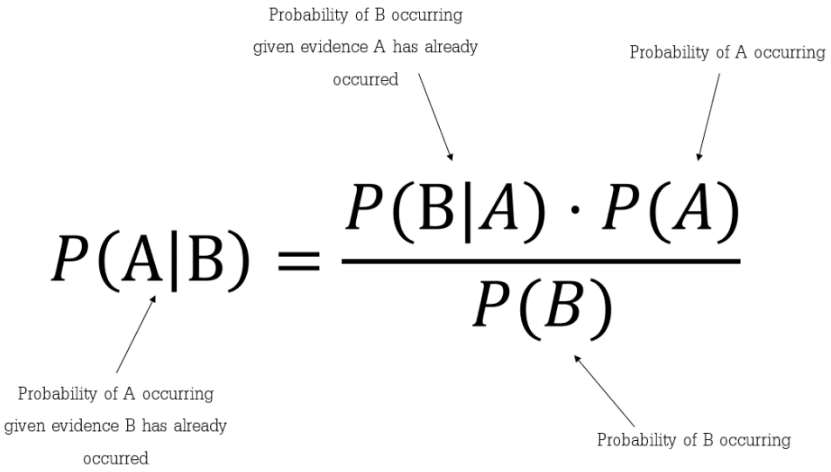

Figure 3: Bayes Theorem Formula (Rossman, 2019)

- ***P(A|B)*** is the conditional probability of event ***A*** occurring given ***B*** is true, also referred to as the posterior probability
- ***P(B|A)*** is the conditional probability of event ***B*** occurring given ***A*** is true or the likelihood of ***A*** given ***B***
- ***P(A) and P(B)*** are the probability of ***A*** or ***B*** occurring***,*** also referred to as prior probability

Below is a simple example of the algorithm with relevant data to this paper involving a Microservice category, CPU numerical value and a label. The following calculations have been adapted from an article (Chatterjee, 2019).

Raw Data

Microservice	**CPU**	**label**
Bookshop-ui	42	normal
Bookshop-ui	2	anomalous
Bookshop-ui	68	normal
Bookshop-ui	55	normal
Bookshop-ui	92	anomalous

Frequency/Likelihood Table for Microservice

Microservice	**normal**	**anomalous**
Bookshop-ui	3/5	2/5

Frequency/Likelihood Table for CPU

		CPU	**Mean**	**Standard Deviation**
Label	anomalous	2, 92	47	63.6
	normal	42, 55, 68	55	13

Test Data

Microservice	**CPU**	**label**
Bookshop-ui	50	normal

Maximum Likelihood Estimation (MLE)

For the label prediction to be calculated, the likelihood of the test data belonging to either the normal or anomalous class must first be calculated. This is calculated by selecting a distribution such as Gaussian. The distribution can be altered to Multinomial or Bernoulli, depending on the data. The mean and standard deviation previously calculated for each class can be substituted into the following formula.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ mean of x
$\sigma =$ standard deviation of x
$\pi \approx 3.14159 \ldots$
$e \approx 2.71828 \ldots$

Figure 4: Gaussian Distribution Formula (Normal Distribution (solutions, examples, formulas, videos), 2020)

Normal Case

x = {Microservice= Bookshop-ui, CPU = 50}

P(class = normal| ***x***) = ***P***(class = normal| CPU = 50) * ***P***(class = normal| Microservice = Bookshop-ui)

$$f(50) = \frac{1}{13\sqrt{2\pi}} e^{\frac{-(50-55)^2}{2*13^2}}$$

$$f(50) = \frac{1}{32.586168} e^{\frac{-25}{338}}$$

$$f(50) = 0.030688 * e^{-0.073964}$$

$$f(50) = 0.083419^{-0.073964} = 1.201676$$

Anomalous Case

P(class = anomalous| ***x***) = ***P***(class = anomalous | CPU = 50) * ***P***(class = anomalous | Microservice = Bookshop-ui)

$$f(50) = \frac{1}{63.6\sqrt{2\pi}} e^{\frac{-(50-47)^2}{2*63.6^2}}$$

$$f(50) = \frac{1}{159.421558} e^{\frac{-9}{8089.92}}$$

$$f(50) = 0.006272 * e^{-0.001112}$$

$$f(50) = 0.017049^{-0.001112} = 1.004537$$

Prediction calculation

2/5 and 3/5 values are derived from the Frequency/Likelihood Table for the Microservice feature above.

Normal Case

P(class=normal | x) = ***P***(class=normal | Microservice= Bookshop-ui) * = ***P***((class=anomalous | CPU= 50)

P(class=normal | x) = 3/5 * 2/5 * 1.20165 = **0.288396**

Anomalous Case

P(class=normal | x) = ***P***(class= anomalous | Microservice= Bookshop-ui) * = ***P***(class=normal | CPU= 50)

P(class=normal | x) = 2/5 * 3/5 * 1.004537 = **0.24108888**

It can be observed that the result for normal is greater than anomalous, meaning the data point will be labelled as normal, which is also a correct prediction.

2.4.3 Random Forests

Random forests are a supervised machine learning algorithm which builds multiple decision trees and aggregates the results picking the mode result to create a final prediction. This idea is derived from the “bagging” method, which states that a combination of models increases the overall result.

A basic decision tree has multiple questions throughout with a singular question at each branch depicting the direction to navigate the tree to a specific class. Binary classification would include two classes, normal and anomalous. In the below example, we can see an anomalous data point would be when the CPU usage is above 80 and throughput is below 100.

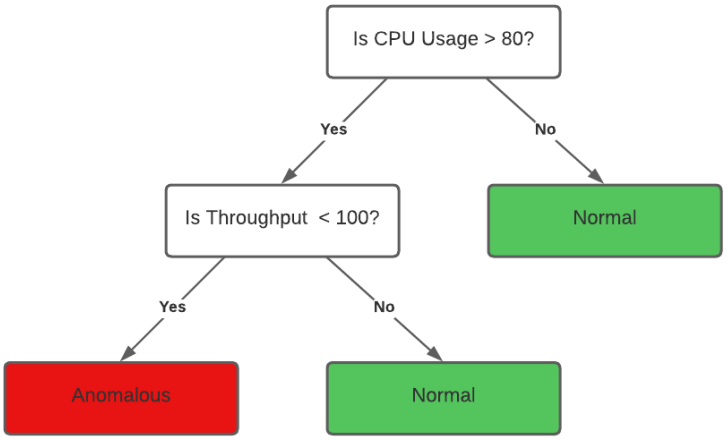

Figure 5: Decision Tree predicting normal and abnormal classes

Random forests generate a random subset of features which reduces the correlation between decision trees, known as the random subspace method, which helps to minimise feature space and time/storage spent per tree. This method limits the effect of the curse of dimensionality, which is the idea that when dimensions increase, so does space between data points (Baskin et al., 2017).

Below is an example of the random forest working for a binary classification problem of just normal and anomalous classes. This idea can be extended to the multi-classification problem.

Binary Classification

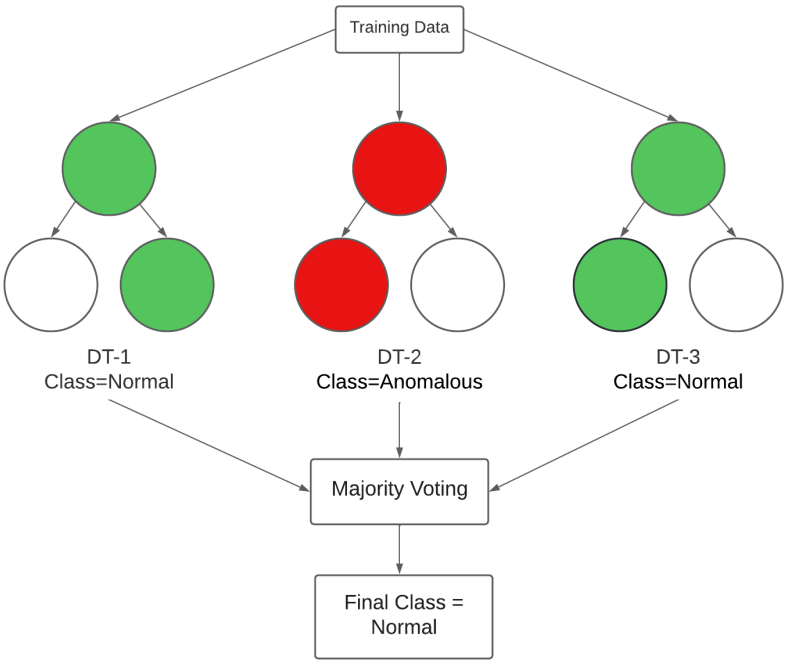

Figure 6: Random Forrest predicting normal and abnormal classes

Binary is straightforward, with the chance of one of two classes being predicted per tree.

Multiclass Classification

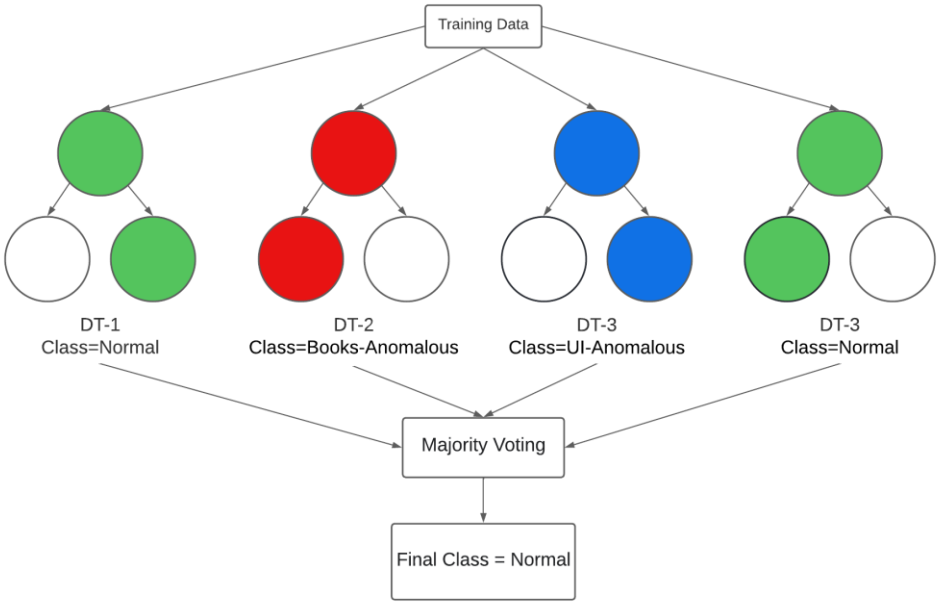

Figure 7: Random Forests predicting multiclass

The above Random Forest predicts three classes: Books-Anomalous, UI-Anomalous, and normal. The idea is that an anomalous class can be compounded with a specific microservice to offer further insights. The reason for one of the trees deciding a different label could be due to the features it has chosen out of the features subset, as each tree is different. It is important to note that a similar implementation, “Isolation Forest" can be used for anomaly detection exclusively for binary classification. Random forests will be needed for multiclass.

2.4.4 Support Vector Machine (SVM)

SVM is a supervised machine learning model which natively only supports binary classification involving two classes. SVMs use a combination of hyperplanes to separate the data points in order for classes to be identifiable depending on where the point lies in the dimensional space. For multiclass classification, the problem is broken down into multiple binary classifications. As SVM does not support multiclass classification natively, we must employ one of the two below strategies.

One-vs-One

m = number of classes

Binary classifier per each pair of classes with $\frac{m(m-1)}{2}$ SVMs needed.

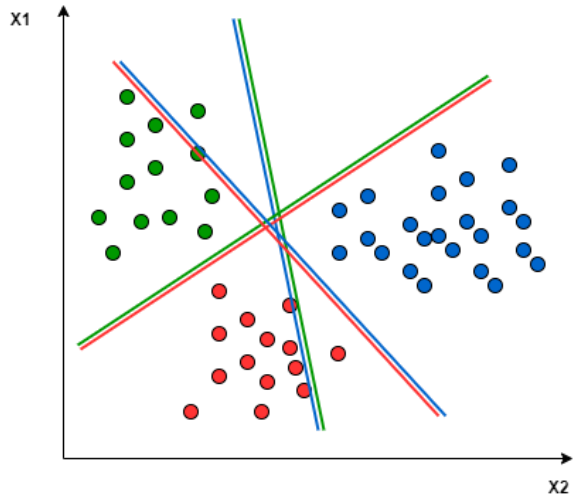

Figure 8: One-vs-One SVM with three classes(Multiclass Classification Using Support Vector Machines, 2021)

The above figure shows that a separate hyperplane has been placed for every pair of classes to segregate the data points into classes.

One-vs-Rest

Binary classifier per each class with ***m*** SVMs needed.

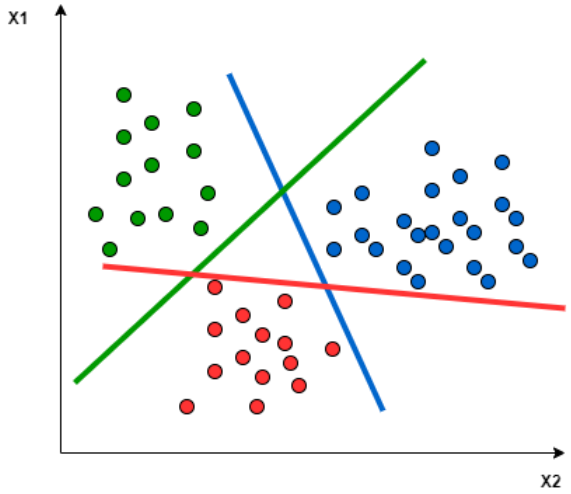

Figure 9: One-vs-Rest SVM with three classes (Multiclass Classification Using Support Vector Machines, 2021)

Only one hyperplane can be seen for each class and points to one side become apart of that class.

Non-linear Data

If a hyperplane cannot be drawn due to non-linear data, dimensions can be added to uplift the data into a dimension in which a hyperplane can be fitted. Increasing dimensions can be computationally expensive, so a kernel function should be used.

The kernel is a similarity function which transforms data into a higher dimension by calculating the dot product between two points. In other words, it defines how similar two data points are in order for a classification to be made by calculating how far apart they are. Further away would mean less likely to be similar, and closer together increases similarity. The Kernel function is a parameter of the SVM algorithm provided by a domain expert for the best results prior to training. Different Kernel functions include linear, non-linear, polynomial, radial basis function (RBF), and sigmoid.

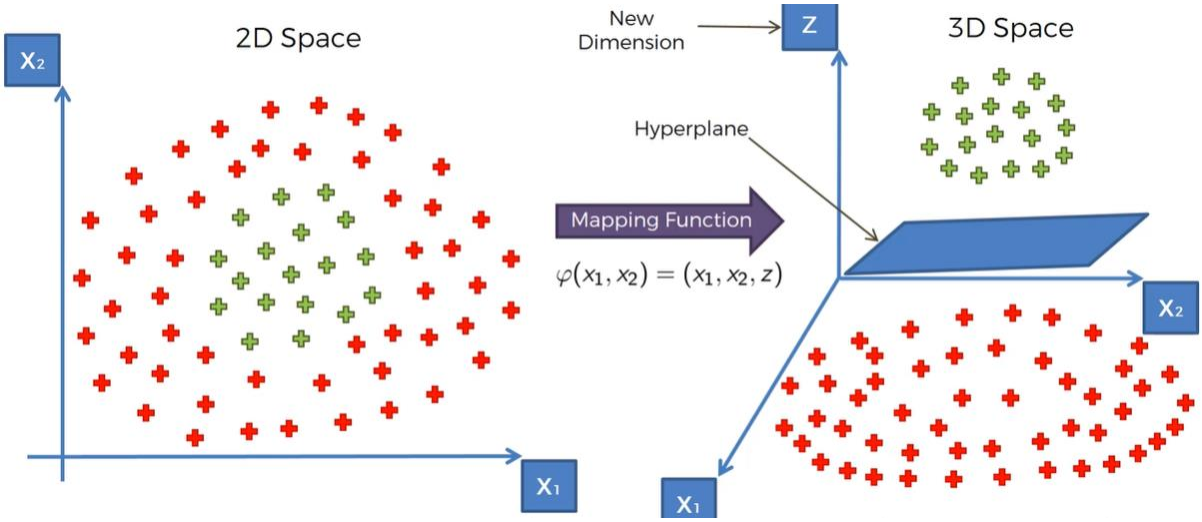

Figure 10: SVM RBF function adding a dimension to a 2D feature space (Saxena, 2020)

Figure 10 shows that a hyperplane cannot be drawn to linearly separate data into classes when the data only has two dimensions. When an extra dimension is added (***Z***), the data can be separated, allowing classes to be established.

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

Figure 11: RBF kernel Function (Sreenivasa, 2020)

The RBF kernel is widely used for non-linear data problems, the above image shows how data has been transformed into 3D space, and a hyperplane has been fitted to separate two classes. Without seeing the data in this paper, I cannot be sure which function would be best, but knowing why a function is preferred in specific scenarios will help when picking the kernel.

2.4.5 K-nearest neighbor (KNN)

KNN is a supervised machine learning algorithm which works by finding distances between data points. KNN will compute the distance from one single data point to all other data points in the sample and determine the closest points to that data point. It will pick ***k*** number of points, e.g., if ***k*** is 3, it will pick the 3 closest points and use the most frequent label to decide which class the selected data point should be mapped to. If 2/3 of neighbouring data points are labelled normal, the selected data point will be given the normal label. The distance is calculated using an equation, with Euclidean distance being the most popular.

$$d(u,v) = \sqrt{((u_1 - v_1)^2 + (u_2 - v_2)^2 + \cdots + (u_n - v_n)^2)}$$

*Figure 12: Euclidean Distance Formula (*Multiclass Classification- Explained in Machine Learning*, 2020)*

The main hyperparameter is ***k*** which, if low, will show high variance in the model and be overfitted to the data sample. A large ***k*** will have low variance but high bias, implying that a middle ground must be met through experimentation.

Chapter 3: Implementation

Due to the complexity of setting up a multi-cloud microservice architecture and the fees this would involve. This paper will use a research dataset consisting of monitoring metrics using the M3 design previously discussed in the background research section. This approach allows for focus on the machine learning section of the dissertation while still using domain knowledge of M3 to explain results in the results section. 3.1 will discuss the tools and technologies used in the project with a brief explanation of why they have been chosen. 3.2 explores the research dataset with data analytics to establish how the data will be classified and labelled. 3.3 involves data processing that must be taken prior to 3.4, where the ML models are implemented.

3.1 Development Tools and Technologies

3.1.1 Programming Language and Libraries

The entire project can be programmed using Python and various libraries to analyse, clean and implement machine learning models on the dataset. The programming will take place inside an Integrated Development Environment (IDE), allowing for an extensive collection of open-source libraries and support to be harnessed for ease of implementation.

Multiple languages could have been used, such as the close competitor of Python, R. The reason for the preference towards Python is that R is more focused on statistical analysis, whereas Python is a multi-purpose language. Being a multi-purpose language means it can be used for data science projects and further applications such as an Application Programming Interface (*Python vs. R: What's the Difference?* 2021). For example, a developer may want to create a website which uses an API to send images to a machine learning model in-order for some analysis to be computed.

Two further criteria need to be discussed before picking Python as the preferred language.

Firstly, the developer's previous experience needs to be considered. This can decrease development time, making the final product of a higher quality, which is paramount when the development window is short. I had a few months of Python experience before this project. I had undertaken two data science modules where Python was the core language, which is significantly more experience than I have with R.

Secondly is the need for the project to be written in a multi-purpose language. Python would be the only option that provides extensive data science support alongside the support for API creation through Django or Flask. A future goal would be to integrate the final machine learning model into a working production system.

After deciding on a language, the next course of action is to identify libraries which will be helpful. The below table displays each library used in this project.

Use of Library	**Library Name**
Data Manipulation	Pandas
	NumPy
Data Visualization	seaborn
	Matplotlib
Machine Learning Models	scikit-learn

Pandas is an instrumental library for data manipulation and analysis through easy-to-use data structures such as a DataFrame and Series. Python's core data structures will not be suitable for housing large datasets being read in from comma-separated value (CSV) files. A multi-dimensional solution is needed, such as Panda's DataFrame, which allows extreme flexibility in a spreadsheet row and columns style. Solutions such as arrays, dictionaries (HashMap) or lists would not be able to offer the same versatility. Also, most machine learning models work with either arrays or DataFrame making it an obvious choice to use.

Pandas is dependent on NumPy, making it an obvious option for the project. Further to needing NumPy for Pandas, NumPy offers a host of mathematical functions alongside NumPy Arrays which are much faster as they consume less memory than regular Python arrays (*What is NumPy? — NumPy v1.22 Manual*, 2022). As data sets grow, this is highly advantageous and will make it easier for operations such as sorting or replacing values.

Data visualisation is one area in which R excels, making a replacement challenging to find for Python. Looking towards a third-party solution for visualisation, seaborn and Matplotlib are the most popular. Both are being used because seaborn provides functions such as the "pairplot" and "heatmap", which is a quicker solution than Matplotlib provides. Matplotlib is preferred for basic tasks as seaborn contains fewer plot functions. Matplotlib will be used for simple plots such as bar charts displaying the results of each model.

Machine learning models will be implemented using scikit-learn (*Documentation scikit-learn: machine learning in Python — scikit-learn 0.21.3 documentation*, 2022). Scikit-learn allows for training, testing and hyper-parameterisation of all algorithms discussed in the background research section with high-quality, easy-to-use functions. Furthermore, it provides functions for calculating confusion matrix metrics such as accuracy, precision, recall and F1-score used in the evaluation section.

3.1.2 Integrated Development Environment (IDE)

The IDE is where all the project's development will take place and be complied. The building blocks of a sufficient solution include ease of development, collaboration, and layout of code. For this project, I have chosen Google Colabs due to its easy-to-use nature and ability to format code as a notebook with isolated cells. Each isolated cell can be executed separately, removing the need for an entire code base to be re-run when modifying a single line of code found in traditional IDEs such as VSCode. Errors in the code can then be isolated and fixed, preventing time wasted on fixing bugs, which will be advantageous for a project with a small development time window.

Furthermore, Google Colabs provides code to be run on the Cloud via a GPU, removing the burden of needing a high-performance laptop for this project. Having a virtual GPU ensures performance is standardised in the development stage and eliminates the need for regular backups of workbooks as Google Drive offers autosave after every modification. Inconsistent performance of the development machine may lead to machine learning models taking a long time to compute, slowing down the process.

3.2 Data Exploration

This section describes the dataset used in this project with graphical representations and explanations of how classes and labels have been created and applied.

3.2.1 Initial Exploration

The dataset given is comprised of data from three microservices:

- bookshop-UI
- bookshop-books
- bookshop-purchases

The microservice architecture design details can be found in the Background Research section.

The dataset contained timestamps showing the monitoring of the microservices started at 17:19 on 07/07/2020 until 00:43 on 23/07/2020. Throughout this timeframe, fault injection was conducted to artificially cause anomalies which can later be labelled.

3.2.2 Features

	Microservices	CPU	mem	response	throughput	timestamp
0	bookshop-ui	41.9	283	1.033	207.43	13/07/2020 20:29
1	bookshop-books	49.6	1274	0.873	415.66	13/07/2020 20:30
2	bookshop-purchases	41.9	441	3.923	207.93	13/07/2020 20:29
3	bookshop-ui	37.9	281	1.251	206.36	21/07/2020 12:43
4	bookshop-books	58.9	1133	0.997	378.43	13/07/2020 19:50

Figure 13: Head of the dataset

Every minute for each microservice, a log of the CPU usage, memory, response time, throughput and timestamp were inserted as a row leading to 2727 rows in the dataset.

Breakdown of each feature:

- **Microservices**: Microservice the data is being monitored from
- **CPU**: CPU usage
- **mem**: memory usage
- **response**: the
  time it takes from when the system receives a request to generating a response
- **throughput**: the rate of data or messages transferred in a fixed time interval
- **timestamp**: time in which a metric was captured

The first task I conducted was to observe correlations in features and if any may need to be dropped. An early question had to be asked if a time-series based approach was to be used or if a multivariate choice would be more beneficial. I decided to originally take a time-series approach which generated poor results. This led to revisiting the features and dropping the timestamp column to take a non-timeseries multivariate approach. This led to only using the following features, Microservices, CPU, mem, response, and throughput.

3.2.3 Features Analysis

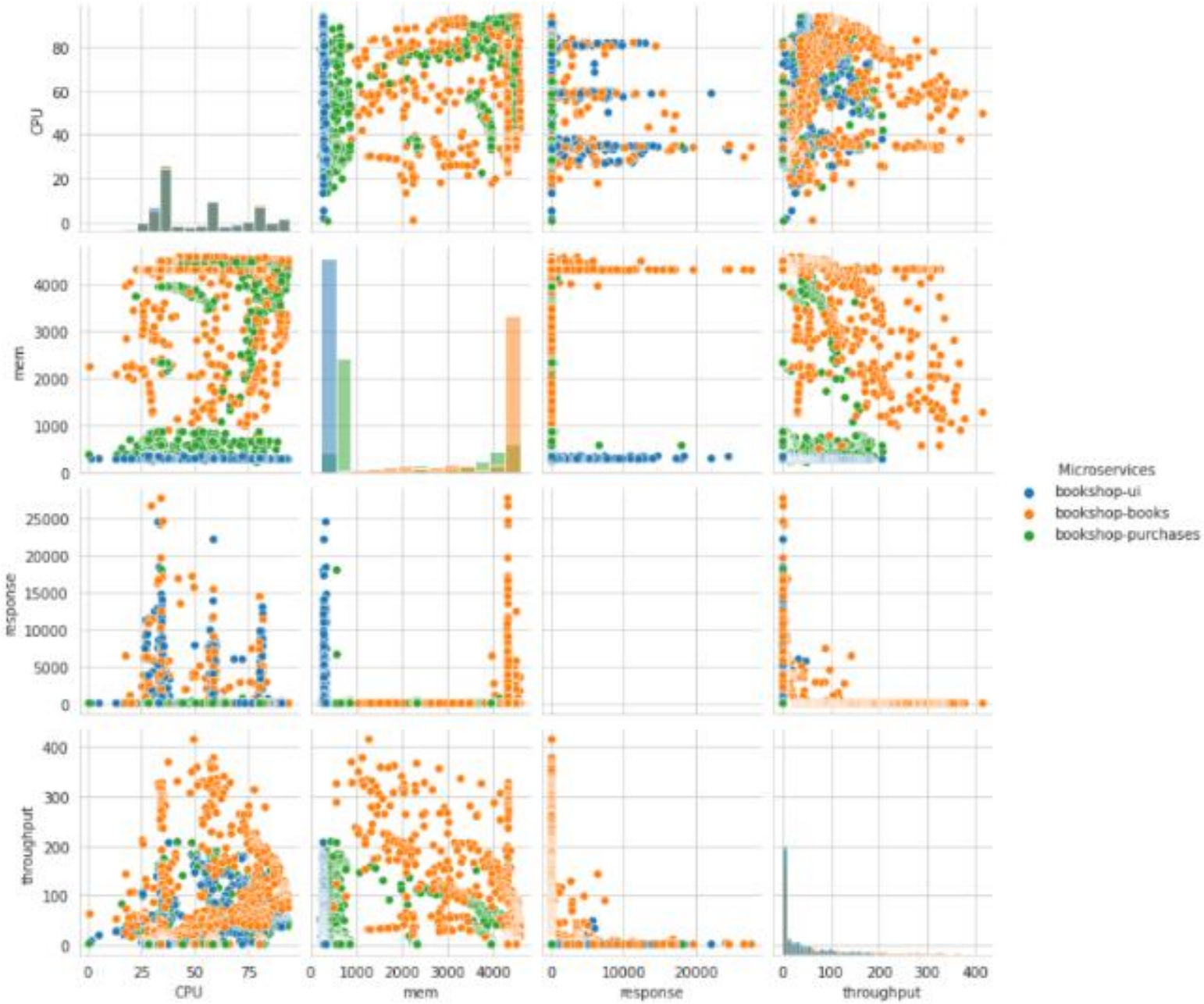

Figure 14: Seaborn pairplot of the dataset

As shown in the figure above, the values of each feature highly depend on the microservice they occur in, e.g., bookshop books have values spread out more with a higher degree of variation than the other two microservices.

Books

CPU	4.537621e+02
mem	8.132352e+05
response	1.187639e+07
throughput	8.463176e+03

UI

CPU	4.592946e+02
mem	2.902062e+02
response	9.897879e+06
throughput	2.230605e+03

Purchases

CPU	4.543899e+02
mem	2.594638e+06
response	4.068667e+05
throughput	2.114963e+03

Figure 15: Variance of the dataset split by Microservices

After further exploration, books had significant variance in memory (8.13) and throughput (8.46), with the UI having the highest variance value in response at 9.89. Purchases has low levels of variance throughout the features. This analysis makes sense when paired with the pair plot graph as the initial analysis of the graph is that bookshop-books has a greater variance.

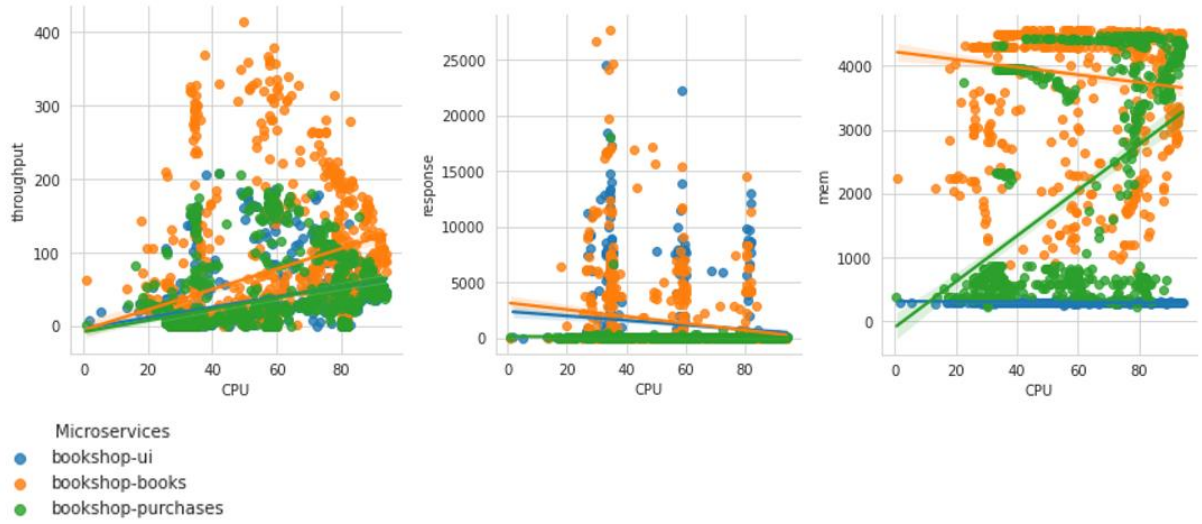

Figure 16: Correlation of each service

When calculating the correlation coefficient, it can be seen that correlations in data dramatically change between microservices. For example, in the CPU-throughput graph, all three services have a different correlation with bookshop-ui values staying very low.

Heatmap

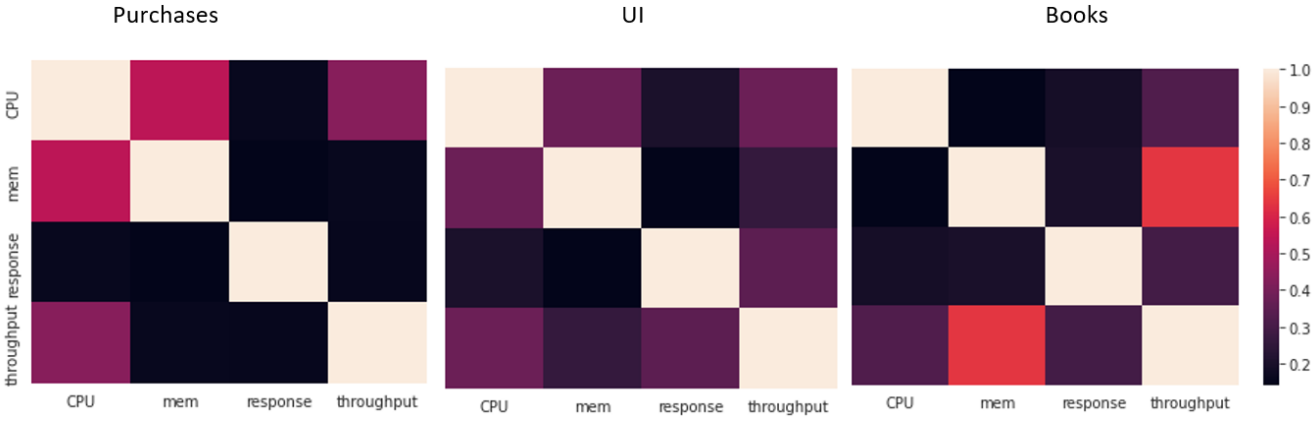

Figure 17: Heatmaps of each service

After producing heatmaps, it has become evident that correlations can be found in different features for different services. Books has a higher correlation between throughput and memory, where Purchases has a low correlation in those features and a high correlation between CPU and memory. UI seems to have little correlation between all features.

3.2.4 Binary Classification of Data

The first method of classification I implemented was Binary. Binary is simple to implement as it only involves two classes being created, normal and anomalous. To classify a single data point as anomalous, I calculated the standard deviation of a feature and deemed it anomalous if it was outside of 3 standard deviations from the mean. I did this for each feature and filled the remaining unlabelled data points with the normal label.

3.2.5 Multiclass Classification of Data

Due to each service differing in feature values, I thought it would be sensible to classify specific anomalies depending on their service. For example, if a feature value, let us say throughput is over three standard deviations (SD) from the book service's mean for throughput, the data point will be labelled “books_a” to signal that the data row is anomalous. If all values for that timestamped row are within three standard deviations, the label will be “normal”.

Four distinct classes have been made using this logic:

- **normal**: all data points in the row are within three SDs
- **books_a**: data point is outside of three SDs and originated on the books microservice
- **purchases_a**: data point is outside of three SDs and originated on the purchases microservice
- **ui_a**: data point is outside of three SDs and originated on the UI microservice

A second approach not taken in this paper would be classifying the data on components. For example, if a CPU data point from the books service is anomalous, it could have been classed as “CPU_a” and the same if the data point was from the UI service. I chose the method in this paper to be service-specific, with the ability to extend this further to components later.

This classification method was chosen because it would be difficult to accurately identify which component an anomaly had occurred in without specialised knowledge of how and when a fault injection occurred. More knowledge of the data set would allow for more specific labelling, which could be done in future versions of this paper. Furthermore, as this paper compares models’ performance to each other and not how they translate to the real world, I believe this approach should suffice. Once classes were made, the data was adequately labelled using the SD, leaving the following label distribution.

normal	1939
ui_a	361
books_a	350
purchases_a	76

Figure 18: Class Labels

Books and UI have similar amounts of faults, with purchases nearly containing seven times fewer anomalous labels. The imbalanced dataset could be due to an uneven distribution of fault injections of services or an inefficient labelling method. I cannot confirm the reasoning without further domain knowledge, but it will be an important point to remember when evaluating the dataset.

3.3 Data Pre-processing

Any dataset gathered from real-world systems is littered with impurities, leading to a machine learning model not performing to the best of its abilities. For the effects of impurities to be limited, the dataset must be pre-processed to increase accuracy and performance by removing null values, transforming categorical features, and dropping unnecessary features.

3.3.1 Reading in the dataset to Colabs

The first operation is to shuffle the dataset's ordering randomly to remove bias and diversify class labels that may be clustered together in the CSV. This will ensure the training and testing sets are more optimal as similar data tends to be sequential, leading to the training being inefficient for the data in the testing section. The dataset was labelled previously to be uploaded to my GitHub. Uploading to GitHub allows easy access to read the CSV into Google Colabs.

```
df = pd.read_csv("https://raw.githubusercontent.com/JacobJS56/Dissertation/main/Datasets/labelled_dataset.csv")
np.random.shuffle(df.values)
```

Figure 19: Code reading and shuffling the dataset

Before implementing the ML models, the dataset must first be prepared into a state that each model can take. Firstly, the categorical data must be transformed into numerical data. This is because ML models do not perform well with non-ordinal categorical features. Ordinal data follows an order or scale that the "Microservices" column does not. Panda's get_dummies() function can be called by giving the function the DataFrame and column in which one hot encoding should be applied.

```
df = pd.get_dummies(df, columns=["Microservices"])
df.drop(['timestamp'], axis=1)
```

	CPU	mem	response	throughput	label	Microservices_bookshop-books	Microservices_bookshop-purchases	Microservices_bookshop-ui
0	41.9	283	1.033	207.430	ui_a	0	0	1
1	49.6	1274	0.873	415.660	books_a	1	0	0
2	41.9	441	3.923	207.930	purchases_a	0	1	0
3	37.9	281	1.251	206.360	ui_a	0	0	1
4	58.9	1133	0.997	378.430	books_a	1	0	0

Figure 20: Dataset after dropping timestamp column and applying one-hot encoding

The previous "Microservices" column has been removed, and three columns have been added. Each of the new columns represents a category found in the "Microservices" column. Only one of the new columns will be populated with a "1" representing true for that category, where a "0" represents false. The timestamp column has also been removed as the models will not use this feature.

3.3.2 Power Transformation

In statistics, Power Transform is a commonly used technique to stabilise a feature's variance in a dataset leading to a Gaussian distribution. Most ML models will, by default, work better with a Gaussian distribution (*Choosing What Features to Use*, 2022). In this paper, I have used the Box-Cox transformation function to apply to features in my dataset.

3.3.3 Box-Cox Transformation

The core of the equation is an exponent, lambda (λ), which varies from -5 to 5. All values of (λ) are compared, and one value is picked, which is the result closest to a normal distribution.

$$y(\lambda) = \begin{cases} \frac{y^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0; \\ \log y, & \text{if } \lambda = 0. \end{cases}$$

Figure 21: Box-Cox Transformation (Transformation, 2022)

One requirement is for the inputted values to be positive, which all the features are.

The following figures show each feature pre and post the box-cox function application.

- Q-Q plots (top two graphs) is a probability plot comparing the first quantile against the second quantile. The closer the dots are to the red line, the more normalised a dataset is.
- Histograms show the distribution of data on an interval scale by displaying frequency.

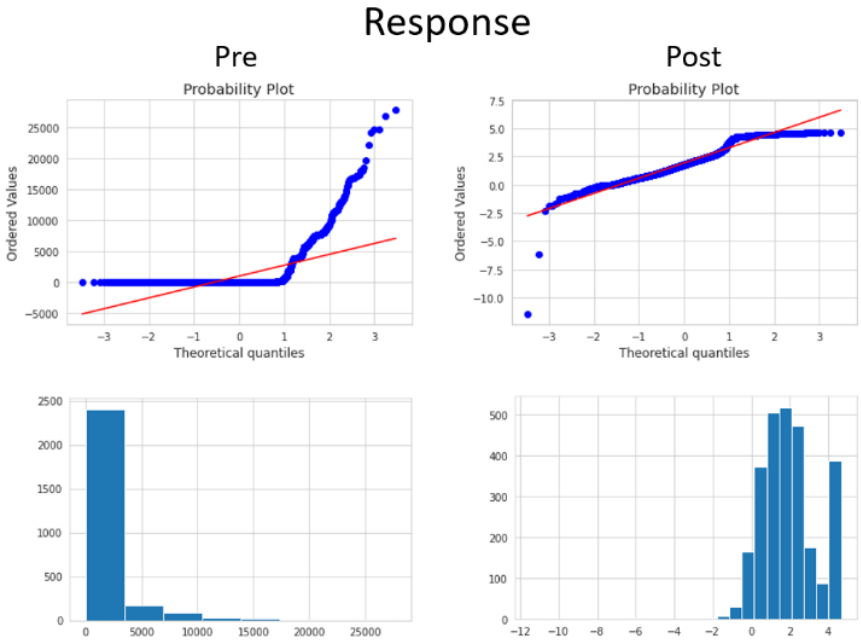

Figure 22: Response Power Transformation

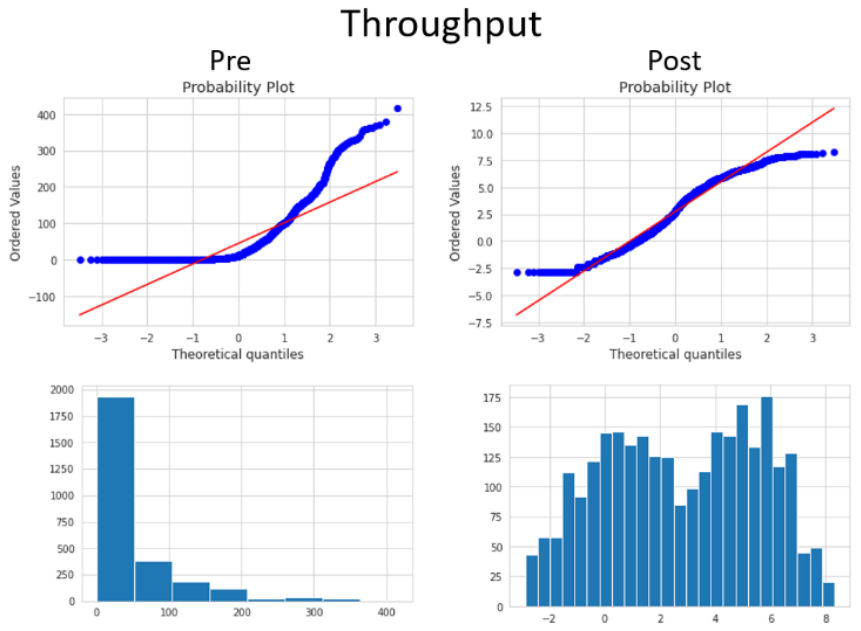

Figure 23: Throughput Power Transformation

<u>Response and Throughput</u>

- Pre: The majority of the Q-Q plot's values do not fit on the red line, and the histogram is positivity skewed
- Post: Most values fit onto the line with a histogram closer to a Gaussian distribution than before.

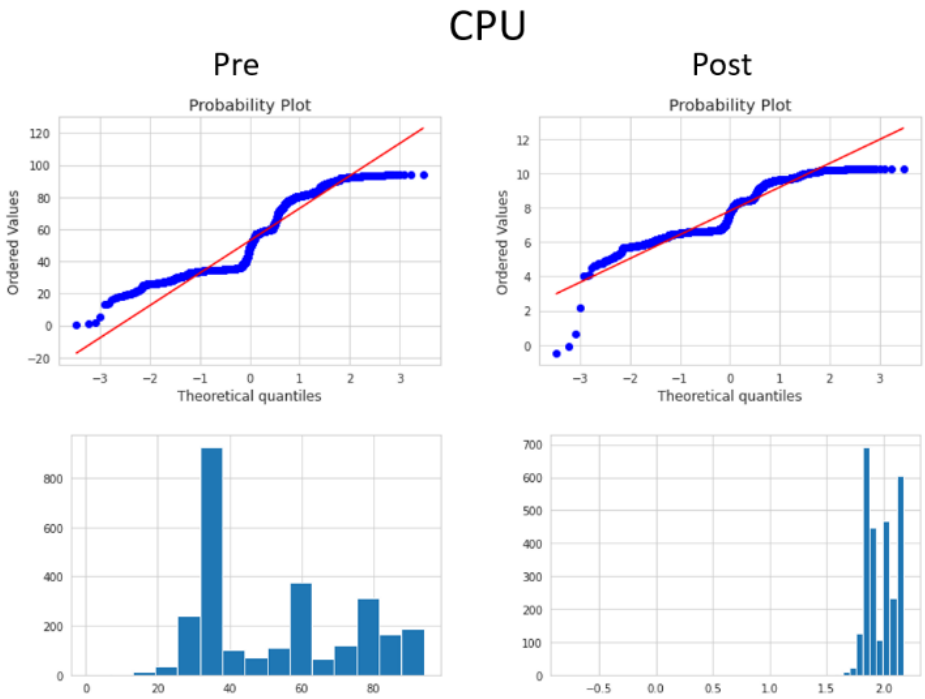

Figure 24: CPU Power Transformation

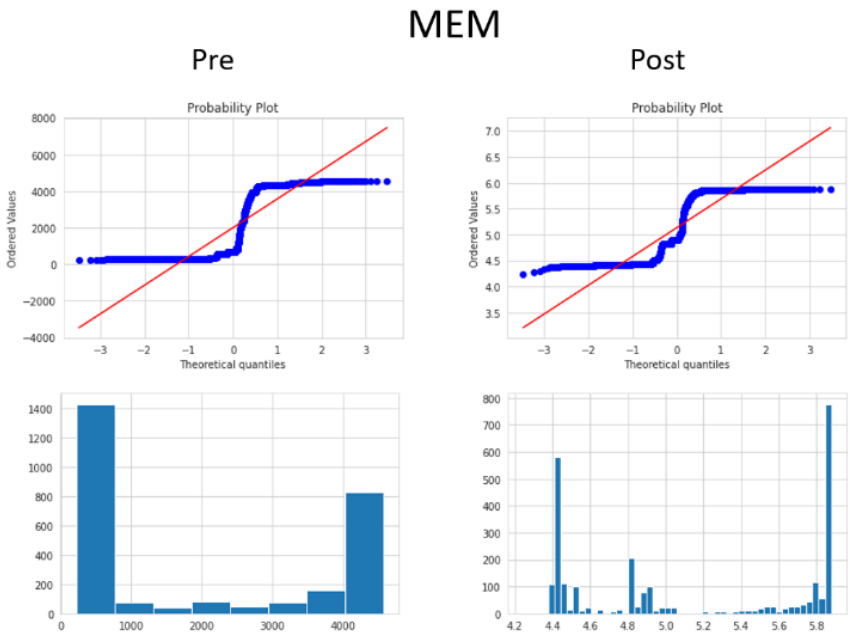

Figure 25: MEM Power Transformation

CPU and Memory

The two features did not transform to a more normalised state which led to not carrying out the transformation on these features. This may have been due to more outliers in the dataset, and removing these could help normalise the data, but this would go against the need to use machine learning to identify the anomalies. Further methods may need to be explored, with other methods like logarithmic and cubic, generating worse results when tested.

3.4 Machine Learning Models

This section explores how each model has been created, trained, and tested on the above dataset. ML models have been implemented using scikit-learn's library, including Random Forests, Naïve Bayes, Support Vector Machines, and k-Nearest Neighbor. SVM and Naïve Bayes have been implemented multiple times with different kernels/estimators to determine how different model configurations can change the results. Each model was initially created and then tested later, where the hyperparameter was optimised.

3.4.1 Terminology

Model Name	**Model Abbreviation**
Random Forest	RF
Naïve Bayes Gaussian	NB-G
Naïve Bayes Bernoulli	NB-B
Support Vector Machine Linear Kernel	SVM-L
Support Vector Machine Polynomial Kernel	SVM-P
Support Vector Machine Radial Basis Function Kernel	SVM-RBF
k-Nearest Neighbor	KNN

3.4.2 Random Forests

```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
pred = rfc.predict(X_test)
```

Figure 26: Random Forest Python Code

Sklearn's RandomForestClassifier has been implemented using the default hyperparameters found in their documentation. As this was the first model implemented, I had to ensure the model functioned as theorised by splitting the primary dataset into training and testing, which initially was a 60% train and 40% test. The predictions were generated and saved into the pred variable, which could then be used to generate evaluation metrics. As the scores generated were initially high, I felt no need to change the hyperparameters, which is why they are left as default.

3.4.3 Naïve Bayes

For Naïve Bayes, I implemented two versions of the algorithm, each using a different estimator. Picking more than one algorithm was done for two reasons. The first reason is because the code is very similar, it made it much easier to implement a second Naïve Bayes model. Secondly, it would be interesting to see how the different models performed when changes are made. In the evaluation section, the performances can be combined with the data exploration assumptions from earlier in this chapter to conclude which estimator creates the best results with coherent reasoning. Because the power transformation made some of the data negative multinomial and compliment Naïve Bayes could not be used.

Gaussian Naïve Bayes

Gaussian Naïve Bayes estimates the distribution of data using only the mean and standard deviation of the training data. Used in cases where the data is continuous which would be an appropriate choice for the dataset given as CPU, memory, throughput, and response are all continuous.

Hyperparameter Optimization

Once evaluated, it became clear this implementation needed a revision as the results were poor. The only modification that would improve the performance is to pick different Hyperparameter values.

A preliminary method of identifying high-performing hyperparameter values was to manually pick values, checking the results every time to ensure the model improves. However, automating this process would decrease time spent implementing and most likely find better values as it can test thousands more combinations.

A hyperparameter is a parameter that controls a machine learning model's learning process and is predefined prior to training. Hyperparameter optimisation is a way of automating the process of choosing the best parameters by trying multiple combinations with the best performing model being chosen. GridSearchCV, which can be found within scikit-learn's model_selection package, is a solution that enables automatic optimisation.

```
params = {
    'var_smoothing': np.logspace(0, -9, num=75)
    }
gnv = GridSearchCV(estimator=GaussianNB(), param_grid=params, scoring='accuracy', n_jobs=-1, cv=10)
```

Figure 27: Hyperparameter Optimisation Python Code

params is a dictionary (hash map) which contains the model's parameters with ranges the automation can cycle through.

GridSearchCV parameters:

- **estimator**: is the model which will be used
- **param_grid**: dictionary (hash map) consisting of configuration settings
- **scoring**: a strategy to evaluate the performance of the cross-validated model
- **n_jobs**: number of jobs running in parallel, -1 means using all processors
- **cv**: determines cross-validation splitting strategy

Python Implementation of NB-G

```
gnv = GaussianNB(var_smoothing=0.0001)
gnv.fit(X_train, y_train)
pred = gnv.predict(X_test)
```

Figure 28: Gaussian Naïve Bayes Python Code

- ***var_smoothing*** is defaulted to 1e-9 and is responsible for calculation stability by adding variance to the distribution and mitigating the issue of zero probability of a class.

Bernoulli Naïve Bayes

Bernoulli distribution is used for data that contains Boolean/binary values only present in the one-hot encoding microservices categories in the dataset. The equation below is the probability density function for the Bernoulli distribution.

$$p(x) = P[X = x] = \begin{cases} q = 1 - p & x = 0 \\ p & x = 1 \end{cases}$$

*Figure 29: Bernoulli Distribution Formula (*Bernoulli Naive Bayes*, 2022)*

```
bnv = BernoulliNB(alpha=10, binarize=0.75)
bnv.fit(X_train, y_train)
pred = bnv.predict(X_test)
```

Figure 30: Bernoulli Naïve Bayes Python Code

- ***alpha:*** acting like ***var_smoothing*** but is set to 1.0 by default. A high alpha will lead to underfitting (high bias), while a lower value can lead to overfitting. Picking a mid-range value would make sense and can be confirmed by the previously mentioned hyperparameter optimisation technique.
- ***binarise*** is the threshold of sample features being mapped to Booleans and, by default, is 0.

An exception was observed when calculating the recall on the predicted test labels.

```
/usr/local/lib/python3.7/dist-
packages/sklearn/metrics/_classification.py:1318:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.

  _warn_prf(average, modifier, msg_start, len(result))
```

This exception occurred when the model did not predict a value for a particular class. In this case, the purchases_a label was not predicted for data even when the test data contained values in this class. One solution was to reconfigure the test/train split

normal	646
ui_a	121
books_a	116
purchases_a	26

books_a	0.34	0.66	0.45	59
normal	0.97	0.79	0.87	795
purchases_a	0.00	0.00	0.00	0
ui_a	0.45	1.00	0.62	55

Figure 31: Bernoulli Naïve Bayes Labels

On the left is the ground truth with the label's count, and on the right is the test data prediction of labels. It can be observed that purchases_a is 0.

3.4.4 Support Vector Machine (SVM)

SVM was implemented three times using different kernels to manipulate the dataset's dimensions. Each kernel invokes a different function for adding dimensions to a feature set in order for a hyperplane to be drawn, separating data points into classes.

Linear *SVM*

The linear kernel is the most basic form of a kernel and is typically used for text classification problems performing best on large datasets. The linear function should compute faster than the following two kernels due to the similar equation.

$$F(x, xj) = (x * xj)$$

Figure 32: Linear SVM Function (Awasthi, 2022)

x, xj are the data points the model is trying to classify.

F(x,xj) represents the decision boundary separating classes using the dot product of two values.

The sklearn implementation svm.SVC is based on LIBSVM (open-source library for SVMs) and is used for relatively small datasets, and supports multiclass classification according to the one-vs-one scheme.

```
linear= svm.SVC(kernel='linear', C=6)
linear.fit(X_train, y_train)
pred = linear.predict(X_test)
```

Figure 33: Linear SVM Python Code

C: can be considered as the degree of correct classification the algorithm has to meet, must be a positive value, and by default is set to 1.

Gaussian RBF SVM

RBF is a popular general-purpose kernel that favours data in a Gaussian distribution. Typically, this kernel is used when little is known about the dataset.

$$F(x, xj) = exp(-gamma * ||x - xj|^2)$$

Figure 34: Gaussian RBF SVM Function (Awasthi, 2022)

```
rbf= svm.SVC(kernel='rbf', gamma=0.001, C=4)
rbf.fit(X_train, y_train)
pred = rbf.predict(X_test)
```

Figure 35: Gaussian RBF SVM Python Code

gamma: The inverse of the standard deviation of the RBF kernel is a hyperparameter used for non-linear kernels. By default, the value is set to 1 and is preferred to be a low value.

Polynomial SVM

The polynomial kernel is a more generalised version of the linear kernel and is theorised to be less efficient and accurate than its counterparts but produces high results for normalised data.

$$F(x, xj) = (x * xj + 1)^d$$

Figure 36: Polynomial SVM Function (Awasthi, 2022)

The hyperparameter ***d*** denotes the degree.

```
poly = svm.SVC(kernel='poly', gamma=0.01, C=2, degree=3)
poly.fit(X_train, y_train)
pred = poly.predict(X_test)
```

Figure 37: Polynomial SVM Python Code

degree: the degree of the polynomial used to find the hyperplane to split data points into classes. Exclusively used for the poly kernel and is by default set to 3. If the degree were to be set to 1, it would act the same as the linear kernel.

3.4.5 k-Nearest Neighbor (KNN)

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

Figure 38: KNN Python Code

Like the Random Forest's implementation, the classifier did not require any changing of parameters to produce high performing results. All parameters are set to default as per scikit-learn's documentation.

3.4.6 Test and Training

Once each model was implemented, the next course of action was to establish an effective testing procedure allowing the models to be compared. The initial idea was to choose a train/test split percentage on the dataset, e.g., 40% of the data will be used for training whilst 60% will be used for testing. One issue with this approach is that the models would only be trained on a specific portion of the data leading to underfitting/overfitting, under-represented classes, and bias. For example, bookshop-books may be solely found in the testing data and not the training, leading to the model being poorly trained for identifying the books class.

A technique known as K-Folds was implemented to counteract the above issues. This technique allows the data to be broken down into smaller subsets where one is used for testing (validation) whilst the rest are used for training. The model's memory is wiped between iterations to ensure it is only trained on subsets not being tested. This is iteratively repeated until all subsets have been used for testing once. The following diagram depicts how the dataset was divided and used to be able to test a model's performance.

$$Avg\ (P) = \frac{1}{4}\sum_{i=1}^{4} P_i$$

Figure 39: K-Fold Split Diagram

P = Performance score of the K-fold, e.g., recall

k = How many folds = 4

The dataset has been broken down into four distinct subsets, equating to 25% of the dataset. One subset is used for testing, while the remaining four will be used for training. The score for a single confusion matrix calculation will be added to an array where the sum of the array will be divided by the total number of folds (***k***), resulting in one single average.

If I were to use the timestamp feature, K-fold validation would be unable to be used as the sequential order of data needs to be maintained, which is impossible when splitting up data in this fashion. Forward chaining was used instead for the previous implementation using the timestamp feature.

Although the entire dataset is now being trained and tested, an imbalanced dataset can still occur inside one of the subsets. This imbalanced subset will lead to poor performance for this particular subset and risk the model not being trained appropriately. Stratified KFold can be deployed instead of the original KFold as the stratified implementation equally distributes values into each subset to ensure the subsets are proportional.

3.4.7 Evaluation Calculation

```python
def saveScore(scoresDf, model_name, precision_score, recall_score, f1_score, t1):
  scoresDf = scoresDf.append({
    'model_name' : model_name,
    'precision' : precision_score,
    'recall' : recall_score,
    'f1_score' : f1_score,
    't1' : t1,
  }, ignore_index=True)
  return scoresDf
```

Figure 40: Evaluation Helper Function Python Code

For the models to be compared, I needed a way of storing performance scores for each model. Above is a function that takes confusion matrix calculations and the time to compute an algorithm and stores them into a DataFrame.

3.4.8 Skeleton Structure for ML Model

```python
from sklearn.ensemble import RandomForestClassifier

# score arrays
precision_score = []
recall_score = []
f1_score = []
t1 = []

for train_index , test_index in kf.split(X, y):
  X_train, X_test = X.iloc[train_index], X.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]

  # create classifier, fit data and predict on test data
  t1_start = process_time()
  rfc = RandomForestClassifier(n_estimators=100)
  rfc.fit(X_train, y_train)
  pred = rfc.predict(X_test)
  t1_stop = process_time()

  # metrics append to arrays
  precision_score.append(metrics.precision_score(pred , y_test, average='macro'))
  recall_score.append(metrics.recall_score(pred , y_test, average='macro'))
  f1_score.append(metrics.f1_score(pred , y_test, average='macro'))
  t1.append(t1_stop-t1_start)

scoresDf = saveScore(scoresDf, "RF", sum(precision_score) / k, sum(recall_score) / k, sum(f1_score) / k, sum(t1) / k)
```

Figure 41: Full Structure of Random Forest Python Code

The above figure is the final implementation of the random forest model. Each model is surrounded by the same skeleton code allowing for K-fold validation and evaluation scores to be stored. Evaluation scores are stored into an array and then divided by the number of K-folds to return an average over each fold. Scikit-learn functions can be used to calculate the individual score to ensure the score has been calculated correctly without needing additional helper functions. Code to calculate the time elapsed is wrapped around each classifier in order for the time to compute training and testing to be measured.

Chapter 4: Results and Evaluation

Section 4.1 explores how the models were compared and how the calculations were made. 4.2 includes results using graphical representations alongside an explanation for binary and multiclass classification. 4.3 includes an evaluation of the methodology used throughout the implementation.

4.1 Evaluation Methodology

The models are evaluated based on calculations derived from the confusion matrix. The confusion matrix is a technique for summarizing the performance of a classification model by comparing predicted and actual labels. Poorly performing models identify labels wrongfully, whereas a good model will be able to identify labels correctly whilst making minor mistakes.

The below confusion matrices will show how the values that make up the calculations in section 4.1.3 can be determined using information based on a resource from Analytics Vidhya (*Confusion Matrix for Multi-Class Classification - Analytics Vidhya*, 2021).

4.1.1 Binary Confusion Matrix

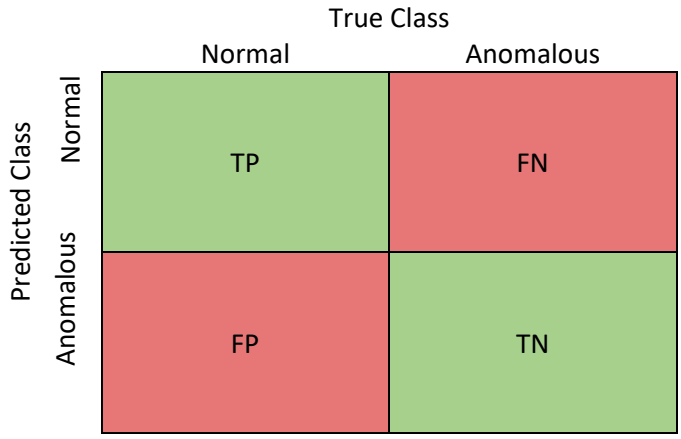

Figure 42: Confusion Matrix

- **True Positive (TP)**: Number of labels predicted correctly for normal
- **False Positive (FP)**: Number of labels that have been predicted as normal but are, in fact, anomalous
- **False Negatives (FN):** Number of labels that are normal but predicted as anomalous
- **True Negative (TN)**: Number of labels predicted correctly for anomalous

4.1.2 Multiclass Confusion Matrix

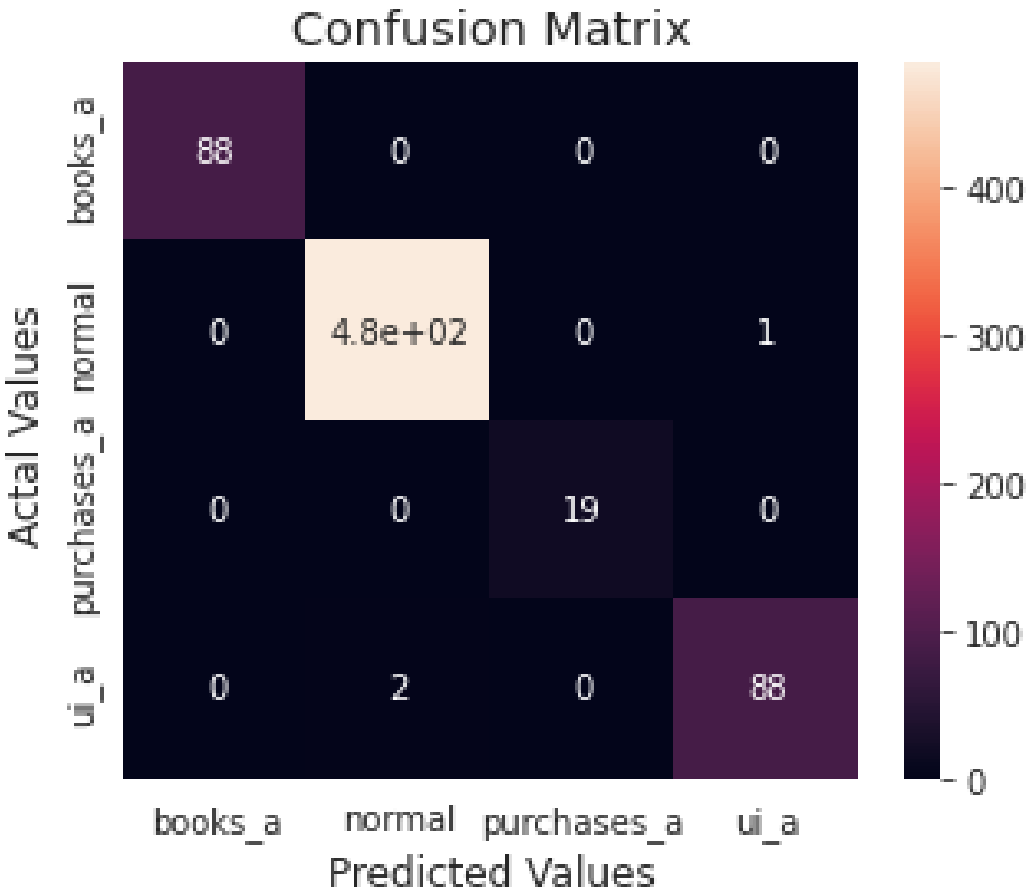

Figure 43: Multiclass Confusion Matrix

Multiclass classification confusion matrix must be able to compare a class against multiple other classes instead of just one. The below example uses books_a:

- **True Positive**: Number of labels predicted correctly, e.g., this would be just the top-left cell equalling 88.
- **False Positive**: Sum of values on the corresponding columns except for the TP value, e.g., adding the cells to the right of the TP value would be 0+0+0 = 0
- **True Negative:** Sum of values of all columns and rows except the class being calculated. e.g., 483+0+1+0+19+0+2+0+88 = 593
- **False Negative** Sum of values of the corresponding rows except for the TP value, e.g., adding rows below the TP value would be 0+0+0 = 0

Final Values: TP = 88, FP=0, TN=593, FN=0

4.1.3 Calculations

Name	Description	Significance	Equation
Accuracy	The ratio of correctly predicted observations to the total observations.	High accuracy is optimal, with a low accuracy a basic indicator of a low performing model.	$\frac{TP + TN}{TP + FP + FN + TN}$
Precision	Correctly predicted positive observations to the total predicted positive observations.	The lower the precision means more false positives, which could seem like an issue but should not cause significant problems.	$\frac{TP}{TP + FP}$
Recall (Sensitivity)	Correctly predicted positive observations to all observations in the actual class.	A low recall means that many false negatives have been observed. A low recall would mean there is a chance of an anomaly going undetected.	$\frac{TP}{TP + FN}$
F1-score	A weighted average of precision and recall.	It can be more valuable than accuracy when there are uneven class distributions.	$\frac{2(Recall * Precision)}{Recall + Precision}$
Time	Time taken to train and test a model on the given dataset.	A long compute time may be costly when working with large datasets.	$t_{stop} - t_{start}$

4.2 Results and Evaluation of Models

4.2.1 Binary Classification Results

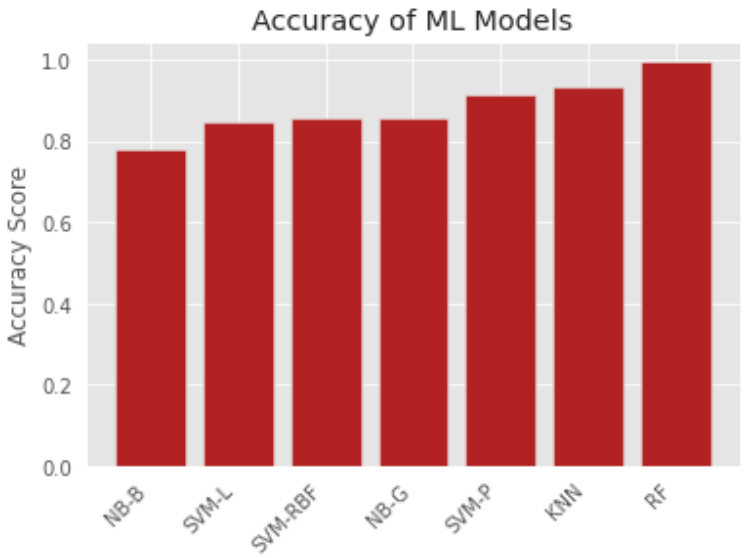

Figure 45: Binary Accuracy Bar Chart

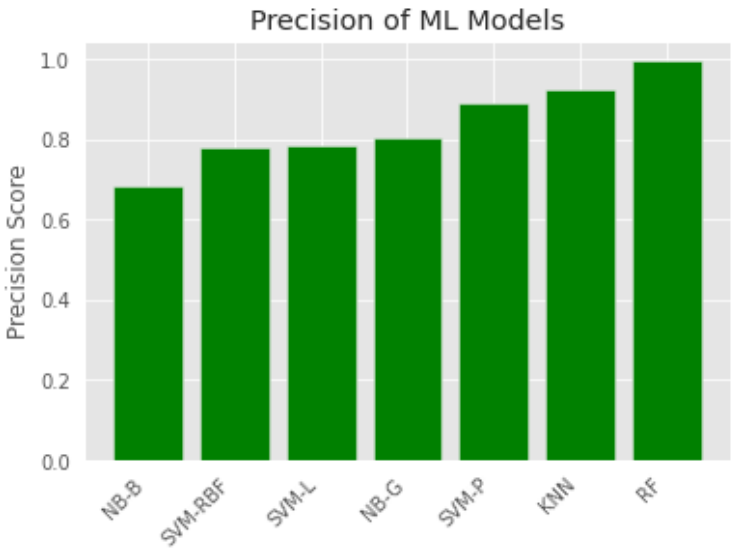

Figure 44: Binary Precision Bar Chart

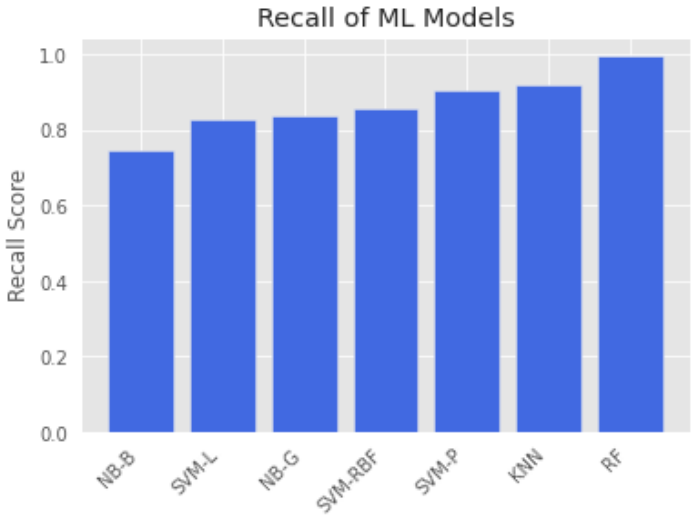

Figure 48: Binary Recall Bar Chart

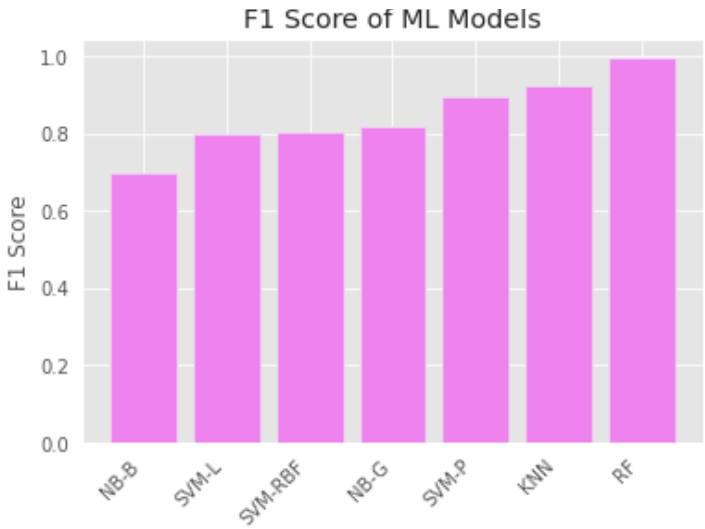

Figure 46: Binary F1 Score Bar Chart

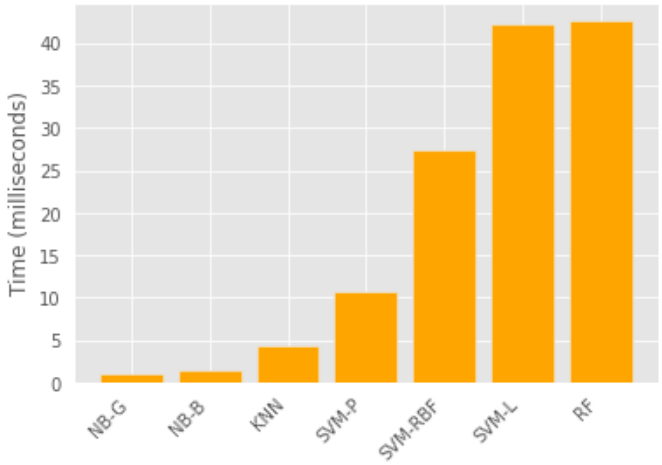

Figure 49: Binary Time Elapsed Bar Chart

Model	Accuracy	Precision	Recall	F1-score	Time (ms)
RF	0. 996699	0. 995420	0.996576	0.995976	42.6569
NB-G	0.857665	0.804457	0.837671	0.818097	0.9994
NB-B	0.780627	0.680461	0.743981	0.696705	1.4784
SVM-L	0.845929	0.785246	0.825540	0.800415	42.3209
SVM-RBF	0.856198	0.780405	0.857749	0.805469	27.3236
SVM-P	0.915997	0.890402	0.903573	0.896178	10.7272
KNN	0.935068	0.922279	0.920400	0.921200	4.3032

Figure 47: Binary Table of Results

4.2.2 Explanation of Binary Classification Results

Accuracy

RF (0.9966), KNN (0.9350) and SVM-P (0.9159) performed the highest, meaning the models were able to classify data correctly for over 90% of the data set. NB-G(0.8576), SVM-RBF(0.8561) and SVM-L(0.8459) performed similarly to one another. NB-B(0.7806) performed the worst, which becomes a reoccurrence for every calculation.

Precision

Precision follows a similar order with RF (0.9954), KNN (0.9222) and SVM-P (0.8904) performing the highest, followed by NG-G (0.8044), SVM-L(0.7852), SVM-RBF(0.7804) and NB-B(0.6804) being the lowest with a 10% decrease from SVM-RBF to NB-B.

Recall

RF (0.9965), KNN (0.9204) and SVM-P (0.9035) again were the top 3 generating values close to precision. SVM-RBF (0.8577), NB-G (0.8376) and SVM-L (0.8255) followed with tightly coupled results. NB-B performed 8% worse than SVM-L at 0.7439.

F1-score

RF (0.9959), KNN(0.9212), SVM-P(0.8961), NB-G (0.8180), SVM-RBF (0.8054), SVM-L(0.8004) and NB-B(0.6967). NB-B performed the lowest with over 10% worse performance and a significant gap between SVM-P and NB-G of nearly 8%. F1-score follows the Recall and Precision, which is as expected.

Overall

The top 3 performing models were RF, KNN and SVM-P, respectively. RF performed the best in the confusion matrix calculations whilst being the worst in time to compute at 42.6569 milliseconds. KNN has an impressive 4.3032 milliseconds, making this model the best for compute conscious scenarios such as the dataset increasing in observations. SVM-P was the best performing SVM implementation and the fastest at 10.7272 milliseconds.

SVM-L, SVM-RBF and NB-G performed similarly on every calculation. The main distinction between the models is the time elapsed. NB-G was the fastest model, with only 0.9994 milliseconds to compute. SVM-RBF takes 27 times longer at 27.3236, and SVM-L is 42 times longer at 42.6569 milliseconds. Making NB-G the fourth-best performing model with SVM-RBF fifth and SVM-L sixth. The close results of NB-G and SVM-RBF could be due to the models being tailored towards a Gaussian distribution which the dataset's features were modified to resemble through power transformations. Naïve Bayes would be the fastest due to the "naïve" aspect only considering each feature independently to determine labels.

NB-B performed the worst due to the nature of the dataset containing numerical over Boolean data. NB-B algorithm is created to handle Boolean values alongside the Naïve Bayes algorithm favouring categorical over numerical data. In theory, this algorithm would be the worst performer, translating into reality.

An isolation forest could have been used instead of a Random Forest as only two classes are present in binary classification. Only predicting two classes is actual anomaly detection, while multi-class anomaly detection is a classification problem. However, I do not think this would have generated better results apart from the time to compute may have been faster using the isolation implementation.

4.2.3 Multiclass Classification Results

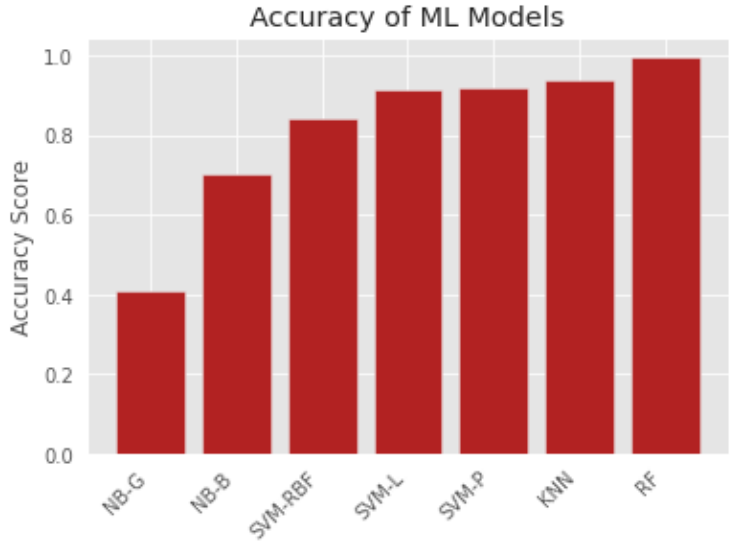

Figure 51: Multiclass Accuracy Bar Chart

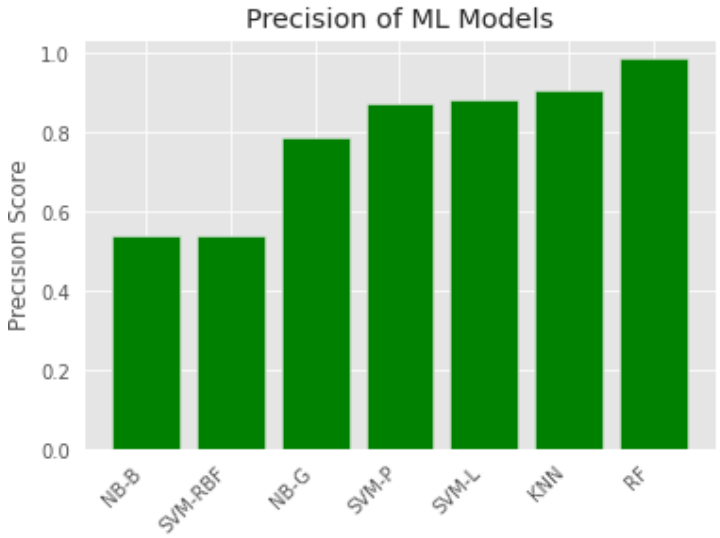

Figure 50: Multiclass Precision Bar Chart

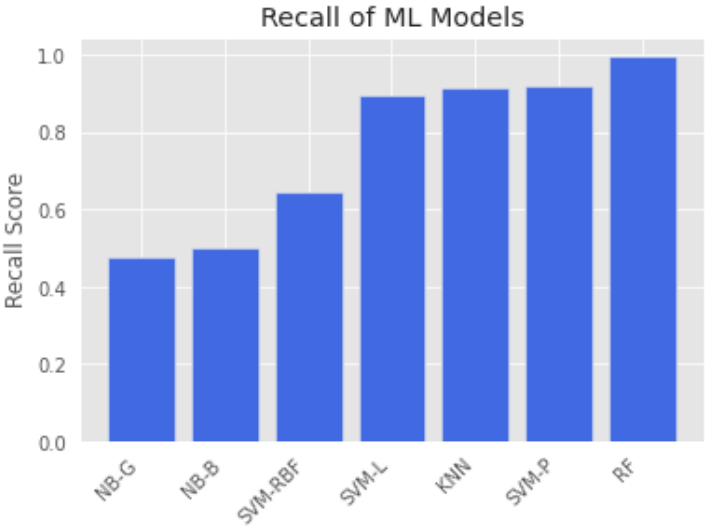

Figure 53: Multiclass Recall Bar Chart

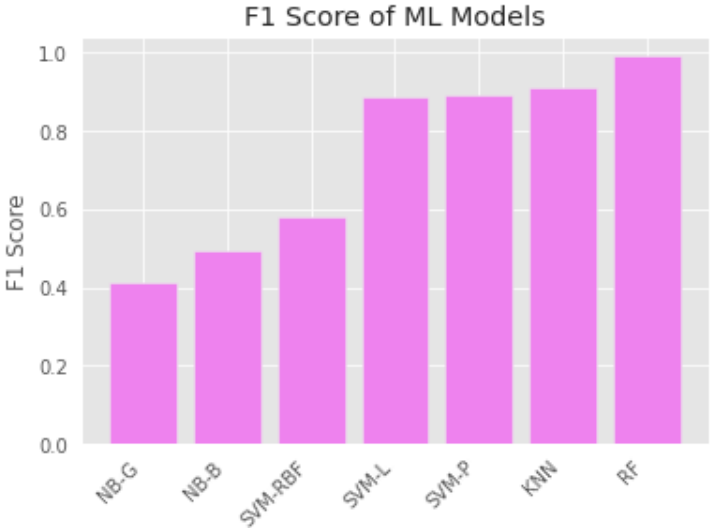

Figure 52: Multiclass F1 Score Bar Chart

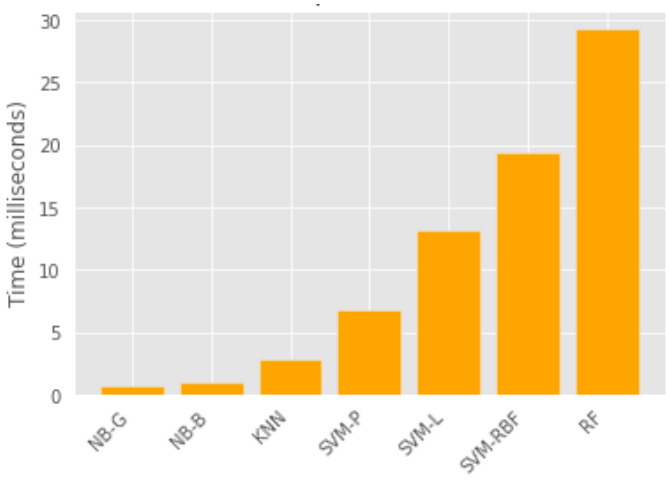

Figure 55: Multiclass Time Elapsed Bar Chart

Model	Accuracy	Precision	Recall	F1-score	Time (ms)
RF	0.996698	0.987646	0.997147	0.992140	29.3254
NB-G	0.408294	0.785704	0.477061	0.410340	0.6669
NB-B	0.703968	0.537947	0.500110	0.495278	1.0067
SVM-L	0.916353	0.881591	0.895286	0.886765	13.1285
SVM-RBF	0.842625	0.540189	0.645171	0.577365	19.4544
SVM-P	0.919665	0.871092	0.918990	0.891256	6.8037
KNN	0.936901	0.907992	0.916734	0.911336	2.8906

Figure 54: Multiclass Table of Results

4.2.4 Explanation of Multiclass Classification Results

Accuracy

RF (0.9966), KNN (0. 9369), SVM-P (0. 9196) and SVM-L (0.9163) all achieved over 90%. SVM-RBF (0. 8426), NB-B (0.7039) and NB-G (0.4082), the performance dramatically decreases, with NB-G performing very poorly.

Precision

RF (0.9876), KNN (0. 9079), SVM-L (0.8815) and SVM-P (0.8710) were the high performers. NB-G(0.7857) drops by 9% followed by SVM-RBF (0.5401) and NB-B (0.5379) with a 24% decrease.

Recall

RF (0. 9971), SVM-P (0.9189), KNN (0.9167) and SVM-L (0. 8952) dominate the top. SVM-RBF (0.6451), NB-B (0.5001), and NB-G (0. 4770) underperformed with a lower score than they achieved for precision. SVM-RBF/SVM-P has the opposite, with a considerably greater score for recall than precision. The recall is a more important metric as undetected anomalies are more dangerous compared to detecting a normal data point as an anomaly.

F1-score

RF (0.9921), KNN (0.9113), SVM-P (0.8912) and SVM-L (0.8867) again are the top performers with a graph which resembles the recall scores. SVM-RBF (0.5773) is next with a decrease of 30%, followed by NB-B (0.4952) and NB-G (0.4103).

Overall

RF, KNN and SVM-P were the top three models, with SVM-L closely behind. SVM-L performs much better in a multiclass classification problem which may be down to it being easier to draw linear decision boundaries when more classes are introduced. SVM-RBF was the fifth-best performing model but did not perform to the same standard as it was lower than the other two SVM implementations. NB-B and NB-G were the worst performers due to Naïve Bayes favouring datasets with categorical data. Interestingly, NB-G and SVM-RBF performed worse than the non-gaussian counterparts, suggesting that the dataset was not normalised enough. Revisiting the power transformation section and trying different methods to garner a Gaussian distribution would help this.

KNN works best on lower-dimensional data, whereas in this paper, the dataset uses a modest five features. Contractionary RF performed better, which tends to perform better on larger high dimensional datasets. Five features may be a low enough number of features for both RF and KNN to perform to a good standard, as RF should still be able to perform well in low feature spaces. Also, the dataset's dimensions were decreased by one due to the timestamp column being dropped, which may explain why the timeseries approach performed worst for KNN, as shown in section 4.2.5.

Multiclass has a much more varied performance range, with the lowest value being around 0.4, whilst binary classification's lowest was 0.65. Although the range in data is greater, the top three models were still the best and held relatively similar results to the binary approach. Interestingly the runtime either stayed the same or decreased for the multiclass approach in some instances, such as RF going from 42 to 29 milliseconds for binary classification. This observation could be due to Google Colab's resource allocation at the time, e.g., having access to a faster GPU or more volume of users as they were running on two separate workbooks.

A model can perform well in binary classification due to chance as it is only guessing between two possible labels. When more classes are introduced, it becomes a question of how well the model is, which is why Naïve Bayes performs worse in the multiclass problem.

4.2.5 Timeseries Approach

The first approach used the timestamp column as a feature to train and predict anomalies. The reason for this feature being dropped was due to worse performance. K-Fold validation was switched to forward chaining validation, a similar technique that can ensure sequential order. RF accuracy when using the timestamp column is 0.62, which is a significant decrease from 0.99 when this feature is dropped.

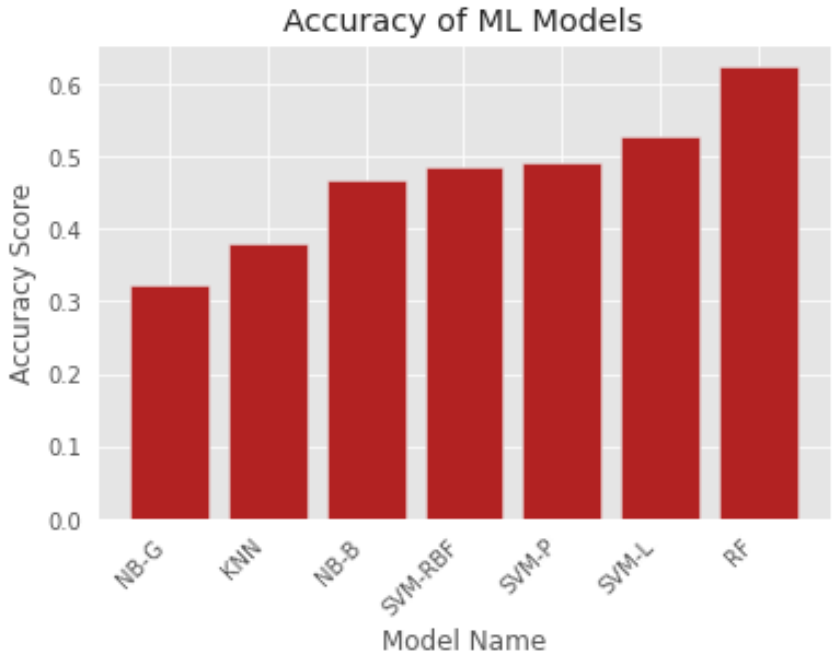

Figure 56: Timeseries Accuracy Bar Chart

One reason is that the models used may not favour a time-series approach. After further research, neural networks such as Recurrent Neural Networks and Long Short-Term Memory can be used (Xie et al., 2021), which may be preferred for a dataset containing the timestamp column.

4.3 Evaluation of Implementation

4.3.1 Development Tools and Technologies

Google Colabs played a role in limiting the time spent on optimising hyperparameters. GPUs and TPUs are prioritised for users who use Colab interactively rather than long-running computations (*Google Colab*, 2022). A substantial search for hyperparameters could not be achieved as Colabs would timeout when reaching usage limits. Without this, the hyperparameter values used cannot be confirmed to be the best and lead to unexplainable results. A solution would be to pay for Colab Pro or run the computationally expensive operations on a high-performing local machine, which I did not have for this project.

4.3.2 Methodology

The methodology followed in this paper was to classify and label data manually prior to implementing the ML models. In order to adequately label data correctly, in-depth knowledge of how the microservices were monitored and how/when the fault injection occurred would be needed. In this paper, I could not establish exactly when fault injection occurred, leading to the potential wrongful labelling of data. Anomaly detection research advocates against this practice unless an expert with extensive knowledge of the dataset is present. As the data was previously collected, the finer details were unable to be established, e.g., exact time of injection, how the injection occurred, projected effects of the injection, etc. Furthermore, as the dataset is spanned over a month, it would be unreasonable to reproduce a dataset of this length in the timeframe of this project.

If time was not a factor, the experiments could be reproduced by launching the microservices into the Cloud with the help of M3 to monitor metrics. Two approaches can then be followed:

1. Normal dataset (no fault injection): Using an Autoencoder, a semi-supervised model can be used to detect anomalies in the research dataset. Once labelled with the autoencoder's prediction, the newly labelled data can be used instead of manual labelling.
2. Normal and anomalous dataset: Inject faults into the system with the times of the event noted to allow for a more informed labelling approach.

Having a dataset which may not translate to the real world could cause delays in the project further down the line and invalidate some of the conclusions generated from this paper. With this being said, an ML model's primary goal is to take the training set and create predictions on new test data which closely matches the ground truth. In this paper, I believe this requirement was met as the extent to which the prediction matches the ground truth is measured.

Another issue with the dataset is the number of observations in each class. Normal as expected had the most (1939), with anomalous UI (361) and anomalous books (350) having a similar value. However, anomalous purchases (76) were much lower, leading to an imbalanced dataset. The imbalanced dataset issue was only amplified using K-Fold validation, further splitting the data.

This may have contributed to poor models for algorithms which rely on a balanced dataset, e.g., SVM assigns weights to counteract this through the ***gamma*** hyperparameter whilst Naïve Bayes uses the distribution from the training phase to predict test values. In the implementation section, Naïve Bayes found it challenging to detect any observations belonging to the "purchases_a" class leading to an exception.

Multinomial Naïve Bayes was admitted from the study due to negative values produced by the power transformations. The data could have been transformed to be positive again, allowing this model to

be used. The power transformations used may not have transformed the data into normal form, meaning there is a need for further testing using different techniques to establish the distribution of data.

4.3.3 Satisfaction of Functional/Non-Functional Requirements

Determining that some of the requirements have been met can be arbitrary, whilst other requirements are more complex. Below are the requirements created in the introduction section, alongside an explanation of if each requirement has been met.

Functional Requirements

1. **Detect labelled anomalies for both Binary and multiclass classification using Machine learning.**
   Both classification techniques have been implemented successfully, proven by the results.

2. **Diagnose the microservice in which an anomaly occurred.**
   Each class referred to a specific microservice, allowing the ability to discover which service the anomaly has occurred in. It could have been extended to specific components, e.g., CPU.

3. **Produce graphical representations of the performance of a model**
   Graphs can be found in the results section.

4. **Create the product to be deployable into any environment**
   The project could be transformed into an API using Flask or Django, which can be deployed into a microservice environment. However, this project could have taken further steps to address this requirement to be fully met.

Non-Functional Requirements

5. **Obtain an accuracy, precision, recall and F1 score above 90% for at least one model**
   Random Forest and KNN, in most cases, achieved over 90%.

6. **Ensure scalability with the number of observations in both training and prediction phases**
   Scalability is hard to measure as I only had access to one dataset. Further datasets must be tested to fulfil this requirement, allowing further results to be compared.

7. **The model's time to train and predict must be suitable for the number of observations**
   Time taken was under half a second for most implementations, with some taking less than a millisecond, which I would class as suitable.

8. **Follow the implementation of a proven anomaly detection algorithm correctly**
   Scikit-learn made it possible for highly accurate implementation of algorithms to be implemented on my given dataset.

Overall, I believe the requirements have been achieved with room to improve, as discussed in the following conclusion section.

Chapter 5: Conclusion

5.1 Achieving the Aim and Objectives

Prior to the project's start date, an aim and objectives were identified within the research phase. To determine if the project's main aim has been met, I first must establish if all objectives have been completed. The project has five objectives listed below, alongside an explanation of how they have been completed.

1. **Research and identify three existing machine learning models for detecting anomalies through research papers**

In the background research section Naïve Bayes, Random Forest, Single Vector Machine and KNN were identified, explained, and then further implemented in the project. Each model had a proven track record for effectiveness in this use case which is evident in the final results section. Naïve Bayes and SVM were further explored by introducing multiple distributions/kernels, aiding in conclusions about the type of ML models the dataset performs best on.

Through the use of scikit-learn, the implementation of the model was a relatively trivial task but having the underlying knowledge of what a hyperparameter would change, and the inner mechanisms of the algorithm helped to theorise which models would perform best on my dataset. I believe this objective has been met with one criticism: more research could have come from academic resources instead of web pages.

2. **Explore the current limitations and difficulties of detecting anomalies in a microservice architecture**

Throughout the background section, the microservice architecture was researched and explored to give this paper purpose and aid in the implementation/explanation of the results sections. Research conducted includes the history of microservices, issues microservices face, case studies, and a comparison of monitoring solutions. The research was gathered from multiple sources such as academic papers, official websites, and blogs for a cross-analysis to ensure the information put forward is valid. I believe that with this due diligence, this objective was achieved.

3. **Utilise online resources to implement machine learning techniques on my given datasets**

Similar to the previous two objectives, research has been gathered from multiple sources such as papers, scikit-learn documentation, and online web pages to effectively implement ML techniques. Further techniques explored include K-fold validation, power transformations, and one-hot encoding. Some of the concepts were taken from previous modules, with further learning to enhance my project and personal knowledge. I believe this objective has been met and has furthered my knowledge.

4. **Train and test the three machine learning models to improve the performance**

A training and testing strategy involving Stratified K-Fold validation limits the effects of an imbalanced dataset. It ensures that all data points can be used for training and testing, which is essential with a small dataset. Hyperparameter optimisation helped obtain appropriate values to be chosen to fulfil this objective. The optimisation could have been taken further to run for longer computations to pick better values to tune each model. As Google Colab's only allows short computations, only a small range of values could be picked, leading to some models not being fully optimised. I also implemented a further ML model and multiple versions of Naïve Bayes and SVM to surpass this portion of the objective.

5. **Evaluate and compare the different machine learning techniques to produce a conclusion and consolidate my findings**

All models implemented were compared against each other using calculations based on the confusion matrix. The comparison was made for binary and multiclass classification alongside a primitive implementation using the timestamp feature. Time elapsed for each model was calculated, allowing an in-depth analysis of which model would be best for specific scenarios in the evaluation section.

5.2 Future work

I believe the project has fulfilled many requirements put forward in this paper with the potential to build and improve on its shortcomings. With further support, the project can be used in the real world to solve modern complexities found in the Cloud today. Recommendations for further work to be carried out include:

1. **Train and test the models on a larger scale dataset**
   The dataset in this paper has a relatively low number of observations (2727) for a data science project, prompting the need for a larger dataset either with more data points per minute or over a more extended time period.

2. **Optimise hyperparameters**
   Due to Google Colabs not allowing long computations, the hyperparameters were not fully optimised. Once the models have been trained using a larger dataset, in-depth optimisation can be applied to further tune the algorithm to the dataset produced by the microservices.

3. **Deploy the model into a real-time tool**
   Once the test results are completed to a good standard, the model can be deployed into a microservice environment for real-time monitoring.

4. **Alert developers when an anomaly is detected**
   When the model identifies an anomaly, the system could alert a developer and offer insights such as the microservice the anomaly occurred in or the component responsible. This functionality would streamline the identification process, thus decreasing the time spent resolving an issue. Additionally, a smoother user experience will be a by-product as anomalies will be detected before serious harm has been done to the user.

5. **Explore further models for more complex datacentres**
   Aside from the traditional models used in this paper, approaches such as deep learning have been used to solve highly complex problems, such as DeepMind's Alpha Go (*AlphaGo*, 2022). If the complexity of the microservices increases, a more adaptive method may be needed to create a product which performs well in a real-world setting.

5.3 Final Thoughts

Reflecting on the final project and paper, many technical and personal challenges have been presented and overcome. Prior knowledge of microservices and data science helped identify the project's scope with an appropriate starting point. Learning about the history of microservices and why this software engineering architecture is preferred in the modern-day, alongside limitations, has helped to appreciate software architecture further. A missing element of this paper was setting up the microservice ecosystem myself, which would have added a lot to the data I would have collected, e.g., more samples and more diverse datasets with expert knowledge. Overall, I believe the aim of this paper has been met with many lessons learnt in many ways, which I will take to the next project I conduct.

References

AlphaGo (2022) [Online]. 2022. Available at: https://www.deepmind.com/research/highlighted-research/alphago. (Accessed: 17 May 2022).

Amazon CloudWatch - Application and Infrastructure Monitoring (2022) [Online]. 2022. Available at: https://aws.amazon.com/cloudwatch/. (Accessed: 17 April 2022).

Awasthi, S. (2022) *Seven Most Popular SVM Kernels.* [Online]. 2022. Dataaspirant. Available at: https://dataaspirant.com/svm-kernels/#t-1608054630730. (Accessed: 8 May 2022).

Baskin, I., Marcou, G., Horvath, D., and Varnek, A. (2017) *Random Subspaces and Random Forest. Tutorials in Chemoinformatics*, pp. 263-269. Available at: (Accessed: 19 May 2022).

Bernoulli Naive Bayes (2022) [Online]. 2022. Available at*: https://iq.opengenus.org/bernoulli-naive-bayes/*. (Accessed: 5 May 2022).

Butow, T. (2021*) Chaos Engineering: the history, principles, and practice.* [Online]. 2021. Gremlin.com. Available at: https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/. (Accessed: 11 April 2022).

Chatterjee, S. (2019) *Use Naive Bayes Algorithm for Categorical and Numerical data classification.* [Online]. 2019. Medium. Available at: https://medium.com/analytics-vidhya/use-naive-bayes-algorithm-for-categorical-and-numerical-data-classification-935d90ab273f. (Accessed: 29 April 2022).

Choosing What Features to Use (2022) [Online]. 2022. Available at: https://www.coursera.org/lecture/machine-learning/choosing-what-features-to-use-LSpXm. (Accessed: 3 May 2022).

Cloud Computing History and Evolution | Unboundsecurity (2021) [Online]. 2021. Available at: https://www.unboundsecurity.com/blog/evolution-and-future-of-cloud/#:~:text=While%20the%20concept%20technically%20launched,Amazon%20Web%20Services%20(AWS). (Accessed: 13 April 2022).

Confusion Matrix for Multi-Class Classification - Analytics Vidhya (2021) [Online]. 2021. Available at: https://www.analyticsvidhya.com/blog/2021/06/confusion-matrix-for-multi-class-classification/#:~:text=Confusion%20Matrix%20is%20used%20to,number%20of%20classes%20or%20outputs. (Accessed: 12 May 2022).

Containerization (2022) [Online]. 2022. Available at: https://www.ibm.com/uk-en/cloud/learn/containerization. (Accessed: 15 April 2022).

Documentation scikit-learn: machine learning in Python — scikit-learn 0.21.3 documentation (2022) [Online]. 2022. Available at: https://scikit-learn.org/0.21/documentation.html. (Accessed: 7 May 2022).

Garraghan, P., Yang, R., Wen, Z., Romanovsky, A., Xu, J., Buyya, R., and Ranjan, R. (2018) Emergent Failures: Rethinking Cloud Reliability at Scale. IEEE Cloud Computing, 5(5), pp. 12-21. Available at: (Accessed: 15 February 2022).

Gillis, A. (2022) *What is Datadog? Definition from SearchITOperations.* [Online]. 2022. SearchITOperations. Available at: https://www.techtarget.com/searchitoperations/definition/Datadog#:~:text=Datadog%20is%20a%20monitoring%20and,as%20servers%2C%20databases%20and%20tools. (Accessed: 17 April 2022).

Gluck, A. (2020*) Introducing Domain-Oriented Microservice Architecture.* [Online]. 2020. Uber Engineering Blog. Available at: https://eng.uber.com/microservice-architecture/. (Accessed: 11 April 2022).

Google Colab (2022) [Online]. 2022. Available at: https://research.google.com/colaboratory/faq.html#usage-limits. (Accessed: 18 May 2022).

Gulati, S. (2022) *How Docker uses cgroups to set resource limits?.* [Online]. 2022. Shekhar Gulati. Available at: https://shekhargulati.com/2019/01/03/how-docker-uses-cgroups-to-set-resource-limits/. (Accessed: 18 May 2022).

Hailey, C. and Sensu, I. (2021) *Sensu | An Introduction to Prometheus Monitoring (2021).* [Online]. 2021. Sensu.io. Available at: https://sensu.io/blog/introduction-to-prometheus-monitoring. (Accessed: 19 April 2022).

Horovits, D. (2020) *Monitoring Microservices the Right Way.* [Online]. 2020. InfoQ.com. Available at: https://www.infoq.com/articles/microservice-monitoring-right-way/. (Accessed: 24 April 2022).

Johnson, J. (2020) *Anomaly Detection with Machine Learning: An Introduction.* [Online]. 2020. BMC Blogs. Available at: https://www.bmc.com/blogs/machine-learning-anomaly-detection/. (Accessed: 17 April 2022).

Labs, A. (2016) *Microservices at Netflix: Stop system problems before they start.* [Online]. 2016. Medium. Available at: https://articles.microservices.com/microservices-at-netflix-stop-system-problems-before-they-start-e07c51a1d52d. (Accessed: 14 April 2022).

Metrics (2022) [Online]. 2022. Available at: https://docs.datadoghq.com/metrics/. (Accessed: 21 April 2022).

Microsoft Azure (2022) [Online]. 2022. Available at: https://docs.datadoghq.com/integrations/azure/?tab=azurecliv20. (Accessed: 17 April 2022).

Multiclass Classification Using Support Vector Machines (2021) [Online]. 2021. Available at: https://www.baeldung.com/cs/svm-multiclass-classification. (Accessed: 31 April 2022).

Multiclass Classification- Explained in Machine Learning (2020) [Online]. 2020. Available at: https://www.mygreatlearning.com/blog/multiclass-classification-explained/. (Accessed: 30 April 2022).

Multi-cloud vs. hybrid cloud: What's the difference? (2022) [Online]. 2022. Available at: https://www.cloudflare.com/en-gb/learning/cloud/multicloud-vs-hybrid-cloud/#:~:text=A%20hybrid%20cloud%20infrastructure%20blends,combining%20different%20types%20of%20apples. (Accessed: 5 May 2022).

Noor, A., Jha, D., Mitra, K., Jayaraman, P., Souza, A., Ranjan, R., and Dustdar, S. (2019) A Framework for Monitoring Microservice-Oriented Cloud Applications in Heterogeneous Virtualization *Environments. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD),* [Online]. Available at: https://ieeexplore.ieee.org/document/8814569. (Accessed: 17 April 2022).

Normal Distribution (solutions, examples, formulas, videos) (2020) [Online]. 2020. Available at: https://www.onlinemathlearning.com/normal-distribution.html. (Accessed: 31 April 2022).

Python vs. R: What's the Difference? (2021) [Online]. 2021. Available at: https://www.ibm.com/cloud/blog/python-vs-r. (Accessed: 4 May 2022).

Richardson, C. (2021) Who is using microservices?. [Online]. 2021. Microservices.io. Available at: https://microservices.io/articles/whoisusingmicroservices.html. (Accessed: 11 April 2022).

Rogers, K. (2018) *Black Box vs. White Box Monitoring: What You Need To Know.* [Online]. 2018. DevOps.com. Available at: https://devops.com/black-box-vs-white-box-monitoring-what-you-need-to-know/. (Accessed: 17 April 2022).

Rosen, C. and Rosen, C. (2017) *Monitoring IBM Cloud Container Service with Datadog - Archive of the IBM Cloud Blog.* [Online]. 2017. Archive of the IBM Cloud Blog. Available at: https://www.ibm.com/blogs/cloud-archive/2017/07/monitoring-ibm-bluemix-container-service-datadog/#:~:text=The%20Datadog%20Agent%20supports%20Docker,the%20IBM%20Bluemix%20Container%20Service. (Accessed: 29 April 2022).

Rossman, P. (2019) *How to Decipher False Positives (and Negatives) with Bayes' Theorem.* [Online]. 2019. The Stats Ninja. Available at: https://thestatsninja.com/2019/03/03/how-to-decipher-false-positives-and-negatives-with-bayes-theorem/. (Accessed: 31 April 2022).

Saxena, S. (2020) *The Gaussian RBF Kernel in Non Linear SVM.* [Online]. 2020. Medium. Available at: https://medium.com/@suvigya2001/the-gaussian-rbf-kernel-in-non-linear-svm-2fb1c822aae0. (Accessed: 24 April 2022).

Sreenivasa, S. (2020) *Radial Basis Function (RBF) Kernel: The Go-To Kernel.* [Online]. 2020. Medium. Available at: https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a. (Accessed: 28 April 2022).

Staff, C., Staff, C., Parayil, A., Johnson, B., Lappin, K., Lappin, K., Lappin, K., Naparstek, D., Shastri, V., Burke, M., Papazov, Y., Srivatsa, S., and Staff, C. (2021*) Cloud Services Comparison: AWS CloudWatch and Azure Monitor.* [Online]. 2021. CloudHealth by VMware Suite. Available at: https://blogs.vmware.com/cloudhealth/cloud-services-aws-cloudwatch-azure-monitor/. (Accessed: 26 April 2022).

The Incredible Rise of Datadog: An Entrepreneurial Success Story (2022) [Online]. 2022. Available at: https://growfers.com/story/datadog. (Accessed: 19 April 2022).

The Story of Netflix and Microservices (2020) [Online]. 2020. Available at: https://www.geeksforgeeks.org/the-story-of-netflix-and-microservices/. (Accessed: 14 April 2022).

Transformation, I. (2022) *Box Cox Transformation: Definition, Examples*. [Online]. 2022. Statistics How To. Available at: https://www.statisticshowto.com/box-cox-transformation/. (Accessed: 4 May 2022).

What Are Microservices? And How Do They Work? (2022) [Online]. 2022. Available at: https://www.netapp.com/knowledge-center/what-are-microservices/#:~:text=Microservices%20are%20an%20architectural%20approach,more%20than%2020%20years%20ago. (Accessed: 14 April 2022).

What is Microservices Architecture | Sumerge (2020) [Online]. 2020. Available at: https://www.sumerge.com/what-is-microservices-architecture/. (Accessed: 13 April 2022).

What is NumPy? — NumPy v1.22 Manual (2022) [Online]. 2022. Available at: https://numpy.org/doc/stable/user/whatisnumpy.html#:~:text=There%20are%20several%20important%20differences,array%20and%20delete%20the%20original. (Accessed: 19 May 2022).

Xie, L., Pi, D., Zhang, X., Chen, J., Luo, Y., and Yu, W. (2021) *Graph neural network approach for anomaly detection. Measurement*, 180, p. 109546.