SAPIENZA
UNIVERSITÀ DI ROMA

Building Real-Time Multivariate Anomaly Detection Systems in Industry's 5G Networks: A Comprehensive Workflow

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Applied Computer Science and Artificial Intelligence

Candidate

Alessio Borgi

ID number 1952442

Thesis Advisor

Prof. Iacopo Masi

Co-Advisor

(HPE) Dr. Paolo Ceccherini

Academic Year 2022/2023

Building Real-Time Multivariate Anomaly Detection Systems in Industry's 5G Networks: A Comprehensive Workflow

Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: alessioborgi3@gmail.com

Thanks

Dedico questa tesi a chiunque mi abbia supportato durante questo viaggio.

Inizio dalla mia Famiglia, e in particolare dalle due colonne portanti della mia vita: Mamma e Papà. Avete sempre creduto in me e se sono arrivato fino a qua, è soprattutto grazie alla vostra forza e il vostro esempio. Mi avete cresciuto insegnandomi ogni piccolo dettaglio, direzionandomi in ogni piccolo passo, e grazie a voi il percorso è stato, e sempre sarà, più facile, perchè mi spianate la strada da sempre. Continuo con mia Sorella, la persona su cui potrò sempre contare, che da sempre è la mia piccola colonna portante, sin da quando giocavamo insieme da piccoli. Spero che il nostro sogno che condividiamo, ossia aprire una azienda medico-tecnologica che curi ogni tipo di malattia in pochi secondi, si avveri, perchè con te abbiamo fatto sempre cose impensabili insieme. Arrivo poi alla mia Fidanzata, Samantha, che mi hai sempre supportato e sopportato, si può dire, da quando eravamo piccoli. Ti sei tuffata con me nel mare aperto, fidandoti incondizionatamente, e capendomi sempre. Ringrazio mia Nonna, mio Nonno e mia Zia, che, fin da piccolo, mi avete cresciuto dimostrando un amore senza limiti.

I would like to also thank my professor, Iacopo Masi, and all the other great professors I had the privilege to be a student of, that has transferred to me the love they put into their work, making me, in turn, completely fascinated by the Computer Science and Artificial Intelligence Field.

In the very end, I would like to thank all the colleagues I had the privilege to work with during this unforgettable experience at HPE.

We will meet again in the Master's Degree! Thank you all for letting my life be better!

Contents

List of Figures

List of Tables

Abstract

The evolution of 5G networks brings forth the promise of innovative business services for enterprises and consumers, including applications in autonomous vehicles, virtual and augmented reality, telemedicine, and remote surgery. However, the complexity and expectations of operations in this rapidly evolving field have dramatically increased. The data generated by network elements are high-dimensionality multivariate time series with the added complexity of weak correlation, making Anomaly Detection challenging. Additionally, the data generated by customers is often unlabeled, requiring an additional step for labeling before employing supervised machine learning models for anomaly prediction. Building a large number of supervised models for each network function, is currently impracticable, given the exponential increase in hardware/resources consumption and time spent.

In this thesis, we address these challenges by proposing a comprehensive workflow that provides a robust and suitable analytical method for dealing with weakly correlated, high-dimensional, and unlabeled multivariate time series data in 5G networks. The workflow encompasses key steps, starting from dataset gathering, preprocessing techniques, clustering algorithms, labeling phases, and the core of the workflow, real-time model building, where various machine learning and deep learning models, as well as ensemble methods, are explored. Model selection is performed using a decision workflow that considers a combination of metrics, time performance, and resource consumption, evaluating them both a Training and at Evaluation Time, allowing developers to fine-tune models based on customer needs. Additionally, as the last step of the Workflow, we conclude with a comprehensive Model's Feature Explanation, where we choose to employ two state-of-the-art Explainability Techniques that have not been previously applied to this specific type of workflow, rendering this approach novel and groundbreaking: (Shapley and LIME) to ensure transparency and interpretability, enabling an understanding of the factors contributing to anomaly detection decisions, providing respectively a Global and a Local understanding.

The proposed workflow, developed and validated (as shown in the last chapter) during a six-month internship at Hewlett Packard Enterprise (HPE) within the Path Finder Research Group, demonstrates its effectiveness in accurately detecting anomalies in 5G networks while considering computational efficiency and resource consumption. Notably, the workflow achieves a significant reduction in the number of models required, in the resources, and in the time needed, proving that a single model per cluster is sufficient to obtain informative results. This reduction in model complexity offers practical advantages for implementation.

The internship culminated in noteworthy achievements, including the Honourable Mention in the CTG's Hackathon as a personal accomplishment and the Best-Paper Award as a collective recognition of the team's efforts. These accolades highlight the significance and impact of the research conducted. By addressing the challenges posed by large-scale, high-dimensional, and weakly correlated multivariate time

series data, this research makes a valuable contribution to the field of anomaly detection in 5G networks in the Industry. The proposed workflow not only offers insights into effective network management but also provides practical solutions for monitoring and ensuring the reliable and efficient delivery of 5G services. Overall, this research project showcases the successful application of the developed workflow, validating its effectiveness and potential for real-world implementation.

Chapter 1

Introduction

The complexity of managing 4G and emerging 5G digital service provider networks, and associated services, has dramatically increased due to the virtualization and containerization of the network functions. A completely virtualized or cloud-native 4G/5G network challenges the cognitive abilities of earlier-generation human-based operations because of the volume, throughput, and dimensionality of management data generated. This is all the more true for the 4G/5G Edge or Radio Access Network (RAN). Automated systems needing minimal to no human intervention and capable of detecting similar, dissimilar, and anomalous behavior in real-time offer a powerful solution to address such needs. However, not only the inference or prediction processing time but also the amount of compute resources used must remain acceptable. Furthermore, network operations staff often need answers to very simple questions like comparing the behavior of the network now with the behavior used as a reference. Although the question is simple, the answer involves significant analysis and computation.

As a result, also current-generation Digital Service Provider (DSP) networks have also evolved, significantly increasing the complexity of operations. In addition, the size of current and emerging networks is another challenge. Starting from example, the DSP's RANs, also referred to as the 5G Edge, of great customers in the USA, have in excess of 300,000 RAN elements such as eNodeBs (groups of 5G cells) and gNodeBs (groups of 4G cells). The size augments more if we shift our attention to DSP RANs in the highest-populated countries like India and China, where the number of RANs elements levitate to around 450,000. From this example, we can infer that telecommunication operators must process vast amounts of data to have good visibility into the status and health of their networks.

The data generated by the network elements, or functions, are high dimensionality Multi-Variate Time Series with the added complexity of being weakly correlated. This essentially implies that a subset of dimensions, or even all the dimensions, cannot be dropped, which is often the practice when dimensions are highly correlated. In most situations, we have also the difficulty that the raw data generated by the network are unlabeled. Another ulterior step is therefore required in order to have data to be labeled before Supervised Machine Learning Models can be used to predict anomalous behaviors.

Another problem that arose is due to the fact that, as data generated by each of the Network Functions could exhibit vastly different behavior, one is in the unenviable position of having to label vast amounts of data, and then build a large number of supervised models. This is a big challenge if we view it from a Machine Learning operations life-cycle perspective, due to the exponential increase of Hardware/Resources Consumption and Time spent both at Design Time (a.k.a.

Training Time) and at Run Time (a.k.a. Evaluation Time).

All the problems explained, clearly identify the need to have a robust, deterministic, and suitable analytical method to deal with weakly correlated, high dimensionality, unlabeled multi-variate time series data, with the added complexity of spatial diversity and scale. This is the principal challenge through which I have been traveling during the unforgettable six-month period (from the 1st of March 2023 to the 31st of August 2023) passed at the renowned **Hewlett Packard Enterprise (HPE)** as **AI & ML Software Research Engineer** as Intern, in the **Path Finder Research Group**.

Figure 1.1. Hewlett-Packard's Enterprise

1.1 Hewlett Packard Enterprise (HPE)

The history of **Hewlett-Packard Enterprise** is intertwined with the history of **Silicon Valley**, a fertile stretch of land north of San Jose, California. [11] Today the area is known worldwide as a center for innovation, where companies have changed the world through technological advances.

But when Bill Hewlett and Dave Packard met in the 1930s at Stanford University, the valley was largely agricultural, with orchards and farms dotting the hills. Bill and Dave's success transformed the landscape and the culture of the area, so much so that the small garage on Addison Avenue in Palo Alto where the two first cemented their partnership has been designated a historical landmark, the **"Birthplace of Silicon Valley"**.

Figure 1.2. Hewlett and Packard's Garage

Bill and Dave were encouraged by their former professor Fred Terman to start a technology company of their own. One of the first prototypes they developed was a diathermy machine (a medical device that produced heat) that they sold to a Palo Alto clinic. Another device implemented helped astronomers at the nearby Lick Observatory to set a telescope accurately. They found their stride with a device to measure sound frequencies: the resistance-tuned audio oscillator, which spawned HP's first product line. Together in the garage, Bill and Dave produced the Model 200A audio oscillator, the first practical, low-cost method of generating high-quality frequencies.

Flush with their success, they formalized their partnership on January 1, 1939, deciding the company name on a coin toss. For the next year, Bill and Dave worked together until, with the addition of two employees, they finally outgrew the cramped garage and moved to new headquarters on Palo Alto's Page Mill Road in 1940.

1.1.1 Communications Technology Group (CTG)

The telecommunication industry is in the midst of an unprecedented technological transformation. 5G promises to revolutionize the telecommunication landscape, but this journey is still in its early stages. The shift from previous generation networks built on proprietary systems to open, cloud-native platforms utilizing commercial off-the-shelf infrastructure along with modular software components from multiple vendors is the biggest challenge facing the industry today.

For this transformation to succeed, telecommunication companies need to be able to call upon an ecosystem of trusted partners to reduce the risk of deploying open, multi-vendor 5G networks. This is why **HPE Communications Technology Group** has been established: helping telecommunication companies and enterprises to take advantage of the huge 5G market opportunity.

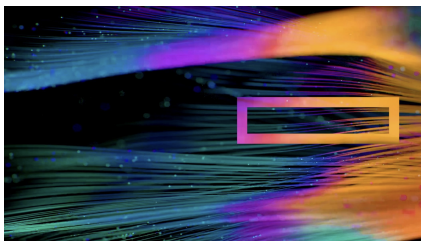


Figure 1.3. HPE's Communications Technology Group (CTG)

HPE continues a legacy of more than 30 years of experience designing, building, and tuning telco-grade infrastructure and software, by serving more than 300 telco customers across 160 countries and a monumental 850 million mobile devices worldwide are connected to HPE software. HPE's CTG collaborates with customers and partners to build open 5G solutions that deliver efficiency, reduce risk and complexity, and future-proof the network across the telco core, the Radio Access Network, and the telco edge.

1.1.2 Path Finder Research Group

The **Path Finder Research Team** is a CTG's Research Group whose focus is addressing challenges arising in operating large-scale communications / digital service provider networks.

The Path Finder Research Team is at the forefront of addressing the ever-evolving challenges faced by communications service providers (CSPs) and digital service providers (DSPs). As pioneers in the field, our primary focus is to enhance customer satisfaction, reduce operational costs, and drive competitive differentiation. To achieve these objectives, we recognize the paramount importance of optimizing the underlying network infrastructure for peak effectiveness and efficiency. In pursuit of our mission, we are spearheading the evolution of the service assurance process toward "zero-touch" operations. This transformative approach entails harnessing the potential of automation driven by the vast wealth of knowledge concealed within the extensive network data. The **Intelligence Assurance Suite**, plays a crucial role in elevating the range of services offered by the HPE Intelligent Assurance Suite platform to customers. The HPE Intelligent Assurance Suite is a transformative platform that unlocks the true potential hidden within vast volumes of network data. By applying advanced Machine Learning (ML) and Artificial Intelligence (AI) techniques, this suite empowers telecom service providers to extract invaluable insights from network data, including service and customer information. The platform seamlessly collects events, logs, and metrics, and through

Figure 1.4. Path Finder Research Group

the application of ML and AI techniques, identifies recurrent patterns and anomalies. As a pioneering research group, our ultimate goal is to harness cutting-edge Cloud and AI-based technologies to augment the capabilities of this platform, driving it toward unparalleled levels of automation and efficiency.

In the New-Services Development Chain, we, in fact, are the first link in the chain, as figure 1.5 explains.

Figure 1.5. The New-Services Development Chain. Path Finder is the very first part of the new services development chain. At the very end of the same chain, we have Product Management (and the Intelligence Assurance Suite). In the middle between these two parts, there is the Value-Pack Team, whose role is to deploy large-scale Path Finder's research results.

1.2 5G Connection

The Path Finder Research Group at HPE is actively engaged in exploring and leading advancements in the field of **5G Networks**, which stands as one of the primary domains where HPE holds a prominent market position. Over the last three decades, the evolution of telco networks has witnessed steady progress, moving from 2G to 3G and eventually to 4G/LTE, with each generation embracing the increasing dominance of IP as the foundational framework for all services. Throughout this transformation, communications service providers (CSPs) have been gradually decomposing their networks, distributing service functions closer to the edge, resulting in incremental improvements and functionality without substantial changes to the underlying software architecture.

5G, however, marks a significant departure from these evolutionary steps. With the advent of 5G, the landscape of proprietary telco networks is undergoing a paradigm shift, aiming to construct more scalable cloud-native architectures, similar to those employed by tech giants like Amazon, Google, and Microsoft in their hyper-scale data centers. The focus now lies in the development of service-based, microservices-oriented architectures that empower CSPs to swiftly create, deliver, and scale new services with heightened agility and automation.

This shift towards cloud-native architectures signifies a revolutionary step in 5G networks, enabling CSPs to embrace more scalable and flexible solutions to meet the ever-growing demands of their customers. By embracing cloud-native approaches, CSPs can harness the full potential of microservices-oriented architectures, resulting in faster and more efficient service delivery, enhanced agility, and seamless scalability to address evolving market needs. As a market leader, HPE's Path Finder Research Group recognizes the transformative potential of 5G networks and is committed to driving innovation and advancements in this domain to provide cutting-edge solutions to CSPs in their journey toward embracing the full potential of 5G technology.

1.2.1 5G: Characteristics

5G technology aims to achieve greater efficiency and versatility in supporting network applications through the following characteristics:

- **Slicing**: a.k.a. "Vertical Partitioning of the Network", is a network architecture that allows you to define on the same physical infrastructure a set of logical and/or virtual networks that are independent of each other capable of operating simultaneously, at full efficiency and without interference as if they each had a dedicated physical network. Each "slice" of the network is therefore in effect a complete network specifically cut to meet all the requirements of a particular application.
- **Network Devices Virtualization**: Virtualization of most of the network devices and dynamic management of the bandwidth available through automated systems such as SDN.
- **Device Management**: Ability to manage a greater number of devices per unit area (about 1,000,000 devices per km^2 against 1,000-100,000 per km^2 for 4G).
- **Response Time**: Support of higher characteristics in terms of latency to ensure real-time response times, necessary for critical applications.
- **Data Rate**: Higher data rates, theoretically up to 10 gigabits per second (GBit/s).
- **Energy Consumption**: Significant reduction in energy consumption (90% less than 4G for each bit transmitted).

With these characteristics, 5G networks, in addition to supporting mobile telephony, are expected to be mainly used as general internet service providers, competing with existing ISPs providing landline services, and will enable new applications in the Internet of Things (IoT) and in the machine-to-machine areas.

1.2.2 Network Cells

Like its predecessors, the 5G network is a digital-type cellular network, in which the area covered by the service is divided into small geographical areas called **Cells**. All 5G devices within a cell receive and transmit the signal via radio to the local antenna, which in turn is connected to the telephone network and the Internet via high-capacity optical fiber or via radio link through the backhaul network. As in all cellular networks, mobile devices moving from cell to cell are taken care of automatically and transparently from the new cell without losing the link. More, in particular, we have:

- **NodeB**: It is the radio base station in 3G UMTS networks;
- **eNodeB**: It is the radio base station in 4G LTE networks;
- **gNodeB**: It is the radio base station in 5G NR networks.

These radio base stations (nodes) are the cell towers mobile operators use to connect our mobile phones to 3G, 4G, and 5G networks, and are the ones in which I have worked during this Thesis.

1.3 Contribution

Comprehensive Workflow for Anomaly Detection in 5G Networks The core contribution of this internship is the proposal and development of a comprehensive workflow for anomaly detection in 5G networks. The HPE's previous solution, initially developed by the Path Finder Research Team and denoted as **HPE's Univariate Anomaly Detection Tool (UAD Tool)**, is designed to identify anomalies in large-scale datasets by employing Univariate Analysis techniques. The UAD Tool operates by creating a separate model for each individual cell within the dataset. Each model is responsible for detecting anomalies specific to its corresponding cell. This approach offers the advantage of capturing cell-level variations and detecting anomalies at a granular level. However, the main challenge associated with the UAD Tool is its **Scalability**. As the number of cells in the dataset increases, especially for large customers in countries like India and the USA, the tool's scalability becomes a significant concern. To cover a substantial dataset, the UAD Tool requires an enormous number of models, reaching up to 400,000 models for certain customers in India, China, and the USA. The proliferation of models poses several issues, including increased computational resources, storage requirements, and maintenance overhead. Managing such a massive number of models becomes impractical and inefficient, hindering the tool's overall performance and usability. Considering the limitations and scalability concerns of the UAD Tool, it became evident that a more efficient and scalable solution was necessary to handle large datasets without compromising accuracy and performance. Therefore, my main research focus has been on developing an alternative approach that addresses these challenges and provides a more effective anomaly detection solution in networking environments, both embracing multiple metrics (features) and trying to overcome the one-model per cell previous solution.

The Multivariate Anomaly Detection (MAD) Workflow proposed during my research encompasses crucial steps, starting from dataset gathering, which involves collecting data from various customers, and passing through preprocessing techniques for cleaning and standardizing the data. Subsequently, the data undergoes a clustering step, where similar network cells are grouped together, and a labeling phase, where the data is classified into either an anomaly or normal behavior using an ensemble of methods composed of seven algorithms. The centerpiece of the workflow lies in real-time (RT) model building, where various machine learning and deep learning models, along with ensemble methods, are explored and fine-tuned to achieve a good F1 Score. At the end of this step, the best algorithm for a particular customer and its specific needs is chosen using a novel Model Selection Workflow, by taking into account the best algorithms obtained during the previous fine-tuning step.

The Model Selection Workflow considers multiple criteria, such as Correctness (a combination of 10 different metrics in an Ensemble of Metrics), Time Performance (both Design Time and Run Time), and Resource Consumption (CPU, Memory, and Disk). This allows developers to fine-tune the different decisions based on the customer's unique requirements. By determining the best algorithm at Training Time (by analyzing Time and Resource Performances) and at Evaluation Time (by analyzing Correctness, Time, and Resource Performances), the overall best algorithm is chosen. Different decision choices and weights can be adjusted to make highly specific decisions, making the workflow adaptable to various scenarios and customers, reaching the Generalization that was required. This approach significantly reduces the number of models required, streamlining implementation and resource utilization.

To ensure transparency and interpretability of the AD system, we conclude the

Workflow with a comprehensive Model's Feature Explanation, where we choose to employ two state-of-the-art Explainability Techniques that have not been previously applied to this specific type of workflow, rendering this approach novel and groundbreaking. SHAP(SHapley Additive exPlanations) is employed for a comprehensive and collective model explanation (Global Explanation), performed over multiple sample instances, enabling a deeper understanding of feature contributions over, for example, a whole day. However, due to its computational complexity and resource requirements, but also in order to provide a more straightforward and instantaneous option for model explanation, we implemented a second explanation using LIME (Local Interpretable Model-agnostic Explanations), to provide insights over a single sample instance (Local Explanation).

The proposed workflow was rigorously validated using real customer data, demonstrating its effectiveness in accurately detecting anomalies in 5G networks while considering computational efficiency and resource consumption. The results showed that a single model per cluster is sufficient to obtain informative results, drastically reducing model complexity and offering practical advantages for real-world implementation.

Recognition and Awards The successful application of the developed workflow earned notable recognition during the internship. As a personal accomplishment, the project received an **Honourable Mention** in the **CTG's Hackathon**, highlighting the excellence and impact of the research conducted. Additionally, the team's collective efforts were rewarded with the **Best-Paper Award**, further validating the significance of the research contributions.

In conclusion, the internship's contributions have advanced the field of anomaly detection in 5G networks at HPE, offering a comprehensive and efficient approach to handling complex customer data while ensuring transparency and interpretability. The proposed workflow's practical advantages and validation results underscore its potential to drive impactful and transformative changes in the 5G network management domain. The workflow not only provides insights into effective network management but also offers practical solutions for monitoring and ensuring the reliable and efficient delivery of 5G services. The successful application and validation of the developed workflow showcase its effectiveness and potential for real-world implementation in the industry.

Chapter 2

Related Work

In this chapter we will delve into one of the most critical aspects of our research, focusing on identifying and characterizing anomalous patterns within complex datasets. **Anomaly Detection** plays a crucial role in various domains, including CyberSecurity(in which we aim to detect attacks on networks), Banking (in which we aim to detect Frauds), Finance(in which we aim to spot Anomalous Stock Prices), Healthcare(in which we aim to detect tumors in Medical Images), and Industrial Systems, where the early detection of abnormal behavior can lead to proactive decision-making and risk mitigation. In this chapter, we explore the fundamental concepts, techniques, and methodologies employed in AD, showcasing their applicability and effectiveness in real-world scenarios.

In today's data-driven world, the ability to identify anomalies is of paramount importance to ensure the integrity, security, and efficiency of complex systems. Anomalies, also known as outliers or deviations from normal behavior, can signify critical events, fraud attempts, system failures, or anomalies in sensor readings. By accurately detecting and understanding these anomalies, organizations can proactively respond, mitigate risks, and optimize their operations. Detecting anomalies poses unique challenges compared to traditional data analysis and classification tasks. Anomalies often exhibit different statistical properties, limited labeled training data, and evolving patterns over time. Moreover, the high dimensionality, complexity, and dynamic nature of modern datasets require sophisticated techniques capable of capturing subtle abnormalities amidst vast amounts of normal data. In this chapter, we address these challenges and explore techniques specifically designed to tackle anomaly detection problems.

2.1 Anomaly

Frank E. Grubbs, a prominent figure in the field of anomaly detection, made a significant contribution to the early understanding of **Anomalies**. In 1969, Grubbs' pioneering work laid the foundation for the identification and characterization of unusual observations that deviate significantly from the expected patterns within a dataset. His definition of outliers[1] marked a crucial milestone in AD research and has since influenced numerous studies in the field. An **Outlying Observation**, or "Outlier," is one that appears to deviate markedly from other members of the sample in which it occurs.

[1] In this thesis, we refer to **Outliers** and **Anomalies** interchangeably. We also rely on the following definition where: *"Outliers are also referred to as abnormalities, discordants, deviants, or anomalies in the data mining and statistics literature."* [1]

D. Hawkins [14] has made noteworthy contributions by providing a comprehensive definition of outliers. In his work, Hawkins presented a refined understanding of anomalies, considering factors such as data distribution, statistical significance, and deviation from expected patterns.

An **Outlier** is an observation that deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.

All these definitions highlight two main characteristics of anomalies:

- The distribution of the anomalies deviates remarkably from the general distribution of the data.
- The big majority of the dataset consists of normal data points. The anomalies form only a very small part of the dataset.

In the **context of this thesis**, we define anomalies and outliers as follows:

An **Anomaly** is an observation or a sequence of observations that deviate remarkably from the general distribution of data. The set of anomalies forms a very small part of the dataset.

This definition captures the essence of anomalies, highlighting their distinct nature and emphasizing their departure from the norm. By precisely defining anomalies, we establish a foundation for developing effective anomaly detection techniques that can identify and distinguish these rare and abnormal instances from the bulk of the data.

Anomalies VS Noise It is crucial to differentiate between **Anomalies** and **Noise** when dealing with data analysis. Noise can arise from various sources, such as mislabeled examples (class noise) or errors in the attributes of the data (attribute noise). Noise is considered unwanted and does not provide meaningful insights for data analysts. On the other hand, anomalies are of great interest to analysts as they represent observations that significantly deviate from the expected patterns or distributions in the data. Anomalies often hold valuable information and can reveal important insights or indicate potential issues within the dataset. For example, in the context of medical imaging, an anomaly could represent the presence of a tumor, while noise may simply manifest as random variations in brightness or color information, which are unrelated to the underlying phenomena.

In a nutshell, while noise is generally regarded as undesirable and does not contribute to the analytical process and therefore we aim to eliminate it, anomalies are the focal point of analysis and can provide valuable knowledge and actionable findings for further investigation and decision-making. In many applications, anomalies themselves are of interest and are the observations most desirous in the entire data set, which need to be identified and in some cases eliminated from the dataset, in such a way as to obtain a smoother model.

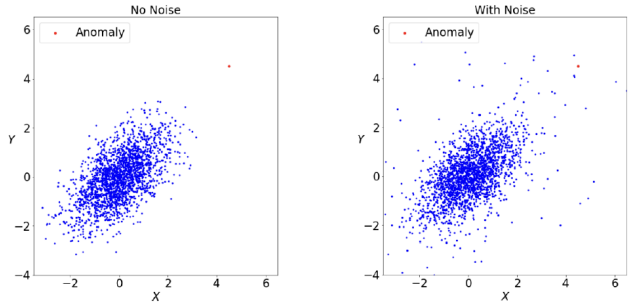


Figure 2.1. Noise VS Anomalies[4]: In both figures the main distribution is the same. In the first figure, the anomalous point marked in red seems to be obvious as it deviates significantly from the rest. But in the second figure, it is difficult to distinguish the anomaly point from the other points in the sparse space. This example shows that the difficulty to distinguish between anomalies and noises depends on the dataset. Thus, a deep understanding of the dataset is necessary to distinguish between anomalies and noises.

Anomalies vs Novelties It is also important to distinguish between **Anomalies** and **Novelties** in the context of data analysis. Novelties refer to patterns or data points that have not been previously observed in the dataset. These can include new or previously unknown phenomena or instances that have emerged over time. For example, in a network communication scenario, a novelty may arise from the introduction of a new communication protocol, resulting in a different pattern of interactions with a server. Unlike anomalies, novelties are considered normal or expected after they have been detected and understood. Once identified, they become part of the established patterns or distributions in the data. However, it is worth noting that many methods used for novelty detection are also applicable to AD and vice versa. As a result, in this thesis, we treat anomalies and novelties on an equal footing, recognizing that both types of observations require careful analysis and consideration.

2.1.1 Types of Anomalies

Anomalies can manifest in various forms, and understanding these different types is crucial for effective AD. In general, there are three commonly recognized types of anomalies:

- **Point Anomalies**: A point anomaly refers to an individual data point that significantly deviates from the rest of the dataset. For example, in a credit card transaction dataset, a transaction with an unusually large amount compared to other transactions would be considered a point anomaly. Mathematically, a point X_t is deemed anomalous if its value differs significantly from the values of neighboring points within a certain interval, such as $[X_{t-k}, X_{t+k}]$.
- **Collective Anomalies**: Collective anomalies, a.k.a. **Group Anomalies**, involve a sequence or a group of data points that collectively exhibit anomalous behavior. For instance, consider a bank customer who regularly withdraws $500 from their account. While an individual withdrawal of $500 may be normal, a sequence of such withdrawals within a short time span might indicate an anomalous pattern.
- **Contextual Anomalies**: Contextual anomalies occur when the normality or anomaly of a data point depends on the specific context or conditions under

which it is observed. A data point that appears normal in one context may be deemed anomalous in another context. For example, an average temperature of 35°C during the summer in Germany is considered normal, but the same temperature during winter would be regarded as an anomaly.

Understanding the specific type of anomalies present in a given dataset is crucial for selecting an appropriate AD method. Different detection approaches may excel at identifying specific types of anomalies while struggling with others.

In the scope of this thesis, we focus primarily on **Point Anomalies**, exploring techniques and algorithms specifically tailored for their detection and characterization.

2.2 Time Series

In this thesis, the data under analysis consists of **Time Series**, which are of fundamental importance in the context of AD. A formal distinct definition for time series is as follows: A Time Series is a sequence of observations taken by continuous measurements over time. Generally, the observations are picked up in equispaced time intervals (a.k.a. **Granularity**):

$$T = (t_0^d, t_1^d, \ldots, t_t^d), d \in N_+, t \in N \tag{2.1}$$

where d defines the dimension of time series.

In essence, a Time Series is a sequence of data points that are recorded and ordered based on their respective time stamps.

Univariate VS Multivariate In the context of Time Series Analysis, it is important to distinguish between **Univariate Time Series** and **Multivariate Time Series**, depending on the source of observations.

- **Univariate Time Series**: This refers to a sequence of observations derived from a single source, typically represented by a single sensor or variable. In this case, the dimension of the time series, denoted by d, is equal to $1 (d = 1)$. The observations are collected and recorded over time from a single data stream.
- **Multivariate Time Series**: This refers to a sequence of observations obtained from multiple sources or sensors, involving more than one variable. In this case, the dimension of the time series, denoted by d, is greater than 1 $(d > 1)$. The observations are collected simultaneously or in parallel from different data streams, capturing multiple aspects or dimensions of the underlying phenomenon.

For the purpose of this thesis, we specifically focus on **Discrete Time Series**, where the time index t takes on integer values, indicating observations recorded at discrete time points. Additionally, the time series data is perceived in equal time intervals, ensuring equal granularity between consecutive observations.

2.3 Anomaly Detection

After having a general definition for anomaly and time series, we will define what **Anomaly Detection** means and what kind of methods exist. In literature, different

terms are used that have the same or similar meaning to Anomaly Detection: **Event Detection**, **Novelty Detection**, **(Rare) Event Detection**, **Deviant Discovery**, **Change Point Detection**, **Fault Detection**, **Intrusion Detection** or **Misuse detection**.

The different terms reflect the same objective: to detect rare data points that deviate remarkably from the general distribution of the dataset. The amount of deviation is usually regarded as a measure of the strength of the anomaly or probabilistic looked as the likelihood of being an anomaly which is called: **Anomaly Score**. Thus formally, anomaly detection can be defined as a function $\phi : \mathbb{R}^N \to \mathbb{R}$:

$$\phi(x) \mapsto \gamma \tag{2.2}$$

where γ is the anomalous score and $x \in X \subseteq \mathbb{R}^{\mathbb{N}}$ and where X is the Dataset.

To convert the continuous value γ into a **Binary Label Normal vs Anomaly**, a threshold δ is defined where all points with an anomaly score greater than δ are marked as an anomaly. Thus, let $\phi_{score} := \phi$, then the binary labeling AD method ϕ_{binary} can be defined as:

$$\phi_{binary} : \mathbb{R}^N \to normal, anomaly \tag{2.3}$$

and

$$\phi_{binary}(x) \mapsto \begin{cases} anomaly, & \text{if } \phi_{score}(x) > \delta \\ normal, & \text{otherwise} \end{cases} \tag{2.4}$$

As stated in Anomaly Detection(section 2.3), anomalies form a very small part of the dataset. Often the anomalous part of a dataset is less than 1%. Therefore, usual binary classifiers would achieve above 99% accuracy if all data points would be labeled as normal (a.k.a. **Base Constant Classifier**), making AD more challenging. Nevertheless, to achieve satisfactory results in AD, it is crucial to select the appropriate method, which depends on the properties of the data under investigation. The following properties play a significant role in the selection process:

- Temporal vs Non-Temporal Data
- Univariate vs Multivariate Data
- Labeled or Unlabeled Data
- Types of Anomalies in the Dataset

In this thesis, our primary focus is to extend the capabilities of the previous version of the UAD Tool. We aim to broaden its scope by incorporating support for Multivariate Temporal Data, particularly Time Series which consists of both labeled normal and anomalous points.

Anomaly Detection on Univariate Time Series Anomaly Detection in Univariate Time Series involves fitting a forecasting model to the training data and then using the test data for making predictions. Typically, a sliding window approach is employed, where a sub-sequence of the dataset acts as the input to the model, enabling it to predict the subsequent timestamp. Formally, let w be the width of the sliding window, and suppose we want to predict x_i, and ψ be the learned forecasting model. To forecast x_i the following function and input data are used:

$$\hat{x}_i = \psi(x_{i-w}, \ldots, x_{i-1}) \tag{2.5}$$

The **Anomaly Score** can be computed by measuring the distance between the predicted value $\hat{x}_i$ and the real value x_i:

$$e_i = d(x_i, \hat{x}_i) \tag{2.6}$$

where d is a distance function. In Univariate Time Series, usually, the **Euclidean Distance** is used. The **Deviation** e_i, also called **Error Value**, is proportional to the anomaly score. If the anomaly score is above a threshold δ, it is marked as an anomaly.

Alternative approaches for detecting anomalies in univariate time series involve clustering or density-based methods. These methods operate under the assumption that contextual or collective anomalies exhibit distinct shapes that can be identified using clustering or density estimation techniques.

Anomaly Detection on Multivariate Time Series Detecting anomalies in Multivariate Time Series is a way more complex task compared to univariate time series. During this internship, I specifically focused on addressing the challenges associated with multivariate anomalies. As the dimensionality of the data increases, the need for a larger set of data points becomes more pronounced, leading to increased computational complexity. Hereinafter, Anomaly detection in multivariate time series is not as straightforward as in univariate time series. The higher number of random variables in multivariate time series introduces a greater likelihood of prediction errors in forecasting methods. While the Euclidean distance is commonly used in univariate time series to measure the deviation between dependent variables, this approach may not be directly applicable to multivariate time series. Instead, alternative similarity functions have been proposed, such as the **Mahalanobis Distance**[2], that is unit-less and scale-invariant, which is defined as:

$$D_{Mahalonobis} = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{2.7}$$

In the context of multivariate time series, anomalies can be detected based on deviations in the correlations among multiple univariate variables. The deviation of correlation patterns can serve as an indicator of anomalies within the data. By considering the inter-dependencies and relationships among different variables, we can develop techniques that effectively identify anomalies in multivariate time series data.

[2]The **Mahalanobis Distance** of a point x, is a multi-dimensional generalization of the idea of measuring how many standard deviations x is away from the mean of the Multivariate Gaussian $N(\mu, \Sigma)$. We can interpret this distance as the Euclidean Distance in a new coordinate system frame z, in which we rotate x by μ and we scale by Σ^{-1}. If the axis, a.k.a. **Principal Components**, is scaled to have unit variance, the Mahalanobis Distance is the Euclidean Distance.

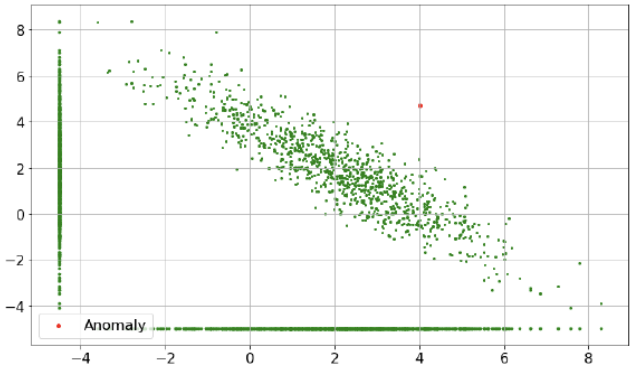


Figure 2.5: Anomaly in multivariate data

Figure 2.2. Anomaly in Multivariate Data[4]: In this figure is represented a bi-variate dataset. We assume that the data is the content of a sliding window with width w of a multivariate time series with $x_i \in \mathbb{R}^2, \forall i \in \{1, \ldots, w\}$. The red point is an anomaly we want to detect. If we consider each anomaly dimension as a Univariate time series, the projected points parallel to the x- and y-axis will be analyzed. The abnormal point in the bi-variate time series will be projected in the densest area on each of the univariate time series. Therefore, it will not be detected!

2.3.1 Supervised vs Unsupervised Anomaly Detection

If the time series dataset is Labeled, such that for each timestamp it is known if it is an anomaly or not and additionally the dataset contains normal and anomalous timestamps, then a **Supervised Anomaly Detection** method can be used. Supervised AD methods are able to detect an appropriate value for δ to classify all timestamps x_i as an anomaly if the corresponding anomaly score is $\phi(\hat{x}_i) > \delta$.

Unsupervised Anomaly Detection methods assume that the time series data is unlabeled. Most Unsupervised AD methods try to determine δ by analyzing the distribution of all $e_i, i \in \{1, \ldots, N\}$ and use the τ-percentile value as δ. One widespread approach is to set $\delta = 3\sigma$, where σ is the Standard Deviation of the distribution of $e_i, i \in \{1, \ldots, N\}$.

In this thesis, we will focus first on the Unsupervised Anomaly Detection Methods for finding and categorizing(labeling) Anomalies from the Customer Datasets, and, subsequently, on Supervised Anomaly Detection methods, with the use of the labeled datasets in our experiments to evaluate them.

2.3.2 Statistical vs Machine Learning vs Deep Learning Anomaly Detection Approaches

All AD methods on time series data can be divided into three main categories:

- **Statistical Methods**
- **Classical Machine Learning Methods**
- **Methods using Neural Networks (Deep Learning)**

The categorization of AD Methods in this thesis is driven by the objective of examining their distinct behaviors. It is worth noting that some studies combine the second and third classes, or the first and the second ones, considering them as a unified category of machine learning approaches. The boundary of the third

category, which encompasses methods utilizing neural networks, is relatively clear as it specifically involves the use of neural networks. However, it is important to consider the **Cost Aspect** within the context of HPE's objectives. Deep learning methods, while powerful, can be computationally expensive. Therefore, our focus for Deep Learning methods will be specifically on the application of Multilayer Perceptron (MLP) neural networks, which strike a balance between performance and computational efficiency. This choice aligns with the goal of minimizing costs for HPE's customers while still providing effective AD capabilities. Furthermore, it is important to note that our exploration of Deep Learning techniques was limited due to the framework constraints imposed by using PySpark. We were unable to utilize more advanced deep learning libraries such as PyTorch or TensorFlow. Nonetheless, we believe that the evaluation and analysis of MLP neural networks within the PySpark framework will provide valuable insights into their effectiveness for AD in the given context.

In Methodology & Theory(chapter 4), we will provide a comprehensive definition and explanation of every single algorithm and decision we have employed during the Internship, while in Experiments and Results(chapter 5), we will show every practical result obtained.

2.4 Class Imbalance

One of the biggest challenges we dealt with during my Research Intern was how to manage **Highly Imbalanced Datasets**. An Imbalance occurs when one or more classes (minority class, in our case the Anomaly class) have very low proportions in the data as compared to the other classes (majority class, in our case Normal-Behaviour class). When rare examples are infrequently present, they are most likely predicted as rare occurrences, undiscovered or ignored, or assumed as noise or outliers which resulted in more misclassifications of the positive class (minority) compared to the prevalent class.

Ironically, the smaller class (minority class) is often of more interest and more importance, therefore it called for a strong urgency to be recognized. For example, in a medical diagnosis of a rare disease where there is a critical need to identify such a rare medical condition among the normal populations. Any errors in diagnosis will bring stress and further complications to the patients. The physicians could not afford any incorrect diagnosis since this could severely affect the patient's well-being and even change the course of available treatments and medications. The same occurs in our field, where it has a pivotal role to recognize Anomalies that occur in the network, in such a way as to avoid further problems. Thus, it is crucial that a classification model should be able to achieve a higher identification rate on the rare occurrences (minority class) in datasets.

2.4.1 Challenges with Class Imbalance Classification

The **Nature of Class Imbalance Distribution** could occur in two situations:

- When the class imbalance is an **Intrinsic Problem**: This occurs when the data is not naturally imbalanced, but it is too expensive to acquire such data for minority class learning due to cost, confidentiality, and tremendous effort to find a well-represented data set, like a very rare occurrence of the failure of a space-shuttle.

- When class imbalance happens **Naturally**: This happens in the case of credit card fraud or in rare disease detection.

In our research application, we deal with a Class Imbalance that happens naturally since it is very difficult to gather more Anomaly Samples due to the fact that happens very rarely w.r.t. the Normal-Behaviour Samples. Class imbalance involves a number of difficulties and factors that can be explained as:

- **Imbalanced Class Distribution**: It can be defined by the ratio of the number of instances of the minority class to that of the majority class, as the following formula explains:

$$ClassImbalance = \frac{\#AnomalousSamples}{\#NonAnomalousSamples} \tag{2.8}$$

  In certain domain problems, the imbalance ratio could be as extreme as 1:10000, or even more. The study of [25] investigated the correlation between ratio imbalance in training set with the classification results using a decision tree classifier, and found out that a relatively balanced distribution between classes in datasets generally gives better results. However, as pointed out by [30], the degree of imbalance class distribution that will start to hinder the classification performance is still not explicitly known. An experiment from the study in [28] discovered that a balance distribution among classes is not a guarantee to improve a classifier performance since a 50:50 population ratio does not always the best distribution to learn from.

- **Lack of Information caused by Small Sample Size**: In addition to imbalance class distribution, another primary reason why class imbalance classification is challenging is because of the lack of data due to the small sample size in the training set. An inadequate number of examples will cause difficulties to discover regularities, that is, pattern uniformity, especially in the minority class.

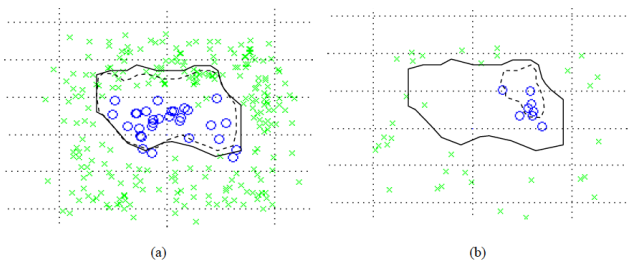


Figure 2.3. The Impact of the Dataset Size: In these two figures, it is represented the importance of having more data samples we can. The more data samples we have, the more the estimated decision boundary will be similar to the true decision boundary.[3]

A reported work found out that as training sample size increases, the error rate of the imbalanced class classification reduces[17]. This discovery is explainable since a classifier builds better representation for classes with more available training samples since more information could be learned from a variety of instances provided by a larger number of training sizes. It also reveals that a classifier should not be affected much by a high imbalance ratio providing that there is a large enough number of training data.

- **Class Overlapping or Class Complexity**: One of the leading problems in class imbalance classification is class overlapping occurrences in the datasets.

  Class Overlapping or sometimes referred to as **Class Complexity** or **Class Separability** corresponds to the degree of separability between classes within the data [22]. The difficulty to separate the minority class from the majority class is the major factor that complicates the learning of the smaller class. When overlapping patterns are present in each class for some feature space, or sometimes even in all feature space, it is very hard to determine discriminative rules to separate the classes. Previous work in [17] discovered that as the level of data complexity increases, the class imbalance factor begins to affect the generalization capability of a classifier. The work from [9] suggested that there is a relationship between overlap and imbalance in class imbalance classification however the level is not well-defined. Many investigations into class separability bring pieces of evidence that class overlapping problems bring severe hindrances to a classifier performance compared to imbalanced class distribution.

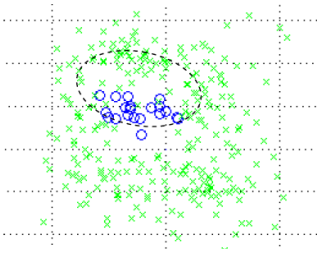


Figure 2.4. Overlapping Between Classes: In this figure, we can see the difficulties in the boundary definition due to the high overlapping present between the two classes.[3]

- **Small Disjuncts: Within Class Imbalance**: While in learning class imbalance classification, the imbalance ratio between minority class and majority class is obvious, sometimes an imbalance present within a single class might be overlooked. The **Within-Class Imbalance** or sometimes referred to as **Small Disjunct** appears when a class is comprised of several sub-clusters of different amounts of examples. Figure 2.5 below illustrates the concept of small disjuncts and overlapping classes in the class imbalance problem.

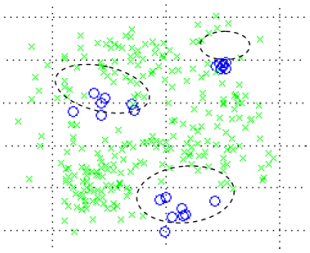


Figure 2.5. Within Class Imbalance: In this figure, we can see the segmentation we have within the biggest cluster. [3]

Chapter 3

Background

In the first part of this chapter, we focus on unraveling the intricacies of the PySpark architecture, which served as the backbone of our research efforts within the Path Finder Research Group. Throughout this chapter, we explore the various components and layers of the PySpark architecture, shedding light on their functionalities and interactions. We discuss the core concepts and mechanisms that underpin PySpark's distributed computing model, allowing for seamless parallel processing of data across clusters. Understanding the inner workings of PySpark is crucial for harnessing its full potential and leveraging its capabilities in the realm of big data analytics. Additionally, we highlight the advanced analytics capabilities that PySpark supports through its MLlib library.

Moving forward, the second part of this chapter delves into the starting point of our research journey, namely the previous UAD Tool.

3.1 PySpark

PySpark is a powerful open-source data processing framework that has played a pivotal role in my research endeavors. It provides seamless integration of Python and Apache Spark [16], a distributed computing system, enabling efficient and scalable data processing and analysis. At its core, PySpark[24] leverages the capabilities of Spark to handle large-scale data processing tasks. Spark's distributed computing model allows it to distribute data across a cluster of machines, enabling parallel processing and faster execution of complex data operations. This distributed nature of Spark makes it ideal for handling big data applications and performing computations on massive datasets.

In my research, PySpark[29] has been instrumental in handling and analyzing massive volumes of data efficiently. Its distributed computing capabilities have enabled me to process and transform complex datasets at scale, extract meaningful insights, and uncover patterns and trends that would have been challenging or time-consuming using traditional data processing methods. This has also been one of the meeting points between my research in the Path Finder Research Group and the Value Pack Development Team.

Apache Spark This is a powerful open-source big data processing framework originally developed by Matei Zaharia[10] as a part of his Ph.D. thesis while at UC Berkeley. The first version was released in 2012, and since then, in 2013, Zaharia co-founded and has become the CTO at Databricks. This framework has revolutionized the way large-scale data analytics and processing are performed. It provides a unified

computing engine and a comprehensive set of tools and libraries for distributed data processing and analysis. At the core of Apache Spark lies its ability to distribute data across a cluster of machines and perform parallel computations, resulting in significantly faster processing times compared to traditional data processing frameworks. Spark achieves this through its Resilient Distributed Dataset (RDD) abstraction, which allows data to be partitioned and processed in parallel across the cluster.

PySpark On the other hand, PySpark is the Python library and API for Apache Spark. It seamlessly integrates the capabilities of Spark with the simplicity and ease of use of Python, making it a popular choice among data scientists and researchers.

The combination of Apache Spark and PySpark empowers researchers to tackle complex big data challenges efficiently. Here are some key features and benefits:

- **Distributed Computing**: Apache Spark's distributed computing model enables processing large volumes of data by distributing it across a cluster of machines.
- **In-Memory Processing**: Spark's ability to perform computations in-memory significantly speeds up data processing tasks by reducing disk I/O.
- **Scalability**: Spark's scalability allows researchers to handle large and growing datasets effortlessly.

3.1.1 PySpark Architecture

Next, we explore the architecture of PySpark, which comprises various components that work together to execute data processing tasks efficiently. The key components include:

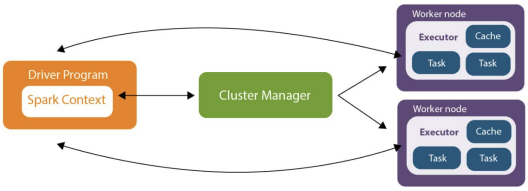


Figure 3.1. The PySpark Architecture.

- **Driver Program**: The driver program acts as the main entry point for PySpark applications. It coordinates the execution, manages resources, and communicates with the cluster manager to distribute tasks to the worker nodes. The PySpark application is initiated by the driver program, which runs the user's PySpark code. The Driver Program performs the following tasks:
  - Creates a SparkSession, which serves as the entry point for interacting with Spark and executing operations.
  - Divides the code into a series of tasks and schedules their execution.
  - Communicates with the cluster manager to request resources and allocate tasks to worker nodes.
  - Manages the execution flow, coordinates data processing, and collects results.

- **Cluster Manager**: The cluster manager, such as Hadoop YARN, is responsible for allocating resources and managing the cluster of machines where the Spark application runs. The Cluster Manager performs the following functions:
  - Allocates resources, such as CPU and memory, based on the application's requirements and availability.
  - Schedules tasks on worker nodes, taking into account resource availability and data locality for efficient processing.
  - Monitors the health of worker nodes and restarts failed tasks or reschedules them on healthy nodes if necessary.
- **Executors**: Executors are worker processes running on the cluster's worker nodes. They are separate JVM processes responsible for executing tasks assigned by the driver program, managing data partitions, and caching intermediate results in memory. Worker Nodes perform the following actions:
  - Receive task assignments from the driver program via the cluster manager.
  - Load data into memory, perform computations, and store intermediate results.
  - Communicate with other nodes to exchange data or perform shuffle operations.
  - Report task progress and results back to the driver program.

3.1.2 MLlib

The MLlib[23] library in PySpark is a powerful tool for performing machine learning tasks at scale. It provides a comprehensive set of algorithms and tools that enable efficient processing and analysis of large datasets for various machine-learning tasks. MLlib leverages the distributed computing capabilities of PySpark to perform machine learning tasks on large-scale datasets. This distributed approach enables parallel execution and significantly reduces the processing time for complex machine-learning tasks. MLlib exposes three core machine learning functionalities.

- **Data Preparation**: Feature extraction, transformation, selection, hashing of categorical features, and some natural language processing methods.
- **Machine Learning Algorithms**: Some popular and advanced regression, classification, and clustering algorithms are implemented.
- **Utilities**: Statistical methods such as descriptive statistics, chi-square testing, linear algebra (sparse and dense matrices and vectors), and model evaluation methods.

3.1.3 Scalability and Performance

One of the key advantages of PySpark is its ability to scale horizontally and efficiently process large-scale datasets. It leverages in-memory computation and a directed acyclic graph (DAG) execution engine to optimize data processing operations. By distributing data and computations across multiple nodes, PySpark can achieve high-performance processing and handle big data workloads effectively.

In summary, I chose to utilize PySpark during my Research work due to its integration with the Value-Pack Team's Scala-based Intelligent Assurance Suite, enabling seamless integration of new code and leveraging the advanced capabilities of Apache Spark for big data processing.

Chapter 4

Methodology & Theory

In this chapter, we will delve into the intricate details of the **Workflow** that we proposed during my Internship in the Path Finder Research Group, as well as provide a comprehensive explanation of the **Theoretical Background** that underpins our research.

4.1 Workflow: Overview

The **Multivariate Anomaly Detection Workflow** presented in this thesis is a pioneering and customer-centered approach designed specifically for AD in 5G networks. This comprehensive 7-step workflow revolves around the customer, making it the focal point of the entire process since each customer's 5G network data exhibits unique characteristics and requirements. To achieve optimal AD results and cater to individual customer needs, we prioritize the customer's data and preferences from the very beginning of the workflow until the final model selection.

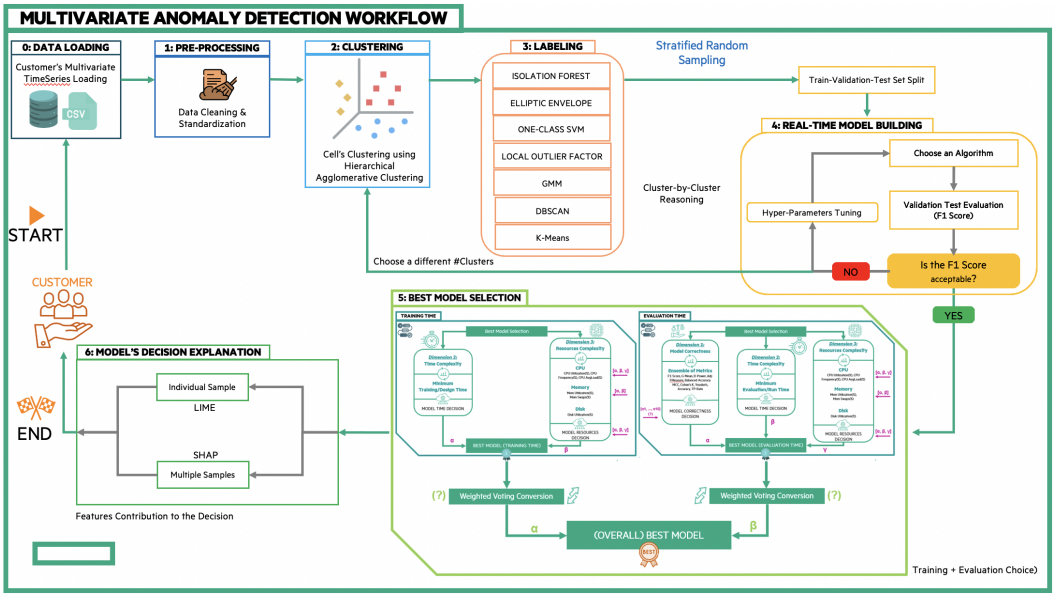


Figure 4.1. Multivariate Anomaly Detection Workflow: Detailed Version

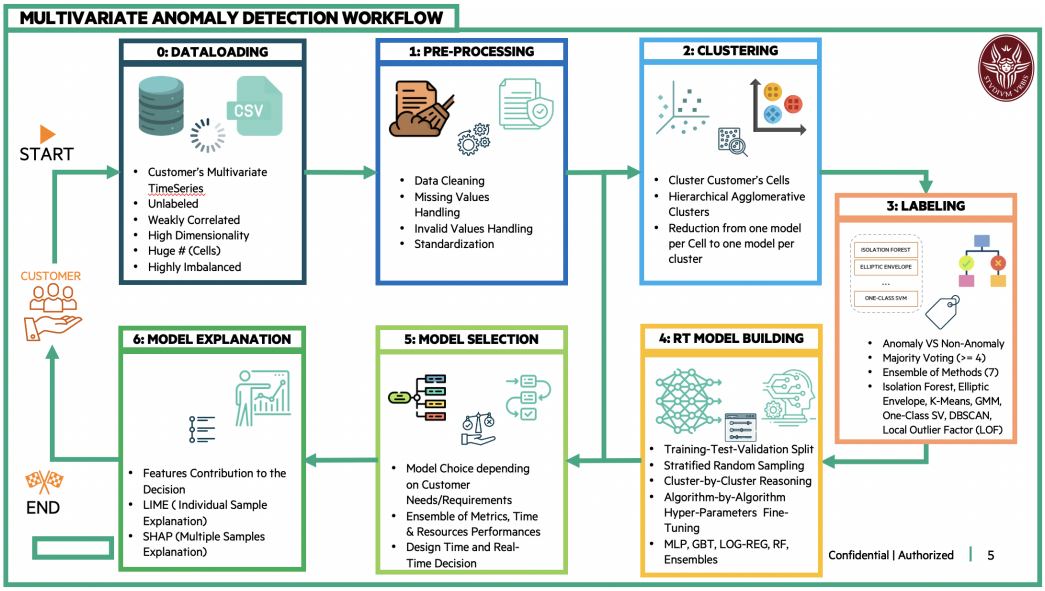


Figure 4.2. Multivariate Anomaly Detection Workflow: The workflow proposed encompasses crucial steps, starting from dataset loading, which involves collecting data from various customers, and passing through preprocessing techniques for cleaning and standardizing the data. Subsequently, the data undergoes a clustering step, where similar network cells are grouped together, and a labeling phase, where the data is classified into either an anomaly or normal behavior using an ensemble of methods composed of seven algorithms. The centerpiece of the workflow lies in real-time (RT) model building, where various machine learning and deep learning models, along with ensemble methods, are explored and fine-tuned to achieve a good F1 Score, with the possibility to go back to the Clustering Step to fine-tune the number of Clusters chosen. At the end of this step, the best algorithm for a particular customer and its specific needs is chosen using a novel Model Selection Workflow, by taking into account the best algorithms obtained during the previous fine-tuning step. The Model Selection Workflow considers multiple criteria, such as Correctness (a combination of 10 different metrics in an Ensemble of Metrics), Time Performance (both Design Time and Run Time), and Resource Consumption (CPU, Memory, and Disk), by allowing developers to fine-tune the different decisions based on the customer's unique requirements. To ensure transparency and interpretability of the AD system, as the last step of the workflow, two model explanation techniques, SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations), were incorporated into the workflow. This explanation step ensures that customers can understand and trust the AD system, fostering better model adoption and engagement. SHAP is employed for a comprehensive and collective model explanation, performed over multiple sample instances, enabling a deeper understanding of feature contributions. However, due to its computational complexity and resource requirements, we also introduced a more straightforward option for model explanation, using LIME, to provide insights over a single sample instance. This choice allows us to cater to specific scenarios where rapid explanations are necessary.

Throughout the entire workflow, the customer's data and needs remain at the forefront, guiding the choices made at each step. This customer-centered approach not only ensures the effectiveness and practicality of the AD process in 5G networks but also strengthens the relationship between developers, stakeholders, and end-users. By prioritizing the customer and their data, we demonstrate a commitment to providing tailored and impactful solutions that align with the real-world challenges of AD in 5G networks.

4.2 Workflow's Step 0: Data Loading

In the initial phase of the project, our primary objective was to efficiently **Load the Data**. To accomplish this, we implemented two main approaches, each offering distinct advantages and serving different purposes. These approaches were designed to streamline the development process and provide direct access to the cluster's database. Let's explore these approaches in more detail:

- **Data Loading from CSV Files:** This approach enables us to load the data from comma-separated value (CSV) files, a common and easily accessible format for tabular data.
- **Data Loading from Kairos DB:** Kairos DB, is a local database cluster, that is constantly receiving updates with new data and facilitates querying large datasets.

Once Loaded, we have a set of Multivariate Time Series Data, possessing the following characteristics:

- **Unlabeled Data**: The customer's dataset does not include explicit labels for AD. Therefore, an additional step is required to label the data before employing supervised machine learning models for anomaly prediction.
- **Weakly Correlated:** The data exhibits a weak correlation between different variables, making the AD task more challenging due to the lack of strong patterns in the data.
- **High Dimensionality:** The dataset comprises a high number of dimensions.
- **Huge Number of Cells:** The data represents a large number of cells in the 4G-5G network, leading to an extensive dataset that requires robust and scalable processing methods.
- **Highly Imbalanced:** The dataset is highly imbalanced, where normal behavior significantly outweighs anomaly instances.

Given these characteristics, it is crucial to employ a comprehensive preprocessing strategy that can handle the complexity of the data and prepare it for subsequent AD and Model-Building steps.

4.3 Workflow's Step 1: Pre-Processing

After the Workflow's Step 0: Data Loading (section 4.2), the very Workflow's first step is the **Pre-Processing Step**. Preprocessing is an essential component in data analysis and machine learning tasks as it lays the foundation for accurate and reliable results. This phase encompasses several crucial steps, starting from data cleaning to rectifying inconsistencies, errors, and anomalies. Additionally, we will explore the crucial process of standardization, which enables fair comparison and interpretation of data across different scales and units. Lastly, we will delve into handling missing values, a critical task that ensures no data points are overlooked or improperly imputed.

The primary objective of the Preprocessing Phase is to ensure the quality and integrity of the dataset. By carefully addressing various aspects of data preprocessing,

we can mitigate potential issues that may arise during analysis and modeling. This meticulous process plays a pivotal role in enhancing the overall data quality and prepares the dataset for subsequent steps in our Workflow. By diligently following these preprocessing steps, we can lay a solid foundation for our analysis, leading to accurate insights and reliable results.

4.3.1 Data Cleaning & Standardization

In our research, we recognized the importance of data quality and integrity and therefore focused on the crucial step of Data Cleaning. This step ensures that our dataset is free from corrupted cells that could potentially introduce errors or bias into our analysis. Corrupted cells refer to data entries that are either missing or contain invalid values. These issues can arise due to various factors, including data entry errors, measurement issues, or data transmission problems. It is essential to address these issues to prevent the dataset from being influenced by inaccurate or unreliable data.

To enhance data quality, we implemented a series of data-cleaning steps. Let's delve into the specific sub-steps involved in the Data Cleaning process:

Missing Values Management One challenge encountered in our dataset is the presence of missing measurements for certain time intervals. To handle this, we needed to develop strategies for managing and filling in these missing values. The presence of missing values in a time series dataset presents challenges in the analysis process. Interpolation and imputation techniques are commonly employed to estimate missing values based on neighboring data points or statistical methods. However, in our case, we made a deliberate decision to **exclude the time reference in our modeling approach** and leave it as a potential area for future work.

By disregarding the time reference, each data point was treated as an independent observation, irrespective of its sequential order. This decision was motivated by several factors. Firstly, the missing values were present across different time intervals, making it challenging to accurately infer the true temporal relationships. Secondly, our analysis focused on a global perspective, considering clusters of data rather than individual cells. This allowed us to generalize the information and develop a smaller number of models. Our objective was to capture the overall patterns and characteristics of the data, rather than explicitly modeling the temporal dependencies. This approach streamlined the analysis process and provided a more generalized framework for AD in multivariate time series data.

Additionally, we performed the step of **deleting rows** that contained missing values **(NaN values)** to ensure the integrity of the dataset, and avoid Data Distribution Shifting due to the over-filling that techniques like Forward Fill and Backward Fill could lead to.

Data Standardization Another important preprocessing task we performed on the dataset is Data Standardization. Standardization ensures that the dataset has a mean (μ) of zero and a standard deviation (σ) of one. This process involves transforming each data point (x) in the dataset according to the following formula:

$$\hat{x} = \frac{x - \mu}{\sigma}, \forall x \in D \tag{4.1}$$

Standardization **eliminates scale differences between features**, which is particularly useful when different features have different measurement units or scales,

ensuring that no single feature dominates the analysis simply due to its larger magnitude, and specifically suited in our case, given the scale differences present in the Datasets.

By performing these Data Cleaning steps, including missing values management and data standardization, we ensure the quality and integrity of our dataset, thereby laying a solid foundation for subsequent analysis and modeling steps.

4.4 Workflow's Step 2: Clustering

After the Workflow's Step 1: Pre-Processing(section 4.3), the **Clustering Phase** is the next step in our workflow. The primary objective of this phase is to cluster the total number of cells we have in our Dataset into general distinct groups, enabling better handling of the data by training separate models for each cluster. This phase lies at the core of our thesis goal, which is, again, to significantly reduce the number of models required for AD over Multivariate Time Series, without compromising overall performance.

Throughout this phase, we explored different clustering algorithms to identify the most suitable solution. Initially, we implemented the Naive K-Means algorithm; however, we later decided to adopt Agglomerative Clustering as our final choice (as explained by the experiments in Workflow's Step 2: Clustering(section 5.4).

4.4.1 Clustering Problem

One of the most common forms of unsupervised learning is **Clustering**[8], whose goal is to assign similar data points to the same cluster. Under a more formal perspective, we have that the **Clustering Problem** provides:

- **Input**: In input it receives a set of points $\{x_1, x_2, \ldots, x_n\} \in R^d$ and k number of clusters we want to identify.
- **Output**: What we expect in output is, instead, a set of points, a.k.a. **Centroids** or **Prototypes** $\{\mu_1, \mu_2, \ldots, \mu_n\} \in R^d$ each disjoint from the others, representing the cluster's centers, and a set of assignment labels $\{y_1, y_2, \ldots, y_n\}$ that creates a mapping from data points to centroids, and that must be aligned with the input points.

The Clustering Problem is NP-Hard [1]. In order to find a solution for this problem, some **Non-Optimal** solution has been developed.

4.4.2 Hierarchical Agglomerative Clustering

The Clustering Solution employed in this phase is the **Hierarchical Agglomerative Clustering** or **HAC**. Here, we have:

- **Input**: The input to the algorithm is an $N \times N$ dissimilarity matrix $D_{ij} \geq 0$.
- **Output**: The output is a tree structure in which groups i and j with small dissimilarities are grouped together in a hierarchical fashion.

[1] In computational complexity theory, **NP-Hardness** (non-deterministic polynomial-time hardness) is the defining property of a class of problems that are informally "at least as hard as the hardest problems in NP".

Agglomerative clustering starts with N groups, each initially containing one object, and then at each step it merges the two most similar groups until there is a single group, containing all the data. There are actually three **Agglomerative Clustering Variants**, depending on how we define the dissimilarity between groups of objects: Single Link, Complete Link, and Average Link.

Since the **Complete Link** is the more robust and optimal one in our case, we decided to operate with this Link Technique. In complete link clustering, also called **Furthest Neighbor Clustering**, the distance between two groups is defined as the distance between the two most distant pairs:

$$d_{CompleteLink}(G, H) = \max_{i \in G, i' \in H} d_{i,i'} \tag{4.2}$$

Single linkage only requires that a single pair of objects be close for the two groups to be considered close together, regardless of the similarity of the other members of the group. Thus clusters can be formed that violate the compactness property, which says that all the observations within a group should be similar to each other. In particular, if we define the diameter of a group as the largest dissimilarity of its members, $d_G = \max_{i \in G, i' \in H} d_{i,i'}$, then we can see that single linkage can produce clusters with large diameters. Complete linkage represents the opposite extreme: two groups are considered close only if all of the observations in their union are relatively similar. This will tend to produce clustering with small diameters, i.e., more compact clusters.

4.5 Workflow's Step 3: Labeling

After completing the Clustering Phase (discussed in Workflow's Step 2: Clustering(section 4.4)), the subsequent step in our Workflow is the **Labeling Phase**. This phase plays a crucial role in our overall methodology as it involves the labeling of data samples as either **Anomalous** or **Normal Behavior**. Unlike supervised machine learning scenarios where the labels are provided, the datasets we worked with do not come pre-categorized.

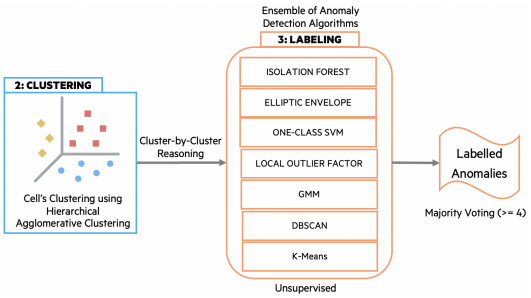


Figure 4.3. Multivariate Anomaly Detection Workflow: Labeling Step

In our workflow, we, therefore, approach the Labeling Phase as an **Unsupervised Machine Learning Task**. We leverage the concept of an **Ensemble of Methods**, which allows us to harness the collective intelligence of diverse algorithms, enabling a comprehensive analysis of the dataset. By incorporating a variety of methods, we can capture different perspectives and patterns present in the data, enhancing the overall labeling accuracy. In our specific case, where the precise labeling of anomalies is crucial, the ensemble approach proves highly beneficial. Our ensemble

consists of seven distinct algorithms (K-Means, DBSCAN, GMM, Isolation Forest, Local Outlier Factor, Robust Covariance, and One Class SVM), accurately described in the sections that follow, carefully selected based on their individual merits and performance in AD. Each algorithm contributes its insights and expertise to the labeling process, ensuring a comprehensive evaluation of the data. To arrive at the final labels, we employ a **Majority-Voting Decision** mechanism, taking into account the consensus among the ensemble members.

4.5.1 Ensemble of Methods

In the realm of machine learning, addressing highly imbalanced datasets(like our cases) poses a significant challenge. Traditional algorithms often struggle to achieve satisfactory performance when faced with severe class imbalances leading to high-variance and sub-optimal results. Recognizing the limitations of relying solely on a single algorithm in such scenarios, the exploration of **Ensemble Learning** techniques becomes imperative, both offering improved predictive power and robustness. Instead of relying on a single model, ensemble learning leverages the collective wisdom of multiple models (a.k.a. **Base Learners**), denoted with the λ symbol, combining their predictions to achieve superior performance. By aggregating the outputs of diverse models, ensemble methods can capture a broader range of patterns and overcome the inherent bias toward the majority class. The resulting model has the form:

$$f(y|\boldsymbol{x}) = \frac{1}{|\lambda|} \sum_{m \in \lambda} f_m(y|\boldsymbol{x}) \tag{4.3}$$

where f_m is the m-th base model. The ensemble will have a similar bias to the base models, but lower variance, generally resulting in improved overall performance. Once we have obtained the different predictions from the models, we need to somehow aggregate them into one final decision.

For this workflow's step, we decided to apply the **Majority Voting** Decision. This strategy, a.k.a. **Committee Voting** or even **Hard Voting**, provides that each base learner in the ensemble predicts the class label for a given instance, and the class label with the majority of votes is chosen as the final prediction. More formally, if we have:

- Let N be the number of base learners in the ensemble and K be the number of classes (in our case, K=2 always).
- Let $y_1, y_2, ..., y_N$ be the predicted class labels by the base learners, for a given instance.
- Let S be the final result embracing the different sums of votes for each of the $j_1, j_2, ..., j_K$ classes. Here, we have that each final value for each class j_i is obtained as:

$$S_{j_i} = \sum_{n=1}^{N} (y_n = j_i) \tag{4.4}$$

The final prediction, denoted as y_{final}, is obtained by selecting the class label:

$$y_{final} = \arg\max(S_{j_i}) \tag{4.5}$$

where $i = \{1, \ldots, N\}$ and $j = \{1, \ldots, K\}$. When dealing with **Classification Problems**, this decision is the most used. Suppose each base model is a binary

classifier with an accuracy of θ, and suppose class 1 is the correct class. Let $Y_m \in \{0, 1\}$ be the prediction for the m-th model, and let $S = \sum_{m=1}^{M} Y_m$ be the number of votes for class 1. We define the **Final Predictor** to be the **Majority Vote**, i.e., class 1 if $S > M = 2$ and class 0 otherwise. The probability that the ensemble will pick class 1 is:

$$p = P(S > M/2) = 1 - B(M/2, M, \theta) \tag{4.6}$$

where $B(x, M, \theta)$ is the *cdf* of the binomial distribution with parameters M and θ, evaluated at x.

K-Means The first algorithm to be applied in the Workflow Labeling Step, within the Ensemble, is the most widespread Clustering Algorithm: the **K-Means Clustering**. Its goal, as with all the clustering algorithms, is to partition the data into clusters. In particular, we decided to opt for the simples Naive version (Lloyd Algorithm), used together with the **K-Means++ Initialization**, a.k.a **Farthest Point Clustering**, where we have that higher distances should be more likely to be selected as initial data points.

The application of K-Means Clustering in our workflow follows a series of steps:

- **Clustering**: Using the K-Means algorithm from the scikit-learn library, the preprocessed and scaled dataset is clustered into a predefined number of clusters.
- **Cluster Assignment**: Each data sample is assigned to one of the clusters based on its proximity to the cluster centers.
- **Distance Calculation**: The Squared Euclidean distances between each data sample and the cluster centers are calculated.
- **Anomaly Labeling**: To label Anomalies, a new column is added to the dataset, initialized with a value of 0 for all data points. The labeling process is performed for each cluster individually. The sub-steps here are:
  - For each cluster, store the squared distances corresponding to that cluster.
  - A threshold value is determined based on the desired Contamination Percentage[2].
  - Using the threshold, the new column created at the very start is updated to assign a label of 1 to data points that exceed the threshold, indicating that they are anomalies within their respective clusters.

[2]The **Contamination Percentage** represents the proportion of anomalies that we expect to find in the dataset. In the context of the K-Means Clustering algorithm used for anomaly labeling, the contamination percentage is used to determine the threshold value for identifying anomalies within each cluster, calculated based on the percentile of the distance values within a particular cluster. By specifying the desired contamination percentage, we can determine the cutoff point above which data points are considered anomalies within their respective clusters. (For example, if we set a contamination percentage of 0.05 (5%), it means that we expect approximately 5% of the data points within each cluster to be labeled as anomalies. The threshold value is then determined as the distance value below which 95% of the data points lie.)

DBSCAN The second algorithm employed in our Workflow's Labeling Step is the **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**. While Partition-Based Clustering and Hierarchical Clustering Techniques are highly efficient for normal-shaped clusters, Density-Based Clustering Techniques, such as DBSCAN, excel in handling arbitrarily shaped clusters and detecting outliers.

Density-Based Clustering algorithms are particularly effective in identifying high-density regions and outliers, making them well-suited for AD tasks. The fundamental principle behind DBSCAN is that a point is considered to belong to a cluster if it is close to many other points in that cluster. Two key **parameters** govern the DBSCAN algorithm:

- ϵ**(eps)**: It defines the distance that specifies the neighborhood for determining cluster membership. In other words, two points p_1 and p_2 are considered to be neighbors if:
$$d(p_1, p_2)^{3} \leq \epsilon \tag{4.7}$$
The choice of ϵ can be estimated using techniques like the **Nearest Neighbors Elbow**, described in [26], which calculates the average distance between each point and its k nearest neighbors. The optimal ϵ value is typically determined at the point of maximum curvature on the k-distance graph.

- **minPts**: It represents the minimum number of data points required to form a cluster. The choice of this parameter depends on the number of dimensions in the dataset. A rule of thumb suggests setting $MinPts$ to be greater than or equal to the number of dimensions D plus one ($MinPts \geq D + 1$).

The application of DBSCAN in our workflow follows a series of formal steps:

- ϵ**(Epsilon) Value Choice**: The implementation of DBSCAN begins by determining an appropriate value for epsilon (ϵ). Epsilon defines the maximum distance between points to be considered part of the same neighborhood. In our workflow, we perform an iterative process to select the optimal ϵ value. We iterate through a range of epsilon values, incrementing by a step size of 0.2. The iteration starts from 1 and halts at NEIGHBOUR/2, which is the specified upper limit. For each epsilon value, we instantiate the DBSCAN algorithm with the given epsilon and apply it to the preprocessed data represented by the scaled dataset. The algorithm assigns cluster labels to data points, where non-anomalous points are assigned positive integers representing the clusters, and anomalies are labeled as -1. To evaluate the suitability of each epsilon value, we calculate the percentage of anomalies in the dataset based on the cluster labels obtained from DBSCAN. This calculation involves counting the number of points labeled as -1 (indicating anomalies) and dividing it by the total number of data points in the dataset. We compare the calculated percentage of anomalies to the specified contamination threshold. If the percentage falls below the contamination threshold, we store the current epsilon value as the final ϵ value and break the iteration.

- **DBSCAN Application**: After determining the appropriate ϵ value, we instantiate the DBSCAN algorithm again using the final epsilon value. We apply this final DBSCAN clustering to the scaled dataset, and the resulting cluster labels are used to label the anomalies in the dataset.

[3] The usual Distance used is the **Euclidean Distance**, but also other Distances could be employed.

GMM The third algorithm employed in our Workflow's Labeling Step, is the **GMM (Gaussian Mixture Model)**.

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k) \tag{4.8}$$

where:

$$\sum_{k=1}^{K} \pi_k = 1 \text{ and } \pi_k \geq 0, \forall k$$

GMM is a probabilistic clustering technique that models the data distribution as a **Mixture of Gaussian**[4] components.

The application of GMM within our ensemble follows a series of steps:

- **Initialization and Model Fitting**: Once initialized the GMM, using K-Means++ Initialization, the algorithm is then fitted to the preprocessed and scaled data, during which GMM estimates the parameters of the Gaussian components that best represent the underlying data distribution.
- **Cluster Assignment**: After, each data point is assigned a cluster label based on its maximum posterior probability.
- **Anomaly Score Calculation**: To quantify the anomaly level of each data point, we calculate the anomaly score, that represents the log-likelihood of each data point given the fitted GMM model. Lower scores indicate a higher likelihood of being an anomaly.
- **Threshold Determination**: To distinguish anomalies from normal data points, we determine a threshold based on the anomaly scores. In the provided code, we calculate the percentile threshold, specifying the desired contamination percentage. Data points with scores below this threshold are labeled as anomalies.
- **Anomaly Labeling**: Finally, we assign anomaly labels to the data points based on the determined threshold, by assigning a value of 1 to data points with anomaly scores below the threshold, indicating they are anomalies. Normal data points are labeled as 0.

Isolation Forest The fourth algorithm employed in the Ensemble is the **IsolationForest** [19] algorithm, also known as **iForest**, a powerful outlier detection technique that builds an ensemble of Isolation Trees (iTrees)[5] to identify anomalies in a given dataset. The core idea behind the algorithm is to isolate anomalies by measuring their short average path lengths on the iTrees. In an Isolation Forest, a random subsample of the data is processed using a tree structure based on randomly selected features. Data points that require fewer cuts to isolate them end up in shorter branches of the trees, making them likely anomalies. On the other hand, points that traverse deeper into the trees are less likely to be anomalies, as they needed more cuts to reach them.

[4] The **Gaussian Mixture Model** is the Computer Scientist's way of calling this method. This method has its density function which is a linear combination of Gaussian distributions, which is called **Mixture of Gaussians (MoG)**, defined as equation 4.8.

[5] An **iTree** is a proper binary tree, where each node in the tree has exactly zero or two daughter nodes.

Our implementation of the Isolation Forest algorithm follows a series of steps to ensure accurate AD:

- **Number of Estimators and Contamination Settings**: We start by specifying the number of estimators, which determines the number of trees used in the Isolation Forest ensemble. Additionally, we set the contamination parameter to the desired value, representing the expected proportion of anomalies in the dataset.
- **Isolation Forest Instantiation**: We select the relevant columns of data from the preprocessed dataset. Then, we instantiate an Isolation Forest object with the specified contamination and number of estimators. This prepares the algorithm for AD on the selected data.
- **Isolation Forest Fitting**: We fit the Isolation Forest model to the selected data, allowing it to learn the characteristics of normal data points and identify anomalies. During this process, the model assigns anomaly scores to each data point, indicating the degree of abnormality. Negative scores are assigned to anomalies, while positive scores indicate normal data points.
- **Anomaly/Non-Anomaly Prediction**: To facilitate further analysis and evaluation, we create a new column in the dataset specifically for storing binary labels representing anomalies. By using the predict method of the Isolation Forest model, we assign labels to each data point, labeling anomalies as -1 and normal data points as 1. To ensure consistency, we map these labels to 1 for anomalies and 0 for normal data points, and store the resulting values in the designated prediction column of the dataset.

Local Outlier Factor The fifth algorithm brought into play is the **Local Outlier Factor (LOF)**[6] algorithm, an unsupervised machine learning technique used to identify outliers in a dataset. Unlike traditional outlier detection methods that rely on global data distribution, LOF focuses on the local neighborhoods of each data point to determine its anomaly score.

In our implementation, we follow these steps to apply the LOF algorithm:

- **Number of Neighbors and Contamination Settings**: As the first step, we specify the number of neighbors to consider when calculating the local density for each data point. Additionally, we set the contamination parameter to the desired value, which represents the expected proportion of anomalies in the dataset.
- **LOF Instantiation**: We instantiate the LOF algorithm by creating a LOF object with the specified number of neighbors and contamination.
- **LOF Fitting and Anomaly Labeling**: Next, we fit the LOF model to the preprocessed and scaled dataset. The LOF algorithm calculates the anomaly scores for each data point based on its local density compared to its neighbors. Negative scores indicate anomalies, while positive scores represent normal data points. **Anomaly/Non-Anomaly Prediction**: To facilitate further analysis and evaluation, we create a new column in the dataset to store the binary labels for anomalies. We use the prediction method of the LOF model to assign labels to the data points: -1 for anomalies and 1 for normal data points. We map these labels to 1 for anomalies and 0 for normal data points and assign the resulting values to the LOF anomaly column in the dataset.

Robust Covariance: Elliptic Envelope Algorithm The second-last algorithm utilized is the **Robust Covariance (RC)** algorithm, a statistical method used to estimate the covariance matrix of a multivariate distribution, taking into account the presence of outliers. It aims to provide a more robust estimation of the covariance by reducing the influence of outliers on the estimation. This is achieved by using robust statistical measures, such as the median and the median absolute deviation, instead of the mean and the standard deviation. The robust covariance estimation is useful when dealing with data that may contain outliers or follow non-Gaussian distributions. It helps in modeling the data distribution more accurately and detecting anomalies that deviate significantly from the estimated covariance structure.

The **Elliptic Envelope**[15] is an algorithm that applies the concept of robust covariance to perform outlier detection. It assumes that the data follow a multivariate Gaussian distribution and aims to fit an ellipse (or ellipsoid) to the central region of the data distribution. This ellipse represents the majority of the data points, assuming they are not outliers. Any data point located outside the ellipse is considered an anomaly. The algorithm estimates the robust covariance matrix to determine the shape and size of the ellipse. The contamination parameter in the elliptic envelope determines the proportion of data points that are expected to be outliers.

The application of the Elliptic Envelope algorithm in our Ensemble Method, employed in the Workflow's Labeling Step, involves the following steps:

- **Contamination Setting**: Specify the contamination parameter, which represents the expected proportion of anomalies in the dataset. This parameter is used to determine the threshold for classifying points as anomalies.
- **Robust Covariance Estimation**: Instantiate the Robust Covariance detector, i.e., the Elliptic Envelope, with the specified contamination parameter. The algorithm fits the model to the preprocessed and scaled dataset, estimating the robust covariance matrices and other necessary parameters.
- **Anomaly Prediction**: Use the trained RC detector to predict anomalies in the scaled dataset, by returning a prediction label for each data point. Anomalies are typically labeled as -1, while normal data points are labeled as 1.
- **Anomaly Label Assignment**: Create a new column in the dataset to store the binary labels for anomalies. Map the predicted labels to 1 for anomalies and 0 for normal data points, assigning the resulting values to the anomaly column.

One Class SVM The last algorithm treated is the**One-Class Support Vector Machine (One-Class SVM)**[18], a robust AD algorithm designed to learn a decision boundary that encompasses the normal data points in an unsupervised manner. The objective of the One-Class SVM is to minimize the hypersphere's radius while enclosing the majority of the training data points, considering them as inliers, and treating the remaining data points as outliers.

The OneClass-SVM implementation involved in the Workflow's Labeling Step works as follows:

- **Training with Normal Data**: The One-Class SVM is trained on a set of normal data points, assuming that these points represent the majority of the data. The algorithm aims to find a hyperplane that encloses these normal

data points while maximizing the margin between the hyperplane and the data points. The hyperplane is a decision boundary that separates the normal data points from potential anomalies.

- **Support Vectors**: During the training process, the One-Class SVM identifies the support vectors, which are the data points that are closest to the learned hyperplane. These support vectors play a crucial role in defining the boundary of the normal data distribution.
- **Boundary Estimation**: Based on the support vectors, the One-Class SVM estimates the boundary that encloses the normal data points. This boundary is defined by a set of coefficients that determine the shape and position of the boundary in the feature space.
- **Anomaly Detection**: Once the boundary is established, the One-Class SVM can determine whether a new data point is normal or an anomaly by evaluating its position relative to the learned boundary. If a data point is located inside the boundary, it is considered normal; if it is located outside the boundary, it is classified as an anomaly.
- **Contamination Parameter**: The One-Class SVM algorithm has a hyperparameter called the contamination parameter, denoted by ν, which represents the expected proportion of anomalies in the data. By adjusting this parameter, you can control the sensitivity of the algorithm in detecting outliers. A higher value of ν allows for a higher percentage of anomalies, while a lower value makes the algorithm more conservative in identifying anomalies.

4.5.2 Contamination Factor Estimation: Transfer Learning

In the context of AD algorithms, the **Contamination Factor** plays a crucial role in determining the threshold for classifying samples as anomalies or non-anomalies, and it is employed in most of the algorithm cited above. The contamination factor represents the expected proportion of anomalies in a dataset. In many cases, estimating the contamination factor for each dataset separately can be a **Time-Consuming** and challenging task, especially when dealing with unlabeled data. However, transfer learning provides a valuable solution to this problem by leveraging knowledge from related datasets where the contamination factor has already been estimated. This **Transfer Learning** Approach exploits the assumption that datasets with similar anomaly/non-anomalous sample ratios share similar underlying characteristics. The key advantage of using transfer learning for contamination factor estimation is the significant reduction in time and effort required.

4.6 Workflow's Step 4: Real-Time Model Building

In the fourth step of our **Research Study Workflow**, we reach the core of our workflow, where our focus shifts to building a robust **(RT)Real-Time Anomaly Detection Model**. This model will be specifically designed to operate in a real-time environment, continuously analyzing incoming data from the customer's network. Its primary purpose is to identify and flag anomalous instances or patterns within the 4G/5G RANs' fleet data stream. The R-T AD Model assumes a critical role in safeguarding the network's stability and security. By promptly detecting anomalies, we can respond proactively to potential threats, minimizing downtime and ensuring

the smooth operation of the network infrastructure. This model is designed to analyze the streaming data and make instantaneous classifications. It leverages advanced techniques, employing algorithms such as Logistic Regression, Gradient Boosting Tree, MLP and Random Forest.

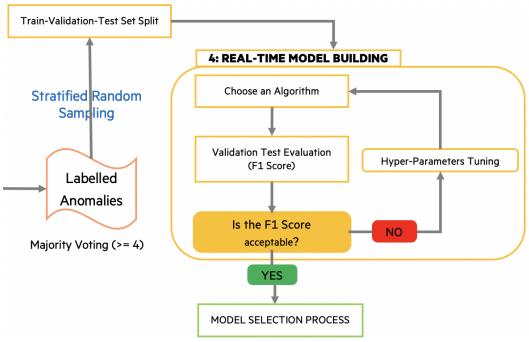


Figure 4.4. Model Building Iterative Process: The fourth step of the MAD Workflow consists in an iterative process, which starts with one model type choice. After the hyper-parameters setting, we evaluate the Validation Test Set of the model. If we are satisfied with the F1 Score it obtains, we save the model for the next Model Selection Decision. Otherwise, we restart the process by applying some Hyper-Parameters Fine-Tuning. At the end of this process, we will have the best versions of each algorithm we have chosen.

4.6.1 Training-Test-Validation Split

Before delving into the complexities of building a robust machine-learning model for AD, it is imperative to prepare the dataset appropriately. Dataset splitting involves dividing the available data into distinct subsets, each serving a specific purpose in the model development and evaluation process. The **Train-Validation-Test Split** is a technique to evaluate the performance of your machine learning model. You take a given dataset and divide it into three subsets, in order to ensure reliable model performance evaluation and generalization: Training, Test, and Validation. The **70-10-20 Train-Validation-Test Split** is the Split we employed in this Thesis.

Stratified Random Sampling Due to the High Imbalance between the two classes (Anomalous/Non-Anomalous) observed in the AD field and in the received dataset, employing a Standard Random Sampling Split might result in biased training, test, and validation sets. Indeed, due to the very low amount of anomalous samples, it could happen that, after the split, the three subset does not respect the actual Dataset's distribution. This means that, in the worst case, one of the subsets could also contain any of the anomalous samples. To address this issue and ensure representative subsets, we employed **Stratified Random Sampling**, that involves dividing the dataset into subgroups based on a specific attribute, such as the target variable or class labels. In our case, we stratified the sampling based on the target variable, ensuring that each subset maintains the original class distribution. By using stratified random sampling, we preserve the proportion of positive(in our case Anomalous) and negative(in our case Non-Anomalous) instances in each subset, allowing the model to learn from and evaluate its performance on a representative sample.

4.6.2 Logistic Regression

The first Model to be included in the **Best Model Version Selection** Process is **Logistic Regression**[7][21]. This ML Model is a widely used discriminative classification model $p(y \mid \mathbf{x}; \boldsymbol{\theta})$, where:

- $\mathbf{x}$[6] $\in \mathbb{R}^{\mathbb{D}}$ is a fixed-dimensional input vector;
- $y \in \{1, \ldots, C\}$ is the class label;
- $\boldsymbol{\theta}$ are the parameters.

If C = 2, this is known as **Binary Logistic Regression**, and if C > 2, it is known as **Multinomial Logistic Regression**, or alternatively, Multiclass Logistic Regression.

Binary Logistic Regression In my case, I focused on the Binary Logistic Regression case, in order to focus on the Anomaly / Normal Behaviour Classes distinction. More in particular, Binary Logistic Regression corresponds to the following model:

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = Ber(y \mid \sigma(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} + b)) \tag{4.9}$$

where:

- σ is the Sigmoid function;
- $\boldsymbol{w}$ are the Weights;
- $\boldsymbol{b}$ is the Bias;
- $\boldsymbol{\theta} = (\boldsymbol{w}, b)$ are all the parameters.

Logistic Regression: Threshold Moving By default, the **Decision Boundary**, i.e., the Logistic Regression threshold is set at 0.5, meaning that if the estimated probability is greater than or equal to 0.5, the instance is classified as Anomaly; otherwise, it is classified as Normal Behaviour Sample. In certain scenarios, however, the default threshold may not be optimal for a specific application or problem domain. Adjusting the threshold can have a significant impact on the model's predictive performance, particularly when dealing with imbalanced datasets or situations where different misclassification costs are involved. **Threshold Tuning**, a.k.a. **Threshold Moving**, or simply known as **Thresholding**[31][13] is a technique that allows us to optimize the trade-off between the model's sensitivity (recall) and specificity. By varying the threshold, we can adjust the balance between false positives and false negatives, depending on the desired emphasis on either class.

More formally, given the F1 Score, we can find the optimal Threshold by:

$$t_{\text{optimal}} = \arg\max_{t} \text{F1-Score}(t) \tag{4.10}$$

A higher threshold may result in an increased sensitivity to anomalies but at the cost of a higher false positive rate. Conversely, a lower threshold may lower the false positive rate, reducing the number of false alarms but potentially leading to an increased number of false negatives (undetected anomalies).

[6] Notice that, we use the **Bold** for the letters which denote Vectors.

Logistic Regression: PySpark Focus In PySpark, the **LogisticRegression** module provides an implementation of Logistic Regression for large-scale datasets. The Logistic Regression model in PySpark is based on the binomial logistic regression algorithm and it has, again, the goal to learn a decision boundary that separates the two classes based on the input features. PySpark's LogisticRegression module provides several **parameters** that can be adjusted to customize the behavior of the model. Here, we will discuss the most important parameters we have used:

- *maxIter*: This parameter determines the maximum number of iterations that the Logistic Regression algorithm will perform. It controls the convergence of the algorithm. Increasing the number of iterations can lead to a more accurate model but may also increase training time. Here we should find a trade-off between the convergence-training time factors.
- *threshold*: This parameter sets the classification threshold for predicting the positive class. It is a value between 0 and 1. This parameter will be used for the Threshold Moving application discussed in Logistic Regression: Threshold Moving(paragraph4.6.2).

4.6.3 MultiLayer Perceptron Classifier

The next Model we explored and evaluated in the **Best Model Version Selection** process is the **MultiLayer Perceptron Classifier (MLPC)**. The MLP is a type of feedforward artificial neural network that consists of multiple layers of interconnected nodes (neurons). It is known for its ability to capture complex non-linear relationships within the data. Its ability to learn and adapt to intricate patterns and dependencies within the data made it an intriguing candidate for our R-T AD Model. We carefully fine-tuned the MLP model, experimenting with different architectures, activation functions, and hyperparameters.

Multilayer Perceptron (MLP) The MLP has essentially three main types of Layers:

- **Input Layer**: It consists of neurons that receive the input data. Each neuron represents a feature or attribute of the input. The number of neurons in the input layer is determined by the dimensionality of the input data. Let's denote the number of input neurons as N. Each neuron in the input layer receives a corresponding input value. The input values are usually represented as a vector $\mathbf{X}$ of length N, where $X = [x_1, x_2, ..., x_N]$.
- **Hidden Layers**: The hidden layers of an MLP are intermediate layers between the input layer and the output layer. They perform computations on the input data using weighted connections between neurons. An MLP can have one or more hidden layers, and each hidden layer contains multiple neurons. The number of hidden layers and the number of neurons in each hidden layer are hyperparameters that need to be specified. Let's assume the MLP has L hidden layers, denoted as $l_1, l_2, \ldots, l_L$. The number of neurons in the i-th hidden layer (l_i) is denoted as N_{l_i}. Each neuron in the hidden layers applies an activation function to the weighted sum of inputs received from the previous layer.
- **Output Layer**: The output layer of an MLP produces the final output or prediction based on the computations performed in the hidden layers. The number of neurons in the output layer depends on the type of problem being

solved. For a binary classification problem (as in our AD case), the output layer usually has a single neuron that represents the probability or score of belonging to one class. For multi-class classification problems, the output layer has multiple neurons, with each neuron representing the probability or score for a specific class. Let's denote the number of neurons in the output layer as N. Each neuron in the output layer produces an output value that represents the predicted output or class probabilities. The output values can be represented as a vector $\mathbf{Y}$ of length N, where $\mathbf{Y} = [y_1, y_2, ..., y_N]$.

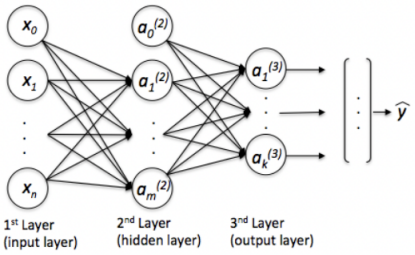


Figure 4.5. The **MLP Basic Architecture**.

MLP: PySpark Focus In PySpark, the **MLPClassifier** class provides an implementation of the Multilayer Perceptron algorithm. It employs a forward propagation approach to process input data through the network and produce predictions. The model can handle both Binary and Multi-class Classification problems.

The MLPClassifier in PySpark offers several important **parameters** that can be tuned to optimize its performance:

- *layers*: This parameter specifies the architecture of the neural network by defining the number of nodes in each layer. It takes a list of integers representing the number of nodes in each layer. The first element should be equal to the number of features in the input data, and the last element should be equal to the number of classes in the classification task.

- *maxIter*: This parameter controls the maximum number of iterations, more commonly known in the Deep Learning field as number of **epochs** the model will undergo during the training process. It helps determine the convergence of the model.

- *blockSize*: The MLPClassifier supports mini-batch training, which improves the efficiency of the training process. The blockSize parameter, a.k.a. **batch-size** determines the number of samples to be processed in each iteration.

- *seed*: This parameter allows for the specification of a random seed, which ensures reproducibility of the results.

- *solver*: The solver parameter determines the optimization algorithm used to train the neural network. PySpark's MLPClassifier supports two solvers: **l-bfgs**, that is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory, and the **gradient descent**.

- *tol*: This parameter stands for tolerance and represents the convergence tolerance for iterative algorithms used in MLP training. It determines the minimum

improvement required for the optimization algorithm to continue iterating. A smaller tol value indicates a stricter convergence criterion, potentially leading to longer training times, but may yield a more accurate model.

- *stepSize*: This parameter defines the step size or learning rate used during the optimization process in MLP training. It controls the magnitude of the updates made to the model's weights at each iteration. A larger stepSize can lead to faster convergence but may also risk overshooting the optimal solution. Conversely, a smaller stepSize might require more iterations for convergence but can offer greater stability during training.

4.6.4 Stochastic Gradient Boosting

In our pursuit of building an effective real-time anomaly detection model, we successively explored the **Stochastic Gradient Boosting (SGB)**, an ensemble learning technique that combines the power of gradient boosting with the added benefit of randomization, making it a robust and versatile algorithm for classification tasks.

After that Breiman [5] introduced the notion that injecting **randomnsess** into function estimation procedures could improve both the accuracy and the execution speed, he has also proposed a hybrid bagging-boosting procedure (a.k.a. **Adaptive Bagging**) intended for least-squares fitting of additive expansions. It replaces the Base Learner in regular boosting procedures with the corresponding Bagged Base Learner, and substitutes "out-of-bag" residuals for the ordinary residuals at each boosting step. Motivated by Breiman [12], a modification has been presented to Gradient Boosting incorporating randomness as an integral part of the procedure. Specifically, at each iteration, a subsample of the training data is drawn at random (without replacement) from the full training dataset. This randomly selected subsample is then used in place of the full sample to fit the Base Learner and compute the model update for the current iteration. This randomized approach also increases robustness against the overcapacity of the base learner.

Stochastic Gradient Boosting: PySpark Focus In PySpark, the **SGBClassifier** module is implemented as part of the MLlib library, providing a scalable and distributed framework for training gradient boosted trees on large datasets. PySpark's GBTClassifier module provides several **parameters** that can be adjusted to customize the behavior of the model. Here, we will discuss the most important parameters we have used:

- *maxDepth*: This parameter controls the maximum depth of the individual decision trees in the gradient boosting process. Increasing the maxDepth allows the trees to capture more complex relationships in the data, but it can also lead to overfitting if set too high.

- *maxBins*: The maxBins parameter determines the maximum number of bins to be used for discretizing continuous features. Higher values can capture more information but may also increase the computational overhead.

- *maxIter*: The maxIter parameter specifies the maximum number of iterations (i.e., the number of boosting stages) to perform. Increasing this value allows the model to learn more complex patterns but requires more computational resources.

- *stepSize*: Also known as the *learning rate*, the stepSize parameter controls the shrinkage applied to the contribution of each tree in the ensemble. A smaller stepSize can make the model more robust to overfitting but may require more iterations to converge.

4.6.5 Random Forest

As part of our comprehensive **Best Model Version Selection** process, we also delved into the **Random Forest** model, an ensemble learning variant that leverages the concept of decision trees to achieve robust classification performance.

Notwithstanding the ease in the interpretation and in handling the mixing of discrete and continuous input, the insensitivity to monotone transformation (i.e., no need for Data Standardization), the automatic variable selection, the relative robustness w.r.t. outliers we have that **Decision Trees** do not predict very accurately compared to other models. This is in part due to the greedy nature of the tree construction. Moreover, another big problem we find in the Decision Trees, is that they are very **unstable**: small changes to the input data can have large effects on the structure of the tree. In other words, decision trees are a high-variance estimators.

A simple way to reduce variance is to average multiple models. This is called **Ensemble Learning**, already explained in Ensemble of Methods(section 4.5.1). The resulting model has the form:

$$f(y|\boldsymbol{x}) = \frac{1}{|\mu|} \sum_{m \in \mu} f_m(y|\boldsymbol{x}) \tag{4.11}$$

where f_m is the m-th base model. The ensemble will have a similar bias to the base models, but lower variance, generally resulting in improved overall performance.

Bagging[5], which stands for "Bootstrap Aggregating", is a simple form of ensemble learning in which we fit M different base models to different randomly sampled versions of the data, in order to encourage the different models to make diverse predictions. The datasets are sampled with replacement (a technique known as **Bootstrap Sampling**), so a given example may appear multiple times until we have a total of N examples per model (where N is the number of original data points). Bagging solves the **Variance Problem**. We can think of this process as injecting noise and then removing it by averaging, thus obtaining a Smoothing Effect. Indeed, Decision Trees, will be trained on different datasets and will learn different decision boundaries. Thus, the application of Bagging to Decision Trees results in the **Random Forest** Machine Learning Model.

The disadvantage of Bootstrap is that each base model only sees, on average, 63% of the unique input examples. The 37% of the training instances that are not used by a given base model are called **Out-of-Bag instances (oob)**. We can use the predicted performance of the base model on these oob instances as an estimate of test set performance. This provides a useful alternative to cross-validation.

Random Forest: PySpark Focus In PySpark, the **Random Forest** Algorithm is implemented as part of the MLlib library. The various **parameters** associated with Random Forest we have experienced with are:

- *NumTrees*: This parameter determines the number of decision trees to be used in the ensemble. Increasing the number of trees generally improves the model's accuracy, but it also increases the computational complexity. Finding an optimal value for NumTrees requires careful consideration of the trade-off between accuracy and efficiency.

- *MaxDepth*: This parameter determines the number of decision trees to be used in the ensemble. Increasing the number of trees generally improves the model's accuracy, but it also increases the computational complexity. Finding an optimal value for NumTrees requires careful consideration of the trade-off between accuracy and efficiency.

- *FeatureSubsetStrategy*: This parameter determines the number of features to consider when splitting each tree node. The options include "auto," "all," "sqrt," "log2," and specific fractional values. Choosing an appropriate value depends on the nature of the dataset and the desired level of feature diversity in each tree.

- *Impurity*: Random Forest supports different impurity measures, such as Gini impurity and entropy, to evaluate the quality of each split. The choice of impurity measure depends on the specific classification problem and the desired trade-off between computational efficiency and model accuracy.

4.6.6 Ensemble of Methods: MLP + LOG-REG + SGB

In addition to the exploration of individual machine learning models, we delved into the realm of **Ensemble Methods**, as expounded in section 4.5.1, to further enhance the performance and robustness of our AD system. The underlying motivation for adopting this ensemble approach was to harness the distinctive strengths and characteristics of each individual model and amalgamate their predictions to arrive at a more informed decision. By amalgamating multiple models, our goal was to encompass a broader range of features and patterns present in the data, ultimately leading to improved AD performance.

The ensembles we proposed and thoroughly experimented with emerged from a process of iterative experimentation, wherein we excluded the Random Forest model due to its observed inadequacy, as will be elucidated in the subsequent chapter. The final **Ensemble** that proved to be most effective comprises a **combination of MultiLayer Perceptron (MLP), Logistic Regression (LOG-REG), and Stochastic Gradient Boosting (SGB)** as its constituent components.

To utilize the ensemble of MLP, LOG-REG, and SGB models, we explored three main **Voting Strategies**:

- **Hard Voting**: In our R-T AD model, we have employed one typology of **Hard-Voting** or **Majority Voting**. This decision-making technique combines the predictions of multiple individual models, including MultiLayer Perceptron (MLP), Logistic Regression, and Gradient Boosting Trees (GBT), to arrive at a final decision. The idea behind Hard-Voting is to leverage the collective wisdom of diverse models and take advantage of their complementary strengths. By combining their predictions, we aim to achieve a more robust and reliable AD system. The **Hard-Voting Decision Process** is straightforward. For a given input sample, each model in the ensemble independently makes a prediction on whether it is an anomaly or not. The final decision is determined by taking the majority vote among the models. If the majority of models classify the sample as an anomaly, it is labeled as such; otherwise, it is considered normal.

  More formally, let $\hat{y}_{\text{MLP}}$, $\hat{y}_{\text{LOG-REG}}$ and $\hat{y}_{\text{GBT}}$ be respectively the predictions (either 0 or 1) of the MLP, Logistic Regression, and Stochastic Gradient Boosting Models, and let τ, be the Hard-Voting Threshold over which the sample would be considered Anomaly(in our case $\tau = 2$). We define the

Hard-Voting as:

$$\mathrm{H}_{\mathrm{Voting}} = \begin{cases} 1, & \text{if } \hat{y}_{\mathrm{MLP}} + \hat{y}_{\mathrm{LOG\text{-}REG}} + \hat{y}_{\mathrm{GBT}} \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

- **Soft Voting**: Our R-T AD model also incorporates the **Soft Voting** approach, providing four different variations. Unlike Hard-Voting, which considers only the majority vote over 0/1 Labels, Soft Voting takes into account the probabilities assigned to each class by each model.

  More formally, we let $\hat{p}_{\mathrm{MLP}}$, $\hat{p}_{\mathrm{LOG\text{-}REG}}$ and $\hat{p}_{\mathrm{GBT}}$ be respectively the probability predictions of the MLP, Logistic Regression, and Stochastic Gradient Boosting Models, and let τ, be the Soft-Voting Threshold over which the sample would be considered Anomaly. We define Soft Voting as:

$$\mathrm{S}_{\mathrm{Voting}} = \begin{cases} 1, & \text{if } \hat{p}_{\mathrm{S}_{\mathrm{Voting}}} \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

  Depending on what **Soft Voting Variation** we decide to opt for, we have that the $\hat{p}_{\mathrm{S}_{\mathrm{Voting}}}$ and the τ parameters change accordingly. The different Soft Voting approaches allow us to tailor the decision-making process based on the desired level of leniency or conservatism in AD. In particular, the variations we have implemented are:

  - **Soft Voting (SUM)**: In this variation, the probability estimates from all the models are summed up, and the final decision is made by comparing this cumulative probability against a predefined threshold τ. We, therefore, consider the sum of probabilities to make the final decision, providing a more lenient approach. By summing up the probabilities, we incorporate the collective confidence of the ensemble in a more inclusive manner, which can be beneficial when the individual models exhibit varying levels of certainty. More formally, we have:

$$\hat{p}_{\mathrm{S}_{\mathrm{Voting}}} = \hat{p}_{\mathrm{MLP}} + \hat{p}_{\mathrm{LOG\text{-}REG}} + \hat{p}_{\mathrm{GBT}} \tag{4.14}$$

  - **Soft Voting (MAX)**: In this variation, the final decision is made by comparing the maximum probability among the models against a predefined threshold τ. We, therefore, consider the most confident prediction among the individual models to make the final decision, providing a more conservative approach compared to the Soft Voting (SUM) variant. By selecting the maximum probability, we focus on the model that exhibits the highest level of confidence, ensuring a stronger signal for detecting anomalies. More formally, we have:

$$\hat{p}_{\mathrm{S}_{\mathrm{Voting}}} = \max(\hat{p}_{\mathrm{MLP}}, \hat{p}_{\mathrm{LOG\text{-}REG}}, \hat{p}_{\mathrm{GBT}}) \tag{4.15}$$

  - **Soft Voting (MEAN)**: In this variation, we take the mean of the predicted probabilities among the models and compare it against a predefined threshold τ. The Soft Voting (MEAN) approach provides a balanced compromise between Soft Voting (SUM) and Soft Voting (MAX) by considering the average confidence level of the ensemble. More formally, we have:

$$\hat{p}_{\text{S}_{\text{Voting}}} = \frac{\hat{p}_{\text{MLP}} + \hat{p}_{\text{LOG-REG}} + \hat{p}_{\text{GBT}}}{3} \tag{4.16}$$

  - **Soft Voting (MEDIAN)**: In this variation, we take the median of the predicted probabilities among the models and compare it against a predefined threshold τ. The Soft Voting (MEDIAN) approach provides a more robust decision by considering the middle value of the predicted probabilities, making it less sensitive to outliers and extreme values. More formally, we have:

$$\hat{p}_{\text{S}_{\text{Voting}}} = \text{median}(\hat{p}_{\text{MLP}}, \hat{p}_{\text{LOG-REG}}, \hat{p}_{\text{GBT}}) \tag{4.17}$$

    The Soft Voting decision offers some **advantages** over Hard-Voting. Indeed, we have a more **Fine-grained Decision-making** due to the fact that Soft Voting takes into account the probability estimates, allowing for more nuanced decision-making. Instead of relying solely on the majority vote, it considers the confidence levels of each model's predictions.

    Despite these advantages, Soft Voting also has its **considerations**. It requires one additional fine-tuning step, the **Threshold Selection**. The choice of the threshold (τ) in the Soft Voting decision rule is critical, determining the balance between FP and FN. Selecting an appropriate threshold requires careful evaluation and consideration of the specific requirements and characteristics of the AD task. However, to simplify the decision-making process and avoid an additional step, we can use predefined thresholds that we have found to work well in general. This approach eliminates the need for fine-tuning the threshold for each specific scenario and allows for a more streamlined implementation of the AD system.

- **Soft Voting Weighted**: This approach aims to enhance the ensemble decision-making process by assigning higher weights to classifiers that are considered more accurate or reliable. By considering the performance metrics of each model, we can fine-tune the ensemble's decision-making process, leveraging the strengths of individual models to improve overall performance. The process for weight computation follows a common backbone across all five metrics explored, ensuring consistency and ease of implementation. The general steps in this process are as follows:

  - **Model Evaluation**: The first step consists in evaluating the performance of each model in the ensemble using a specific performance metric (e.g., F1 Score, Entropy, Margin Maximization, Brier Score, Expected Calibration Error).
  - **Weight Computation**: After that, we can proceed in computing the weight for each model based on its performance metric. As a successive step, we normalize the weights to ensure they sum up to 1.
  - **Weighted Probability**: For a given instance x and model m, the weighted probability $p(x)_w^m$ is calculated as:

$$p(x)_w^m = p(x)^m \cdot w^m \tag{4.18}$$

    where $p(x)^m$ is the predicted probability of model m for instance x, and w^m is the weight assigned to model m.

– **Ensemble Decision**: We can, successively, combine the weighted probabilities for each instance and model, and compare the result to a threshold value τ to make the final prediction:

$$\mathrm{S}_{\mathrm{Votingw}} = \begin{cases} 1, & \text{if} \sum_{m=1}^{M} p(x)_w^m \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{4.19}$$

where M is the number of models in the ensemble (in our case, M=3).

Depending then on the metric we employ, we have some small modifications to do. In particular, we proposed 5 Soft Voting Weighted Solutions:

– **Soft Voting Weighted (F1-Score)**: One of the proposed Weighted Ensemble approaches, we will use the F1 Score, described in section 4.7.1, as weight. In particular, we have:
  * **Model Evaluation**: We evaluate each model in the ensemble using the F1 Score, denoted as $F1^m$ for model m.
  * **Weight Computation**: We compute the weight w^m for each model based on its F1 Score, as:

$$w^m = \frac{F1^m}{\sum_{i=1}^{M} F1^i} \tag{4.20}$$

where M is the number of models in the ensemble.

– **Soft Voting Weighted (Entropy)**: Another Weighted Ensemble approach we use provides the Entropy to weight the Models Predictions. More, in particular, we have:
  * **Entropy Computation**: We first calculate the entropy of each model's predictions to quantify uncertainty. Let $p(x)_i^m$ be the i-th predicted probability of model m for instance x, and N be the number of instances.

$$\mathrm{Entropy}^m = -\frac{1}{N} \sum_{x=1}^{N} \sum_{i=1}^{C} p(x)_i^m \log(p(x)_i^m) \tag{4.21}$$

where C is the number of classes (2 for our binary classification).
  * **Normalization of Entropy**: We proceed then to normalize the entropy values to a range of [0, 1] using Min-Max Scaling to ensure comparable weights across models. Let's define $\mathrm{Entropy}_{\max}$ and $\mathrm{Entropy}_{\min}$ to be:

$$\mathrm{Entropy}_{\max} = \max(\mathrm{Entropy}^1, \ldots, \mathrm{Entropy}^M) \tag{4.22}$$

$$\mathrm{Entropy}_{\min} = \min(\mathrm{Entropy}^1, \ldots, \mathrm{Entropy}^M) \tag{4.23}$$

We can then define formally define the Normalized Entropy for a model m to be:

$$\mathrm{Entropy}_{\mathrm{norm}}^m = \frac{\mathrm{Entropy}^m - \mathrm{Entropy}_{\min}}{\mathrm{Entropy}_{\max} - \mathrm{Entropy}_{\min}} \tag{4.24}$$

  * **Weight Computation**: We then compute the weight w^m for each model by subtracting the normalized entropy from 1:

$$w^m = 1 - \mathrm{Entropy}_{\mathrm{norm}}^m \tag{4.25}$$

– **Soft Voting Weighted (Margin Maximization)**: In this Weighted Ensemble approach we use the Margin Maximization to weight the Models Predictions. More, in particular, we have:
  * **Prediction Probability Matrix**: First of all, we organize the predicted probabilities of each model into a matrix $P \in \mathbb{R}^{N \times M}$, where N is the number of instances and M is the number of models in the ensemble.
  * **Margin Calculation**: We then proceed to calculate the margin for each instance by sorting the predicted probabilities and taking the difference between the top two probabilities:

$$\text{Margin}_x^m = p(x)_{(1)}^m - p(x)_{(2)}^m \tag{4.26}$$

  where $p(x)_{(1)}^m$ is the highest predicted probability and $p(x)_{(2)}^m$ is the second highest predicted probability of model m for instance x.
  * **Margin Normalization**: We then normalize the margins to ensure the weights sum up to 1 for each instance:

$$\text{Margin}_{\text{norm}_x^m} = \frac{\text{Margin}_x^m}{\sum_{i=1}^{M} \text{Margin}_x^i} \tag{4.27}$$

  * **Weight Assignment**: At the very end, we assign weights to each model based on the normalized margins:

$$w_x^m = \text{Margin}_{\text{norm}_x^m} \tag{4.28}$$

– **Soft Voting Weighted (Brier Score)**: In this Weighted Ensemble approach, we use the Brier Score as a metric to determine the weights of individual models in the ensemble. The steps are as follows:
  * **Brier Score Computation Instance-wise**: For each instance in the dataset, calculate the Brier Score by comparing the predicted probabilities with the true binary outcomes. Let $p(x)_i^m$ be the predicted probability of model m for instance x, and y be the true binary outcome for that instance. The Brier Score for each instance is given by:

$$\text{BrierScore}(x) = \frac{1}{C} \sum_{i=1}^{C} (p(x)_i^m - y)^2 \tag{4.29}$$

  where C is the number of classes (2 for our binary classification).
  * **Average the Brier Scores**: We calculate the average Brier Score for the entire dataset:

$$\text{BrierScore}_{\text{avg}} = \frac{1}{N} \sum_{x=1}^{N} \text{BrierScore}(x) \tag{4.30}$$

  where N is the number of instances in the dataset.
  * **Weight Computation**: We then proceed to compute the weight w^m for each model based on its Brier Score. The weight is calculated as the inverse of the Brier Score, normalized across all models to

ensure the weights sum up to 1. Higher Brier Scores result in lower weights for better-performing models:

$$w^m = \frac{1}{\text{BrierScore}^m} \cdot \frac{1}{\sum_{i=1}^{M} \frac{1}{\text{BrierScore}^i}} \tag{4.31}$$

where M is the number of models in the ensemble.

- **Soft Voting Weighted (Expected Calibration Error)**: In this Ensemble approach, we use the Expected Calibration Error (ECE) as a metric to determine the weights of individual models in the ensemble. The steps are as follows:
  - **Predicted Probabilities' Bins Division**: As the first step, we divide the predicted probabilities into several bins or intervals.
  - **Average Predicted Probabilities and Empirical Probabilities bin-wise Computation**: For each bin, we compute the average predicted probabilities of the instances falling within that bin. Additionally, calculate the empirical probability by averaging the true binary outcomes of the instances in that bin.
  - **Difference between Average Predicted Probabilities and Empirical Probabilities bin-wise Computation**: We then calculate the absolute difference between the average predicted probabilities and empirical probabilities for each bin.
  - **Differences' Weighting w.r.t. Instances Proportion in each Bin**: We then proceed to weight the differences by the proportion of instances in each bin.
  - **Weighted Differences Averaging**: We calculate the average of the weighted differences to obtain the ECE value.
  - **Weight Computation**: Compute the weight w^m for each model based on its ECE value. The weight is calculated as the inverse of the ECE, normalized across all models to ensure the weights sum up to 1. A lower ECE value indicates better calibration, indicating that the predicted probabilities align more closely with the empirical probabilities. On the other hand, a higher ECE value suggests miscalibration, indicating a discrepancy between the predicted probabilities and the true probabilities:

$$w^m = \frac{1}{\text{ECE}^m} \cdot \frac{1}{\sum_{i=1}^{M} \frac{1}{\text{ECE}^i}} \tag{4.32}$$

where M is the number of models in the ensemble.

4.7 Workflow's Step 5: Model Selection

In the **fourth** of our **Research Study Workflow**, we have arrived at a critical juncture where we need to make a well-informed decision regarding the best model to be used for our Real-Time Anomaly Detection System. Throughout our research journey, we have diligently explored and evaluated various ML and DL models, employing a robust and systematic **Best Model Decision Workflow** that takes into account their performance across multiple dimensions.

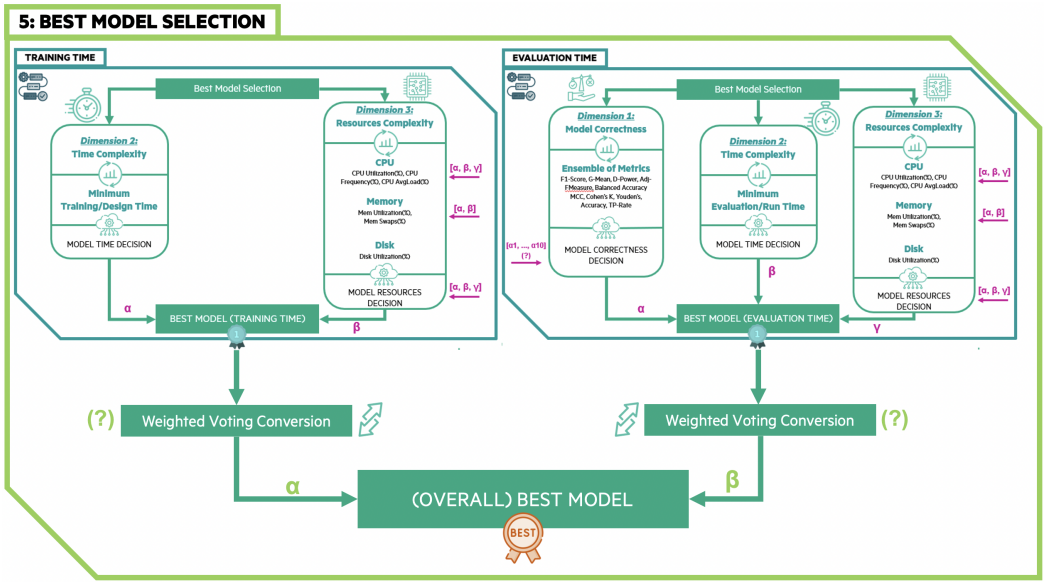


Figure 4.6. Best Model Decision Workflow: The proposed Decision Workflow encompasses two parallel evaluations: **Training Time** (also known as **Design Time**) and **Evaluation Time** (also known as **Run-Time**). During the Training Time evaluation, we assess the model's performance in terms of Time and Resource Consumption. On the other hand, the Evaluation Time evaluation focuses on evaluating the model based on Correctness, Time, and Resource Consumption. After performing the two parallel analyses, we amalgamate the decisions taken in each evaluation to arrive at an Overall Decision that synthesizes the insights gained from both Training Time and Evaluation Time, resulting in an **Overall Decision**. Within the Best Model Decision Workflow, we provide developers with the flexibility to fine-tune various model options and hyperparameters to cater to the specific requirements and constraints of our customer's Real-Time Anomaly Detection System. This tailored approach ensures that the chosen model aligns optimally with the unique characteristics of the data and the specific needs of the customer's application. The image graphically illustrates the step-by-step workflow that the Multivariate Anomaly Detection Tool uses to determine the best model. In the Workflow, hyper-parameters, set by the Developer, are highlighted in fuchsia and light green, while Optional steps, dependent on the hyper-parameter choice, are marked with "?".

4.7.1 Ensemble of Metrics

One of the main issues in **Learning with Class Imbalance** distribution is that most standard algorithms are **Accuracy-Driven**. In simpler terms, this means that many classification algorithms operate by minimizing the overall error that is, trying to maximize the classification accuracy. However, in a class imbalance dataset, classification accuracy tells very little about the minority class. Choosing accuracy

as the performing criterion in class imbalance classification may give inaccurate and misleading information about a classifier's performance.

Suppose a case scenario of a dataset with an imbalance ratio of 1:100(that is very similar to the cases we have dealt with). The ratio suggests that for each example of minority class(Anomaly), there are 100 majority (Normal Behaviour) class examples. A classification algorithm that tries to maximize accuracy to meet its objective-rule, will produce an accuracy of 99% just by correctly classifying all examples from the majority class but misclassifying one example of the minority class(this is a.k.a. **Base Constant Classifier**, i.e. a simple classifier that classifies every sample as being Normal Behaviour).

Having said this, we highlight the critical need to go beyond accuracy as the sole evaluation metric when dealing with highly imbalanced datasets. To overcome the limitations of accuracy and gain deeper insights into model performance, we advocated and proposed the **Ensemble of Metrics** approach[2], comprising 10 diverse evaluation measures, to provide a comprehensive and robust assessment of model performance in highly imbalanced settings. The ensemble of metrics offers several **Advantages** over relying solely on accuracy.

- Firstly, it provides a **more accurate** representation of model performance by considering various performance indicators tailored to imbalanced datasets.
- Secondly, it helps **identify the trade-offs** between different evaluation measures and assess the model's ability to correctly classify both the majority and minority classes.
- Additionally, the ensemble approach **facilitates model comparison and selection**, enabling us during the research to make informed decisions based on a comprehensive evaluation.

The Metrics we have proposed are the following: F1Score, Geometric Mean, Discriminant Power, Adjusted F-Measure, Balanced Accuracy, Matthew's Correlation Coefficient, Cohen's Kappa, Youden's Index, Accuracy, and TP-Rate. Each metric provides unique insights into different aspects of model performance, addressing specific concerns related to imbalanced datasets. By leveraging this diverse set of metrics, we aim to capture a holistic view of the model's performance and address the challenges posed by class imbalance.

Confusion Matrix To measure the effectiveness of our developed machine learning algorithm, we rely on the **Confusion Matrix**, a fundamental tool that provides valuable insights into the classification results.

The confusion matrix is a square matrix that summarizes the predictions made by a machine learning algorithm in comparison to the actual ground truth. It consists of four key elements, in which columns represent the classifier's predictions and the rows are the actual classes:

- **True Positives (TPs)**: It corresponds to the number of positive cases correctly classified as such. In our case, it corresponds to the number of Anomalies recognized as such by the algorithm.
- **False Positives (FPs)**: It corresponds to the number of negative cases that are incorrectly identified as positive cases. In our case, it corresponds to the number of Normal Behavioured Samples that are wrongly classified as Anomalies. Sometimes, they are also referred to as **False Alarms**.

- **False Negatives (FNs)**: It corresponds to the number of positive cases incorrectly classified as negatives. In our case, it corresponds to the number of Anomalies wrongly classified as Normal Behaviored Samples. Sometimes, they are also referred to as **Misses**.

- **True Negatives (TNs)**: It corresponds to the number of negative cases correctly classified as such. In our case, it corresponds to the number of Normal Behavioured Samples recognized as such by the algorithm.

We have followed the convention that provides the minority class(in our case, Anomaly class) in imbalanced data modeling is considered as the positive class whilst the majority class(in our case, Normal Behaviour class) is considered as the negative class.

The confusion matrix has a pivotal role in our networking-related research and it's a powerful tool that has helped us to emphasize our objective of achieving zero False Negatives (FN) and maximizing the True Positive Rate (TPR).

In our networking-related research, the significance of the confusion matrix lies in its ability to measure the accuracy and reliability of our developed machine-learning algorithm. Our ultimate goal is to **Minimize** the occurrence of **False Negatives**, which corresponds to Anomalies instances being incorrectly classified as Normal Behavior ones. In networking scenarios, false negatives can have significant consequences, such as overlooking critical network events and therefore it is of pivotal importance to detect them. To address this challenge, we strive to achieve a **TP-Rate** that is as close to 100% as possible, ensuring that our algorithm can be reliable in real-world networking applications.

Metrics As mentioned earlier, the Ensemble of Metrics we proposed is constituted of 10 metrics, which are:

- **F1Score**: It is a measure of a test's accuracy, being the harmonic mean of the precision and recall in one metric.

$$F1Score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{4.33}$$

  with:

  - **Precision**: It is the number of true positive results divided by the number of all positive results. It is computed as:

$$Precision = \frac{TP}{TP + FP} \tag{4.34}$$

  - **Recall**: It is a.k.a. **TP-Rate**, **Specificity** or **Hit-Rate** and it measures the accuracy of positive cases(Anomaly Samples). It is computed as:

$$Recall = \frac{TP}{TP + FN} \tag{4.35}$$

  The F1-Score Range is $[0, 1]$ where if it is 1.0, it indicates perfect precision and recall, and if it is 0, either precision or recall is zero.

- **Geometric Mean**: The Geometric Mean (G-Mean) is a metric that measures the balance between classification performances on both the majority and minority classes. A low G-Mean is an indication of a poor performance in the Classification of the positive cases even if the negative cases are correctly classified as such. This measure is important in the avoidance of overfitting the negative class and underfitting the positive class. The Geometric Mean can be computed as:

$$G - Mean = \sqrt{Sensitivity \times Specificity} \tag{4.36}$$

with:

  - **Sensitivity**: As mentioned already above, it measures the accuracy of positive cases(Anomalies). Sensitivity assesses the effectiveness of the classifier on the positive/minority class. It is computed as:

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.37}$$

  - **Specificity**: It is a.k.a. **TN-Rate** or **Selectivity** and it measures the accuracy of negative cases(Normal Behaviour Samples). Specificity assesses the classifier's effectiveness on the negative/majority class. It is computed as:

$$Specificity = \frac{TN}{TN + FP} \tag{4.38}$$

- **Discriminant Power (D-Power)**: It is another measure that summarizes sensitivity and specificity. The formula is given by:

$$D - Power = \frac{\sqrt{3}}{\pi}(\log X + \log Y) \tag{4.39}$$

where:

$$X = \frac{Sensitivity}{1 - Sensitivity} \tag{4.40}$$

and

$$Y = \frac{Specificity}{1 - Specificity} \tag{4.41}$$

The Discriminant Power assesses how well a classifier distinguishes between positive and negative cases. The classifier can be interpreted as follows:

$$D - Power_{Interpretation} = \begin{cases} Poor, & \text{if } D - Power < 1 \\ Limited, & \text{if } 1 \leq D - Power < 2 \\ Fair, & \text{if } 2 \leq D - Power < 3 \\ Good, & \text{if } D - Power \geq 3 \end{cases} \tag{4.42}$$

- **Adjusted F-Measure**: The F-Measure (or F1 Score) conveys the balance between precision and sensitivity. The measure is 0 when either the precision or the sensitivity is 0. The formula for this measure is given by Metrics(equation 4.33). This measure however performs well when the data is fairly balanced. The **Adjusted F-Measure (AGF)** is an improvement over the F-Measure

especially when the data is imbalanced. The AGF is computed by first computing:

$$F_2 = 5 \times \frac{Sensitivity \times Precision}{(4 \times Sensitivity + Precision} \tag{4.43}$$

After, the class labels of each case are switched such that positive cases become negative and vice versa. A new confusion matrix with respect to the original labels is created and the quantity:

$$InvF_{0.5} = \frac{5}{4} \times \frac{Sensitivity \times Precision}{(0.5^2 \times Sensitivity) + Precision} \tag{4.44}$$

The AGF is finally computed by taking the geometric mean of F_2 and $InvF_{0.5}$ as:

$$AGF = \sqrt{F_2 \times InvF_{0.5}} \tag{4.45}$$

This measure accounts for all elements of the original confusion matrix and provides more weight to patterns correctly classified in the minority class. A high F- or adjusted F-measure indicates a good-performing classifier in the minority class.

- **Balanced Accuracy**: It is the average between the sensitivity and the specificity, which measures the average accuracy obtained from both the minority and majority classes. This quantity reduces to the traditional accuracy if a classifier performs equally well on either classes. Conversely, if the high value of the traditional accuracy is due to the classifier taking advantage of the distribution of the majority class, then the balanced accuracy will decrease compared to the accuracy. Its formula is:

$$BalancedAccuracy = \frac{1}{2}(Sensitivity \times Specificity) \tag{4.46}$$

- **Matthew's Correlation Coefficient (MCC)**: It is least influenced by imbalanced data. It is a correlation coefficient between the observed and predicted classifications. the worst possible prediction. Its formula is:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.47}$$

The value that comes out from the computation has a $[-1, +1]$ range, where:

$$MCC_{Interpretation} = \begin{cases} PerfectPrediction, & \text{if } MCC = +1 \\ NoBetterthanRandom, & \text{if } MCC = 0 \\ WorstpossiblePrediction, & \text{if } MCC = -1 \end{cases} \tag{4.48}$$

- **Cohen's Kappa**: Kappa takes into account the accuracy that would be generated purely by chance. The form of the measure is:

$$Cohen'sKappa = \frac{TotalAccuracy - RandomAccuracy}{1 - RandomAccuracy} \tag{4.49}$$

where:

$$TotalAccuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.50}$$

and

$$RandomAccuracy = \frac{(TN+FP)(TN+FN)+(FN+TP)(FP+TP)}{(TP+TN+FP+FN)^2} \tag{4.51}$$

In a similar fashion to the MCC, kappa takes on values in the range $[-1,+1]$ with the value indicating the level of agreement between the actual and classified classes:

$$Cohen'sKappa_{Interpretation} = \begin{cases} PerfectConcordance, & \text{if } Cohen'sK = +1 \\ NoAgreement, & \text{if } Cohen'sK = 0 \\ TotalDisagreement, & \text{if } Cohen'sK = -1 \end{cases} \tag{4.52}$$

- **Youden's Index**: Youden's index evaluates the ability of a classifier to avoid misclassifications. This index puts equal weights on a classifier's performance in both positive and negative cases. Thus:

$$Youden'sIndex = Sensitivity - (1 - Specificity) \tag{4.53}$$

- **Accuracy**: The most commonly reported measure of a classifier is the accuracy. This measure evaluates the overall efficiency of an algorithm. However, as illustrated earlier, predictive accuracy can be a misleading evaluation measure when the data is imbalanced. This is because in such cases, more weights are placed on the majority class than on the minority class making it more difficult for a classifier to perform well on the minority class. The Accuracy can be computed as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.54}$$

- **TP-Rate**: The TP-Rate, also known as **Sensitivity**, **Recall**, or **Hit-Rate**, measures the proportion of positive instances correctly classified by the algorithm. It is computed in the following way:

$$TPRate = \frac{TP}{TP+FN} \tag{4.55}$$

4.7.2 Ensemble of Metrics: Model Selection Process

The selection of the ten diverse metrics in our ensemble has been driven by a strategic intention to comprehensively evaluate and determine the best algorithm for our R-T AD System. Each metric plays a crucial role in capturing distinct aspects of model performance, providing a multifaceted view that goes beyond traditional accuracy measures. Notably, the G-Mean, Balanced Accuracy, and TPR have been specifically chosen due to their demonstrated suitability for handling imbalanced datasets, as highlighted in the study [3].

In the **Ensemble of Metrics Decisions** stage, we have meticulously devised four distinct methods to consolidate the metric results and arrive at a final decision, thus enabling effective comparison of models on a level playing field. These ensemble approaches have been carefully selected to provide developers with a wide array of possibilities and the flexibility to cater to diverse customer necessities. Our aim is to empower developers to make informed decisions about the best model for the R-T AD System, considering the specific context and characteristics of the dataset at hand. The four proposed ensemble decision strategies are as follows:

- **Naïve Voting**: This straightforward approach facilitates model selection by prioritizing models based on their **Cumulative Rankings** across all metrics. The model with the highest number of 1st places is selected as the top choice, ensuring simplicity and ease of implementation. In case of ties, we will select the model with the highest number of 2nd places, as a tiebreaker. We continue in this way if the case of other ties.

  We can formalize this decision-making process by first defining the following terms:

  - Let M be the total number of models in the ensemble.
  - Let N be the total number of metrics used for evaluation.
  - Let R_i^m represent the ranking of model m for metric i, where $1 \leq R_i^m \leq M$.
  - Let V^m denote the final voting score for model m.

  The **Naïve Voting** method can be defined as follows:

  - Calculate the cumulative rankings for each model based on their performance on all metrics:

$$R_{\text{total}}^m = \sum_{i=1}^{N} R_i^m \quad \text{for each model } m \tag{4.56}$$

  - Select the model with the highest number of 1st places:

$$\text{Best}_{\text{M}(1st)} = \{m \mid R_1^m = 1\} \tag{4.57}$$

    The set $\text{Best}_{\text{M}(1st)}$ contains all models that received 1st place in at least one metric.
  - If there are ties, choose the model with the highest number of 2nd places as a tiebreaker:

$$\text{Best}_{\text{M}(2nd)} = \{m \mid R_1^m = 2\} \cap \{m \mid m \in \text{Best}_{\text{M}(1st)}\} \tag{4.58}$$

    The set $\text{Best}_{\text{M}(2nd)}$ contains all models that received a 2nd place in at least one metric among the models in $\text{Best}_{\text{M}(1st)}$.
  - Continue this process for lower rankings in case of further ties until a single model is selected.

  The model with the highest cumulative ranking is the final choice:

$$\text{Final Model} = \arg \max_{m \in \text{Best}_{\text{M}(1st)}} R_{\text{total}}^m \tag{4.59}$$

  This approach ensures a simple and interpretable model selection process by prioritizing models based on their collective performance across all metrics. In the case of ties, the tiebreaker mechanism ensures that a single model is ultimately chosen.

- **Weighted Voting**: To further fine-tune the decision-making process, we introduced another variation of voting, weighted voting, allowing developers to assign varying degrees of importance to different metric positions. By leveraging coefficients α, β, γ, ... (summarized in the vector ζ) one can emphasize the significance of certain metric rankings over others, adapting the decision process to the specific needs of the application. The process is very similar to before, except for some small variations:

  - **Define the Coefficients**: Let there be N ranks in total (e.g., 1st, 2nd, 3rd, ..., Nth place). For each rank i, assign a coefficient ζ_i that represents the weight for that rank. Developers can adjust these coefficients based on the relative significance of different metric positions for the specific application.
  - **Calculate the Weighted Voting Score for each model**: For each model m, compute the **Weighted Voting Score** as follows:

$$\text{WeightedVotingScore}^m = \sum_{i=1}^{N} \zeta_i \cdot \#\text{i-th place}^m \tag{4.60}$$

    Here, $\#\text{i-th place}^m$ represents the number of times model m received the i-th place across all metrics.
  - **Select the Final Model**: The model with the highest **Weighted Voting Score** is chosen as the final selection:

$$\text{Final Model} = \arg\max_m \text{WeightedVotingScore}^m \tag{4.61}$$

    The **Final Model** is the one with the highest weighted voting score, which effectively considers the prioritization of different metric rankings based on the assigned weights.

- **Naïve SumUp**: This approach efficiently aggregates metric results by computing their sum after applying MinMaxScaler normalization. The normalized metrics are then combined to yield a cumulative score for each model, providing a comprehensive perspective of overall performance. The steps more in detail are:

  - **MinMaxScaler Normalization**: Before summing up the metrics, we apply MinMaxScaler normalization to each metric value to bring them within a common range [0, 1]. By using MinMaxScaler normalization, it ensures that each metric contributes fairly to the final decision, regardless of its original scale. Let Metric_i represent the original value of the i-th metric, and Min(Metrics) and Max(Metrics) represent the minimum and maximum values among all metrics, respectively. The normalized value $\text{Metric}_i^{\text{norm}}$ for each metric is computed as follows:

$$\text{Metric}_i^{\text{norm}} = \frac{\text{Metric}_i - \text{Min(Metrics)}}{\text{Max(Metrics)} - \text{Min(Metrics)}} \tag{4.62}$$

  - **Calculate the Naïve SumUp Score for each model**: For each model m, compute the **Naïve SumUp Score** by summing up the normalized metric values:

$$\text{Naïve SumUp Score}^m = \sum_{i=1}^{N} \text{Metric}_i^{\text{norm}} \tag{4.63}$$

    Here, N is the number of metrics used in the ensemble.
  - **Select the final model**: The model with the highest **Naïve SumUp Score** is chosen as the final selection:

$$\text{Final Model} = \arg\max_m \text{Naïve SumUp Score}^m \tag{4.64}$$

    The **Final Model** is the one with the highest cumulative score, indicating superior overall performance when considering all metrics.

- **Weighted SumUp**: This approach recognizes the significance of context and data characteristics, offering a customizable method for developers to allocate weights (using the α vector of weights) to each metric based on its relevance and impact. This empowers developers to make informed decisions, resulting in the selection of the most suitable algorithm to meet the unique requirements of the R-T AD System. The detailed process provides:

  - **Metric Weight Allocation**: Developers can assign weights $\alpha = \alpha_1, \ldots, \alpha_n$ to each metric, reflecting their importance in the specific context of the application. These weights are determined based on domain knowledge, problem characteristics, or customer preferences.
  - **Calculate the Weighted SumUp Score for each model**: For each model m, compute the **Weighted SumUp Score** by summing up the products of each metric value with its corresponding weight:

$$\text{Weighted SumUp Score}^m = \sum_{i=1}^{N} \alpha_i \cdot \text{Metric}_i^{\text{norm}} \tag{4.65}$$

  - **Select the final model**: The model with the highest **Weighted SumUp Score** is chosen as the final selection:

$$\text{Final Model} = \arg\max_m \text{Weighted SumUp Score}^m \tag{4.66}$$

    The **Final Model** is the one with the highest cumulative score, considering the weighted contributions of each metric. This approach enables a tailored and context-aware model selection process, ensuring the R-T AD System meets the specific needs and requirements of the application.

After having selected one of these approaches, we will have a **Final Model Decision w.r.t. the Correctness** point of view.

4.7.3 Time Performances

In our comprehensive Model Selection Process, we also consider a critical factor: the **Computation Time** required by each method to analyze the data. The efficiency of an algorithm in predicting abnormal timestamps significantly influences the overall success of the approach. To measure the computation time accurately, we adopt a structured approach that distinguishes between two crucial times:

- **Training Time**: This time, a.k.a. **Design Time**, represents the duration required for the algorithm to learn and adapt to the training dataset, forming the foundation of its predictive capabilities. During this phase, the model adjusts its internal parameters and gains insights from the provided data, optimizing its performance for subsequent evaluations.
- **Evaluation Time**: This time, a.k.a **Run Time**, represents the measure of the algorithm's efficiency in processing new, unseen data and making real-time predictions. This operational phase is particularly crucial for time-sensitive applications, where quick responses to emerging anomalies are paramount.

By analyzing both Training Time and Evaluation Time, we gain a comprehensive understanding of the algorithm's temporal performance, empowering us to make

informed decisions about the most suitable approach. Throughout this thesis, we employ various methods tailored to the nature of individual algorithms to precisely measure the computation time. These assessments contribute to our quest for an efficient and effective AD system capable of meeting the dynamic demands of diverse real-world scenarios.

4.7.4 Time Performances: Model Selection Process

In our Model Selection Process, we carefully consider both Training Time and Evaluation Time to make informed decisions.

- **Best Model Selection at Design Time**: At Design Time, we aim to identify the best model that achieves optimal performance while minimizing the Training Time. Lower Training Time indicates faster model training and optimization, making it a favorable attribute. To determine the best model at Design Time, we calculate the DesTime score for each model m:

$$\text{BestModel}_{\text{DesTime}} = \arg\min_{m}(\text{DesTime}(m_1), \ldots, \text{DesTime}(m_n)) \tag{4.67}$$

- **Best Model Selection at Run Time**: At Run Time, we focus on the Evaluation Time of each model. The Evaluation Time represents the time required to process new data and make real-time predictions. Similar to the process at Design Time, we calculate the RunTime score for each model m:

$$\text{BestModel}_{\text{RunTime}} = \arg\min_{m}(\text{RunTime}(m_1), \ldots, \text{RunTime}(m_n)) \tag{4.68}$$

4.7.5 Resources Consumption

In our Workflow Decision Process, **Hardware Consumption** represents a crucial aspect that directly impacts the overall system performance, efficiency, and cost-effectiveness. We conduct a comprehensive study of three main hardware components: CPU, Memory, and Disk. The hardware consumption study plays a vital role in the Model Selection Process, as it enables us to make informed decisions about resource allocation, performance optimization, and capacity planning. By understanding how each component contributes to the overall system behavior, we can tailor our approach to meet specific customer needs and achieve efficient and effective R-T AD. Through a meticulous examination of CPU, Memory, and Disk consumption, we strive to deliver a robust and reliable solution that maximizes hardware utilization, minimizes resource contention, and ensures a smooth and efficient AD system.

CPU Analysis The CPU is a fundamental component responsible for executing computational tasks in the system. The CPU analysis we propose involves evaluating three main metrics, including:

1. **CPU Utilization(%)**: Monitoring the CPU utilization percentage provides insights into the workload and efficiency of the CPU. By assessing CPU utilization, we can identify periods of high or low activity and optimize task scheduling for improved system performance.

2. **CPU Frequency(%)**: Tracking the CPU frequency in MHz reveals the speed at which the CPU operates. Understanding the CPU frequency is essential for evaluating its capabilities and assessing the overall system performance.

3. **CPU Average Load(%)**: The average system load, represents the number of processes in a runnable state (i.e., using or waiting to use the CPU). By monitoring the average load, we gain insights into the workload distribution across CPU cores or threads.

The analysis of CPU consumption aids in identifying potential bottlenecks, detecting instances of CPU Throttling[7], and optimizing the allocation of computational resources.

Memory Analysis Memory is a critical resource that directly influences system performance and responsiveness. Our study of memory consumption includes the following metrics:

1. **Memory Utilization(%)**: Monitoring the memory utilization percentage allows us to assess how much of the available physical memory (RAM) is currently in use. Understanding memory utilization is crucial for efficient memory management and avoiding resource shortages.

2. **Memory Swaps(%)**: Memory swapping occurs when the system relies on swap space to compensate for limited physical memory. Monitoring memory swaps helps identify cases of Memory Over-Commitment[8] and assesses overall memory stress on the system.

The analysis of memory consumption provides valuable insights into memory usage patterns, workload demands, and potential performance bottlenecks.

Disk Analysis The Disk is a crucial component for data storage and retrieval. Our study of disk consumption focuses on **Disk Utilization(%)**, which measures the extent of data stored on the disk. As the slowest of the three hardware components, it is essential to minimize disk usage to ensure optimal system performance.

4.7.6 Resources Consumption: Model Selection Process

Within our Model Selection Process, a crucial step involves the evaluation of **Resource Consumption** for each model with respect to three key components: Memory, CPU, and Disk. We perform individual assessments(always by providing different choices to the developer in order to handle different customers' needs) for each component to determine the best algorithm, considering their efficiency and resource-friendliness. At the very end, then, we amalgamate these decisions and we come out with the Best Model w.r.t. Resources Consumption.

[7] **CPU's Throttling** occurs when the CPU reduces its operating frequency to manage heat or power consumption, resulting in decreased performance.

[8] **Memory Over-Commitment** occurs when the system allocates more memory than it has available, leading to excessive swapping and degraded performance

Memory Consumption Decision For Memory Consumption, we propose two decision choices:

- **Naïve SumUp**: It involves summing up the two percentage values: Memory Utilization and Memory Swaps.

$$\text{Naïve}_{\text{SumUp}} = \text{Memory}_{\text{Utilization(\%)}} + \text{Memory}_{\text{Swaps(\%)}} \tag{4.69}$$

- **Weighted SumUp**: It assigns weights (α and β) to the Memory Utilization and Memory Swaps percentages, respectively.

$$\text{Weighted}_{\text{SumUp}} = \alpha \cdot \text{Memory}_{\text{Utilization(\%)}} + \beta \cdot \text{Memory}_{\text{Swaps(\%)}} \tag{4.70}$$

CPU Consumption Decision For CPU Consumption, we introduce, two decision choices:

- **Naïve SumUp**: It involves summing up the three percentage values: CPU Utilization, CPU Frequency, and CPU Average Load.

$$\text{Naïve}_{\text{SumUp}} = \text{CPU}_{\text{Utilization(\%)}} + \text{CPU}_{\text{Frequency(\%)}} + \text{CPU}_{\text{AvgLoad(\%)}} \tag{4.71}$$

- **Weighted SumUp**: It assigns weights (α, β, and γ) to the CPU Utilization, CPU Frequency, and CPU Average Load percentages, respectively.

$$\text{Weighted}_{\text{SumUp}} = \alpha \cdot \text{CPU}_{\text{Utilization(\%)}} + \beta \cdot \text{CPU}_{\text{Frequency(\%)}} + \gamma \cdot \text{CPU}_{\text{AvgLoad(\%)}} \tag{4.72}$$

Disk Consumption Decision For Disk Consumption, we select the model with the lowest Disk Utilization Percentage.

Best Model Decision for Resources Consumption To ascertain the overall best model concerning Resources Consumption, we synthesize the individual decisions made for Memory, CPU, and Disk. The most favorable algorithm is identified by selecting the one that exhibits the lowest ratio among the decision scores. This ratio represents the algorithm with the most substantial disparity between its first and second-best values. By adopting this approach, we prioritize the algorithm that demonstrates the most significant advantage over the alternative contenders, ensuring the selection of an optimal solution that effectively balances resource utilization and high-performance anomaly detection.

More formally, we have that, given the three final decisions over Memory, Disk, and CPU respectively, denoted as Mem, Disk, and CPU, the **Final Decision w.r.t. Resources Consumption** is:

$$\text{Resources Decision} = \min\left(\frac{\text{Mem}_{\text{1stValue}}}{\text{Mem}_{\text{2ndValue}}}, \frac{\text{Disk}_{\text{1stValue}}}{\text{Disk}_{\text{2ndValue}}}, \frac{\text{CPU}_{\text{1stValue}}}{\text{CPU}_{\text{2ndValue}}}\right) \tag{4.73}$$

4.8 Best Model at Training Time and at Evaluation Time

We are interested, at the very end of the Model Selection Decision Workflow, to arrive at one decision at Training Time and one decision at Evaluation Time.

Final Decision: Training Time For the final decision for model selection at Training Time, we propose two distinct approaches:

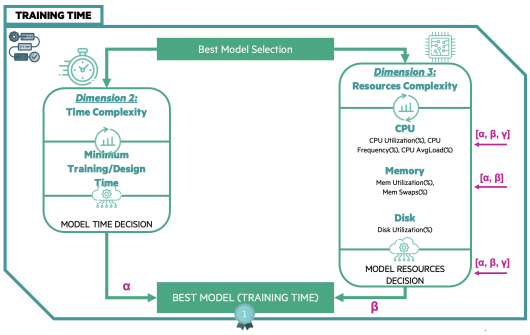


Figure 4.7. Training Time Decision

- **Naïve Voting**: The Naïve Voting method facilitates model selection by prioritizing the model that emerges as the best across most of the analyses conducted during the training phase. This straightforward approach offers simplicity and ease of implementation. The model with the highest cumulative ranking across Time Performances and Resources Consumption is chosen as the top choice.
- **Weighted Voting**: In the Weighted Voting approach, we introduce the ability to assign weights, represented by coefficients α and β, to the Time Performances and Hardware/Resource Performances, respectively. This customization empowers developers to emphasize the significance of specific analyses according to the context of the application. The formula for the Weighted Voting Score is defined as follows:

$$\text{Weighted}_{\text{Voting}}(\text{Training}) = \alpha \cdot \text{Time}_{\text{Design}} + \beta \cdot \text{Resources}_{\text{Design}} \tag{4.74}$$

Final Decision: Evaluation Time For the final decision for model selection at Evaluation Time, we propose two distinct approaches:

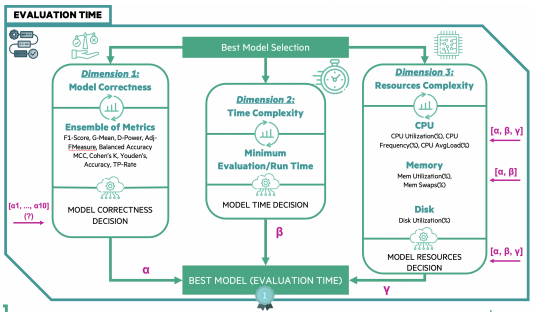


Figure 4.8. Evaluation Time Decision

- **Naïve Voting**: Similar to the Training Time approach, this method is employed to select the model that performs the best across most of the analyses conducted during Evaluation Time. The model with the highest cumulative ranking across Correctness, Time Performances, and Resources Consumption is chosen as the top choice.

- **Weighted Voting**: In the Weighted SumUp approach, we introduce the ability to assign weights, represented by coefficients α, β, and γ, to the Correctness, Time Performances, and Hardware/Resource Performances, respectively. This allows for a context-specific emphasis on particular analyses, ensuring a more informed and flexible decision-making process. The formula for the Weighted SumUp Score is defined as follows:

$$\text{Weighted}_{\text{Voting}}(\text{Evaluation}) = \alpha \cdot \text{Correctness}_{\text{Run}} + \beta \cdot \text{Time}_{\text{Run}} + \gamma \cdot \text{Resources}_{\text{Run}} \tag{4.75}$$

Final Decision: Overall To arrive at the Overall Final Decision, we consolidate both the Training Time and Evaluation Time decisions. As a **Pre-Processing Step**, we ensure that both decisions are of the same type, which is achieved by translating any Naïve Voting decisions to the corresponding **Weighted Voting** version. This step facilitates a unified approach for the final evaluation.

Once the decisions are in the Weighted Voting format, we combine them to form the overall final decision. In this process, we introduce coefficients α and β to assign weights to the Training Time decision ($\text{Decision}_{\text{Design}}$) and Evaluation Time decision ($\text{Decision}_{\text{Run}}$), respectively. These weights allow for a flexible and context-specific approach, where the importance of each decision can be adjusted according to the specific requirements of the R-T AD System.

The formula for the overall final decision is defined as follows:

$$\text{Final}_{\text{Decision}} = \alpha \cdot \text{Decision}_{\text{Design}} + \beta \cdot \text{Decision}_{\text{Run}} \tag{4.76}$$

4.9 Hyper-Parameters Configuration for Tailored AD Solution: Summary

After the crucial phase of **Customer's Requirements Gathering**, the **Developer** assumes the responsibility of manually setting the following hyper-parameters to tailor the AD solution precisely to the customer's needs. In summary, we have:

Decision Hyper-Parameters:

- **CPU Decision**: This hyper-parameter regulates the CPU Decision over CPU Utilization, CPU Frequency, and CPU Load, offering the choice between *Naive SumUp* and *Weighted SumUp* methods, along with the option to specify a custom weights vector in case the latter solution is chosen.
- **Memory Decision**: This hyper-parameter regulates the Memory Decision over Memory Utilization and Memory Swaps, offering the choice between *Naive SumUp* or *Weighted SumUp*, along with the option to specify custom weights vector in case the latter solution is chosen.
- **Ensemble of Metrics Choice**: This hyper-parameter presents a range of options, such as *Naive SumUp*, *Weighted SumUp* with custom weights, *Naive Voting*, and *Weighted Voting*.
- **Training Time Best Decision**: This hyper-parameter regulates the Training Time Choice between *Naive Voting* and *Weighted Voting*, along with the option to specify a custom weights vector in case the latter solution is chosen.

- **Evaluation Time Best Decision**: Similar to the previous one, this hyper-parameter regulates the Evaluation Time Choice, offering the same options: *Naive Voting* and *Weighted Voting*, along with the option to specify a custom weights vector in case the latter solution is chosen.
- **Final Decision Overall**: This hyper-parameter regulates the Final decision where, since the only option is *Weighted Voting*, we need to specify a custom weights vector that weights the Training and the Evaluation Time choice in the Final Overall Decision.

Model Hyper-Parameters:

- **MLP**: The developer has the flexibility to set MLP parameters such as the number of layers, block size, and maximum iterations for the Multilayer Perceptron Classifier.
- **LOG-REG**: The logistic regression model offers the *log_max_iter* hyper-parameter to control the maximum number of iterations.
- **GBT**: The Gradient Boosting Trees model allows the developer to fine-tune *max_depth*, *max_bins*, and *max_iter* for optimal performance.

Once these hyper-parameters are set, the AD system is ready to execute the code. The developer can sit back, relax, and enjoy a cup of coffee while waiting for the responses, knowing that the solution has been tailored to the specific requirements of the customer.

4.10 Workflow's Step 6: Model Explanation

After completing the Workflow's Step 5: Model Selection (Section 4.7), we embark on the final step of our comprehensive Research Workflow—the **Model's Feature Explanation**.

In this pivotal stage, we underscore the paramount importance of understanding the factors and features that drive the model's predictions. By shedding light on the reasoning behind the model's decisions, we bolster model validation, engender trust among end-users, and gain insights into potential biases or limitations within the system. Transparency and interpretability hold tremendous value in our domain, where providing transparent and trustworthy models is crucial. By offering explanations of how the model arrives at its decisions, we enable customers to grasp the rationale behind the predictions and take informed actions based on the model's outputs.

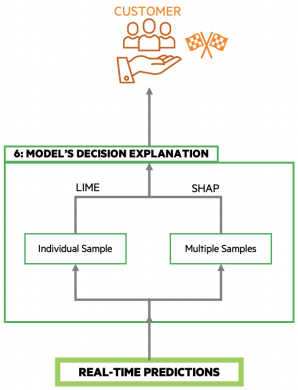


Figure 4.9. MAD Workflow: Model Explanation Step

To achieve a comprehensive Model's Feature Explanation, we have chosen to employ two state-of-the-art Explainability Techniques that have not been previously applied to this specific type of workflow, rendering this approach novel and groundbreaking. The selected techniques are **LIME (Local Interpretable Model-Agnostic Explanations)** [27] and **SHAP (SHapley Additive exPlanations)** [20].

Both SHAP and LIME offer unique **advantages** in interpreting complex models:

- **Model-Agnostic**: One of the key strengths of both techniques is that they are model-agnostic, meaning that they can be applied to any machine learning model, regardless of its underlying architecture or complexity.
- **Global and Local Insights**: SHAP and LIME offer distinct perspectives on feature importance. SHAP provides a global view of feature importance, helping identify the overall impact of features on the model's predictions across multiple samples. On the other hand, LIME offers local explanations, pinpointing feature importance at the instance level, allowing us to understand how individual features contribute to specific predictions.

4.10.1 SHAP (SHapley Additive exPlanations)

SHAP[20] values are rooted in cooperative game theory and provide a unified framework for feature importance analysis in machine learning models. The concept of Shapley values originates from the question of fairly distributing the "credit" or "contribution" among a group of players in a cooperative game. In our context, the "players" represent the features of the model, and SHAP values measure the impact of each feature on the model's predictions. These values represent the average marginal contribution of each feature when it is added to all possible coalitions of features. In other words, SHAP values quantify the change in prediction caused by including a particular feature in the model.

Kernel SHAP To achieve comprehensive feature explanation in our R-T AD System, we employ a particular version of SHAP known as **Kernel SHAP**. This method stands out for its scalability and computational efficiency, making it well-suited for analyzing complex ML models. The foundation of Kernel SHAP lies in approximating the Shapley values using a weighted sampling of feature subsets.

Let x be the input sample of interest for which we seek to interpret the model's prediction. Kernel SHAP calculates the contribution of each feature i to the prediction for x by comparing the model's predictions for subsets of features containing and excluding feature i. The Shapley value ϕ_i for feature i is expressed as the average of these differences across all possible subsets of features:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(f(S \cup \{i\}) - f(S)\right) \tag{4.77}$$

Where:

- N is the set of all features.
- S is a subset of features, excluding i.
- f is the model's prediction function.

The process involves computing these Shapley values for multiple individual predictions, allowing us to identify the contributions of each feature to specific sequences of model outputs. By quantifying the change in prediction caused by the inclusion of a particular feature, Kernel SHAP empowers us to accurately interpret the model's decision-making process.

4.10.2 LIME (Local Interpretable Model-agnostic Explanations)

LIME[27] takes a different approach by providing local, interpretable explanations for individual predictions of any black-box model. The idea behind LIME is to approximate the behavior of the complex model locally around a specific data point by training an interpretable, simple model on a subset of perturbed data around the instance of interest. LIME generates locally faithful explanations by perturbing the original data point and observing how the model's predictions change. The simple interpretable model is then fitted to approximate the black-box model's behavior within that local region.

In our Workflow, LIME serves as a crucial technique for **Local Explainability**, specifically tailored to provide insights into the features' contributions over individual samples. When it comes to explaining predictions for a single data instance, LIME excels in highlighting the relevance of each feature in driving the model's decision for that specific sample. By utilizing LIME for Local Explanation, we can indeed gain a granular understanding of how the model arrives at its predictions for specific data points.

4.10.3 Complementary Model Explanations: SHAP and LIME

In our pursuit of achieving a highly transparent and interpretable machine learning model, we recognize the importance of providing comprehensive and accurate explanations for its predictions. To this end, we have chosen to include both SHAP and LIME as part of our model interpretability toolkit. This decision was made to cater to different use cases and RunTime requirements, enabling us to offer a more tailored and versatile approach to explaining model predictions.

In our Workflow, SHAP assumes the role of a powerful **Global Explanation** technique, specifically suited for providing our customer with a deeper understanding of the contributions of features across multiple samples, such as those accumulated during the last day, by comprehensively assessing the overall impact of features on model predictions. While SHAP offers comprehensive explanations, it comes at a computational cost, especially when analyzing a large number of samples individually. In some R-T scenarios, we may need to **provide explanations quickly and efficiently for individual local samples**. For such cases, **LIME** presents an ideal solution. In our Workflow, LIME serves as a crucial technique for **Local Explainability**, specifically tailored to provide insights into the features' contributions over individual samples.

4.11 Workflow: Conclusions

In conclusion, the **MAD Workflow** presented in this thesis embodies a customer-centered approach that emphasizes the significance of understanding and meeting the unique requirements of each customer's 5G network data. In essence, the MAD Workflow is a full circle, starting and ending with the customer. As we progress through preprocessing, clustering, labeling, model building, and model selection, we continually prioritize the customer, empowering them to make data-driven decisions aligned with their needs. Ultimately, the workflow concludes by providing comprehensive model explanations, giving the customer full visibility into the inner workings of the model. Through this final step, we close the loop and go back to the customer, enabling them to not only trust the model but also fully comprehend the factors contributing to AD decisions.

Chapter 5

Experiments and Results

In this chapter, we present the experimental settings, a description of the datasets employed, and the results of our thesis, providing a comprehensive overview of the MAD Workflow application over one Customer's Dataset.

5.1 Experimental Setting

In this section, we delve into the experimental settings employed in our thesis. These settings lay the foundation for our research, enabling us to conduct rigorous experiments and obtain meaningful insights.

5.1.1 PC Settings

In the realm of machine learning, the training of algorithms is a resource-intensive task that demands careful consideration of various factors. Among these, the configuration of a computer system plays a pivotal role in determining the effectiveness and efficiency of the training process. The PC settings wield a profound influence on both the performance of machine learning algorithms and the consumption of valuable time and computational resources. We outline the PC specifications below.

For these experiments, we have been using an HP Elite-Book with Windows 10 OS. More in detail, the PC is a 64-bit Operating System, and it has the following characteristics:

- **Processor**: Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz to 2.30 GHz, x64-based.
  - **Physical Processors**: Quad-Core (4 processors).
  - **Logical Processors**: Octa-Core (8 processors). [1]
- **Cache**: The PC is provided with 3-level caches:
  - **L1 Cache**: 256 KB.
  - **L2 Cache**: 1 MB.
  - **L3 Cache**: 8 MB.

[1] Notice that the Logical Processors are the double of the Physical Processors (in our case), indicating how many of those cores are divided using the **Hyper-Threading** that provides the ability to have multiple instructions (threads) to be processed on each core simultaneously)

- **RAM**: RAM: 16,0 GB (15,8 GB usable) with 2667 MHz speed.

During both the Training Phase and the Real-Time Phase application, in order to have always the same settings for the Time and Resources Consumption Measurement/Monitoring, no other application was opened except for VSC. [2] that was executing the Code. The entire algorithms have been implemented in Python and PySpark, with the help of some external libraries.

5.1.2 Dataset: Description

In this subsection, we provide a comprehensive overview and description of the dataset utilized in our research.

The Dataset used in this thesis was obtained from one HPE's customer [3]. This customer is one of the largest telecommunications companies in Northern Europe and comprises a collection of Multivariate Time-Series data representing various metrics related to their business operations.

Table 5.1. Customer Dataset: Description The dataset consists of 814 cells (both eNodeB and gNodeB, i.e., both 4G and 5G cells respectively). It contains 51 metrics (dimensions, features, or Dataset columns) capturing different aspects of the customer's operations. The data was collected by HPE's Intelligent Assurance internal monitoring system, which captures measurements of customers' RANs at regular intervals. The dataset covers a specific time period, from 16th December 2021 to 4th March 2022, taken with a granularity of 15 minutes, and is unlabeled.

# Cells	# Metrics	Period	Granularity	#Samples
814	51	16/12/2021 - 4/03/2022	15 minutes	3101856

5.2 Step 0: Data Loading

In the initial phase of the project, our primary objective was to efficiently **Load the Data**. In particular, we Load the Data from 24 different CSV files, and, as first thing, we create a unique Dataset for them. After this initial loading step, we have the Customer Dataset to be like in the table 5.2.

5.2.1 Covariance Matrix Visualization

Understanding the relationships and dependencies between variables is crucial in data analysis. One common approach to assess these relationships is by examining the **Covariance Matrix**, which provides insights into the co-variability between pairs of variables. In our general analysis, we want to spot any strong correlations. We carry out this step for several Cells in order to gain a better understanding of the dataset.

As a first step, we compute the covariance matrix, which measures the extent to which each variable in the dataset varies with others. By examining the elements

[2] Visual Studio Code (or more simply VS Code) is a source code editor developed by Microsoft for Windows, Linux, and Mac.

[3] **Usage and Ethical Considerations**: The dataset was used in this thesis to analyze the customer's business operations, identify patterns, and derive insights for decision-making purposes. To maintain the confidentiality and privacy of the customer's sensitive information, all personally identifiable data was anonymized or removed from the dataset. The usage of the dataset strictly adhered to ethical guidelines and data protection regulations.

Table 5.2. Customer Dataset First-Lines[4]: The first lines of this Dataset show how the data are loaded. There are actually 52 columns (eCell name + 51 features, with each feature representing some important metrics for the customers), and all the cells are within one gigantic Dataset comprising more than 3 million data samples rows.

eCell	Feature_1_KPI	Feature_2_KPI	...	Feature_51_KPI
eNode-1-...-LB11	15120.0	10176.0	...	48959840.0
eNode-2-...-LB11	0.0	0.0	...	6465952.0
gNode-1-...-LB11	5911.0	5432.0	...	23502696.0
gNode-1-...-LB11	0.0	0.0	...	15725888.0
...	...	...	...	...
eNode-178-...-LB11	15120.0	10176.0	...	17009752.0

of the covariance matrix, we can identify variables that tend to move together or in opposite directions. Strong positive or negative values in the covariance matrix indicate a significant relationship between the corresponding variables. Next, we analyze the **Correlation Matrix**, which is derived from the covariance matrix by standardizing the values. The correlation matrix provides a measure of the linear relationship between variables, ranging from -1 (strong negative correlation) to 1 (strong positive correlation).

One example of the Correlation Matrix is shown in figure 5.1.

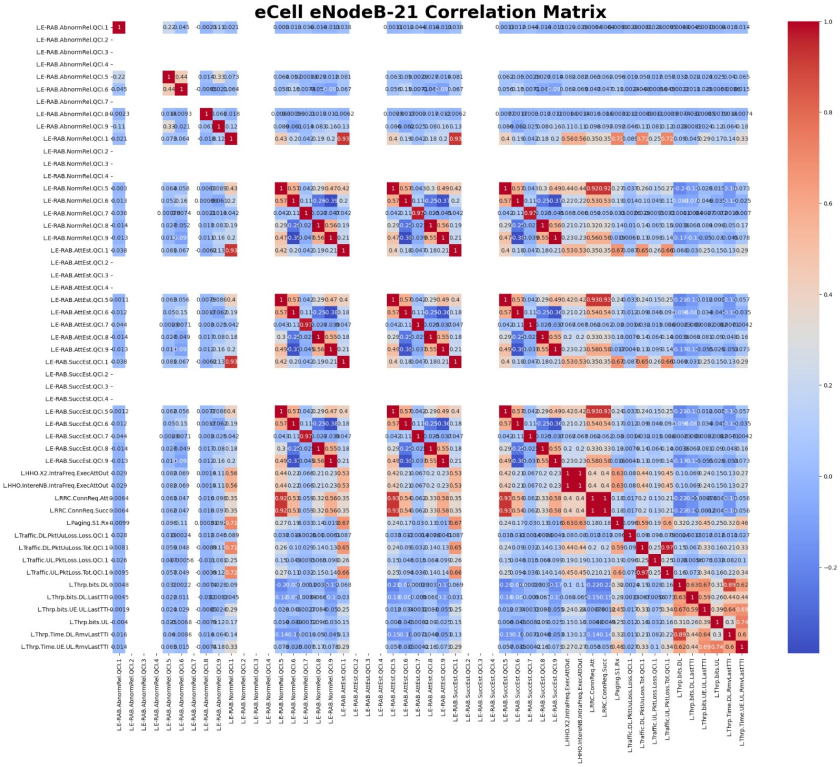


Figure 5.1. eNodeB-21 Correlation Matrix

Upon the above analysis, which repeats for most of the Cells, it becomes apparent that the features within each cell exhibit a concerning characteristic: they are all **Weakly Correlated**. This lack of correlation poses further complications in the problem.

5.3 Step 1: Dataset Preprocessing

Following the Loading phase, we focus on the **Data Cleaning**, for ensuring the quality and integrity of the dataset, by removing lots of the corrupted cells (representing the 0.68% of the entire Customer Dataset), that could potentially introduce errors or bias into our analysis. The first thing that we do is to group the big Dataset by Cells, namely, by eNodeB, by using PySpark's very powerful aggregation tools. Reasoning Cell-by-Cell, i.e., grouping the data samples in the dataset into subsets, we perform a series of steps to ensure data quality and facilitate subsequent analysis. We go through the following substeps: Missing Values Management(section 4.3.1) and Data Standardization(section 4.3.1).

Once we finish with this step, we have the Dataset that is ready for accurate and reliable analysis. By following these steps, we can effectively preprocess the data and address any issues that may arise.

5.4 Workflow's Step 2: Clustering

After the Pre-Processing phase, there is the actual **Clustering Phase**, as explained in Workflow's Step 2: Clustering(section 4.4).

During this phase, we explored various approaches. Initially, we implemented the Naive K-Means algorithm, but in the end, we decided to adopt the Agglomerative Clustering solution.

5.4.1 Agglomerative Clustering

After having recognized the limitations of K-Means clustering, we explored alternative clustering techniques and decided to employ **Agglomerative Clustering**.

After the experiments we conducted, Agglomerative Clustering proved to be a more effective approach, resulting in improved F1 scores (and in general improved overall performances), and a better representation of the underlying structure within the dataset. The enhanced performance of Agglomerative Clustering can be attributed to:

- **Handling Non-Convex Clusters**: K-means clustering assumes that clusters are **Convex and Isotropic**[5]. However, in real-world datasets, clusters have complex shapes and non-convex structures (we have 51 dimensions and the space could potentially have an unimaginable structure). Agglomerative Clustering, being a hierarchical clustering algorithm, is more flexible in capturing such complex cluster shapes and can adapt to various cluster geometries.

- **Outlier Sensitivity**: K-means clustering is sensitive to outliers, as an outlier can significantly impact the position of cluster centers. Agglomerative Clustering, with its linkage-based approach, is more robust to outliers as it considers the pairwise distances between all data points. In our Customer Dataset, it's

[5] A **Convex Cluster** refers to a cluster in which all points within the cluster are connected in a continuous and unbroken manner. In other words, if we were to draw a line between any two points within the cluster, the line would always remain within the cluster boundaries. This implies that the cluster has a smooth and convex shape. An **Isotropic Cluster**, on the other hand, refers to a cluster in which the distribution of points within the cluster is symmetric and does not exhibit any preferential orientation. In other words, the cluster does not have any inherent bias or direction. The points are evenly distributed in all directions, and there is no dominant axis or direction within the cluster.

very likely that, given the huge amount of data samples and dimensions we have, we could have also some outliers.

The utilization of Agglomerative Clustering overcomes the limitations experienced with K-means Clustering and set the foundation for further analysis and modeling within our workflow.

To enhance the performance of the Agglomerative Clustering algorithm, several experimental changes were implemented, focusing on the following key aspects:

- **n_clusters**: The impact of the number of clusters on the algorithm's performance was investigated. However, increasing the number of clusters did not yield notable improvements.
- **metric**: Initially, the algorithm employed the default metric, which is the Euclidean distance. However, after conducting experiments using alternative distance metrics such as L1, L2, Manhattan, and Cosine, it was observed that the **Euclidean Metric** consistently performed better in terms of validation results. This can be attributed to several factors. Firstly, the nature of the dataset and the underlying characteristics of the cells' behavior lend themselves well to the Euclidean distance metric. The Euclidean distance measures the straight-line distance between two points in a multidimensional space, which aligns with the intuitive notion of proximity and similarity between cells. Furthermore, the Euclidean metric is sensitive to the magnitudes and variances of the features, allowing for effective discrimination between different cell behaviors. In contrast, other metrics such as L1 (Manhattan) distance, L2 (Euclidean) distance, Manhattan distance, and Cosine distance either failed to capture the subtle nuances in cell behavior or introduced unnecessary biases that adversely affected the clustering performance. Therefore, based on these considerations and the superior performance demonstrated on the validation set, the decision was made to retain the Euclidean distance as the preferred metric for the Agglomerative Clustering algorithm.
- **linkage**: The linkage criterion, which determines the distance measure between sets of observations during the clustering process, was investigated. Initially, the algorithm used the default Ward Linkage, aiming to minimize the variance of the clusters being merged. Subsequently, the Average Linkage criterion, which calculates the average distance between observations of two sets, was explored. However, after careful evaluation, it was determined that **Complete Linkage** (also known as **Maximum Linkage**) was the most suitable choice. This criterion considers the maximum distance between any pair of observations from the two sets being merged. The selection of Complete Linkage was motivated by its ability to capture more diverse and distinct patterns within the clusters.

These experimental changes, combined with careful consideration of the selected hyper-parameter settings, allowed for the optimization of the Agglomerative Clustering algorithm's performance. At the end of the **Hyper-parameters Tuning Step** for Agglomerative Clustering, we individuated the final settings as described in table 5.3.

Table 5.3. Customer Dataset Agglomerative Clustering Algorithm: Cluster's Hyper-parameters

Hyper-Parameter	Value
n_clusters	4
metric	euclidean
compute_full_tree	auto
linkage	complete

After the application of the Agglomerative Clustering on the Customer Dataset, we obtained the results shown in figures 5.2 and 5.3.

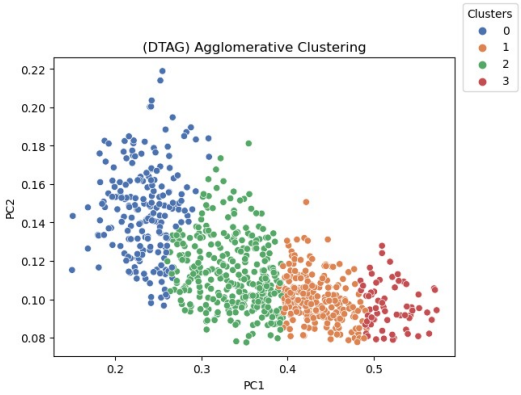


Figure 5.2. (Customer Dataset) Workflow's Step 1 (Clustering): Agglomerative Algorithm Application (2D Visualization)

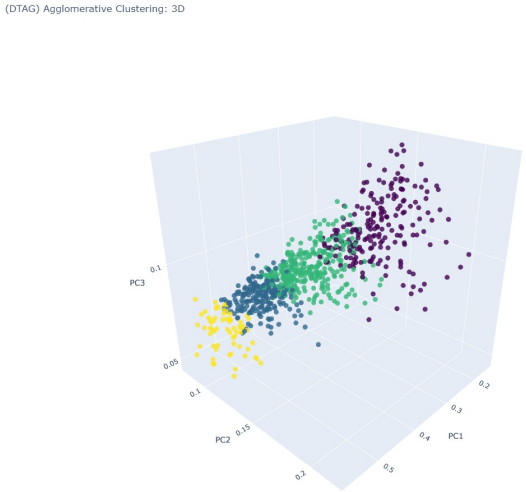


Figure 5.3. (Customer Dataset) Workflow's Step 1 (Clustering): Agglomerative Algorithm Application (3D Visualization)

Upon examining the table (5.5), it is evident that the clusters have varying numbers of cells. However, this discrepancy in cluster sizes does not negatively

impact the overall performance of the clustering algorithm. In fact, it can be argued that this variation in cluster sizes actually strengthens the clustering results. One notable observation that we can note in the figures below is that Cluster2 contains a significantly higher number of cells compared to Cluster3. Specifically, Cluster2 comprises 321 cells, whereas Cluster3 consists of only 66 cells. While this uneven distribution may initially seem imbalanced, it can provide valuable insights into the underlying patterns and behaviors of the cells. The larger size of Cluster2 suggests that there is a distinct subset of cells exhibiting similar characteristics or behaviors that are more prevalent in the dataset. This cluster represents a significant portion of the cells and indicates a common pattern or behavior shared by a substantial number of cells. On the other hand, the smaller size of Cluster3 suggests that there is a subset of cells exhibiting unique or rare characteristics that are less common within the dataset. By identifying these imbalances in cluster sizes, the Agglomerative clustering algorithm effectively highlights the heterogeneity within the dataset. It allows for a more nuanced understanding of the different cell behaviors and their distribution.

Furthermore, the varying cluster sizes do not compromise the predictive capabilities of the trained models. Each cluster is still capable of capturing the distinctive characteristics of the cells it represents, regardless of the absolute number of cells within the cluster. The focus of the clustering algorithm is to group together cells with similar behaviors, irrespective of the size of the resulting clusters. Therefore, the presence of clusters with different numbers of cells in the Agglomerative clustering results does not diminish the algorithm's performance. Instead, it provides valuable information about the distribution and prevalence of specific cell behaviors within the dataset, ultimately enhancing the effectiveness of subsequent analysis and modeling tasks.

Table 5.4. Customer Dataset Agglomerative Algorithm Cluster's Distribution: Number of Cells for each Cluster

TotalCells	Cluster0Cells	Cluster1Cells	Cluster2Cells	Cluster3Cells
814	200	227	321	66

5.5 Workflow's Step 3: Labeling

After the Clustering phase, there is the **Labeling Phase**, as explained in Workflow's Step 3: Labeling(section 4.5). This phase involves the labeling of data samples as either **Anomalous** or **Normal Behavior**.

During this phase, we develop an Emsemble using 7 different Algorithms consisting of seven distinct algorithms (K-Means, DBSCAN, GMM, Isolation Forest, Local Outlier Factor, Robust Covariance, and One-Class SVM) that will label the data samples in Anomaly/Non-Anomaly points. Successively, the final decision is employed by using a Majority Voting over the algorithms (≥ 4).

We start the Ensemble by first setting the **Contamination** Factor, denoted as ν, as explained in Contamination Factor Estimation: Transfer Learning (paragraph 4.5.2). Consequently, we proceed to undergo a set of tries to determine the best **Number of Clusters**, that will be used only in the K-Means application since the other algorithms either determine this value by themselves or do not need them.

Table 5.5. Labeling Step: Contamination Factor & Number of Clusters In the Customer Dataset's labeling step, we employ **Transfer Learning** to assign a **Contamination Factor** of 0.02 to our dataset. The contamination factor value of 0.02 is derived from a previous estimation conducted on a related dataset where the ratio of the anomaly to non-anomalous samples is expected to be similar. By setting the contamination factor to 0.02, we consider approximately 2% of the samples in our dataset to be anomalies, while the remaining 98% is considered normal data. This choice allows us to focus on identifying and analyzing the anomalous instances within the dataset. Regarding the **Number of Clusters**, we performed a tuning step to determine the optimal value based on the performance of the anomaly detection algorithms. After careful evaluation, we determined the number of clusters to be 15.

Contamination Factor	Number of Clusters
0.02	15

After the determination of these two fundamental parameters, we can start applying the 7 algorithms of the Ensemble:

- **K-Means**: Here, we make all the steps as explained in paragraph 4.5.1, making use of the K-Means settings described in table 5.6.

Table 5.6. Ensemble: K-Means Parameters Setting

Parameter	Value
n_clusters	15
init	k-means++
n_init	10
max_iter	300
tol	1e-4
verbose	0
random_state	1
algorithm	elkan

- **DBSCAN**: For this algorithm, we make all the steps as explained in paragraph 4.5.1. In order to find the appropriate value for the ϵ parameter in the DBSCAN algorithm, we need to go over an iterative step. The range of epsilon values to iterate over is defined from eps_{start} to eps_{end}, with a step size of eps_{step}. The parameter $Neighbours$ is set to $2 \times$ Columns, as indicated in table 5.7. For each epsilon value, we instantiate the DBSCAN algorithm with the selected epsilon value and fit it to the scaled dataset. To determine the number of anomalies in the dataset, we count the occurrences of -1 in the cluster labels. We then compute the percentage of anomalies by dividing the number of anomalies by the total number of data points, N. Subsequently, we compare the calculated percentage of anomalies to the desired contamination factor. If the calculated percentage is lower than the desired contamination factor, we update the final epsilon value as the current epsilon minus 0.2 and terminate the iteration. The ϵ **Selection Process** is shown in figure 5.4 as an example.

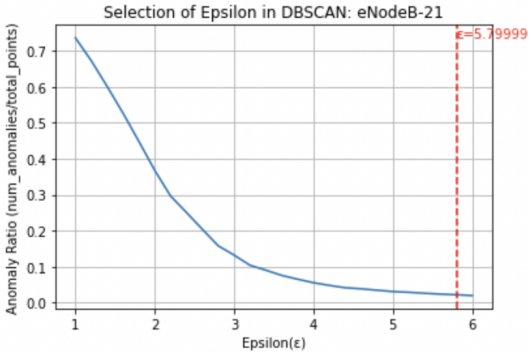


Figure 5.4. DBSCAN: Epsilon(ϵ) Value Selection: eNodeB-21

Finally, we instantiate the DBSCAN algorithm once again, this time using the final epsilon value that satisfies the desired contamination factor, with the parameters as table 5.7 describes.

Table 5.7. Ensemble: DBSCAN Parameters Setting

Parameter	Value
eps	final_eps
min_samples	5
metric	euclidean
metric	102
eps_start	1
eps_end	Neighbours/2
eps_step	2

- **GMM**: For the GMM, we follow the steps as explained in paragraph 4.5.1. After we instantiate a GMM object with the parameters specified in table 5.8, we fit the GMM model to the scaled dataset and it assigns cluster labels to the data points. After that, we calculate the anomaly score for each data point. We define a threshold based on the desired contamination factor specified in table 5.5 by calculating the percentile of the anomaly scores.

Table 5.8. Ensemble: GMM Parameters Setting.

Parameter	Value
n_components	1
covariance_type	full
tol	1e-3
reg_covar	1e-6
max_iter	100
n_init	eps(ϵ)
init_params	k-means++

- **Isolation Forest**: Also for this algorithm, we follow the steps as explained in paragraph 4.5.1. In order to include the Isolation Forest algorithm as part of the ensemble, we begin by creating an instance of the Isolation Forest with the specified parameters, as detailed in table 5.9. By fitting the Isolation Forest

model to the data, we train it to learn the underlying patterns and structures of the normal instances. Subsequently, we use the trained model to predict the anomaly labels for the data.

Table 5.9. Ensemble: Isolation Forest Parameters Setting. The parameter selection for the Isolation Forest algorithm specifically involved the *n_estimators* parameter, which determines the number of base estimators in the ensemble. This was carefully evaluated through a series of experiments. Our goal was to identify the optimal value that maximizes the algorithm's performance for our specific task while considering the trade-off between computational cost and accuracy. After conducting extensive evaluations, we determined that setting the number of estimators to 1000 yielded the best results. This selection strikes a balance between computational efficiency and the ability to effectively detect anomalies in the dataset, providing a robust solution for our anomaly detection needs.

Parameter	Value
n_estimators	1000
max_samples	auto
contamination	0.02
max_features	1

- **Local Outlier Factor**: For the LOF algorithm we go after the paragraph 4.5.1 indications. The LOF algorithm is instantiated with the parameters outlined in Table 5.10. Subsequently, the algorithm is fitted to the preprocessed and scaled dataset.

Table 5.10. Ensemble: Local Outlier Factor Parameters Setting.

Parameter	Value
n_neighbors	Neighbours
algorithm	auto
contamination	0.02
leaf_size	30
metric	minkowski

- **Elliptical Envelope**: The Robust Covariance (RC) algorithm is utilized as paragraph 4.5.1 explains. The RC algorithm is instantiated with the specified parameters, as described in Table 5.11. Following the instantiation, the algorithm is fitted to the preprocessed and scaled dataset.

Table 5.11. Ensemble: Elliptical Envelope Parameters Setting.

Parameter	Value
store_precision	True
support_fraction	None
contamination	0.02

- **One Class SVM**: The One-Class Support Vector Machine (SVM) algorithm is employed as paragraph 4.5.1 explains. The One-Class SVM algorithm is instantiated with the specified parameters, as described in Table 5.12. Once instantiated, the algorithm is fitted to the preprocessed and scaled dataset using the fit method.

Table 5.12. Ensemble: Elliptical Envelope Parameters Setting.

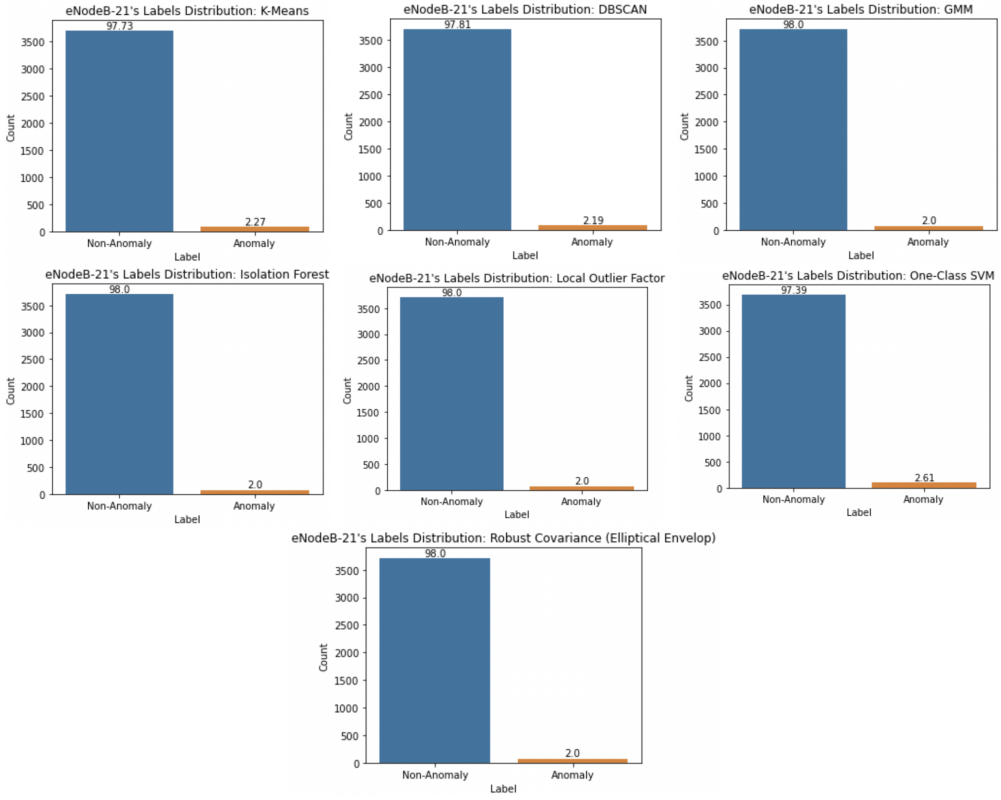
Parameter	Value
kernel	rbf
degree	3
nu	0.02
gamma	auto
shrinking	true

5.5.1 Ensemble Method: Majority Voting and Results

In order to combine the results of the seven anomaly detection algorithms used in the workflow, a **Majority Voting** approach is employed. To determine the final anomaly label for each data point, a threshold is set. In this case, the threshold is set at four ($\mathbf{t \geq 4}$) or more algorithms detecting an anomaly.

Hereinafter, we present, the Algorithm-by-Algorithm Distribution over the two cells taken as examples.

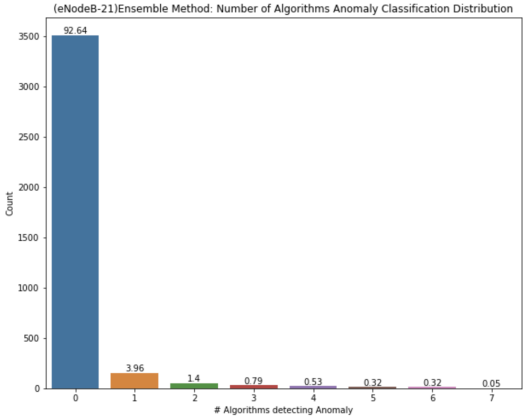
Figure 5.5. Ensemble Algorithm-by-Algorithm Distribution over eNodeB-21: The figure displays the distributions of "anomalies" and "non-anomalies" labels generated by the seven different anomaly detection algorithms employed in the Ensemble. Approximately 2% of the data points are labeled as anomalies, while the remaining 98% are classified as non-anomalies. This consistent proportion reinforces the notion that the algorithms are effective in identifying anomalies within the dataset.



The ensemble method utilizing majority voting enhances the accuracy and robustness of the AD process. By considering the collective decisions of multiple

algorithms, the ensemble method reduces the influence of individual algorithm biases and provides a more reliable identification of anomalies, as described in figure 5.6.

Figure 5.6. (eNodeB-21) Ensemble Method: Number of Algorithms (0-to-7) Anomaly Classification: The graph in the first figure, is a histogram with the x-axis representing the number of algorithms that classify samples as anomalies, ranging from 0 to 7. The y-axis represents the frequency or count of occurrences for each number of algorithms. The histogram bars are plotted with heights corresponding to the frequency of each number of algorithms. As we move from the bar representing 0 algorithms to the bar representing 7 algorithms, the height of the bars generally decreases, indicating a decrease in the number of samples classified as anomalies by that specific number of algorithms. This trend suggests that as more algorithms classify a sample as an anomaly, the stronger the indication that it is indeed an anomaly. Conversely, when fewer algorithms classify a sample as an anomaly, there may be more uncertainty or disagreement among the algorithms regarding its classification.



In conclusion of this step, the ensemble method comprising seven different anomaly detection algorithms has been successfully applied to the dataset, resulting in a labeled dataset that accurately distinguishes between anomalies and non-anomalies. This labeling process marks an important milestone as we now possess a robust foundation for building a real-time anomaly detection model, that will be applied over the customer's new incoming data.

5.5.2 Dataset Imbalance

Upon labeling the Customer Dataset for anomaly detection, we confirmed a characteristic that we had anticipated and is commonly observed in the anomaly detection field: the dataset exhibits **Highly Imbalanced** class distribution. In this context, the majority of data points correspond to normal instances (i.e., non-anomalous samples), while only a small fraction represents anomalies.

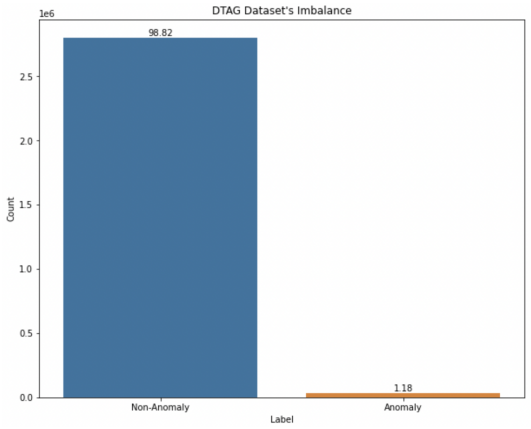


Figure 5.7. Customer Dataset's High Imbalance: The dataset exhibits a significant class imbalance, with approximately 98.82% of the samples labeled as non-anomalous and only 1.18% representing Anomalies. This imbalance poses a challenge for anomaly detection algorithms, as they are often biased towards the majority class and may struggle to accurately detect anomalies within the minority class.

The presence of such class imbalance reinforces the need for careful handling and consideration during model development and evaluation.

5.6 Workflow's Step 4: Real-Time Model Building

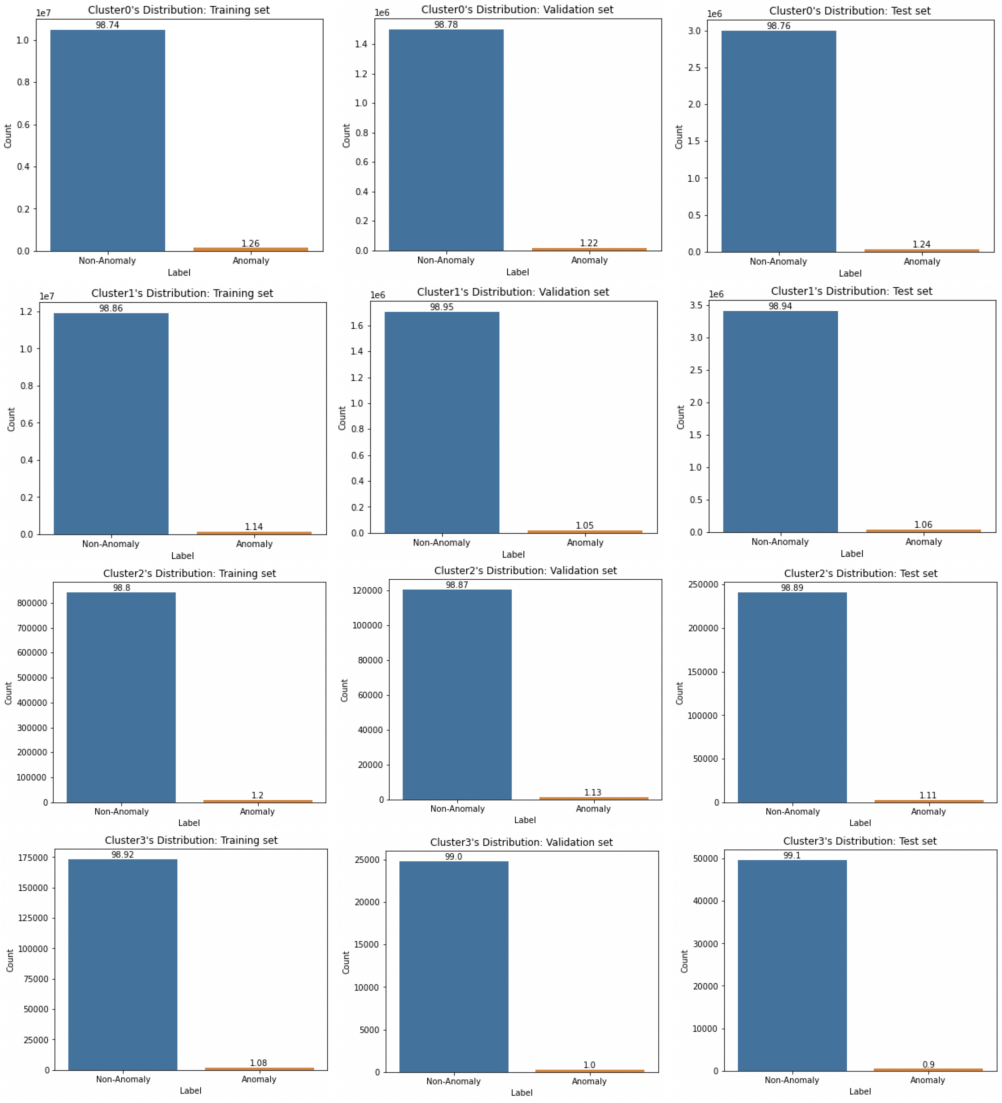
After the Labeling phase, there is the **RT Model Building Phase**, described in Workflow's Step 4: Real-Time Model Building, section 4.6). We now move on to the core part of our workflow, which is the **(RT) Real-Time Model Building** phase. This phase is of utmost importance as it involves the development of a robust model that can effectively detect anomalies in real time.

It is important to note that throughout the R-T model-building phase, we applied the models **cluster-by-cluster**. This means that we built a separate model for each of the four clusters identified during the labeling phase. By modeling each cluster individually, we were able to capture the unique characteristics and patterns within each cluster, improving the overall accuracy and effectiveness of our AD system.

5.6.1 Stratified Random Sampling: Cluster-wise Train-Test-Validation Split

Following the labeling step, we obtained four distinct clusters, each containing labeled data. In order to proceed with the R-T Model Building, it becomes imperative to ensure that the distribution of anomalies and non-anomalies within each cluster is accurately preserved. To achieve this, we employ **Stratified Random Sampling**, performed to maintain the same distribution of non-anomalies and anomalies **within each cluster's Train-Test-Validation Split**. This step is crucial due to the inherent imbalance in the dataset, with anomalies being present in a significantly smaller proportion. A simple random sampling approach, as employed at the very start of the experiment setting, has led us to an inadequate representation of anomalies in cases where we had a complete absence of anomalies in certain sets (in particular, in the Validation Set, that is the smallest one).

Figure 5.8. Stratified Random Sampling: Cluster-wise Distribution (Train-Validation-Test): The following figures depict the outcome of the Stratified Random Sampling technique applied to the labeled data. The figures clearly demonstrate that the stratified random sampling process successfully maintains the desired distribution, ensuring a representative and balanced distribution of data for training, validation, and testing purposes.



5.6.2 Real-Time Anomaly Detection: Model's Fine Tuning

In the pursuit of finding the most effective solution for AD, I conducted a thorough exploration and experimentation with the **Logistic Regression** model. This initial choice was driven by the model's interpretability and simplicity, aligning well with the research objectives. In the first phase, we implemented a simple version of Logistic Regression, fine-tuning its parameters to achieve a satisfactory F1 Score. However, we encountered the challenge of dataset imbalance, which prompted us to investigate solutions like Class Weighting and Threshold Moving. While Class Weighting did not yield desired results due to overcompensation, we found success with the Threshold Moving technique. This method involved dynamically

adjusting the classification threshold, resulting in significant improvements in overall performance and F1 Score.

Continuing the exploration of effective models for AD, the research process advanced to assess the performance of three additional algorithms: **Multilayer Perceptron Classifier (MLP)**, **Random Forest**, and **Stochastic Gradient Boosting (SGB)**. Each of these models underwent iterative testing and fine-tuning to optimize their performance. For MLP, we experimented with various hyperparameters, activation functions, and layer configurations to find the best combination that yielded favorable results. The Random Forest algorithm was fine-tuned by adjusting the number of trees and the depth of each tree in the ensemble. Similarly, the SGB model underwent extensive parameter tuning to achieve optimal performance.

Table 5.13. (Customer Dataset's Cluster 2) Models' Hyper-Parameters Fine-Tuning Step: Final F1 Score Results. In this table, we present the result obtained over the Customer Dataset's Cluster2 (Validation Set), by comparing the Model's versions developed.

Model	(Validation)F1-Score
(Basic) Logistic Regression	0.64359
(Class Weighted) Logistic Regression	0.33978
(Threshold Moving) Logistic Regression	0.73161
(Best) Random Forest	0.57271
(Best) Stochastic Gradient Boosting)	0.72861
(Best) Multilayer Perceptron Classifier)	0.73574

Table 5.14. (Customer Dataset's Cluster 2) Models' Hyper-Parameters Final Setting. In these tables, we present the final Hyper-Parameters setting of the best models, obtained over Customer Dataset's Cluster2, in terms of F1 Score, obtained in the previous table (5.13). Note that the other parameters not present, are set to their default value.

LOG	Value
maxIter	10
threshold	0.26307

GBT	Value
maxIter	200
maxBins	1500
maxDepth	4
stepSize	0.1

MLP	Value
layers	[51, 51, 2]
maxIter	200
blockSize	128
seed	1
solver	l-bfgs
tol	1e-06
stepSize	0.03

In conclusion of this step, we can say that, after an extensive exploration and evaluation of various machine learning and deep learning models, we have identified the top-three-performing algorithms that outperform all others (in the Customer Dataset's Cluster2 Data).

5.7 Workflow's Step 5: Model Selection

After the Labeling phase, there is the **Best Model Selection Phase**, described in Workflow's Step 5: Model Selection, section 4.7). In this step, we will subject the **Top-Three-Performing** Algorithms to the Model Selection Workflow Decision

Process. This process will comprehensively evaluate these algorithms, together with several Ensemble versions of them, based on Correctness(Ensemble of Metrics), Time Performance, and Resource Consumption. By fine-tuning the decision choices and weights in this workflow, we aim to identify the best algorithm overall—one that excels in all aspects. This rigorous selection process will ensure that the chosen algorithm aligns optimally with the specific requirements and constraints of our customer's Real-Time AD System, providing a robust and reliable solution for AD in 5G networks.

5.7.1 Customer's Requirements Gathering

The Workflow Decision process is carefully designed to be applicable to various customers, necessitating the fine-tuning of hyper-parameter weights to cater to each customer's unique requirements and preferences. During a comprehensive meeting with the Customer, we acquired invaluable insights into their priorities and concerns related to AD in 5G networks.

First and foremost, for the Customer, the distinction between Training Time and Evaluation Time holds paramount importance, with a pronounced focus on the latter performance, where timely and accurate AD predictions during RunTime are of utmost significance to facilitate prompt decision-making. While hardware is crucial during Design Time, it assumes a secondary position compared to the precision of predictions. The Customer places central importance on achieving high correctness and minimizing False Negatives (undetected anomalies). They are open to accepting more False Alarms, as these can be managed efficiently by their Orchestrator.

At Training Time, the Customer expresses greater concern about reducing hardware consumption, even if it leads to a longer design process. They are willing to trade off some design time efficiency to ensure the hardware is less utilized and subjected to reduced stress.

For the Customer, in both two phases, the Hardware Consumption is focused on the CPU and Memory. They prioritize maintaining low CPU utilization and load to manage expenses effectively, while simultaneously ensuring memory utilization remains under control. Disk utilization passes in the secondary plan, but it's important always to limit access to it, for avoiding slowing down the process.

Armed with a comprehensive understanding of the Customer's specific requirements, we can adeptly customize the Model Selection Workflow to precisely evaluate the decision factors. Consequently, we can translate these collated requirements into the necessary weight choices essential for the decision-making process, as shown in table 5.15.

Table 5.15. (Customer Dataset's Cluster 2) Decision Hyper-Parameters: In this table, we present the Decision Hyper-parameters that we set in the Decision Workflow Step, corresponding to the Customer Requirements.

Hyper-Parameter	Value
CPU Decision	Weighted SumUp
CPU Decision (Weights)	[0.4, 0.2, 0.4]
Memory Decision	Weighted SumUp
Memory Decision (Weights)	[0.8, 0.2]
Ensemble of Metrics Decision	Weighted SumUp
Ensemble of Metrics Decision (Weights)	[0.15, 0.1, 0.05, 0.1, 0.05, 0.05, 0.05, 0.1, 0.05, 0.3]
Training Time Decision	Weighted Voting
Training Time Decision (Weights)	[0.2, 0.8]
Evaluation Time Decision	Weighted Voting
Evaluation Time Decision (Weights)	[0.5, 0.3, 0.2]
Final Decision (Weights)	[0.2, 0.8]

Together with the three top-performing algorithms, the comparison is comprised also of the different Ensemble Versions. For what concerns the **Ensembles Threshold(t)'s value**, we decided to leave them with their default values, since they are already optimal, as table (5.16) shows.

Table 5.16. (Customer Dataset's Cluster 2) Ensembles Threshold(t)'s Value: In this table, we present the Threshold Values we use in the various Ensemble Versions.

Model	Threshold (t) Value
Ensemble Hard-Voting	2
Ensemble Soft-Voting (Sum)	2.75
Ensemble Soft-Voting (Max)	0.95
Ensemble Soft-Voting (Mean)	0.95
Ensemble Soft-Voting (Median)	0.95
Ensemble Soft-Voting-Weighted (F1-Score)	0.95
Ensemble Soft-Voting-Weighted (Entropy)	0.95
Ensemble Soft-Voting-Weighted (MM)	0.95
Ensemble Soft-Voting-Weighted (Brier Score)	0.95
Ensemble Soft-Voting-Weighted (ECE)	0.95

5.7.2 Model Selection: Decision Workflow Application (Training Time)

With the hyper-parameter weights meticulously determined in the previous step (table 5.15), the Model Selection Workflow can now be applied to select the optimal AD algorithm. During the evaluation process, the initial outcome obtained is focused on the **Training (Design) Time Performances**. The following table presents the Design Time Performances (table 5.17). Subsequently, the results for the Memory Decision are presented in table 5.18, followed by the results for the CPU Decision in table 5.19. The evaluation further extends to the Resources Decision (table 5.20). Finally, we culminate the process with the **Final Design Time Table**, which includes the outcome of the best model at Training Time (table 5.21).

Table 5.17. (Customer Dataset's Cluster 2) Time Performances: Training Time: Here, we present the Training Time Performances results and we identify the best model at Training Time (Time).

Model	(Design) Time Performance
mlp	597.072
gbt	541.260
log	**61.662**
Ensemble Hard-Voting	1199.994
Ensemble Soft-Voting (Sum)	979.903
Ensemble Soft-Voting (Max)	1004.844
Ensemble Soft-Voting (Mean)	1003.487
Ensemble Soft-Voting (Median)	1346.038
Ensemble Soft-Voting-Weighted (F1-Score)	1323.706
Ensemble Soft-Voting-Weighted (Entropy)	1346.530
Ensemble Soft-Voting-Weighted (MM)	1373.255
Ensemble Soft-Voting-Weighted (Brier Score)	1335.865
Ensemble Soft-Voting-Weighted (ECE)	1407.821

Table 5.18. (Customer Dataset's Cluster 2) Memory Performances: Training Time: In this table, we present the Training Time Memory Performances results and we identify the best model at Training Time (Memory).

Model	Memory Utilization(%)	Memory Swaps(%)	Memory Decision
mlp	82.00	45.70	74.74
gbt	81.00	45.80	**73.96**
log	81.10	45.70	74.02
Ensemble Hard-Voting	81.37	45.73	74.24
Ensemble Soft-Voting (Sum)	81.37	45.73	74.24
Ensemble Soft-Voting (Max)	81.37	45.73	74.24
Ensemble Soft-Voting (Mean)	81.37	45.73	74.24
Ensemble Soft-Voting (Median)	81.37	45.73	74.24
Ensemble Soft-Voting-Weighted (F1-Score)	81.37	45.73	74.24
Ensemble Soft-Voting-Weighted (Entropy)	81.37	45.73	74.24
Ensemble Soft-Voting-Weighted (MM)	81.37	45.73	74.24
Ensemble Soft-Voting-Weighted (Brier Score)	81.37	45.73	74.24
Ensemble Soft-Voting-Weighted (ECE)	81.37	45.73	74.24

Table 5.19. (Customer Dataset's Cluster 2) CPU Performances: Training Time: In this table, we present the Training Time CPU Performances results and we identify the best model at Training Time (CPU).

Model	CPU Utilization(%)	CPU Frequency(%)	CPU Load(%)	CPU Decision
mlp	63.20	78.25	25.00	50.93
gbt	56.80	78.25	3.12	**39.62**
log	55.60	78.25	7.50	40.89
Ensemble Hard-Voting	58.53	78.25	11.87	43.81
Ensemble Soft-Voting (Sum)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting (Max)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting (Mean)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting (Median)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting-Weighted (F1-Score)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting-Weighted (Entropy)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting-Weighted (MM)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting-Weighted (Brier Score)	58.53	78.25	11.87	43.81
Ensemble Soft-Voting-Weighted (ECE)	58.53	78.25	11.87	43.81

Table 5.20. (Customer Dataset's Cluster 2) Resources Decision: Training Time: In this table, we present the Training Time Resources Decision results, identifying the best model at Training Time in terms of Hardware Consumption.

Model	Memory Decision	CPU Decision	Disk Decision	Resources Decision
mlp	74.74	50.93	45.4	0.1
gbt	73.96	39.62	45.4	**0.8**
log	74.02	40.89	45.4	0.4
Ensemble Hard-Voting	74.24	43.81	45.4	0.2
Ensemble Soft-Voting (Sum)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting (Max)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting (Mean)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting (Median)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting-Weighted (F1-Score)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting-Weighted (Entropy)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting-Weighted (MM)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting-Weighted (Brier Score)	74.24	43.81	45.4	0.2
Ensemble Soft-Voting-Weighted (ECE)	74.24	43.81	45.4	0.2

Table 5.21. (Customer Dataset's Cluster 2) Training Time Overall Decision: In this table, we present the Training Time Final Decision obtained by the Decision Workflow application, identifying the best model for Training Time overall, by taking into account both Time Performances and Resources Consumption at Design Time.

Model	Time Decision	Resources Decision	Final Design Time Decision
mlp	597.072	0.1	0.150
gbt	541.260	0.8	**0.900**
log	60.662	0.4	0.600
Ensemble Hard-Voting	1199.994	0.2	0.225
Ensemble Soft-Voting (Sum)	979.903	0.2	0.425
Ensemble Soft-Voting (Max)	1004.844	0.2	0.425
Ensemble Soft-Voting (Mean)	1003.487	0.2	0.425
Ensemble Soft-Voting (Median)	1346.038	0.2	0.225
Ensemble Soft-Voting-Weighted (F1-Score)	1323.706	0.2	0.425
Ensemble Soft-Voting-Weighted (Entropy)	1346.530	0.2	0.425
Ensemble Soft-Voting-Weighted (MM)	1373.255	0.2	0.425
Ensemble Soft-Voting-Weighted (Brier Score)	1335.865	0.2	0.225
Ensemble Soft-Voting-Weighted (ECE)	1407.821	0.2	0.225

Upon thorough evaluation, the **GBT** model emerged as the **Training Time Top-Performing algorithm**, excelling in terms of Resources (both Memory and CPU utilization). While Logistic Regression showcased the primary over Design-Time Performances, GBT's superior Resource Performance tilted the scale in its favor, reflecting the significance placed on efficient resource utilization in the customer's R-T AD system.

5.7.3 Model Selection: Decision Workflow Application (Evaluation Time)

With the Model Selection Workflow in action, we now focus on its next precious output, the **Best Model at Evaluation Time**. Throughout this evaluation process, the primary focus lies on the **Evaluation (RunTime) Time Performances**. Evaluation Time Performances are presented in table 5.22, showcasing the outcomes of the R-T evaluation. Subsequently, the results for the Memory Decision are showcased in table 5.23, and the CPU Decision results are detailed in table 5.24. The evaluation further extends to the Resources Decision (table 5.25). We then focus on the core part of the Evaluation phase, i.e. the Correctness Decision, shown in table 5.26, Finally, the process culminates with the presentation of the **Final Evaluation Time Table**, which encompasses the outcome of the best model at Evaluation Time (table 5.27).

Table 5.22. (Customer Dataset's Cluster 2) Time Performances: Evaluation Time: In this table, we present the Evaluation Time Performance results, identifying the best model at Evaluation Time in terms of Time.

Model	(Evaluation) Time Performance
mlp	**44.783**
gbt	45.995
log	48.610
Ensemble Hard-Voting	6840.705
Ensemble Soft-Voting (Sum)	4682.420
Ensemble Soft-Voting (Max)	4736.383
Ensemble Soft-Voting (Mean)	5002.911
Ensemble Soft-Voting (Median)	6691.061
Ensemble Soft-Voting-Weighted (F1-Score)	8185.423
Ensemble Soft-Voting-Weighted (Entropy)	8146.542
Ensemble Soft-Voting-Weighted (MM)	8051.207
Ensemble Soft-Voting-Weighted (Brier Score)	8371.063
Ensemble Soft-Voting-Weighted (ECE)	8360.746

Table 5.23. (Customer Dataset's Cluster 2) Memory Performances: Evaluation Time: In this table, we present the Evaluation Time Memory Performance results, identifying the best model at Evaluation Time in terms of Memory.

Model	Memory Utilization(%)	Memory Swaps(%)	Memory Decision
mlp	82.30	45.60	74.96
gbt	73.70	47.70	68.50
log	79.00	47.00	72.60
Ensemble Hard-Voting	76.90	49.50	71.42
Ensemble Soft-Voting (Sum)	73.20	48.30	68.22
Ensemble Soft-Voting (Max)	68.10	50.50	**64.58**
Ensemble Soft-Voting (Mean)	80.40	49.00	74.12
Ensemble Soft-Voting (Median)	77.10	58.70	73.42
Ensemble Soft-Voting-Weighted (F1-Score)	74.40	56.10	70.74
Ensemble Soft-Voting-Weighted (Entropy)	73.80	56.60	70.36
Ensemble Soft-Voting-Weighted (MM)	74.00	55.40	70.28
Ensemble Soft-Voting-Weighted (Brier Score)	77.50	53.90	72.78
Ensemble Soft-Voting-Weighted (ECE)	76.10	61.80	73.24

Table 5.24. (Customer Dataset's Cluster 2) CPU Performances: Evaluation Time: In this table, we present the Evaluation Time CPU Performance results, identifying the best model at Evaluation Time in terms of CPU.

Model	CPU Utilization(%)	CPU Frequency(%)	CPU Load(%)	CPU Decision
mlp	62.80	78.25	29.00	52.37
gbt	66.20	78.25	14.50	47.93
log	76.30	78.25	14.62	52.02
Ensemble Hard-Voting	53.80	78.25	27.12	48.02
Ensemble Soft-Voting (Sum)	26.70	78.25	2.62	27.38
Ensemble Soft-Voting (Max)	26.10	78.25	2.75	**27.19**
Ensemble Soft-Voting (Mean)	30.50	78.25	1.37	28.40
Ensemble Soft-Voting (Median)	51.90	78.25	5.62	38.66
Ensemble Soft-Voting-Weighted (F1-Score)	55.00	78.25	33.00	50.85
Ensemble Soft-Voting-Weighted (Entropy)	55.60	78.25	16.50	44.49
Ensemble Soft-Voting-Weighted (MM)	55.50	78.25	12.87	43.00
Ensemble Soft-Voting-Weighted (Brier Score)	57.20	78.25	17.12	45.38
Ensemble Soft-Voting-Weighted (ECE)	57.40	78.25	8.75	42.11

Table 5.25. (Customer Dataset's Cluster 2) Resources Decision: Evaluation Time: In this table, we present the Evaluation Time Resources Decision, identifying the best model at Evaluation Time in terms of Hardware Consumption.

Model	Memory Decision	CPU Decision	Disk Decision	Resources Decision
mlp	74.96	52.37	45.4	0.025
gbt	68.50	47.93	45.4	**0.200**
log	72.60	52.02	45.4	0.050
Ensemble Hard-Voting	71.42	48.02	45.4	0.100
Ensemble Soft-Voting (Sum)	68.22	27.38	45.6	**0.200**
Ensemble Soft-Voting (Max)	64.58	27.19	45.6	**0.200**
Ensemble Soft-Voting (Mean)	74.12	28.40	45.6	**0.200**
Ensemble Soft-Voting (Median)	73.42	38.66	45.6	**0.200**
Ensemble Soft-Voting-Weighted (F1-Score)	70.74	50.85	45.6	**0.200**
Ensemble Soft-Voting-Weighted (Entropy)	70.36	44.49	45.6	**0.200**
Ensemble Soft-Voting-Weighted (MM)	70.28	43.00	45.6	**0.200**
Ensemble Soft-Voting-Weighted (Brier Score)	72.78	45.38	45.7	**0.200**
Ensemble Soft-Voting-Weighted (ECE)	73.24	42.11	45.8	**0.200**

Table 5.26. (Customer Dataset's Cluster 2) Correctness Performance (Ensemble of Metrics): Evaluation Time: In this table, we present the Correctness Performances Results, identifying the best model under a Correctness point of view, by using the application of the novel Ensemble of Metrics Method we proposed.

Model	F1	G-Mean	D-Power	AdjF	Bal-Acc	MCC	Cohen-K	Youden-I	Acc	TPR	CORRECTNESS
mlp	0.736	0.847	0.739	0.738	0.361	0.646	0.661	0.721	0.995	0.723	0.728
gbt	0.729	0.835	0.737	0.730	0.355	0.543	0.540	0.709	0.995	0.711	0.708
log	0.732	0.907	0.571	0.735	0.415	0.497	0.490	0.830	0.995	0.834	0.756
Ens H-V	0.801	0.884	0.491	0.803	0.394	0.602	0.654	0.789	0.996	0.789	0.761
Ens S-V (Sum)	0.644	0.950	0.461	0.655	0.452	0.555	0.569	0.904	0.989	0.914	**0.773**
Ens S-V (Max)	0.662	0.902	0.602	0.668	0.409	0.601	0.628	0.817	0.991	0.824	0.747
Ens S-V (Mean)	0.584	0.965	0.352	0.600	0.467	0.498	0.522	0.933	0.985	0.948	0.763
Ens S-V (Median)	0.614	0.956	0.411	0.627	0.458	0.536	0.538	0.915	0.987	0.927	0.767
Ens S-V-W (F1-Score)	0.584	0.965	0.352	0.600	0.467	0.498	0.522	0.933	0.985	0.947	0.763
Ens S-V-W (Entropy)	0.583	0.961	0.406	0.599	0.462	0.506	0.516	0.924	0.985	0.938	0.761
Ens S-V-W (MM)	0.535	0.925	0.556	0.550	0.429	0.547	0.545	0.857	0.983	0.873	0.728
Ens S-V-W (Brier Score)	0.587	0.965	0.353	0.602	0.466	0.479	0.505	0.932	0.985	0.947	0.761
Ens S-V-W (ECE)	0.584	0.966	0.355	0.600	0.467	0.515	0.538	0.933	0.985	0.948	0.765

Table 5.27. (Customer Dataset's Cluster 2) Evaluation Time Overall Decision: In this table, we present the Evaluation Final Decision obtained by the Decision Workflow application, identifying the best model at Evaluation Time overall, by taking into account Correctness, Time Performances and Resources Consumption at Design Time.

Model	Time Decision	Resources Decision	Correctness	Final Evaluation Time Decision
mlp	44.783	0.025	0.728	0.450
gbt	45.995	0.200	0.708	0.413
log	48.610	0.050	0.756	0.375
Ensemble Hard-Voting	6840.705	0.100	0.761	0.638
Ensemble Soft-Voting (Sum)	4682.420	0.200	0.773	**0.757**
Ensemble Soft-Voting (Max)	4736.383	0.200	0.747	0.717
Ensemble Soft-Voting (Mean)	5002.911	0.200	0.763	0.737
Ensemble Soft-Voting (Median)	6691.061	0.200	0.767	0.737
Ensemble Soft-Voting-Weighted (F1-Score)	8185.423	0.200	0.763	0.675
Ensemble Soft-Voting-Weighted (Entropy)	8146.542	0.200	0.761	0.675
Ensemble Soft-Voting-Weighted (MM)	8051.207	0.200	0.728	0.487
Ensemble Soft-Voting-Weighted (Brier Score)	8371.063	0.200	0.761	0.737
Ensemble Soft-Voting-Weighted (ECE)	8360.746	0.200	0.765	0.737

Upon thorough assessment, the **Ensemble Soft-Voting (Sum)** Model emerged as the top-performing algorithm at Evaluation Time, showcasing exceptional AD Correctness. Moreover, in the realm of Hardware Resources, the Ensemble Soft-Voting (Sum) model, along with other Ensemble versions (excluding Hard Voting), demonstrated exemplary performance. These Ensemble models efficiently utilized HW resources while maintaining high accuracy, making them a compelling choice for the customer's R-T AD system. While MLP exhibited impressive Run Time Performances, the Ensemble Soft-Voting (Sum) model's superiority in Correctness and HW Resource utilization decisively influenced the selection.

5.7.4 Model Selection: Overall Final Decision

In this final Model Selection step, the Design Time and RunTime evaluation results will be merged, taking into account the specific requirements gathered from the customer. By combining these evaluations, we can determine which model offers the most accurate and timely AD predictions while keeping hardware utilization and resource consumption within the desired limits. The Model Selection Workflow's final output will be the **Optimal AD Algorithm** that satisfies all the customer's specifications.

At the very end of the Model Selection Process, we can therefore output the best Model, as the following table (5.28) showcases.

Table 5.28. (Customer Dataset's Cluster 2) Overall Decision: In this table, we present the Overall Decision obtained by the Decision Workflow application, identifying the best model Overall, by taking into account the Training Time and Evaluation Time Decision, according to the Customer Requirements.

Model	Training Time Decision	Evaluation Time Decision	Overall Final Decision
mlp	0.150	0.450	0.390
gbt	0.900	0.413	0.510
log	0.600	0.375	0.420
Ensemble Hard-Voting	0.225	0.638	0.555
Ensemble Soft-Voting (Sum)	0.425	0.757	**0.685**
Ensemble Soft-Voting (Max)	0.425	0.717	0.665
Ensemble Soft-Voting (Mean)	0.425	0.737	0.675
Ensemble Soft-Voting (Median)	0.225	0.737	0.635
Ensemble Soft-Voting-Weighted (F1-Score)	0.425	0.675	0.655
Ensemble Soft-Voting-Weighted (Entropy)	0.425	0.675	0.655
Ensemble Soft-Voting-Weighted (MM)	0.425	0.487	0.475
Ensemble Soft-Voting-Weighted (Brier Score)	0.225	0.737	0.635
Ensemble Soft-Voting-Weighted (ECE)	0.225	0.737	0.635

After a comprehensive evaluation, the **Ensemble Soft-Voting (Sum)** Model emerged as the **Best Model Overall**. It excelled in terms of Correctness, outperforming all other algorithms, including GBT, which had been elected as the best algorithm at Training Time. The Ensemble Soft-Voting (Sum) model's exceptional accuracy and AD capabilities positioned it as the top choice for the Customer's real-time application.

5.8 Workflow's Step 6: Model Explanation

With the Model Selection phase concluded and the optimal algorithm, the **Ensemble Soft-Voting (Sum)** Model, chosen for our R-T AD System, we proceed to the **Model Explanation Phase** (as described in Workflow's Step 6: Model Explanation, section 4.10). This critical step focuses on gaining a comprehensive understanding of our selected algorithm's behavior, providing the Customer with detailed explanations regarding its predictions during R-T AD. Specifically, we seek

to elucidate why the Model categorizes specific data samples as anomalies and the key metrics (features) that significantly influence its decision-making process.

To achieve this goal, we employ two advanced Explainability Techniques: **LIME (Local Interpretable Model-Agnostic Explanations)** and **SHAP (SHapley Additive exPlanations)**, providing two complementary perspectives on feature importance and model behavior. LIME enables us to provide rapid local explanations for a specific data sample, offering quick insights into the model's predictions. On the other hand, SHAP delivers more in-depth explanations, albeit with a higher computational cost, giving us a global view of feature importance across multiple data samples.

5.8.1 Local Explainability: LIME over a Data Sample

In this section, we employ the LIME technique to analyze a single anomalous data sample, offering insight into the local explainability of our model. Through the real-time application of the model, we aim to provide a focused interpretation of the model's decision, as discussed in Section 5.29.

Table 5.29. (Customer Dataset's Cluster 2) LIME Local Explanation Settings: In this table, we present the settings used for the LIME Local Explanation over a specific Data Sample (instance).

Hyper-Parameter	Value
discretize_continuous	True
discretizer	quartile
verbose	False
kernel	Gaussian
kernel_width	None
feature_selection	auto

Upon executing the LIME Model Explanation on the selected data sample, the Workflow generates a comprehensive output that represents the ultimate result conveyed to the customer. This output encapsulates the entirety of the MAD Workflow, encapsulating the pertinent insights into the specific anomaly detection decision rendered by the Ensemble Soft-Voting (Sum) Model for the given data instance.

The straightforward and transparent nature of the LIME Model Explanation output empowers the customer to comprehend and take appropriate action on the identified anomalies, thereby underscoring the system's efficacy in real-world AD scenarios.

In Figure 5.9, an illustrative example of a LIME Model Explanation over a Data Sample is provided, emphasizing the clarity and utility of the explanation output for the customer's informed decision-making process.

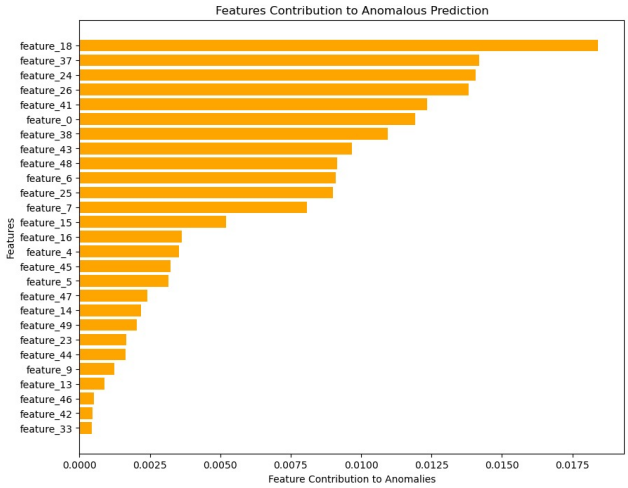


Figure 5.9. LIME: Local Model Explanation Over a Single Data Sample: The illustration showcases the outcome presented to the customer, offering a comprehensive breakdown of the features that exerted the most significant influence on the model's anomalous prediction for the given data instance.

5.8.2 Global Explainability: SHAP over an Entire Day

In the event that the Customer seeks a broader understanding of the AD model's behavior, we employ **SHAP** as a **Global Explainability** technique. Supposing the Customer's interest lies in observing the Model Explanation over the past day, along with the corresponding feature importance that influenced the model's decisions during that time frame, we aim to analyze the Customer Data, which is gathered at a Granularity of 15 minutes, allowing us to obtain Model Explainability over 96 data samples. Our aim is to uncover the collective contributions of individual features across these multiple samples, shedding light on how these features have collectively influenced the model's decision-making process during the specified time frame.

We will apply the SHAP Global Explanation with the hyper-parameters set as explained in the following table 5.30.

Table 5.30. (Customer Dataset's Cluster 2) SHAP Global Explanation Settings: In this table, we present the settings used for the SHAP Global Explanation over 96 data samples representing a particular whole day of the Customer Dataset.

Hyper-Parameter	Value
kernel	Gaussian
link	logit
l1_reg	None
l2_reg	None
batch_size	None

After executing the SHAP Global Explainability on the selected data sample, the Workflow generates an extensive output that is subsequently shared with the Customer, who will carefully analyze and interpret the information, particularly when anomalies are detected. The Customer's ability to comprehend and act upon the comprehensive and detailed SHAP Model Explanation output plays a crucial role in ensuring the system's efficacy in real-world AD scenarios. Equipped with comprehensive and interpretable information, the Customer can make well-informed

decisions and take appropriate actions based on the anomaly detection results. An example of a SHAP Model Explanation over a Data Sample is provided in Figure 5.10.

Figure 5.10. SHAP Global Model Explanation: Force Plot: In this figure, we present a visual representation of the feature contributions over the 96 data samples Anomaly Detection Prediction.

The utilization of the **Force Plot** showcased above allows us to present the **Global Feature Contributions concerning Anomalies**. As anticipated, the overall sum of contributions is positive, indicating that the features contribute predominantly to the Normal-Behaviour Prediction across all the samples. However, the Force Plot also provides valuable insights into the specific features that significantly influence the Model's Anomaly Predictions for individual samples. To further elucidate this aspect, we employ a **Bar Plot** (as shown in Figure 5.11), which offers a more comprehensive and refined explanation of the results.

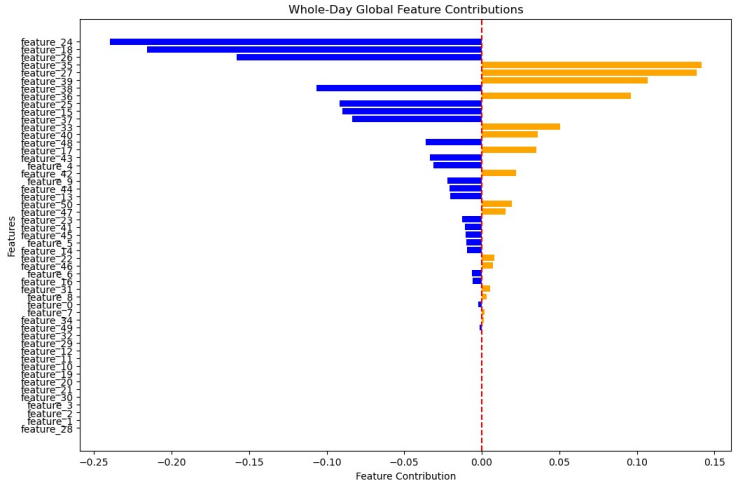


Figure 5.11. SHAP Global Model Explanation: Bar Plot: This figure presents a comprehensive visual representation of the feature contributions across 96 data samples in the Anomaly Detection Prediction. The blue bars depict the features that predominantly contribute to the Model predicting Normal-Behaviour Samples, while the orange bars represent the features that significantly influence the Model's Anomalous Predictions. By analyzing this Bar Plot, we gain valuable insights into how the individual features contribute to the Model's decision-making process, providing a clear and detailed understanding of the anomaly detection behavior.

Undoubtedly, the Customer's primary concern lies in understanding the reasons behind the detected anomalies. As such, the final outcome provided to the Customer, through the SHAP Model Explanation, will serve as a vital tool for gaining comprehensive insights into the anomaly detection process. The visual representation depicted in Figure 5.12 will be presented to the Customer, showcasing the feature contributions over the 96 data samples used in the Anomaly Detection Prediction.

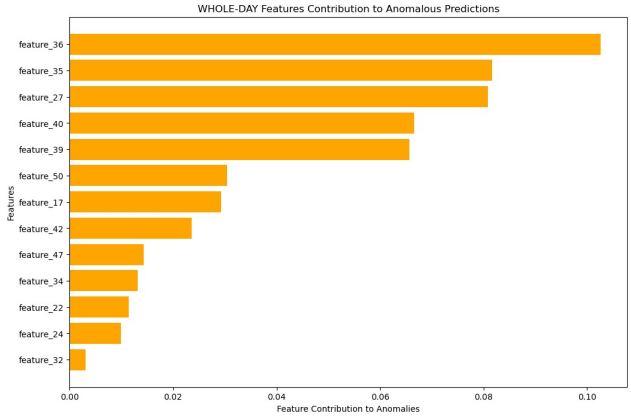


Figure 5.12. SHAP Global Model Explanation: Insights for the Customer: This figure represents the final result provided to the Customer, presenting the feature contributions over the 96 data samples used in the Anomaly Detection Prediction.

5.9 Previous Solution VS My Solution: Comparisons

In the context of our R-T AD System, a fundamental aspect lies in selecting the most suitable algorithm that can efficiently detect anomalies in 5G networks. In the previous solution (the starting point of our exploration), we faced the challenge of managing and working with an extensive set of models, with each cell requiring a separate model (approximately 814 models in our dataset). This approach proved to be resource-intensive and time-consuming, making it impractical for large-scale deployment in real-world scenarios.

In contrast, the solution proposed by our Workflow significantly streamlines the model selection process by employing a more efficient ensemble approach. Instead of maintaining one model per cell, we transition to either one model per cluster or three models per cluster when the Workflow selects one of the Ensemble Versions as the best model. This strategic shift drastically reduces the number ofs models needed to handle the dataset while maintaining a high level of AD accuracy, as figure 5.13 showcases.

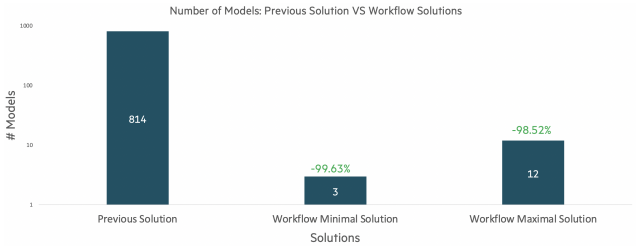


Figure 5.13. Number of Models: Previous Solution VS Workflow Solutions Comparison: This figure illustrates the substantial reduction in the number of models required for AD achieved by the proposed Workflow compared to the previous solution. In the best-case scenario, where the Workflow's decision leads to the selection of a single algorithm, we observe a remarkable decrease of 99.36% in the number of models to train and maintain. Even in the case where the Ensemble solution is chosen, the number of models is still substantially reduced by 98.52%.

In addition to the notable reduction in the number of models, the transition to

the Workflow proposed also yields a gargantuan **Reduction** in both **Time and Hardware (HW) Performances**, as shown in the three following tables 5.14, 5.15 and 5.16.

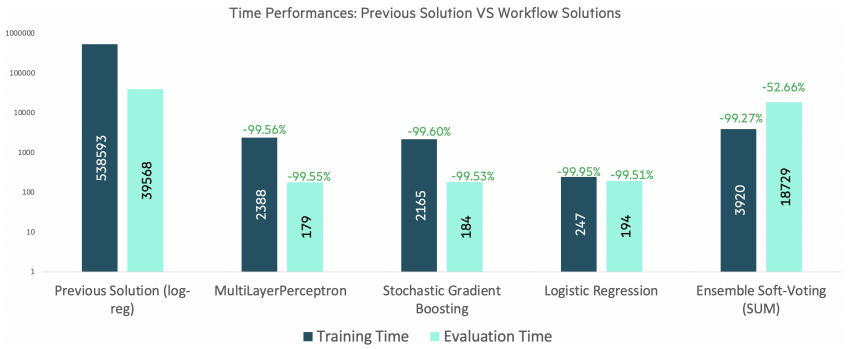


Figure 5.14. Time Performance Comparison: Previous Solution VS Workflow Solutions: This figure vividly illustrates the tremendous reduction in Time Performances achieved by the proposed Workflow compared to the previous solution. In the best-case scenario, where the Workflow's decision leads to the selection of a single algorithm, we observe a remarkable decrease of 99.95% in the Training Time and 99.55% in the Evaluation Time. Even in the case where the Ensemble solution is chosen, these reductions remain substantial, with a 99.27% decrease in Training Time and a 52.66% decrease in Evaluation Time.

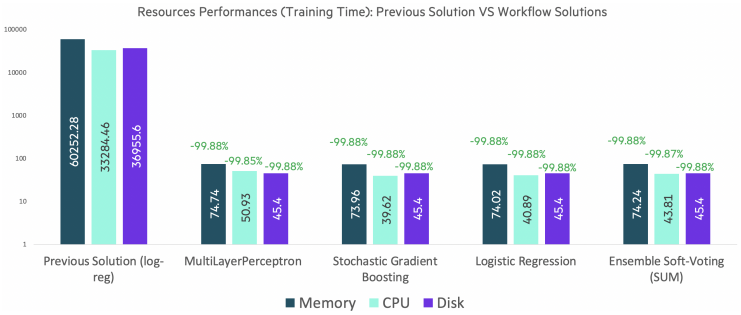


Figure 5.15. Resources Performance (Training Time) Comparison: Previous Solution VS Workflow Solutions: This figure presents a detailed comparison of the Resources Consumption (in particular of the three main components: Memory, CPU, and Disk) between the previous solution and the proposed Workflow during the Training Time phase, with reduction ranging from a 99.85% to a 99.88% reduction at Training Time.

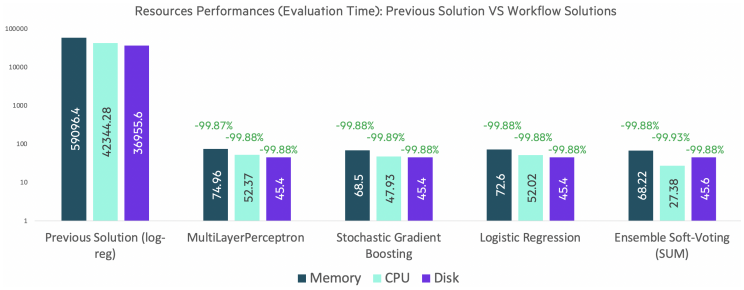


Figure 5.16. Resources Performance (Evaluation Time) Comparison: Previous Solution VS Workflow Solutions: This figure provides a comprehensive comparison of the Resources Consumption, specifically focusing on the three main components: Memory, CPU, and Disk, between the previous solution and the proposed Workflow during the Evaluation Time phase. The chart vividly illustrates the remarkable reduction in Resources needed for AD, ranging from an astounding 99.87% to an impressive 99.93% reduction at Evaluation Time.

While the proposed Workflow achieves significant gains in time and hardware performances, it is essential to carefully assess the potential trade-offs in terms of **Correctness**. As shown in figure 5.17, we observe a slight reduction in correctness when comparing the proposed Workflow with the previous solution.

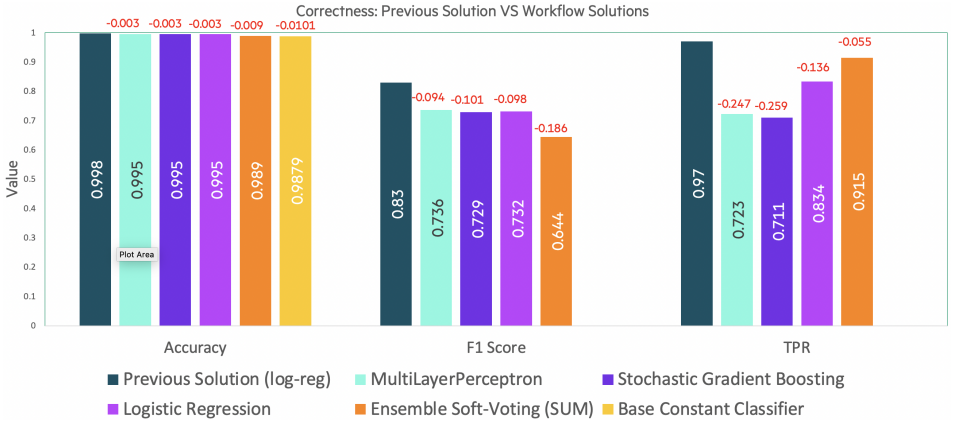


Figure 5.17. Correctness Comparison: Previous Solution VS Workflow Solutions: This figure presents a Correctness comparison between the previous solution and the proposed Workflow. From the figure, we can observe that the Accuracy, in general, experiences only a marginal decrease, with the Ensemble being the worst-case scenario, losing 0.009 points compared to the previous solution. However, it is essential to note that even in the worst-case scenario, the Workflow's solutions still outperform the **Base Constant Classifier**, representing a rudimentary approach where all data points are predicted with the same constant value, in this case, being Normal Behavior Samples. When evaluating the F1-Score, the Ensemble also shows a slightly larger reduction, with a loss of 0.186 points compared to the previous solution. However, it is crucial to consider the most critical metric, the TPR. Here, the Ensemble performs remarkably well, nearly reaching the level of the Previous Solution, with only a 0.055-point loss in the Ensemble case.

The slight reduction in correctness is a result of the ensemble approach employed by the Workflow. In the previous solution, each cell had its dedicated model, which could lead to overfitting specific cell characteristics. In contrast, the ensemble approach in the Workflow generalizes better across cells but may introduce some

loss of correctness in specific scenarios.

It is crucial to acknowledge that while there is a trade-off between correctness and the proposed Workflow approach, the achieved correctness in the Workflow remains high and satisfies the requirements of the Customer for R-T AD in 5G networks. Additionally, the Workflow's ability to dramatically reduce time and hardware resource usage significantly outweighs the slight reduction in correctness. The slight trade-off in correctness is an acceptable compromise given the significant gains in time and hardware performances, making the Workflow a highly valuable and practical tool for R-T AD in large-scale 5G networks.

Chapter 6

Conclusions

This thesis represents a transformative journey that began with the task of replacing the previous **HPE's Univariate Anomaly Detection Tool (UAD Tool)** with a Multivariate solution, aimed at elevating the Anomaly Detection capability in 5G networks to new heights. Throughout the internship, numerous challenges were encountered, ranging from handling weakly correlated, high-dimensional, and unlabeled multivariate time series data in 5G networks to ensuring the scalability, accuracy, and interpretability of the proposed solution.

The proposed Multivariate Anomaly Detection Workflow serves as a powerful response to these challenges, providing an elegant and effective approach that generalizes across diverse customer datasets while achieving significant reductions in time and resource consumption without compromising on correctness. By harnessing the potential of machine learning, deep learning, and ensemble methods, the workflow enables accurate and efficient AD across large-scale 5G networks.

At the heart of this research lies the development of the MAD Workflow, revolutionizing the traditional one-model-per-cell approach of the **UAD Tool**. The introduction of cluster-based modeling and ensemble methods has been instrumental in overcoming scalability limitations, resulting in a drastic reduction in the number of models needed for detection. The novel Model Selection Workflow further enriches the process by facilitating informed decision-making, considering various factors such as correctness, time performance, and resource consumption. This adaptive and flexible approach allows the best algorithm to be tailored precisely to each customer's unique requirements, empowering network managers with a customized AD solution.

To ensure transparency and interpretability, the workflow incorporates two state-of-the-art Explainability Techniques, SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations). These techniques offer invaluable insights into feature contributions at both global and local levels, enhancing the understanding of the model's decision-making process.

In conclusion, this thesis represents a significant contribution to Hewlett Packard Enterprise (HPE) and the field of AD in 5G networks. The proposed MAD Workflow not only addresses the limitations of the **UAD Tool** but also offers a robust and scalable solution for real-world implementation. The transformative impact of this research is further acknowledged by the recognition received in the form of an Honourable Mention in the CTG's Hackathon and the Best-Paper Award. With its efficiency, accuracy, and interpretability, the workflow has the potential to drive transformative changes in 5G network management, ensuring reliable and efficient delivery of services in the dynamic landscape of modern telecommunications.

6.1 Celebrating Success: CTG's AI Hackathon Honorable Mention

During the prestigious CTG's AI Hackathon, a thrilling event dedicated to showcasing innovative AI solutions, I had the privilege to participate as an individual participant, by presenting my groundbreaking workflow aimed at revolutionizing decision-making in AI model selection customer-wise. The Hackathon brought together brilliant minds from diverse backgrounds, all striving to push the boundaries of AI technology. My workflow, titled "New Prescriptive Approach to Select the Optimal Multi-Variate Anomaly Detection Model" earned an esteemed **"Honorable Mention" recognition**. The details of this Workflow are explicitly explained in Workflow's Step 5: Model Selection(section 4.7).

6.2 Celebrating Success: Best HPE Paper Award

During my six-month internship, I had the privilege of participating, with the Path Finder group, in the esteemed internal HPE context aimed at recognizing and rewarding the **Best HPE Paper**. The internal company context showcased a diverse range of cutting-edge research conducted by talented teams across various domains. Our team's triumphant paper, titled "Simplifying the Application of AI/ML at the 5G Edge," showcases the remarkable intersection of artificial intelligence (AI), machine learning (ML), and transformative 5G technology. Through rigorous experimentation and meticulous analysis, our paper sheds light on the immense potential of AI/ML techniques in enhancing the performance, efficiency, and security of 5G networks.

6.3 Future Work

As we move forward with the development and implementation of our R-T AD System, several avenues present themselves for further exploration and improvement. These areas of future work are crucial for enhancing the system's capabilities and ensuring its effectiveness in a dynamic and evolving environment. In this section, we highlight key areas that warrant particular attention in the ongoing research of our proposed Workflow:

- **Seasonality Study: Customer-Centric Analysis**: One of the critical challenges in AD lies in accounting for seasonality and periodic patterns inherent in various data streams. To achieve more accurate AD, we must acknowledge that each customer may exhibit unique seasonal patterns, peaks, and periodic trends. Therefore, we propose to embark on a comprehensive **Seasonality Study** that caters to each customer's specific characteristics, by carefully analyzing historical data from each customer and identifying recurring patterns and seasonal fluctuations in such a way as to better distinguish between genuine anomalies and regular fluctuations, minimizing false alarms(FP) and optimizing the system's overall performance.
- **Transfer Learning for Enhanced Model Generalization**: In our pursuit of developing a robust and scalable AD system, the challenge of effectively adapting the model to new customers with limited data arises. The scarcity of labeled data from new customers can hinder the system's ability to generalize and perform optimally for each unique case. To address this challenge, we propose exploring **Transfer Learning** techniques to leverage knowledge learned from existing customers and apply it to new, unseen customer data.

- **Continuous Model Monitoring and Retraining**: As the R-T AD System operates in dynamic environments, the concept of continuous model monitoring and retraining becomes crucial. To maintain peak performance and adapt to evolving data distributions, we propose implementing mechanisms for ongoing model evaluation and updating. Monitoring the system's performance and tracking model drift over time will enable us to detect changes in data patterns and assess model accuracy. When significant drift is detected, automatic retraining or fine-tuning of models can be triggered to ensure the system remains up-to-date and reliable.

- **User Interface and Visualization Enhancements**: To maximize the system's usability and accessibility, investing in user interface (UI) and visualization enhancements is paramount. Presenting complex model outputs in an intuitive and easy-to-understand manner will empower users to make informed decisions based on the system's insights.

  The proposed workflow for the R-T AD System serves as a strong foundation for future advancements and improvements. As we continue to refine the system and adapt it to real-world scenarios, several avenues for future work emerge to enhance its capabilities and performance. Through a comprehensive and dedicated approach to future work, we aim to elevate the system's capabilities, providing our customers with a state-of-the-art solution for R-T AD in diverse and dynamic environments.

Feedbacks

Upon the culmination of this enriching experience, I took the initiative to solicit written feedback from my esteemed colleagues with whom I had the privilege of collaborating. This feedback will serve as a valuable resource for my continued personal and professional growth.

Ceccherini Paolo, Product Management Lead
Alessio has demonstrated a structured and analytical problem-solving approach reducing the complexity and adopting innovative data modeling and prediction analysis technique. During his internship, he has been extremely professional, engaged, and committed to outperform on his assignments.

Biswadeb Dutta, Lead Technologist
It has been a pleasure to have Alessio working with our team for his internship. Alessio is extremely motivated and self-driven to learn new subjects and apply them to problems he is working on. He is creative and able to come up with interesting questions that lead to interesting problems. The work he has done for his thesis is a testament to this quality. I am confident that he will be an asset to any organization that he joins. I wish him all the best.

Sanna-Anziani Isabelle, Delivery Manager
Alessio has been showing from the very first day his autonomy, nice and clear communication, and was able to integrate smoothly the team of AI/ML experts he was assigned to. Working in close collaboration with other engineers and our lead technologist he has been able to demonstrate an understanding of the challenges, the ability to propose technical approaches and to implement up to a demonstrable result. It was a pleasure to interact with him and I received only very good feedback from anyone who collaborated with him.

De Diego Raul, Product Manager
Alessio has brought a good contribution in one of the most innovative areas in AI applied to Telco Network Operations: Multi-variate Anomaly-Detection. We will reuse if not all, a big part of the job presented for the evolution of the HPE products in this area. What we have observed during the 6-month is full engagement with passion. He can be really proud of this journey and feel like a real Path-Finder.

Kamal Zishan, Senior Data Scientist
I'm truly impressed by Alessio's commitment to the project. His thorough understanding of the subject matter, along with his analytical skills, greatly contributed to the project's success. His insights into multivariate anomaly detection techniques were instrumental in driving the project forward. His collaborative approach was noticeable throughout the project. He consistently engaged with team members, shared

his knowledge, and actively participated in weekly meetings. His technical expertise was evident in the innovative solutions you proposed and implemented.

Akanksha Jain, Senior Data Scientist
It was nice working with Alessio. He is meticulous and enthusiastic about learning new things. He has picked up several fresh ideas in a very short period of time and applied them to his project. He is very adept at articulating and presenting his work. I wish him all the best in his upcoming ambitions.

Bibliography

[1] C. C. Aggarwal. *Outlier Analysis*. 2016.

[2] Josephine Sarpong Akosa. Predictiveaccuracy : A misleading performance measure for highly imbalanced data. 2017.

[3] A. Ali, Siti Mariyam Hj. Shamsuddin, and Anca L. Ralescu. Classification with class imbalance problem: A review. In *Soft Computing Models in Industrial and Environmental Applications*, 2015.

[4] Mohammad Braei. *Master thesis*. PhD thesis, 12 2019.

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] Markus Breunig, Peer Kröger, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000.

[7] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[8] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[9] Misha Denil and Thomas Trappenberg. Overlap versus imbalance. pages 220–231, 05 2010.

[10] Tomasz Drabas. *Learning PySpark*. Packt, 2017.

[11] Hewlett Packard Enterprise. The original pioneers of silicon valley. url: https://www.hpe.com/us/en/about/history.html (accessed: 19.06.2023).

[12] Jerome Friedman. Stochastic gradient boosting. *Computational Statistics Data Analysis*, 38:367–378, 02 2002.

[13] Samingun Handoyo, Ying-ping Chen, Gugus Irianto, and Agus Widodo. The varying threshold values of logistic regression and linear discriminant for classifying fraudulent firm. *Mathematics and Statistics*, 9:135–143, 03 2021.

[14] D. Hawkins. *Identification of Outliers*. 1980.

[15] Ben Hoyle, Markus Michael Rau, Kerstin Paech, Christopher Bonnett, Stella Seitz, and Jochen Weller. Anomaly detection for machine learning redshifts applied to SDSS galaxies. *Monthly Notices of the Royal Astronomical Society*, 452(4):4183–4194, aug 2015.

[16] What is Apache Spark? What is apache spark? url: https://www.hpe.com/it/it/what-is/spark.html (accessed: 30.06.2023).

[17] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6:429–449, 11 2002.

[18] Shehroz S. Khan and Michael G. Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, jan 2014.

[19] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 01 2009.

[20] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[21] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[22] Giang Nguyen, Abdesselam Bouzerdoum, and Son Phung. *Learning Pattern Classification Tasks with Imbalanced Data Sets*. 10 2009.

[23] MLlib Overview. Mllib overview url: https://spark.apache.org/docs/latest/ml-guide.html (accessed: 30.06.2023).

[24] PySpark Overview. Pyspark overview url: https://spark.apache.org/docs/latest/api/python/ (accessed: 30.06.2023).

[25] Foster J. Provost and Gary M. Weiss. Learning when training data are costly: The effect of class distribution on tree induction. *CoRR*, abs/1106.4557, 2011.

[26] Nadia Rahmah and Imas Sitanggang. Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra. *IOP Conference Series: Earth and Environmental Science*, 31:012012, 01 2016.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[28] Sofia Visa and Anca Ralescu. The effect of imbalanced data class distribution on fuzzy classifiers - experimental study. pages 749 – 754, 06 2005.

[29] Using Python with Apache Spark. Using python with apache spark? url: https://developer.hpe.com/blog/using-python-with-apache-spark/ (accessed: 30.06.2023).

[30] Yanminsun, Andrew Wong, and Mohamed S. Kamel. Classification of imbalanced data: a review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23, 11 2011.

[31] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *Knowledge and Data Engineering, IEEE Transactions on*, 18:63– 77, 02 2006.