NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Anomaly Detection and Prediction on Kubernetes Resources

Vyron-Georgios I. Anemogiannis

Supervisor: **Stathes Hadjiefthymiades,** Professor

ATHENS

FEBRUARY 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση και Πρόβλεψη ανωμαλιών σε πόρους του Κυβερνήτη

Βύρων-Γεώργιος Ι. Ανεμογιάννης

Επιβλέπων: **Ευστάθιος Χατζηευθυμιάδης,** Καθηγητής

ΑΘΗΝΑ

ΦΕΒΡΟΥΑΡΙΟΣ 2023

BSc THESIS

Anomaly Detection and Prediction on Kubernetes Resources

Vyron-Georgios I. Anemogiannis
S.N.: 1115202000008

SUPERVISOR: **Stathes Hadjiefthymiades,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση και Πρόβλεψη ανωμαλιών σε πόρους του Κυβερνήτη

Βύρων-Γεώργιος Ι. Ανεμογιάννης
Α.Μ.: 1115202000008

ΕΠΙΒΛΕΠΩΝ: Ευστάθιος Χατζηευθυμιάδης, Καθηγητής

ABSTRACT

In recent years, the deployment of software projects on cloud infrastructure managed through Kubernetes has become commonplace. This trend has led to the management of numerous components spread across nodes, each one with its unique specifications. Effectively overseeing and ensuring the smooth operation of such infrastructures poses a challenging and resource-intensive task.

This thesis, a part of the EO4EU project, seeks to streamline monitoring resources by employing machine learning techniques. To achieve this objective, we initially constructed a parallel representation of the Kubernetes cluster using a Graph Database, regularly maintained and kept up-to-date. By monitoring the graph and leveraging its structure to highlight interconnections among components, we gained insights into the cluster's behavior.

Utilizing unsupervised machine learning models, we categorized our observations as either anomalous or not. Subsequently, we employed these labeled observations to train a supervised machine learning model. This model facilitates the updating of the graph with an anomaly score assigned to each component. While the project is still in its early stages, preliminary tests have demonstrated promising results, even though real-world data has not yet been incorporated.

SUBJECT AREA: Anomaly Detection on Kubernetes

KEYWORDS: Cloud Computing, Kubernetes, Supervised Machine Learning, Unsupervised Machine Learning, Infrastructure Management, Graph Databases, Anomaly Detection, Prediction, Monitoring

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, ο προγραμματισμός έργων λογισμικού σε υποδομές στον νέφος που διαχειρίζονται μέσω του Kubernetes έχει γίνει συνηθισμένος. Αυτή η τάση έχει οδηγήσει στη διαχείριση πληθώρας στοιχείων που διασκορπίζονται σε κόμβους, καθένας με τις δικές του μοναδικές προδιαγραφές. Η αποτελεσματική επίβλεψη και εξασφάλιση της ομαλής λειτουργίας τέτοιων υποδομών αποτελεί μια προκλητική και πόρο-επιβαρυντική εργασία.

Αυτή η διατριβή, η οποία αποτελεί μέρος του έργου EO4EU, έχει ως στόχο τον ευθυγράμμιση των πόρων παρακολούθησης με τη χρήση τεχνικών μηχανικής μάθησης. Για να επιτευχθεί αυτός ο στόχος, κατασκευάσαμε αρχικά μια παράλληλη αναπαράσταση του συστήματος Kubernetes χρησιμοποιώντας μία Γραφική Βάση Δεδομένων, το οποίο συντηρούνται και ενημερωνόνται τακτικά. Μέσω της παρακολούθησης του γραφήματος και της αξιοποίησης της δομής του για να επισημαίνουμε τις αλληλεπιδράσεις μεταξύ των στοιχείων, αποκτήσαμε εισηγήσεις σχετικά με τη συμπεριφορά του συστήματος.

Χρησιμοποιώντας μοντέλα μη υποβλητικής μάθησης, κατηγοριοποιήσαμε τις παρατηρήσεις μας ως ανομαλίες ή μη. Στη συνέχεια, χρησιμοποιήσαμε αυτές τις επισημειωμένες παρατηρήσεις για να εκπαιδεύσουμε ένα μοντέλο υποβλητικής μάθησης. Αυτό το μοντέλο διευκολύνει την ενημέρωση του γραφήματος με ένα βαθμό ανομαλίας που αντιστοιχεί σε κάθε στοιχείο. Ενώ το έργο βρίσκεται ακόμη στα αρχικά του στάδια, πρόχειρες δοκιμές έχουν επιδείξει ελπιδοφόρα αποτελέσματα, αν και δεν έχει ενσωματωθεί ακόμη πραγματικά δεδομένα από τον πραγματικό κόσμο.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ανίχνευση Ανωμαλιών στο Κυβερνήτη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Υπολογιστικό Νέφος, Kubernetes, Μηχανική Μάθηση, Διαχείριση Υποδομής, Γραφικές Βάσεις Δεδομένων, Εντοπισμός Ανωμαλιών, Παρακολούθηση, Έργο EO4EU, Μοντέλα Μη Υποβλητικής Μάθησης, Μοντέλα Υποβλητικής Μάθησης, Παράλληλη Αναπαράσταση, Τακτική Συντήρηση, Αλληλεπιδράσεις Στοιχείων, Προκαταρκτικές Δοκιμές, Δεδομένα Πραγματικού Κόσμου

This thesis is dedicated to the people who support and love me, providing unwavering encouragement and strength that fueled my journey to completion.

ACKNOWLEDGEMENTS

I extend my deepest gratitude to Professor Stathes Hadjiefthymiades for his continuous guidance and the opportunity to be part of his research team since my undergraduate years. I am also thankful to Dr. Kakia Panagidi and Babis Andreou for their unwavering support and insightful comments throughout the thesis writing process. Finally, a special thanks to my colleagues in the I9 office for their camaraderie and support, turning our workspace into an environment where both dedication and goofiness coexist. Their collective influence has profoundly shaped this thesis and my academic journey, and I am truly appreciative of their contributions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In today's dynamic technological landscape, the deployment of applications on cloud infrastructure has become increasingly synonymous with efficiency, scalability, and adaptability. Central to this paradigm shift is the ubiquitous adoption of Kubernetes, a powerful container orchestration system that has redefined the way applications are managed and scaled. Kubernetes provides a robust framework for automating the deployment, scaling, and operation of application containers, thereby revolutionizing the landscape of cloud-native computing.

The widespread adoption of Kubernetes has ushered in a new era of agility and responsiveness in deploying and managing complex applications. However, with the growing complexity of these distributed systems, there arises an imperative need to ensure their seamless operation, performance, and security. As organizations embrace the scalability benefits of Kubernetes, they are confronted with the challenge of effectively monitoring and safeguarding the health and performance of their resources.

This thesis, created in the context of the EO4EU project, endeavors to address this critical challenge by delving into the realm of anomaly detection and prediction for Kubernetes resources. Anomalies, deviations from the expected behavior of the system, can be indicative of potential issues and inefficiencies. By harnessing machine learning techniques, this research aims to develop a framework capable of detecting what constitutes as anomalous behavior within Kubernetes and be able to predict such behavior on new monitoring observations.

In the pursuit of enhancing the operational resilience of Kubernetes environments, this thesis unfolds with a structured exploration across six chapters, each contributing uniquely to the overarching research agenda. Chapter 1 serves as the introduction, laying the groundwork for the subsequent chapters. Building upon this foundation, Chapter 2 delves into the intricate landscape of Infrastructure as a Service, encompassing the evolution of cloud computing, the significance of containers, the central role of Kubernetes, and the integration of Neo4j as a graph database to optimize data representation.

Chapter 3 unfolds as an educational voyage into the principles of machine learning (ML), distinguishing between supervised and unsupervised learning, and introducing critical ML algorithms. A special emphasis is placed on hyperparameter optimization, setting the stage for the subsequent chapters. Chapter 4 meticulously conducts a comprehensive review of related work, critically analyzing existing literature and methodologies related to anomaly detection in Kubernetes environments.

The central core of the thesis, Chapter 5, addresses the problem landscape within Kubernetes environments, unveiling our proposed solution for anomaly detection and prediction. A detailed workflow description sheds light on the integration of infrastructure, machine learning, and graph database technologies. The synergy between these components forms the backbone of our innovative approach.

Chapter 6, orchestrates the model selection process and experiments. We bring forward the criteria for selecting ML models, our hyperparameter optimization strategy, and the design of experiments to evaluate the efficacy of our proposed solution. Results, analyses, and lessons learned contribute to the robustness and applicability of our anomaly detection framework, thereby marking the culmination of our contributions to the field. Chapter 7, concludes the thesis and describes the future work.

2. INFRASTRUCTURE AS A SERVICE

2.1 Cloud Computing

2.1.1 What is Cloud Computing

As the demand for computational resources continues to grow [30], many individuals and organizations are turning to cloud computing to fulfill their requirements. Cloud computing eliminates the need for physical infrastructure by providing IT resources over the Internet. These resources include computing power in the form of Nodes, storage, and networking capabilities. The services offered encompass both applications and the underlying hardware infrastructure, leading to the emergence of Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [3].

2.1.2 Key Elements of Cloud Computing

At the core of Cloud Computing lies the key concept of Virtualization [21], i.e. the cloud provides a parallel and distributed service, employing virtual computers to deliver computational resources tailored to the dynamic needs of both cloud computing providers and consumers. Unlike traditional one-time purchases or flat fees, consumers are billed based on their usage of these Virtual Machines (VMs).

Virtualization operates by maintaining a pool of elastic computing and storage resources, allocating them to cloud service providers and users only when required to execute a workload. This approach facilitates load balancing and optimal resource assignment. Furthermore, running applications on VMs enables seamless migration between different VMs and hosts without disrupting ongoing processes. The flexibility arises from the ability to replicate and run the VM specification on various hosts, ensuring the uninterrupted functionality of the application.

The distributed and parallel nature of cloud computing allows services to be extended across various geographical regions with minimal latency penalties. This is achieved by creating VMs for applications in diverse data centers. Cloud computing encompasses three primary deployment models:

1. **Private Cloud:** In a private cloud deployment, a single organization establishes or rents cloud infrastructure exclusively for their use. This ensures heightened privacy and security.
2. **Public Cloud:** Public cloud deployment opens access to VMs for a fee, enabling anyone to run their applications. This model prioritizes flexibility, catering to a broad user base.
3. **Hybrid Cloud:** As the name suggests, the hybrid cloud is a fusion of private and public clouds, combining the advantages of both models. This approach offers a versatile solution tailored to diverse needs.

2.1.3 Advantages of Cloud Computing

The scaling and dynamic nature offered by virtualization, offers many advantages both to the users and the cloud providers. The resource management becomes easy thanks to the abstraction and isolation of the underlying hardware and networking resources. It allows for on demand scaling of the applications, simply by increasing or decreasing the VM copies for a given application, giving much flexibility to the users. It provides security since the configuration for a VM can be scanned for malware and allows for snapshot taking at any time. It allows for copies of the application to exist in multiple data centers, providing low latency for the users of the application, no matter where they are so long as they have a good internet connection. It allows for dynamic payment plans so users can pay their fair share as they use the services instead of a flat fee, no matter their use, as well as, it removes the cost of the hardware acquisition, maintenance and upgrade from the user by splitting it between all of them. It is eco friendly, since it reduces the need for individual hardware, consolidating the computing needs in big data centers while also decreasing the idle time of the computers thanks to dynamic scheduling of workloads.

2.1.4 Cloud Service Tiers

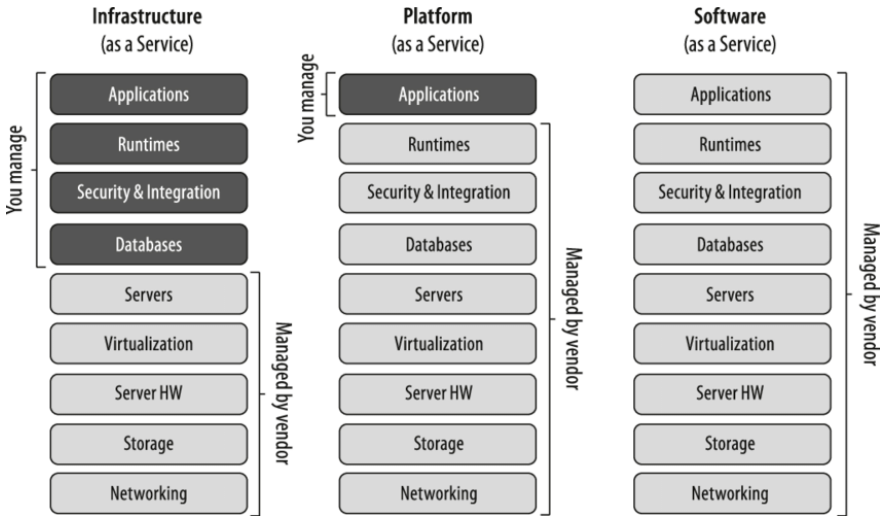

Figure 2.1: IaaS vs PaaS vs SaaS [17]

2.1.4.1 Software as a Service (SaaS)

SaaS allows end users to access application software through the cloud via a web browser, eliminating the need for dedicated hardware. The SaaS provider manages all aspects of the application, from maintenance to updates. Examples of SaaS applications include widely-used social media platforms, such as Facebook and Twitter. Moreover, traditional applications that were once installed on individual devices, like Microsoft's Office 365, have transitioned to online services, enabling users to utilize productivity tools seamlessly over the internet, while facilitating collaborative and flexible work environments.

2.1.4.2 Platform as a Service (PaaS)

PaaS caters to both application developers and end-users. It provides a cloud-based platform for application development and management, along with an end-user portal. Cloud service providers furnish the necessary resources for development and application execution, including servers, operating systems, storage, networking, databases, and frameworks. PaaS also offers an interface for development teams to code, test, deliver, and deploy applications. Notable examples of PaaS include Microsoft Windows Azure, Google App Engine, and AWS Elastic Beanstalk.

2.1.4.3 Infrastructure as a Service (IaaS)

IaaS allows customers to subscribe and gain on-demand access to networking, storage, and servers via the Internet. Users can configure the hardware as if it were on-premises; however, instead of physical hardware, these resources exist in a data center. This means that users have the flexibility to choose the operating system, programming language, frameworks, and resource allocation according to their specific requirements. Examples of IaaS providers include Microsoft Azure, Google Cloud, and AWS.

2.2 Microservices

2.2.1 What are Microservices

Microservices represent an architectural style for application development, characterized by the creation of multiple independently developed, tested, and deployed services. Each service is assigned a single business responsibility. This stands in contrast to the traditional Monolithic Architecture, where all the application's modules are included in a single code base, as shown in 2.2. Microservices are often employed in conjunction with cloud computing to leverage the scalability, flexibility, and distributed nature of cloud environments. [26, 6, 8]

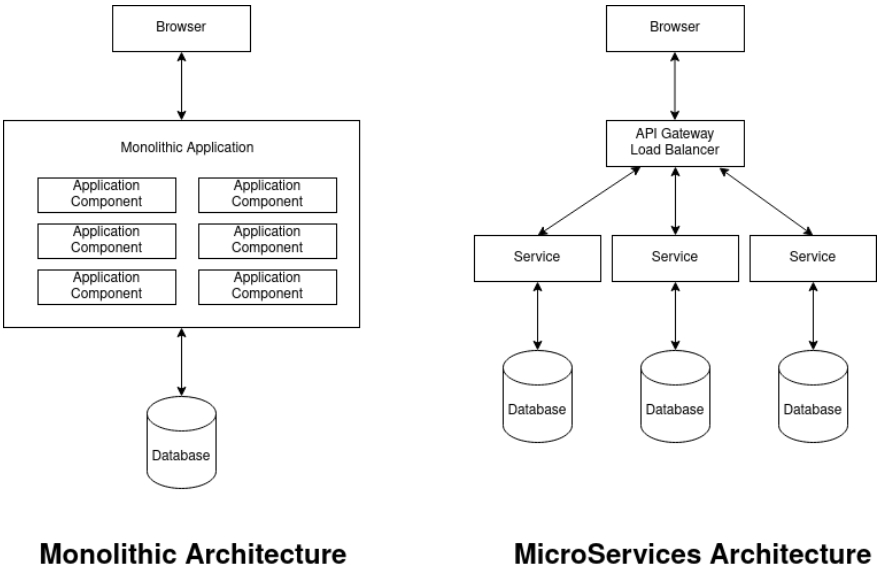

Figure 2.2: Monolithic vs Microservices Architecture

2.2.2 Characteristics of Microservices

2.2.2.1 Independent Development

Developers identify standalone functions within the application, and each identified service can be developed independently. Each developer can focus on their assigned service without being constrained by the progress of other services. Communication between services occurs through APIs, enabling developers to test individual components without dependence on preceding or subsequent components. This distribution of workload among developers enhances the efficiency of the development process.

2.2.2.2 Independent Deployment and Scaling

As each microservice functions as its own small application, it can be deployed independently of others and made production-ready in isolation. Utilizing Containers as presented in section 2.3, developers can package code and its dependencies into a fully functioning application. This independence allows each microservice to have its own resources that can be scaled based on demand, regardless of other application components. This flexibility, compared to traditional architectures where resources would need to scale for the entire application, enables efficient scaling and concurrent operations for tasks without data consistency requirements. Microservices' scalability is a significant advantage, and updates to a specific part of the application do not necessitate redeployment of all components, simplifying fixes and enhancing flexibility.

2.2.2.3 Loose Coupling

In the Microservices Architecture, the concept of loose coupling extends to both runtime and design-time scenarios.

The architecture minimizes runtime coupling by allowing each service to independently complete its assigned tasks asynchronously. This is made possible by the self-contained design of services, which listen for API calls, avoiding direct communication that might create dependencies until a response is received.

While runtime coupling is effectively minimized, design-time coupling—pertaining to the need for simultaneous modifications to multiple services (e.g., API changes)—is also addressed. This occurs as components communicate via API calls, necessitating updates to both components when there are changes to the API.

2.2.2.4 High Availability

Microservices Architecture enhances availability by facilitating quick recovery in case of host failure. Application components are distributed across various servers from different service providers, often with multiple replicas to distribute the load and serve diverse geographical areas. When a server fails, tasks can be redirected to other components on different servers until the downed server is operational. Additionally, containers provide a lightweight, full runtime environment, allowing components to be rapidly deployed on new servers. The Microservices Architecture delivers high availability through its distributed and resilient nature.

2.3 Containers

2.3.1 What are Containers

Containers fundamentally leverage virtualization technology. They encapsulate code, files, and dependencies of a service into a single package, forming a deployable standalone unit. Once created, containers can be executed on various hardware infrastructures, both physical and virtual. Beyond hardware independence, containers offer compatibility with diverse operating systems.

2.3.2 How Containers Work

Containers provide a virtual environment for executing container images. These images, specified by the Open Container Initiative Image Specification, include all essential information for running a container. The images encompass the application's code, provided files, and dependencies. The container stack consists of the Infrastructure (bare metal server), the Operating System (OS), the Container Engine (runtime), and the Application with its dependencies. The Container Engine takes the container image, creates the container, and serves as a liaison between the OS and the Container, scheduling necessary resources for execution.

2.3.3 Containers and Traditional Virtualization

Virtualization involves an abstraction layer over the base system, providing virtual resources that are independent execution units. Traditional Virtualization, utilizing a hypervisor-based approach, employs a Virtual Machine Monitor as a hardware abstraction layer. Each virtual machine (VM) has its own OS and execution context. Figure 2.3 presents a comparison between the two virtualization schemas, i.e. Traditional and Container Virtualization.

2.3.3.1 Traditional Virtualization

In the hypervisor-based Traditional Virtualization, a Virtual Machine Monitor sits atop the original system's OS, offering complete abstraction for VMs. Each VM operates with its own OS and execution context, benefiting from a dedicated hardware abstraction layer.

2.3.3.2 Container Virtualization

Container Virtualization, or OS-level Virtualization, distributes the host machine's resources to VMs that share the same host OS and run on the same kernel. Despite sharing the kernel, VMs remain isolated from each other. This approach is more lightweight than traditional virtualization, as it eliminates the need for a complete OS on each virtual instance. Containers provide an abstraction layer through a system call - ABI layer.

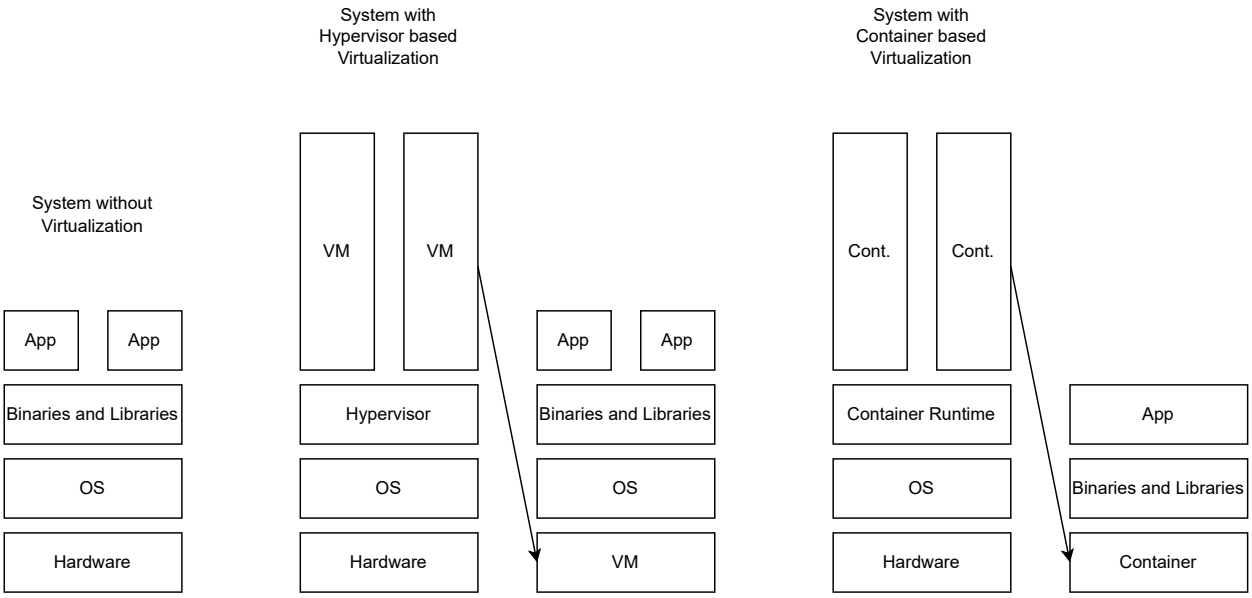

Figure 2.3: Virtualization Comparison

2.3.4 Advantages of Containerization

Containerization offers several advantages that make it a preferred approach in modern software development. According to industry experts, containerization provides a streamlined and efficient solution for software deployment, with benefits spanning efficiency, portability, security, and management processes [5].

2.3.4.1 Efficiency

Containers, being devoid of an entire operating system, boast lightweight characteristics compared to their hypervisor counterparts. This absence of an OS, and the subsequent elimination of its initialization process, results in quicker startup times and, by extension, accelerated scaling. Freed-up resources can be redirected for deploying additional containers or supporting more resource-intensive applications.

2.3.4.2 Portability

Containerization enhances portability, as container deployment is platform-independent, applicable to bare metal servers or virtual infrastructures. This flexibility empowers developers and users to run containers anywhere by simply creating a container image.

2.3.4.3 Security

Containerization ensures security by isolating applications from both the host OS and other containers. Additionally, system administrators can scan container images for potential threats before deployment, fortifying the security posture.

2.3.4.4 Management

Containers benefit from efficient management through container management software. Platforms like Kubernetes offer built-in capabilities for installation, upgrades, and rollback processes of containers, streamlining administrative tasks.

2.3.5 Containerization and Microservices

The Microservices architecture, characterized by loosely coupled small services, aligns seamlessly with containerization. Each container can host a single service, providing its runtime and file system while isolating it from other services, with communication occurring exclusively through APIs. Container scaling enables the application to adjust the number of service instances dynamically, optimizing resource utilization. Leveraging containers allows for easy deployment of application components across various geographical regions to serve a broader customer base, while remaining adaptable to server switches in emergency scenarios.

2.4 Container Management with Kubernetes

In the Microservices Architecture, where the application is segmented into smaller, self-contained services, each running on a container, the need for efficient container management becomes apparent. Enter Kubernetes, often abbreviated as K8s, a powerful platform designed for the management of containerized applications. Originally developed by Google, Kubernetes was open-sourced in 2014 [35]. Operating at the container level, Kubernetes provides users with the freedom to construct their development platform and is capable of supporting any application type that can run on a container, according to the Kubernetes documentation [9].

2.4.1 Capabilities of Kubernetes

Kubernetes (K8s) boasts a rich set of features that make it a robust platform for container orchestration. From seamless automation to efficient load balancing and self-healing, Kubernetes excels in providing a comprehensive solution for managing containerized applications.

2.4.1.1 Automation and Load Balancing

K8s excels in automation by effortlessly creating and deploying containers within designated nodes. It intelligently considers CPU and memory requirements, ensuring optimal resource utilization. At the controller level, K8s achieves the user's desired container state through automated processes, seamlessly replacing old containers during Rollouts or Rollbacks.

2.4.1.2 Service Discovery and Load Balancing

K8s facilitates container communication within the cluster and with the external world through Ingress, serving as a proficient service discovery tool. Additionally, it acts as a load balancer, evenly distributing traffic among containers during periods of high demand.

2.4.1.3 Storage Orchestration and Configuration Management

Kubernetes empowers users to mount diverse storage systems, whether local or from public cloud providers like AWS. It simplifies application configuration through configuration maps and secrets, storing critical information such as credentials or SSH keys. In the event of changes, redeployment is unnecessary; the user can efficiently update the config map or secret with the new information.

2.4.1.4 Self-Healing and Horizontal Scaling

Ensuring the continuous operation of workloads, Kubernetes employs self-healing mechanisms, automatically restarting containers that fail health checks defined by the user. Moreover, based on predefined metrics, Kubernetes dynamically scales applications by adding or removing containers. This ensures optimal resource consumption, allowing the application to efficiently handle the load or scale down to conserve resources as needed.

2.4.2 Architecture of Kubernetes

2.4.2.1 Master Node

The Master Node serves as the management hub for the Kubernetes Cluster, overseeing various controlling processes. It is equipped with essential components, including the API Server, Controller Manager, Scheduler, and ETCD. In production environments, the Master Node is often distributed across multiple machines for redundancy and high availability.

The **API Server** facilitates communication between users and the cluster, executing tasks through the kubectl command. Kubectl enables actions such as deploying applications, managing cluster resources, and viewing logs.

The **Controller Manager** supervises cluster controllers in a unified process. Notable controllers include Node controllers, responding to node failures, EndpointSlice controllers, linking services and pods, and the ServiceAccount controller, creating service accounts for new namespaces. Cloud controllers establish connections between the cluster and the cloud provider's API.

The **Scheduler** assigns newly created Pods to nodes, considering factors like resource requirements, availability, locality, and deadlines.

The **ETCD** serves as a swift and accessible storage solution for all cluster data.

2.4.2.2 Worker Nodes

Worker Nodes, or simply Nodes, host Pods that encapsulate containers constituting the application. Three integral components ensure proper Node functionality:

The **Kubelet** ensures adherence to container specifications, overseeing the creation and health of containers initiated by Kubernetes.

The **Kube-proxy** functions as a network proxy, managing packet forwarding for Pods. It maintains network rules on nodes to enable communication between Pods and external entities.

The **Container Runtime** facilitates container creation, managing their life cycle within the cluster. Kubernetes supports various container runtimes, including containerd and any other implementation adhering to the Container Runtime Interface.

2.4.2.3 Kubernetes Cluster

The culmination of Kubernetes architecture results in the establishment of a Kubernetes Cluster when deploying applications. This cluster seamlessly integrates the Control Plane, represented by the Master Node, and one or more Worker Nodes. An visual representation can also be found in figure 2.4.

The **Control Plane**, distributed across multiple machines for enhanced reliability, orchestrates the cluster's holistic functionality. It encompasses crucial components, including the API Server, Controller Manager, Scheduler, and ETCD.

Complementing the Control Plane, the **Worker Nodes**, or simply Nodes, serve as the operational backbone. They host Pods, which encapsulate the application's containers, working in tandem with components like the Kubelet, Kube-proxy, and Container Runtime to ensure optimal performance.

The management of this cohesive Kubernetes Cluster is facilitated through the **kubectl** command-line tool. This robust interface empowers users to manipulate various API objects, such as Pods and Namespaces, streamlining the orchestration of containerized applications.

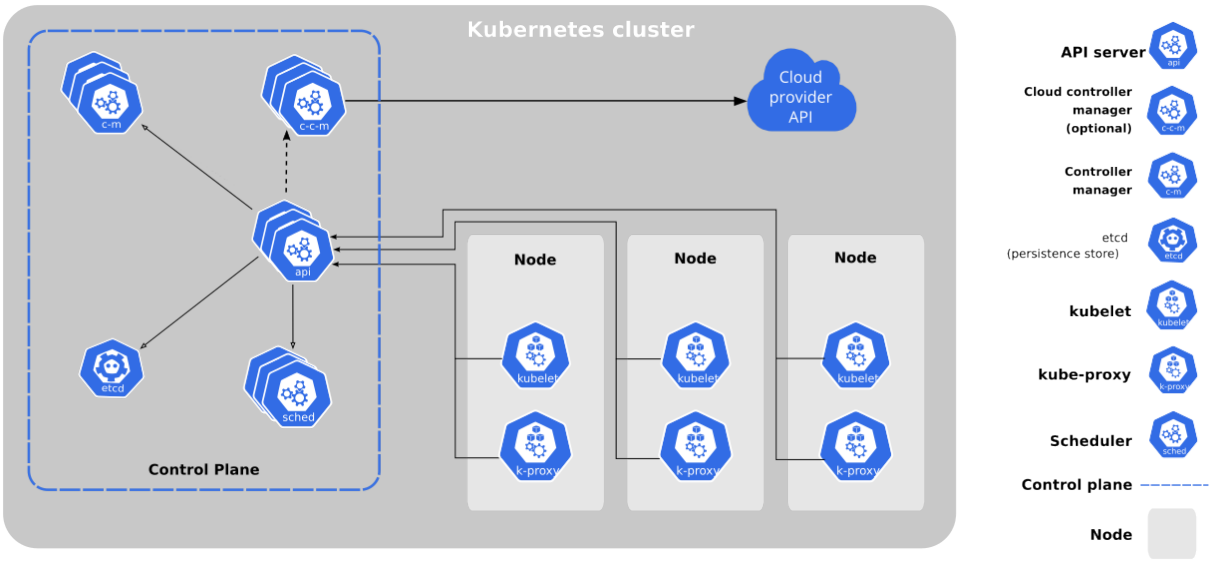

Figure 2.4: Kubernetes Components

2.5 Resource Registry

Within the Kubernetes Cluster, a harmonious interplay of various components brings the application to life. Each component contributes distinct functionalities crucial for the cluster's seamless operation. Central to this orchestration is the Resource Registry, a comprehensive repository that encapsulates a representation of every object deployed within the cluster at any given moment. Each entry in the resource registry comprises the component's type, a unique identifier, and information detailing the behavior of the component. In figure 2.5 one can find a graph represention of all the components interacting with each other.

2.5.1 Namespace

2.5.1.1 Description

The **Namespace** serves as an isolation technique for resource groups within the cluster. By default, all components are deployed in the default namespace. Namespaces are useful when a specific scope is required for names of namespaced objects like Deployments. Each namespace can only have a single instance of a given name, facilitating resource division between users through quotas.

2.5.1.2 Interaction With Other Components

The namespace encompasses various namespaced components. In the **Resource Registry**, the namespace is connected to all components except the non-namespaced ones, such as Persistent Volumes and Nodes that do not belong to any namespace.

2.5.2 Config Map and Secret

2.5.2.1 Description

Config Maps and **Secrets** enable the portability of container images by removing the need to store environment-specific configuration information within them. They consist of key-value pairs designed not to hold large amounts of data. Config Maps hold non-confidential configuration data, whereas Secrets store sensitive information.

2.5.2.2 Interaction With Other Components

Config Maps and Secrets are always contained within a namespace and can be mounted by Pods of the same namespace for use by their containers.

2.5.3 Persistent Volume and Persistent Volume Claim

2.5.3.1 Description

Apart from Nodes, the cluster includes storage. A **Persistent Volume** is a storage class that can be mounted to Pods. When a Pod ceases to exist, the Persistent Volume remains unaffected and available. The Pod can mount a Persistent Volume using a **Persistent Volume Claim**, specifying the needed storage and access modes. Pods can also have **Ephemeral Volumes** whose lifecycle matches the Pod's.

2.5.3.2 Interaction With Other Components

Persistent Volumes are independent entities not connected to Nodes or Namespaces. They only connect to Persistent Volume Claims, which are namespaced and can be mounted by Pods for use by their containers.

2.5.4 Deployment and Replica Set

2.5.4.1 Description

Replica Sets ensure that Pods are running at any given time according to their specifications. The term "set" indicates that it manages one or more Pods, all with the same specification. **Deployments** provide an easy way to manage Replica Sets and their Pods. Users can describe the desired state of their app, and the deployment creates the necessary Replica Sets.

2.5.4.2 Interaction With Other Components

Both Deployments and Replica Sets belong to a namespace. Deployments manage Replica Sets, and Replica Sets are connected to the Pods they manage.

2.5.5 Pod and Container

2.5.5.1 Description

The **Pod** is the smallest deployable unit in the Kubernetes cluster, grouping one or more Containers. Containers within the same Pod are co-managed and run in a shared context. Pods can be managed by various components and are namespaced. They connect to Services for communication and can mount different types of Volumes.

2.5.5.2 Interaction With Other Components

Pods can be managed by Replica Sets, Jobs, Stateful Sets, and Daemon Sets. They are namespaced and can connect to Services for communication within the Cluster and externally. Pods can mount various types of Volumes, and Containers within Pods can mount a subset of these Volumes.

2.5.6 Stateful Set, Daemon Set, Job and CronJob

2.5.6.1 Description

Like replica Sets all of the above components directly manage Pods. Each one of them has its unique characteristics that differentiate them from Deployments.
Stateful Sets are used to run stateful applications like MySQL that need to keep track of their data stored inside them. They manage this by giving a unique identifier to their Pods and in the case of replacement, the new Pod inherits the old ones id and state - data.
Daemon Sets are used when the user wants to run some Pods in all Nodes of the Cluster. This means that a Daemon set is automatically deployed in any new Node and deleted as the Node ceases to exist.
Jobs and CronJobs create a set of Pods that are executed until a specified number of them succeed. In case of failure the Pod is rescheduled. Cron Jobs differentiate by being triggered at specific time inter voles to complete tasks like backups.

2.5.6.2 Interaction With Other Components

All the above components are namespaced meaning they are connected to a namespace. They also manage Pods so they are also connected to the Pods they manage.

2.5.6.3 Stateful Set

Description The **Stateful Set** is a crucial Kubernetes component designed for managing stateful applications within the cluster. Unlike stateless applications, stateful applications, such as databases (e.g., MySQL), require persistent storage and unique network identities. Stateful Sets provide a solution by assigning a unique identifier (hostname) to each Pod they manage. This identifier remains consistent even during replacement or rescheduling, ensuring data consistency across the application's lifecycle. Stateful Sets play a vital role in maintaining the order and reliability of stateful applications within the dynamic Kubernetes environment.

Interaction With Other Components Similar to other components, Stateful Sets are namespaced, linking them to a specific namespace within the cluster. Additionally, Stateful Sets directly manage Pods, maintaining a one-to-one relationship between Stateful Sets and the Pods they control. This close association allows Stateful Sets to preserve the state and identity of stateful applications, ensuring a seamless and reliable operation.

2.5.7 Daemon Set, Job, and CronJob

2.5.7.1 Description

Daemon Sets, Jobs, and CronJobs are Kubernetes components, each with unique characteristics in managing Pods.

Daemon Sets ensure that a copy of a Pod runs on all Nodes in the Cluster. This is particularly useful for scenarios where a specific task or service should be deployed on every Node, ensuring comprehensive coverage across the entire cluster.

Jobs and **CronJobs** are designed for specific task execution. **Jobs** create Pods to perform a task, completing the execution upon success. In contrast, **CronJobs** provide a scheduled approach, triggering tasks at specific time intervals, such as periodic backups or maintenance operations.

2.5.7.2 Interaction With Other Components

These components are namespaced, associating them with a specific namespace within the cluster. As they directly manage Pods, they are connected to the Pods they create and oversee. This direct interaction enables precise control over task execution and resource allocation, contributing to the efficient orchestration of Kubernetes workloads.

2.5.8 Service

2.5.8.1 Description

Services in Kubernetes provide a dynamic abstraction layer for networking between Pods within the cluster, acting as a robust decoupling mechanism. They offer a stable endpoint (Cluster IP or external IP) that abstracts the underlying Pods, allowing for a consistent entry point for communication. Services use label selectors in their specifications, enabling them to intelligently connect to any Pods inside the namespace that match the specified criteria.

One of the key features of Services is their support for **load balancing**, which enhances scalability and availability by efficiently distributing incoming traffic across multiple Pods. This load balancing capability ensures optimal utilization of resources and seamless operation, especially in dynamic and evolving environments.

2.5.8.2 Interaction With Other Components

Services are namespaced components, associating them with a specific namespace within the cluster. They connect directly to the Pods and Containers they service, serving as a

crucial bridge for intra-cluster communication. Additionally, Services have a connection to **Ingresses**, which act as entry points to the Cluster for external network access. Ingresses often use Services as their backends, exposing HTTP and HTTPS routes to the Services within the cluster.

2.5.9 Ingress

2.5.9.1 Description

Ingress serves as a vital component when a Service requires network access outside the Cluster. Ingresses act as entry points, enabling the exposure of HTTP and HTTPS routes to the Services within the Cluster. They provide a gateway for external traffic, facilitating efficient communication between the Cluster and the broader network.

2.5.9.2 Interaction With Other Components

Ingresses are namespaced components, associating them with a specific namespace within the cluster. They play a critical role in facilitating external access by utilizing **Services** as their backends. Ingresses leverage Services to route and manage incoming traffic, serving as a key element in enabling external connectivity for applications deployed within the Cluster.

2.5.10 Node

2.5.10.1 Description

For the successful functioning of a Kubernetes Cluster, **Nodes** are indispensable. Nodes serve as the deployment locations for Pods, constituting the fundamental infrastructure of the Cluster. Each Node is equipped with essential components, including the kubelet, Container Runtime, and Kube Proxy, collectively forming the backbone of the Kubernetes environment.

2.5.10.2 Interaction With Other Components

Nodes are unique in that they are not namespaced components; rather, they are standalone entities in the Kubernetes architecture [9]. Nodes establish a direct connection with the Pods assigned to them, providing the physical or virtual environment for the execution of containerized workloads. This direct association emphasizes the Node's role as the underlying infrastructure that supports and hosts the deployed Pods.

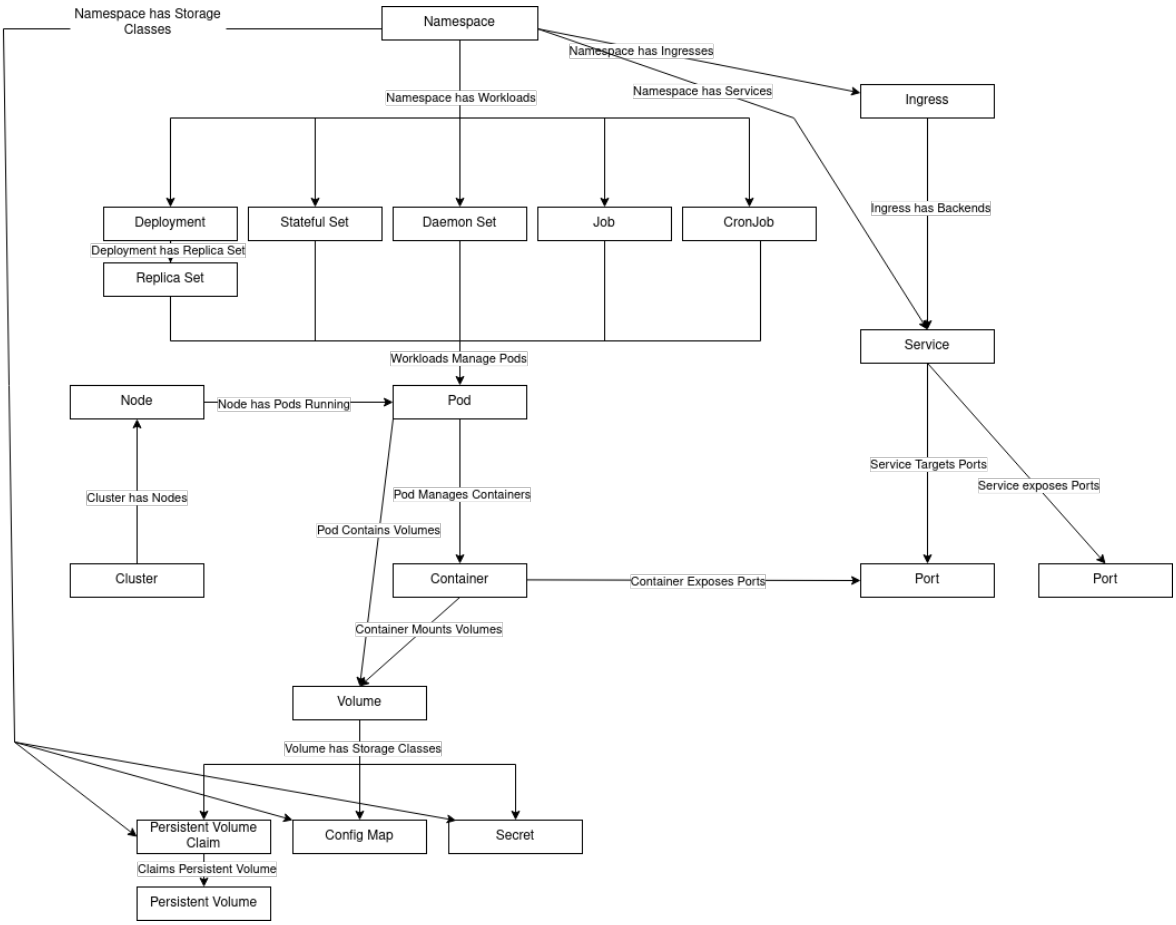

Figure 2.5: Resource Registry

2.6 Neo4j

Neo4j is an open-source database falling under the NoSQL (Not Only SQL) category, distinguishing itself by employing the Cypher query language instead of traditional SQL, commonly associated with relational models. [27, 33, 32]

2.6.1 Architecture

Graphs in Neo4j are composed of nodes and relationships, referred to as vertices and edges, respectively. Traversal of the graph involves navigating through edges between vertices. Neo4j, as a Graph Database, deviates from traditional table structures, representing entities as vertices and relationships as edges within the graph. Nodes and relationships are labeled to denote their groupings, similar to an Heterogeneous graph, and can store key-value pairs as properties. Relationships are directional, connecting nodes in a flexible schema-less structure dynamically built with each addition of nodes and relationships.

2.6.2 What Neo4j offers

2.6.2.1 Transaction Management

Neo4j upholds data integrity through transaction management, supporting ACID (Atomicity, Consistency, Isolation, Durability). This level of transactional consistency is a feature relatively uncommon in many NoSQL databases.

2.6.2.2 High Availability

Implemented with a Master-Worker architecture, Neo4j ensures high availability. The Master manages write operations, serving as a centralized controller, while Workers in the cluster maintain copies of the database, enhancing fault tolerance in case of a Node failure.

2.6.2.3 Cypher Query Language

Neo4j employs the Cypher query language tailored for graph data. Cypher offers users a concise and straightforward method to perform Create, Read, Update, and Delete (CRUD) operations by utilizing ASCII-Art to represent patterns, enhancing readability and simplicity.

2.6.2.4 Efficiency for Relation-Centric Databases

Neo4j's graph structure alleviates the need for complex Join operations, making traversal of relationships efficient. This characteristic makes it particularly suitable for databases with numerous relationships, resulting in low latency even with large datasets.

2.6.2.5 Schema-Free

Dispensing with a fixed schema, Neo4j provides flexibility and scalability. Users can extend the database by seamlessly adding nodes and relationships, including those not previously defined, offering adaptability to evolving data needs.

2.6.3 Use Case: Modeling Kubernetes Resources with Neo4j

Neo4j's capabilities extend to modeling Kubernetes resources, enhancing various aspects of resource management within a cluster. In figure 2.6 we used Neo4j to recreate the graph of figure 2.5.

1. **Resource Registry Management:** Neo4j forms the foundation for creating a robust Resource Registry, efficiently tracking and managing Kubernetes objects.
2. **Dependency Visualization:** Facilitates clear visualization of dependencies between Kubernetes resources, aiding in understanding complex relationships.

3. **Dynamic Schema Evolution:** Accommodates the dynamic nature of Kubernetes environments, allowing for organic addition of nodes and relationships.

4. **Cluster Performance Analysis:** Enables performance analysis by modeling resource utilization, communication patterns, and dependencies.

By leveraging Neo4j's graph database capabilities, administrators gain a powerful tool for effectively modeling, managing, and optimizing the complex relationships inherent in Kubernetes resource orchestration.

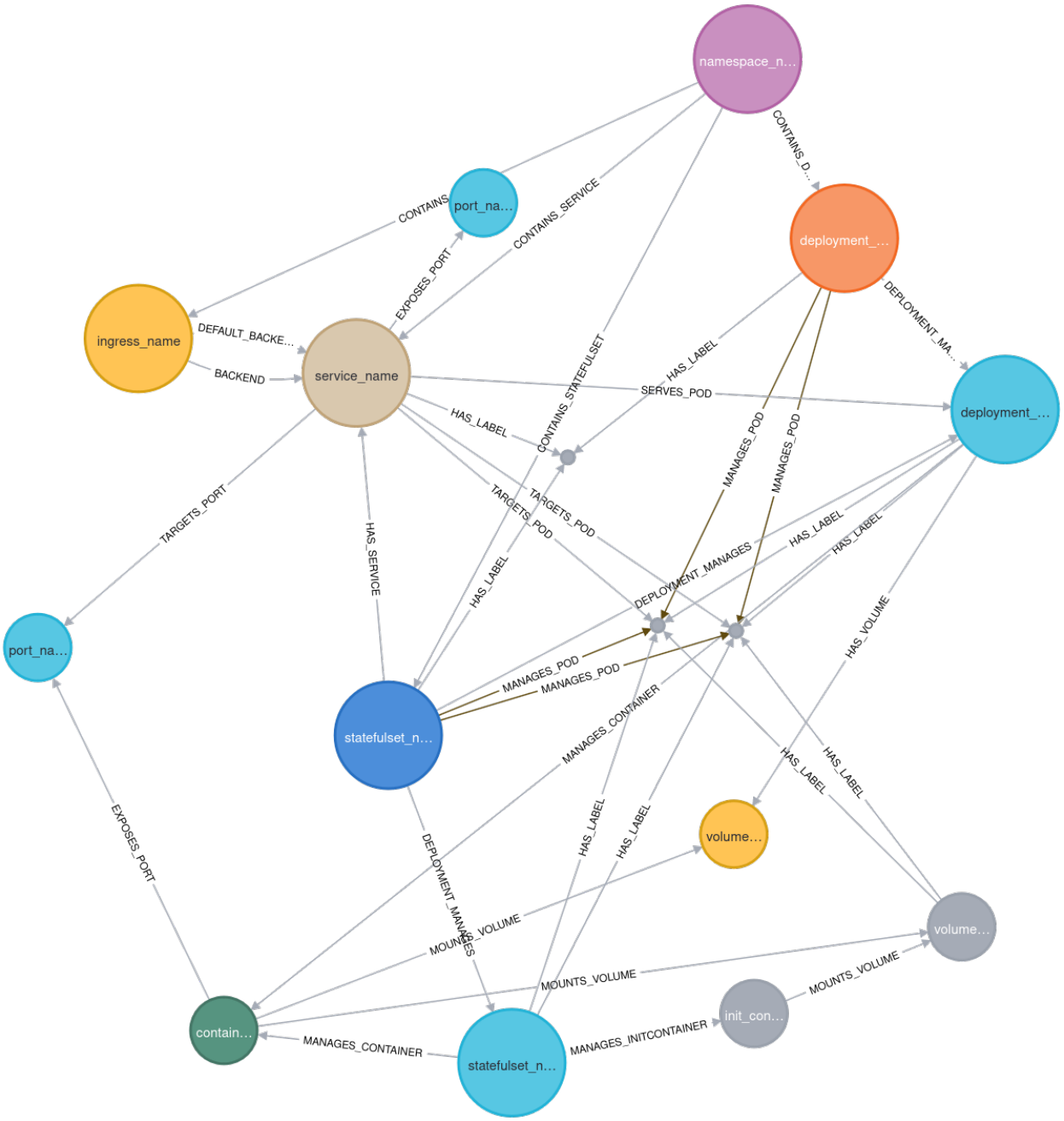

Figure 2.6: Using Neo4J to Represent Figure 2.5

3. MACHINE LEARNING

3.1 Introduction to Machine Learning

3.1.1 What is Machine Learning

Machine Learning (ML), a focal subset of Artificial Intelligence, encompasses a sophisticated set of algorithms and statistical models designed to discern intricate patterns and glean insights from data. In essence, ML empowers systems to recognize patterns, cluster information, classify data, and make predictions, all driven by the inherent ability to learn and adapt without explicit programming. This dynamic capability positions ML as a transformative force, enabling intelligent decision-making based on the analysis of vast datasets. [7, 12]

Machine Learning provides a rich toolbox of algorithms and techniques for various tasks, each with its own set of strengths, weaknesses, and specific requirements. The responsibility falls on the scientist to select a model that optimizes entropy, ensuring effective task performance with sufficient accuracy, all the while striving to keep things straightforward and minimize complexity. In essence, the scientist's role is akin to finding the right balance between maximizing accuracy and keeping things manageable.

3.1.2 Basic Concepts in ML

3.1.2.1 Features and Labels

In Machine Learning, models learn from data, which is split into two parts: features and labels. Not all datasets have both features and labels. There is a big subsection of ML that delves into datasets without labels.

Features serve as the input to the model and encapsulate the descriptive elements of a given state. For instance, in the context of a grayscale image, the input would manifest as a matrix representing the image's pixel dimensions. Alternatively, for a computer, features might include CPU usage, memory allocation, and network activity.

Labels in contrast with features, articulate the desired output anticipated by the user. They play an important role in refining the model. As the model learns from the provided data, labels act as a guide, enabling the model to make informed predictions when confronted with new, unseen data. Drawing on our earlier examples, consider a scenario where the model is trained on images. The labels in this context could represent the digits (0 to 9) corresponding to what the image portrays. Similarly, in the case of monitoring a computer, the label might signify whether the computer experienced a crash or not. This exemplifies how labels provide the necessary context for the model to generalize its understanding and make accurate predictions.

Features and labels are commonly represented in mathematical notation. Adopting a slightly modified notation from the book [12], the dataset, denoted as $\mathcal{X}$, consists of N ordered pairs of features and labels. The features are represented as a vector $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$,

where k is the number of features, and the label is denoted as y.

With features and labels defined, the dataset $\mathcal{X}$ can be expressed as $\mathcal{X} = \{\mathbf{x}^t, y^t\}_{t=1}^{N}$, encompassing N instances of feature-label pairs indexed from 1 to N.

3.1.2.2 Training, Validation and Test Data

To facilitate the learning process of a machine learning model, data is a prerequisite. To optimize this process, the dataset is divided into Training, Validation, and Test sets. The Training set typically encompasses the majority of data points as it is utilized to fine-tune the model. Tuning involves adjusting the model's parameters based on its performance on the training set, evaluated using an objective function. Simultaneously, we assess the model's ability to generalize by introducing a Validation set, containing data points unseen during training. Throughout training epochs, the model classifies this set to gauge its efficacy in generalization.

Upon identifying the model parameters that minimize validation error, a final training phase is conducted with these optimized parameters. Subsequently, the model undergoes evaluation on a distinct Test set, providing an unbiased assessment of its generalization performance. This separation of training, validation, and test sets ensures a robust evaluation framework.

Various techniques exist for splitting datasets, with Cross Validation, where the dataset is iterative split in different subsets, standing out as one of the most well-known methodologies.

3.1.2.3 Parameters and HyperParameters

Two vital elements in the realm of machine learning theory are parameters and hyperparameters. Parameters, often referred to as weights and biases, constitute the internal variables of a model, undergoing fine-tuning throughout the training process. The dynamic adjustments to these parameters are instrumental in enabling models to excel across various machine learning tasks.

Conversely, hyperparameters are external variables that guide the learning process. They encompass crucial aspects such as the model's complexity, learning rate, regularization strength, and the choice of optimization algorithm. Unlike parameters, hyperparameters are established before training commences and require fine-tuning methodologies, such as grid search or random search.

Successful machine learning model development hinges on finding the right combination of parameters and hyperparameters, striking a balance for accurate and generalizable results. This interplay between internal adjustments (parameters) and external configuration choices (hyperparameters) forms the foundation of effective machine learning model building.

3.1.2.4 Learning Process

The Learning Process is a crucial component in the training of machine learning models, organized into multiple **epochs**. In each epoch, the entire training set is processed, often in batches, to iteratively refine the model's hyperparameters. Following each epoch, the

model's performance is assessed using the validation dataset. The learning process concludes either after a predetermined number of epochs or when the generalization error starts to rise, indicating potential overfitting. In figure 3.1 is a visual representation of the learning process from the lecture notes [18].

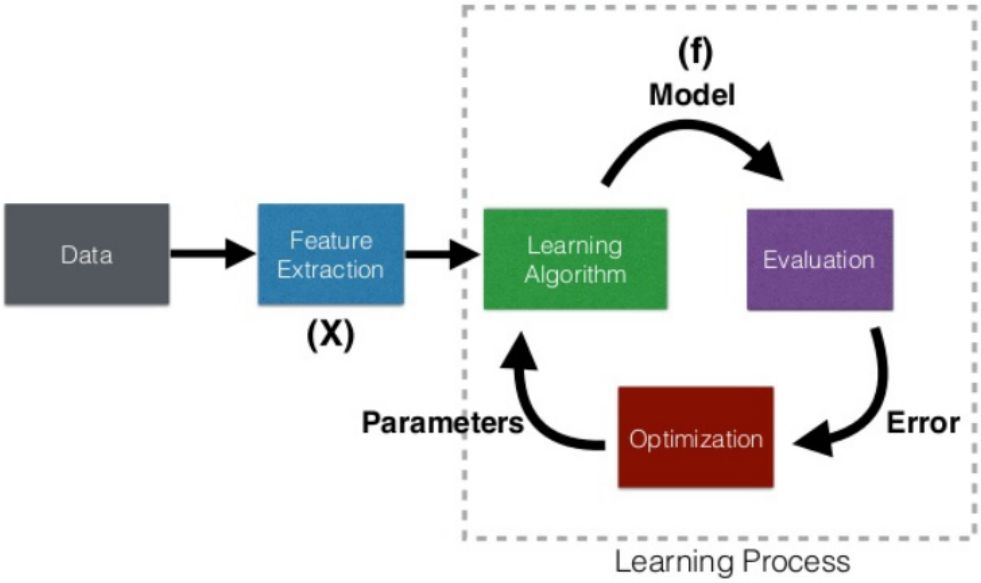

Figure 3.1: Learning Process

The primary objective of the learning process is to minimize the loss function. The loss function is typically a non-convex function with multiple local minima and a global minimum. Figure 3.2 illustrates our goal, which is to reach the global minimum.

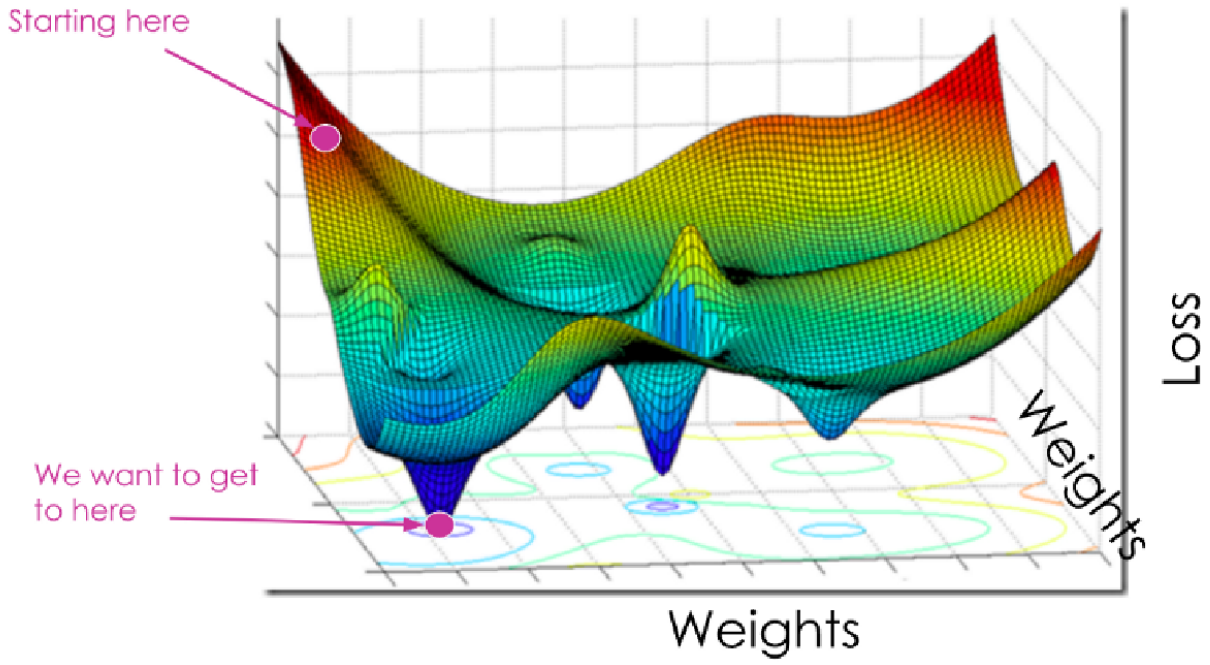

Figure 3.2: Goal of Training

For regression problems, a widely-used loss function is Mean Squared Error, defined as:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n denotes the training instances per batch, y is the true label, and $\hat{y}$ is the predicted value.

In the context of binary class classification problems, the binary cross-entropy loss function is popular and defined as:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1 - y_i) \log (1 \hat{-} y_i))$$

where n denotes the training instances per batch, y is the true label, and $\hat{y}$ is the predicted value.

3.1.2.5 Model Evaluation

In supervised machine learning, where labeled datasets are available, model evaluation is straightforward. A reliable model is characterized by a small loss score and the ability to generalize effectively on new, unseen data. This is why the Test set, an unseen dataset, is crucial during the evaluation process. Various evaluation metrics exist, with F1-score being one of the most popular, computed as $F1 = \frac{\text{True Positive}}{\text{True Positive} + \frac{1}{2}(\text{False Positive} + \text{False Negative})}$.

For unsupervised machine learning models, the evaluation task becomes more challenging. There does not exist an objective function that can measure how well clustering was performed. (Note that for compression tasks, there are ways to train the model on how well it reconstructs the original file.) Various methods exist to evaluate clustering performance, with one of the most popular being the Silhouette coefficient. This value is calculated using the mean distance of samples within the cluster a and the mean nearest cluster distance b for the same samples. The silhouette coefficient is then computed using the formula $\frac{b-a}{\max(a,b)}$, providing a measure of how well clustering was performed. The best possible value is 1, indicating that all samples are appropriately placed, while the worst value is -1, suggesting that samples have been poorly assigned to clusters.

3.1.3 Overfitting and Underfitting

In addition to model evaluation, machine learning practitioners must be careful of the challenges of overfitting and underfitting.

Overfitting occurs when a machine learning model becomes overly attuned to the intricacies of the training data, compromising its ability to generalize to new, unseen data. This phenomenon is often observed when the model is trained for an excessive number of epochs or when the model's complexity surpasses the inherent complexity of the underlying data. In such cases, the model starts to capture noise and outliers in the training data, leading to poor performance on unseen examples.

Conversely, underfitting manifests when a model lacks the capacity to sufficiently capture the underlying patterns within the training data, resulting in poor performance on both the training set and new data. This inadequacy may stem from a model that is too simplistic or undertrained.

To illustrate these phenomena, Figure 3.3 provides a visual representation of overfitting, while Figure 3.4 depicts the relationship between training data and error.

Figure 3.3: Underfitting, Derired and Overfitting

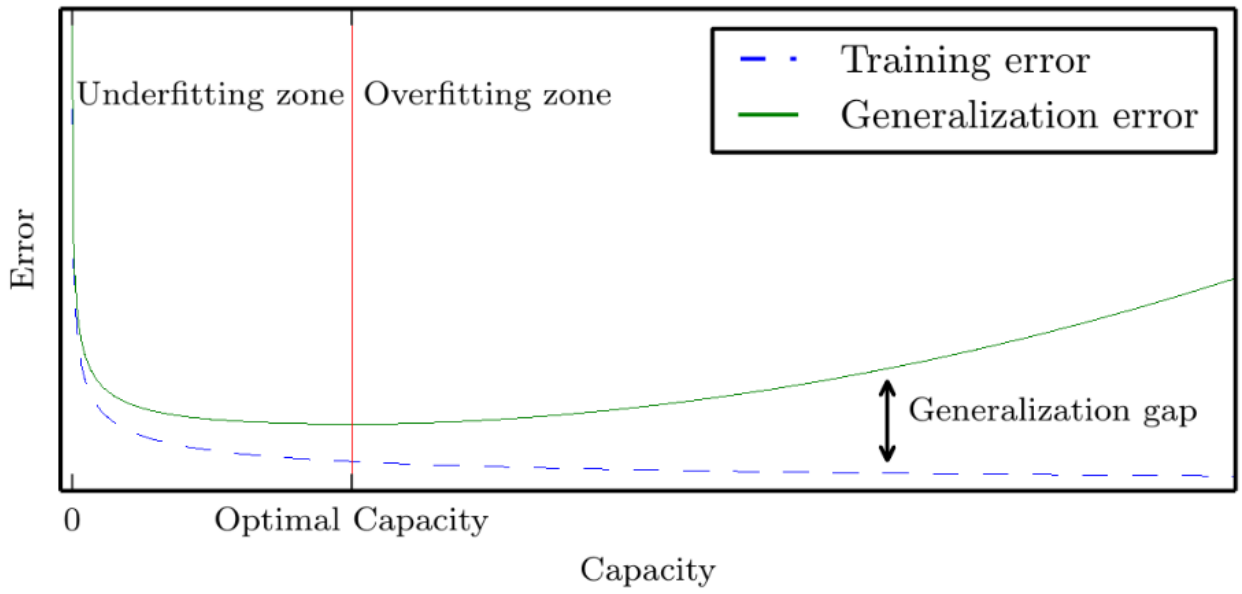

Figure 3.4: Relationship between train-data and error

3.2 Machine Learning Algorithms

3.2.1 Approaches in Machine Learning algorithms

3.2.1.1 Gradient-Based Algorithms

Gradient-Based Algorithms constitute a fundamental category within optimization algorithms, aiming to minimize an objective function $J(\theta), \theta \in \mathbb{R}^d$, where θ signifies the model's parameters as described in [34]. The optimization process involves iteratively updating these parameters in the direction of the gradient of the objective function, denoted as $\nabla_\theta J(\theta)$. Each iteration entails traversing the slope of the algorithm, seeking both local and ideally global minima. A significant factor in these algorithms is the learning rate, represented by

η. The learning rate determines the size of the steps the algorithm takes in navigating the slope towards the minima, influencing the convergence and efficiency of the optimization process.

There exist three variants of gradient descent optimization algorithms. The simplest one, known as Batch Gradient Descent, computes the cost function for the entire training set and then updates the parameters using the rule $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$. Although effective, this approach is memory-intensive as it requires loading the entire dataset for training, and it is not suitable for online learning.

Stochastic Gradient Descent improves upon this by performing parameter updates for each training example $(x^{(i)}, y^{(i)})$ individually: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^{(i)}, y^{(i)})$. This results in faster training and is well-suited for online learning. However, the constant updating introduces high variance, causing the objective function to fluctuate significantly.

To address the variance issue, Mini-Batch Gradient Descent combines the best of both worlds by updating parameters for every mini-batch of n training points: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^{(i:i+n)}, y^{(i:i+n)})$. This approach strikes a balance, reducing variance in parameter updates and providing stable convergence, solving the primary problem associated with the stochastic approach.

3.2.1.2 Hyperplane-Based Algorithms

Hyperplane-based algorithms aim to determine the decision boundary $y = f(\mathsf{x}) \in \mathcal{H}$, where $\mathcal{H}$ represents the hypothesis class, with the objective of minimizing the loss function and ensuring that, when presented with new data from the same distribution, the expected loss remains small. [23] The goal is to find parameters w such that $y = \text{sign}(f_w(\mathsf{x})) = \text{sign}(\mathsf{w}^T\mathsf{x})$, where $y \in \{-1, 1\}$. The task involves identifying the parameter set that maximizes the margin between the two possible classes of y. The distance of x to the hyperplane $f_w(\mathsf{x}) = 0$ is given by $\frac{|\mathsf{w}^T\mathsf{x}|}{\|\mathsf{w}\|}$. Additionally, a bias factor b (commonly denoted as w_0) can be introduced. If included, w becomes orthogonal to the hyperplane $f_{w,b}(\mathsf{x}) = \mathsf{w}^T\mathsf{x} + b = 0$, and the distance of $f_{w,b}(\mathsf{x})$ to the hyperplane $f_{w,b} = 0$ is $\frac{-b}{\|\mathsf{w}\|}$. Bellow in figure 3.5 is a visual representation.

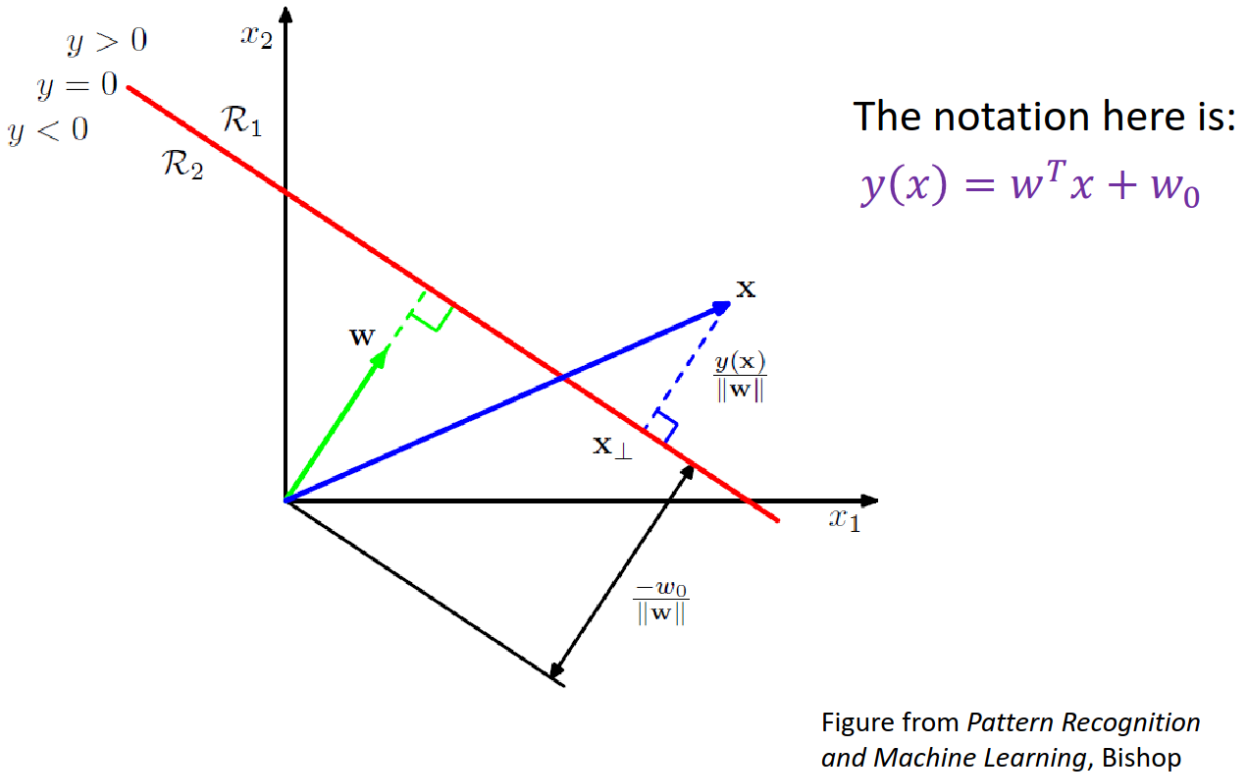

Figure 3.5: Margin With Bias

3.2.1.3 Tree Based Algorithms

Tree-based algorithms constitute a distinct class of machine learning methods employing hierarchical tree structures for decision-making and data classification. These algorithms recursively partition the feature space, creating subsets where each partition corresponds to different decision outcomes. In a decision tree 3.2.2.4, internal nodes represent feature tests, branches represent possible outcomes, and leaves signify the final decision. Notably, one of the key advantages of tree-based algorithms is their inherent interpretability. The decision-making process is transparent and easily traceable, allowing users to understand the logic behind the model's decisions.

3.2.2 Supervised Learning

3.2.2.1 Introduction to Supervised Learning

Supervised Learning is a machine learning paradigm aimed at acquiring the input-output relationship information of a system based on a provided set of paired input-output training samples, as defined by Qiong et al. [31]. In this paradigm, labeled data is essential, with the objective of learning the mapping between inputs and expected outputs of a system. This acquired knowledge enables the model to predict the output of new inputs. Common tasks within supervised learning encompass classification, where the model predicts the class label of input data, and regression, where the model's expected result is a continuous value.

3.2.2.2 Logistic Regression

Logistic Regression is a supervised machine learning algorithm, primarily employed for binary classification as defined in [20]. This algorithm incorporates a Linear Regression model, featuring a bias factor b and a weight vector $\mathbf{w}$ to compute the sum $z = \mathbf{x} \cdot \mathbf{w} + b$, where x represents the input vector containing the data features. To convert this result into a binary classification decision, it undergoes a sigmoid function transformation: $\sigma = \frac{1}{1+e^{-z}}$, ensuring the output lies within the inclusive range of $[0, 1]$. With the obtained probabilities for each class, the final decision is made: 1 if $\sigma(\mathbf{w} \cdot \mathbf{x} + b) > 0.5$, and 0 otherwise.

Parameter tuning in Logistic Regression involves optimizing the model parameters w and b using gradient descent 3.2.1.1, aiming to minimize the chosen loss function. By employing the binary cross-entropy loss (L_{CE}) as our objective, which is represented as:

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$$

where $\hat{y}$ is replaced by the sigmoid function used in decision-making, we ensure the model learns to accurately predict the binary outcomes.

3.2.2.3 Support Vector Machines (SVMs)

Building on the discussion in 3.2.1.2 and following the lecture slides [23, 24], the objective for Support Vector Machines (SVMs) is to maximize the margin over each data point i, where the margin is defined as $\gamma = \min_i \frac{y_i f_{\mathbf{w},b}(x_i)}{\|\mathbf{w}\|} = \frac{y_i(\mathbf{w}^T x_i + b)}{\|\mathbf{w}\|}$ (where $y_i \in \{-1, 1\}$), so $\max_{\mathbf{w},b} \gamma = \max_{\mathbf{w},b} \min_i \frac{y_i f_{\mathbf{w},b}(x_i)}{\|\mathbf{w}\|}$. This margin represents the distance of the data points from the decision boundary.

Solving this problem directly can be challenging, so SVMs introduce a fixed scale factor that does not affect the margin, leading to $y_i^*(\mathbf{w}^T x_i^* + b) = 1$, where x_i^* is the closest point to the hyperplane. Consequently, for all data points $y_i(\mathbf{w}^T x_i + b) \geq 1$, with at least one data point where equality holds, resulting in a margin of $\frac{1}{\|\mathbf{w}\|}$.

Support Vector Machines simplify the optimization problem to $\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$ subject to $y_i(\mathbf{w}^T x_i + b) \geq 1, \forall i$.

The optimization problem is solved using the Lagrange multiplier method:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i a_i[y_i(\mathbf{w}^T x_i + b) - 1]$$

where a is the Lagrange multiplier.

This problem can be reduced to $\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \sum_i a_i - \frac{1}{2} \sum_{i,j} a_i a_j y_i y_j x_i^T x_j$, subject to $\sum_i a_i y_i = 0,\ a_i \geq 0$.

Since $\mathbf{w} = \sum_i a_i y_i x_i$, this gives us $f_{\mathbf{w},b} = \sum_i a_i y_i x_i^T \mathbf{x} + b$.

One last thing to add is the Kernel trick, which maps features from the non-linear feature space to a linear one. This is particularly useful for handling non-linearly separable data, as illustrated in the XOR problem in Figure 3.6.

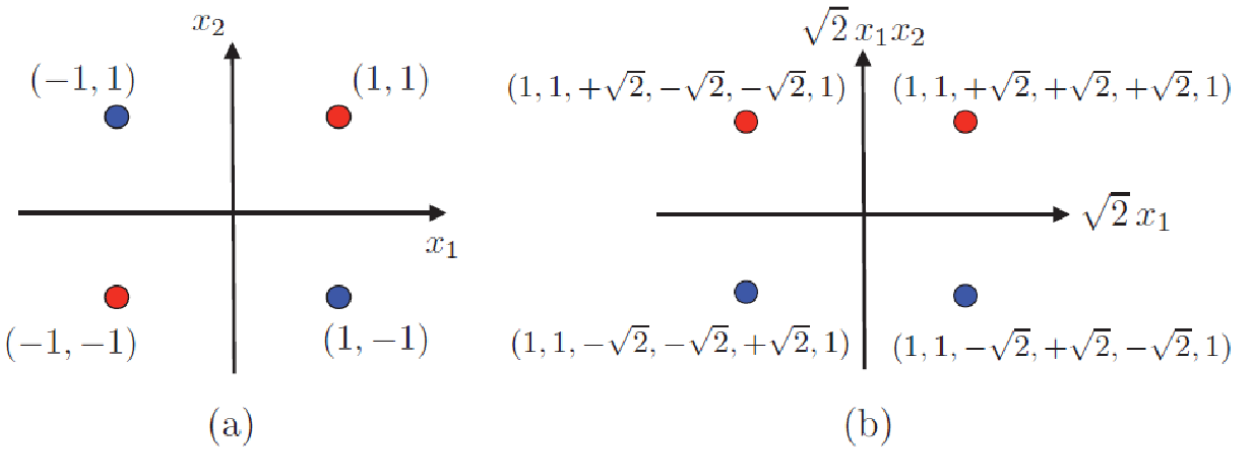

Figure 3.6: (a) Non-Linear Separable Problem, (b) Linearly Separable using a Polynomial Kernel of second degree

3.2.2.4 Decision Trees

Focusing on the binary classification problem and following the discussion in 3.2.1.3, we formalize the representation of decision trees following the paper [19]. The decision tree classifier is a function f that maps the data point x to the hypothesis space $\mathcal{H}$. Each data point x contains m features denoted by x_i.

The decision tree $\mathcal{T}$ is a directed acyclic graph that has at most one path between every pair of nodes. There always exists a root node with no incoming edges in contrast to all the other nodes of $\mathcal{T}$. Every non-leaf node is associated with a feature x_i, each outgoing edge is associated with one or more values of the feature space, and each leaf node is associated with a value from the hypothesis space. You can find an example using Decision Trees from scikit-learn [29, 1] in Figure 3.7.

Each path in $\mathcal{T}$ follows the nodes and edges according to the values of the features of the input until it reaches a terminal leaf node that determines the classification from the hypothesis space.

To construct a decision tree from training vectors $\mathrm{x}_i \in \mathbb{R}^m$ and corresponding label vector y with n samples, a recursive partitioning of the feature space is performed. Each node $\mathcal{Q}_m$, representing a subset of samples, is split based on a selected feature j and threshold t_m. This process continues recursively, creating left ($\mathcal{Q}_m^{left}(\theta)$) and right ($\mathcal{Q}_m^{right}(\theta)$) subsets at each node. The decision to split is based on minimizing an impurity or loss function $H()$, and the quality of the split is measured by $G(\mathcal{Q}_m, \theta) = \frac{n_m^{left}}{n_m} H(\mathcal{Q}_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(\mathcal{Q}_m^{right}(\theta))$. The aim is to find optimal parameters θ^* that minimize the impurity. The recursive splitting continues until a predetermined maximum depth is reached or a node contains samples from a single feature space.

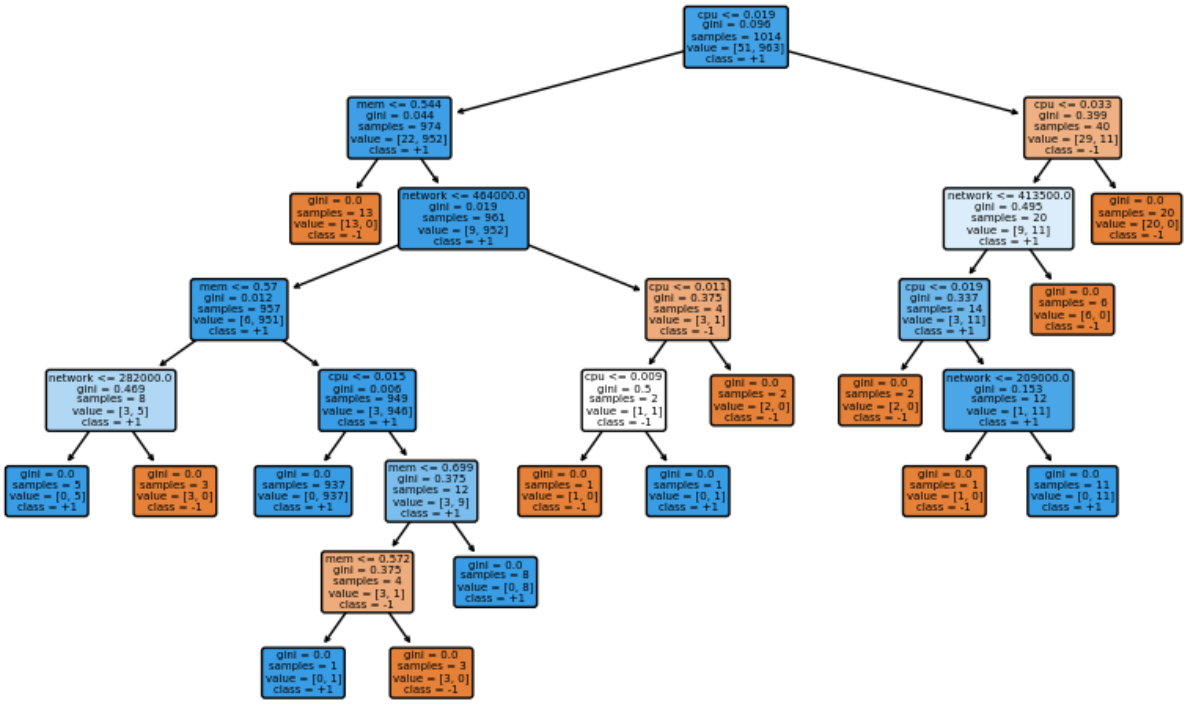

Figure 3.7: Decision Tree

3.2.3 Unsupervised Learning

3.2.3.1 Introduction to Unsupervised Learning

Unsupervised learning is a machine learning technique that, in contrast to supervised learning, does not rely on labeled data. As a result, classification and regression tasks are not applicable, given the absence of expected values without labels. The primary objective of unsupervised learning is to uncover hidden and meaningful patterns within the data. By identifying these patterns, clustering can be performed to group together data points with similar characteristics. An extension of clustering involves anomaly detection, where outliers in the data are identified. Unsupervised learning is also employed for dimensionality reduction and the representation of data in latent spaces.

3.2.3.2 Isolation Forests

Isolation Forests represent an unsupervised machine learning approach designed for outlier detection. The underlying concept of Isolation Trees is based on the observation that anomalies comprise very few instances, and their feature values significantly differ from those of normal instances. Consequently, anomalies tend to be closer to the root of the trees, while normal points are isolated deeper within the tree structure. [25]

The Isolation Forest method constructs a collection of isolation trees for the given dataset. Anomalies are characterized as points with shorter average path lengths from all the trees in the forest.

The user only needs to specify the number of isolation trees and the subsampling size. The algorithm performs well with a small number of trees and converges rapidly.

To partition points in each tree, random partitioning of the feature space is employed, with normal points typically requiring more partitions to be isolated compared to anomalies.

In figure 3.8 we can see how normal point x_i needed a significant amount of partitions to be isolated. This contrasts the anomaly point x_0 that in a few partitions was isolated.

After the creation of multiple isolation trees, the average of the depth of each point is taken in order to reveal the outliers.

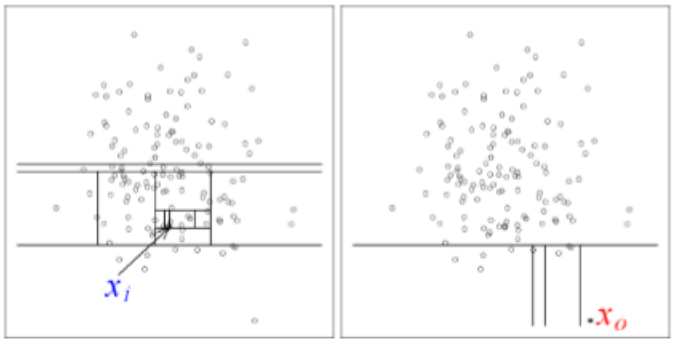

Figure 3.8: Partitions required to isolate normal point x_i in contrast to anomaly x_0

3.2.3.3 One-Class Support Vector Machines (SVMs)

In machine learning, One-Class Support Vector Machines (SVMs) have an important role at anomaly detection. They were first described in the paper [13]. Consider a training set $\mathcal{X}$ comprising n observations, each characterized by m features. In addition, consider a feature map Φ, that maps the original space to a higher-dimensional feature space F. Accompanying this, a kernel function K is employed, offering flexibility in the choice of mapping, ranging from a simple dot product kernel to more sophisticated alternatives like the Gaussian Kernel.

The primary objective of a one-class SVM is to define a function f that effectively maps the majority of points to the positive class (+1), while isolating potential outliers as the negative class (-1), based on their relative positions to a hyperplane.

To achieve this, a quadratic program must be solved, introducing a variable $v \in (0, 1)$ and a slack variable vector ξ:

$$\min_{\mathbf{w}\in F, \xi\in\mathbb{R}^m, \rho\in\mathbb{R}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{vl}\sum_i \xi_i - \rho$$

subject to

$$(\mathbf{w}\cdot\Phi(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0$$

Upon solving this optimization problem and obtaining the parameters $\mathbf{w}$ and ρ, the resulting function $f(\mathbf{x}) = \text{sgn}((\mathbf{w}\cdot\Phi(\mathbf{x})) - \rho)$ classifies the majority of x_i points as positive.

The parameter v in the one-class SVM equation controls the sensitivity of the algorithm. Fine-tuning v allows for customization of the model's performance to strike an optimal balance between identifying outliers and minimizing false positives.

Applying the dot product Kernel function, the classification equation takes the form $f(\mathbf{x}) =$

$\text{sgn}(\sum_i a_i K(x_i, \text{x}) - \rho)$, where a_i represents non-negative support vectors. The determination of these coefficients is realized by solving the following optimization problem:

$$\min_a \frac{1}{2} \sum_{i,j} a_i a_j K(x_i, x_j)$$

subject to

$$0 \leq a_i \leq \frac{1}{vl}, \quad \sum_i a_i = 1$$

3.2.4 Scikit-learn

Scikit-learn is a comprehensive framework designed to deliver high-level implementations of machine learning algorithms. [29, 14] Focused on Python, it provides constantly updated, state-of-the-art implementations of various algorithms. Notable for its commitment to high code quality, Scikit-learn undergoes rigorous unit testing, adheres to a specific writing style, and adopts the permissive BSD licensing model. With an emphasis on an easy-to-use API and community-driven development, Scikit-learn aims to make machine learning accessible to a broad audience. The project further distinguishes itself through robust documentation, ensuring maximum usability and understanding.

In Listing 3.1, you can see the ease with which we created an SVM model using Scikit-learn.

```
import numpy as np
from sklearn import svm
from sklearn.metrics import classification_report

# Create an SVM classifier with specified hyperparameters
clf = svm.SVC(kernel='rbf', C=3.9153433857642326, gamma='scale')

# Train the classifier
clf.fit(Xtrain, Ytrain)

# Make predictions on the test set
Ypred = clf.predict(Xtest)

# Print classification report
print(classification_report(Ytest, Ypred, target_names=genre_mapping.
    values()))
```

Listing 3.1: Scikit-learn SVM Classifier Example

3.3 Hyper Parameter Optimization

3.3.1 What is Hyper Parameter Optimization

When creating a Machine Learning model, the user has to make decisions on a wide range of hyper parameters that significantly influence the model's performance.These parameters span a spectrum of choices, ranging from fundamental structural decisions, such as

the complexity of the model architecture, to training-specific factors, such as the learning rate.

The influence of hyperparameters on model performance cannot be overstated. Their values govern the learning dynamics of the algorithm during the training process and, consequently, impact the model's ability to generalize patterns from the data. Minute adjustments to these parameters can yield substantial variations in the model's learning capacity.

The primary objective of hyperparameter optimization is to discover the optimal set of hyperparameter values. The ideal configuration is one that enables the model, during the training phase, to converge towards the global minimum of the loss function.

In the spirit of computational efficiency and best practices in machine learning, the manual exploration of hyperparameter spaces is often impractical. As computer scientists, our goal is to automate and, when feasible, parallelize this intricate process. Automation streamlines the search for optimal hyperparameters, while parallelization leverages the power of modern computing resources, substantially reducing the time required for experimentation.

In the context of this thesis, an analysis of algorithms described in the paper by Yu et al. [37] will be conducted.

3.3.2 Search Algorithms For Hyper Parameter Optimization

3.3.2.1 Grid Search

Grid Search stands as the simplest hyperparameter optimization algorithm. In this method, the user explicitly defines the values each hyperparameter can take, and the algorithm systematically explores all possible combinations. Notably, Grid Search guarantees the discovery of optimal hyperparameters within the specified search space on each run. However, it faces challenges in scalability due to the curse of dimensionality, as the search space expands exponentially with each additional hyperparameter. Despite this limitation, Grid Search exhibits ease of parallelization, as each combination is independent of others.

3.3.2.2 Random Search

Random Search, another hyperparameter optimization algorithm, explores the hyperparameter space through a random distribution. This approach involves a search conducted within a specified time or iteration budget, allowing for the examination of various hyperparameter combinations. In contrast to Grid Search, which allocates a fixed budget for each hyperparameter, Random Search offers flexibility in distributing its computing resources. This adaptability proves particularly valuable for non-uniformly distributed parameter ranges.

Following the principles of Monte Carlo techniques, Random Search efficiently utilizes an increased budget, increasing the likelihood of discovering an optimal combination within the search budget. Moreover, Random Search is amenable to parallelization and finds particular utility in the initial stages of hyperparameter optimization. It helps identify a preliminary range of optimal parameters, after which other algorithms are often employed to fine-tune and approach the minimum of the loss function more closely.

3.3.2.3 Bayesian Optimization

Bayesian Optimization, a traditional algorithm with applications in various optimization problems, exhibits a key strength in its enhanced accuracy with more data. Operating as a sequential method, it lacks parallelizability but compensates by converging rapidly within a limited number of trials. This algorithm effectively balances exploration—initiating trials based on new data to discover potential global minima—with exploitation, leveraging existing data to make informed decisions and refine the search.

In the realm of hyperparameter search, Bayesian Optimization dynamically updates its probability distribution after each trial, leveraging information from prior trials. This adaptive approach significantly accelerates the search process compared to exhaustive methods like grid and random search, which exhaust their budget without the same adaptive efficiency.

3.3.2.4 Tree Parzen Estimators (TPE)

Tree Parzen Estimators (TPE) represent a powerful extension of Bayesian Optimization, specifically designed to handle categorical and conditional hyperparameters. TPE employs a tree-structured approach to model the objective function, efficiently navigating complex hyperparameter spaces. This algorithm iteratively refines its search by balancing exploration and exploitation, similar to Bayesian Optimization. TPE is particularly adept at handling discrete and conditional parameters, making it a valuable asset in scenarios where traditional optimization algorithms might face limitations.

3.3.3 Optuna

Optuna stands as a cutting-edge, open-source optimization framework, designed for seamless handling of experiments, whether small or large, with minimal setup requirements. It was first introduced in 2019 in the study [11]. Being open source, it continuously integrates the latest optimization algorithms and advancements in the field. Optuna offers a dynamic approach to constructing the search space, incorporating versatile sampling and pruning algorithms, while allowing users the flexibility to define their own. It simplifies the optimization process for various tasks, aiming to minimize or maximize a user-defined objective function.

Optuna supports diverse parameter types, such as integer, floating-point, or categorical, and, through trials, adapts its strategy by identifying underlying relationships and concurrencies among hyperparameters. The framework implements a pruning algorithm based on successive halving to enhance efficiency. Additionally, Optuna provides versatility in execution environments, with options for user-specified storage backends and compatibility with platforms ranging from Kubernetes clusters to simple Jupyter notebooks.

In Listing 3.2, the simplicity of creating an Optuna study for tuning hyperparameters in SKLearn's SVM model is evident. The code defines an objective function that takes hyperparameters as trial suggestions, trains an SVM model, and evaluates its performance using the F1 score. The Optuna study then optimizes this objective function, iteratively exploring the hyperparameter space over 50 trials. Finally, the best hyperparameters are printed, showcasing Optuna's effectiveness in automating the hyperparameter tuning process.

```python
import optuna
from sklearn import svm
from sklearn.metrics import f1_score

def objective(trial):
    kernel = trial.suggest_categorical('kernel', ['linear', 'rbf'])
    c = trial.suggest_float('c', 1e-2, 10, log=True)
    gamma = trial.suggest_categorical('gamma', ['scale', 'auto'])

    clf = svm.SVC(kernel=kernel, C=c, gamma=gamma)
    clf.fit(Xtrain, Ytrain)
    Ypred = clf.predict(Xtest)
    f1 = f1_score(Ytest, Ypred, average='macro')
    return f1

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)
print(study.best_params)
```

Listing 3.2: Optuna Hyperparameter Optimization

4. RELATED WORK

In this chapter, we conduct a thorough examination of existing literature and research that serves as the bedrock for our study on anomaly detection in Kubernetes. This review offers a deep insight into the state-of-the-art in anomaly detection for Kubernetes. The works are categorized based on two crucial dimensions: anomaly detection for resource management and security threats. This dual categorization enables us to unravel cutting-edge methodologies and techniques, providing a comprehensive understanding of how the field has evolved. The subsequent sub-sections delve into each thematic area, providing unique insights and ideas contributing to the collective knowledge in this field.

4.1 Resource Management

4.1.1 Reference Net-Based Performance and Management Model for Kubernetes

Medel et al. introduced a Reference Net-based performance and management model for Kubernetes, as outlined in their work [28]. A Reference Net, belonging to the family of Petri nets, serves as the foundational modeling tool. Petri nets, recognized as mathematical modeling languages, facilitate the depiction of distributed systems. Graphically, Petri nets consist of nodes representing places (system states) and transitions (actions or events affecting the system). Preconditions, depicted by arcs from places to transitions, must be fulfilled to trigger transitions, while opposite arcs represent post conditions that have already been met. This modeling approach specifically captures the pod life cycle, highlighting the nuances of deploying and running applications within containers inside the pod.

Figure 4.1 illustrates the container life cycle using Object Nets, a subtype of Petri nets. The timed transitions (T_i) in the figure delineate critical stages in the model, such as T1 representing the time to create a container, and T6 and T7 representing the times for termination.

To evaluate the proposed approach, micro-benchmarks were conducted across diverse scenarios, including CPU-intensive, Input/Output, and Network-intensive applications. The results not only demonstrate optimal pod and container allocation strategies for minimizing overhead costs per application type but also provide valuable insights into capacity planning and resource management.

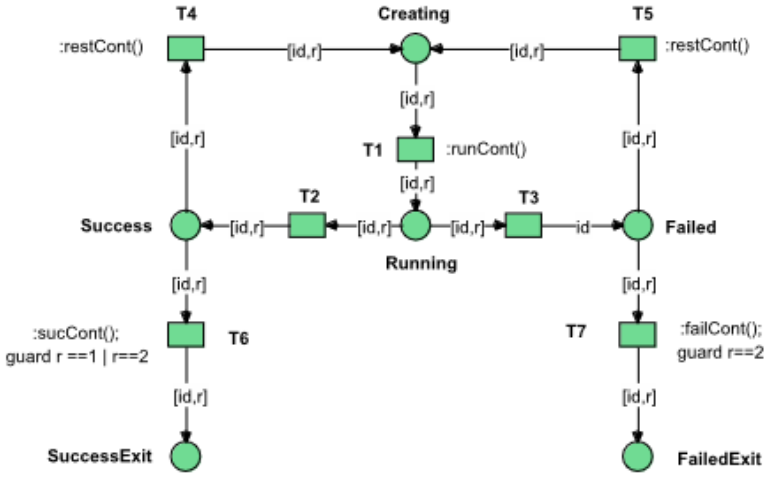

Figure 4.1: Model of the life cycle of a Container [28]

4.1.2 Detection of Cluster Anomalies using ML Techniques - Kubernetes Anomaly Detector

Kosińska et al. present the Kubernetes Anomaly Detector system (KAD) in their recent work [22]. The architecture of KAD is characterized by simplicity and intuitiveness. It initiates with data acquisition from a designated source, preprocesses the data to enhance its suitability for modeling, engages in model training, explores hyperparameters, compares various models, and ultimately selects the most promising one. Subsequently, the chosen and trained model is employed for anomaly detection on future data batches. Figure 4.2 provides a visual representation of this workflow.

For model training and evaluation, a semi-supervised approach is adopted, assuming the training set is free from anomalies. The user can define the selected model, or it can be determined based on its performance on the training set relative to other options. The authors consider four models for selection: Seasonal Autoregressive Integrated Moving Average, Hidden Markov Model, Long Short-Term Memory, and Autoencoder. The former two are statistical models, while the latter are based on Neural Networks. A simplified architecture of KAD is illustrated in Figure 4.3.

To assess their work, the authors split the evaluation procedure into two parts. The first part focused on model selection, utilizing the Numenta Anomaly Benchmark, which provided a labeled dataset. All models yielded satisfactory results in both artificial and real-world data, with each excelling on different datasets, underscoring the value of the model selection mechanism. The second part scrutinized response latency in a production-ready cluster. The open-source demo Sock Shop application, coupled with simulated load generators, was employed for this analysis. The results revealed that statistical models exhibited significantly smaller training times than their Neural Network counterparts, rendering them favorable choices for the given scenario. The study concluded that the preference for different ML models can be contingent on the input, emphasizing the utility of a model selection component.

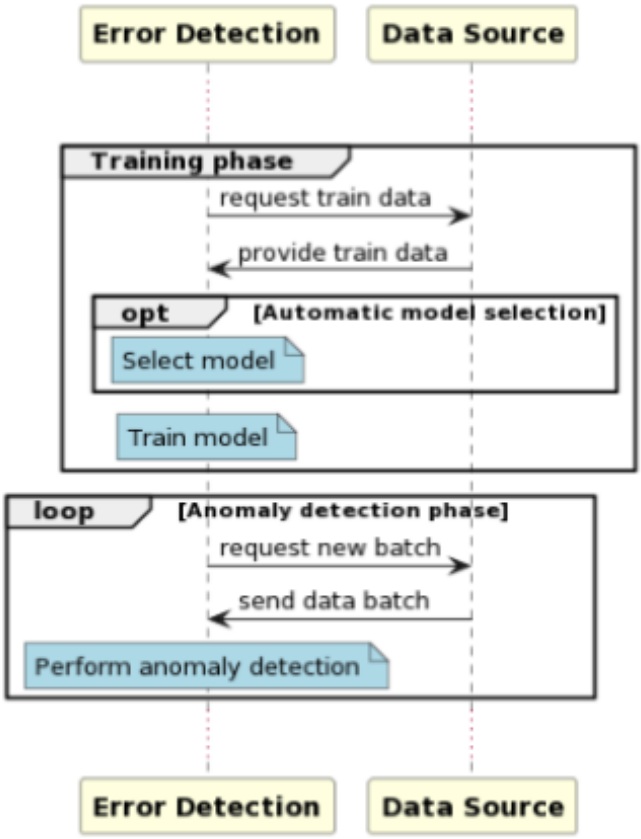

Figure 4.2: Sequence Diagram of Anomaly Based Error Detection

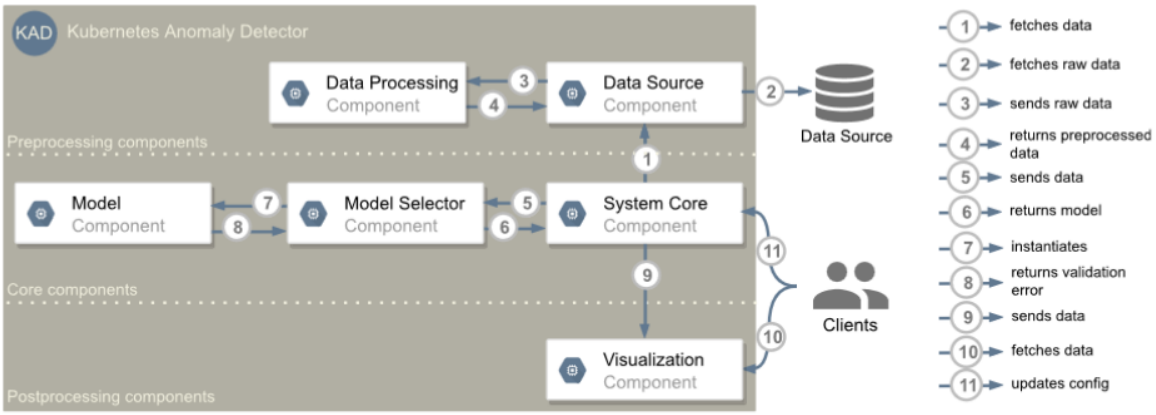

Figure 4.3: Simplified Architecture of KAD

4.1.3 Anomaly Detection and Diagnosis for Container-based Microservices with Performance Monitoring

Qingfeng et al. introduce their Anomaly Detection System in [16], comprising three key modules. The monitoring module gathers performance data from the target system, while the data processing module analyzes this data to detect anomalies. Additionally, a fault injection module simulates service faults and generates datasets for training and validating machine learning (ML) algorithms in anomaly detection.

The monitoring agent is deployable across multiple clusters with containers, monitoring the performance and metrics of individual containers. In this context, each container represents an entity hosting a single microservice, and Container Monitoring focuses on the monitoring of individual containers. Notably, a microservice may be deployed in multiple containers, and Microservice Monitoring aggregates data from all containers running the specific microservice.

The data processing component is responsible for both detecting and diagnosing anomalies. It classifies whether a microservice exhibits anomalous behavior and identifies the specific container responsible for the discrepancy. Anomalies encompass CPU hogging, memory leaks, package losses, and network latency. Supervised ML algorithms are employed for anomaly detection, trained using load generators to simulate user requests. The fault injection module simulates anomalous behavior for training purposes. Once the model is trained, predictions are made based on time series data from each container. When a microservice is identified as anomalous, all containers running that microservice are analyzed using the Dynamic Time Warping algorithm to pinpoint the anomaly.

The experiments involve the use of various algorithms, including Support Vector Machine (SVM), Nearest Neighbor Classifier, Naive Bayes, and Random Forest. The Nearest Neighbor algorithm demonstrated superior performance in most cases.

The fault injection module plays a crucial role in creating ML training and validation datasets. Each container is equipped with an injection agent simulating CPU, memory, and network faults. The resulting datasets include labels for normal behavior and various anomaly types based on fault behavior.

In summary, Qingfeng et al.'s Anomaly Detection System proves effective in monitoring and diagnosing anomalies in container-based microservices, leveraging a combination of supervised ML algorithms, fault injection for training, and innovative anomaly detection methodologies.

4.2 Detection of Events in Cloud Infrastructure

4.2.1 Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster

Cao et al. propose an innovative solution for detecting anomalies within a Kubernetes Cluster, particularly in NetFlow data, in their work [15]. Their approach utilizes a Probabilistic Deterministic Finite Automaton (PDFA) to model the runtime characteristics of the cluster and identify potential attacks. Each transition in the PDFA is associated with a probability, allowing the measurement of sequence probabilities using the multiplication rule.

To learn the cluster's modeling, state merging algorithms are employed. These algorithms initialize with a state machine structured like a tree and iteratively merge states considered similar. The merging process concludes when no similar states persist. Figure 4.4 illustrates an example of a state merge operation.

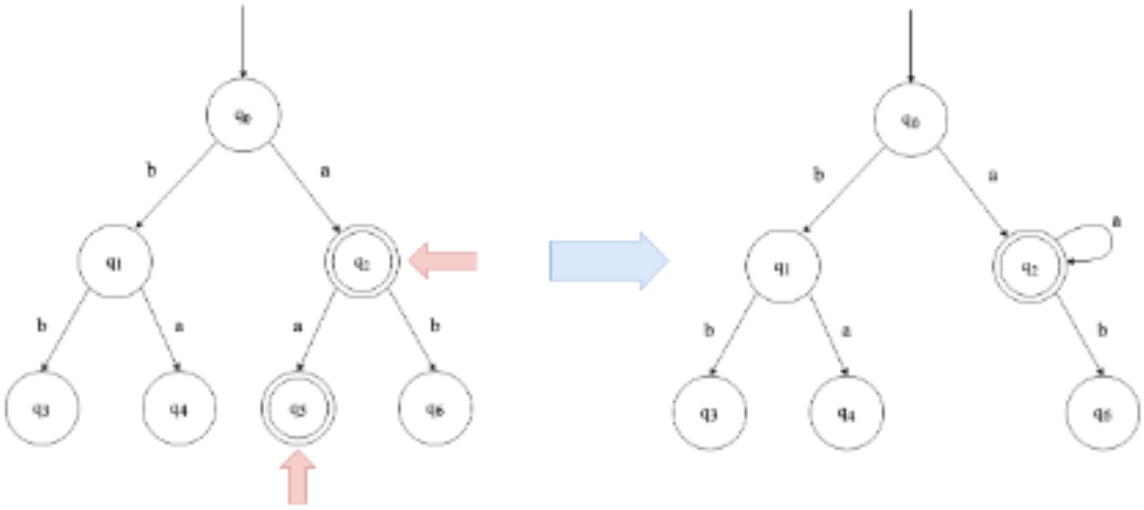

Figure 4.4: Merge Operation Performed in the State Machine [15]

NetFlow data, describing the communication behavior between components, serves as features for the model. Once the PDFA is created and trained, likelihood probabilities are applied to new traces generated by Kubernetes cluster data. Any trace with a low probability is flagged as an anomaly.

The system underwent testing in various scenarios, encompassing remote code execution attacks, denial-of-service attacks, and malicious code in workloads. Training datasets were created using benign users and simulated attacks.

The results are promising, with F1 scores surpassing 0.9 in some instances.

4.2.2 KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches

Tien et al. present the KubAnomaly security monitoring system in their work [36]. KubAnomaly is designed for monitoring and detecting anomalous behavior at the container runtime, tailored for Kubernetes (K8s) use. The system aggregates monitoring logs from each container, extracts pertinent features, normalizes them, and inputs them into a 4-layer fully connected linear neural network. Non-linear activation functions and aggressive dropout layers between each layer are employed to mitigate overfitting. The model's output focuses on individual containers, labeling each as Normal or Anomaly. Re-evaluation of each container occurs every 10 seconds.

To assess the system, the authors created two datasets with varying levels of complexity, in addition to utilizing public datasets. The model achieved a commendable F1 score of 98.1%. In a real-world scenario, the authors tested KubAnomaly by deploying an application with security vulnerabilities, successfully detecting and notifying about potential attackers.

KubAnomaly operates with a modest overhead on the host machine, approximately 5%. This efficiency ensures minimal impact on the overall system performance.

4.3 Commercial Offerings

4.3.1 Service Now

Commercial solutions for anomaly detection in cloud-based systems are also prevalent. One such offering is integrated into the ServiceNow platform, a cloud-based suite providing a range of IT service management tools to streamline and automate diverse business processes. ServiceNow aids organizations in managing workflows, enhancing efficiency, and fostering collaboration across different departments, with a dedicated focus on anomaly detection within its platform.

According to ServiceNow's documentation [4], it monitors both cyclical and non-cyclical metrics, differentiating between those that repeat regularly in a defined order and period (cyclical) and those that exhibit irregular repetition (non-cyclical). In the context of anomaly detection, cyclical metrics include User Interface transaction counts, server response times, SQL response times, average user transactions processed over a one-minute period, and the maximum memory consumption at any given node.

Anomalies related to Jobs are also addressed, where metrics such as the number of scheduled and concurrently run jobs, along with their transaction counts, are monitored on an hourly basis. Anomalies are defined based on metrics using the standard deviation, and alerts are triggered accordingly.

Due to the closed-source nature of ServiceNow, specific details regarding the mechanics of anomaly detection on the metrics side are proprietary and not publicly disclosed.

4.3.2 Edge Impulse

Edge Impulse is a specialized platform designed to facilitate the development and deployment of machine learning models, enabling the seamless integration of AI capabilities into various applications. The platform simplifies the process of building and deploying embedded machine learning solutions tailored for edge devices such as sensors, microcontrollers, and other Internet of Things (IoT) devices.

Edge Impulse provides general-purpose anomaly detection algorithms [2], employing two distinct methods for detecting anomalies:

The first method utilizes K-Means clustering, wherein a predefined number of clusters group points based on their similarity. Anomalies are detected by establishing a threshold that identifies data points placed farther away from all cluster centers.

The second implementation involves a Gaussian Mixture Model, representing a probability distribution as a mixture of multiple Gaussian distributions. Each Gaussian component signifies a cluster of data points with similar features. Essentially, samples within the dataset are modeled by Gaussian distributions. If a data point is assigned a low probability of being generated by the Gaussian Mixture Model, it is labeled as an anomaly.

Edge Impulse notes that these techniques are preferred over neural networks due to their ability to handle situations where a data point doesn't belong to any specific existing class—an aspect that neural networks may struggle to effectively address.

5. ANOMALY DETECTION AND PREDICTIVE CLASSIFICATION IN KUBERNETES ENVIRONMENTS

5.1 Problem Description

In contemporary Kubernetes applications, resource optimization is a critical consideration to enhance operational efficiency. One challenge arises in the context of dynamic workloads where predicting the exact resource requirements for each component beforehand becomes inherently complex. Due to varying workloads, components may receive either excessive or insufficient resources, leading to operational inefficiencies or, worse, compromised functionality. To address this, our proposed solution leverages monitoring data and user-defined metrics to identify anomalies in the behavior of individual components within the Kubernetes environment. By establishing a baseline of normal behavior based on the majority of workflow components, our approach aims to detect deviations that may indicate potential issues or inefficiencies. This proactive anomaly detection mechanism is instrumental in signaling instances where additional resources should be allocated or further investigation is warranted, ensuring the seamless and reliable execution of dynamic workflows.

5.1.1 Dynamic Workflow Execution

The proposed anomaly detection and prediction component is strategically positioned within the same architectural layer as the entity responsible for creating, instantiating, and managing dynamic workflows. By residing in close proximity to the orchestrator, our solution has privileged access to real-time data streams and comprehensive insights into the evolving state of the Kubernetes environment. This strategic placement enables our system to seamlessly integrate with the workflow management process, allowing for timely analysis of anomalies and efficient adaptation to changing resource demands. This close coupling enhances the overall responsiveness and effectiveness of our anomaly detection mechanism, ensuring a holistic approach to dynamic workflow execution within the Kubernetes ecosystem.

5.1.2 Dynamic Workflow Graphical Representation

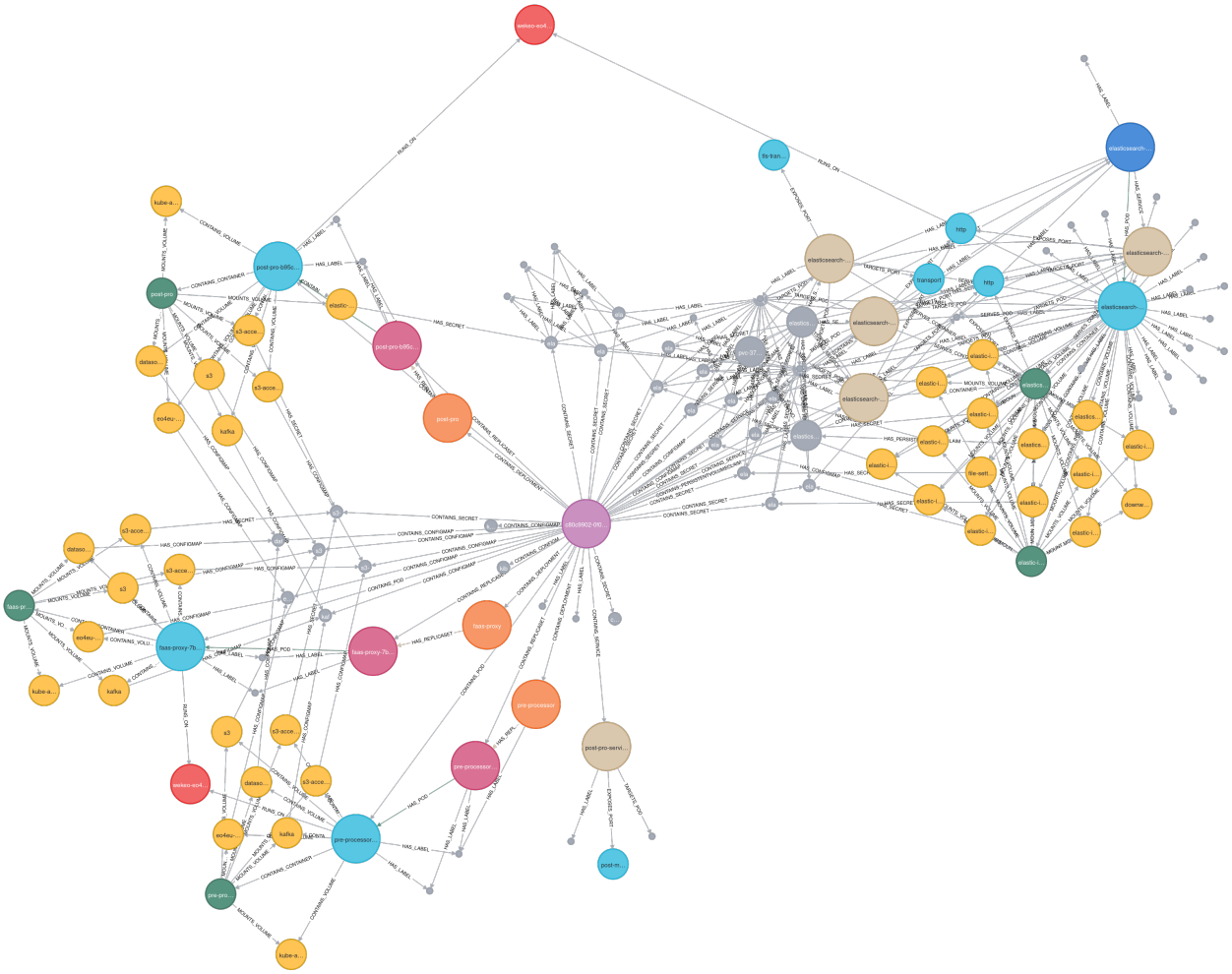

Figure 5.1: Visualization of Workflow Components in Kubernetes

Figure 5.3 illustrates a graphical representation of Kubernetes components employed in a sample Workflow, using their representations in the Neo4j database. The red node at the top signifies the Node the workflow runs on. The pink node represents the dedicated Namespace created to encapsulate and isolate the Workflow. Deployments, depicted in orange, house the applications that are currently running. Associated with each deployment are purple Replica Sets, ensuring redundancy and scalability. In deep blue we can find the Stateful Sets containing applications that require persistent storage. Each big blue node corresponds to a Pod, while green nodes denote Containers. Volumes, Ports, and Services are depicted in yellow, (small) blue and base respectively, highlighting their crucial roles in inter-component communication within the Kubernetes cluster. Edges symbolize the connections between these components. This intricate orchestration of Kubernetes components culminates in the successful execution of the initiated Workflow.

5.1.3 Management of Kubernetes Components

As illustrated in Figure 5.3 and discussed in the previous subsection, Neo4j (2.6) serves as a pivotal tool for visualizing and managing the dynamic landscape of Kubernetes components. Each node in the Neo4j graph encompasses key metrics crucial for informing

the anomaly detection algorithms. These metrics encompass parameters such as CPU and Memory consumption, along with networking information.

Upon the initiation of a workflow, the initiating component notifies the graph manager component of the newly instantiated components. Subsequently, the graph component queries the Kubernetes API to obtain detailed information on the new deployments. This information is then utilized to update the Neo4j graph, providing an accurate representation of the current state of the cluster. Similar procedures are undertaken when a workflow concludes, resulting in the removal of the associated components.

It is essential to note that a Kubernetes component is inherently dynamic, with its attributes continually evolving. Pods may be replaced, and the CPU consumption of a Container undergoes constant fluctuations. To address this dynamism, the monitoring component of the cluster assumes a crucial role. This component collects real-time metrics and ensures the timely update of the Neo4j graph, thereby maintaining a current and comprehensive representation of the cluster.

Figure 5.2 showcases a segment of the information stored in a Node component within the Neo4j graph, exemplifying resource capacity as one of the metrics.

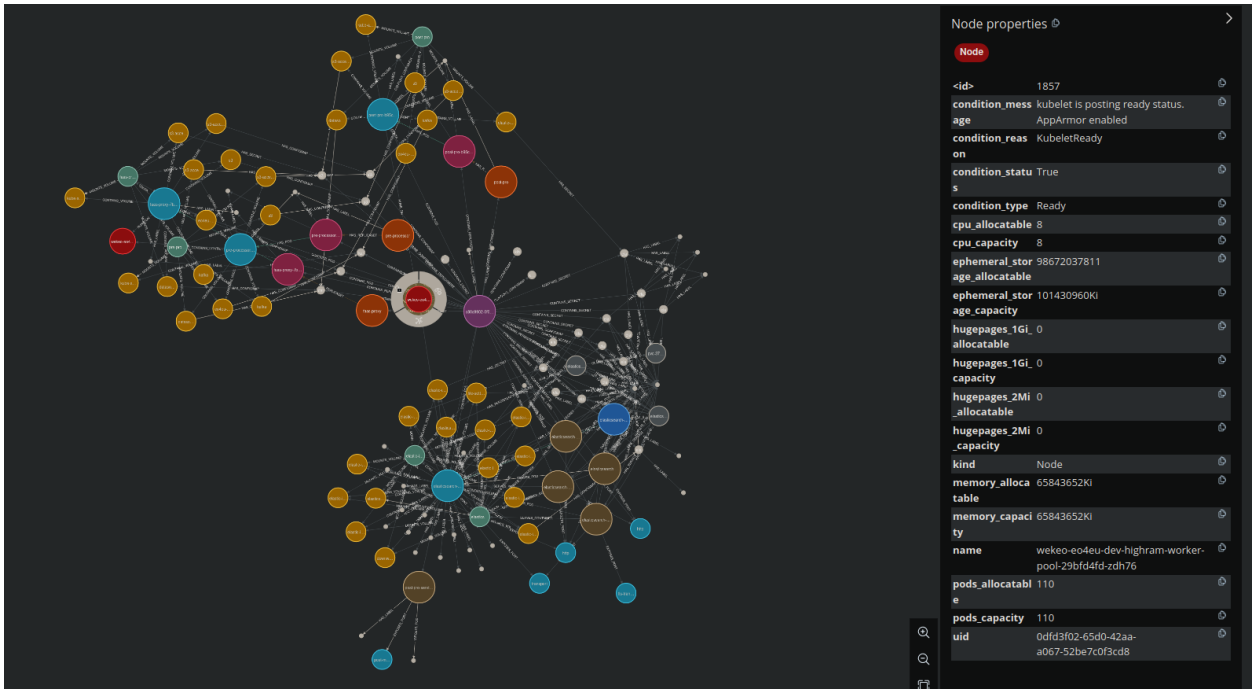

Figure 5.2: Part of the Information the Node component of the graph contains

5.1.4 Anomaly Detection and Monitoring Strategy

With a real-time representation of the Kubernetes cluster's state facilitated by Neo4j (2.6), our focus shifts towards actively monitoring the cluster to identify anomalous behaviors. Such anomalies may not only impact the immediate operation of a particular component but also have cascading effects on interconnected components.

The purpose of our monitoring strategy is to efficiently allocate resources by concentrating on specific areas of the cluster exhibiting anomalous behavior. It is crucial to emphasize that the classification of a component as anomalous does not inherently signify a malfunction; rather, it indicates deviations in behavior compared to other components.

Our ultimate objective is to assign a numerical value between 0 and 1 to each node in the graph, reflecting the degree of anomaly for that specific component. A value of 0 denotes a non-anomalous component, while a value of 1 signifies a definite anomaly. Additionally, our visualization aims to accentuate anomalous areas, with higher numerical values corresponding to darker colors. Unaffected, non-anomalous components remain unmarked. An example of a given anomaly score can be found in figure 5.3

Given the absence of domain expertise to precisely define anomalous behavior and train supervised models, alternative strategies must be employed to discern and address anomalies within the Kubernetes cluster.

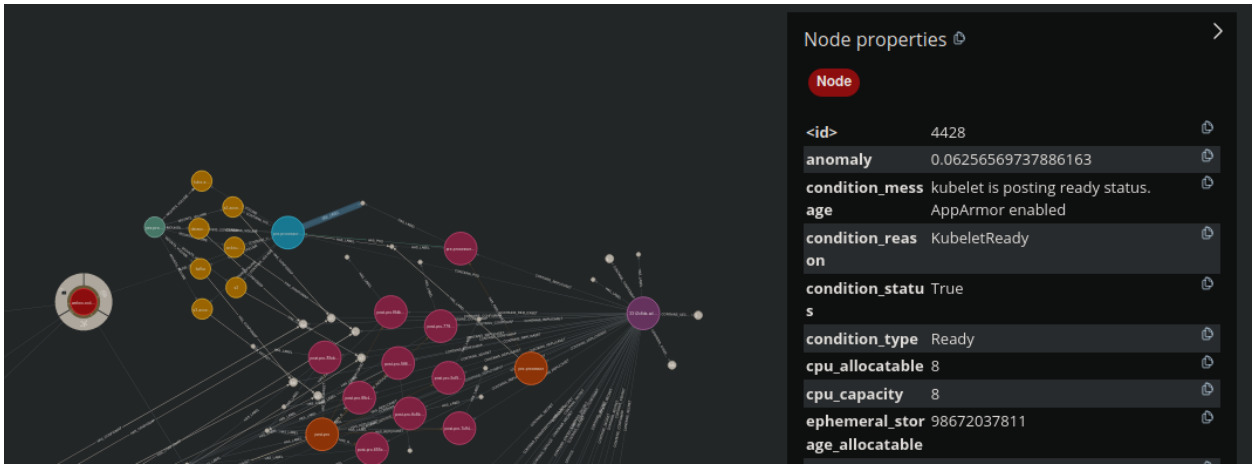

Figure 5.3: Anomaly Score Given to a Node in the Graph

5.2 Proposed Solution

5.2.1 Hierarchical Anomaly Detection Strategy

The graphical representation of the Kubernetes cluster, defined by its heterogeneous and dynamic characteristics, poses challenges for employing a conventional ML model. The presence of different component types, each with its unique metrics, and the dynamic nature of the graph, with components being added or removed continuously, make it impractical to input the entire graph into an ML model and expect individual anomaly scores for each component. Although obtaining an anomaly score for the entire graph is feasible, precisely pinpointing the source of anomalous behavior becomes computationally challenging due to the curse of dimensionality.

To address these challenges, our proposed solution involves splitting the anomaly detection process based on the types of components within the graph. This means that anomalies in Containers are evaluated against other Containers, Pods against Pods, and so forth.

Despite this segregation, we aim to leverage the graph structure for information storage. We conceptualize the graph as a tree, where Containers form the bottom layer and Nodes occupy the top. This tree structure allows information to flow from the bottom layer upwards. For instance, when classifying a Pod as anomalous, the anomaly status of its constituent Containers can be considered.

This hierarchical approach doesn't preclude information from flowing backward. The behavior of a Node, for instance, can influence the behavior of the Containers it hosts. In our

proposed solution, metrics can be shared among all connected components, but anomaly scores are propagated only from the bottom to an upper-level component.

Recognizing that not all components have specific metrics, some components, like Deployments, serve as abstraction layers for their underlying components (e.g., Replica Sets and Pods). Their role is to aggregate the anomaly scores of the components beneath them in the hierarchy.

5.2.2 Anomaly Detection and Classification Process

Having defined the target for machine learning (ML) analysis within the Kubernetes cluster, we delve into the anomaly detection and classification process. Due to the lack of domain expertise for creating a dataset with an accurate representation of normal behavior for each component type, our ML process is bifurcated into two distinct parts.

5.2.2.1 Unsupervised Part

In the unsupervised phase, we leverage observations of the target component and apply an unsupervised anomaly classification model. Potential models include an isolation forest (3.2.3.2) or a One-Class SVM model (3.2.3.3). Our objective is to utilize the existing observations of the graph to discern, for each component type, which observations deviate from the established norm. It is important to note that all observations could be deemed normal.

This process allows us to construct a dynamic dataset of normal and anomalous data points for each component type without prior knowledge of what constitutes anomalous behavior. The dataset evolves over time as new observations are incorporated, and the anomaly detection algorithms are periodically rerun to ensure accuracy. Rerunning the algorithms with an increased number of observations reduces the likelihood of expected behavior being classified as anomalous, as the observations of normal behavior accumulate.

5.2.2.2 Supervised Part

With a dataset of points labeled as normal or anomalous from the unsupervised part, we proceed to employ traditional classification ML models. Models such as Logistic Regression (3.2.2.2), Support Vector Machines (3.2.2.3), or decision trees (3.2.2.4) are considered. The goal is to perform two-class classification, designating points as normal or anomalous. Moreover, we seek to obtain the certainty of a point being classified as an anomaly, enabling us to visually distinguish and color that section of the graph.

Having trained the models on the output of the unsupervised phase, we periodically scrape the graph and update each node with an anomaly score ranging from 0 to 1. Each node type in the heterogeneous graph has its own dedicated supervised trained model for accurate predictions. This approach ensures the existence of many small, lightweight models capable of quickly and efficiently updating the graph at scheduled intervals.

As the unsupervised models evolve with additional data, the supervised models are also retrained to capture new anomalous points or reevaluate points that may have been observed frequently, potentially reclassifying them as non-anomalous.

5.3 Anomaly Detection and Prediction Workflow

The Anomaly Detection and Prediction Workflow encompasses two essential functions executed at user-specified time intervals: the ”update graph” and the ”update models” functions. The ”update graph” function is frequently called to refresh anomaly values and gather data, while the ”update models” function is invoked at longer intervals to retrain the models with new data. A simplified architecture is illustrated in Figure 5.4.

5.3.1 Update Models

Upon invocation of the ”update models” function, all components with detectable anomaly metrics undergo model updates. Components which solely aggregate anomaly scores, bypass this function.

The process involves loading comma-separated value files storing accumulated graph observations. Subsequently, the Unsupervised model is initialized and fitted with the loaded observations. The same model is used to label gathered data as anomalous or not. If there are no anomalies or all points are labeled as anomalies, a dummy classifier model that classifies all inputs using the same class is utilized. Alternatively, if both normal and abnormal labels exist, a Supervised model is trained using the dataset. The trained supervised model is then returned for making predictions on future observations.

As the model updating does not adhere to a hierarchical structure, the model of each component is updated in parallel, harnessing available resources for optimal efficiency.

5.3.2 Update Graph

When the ”update graph” function is invoked, the objective is to refresh the graph with the anomaly scores of the current observations. Starting from components at the bottom of the hierarchy (as described in Section 5.2.1), information is propagated upwards until reaching Nodes and Namespaces. Two types of operations take place: Classification-Prediction workflows and data aggregation workflows.

5.3.2.1 Components with Models

For components with models, the workflow commences by querying the database to obtain a dataframe with all currently deployed components of a specific type and their information. A preprocessing step is then applied to convert data values into numerical formats. Operations, such as counts of connections with other components, status categorization, or averages of data from connected components, are performed. Once the latest observations are acquired, the dataframe is updated with the total observations for the specific component type, retaining a maximum number of observations at any given time.

Subsequently, the current observation data is passed through the supervised model to classify the component as anomalous or not. The classification is performed per component of a specific type from the current observation. The anomaly scores obtained are then used to update the graph, and the process continues to the next component.

5.3.2.2 Aggregation Components

Components that aggregate anomaly scores of connected components are handled through a simple Neo4J database query, calculating the average of the anomaly scores of their connected components and updating the specified component.

5.3.3 Configurability and Adaptability

The user can configure the update frequency of models and the graph with two parameters. Additionally, the user has the flexibility to choose the information considered for each type of component. This adaptability allows the component to be easily reconfigured for different infrastructures. The component is agnostic to the types of activities it monitors, learning from baseline observations and assuming that anomalies will be infrequent within them. As such, it can monitor activities ranging from security threats to hardware resource management with minimal reconfiguration. Users can also specify the amount of component memory taken into account for retraining the models. The more memory allocated, the longer the component retains old observations, contributing to the robustness and historical context of the anomaly detection process.

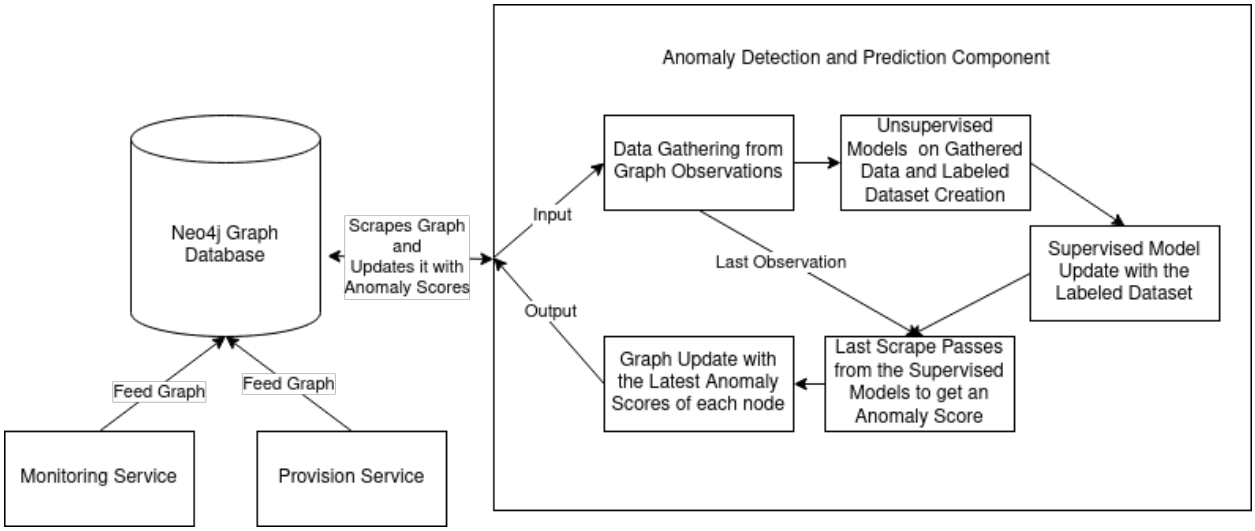

Figure 5.4: Anomaly Detection and Prediction Component Architecture

6. MODEL SELECTION AND EXPERIMENTS

6.1 Contextualizing Testing within the EO4EU Project

This thesis is conducted in the framework of the EO4EU project [10]. The EO4EU project is designed to leverage cutting-edge, pre-exascale High-Performance Computing (HPC) and Cloud infrastructures equipped with GPUs. These resources are intended to efficiently handle the expected processing workloads across a diverse range of use cases. The deployment architecture relies on microservices orchestration using Kubernetes Clusters and higher-level abstraction services like Function as a Service (FaaS), ensuring flexibility, extensibility, and scalability. The dynamic allocation of Persistent Volumes to Kubernetes clusters facilitates the storage requirements of the EO4EU data store.

The primary objective of our component within the project is to perform anomaly detection on the resources used by Kubernetes components that execute user-requested workflows. Since each workflow is unique and exhibits distinct behavior, our goal is to identify components used during the workflow as anomalous when their behavior deviates from the expected patterns.

6.2 Experiment Definition

Our experiments leverage data obtained from EO4EU workflow executions. The datasets derived from this data are utilized for model comparison. To identify optimal configurations for each model, we employ the hyperparameter optimization library, Optuna 3.3.3. This systematic exploration of hyperparameter space ensures the robustness of each model in anomaly detection or classification tasks across various datasets.

For evaluating the performance of each hyperparameter combination, we calculate the average Silhouette or F1 score across all datasets. We conduct an extensive search with 50 trials to maximize the score. Emphasizing the importance of a diverse dataset, the need for diversity is crucial for a meaningful evaluation.

Following each experimental iteration, our analysis will be visually represented through two informative plots: the Parallel Coordinate Plot and the Hyperparameter Importance Plot. The Parallel Coordinate Plot employs darker lines to signify configurations with higher objective values, providing a visual correlation between specific parameter combinations and their effectiveness in achieving desired outcomes. This visualization aids in identifying optimal configurations that contribute to superior model performance. In contrast, the Hyperparameter Importance Plot illustrates the influence of each hyperparameter on the model's overall performance. The percentage assigned to each hyperparameter indicates the magnitude of its impact on the model; a higher percentage signifies a greater influence on performance, offering understanding into the aspects of model tuning and optimization.

6.3 Data Gathering

To facilitate model selection, we initiate the process by gathering data from the resource registry graph. To ensure consistency with the inputs during actual workflow execution, we employ identical queries and data-processing methods for the graph. The quality of our

observations directly impacts the effectiveness of our model selection. Thus, the greater the number of observations, the better the outcomes. Furthermore, diversity in our observations is crucial. We aim to curate a dataset that encompasses a wide array of behaviors, showcasing the diversity inherent in different components. Ultimately, our goal is to compile a dataset for each component type associated with a model.

6.4 Unsupervised Model Tuning and Selection

6.4.1 Unsupervised Model Evaluation

Given our limited expertise in Kubernetes behavior, obtaining a labeled dataset representing real-world behaviors for assessing the unsupervised model's performance is impractical. Consequently, we turn to the Silhouette metric, as detailed in Section 3.1.2.5, to evaluate the model's ability to classify data. This metric serves as a quantitative measure, allowing us to assess the model's proficiency in clustering data. By leveraging this metric, we aim to provide a robust evaluation of the unsupervised model's performance in the absence of labeled real-world data. It's worth noting that a Silhouette score closer to 1 indicates a more optimal clustering solution.

6.4.2 Unsupervised Model Tuning

6.4.2.1 Isolation Forest Tuning

In our pursuit of optimizing the Isolation Forest Model 3.2.3.2, we tried to optimise the hyper parameters bellow:

- `n_estimators`: The number of isolation trees in the forest, ranging from 50 to 500.
- `max_samples`: The proportion of samples used to construct each tree, varying from 10% to 100%, with a step size of 10%.
- `contamination`: Representing the expected proportion of anomalies in the dataset, the contamination parameter was tuned between 1% and 50%, with a step size of 1%.
- `max_features`: The maximum number of features to consider for splitting a node, adjusted between 10% and 100% with a step size of 10%.
- `bootstrap`: A categorical parameter indicating whether bootstrap samples should be used when building trees, with options for both `True` and `False`.

After running an exhaustive search with 50 trials, we successfully identified the optimal hyperparameter configuration for the Isolation Forest Model:

- `n_estimators`: 109
- `max_samples`: 0.9
- `contamination`: 0.45

- `max_features`: 1.0
- `bootstrap`: False

The corresponding Silhouette score achieved with this configuration is 0.9451. This outcome, recorded during Trial 25, signifies the effectiveness of the selected hyperparameter values in maximizing the model's clustering performance.

For the Isolation Forest model, the Parallel Coordinate Plot (6.1) underscores key configurations for optimal performance, with darker lines indicating higher objective values. Notably, superior scores were consistently associated with settings such as `bootstrap: false`, maintaining `contamination` in the range of 0.3 to 0.5, ensuring `max features` and `samples` exceeded 0.8, and limiting `n_estimators` to be less than 200.

In the Hyperparameter Importances Plot (6.2), the significance of specific parameters is evident. The `contamination` parameter stands out with a substantial importance value of 0.78, followed by `max samples` at 0.18. Other hyperparameters exhibit marginal importances, all measuring less than 0.1, underscoring their minor impact on overall model performance.

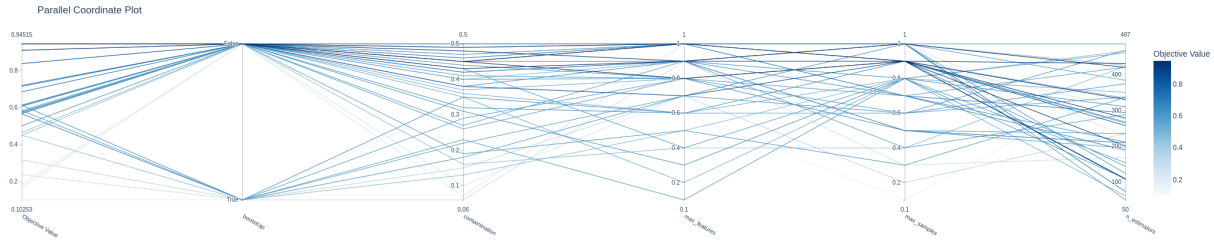

Figure 6.1: Parallel Coordinate Plot for Isolation Forest

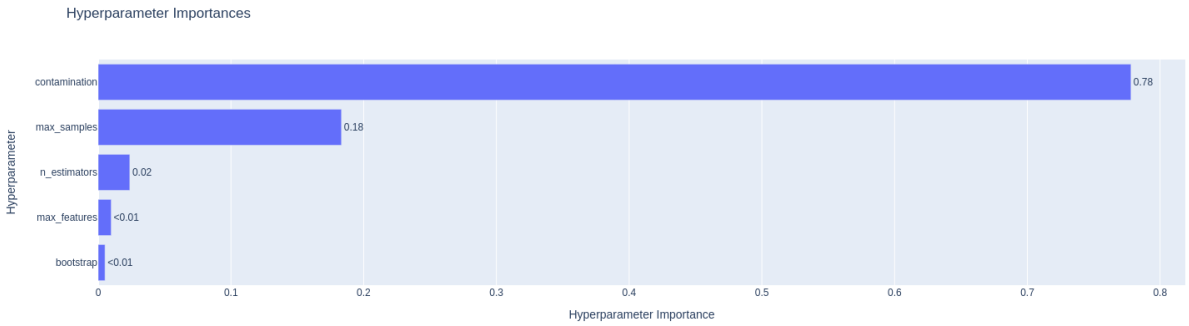

Figure 6.2: Hyper Parameter Importance for Isolation Forest

6.4.2.2 One-Class SVM Tuning

Similar to our approach with the Isolation Forest Model, we applied Optuna to fine-tune the One-Class SVM Model 3.2.3.3. The key parameters subjected to optimization include:

- `nu`: The regularization parameter, ranging from 0.01 to 0.5 with a step size of 0.01.
- `kernel`: The kernel function for the SVM, with options for 'linear', 'rbf', and 'poly'.

- `gamma`: The kernel coefficient (for 'rbf' and 'poly'), varying from 0.01 to 1.0 in a logarithmic scale.

After running the 50 trial search, we successfully identified the optimal hyperparameter configuration for the One-Class SVM Model:

- `nu`: 0.48
- `gamma`: 0.094
- `kernel`: poly

The corresponding Silhouette score achieved with this configuration is 0.7636842037305247. This outcome, was recorded during Trial 45.

For the One-Class SVM model, the Parallel Coordinate Plot (6.3) reveals configurations for optimal performance. Notably, higher objective values are associated with settings such as `gamma > 0.1`, and maintaining `nu > 0.35`. All `kernels` had dark lines passing through them, without highlighting one being much better than the others.

In the Hyperparameter Importances Plot (6.4), the significance of specific parameters is evident. The `nu` parameter stands out with a substantial importance value of 0.76, followed by `gamma` at 0.15. The `kernel` parameter also exhibits importance, albeit to a lesser extent, with a value of 0.09, verifying the conclusion of the previous paragraph.

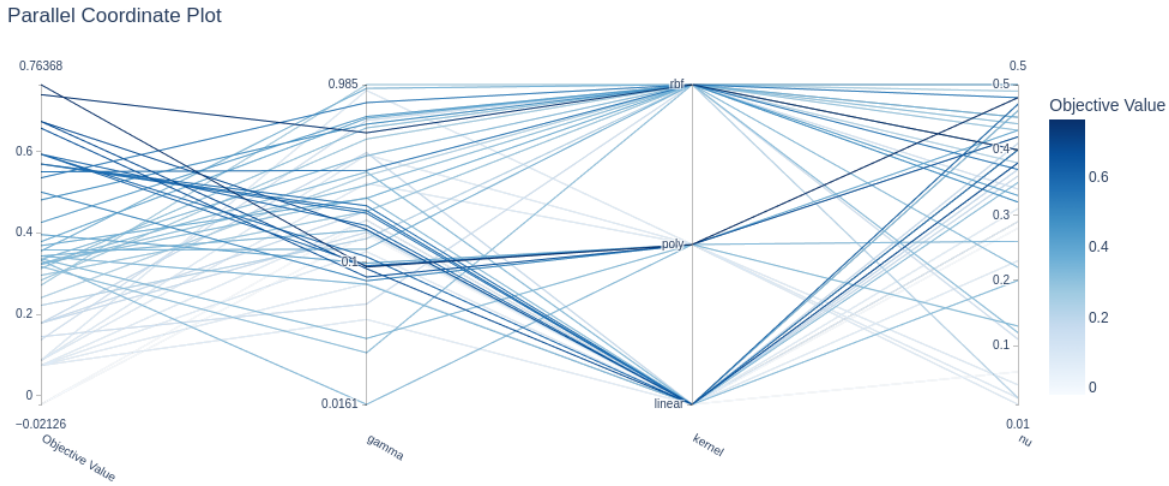

Figure 6.3: Parallel Coordinate Plot for One-Class SVM

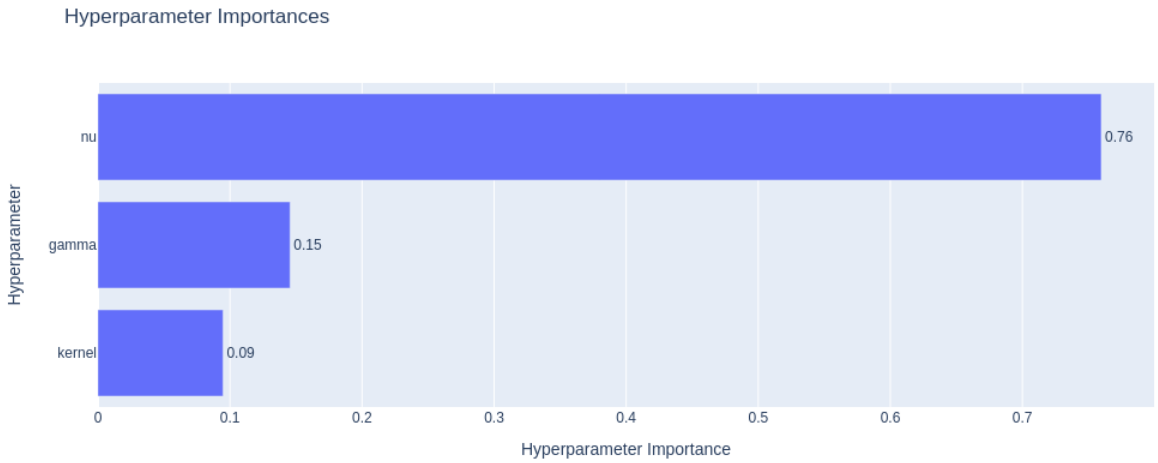

Figure 6.4: Hyper Parameter Importance for One-Class SVM

6.4.3 Unsupervised Model Selection

Model	Silhouette Score	Time (seconds)
Isolation Forest	0.9451	0.8185
One Class SVM	0.7636842037305247	0.0079

Table 6.1: Model Comparison Results

After tuning the models on our data, it becomes evident that the Isolation Forest excels at clustering datasets into anomalies or non-anomalies, while One Class SVM excel at being rapid in performing that task. Since we can afford the luxury of a slower but more precise model during the model training phase of our workflow, we will prefer the Isolation Forest Model. Consequently, we will leverage this optimized model configuration for subsequent runs of the actual anomaly detection and prediction workflow, as outlined in Section 5.3.

To further refine our tuning efforts in the Supervised Models section, we will employ this well-performing model combination to label our datasets. This labeling process is essential for preparing the datasets to be utilized in our ongoing tuning and evaluation endeavors.

6.5 Supervised Model Tuning and Selection

6.5.1 Supervised Model Evaluation

With labeled datasets in hand, our objective with the supervised models is to efficiently classify future observations by predicting how this new data would be clustered. The goal is to learn and apply the rules established by the unsupervised models using their supervised counterparts. To evaluate our success in this endeavor, we rely on the F1 score, as described in Section 3.1.2.5.

Once again, the importance of a diverse dataset is underscored. A diverse dataset ensures an ample number of observations for creating training and testing datasets, particularly with a substantial representation of anomaly observations. From each labeled

dataset, we allocate 80% of observations for training and 20% for testing. This approach enables us to measure the model's ability to generalize and make accurate predictions on new, unseen data.

6.5.2 Supervised Model Tuning

6.5.2.1 Logistic Regression Tuning

Once again we will use Optuna to optimize the Logistic Regression model 3.2.2.2. Among the hyperparameters subjected to optimization were:

- `C`: The regularization parameter, varying from 1×10^{-10} to 1×10^{10}.
- `penalty`: The regularization term, with options for 'l1' and 'l2'.
- `solver`: The optimization algorithm, with choices between 'liblinear', 'saga', and 'liblinear'.
- `max_iter`: The maximum number of iterations, ranging from 50 to 200.

This thorough exploration of hyperparameter space allows us to identify the optimal configurations for our supervised models. In the case of Logistic Regression, the best-performing hyperparameter values were found to be:

- `C`: 1.5983e+09 - `penalty`: 'l1' - `solver`: 'liblinear' - `max_iter`: 189

These hyperparameters achieved an impressive F1 score of 0.9509, highlighting the effectiveness of the selected values in maximizing the model's performance in classifying new observations. The dataset's diversity plays a crucial role in this evaluation, as it ensures a comprehensive representation for creating training and testing datasets.

For the Logistic Regression model, the Parallel Coordinate Plot (6.5) emphasizes that all combinations with the `solver liblinear` showcased excellent results, indicating the effectiveness of this solver.

In the Hyperparameter Importances Plot (6.6), the significance of specific parameters aligns with the observations from the parallel coordinate plot. Notably, the `solver` parameter stands out with a substantial importance value of 0.66, verifying the effectiveness of `liblinear`. Additionally, `max iter` has an importance value of 0.21, while `C` and `penalty` contribute with values of 0.1 and 0.03, respectively.

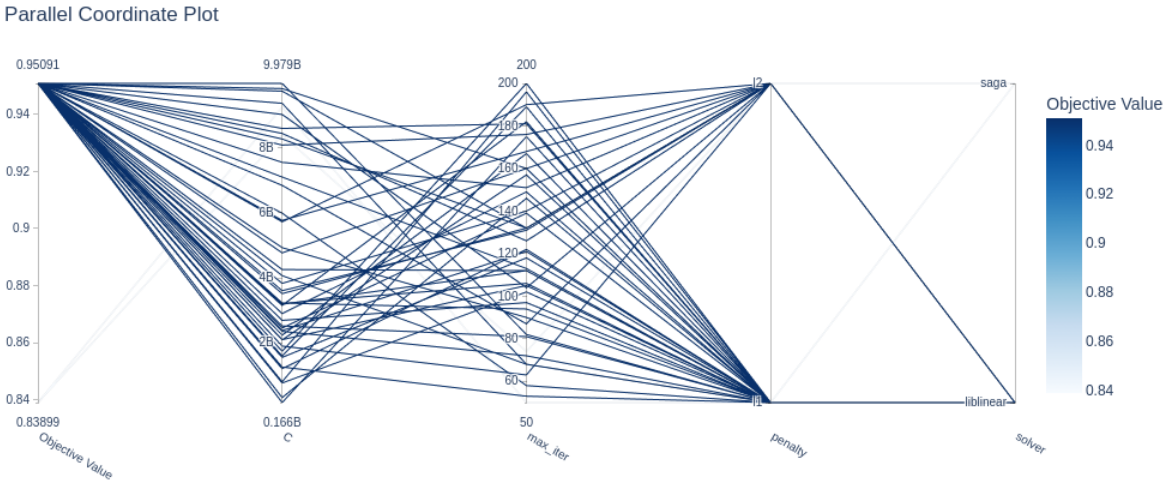

Figure 6.5: Parallel Coordinate Plot for Logistic Regression

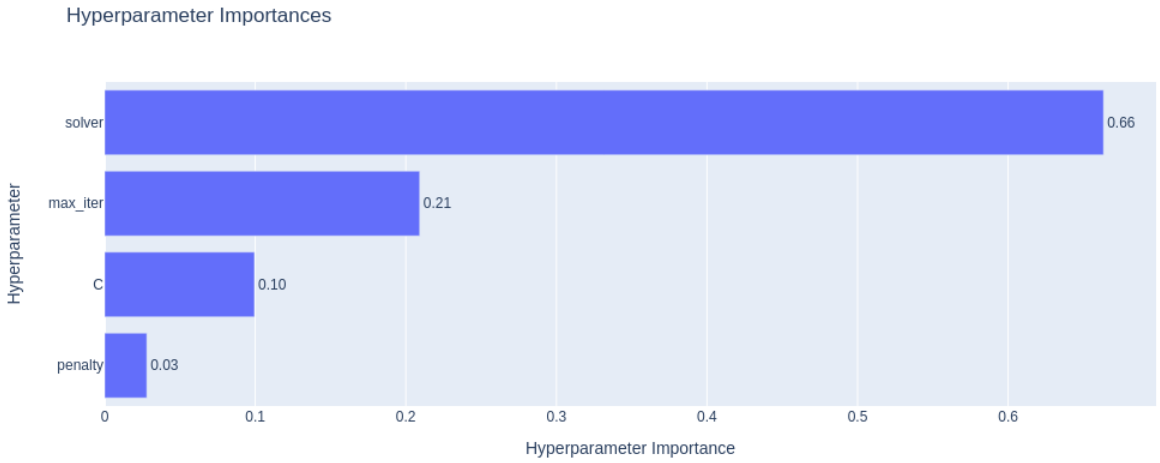

Figure 6.6: Hyper Parameter Importance for Logistic Regression

6.5.2.2 Support Vector Machines Tuning

Once again we will use Optuna to optimize the SVM model 3.2.2.3. Among the hyper-parameters subjected to optimization were:

- `C`: The regularization parameter, ranging from 1×10^{-3} to 1.
- `kernel`: The kernel function for the SVM, with options for 'linear', 'poly', 'rbf', and 'sigmoid'.
- `gamma`: The kernel coefficient, varying from 0.0001 to 10.
- `degree`: The degree of the polynomial kernel function, ranging from 1 to 10.
- `coef0`: The independent term in the kernel function, ranging from 0 to 10.

This thorough exploration of hyperparameter space allows us to identify the optimal configurations for our supervised models. In the case of SVM, the best-performing hyperparameter values were found to be:

- `C`: 0.2601 - `kernel`: 'poly' - `gamma`: 4.9406 - `degree`: 7 - `coef0`: 7

These hyperparameters achieved an impressive F1 score of 1, highlighting the effectiveness of the selected values in maximizing the model's performance in classifying new observations. The dataset's diversity plays a crucial role in this evaluation, as it ensures a comprehensive representation for creating training and testing datasets.

For the SVM model, the Parallel Coordinate Plot (6.7) demonstrates that all values of `C` and `degree` lead to favorable results. Additionally, configurations with `coef` > 6, `gamma` in the range of 4-6, and utilizing the `poly` kernel exhibit good performance.

In the Hyperparameter Importances Plot (6.8), the paramount influence of the `kernel` parameter is evident, with a substantial importance value of 0.95. The remaining hyperparameters show negligible importance in comparison.

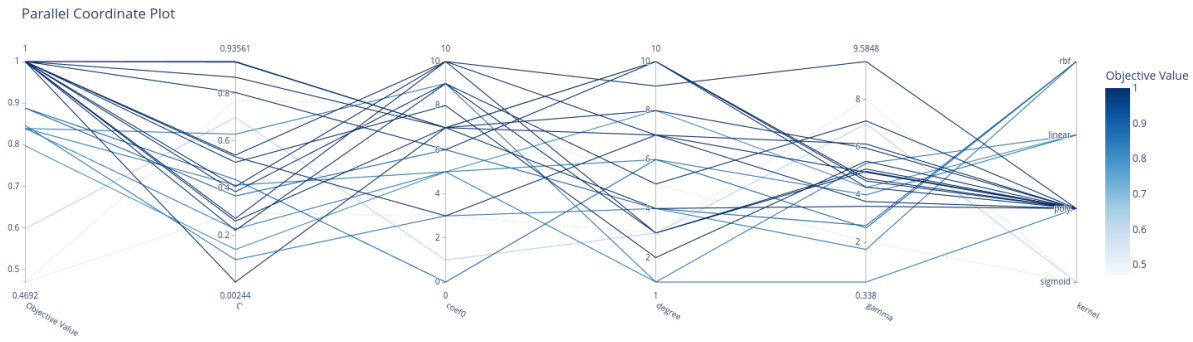

Figure 6.7: Parallel Coordinate Plot for SVM

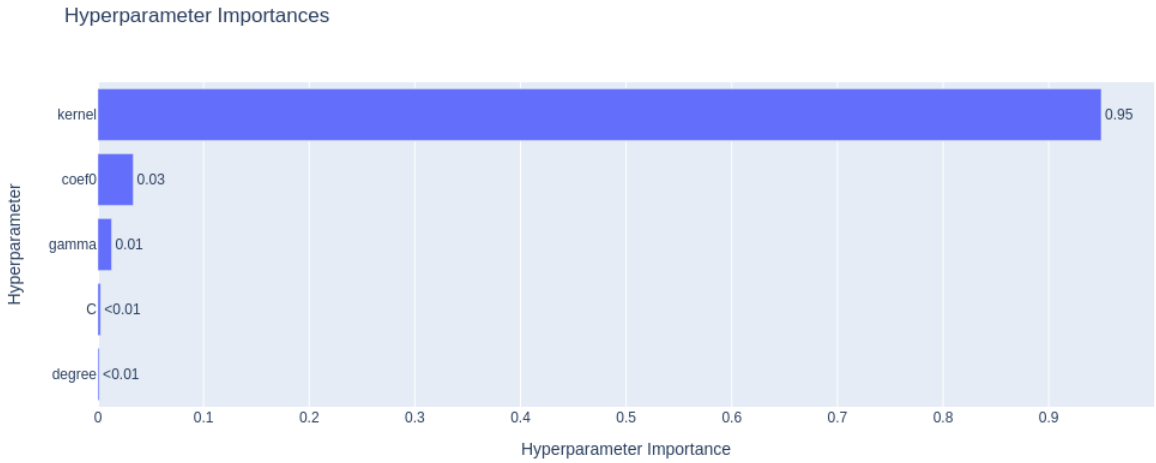

Figure 6.8: Hyper Parameter Importance for SVM

6.5.2.3 Decision Tree Tuning

Once again we will use Optuna to optimize the Decision Tree model 3.2.2.4. Among the hyperparameters subjected to optimization were:

- `criterion`: The function to measure the quality of a split, with options for 'gini' and 'entropy'.
- `splitter`: The strategy used to choose the split at each node, with options for 'best' and 'random'.
- `max_depth`: The maximum depth of the tree, ranging from 1 to 100.
- `min_samples_split`: The minimum number of samples required to split an internal node, ranging from 2 to 100.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node, ranging from 1 to 100.
- `max_features`: The number of features to consider for the best split, varying from 0.1 to 1.0.

For the case of Decision Tree, the best-performing hyperparameter values were found to be:

- `criterion`: 'entropy' - `splitter`: 'random' - `max_depth`: 100 - `min_samples_split`: 2 - `min_samples_leaf`: 1 - `max_features`: 0.1339

These hyperparameters achieved an impressive F1 score of 1, highlighting the effectiveness of the selected values in maximizing the model's performance in classifying new observations. The dataset's diversity plays a crucial role in this evaluation, as it ensures a comprehensive representation for creating training and testing datasets.

For the Decision Tree model, the Parallel Coordinate Plot (6.9) illustrates favorable configurations for both criteria. Effective results were observed with `max depth` > 70, `max features` < 0.5, `min samples leaf` < 10, `min samples split` < 15, and utilizing the `random` splitter.

In the Hyperparameter Importances Plot (6.10), the paramount influence of the `min samples leaf` parameter is evident, with a substantial importance value of 0.94. Additionally, `max features` contributes with an importance value of 0.05, while the remaining hyperparameters show insignificant importance.

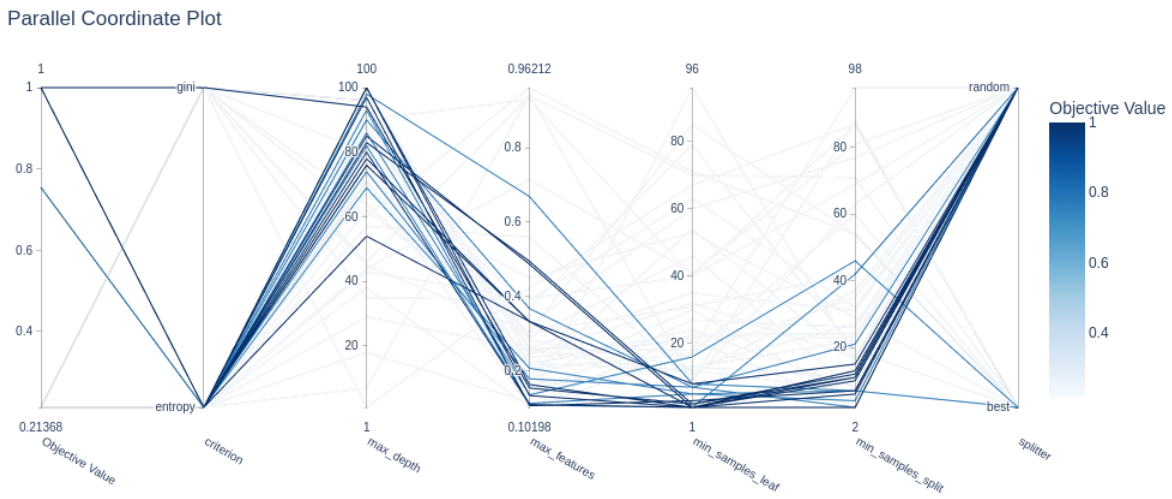

Figure 6.9: Parallel Coordinate Plot for Decision Tree

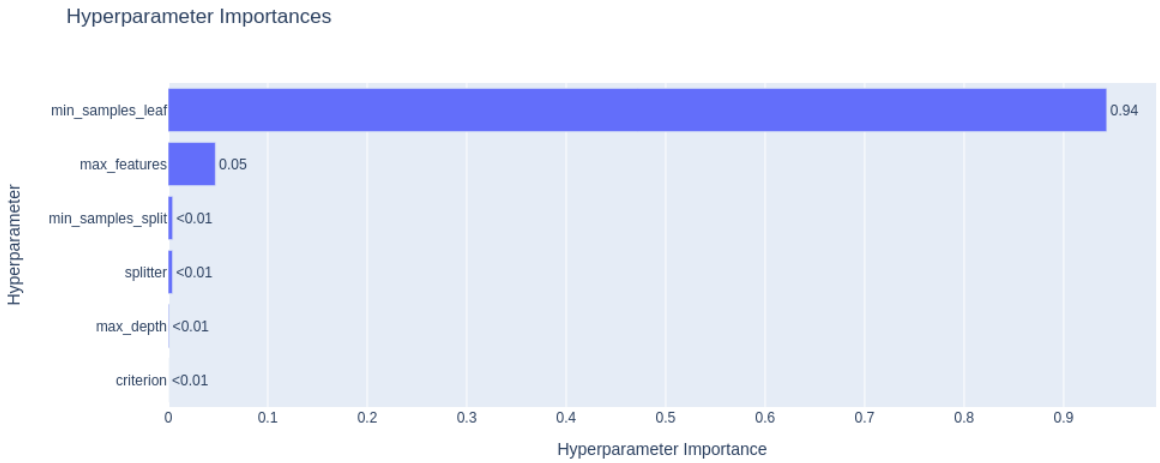

Figure 6.10: Hyper Parameter Importance for Decision Tree

6.5.3 Supervised Model Selection

Model	F1 Score	Train Time Sec	Predict Time Sec	Predicts Prob.
Logistic Regression	0.9509,	0.0168	0.0071	Yes
Support Vector Machines	1	0.0153	0.0070	Yes
Decision Tree	1	0.0145	0.0067	No

Table 6.2: Model Comparison Results

After fine-tuning the models on our data, all three models showcase impressive results. Each model demonstrates the ability to accurately describe the rules defined by the unsupervised models. In terms also of time all models are pretty similar, especially in the Prediction task that we need the most speed.

A crucial consideration for the supervised model is the inclusion of a predict probability function. This function enables a more nuanced classification, allowing for various levels of anomalous behavior to be identified. Without this function, the graph would be colored only by either anomaly or non-anomaly, limiting the range of classifications. The predict probability function allows us to recognize the degree to which an observation deviates from our baseline behavior considered normal.

Taking into account all these factors, Support Vector Machines, showcase the best overall attributes.

6.6 Anomaly Detection

Utilizing our models, we conducted anomaly detection on real-world workflow executions, uncovering irregularities within the system. Notably, our analysis revealed a significant anomaly within a Deployment instance characterized by the presence of nine Replica Sets. Remarkably, one of these Replica Sets contained a Pod replica, while the remaining sets, devoid of replicas, held no functional relevance to the workflow and should have been removed.

Anomaly detection algorithms assigned a remarkably high anomaly score of 0.99 to all Replica Sets lacking replicas, as depicted in Figure 6.11, unequivocally signaling their anomalous status. Conversely, a separate Replica Set within the same Deployment, containing the expected Pod, received a normality designation with a score of 0.35, as shown in Figure 6.12, falling below the anomaly threshold of 0.5.

Figure 6.11: Replica Set designated as Anomalous

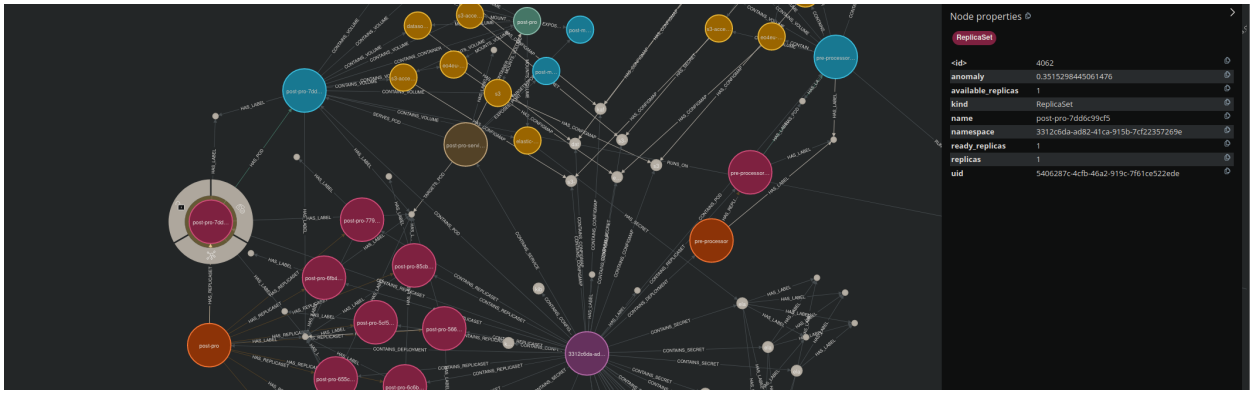

Figure 6.12: Replica Set designated as Normality

Considering the collective anomaly status of these components, particularly within the Deployment, becomes imperative. The Deployment's overall anomaly score, depicted in Figure 6.13, is influenced by the anomaly scores of its eight anomalous Replica Sets along with the score of the normal set, resulting in an alarming score of 0.928.

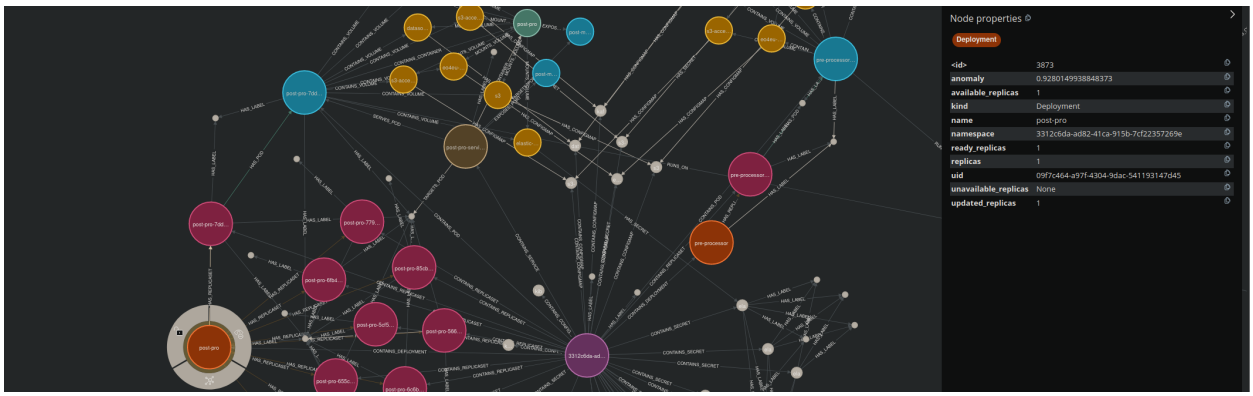

Figure 6.13: Deployment designated as Anomalous

Moreover, as these components are associated with a specific Namespace, it is reasonable to expect the namespace's anomaly score to reflect the irregularities present within its constituent elements. As demonstrated in Figure 6.14, the Namespace is indeed flagged as anomalous, albeit with a slightly lower score of 0.79. This discrepancy in score can be attributed to the presence of other components within the namespace operating as expected, including the aforementioned normal Replica Set.

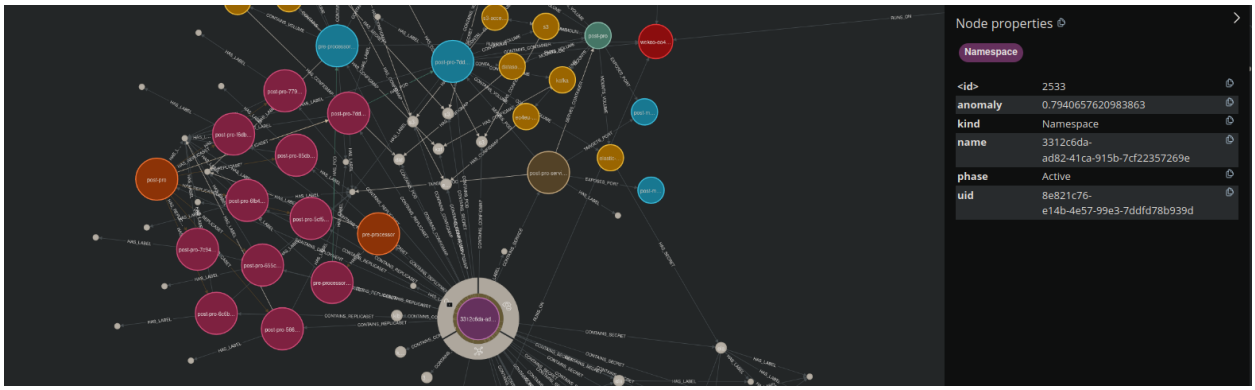

Figure 6.14: Namespace designated as Anomalous

In summary, our experiment successfully identified anomalies within real-world workflow executions, particularly within Deployments and their Replica Sets. The graph visually demonstrated the propagation of anomaly scores from individual Replica Sets to the overarching Deployment and Namespace, highlighting the interconnectedness of these components. These findings underscore the importance of effective anomaly detection in maintaining system integrity.

7. CONCLUSION AND FUTURE WORK

The primary objective of this thesis was to propose a comprehensive workflow for detecting anomalous behavior within Kubernetes components and effectively highlighting such instances. In achieving this goal, we successfully leveraged a graph database to represent the Kubernetes cluster, facilitating seamless querying and showcasing the intricate connections between its components. This approach laid the foundation for a holistic understanding of the cluster's dynamics.

Furthermore, we introduced a combination of Unsupervised Machine Learning (ML) techniques to discern anomalous behaviors among the various components. This was complemented by the incorporation of Supervised ML techniques, allowing for the swift labeling of future observations as anomalous or not. The utilization of anomaly scores assigned to the graph database nodes proved instrumental in streamlining this process.

A critical aspect of our work involved proposing a workflow for fine-tuning and selecting ML models, aiming for optimal anomaly detection aligned with specific graph queries. The achieved Silhouette scores of 0.9451 and F1 scores reaching 1 underscore the efficacy of our approach in accurately identifying and classifying anomalous behaviors within the Kubernetes environment.

Furthermore, the future holds the promise of enhancing predictive capabilities by incorporating Long Short-Term Memory (LSTM) neural networks. This advanced approach seeks to harness the power of time series predictions, enabling the detection of anomalies before they manifest. By delving into the realm of predictive analytics, we aspire to further fortify the resilience and proactive management of Kubernetes components.

ABBREVIATIONS - ACRONYMS

IT	Information Technology
IP	Internet Protocol
HTTP	Hyper text transfer Protocol
HTTPs	Hyper text transfer Protocol secure
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
K8s	Kubernetes
OS	Operating System
VM	Virtual Machine
AWS	Amazon Web Services
API	Application Programming Interface
CPU	Central Processing Unit
ABI	Application Binary Interface
SSH	Secure Shell
SQL	Standard Query Language
CRUD	Create Read Update Delete
ASCII	American Standard Code for Information Interchange
ML	Machine Learning
SVM	Support Vector Machine
BSD	Berkley Software Distribution
KAD	Kubernetes Anomaly Detector
PDFA	Probabilistic Deterministic Finite Automaton
EU	European Union
EO	Earth Observation
EO4EU	Earth Observation for European Union
GUI	Graphic User Interface
CLI	Command Line Interface
XR	Extended Reality

BIBLIOGRAPHY

[1] Decision Trees . https://scikit-learn.org/stable/modules/tree.html, 2023. [Online; accessed 18/12/2023].

[2] Edge Impulse - Anomaly detection . https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/anomaly-detection-gmm and https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/anomaly-detection, 2023. [Online; accessed 27/12/2023].

[3] IaaS vs. PaaS vs. SaaS . Available:https://www.ibm.com/topics/iaas-paas-saas, 2023. [Online; accessed 5/12/2023].

[4] Service Now - Anomalies detection . https://docs.servicenow.com/bundle/vancouver-impact/page/product/impact/task/anomalies-detection.html, 2023. [Online; accessed 22/12/2023].

[5] The Benefits of Containerization and What It Means for You . Available:https://www.ibm.com/blog/the-benefits-of-containerization-and-what-it-means-for-you/, 2023. [Online; accessed 10/12/2023].

[6] What are microservices? . Available:https://microservices.io/, 2023. [Online; accessed 8/12/2023].

[7] What is machine learning? . https://www.ibm.com/topics/machine-learning, 2023. [Online; accessed 16/12/2023].

[8] What's the Difference Between Monolithic and Microservices Architecture? . Available:https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/, 2023. [Online; accessed 8/12/2023].

[9] Kubernetes. Available:https://kubernetes.io/docs/concepts/architecture/, 2023. [Online; accessed 03/01/2024].

[10] EO4EU Proposal . https://www.hesge.ch/heg/sites/default/files/actualite/documents/2022/projet-recherche-alexandros_kalousis-eo4eu-executive-summary.pdf, 2024. [Online; accessed 8/1/2024].

[11] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[12] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, Cambridge, MA, 2014.

[13] Alex Smola John Shawe-Taylort John Platt Bernhard Scholkopf, Robert Williamson. Support vector method for novelty detection. In *NeurIPS 1999 Proceedings*, 1999.

[14] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[15] Clinton Cao, Agathe Blaise, Sicco Verwer, and Filippo Rebecchi. Learning state machines to monitor and detect anomalies on a kubernetes cluster. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES 2022. ACM, August 2022.

[16] Qingfeng Du, Tiandi Xie, and Yu He. Anomaly detection and diagnosis for container-based microservices with performance monitoring. EasyChair Preprint no. 468, EasyChair, 2018.

[17] Darell Edmond, Vijay Soni, Lalit Garg, and Seema Bawa. Adoption of cloud services in central banks: Hindering factors and the recommendations for way forward. *Journal of Central Banking Theory and Practice*, 11:123–143, 05 2022.

[18] Ioannis Emiris. Software development course: Section 3 - neural networks, December 10 2023. Offline lecture slides.

[19] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On explaining decision trees, 2020.

[20] Daniel Jurafsky and James H. Martin. Speech and language processing, 2023. Draft of January 7, 2023.

[21] Ramandeep Kaur and Sumit Chopra. Virtualization in cloud computing : A review. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 01–05, 07 2020.

[22] Joanna Kosińska and Maciej Tobiasz. Detection of cluster anomalies with ml techniques. *IEEE Access*, 10:110742–110753, 2022.

[23] Yingyu Liang. Machine learning basics - lecture 4: Svm i. `https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/ML_basics_lecture4_SVM_I.pdf`, 2016.

[24] Yingyu Liang. Machine learning basics - lecture 5: Svm ii. `https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/ML_basics_lecture5_SVM_II.pdf`, 2016.

[25] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[26] Guozhi Liu, Bi Huang, Zhihong Liang, Minmin Qin, Hua Zhou, and Zhang Li. Microservices: architecture, container, and challenges. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 629–635, 2020.

[27] Félix López and Eulogio Cruz. Literature review about neo4j graph database as a feasible alternative for replacing rdbms. *Industrial Data*, 18:135, 12 2015.

[28] Víctor Medel, Omer Rana, José Bañares, and Unai Arronategui. Modelling performance resource management in kubernetes. pages 257–262, 12 2016.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] Aditi Rajesh Nimodiya Pratik Narendra Gulhane1. A review paper on cloud computing. *International Advanced Research Journal in Science, Engineering and Technology*, 2022.

[31] Ying Wu Qiong Liu. Supervised learning. 2012.

[32] Bibhuti Regmi. Neo4j graph database, 01 2021.

[33] Marko A. Rodriguez and Peter Neubauer. The graph traversal pattern, 2010.

[34] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[35] Jay Shah and Dushyant Dubaria. Building modern clouds: Using docker, kubernetes google cloud platform. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0184–0189, 2019.

[36] Chin-Wei Tien, Tse-Yung Huang, Chia-Wei Tien, Ting-Chun Huang, and Sy-Yen Kuo. Kubanomaly: Anomaly detection for the docker orchestration platform with neural network approaches. *Engineering Reports*, 1(5):e12080, 2019.

[37] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications, 2020.