UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DEPARTMENT OF DIGITAL SYSTEMS

POSTGRADUATE PROGRAM

“INFORMATION SYSTEMS & SERVICES”

Clustering Algorithm Recommendation Platform for Distributed Environments

by

Panagiotis Karamolegkos

MSc Programme “Information Systems & Services”

UNIVERSITY OF PIRAEUS

September 2024

Author: Panagiotis Karamolegkos

ΣΕΛΙΔΑ ΕΓΚΥΡΟΤΗΤΑΣ

Ονοματεπώνυμο Φοιτητή/Φοιτήτριας: Παναγιώτης Καραμολέγκος

Τίτλος Μεταπτυχιακής Διπλωματικής Εργασίας: Clustering Algorithm Recommendation Platform for Distributed Environments

Η παρούσα Μεταπτυχιακή Διπλωματική Εργασία υποβάλλεται ως μερική εκπλήρωση των απαιτήσεων του Προγράμματος Μεταπτυχιακών Σπουδών "Πληροφοριακά Συστήματα & Υπηρεσίες" του Τμήματος Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς και εγκρίθηκε στις 30/09/2024 [ημερομηνία έγκρισης] από τα μέλη της Εξεταστικής Επιτροπής.

Εξεταστική Επιτροπή

Επιβλέπων/ουσα (Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιώς):

Δημοσθένης Κυριαζής, Καθηγητής

Μέλος Εξεταστικής Επιτροπής: Μιχαήλ Φιλιππάκης, Καθηγητής

Μέλος Εξεταστικής Επιτροπής: Ανδρέας Μενύχτας, Επίκουρος Καθηγητής

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΑΥΘΕΝΤΙΚΟΤΗΤΑΣ

Ο/Η Παναγιώτης Καραμολέγκος, γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα ότι η παρούσα εργασία με τίτλο «Clustering Algorithm Recommendation Platform for Distributed Environments», αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές που έχω χρησιμοποιήσει, έχουν δηλωθεί κατάλληλα στις βιβλιογραφικές παραπομπές και αναφορές. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.
Επιπλέον *δηλώνω υπεύθυνα ότι η συγκεκριμένη Μεταπτυχιακή Διπλωματική Εργασία έχει συγγραφεί από εμένα προσωπικά και δεν έχει υποβληθεί ούτε έχει αξιολογηθεί στο πλαίσιο κάποιου άλλου μεταπτυχιακού ή προπτυχιακού τίτλου σπουδών, στην Ελλάδα ή στο εξωτερικό.*
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου. Σε κάθε περίπτωση, αναληθούς ή ανακριβούς δηλώσεως, υπόκειμαι στις συνέπειες που προβλέπονται τις διατάξεις που προβλέπει η Ελληνική και Κοινοτική Νομοθεσία περί πνευματικής ιδιοκτησίας.

Ο/Η ΔΗΛΩΝ/ΟΥΣΑ

Ονοματεπώνυμο: Παναγιώτης Καραμολέγκος
Αριθμός Μητρώου: me2239
Υπογραφή:

[illegible]

In this page, I want to take a moment to reflect. As readers navigate through this document, they will encounter a wide range of tools, methodologies, frameworks, and techniques. For me, this document serves as a testament to my knowledge—a proof not just of my ability to create but to adapt, and a demonstration of my capacity for finding solutions.

This level of understanding is not something that can be achieved in a night, or even in a few. It represents knowledge gained over countless sleepless nights, taking one step at a time to become a better version of myself with each passing day.

No, it hasn't been easy. And no, I do not claim to have reached the pinnacle of my potential. But by looking back, I can see how much I've grown.

When I first entered my Bachelor's program in the Department of Digital Systems, I didn't have a clear sense of what I wanted to learn or how I would approach it. I simply believed in my ability—that if I set my mind to something, I could master it. My goals were scattered, my future uncertain.

But as I immersed myself in research programs with the guidance of my professors, I pushed the boundaries of my knowledge and skills. I began teaching, sometimes to classrooms of just three students, and other times to groups of forty. I've presented at workshops and international conferences, and I've had the privilege of meeting influential people who matter to me and my field.

Now, as I complete my Master's program, my vision has sharpened. I understand what I want to achieve with my skills, and more importantly, I recognize that my actions extend beyond myself—they impact those around me. This, I believe, is the most important realization of all.

"When you get, give.
When you learn, teach."

At this stage in my life, I am committed to continuing to make a positive impact on the students in the Department of Digital Systems. I want to pass on not only my knowledge and techniques but also my enthusiasm to the next generation of university students.

Yet, I must always remember that I am, first and foremost, a lifelong student. In my belief, no great teacher exists who isn't also a great learner.

Neither I nor this document would exist in our current form without the encouragement and support of others. I owe thanks to many:

My parents, for supporting me every step of the way.
My sisters, for listening to me ramble when I needed it most.
My friends, for standing by me, even when we are miles apart.
My professors, for being the role models that guided my ambitions.
And finally, myself, for constantly reflecting and always pushing forward, one step at a time.

Table of Contents

List of Figures

List of Tables

Acronyms

Acronym	Definition
ACID	Atomicity, Consistency, Isolation, Durability
AI	Artificial Inteligence
API	Application Programming Interface
BASE	Basically Available, Soft state, Eventually consistent
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-Separated Value
DBMS	Database Management Systems
DDPG	Deep Deterministic Policy Gradient
DOM	Document Object Model
DRL	Deep Reinforcement Learning
GAN	Generative Adversarial Networks
GMM	Gaussian Mixture Model
GNN	Graph Neural Network
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
ID	Identification
JS	JavaScript
JSON	JavaScript Object Notation
K8s	Kubernetes
KNN	K-Nearest Neighbor
ML	Machine Learning
NCF	Neural Collaborative Filtering
NN	Neural Network
NoSQL	Non-Structured Query Language
OS	Operating System
PKL	Pickle
PS	Parameter Server
RAM	Radnom-Access Memory
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
RF	Random Forest
RL	Reinforcement Learning
S3	Simple Storage Service
SDN	Software-Defined Networks
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVM	Support Vector Machine
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
WCSS	Within-Cluster Sum of Squares
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

Abstracts

Abstract in English

The rapid growth of data in various fields has necessitated the development of advanced tools for effective data analysis. Clustering, a fundamental technique in machine learning, plays a crucial role in organizing and interpreting large datasets. However, selecting the most suitable clustering algorithm for specific data characteristics poses significant challenges, as it requires balancing computational speed and accuracy while considering the intricacies of dataset properties and algorithm parameters. This document presents EverCluster, a comprehensive, cloud-centric platform designed to streamline the clustering process. EverCluster automates the recommendation of optimal clustering algorithms by leveraging machine learning models that adapt to user preferences and dataset features. The architecture of EverCluster is detailed, featuring both high- and low-level descriptions supported by deployment and activity diagrams. Experimental findings highlight the platform's effectiveness, revealing a success rate of 65.5% for speed-based recommendations and an average of 81.1% for accuracy-based recommendations. The platform performs particularly well on datasets with fewer features and higher iteration numbers making the speed-based recommendations to reach success rates of 83.3%. By addressing the complexities involved in clustering algorithm selection and deployment, EverCluster aims to provide a valuable resource for data scientists and researchers, facilitating more efficient and accurate data analysis across diverse applications.

Abstract in Greek

Η ραγδαία αύξηση των δεδομένων σε διάφορους τομείς έχει καταστήσει αναγκαία την ανάπτυξη προηγμένων εργαλείων για την αποτελεσματική ανάλυση δεδομένων. Η συσταδοποίηση, μια θεμελιώδης τεχνική στη μηχανική μάθηση, διαδραματίζει κρίσιμο ρόλο στην οργάνωση και ερμηνεία μεγάλων συνόλων δεδομένων. Ωστόσο, η επιλογή του πιο κατάλληλου αλγορίθμου συσταδοποίησης για συγκεκριμένα χαρακτηριστικά δεδομένων θέτει σημαντικές προκλήσεις, καθώς απαιτεί τη διαχείριση της υπολογιστικής ταχύτητας και της ακρίβειας, λαμβάνοντας υπόψη τις πολυπλοκότητες των χαρακτηριστικών των δεδομένων και των παραμέτρων του αλγορίθμου. Αυτό το έγγραφο παρουσιάζει το EverCluster, μια ολοκληρωμένη, cloud-centric πλατφόρμα σχεδιασμένη για να απλοποιήσει τη διαδικασία συσταδοποίησης. Το EverCluster αυτοματοποιεί τη σύσταση βέλτιστων αλγορίθμων συσταδοποίησης, αξιοποιώντας μοντέλα μηχανικής μάθησης που προσαρμόζονται στις προτιμήσεις των χρηστών και τα χαρακτηριστικά των συνόλων δεδομένων. Η αρχιτεκτονική του EverCluster περιγράφεται λεπτομερώς, περιλαμβάνοντας τόσο υψηλού όσο και χαμηλού επιπέδου περιγραφές που υποστηρίζονται από διαγράμματα εγκατάστασης και διεργασιών. Τα πειραματικά ευρήματα αναδεικνύουν την αποτελεσματικότητα της πλατφόρμας, αποκαλύπτοντας ποσοστό επιτυχίας 65,5% για προτάσεις βασισμένες στην ταχύτητα και μέσο ποσοστό 81,1% για προτάσεις βασισμένες στην ακρίβεια. Η πλατφόρμα αποδίδει ιδιαίτερα καλά σε σύνολα δεδομένων με λιγότερα χαρακτηριστικά και υψηλότερους αριθμούς επαναλήψεων, καθιστώντας τις προτάσεις ταχύτητας να φτάνουν σε ποσοστά επιτυχίας 83,3%. Αντιμετωπίζοντας τις πολυπλοκότητες που εμπλέκονται στην επιλογή και την ανάπτυξη αλγορίθμων συσταδοποίησης, το EverCluster στοχεύει να παρέχει έναν πολύτιμο πόρο για επιστήμονες δεδομένων και ερευνητές, προσφέροντας την πιο αποτελεσματική και ακριβή ανάλυση δεδομένων σε διάφορες εφαρμογές.

1. Introduction

1.1. Purpose of the Document

As the volume of data generated across industries continues to grow exponentially [1], the need for efficient and scalable tools to manage and analyze this data becomes increasingly critical [2]. Clustering algorithms, which group similar data points into clusters, have emerged as one of the key techniques for uncovering patterns and insights from large datasets [3]. However, selecting the most suitable clustering algorithm for a given dataset is not a straightforward task, as it involves balancing trade-offs between factors like computational speed, accuracy, and resource availability. In this context, there is a growing demand for platforms that can automate and optimize this decision-making process.

This document introduces EverCluster, a cloud-centric platform designed to simplify and streamline the clustering workflow. EverCluster allows users to upload datasets, configure clustering parameters, and receive algorithm recommendations based on their performance across various metrics. The platform leverages Machine Learning models to recommend clustering algorithms, tailored to user preferences for speed or accuracy. In addition to this, EverCluster is built to handle multiple deployment environments, offering flexibility in both single-node and multi-node setups.

The architecture of EverCluster is thoroughly detailed, supported by both high- and low-level design descriptions. Through deployment and activity diagrams, the document explains the structural and functional flow of the platform. This allows readers not only to understand the platform's technical design but also to replicate the experiments conducted in this research. The Machine Learning models that power EverCluster's recommendations can be retrained or modified, providing opportunities for users to generate new or improved outcomes based on their own datasets or experimental setups.

By deploying EverCluster in various environments, this document explores the challenges and insights gathered from experimentation, setting the stage for future research and developments. The platform's ability to integrate and adapt to different clustering contexts makes it a valuable tool for data scientists, engineers, and researchers aiming to optimize their clustering processes across diverse datasets and infrastructures.

EVERCLUSTER

Figure 1 – EverCluster Logo

1.2. Problem Statement

The process of selecting the most appropriate clustering algorithm for a given dataset remains a challenging task, particularly in the context of large-scale data and distributed computing environments [3]. The decision to choose one algorithm over another often hinges on multiple factors, including the nature of the dataset, the computational resources available, and the desired balance between speed and accuracy. Without prior knowledge of how different algorithms will perform on a specific dataset, users are left with trial-and-error approaches, which can be time-consuming and resource-intensive.

Moreover, the effectiveness of clustering algorithms can vary significantly depending on the configuration of their hyperparameters, such as the number of clusters or the maximum number of iterations. These parameters must be fine-tuned based on the characteristics of the dataset, which adds further complexity to the decision-making process. The need for specialized expertise in Machine Learning and data analysis presents a barrier for many users, particularly those who lack the technical background required to interpret and evaluate the performance of different algorithms.

In addition to these challenges, the increasing use of distributed systems for data processing introduces another layer of complexity. Clustering in a distributed environment requires careful consideration of factors like infrastructure topology and computational load balancing. In many cases, users must rely on manual intervention to adapt the clustering algorithms to their infrastructure, leading to inefficiencies and suboptimal results.

Given these issues, there is a clear need for a platform that can automatically recommend the most suitable clustering algorithm based on the characteristics of the dataset and the user's preference for speed or accuracy, while accounting for the distributed nature of modern data processing environments. This platform must also be able to dynamically adapt its recommendations based on changing conditions, such as infrastructure adjustments or evolving clustering requirements. EverCluster aims to address these challenges by providing a robust, cloud-centric solution that simplifies the clustering algorithm selection process, improving both efficiency and accuracy for a wide range of users and applications.

1.3. Document Structure

This document is divided into nine main sections, each one divided, if necessary, into more subsections to provide a clear overview of the literature, developed architecture, experimentations and results performed. Each section is used as follows:

- *Section 1 – Introduction*: The introduction section provides a concise overview to guide the reader within the context of the thesis. It summarizes the document by outlining the problem statement and the proposed solution approach. Furthermore, it introduces the set of tools utilized in the platform's development.

- *Section 2 – Background & Related Work*: In this section an Extensive Analysis is provided, regarding the literature review that needed to be performed to find solutions as well as to understand better the problem statement. Different related works and research topics are also provided and explained.

- *Section 3 – Implemented Tools & Methodologies*: With this section, the document provides an overview of the different tools and methodologies that assisted in the creation of the platform. All the tools explained in this section are later being used by *Section 4*, to describe the platform's architecture from high- and low-level perspectives.

- *Section 4 – Platform Implementation*: In this section the objectives of EverCluster and its concerned end-users, with the complete architecture of the platform are analyzed. The section begins with the high-level figure of the architecture and continues by breaking the high-level image into low-level figures while analyzing the tools and connections shown. With the architecture the code of the platform is analyzed providing different deployment diagrams as well. This section provides a quick installation guide and a user manual to the reader for the experimentation that follows to be able to be replicated by different readers and interested parties.

- *Section 5 – Experimentation Methodology*: This section is responsible for analyzing to the reader the Experimentation performed. The section begins with the experimentation methodology used on the developed platform. Continuing the methodology the different experiments performed are described in length, while giving references to the Datasets used and explaining the executed preprocessing procedures to apply the methodology that was analyzed.

- *Section 6 – Experimentation Results*: Continuing the section regarding the methodology of the experimentation, this section is responsible to provide the extracted results. With the results, an evaluation of the platform is analyzed and a quick comparison with the literature work that was given in *Section 2* is performed.

- *Section 7 – Discussion & Next Steps*: In this section there is a discussion on the contents of the whole document. The Section's purpose is to give a perspective of the results provided, as well as to find and explain possible difficulties in the creation of the platform, the methodologies performed as well as the experimentation infrastructure that was used. After the discussion, possible next steps on this work are provided with future guidance for the reader.

- *References*: This section provides all the different referenced tools, methodologies, related work, research information and topics used by this document.

- *Appendices*: This section offers additional information to help readers gain a deeper understanding of the topics discussed in the document. It provides supplementary material for further exploration and clarification.

2. Background & Related Work

This section provides an in-depth analysis of the literature review conducted, aimed at identifying solutions and gaining a deeper understanding of the problem statement. The section is divided into subsections to describe related work on different concepts that are related to the Thesis analyzed in this document. The following subsections regard different Machine Learning techniques that exist, since some are being used by the developed platform. For experimentation purposes, there was a need to use Synthetic Data, thus this is a concept that regards this document heavily. EverCluster is a platform that follows the Controller – Worker model by the use of containerization. Finally, the platform itself is a recommendation system and it is using the infrastructure's network to its advantage. By reading the following subsections, an understanding of similar and core concepts of the work that is described farther on can be achieved.

2.1. Machine Learning

A subfield of Artificial Intelligence (AI) called Machine Learning (ML) focuses on creating machines that can recognize patterns in data, learn from them, and make judgments with little to no help from humans. A thorough introduction of ML, including its fundamental ideas, important algorithms, and many applications, can be found in the survey that follows [4]. The salient features of the survey are enumerated here.

Based on their preferred methods of learning, the various ML algorithms are grouped in the survey. The following are the various styles that are discussed:

- *Supervised Learning*: Trains models with labeled data (e.g., Spam Detection [5]).
- *Unsupervised Learning*: Identifies patterns in unlabeled data (e.g., Clustering [6]).
- *Semi-Supervised Learning*: Combines labeled and unlabeled data [7].
- *Reinforcement Learning* (RL): Learns actions based on rewards (e.g., Game Strategies [8]).

According to the survey, the goal of ML is to create algorithms that allow computers to learn from data and gradually become more efficient without the need for explicit programming. It is a fusion of statistics and computer science that uses data to categorize information, identify trends, and make predictions. Research Topics that are relevant to ML are the following: (i) using unlabeled data in Supervised Learning, (ii) transferring learning experience [9], (iii) linking different ML algorithms [4], (iv) privacy-preserving Data Mining [10] and (v) never-ending learning [11].

Significant research interest is directed toward distributing ML tasks over clusters to handle large datasets and complex models [12]. While frameworks like Hadoop [13] offer simplified distributed programming, they often fail to achieve the performance of specialized ML implementations [12]. The Parameter Server (PS) paradigm [12] provides a middle ground, allowing the distribution of ML programs with asynchronous parameter updates through relaxed consistency models.

2.1.1. Unsupervised Learning Algorithms

The survey [4] explains Unsupervised Learning algorithms as procedures that take for input unlabeled data and output patterns or clusters of the given dataset. An ML task that is categorized as Unsupervised Learning is Clustering. Clustering algorithms are procedures that are used to establish patterns in datasets and label the data appropriately in different clusters. Below are some of the Clustering Algorithms that regard this document:

- *K-Means* [14]: Assigns each data point to exactly one cluster based on distance from the centroid.
- *Bisecting K-Means* [15]: Recursively splits data into clusters starting from one large cluster.
- *Gaussian Mixture Model (GMM)* [16]: Models data with multiple Gaussian distributions and assigns probabilities for membership in clusters.

In Appendices I, II and III an explanation on each of the three algorithms is shown using pseudocodes. From the three algorithms, the most known is K-means and tends to be used as a starting example for Clustering ML tasks. In the research paper [3] a comparison between simple K-means and Bisecting K-means is performed. The conclusions of the paper are that (i) Bisecting K-means is more efficient than simple K-means especially for large datasets. This is because it breaks down the data into smaller clusters incrementally, whereas simple K-means requires multiple iterations to potentially achieve the same results. (ii) Bisecting K-means provides better quality clusters based on the similarity and dissimilarity measure of the cluster quality. This is likely because it refines the clusters throughout the splitting process. (iii) Both algorithms are susceptible to the initial selection of centroids. In other words, the final clustering results can vary depending on where the initial centroids are placed.

K-means is also being compared to the GMM algorithm in the following research work [17]. The conclusions of the paper are that (i) K-means is a simpler and faster clustering method but may not capture the true heterogeneity [18] of cloud workloads. It assigns data points to the nearest cluster centroid based on distance, resulting in spherical clusters. (ii) GMM is a more complex probabilistic method that can better model the heterogeneity of cloud workloads by capturing complex patterns and grouping them into distinct clusters. It utilizes a statistical approach to represent clusters as Gaussian distributions, allowing for clusters with various shapes and sizes. (iii) GMM may require more computation time compared to K-means. This is because it involves an iterative process to estimate the parameters of the Gaussian distributions for each cluster.

2.1.2. Supervised Learning Algorithms

The survey [4] explains that Supervised Learning algorithms use labeled data to train an algorithm to predict outcomes for new, unseen data. The model learns the relationship between input features and their corresponding labels. An ML task that is categorized as Unsupervised Learning is Classification. Thus, a classification algorithm is used to categorize data into predefined classes or labels. It analyzes input labelled data and assigns them to one of several possible discrete categories. In the review that was performed [19] the following classification algorithms are explained:

- K-Nearest Neighbor (KNN): Assigns a class to a data point based on the majority class among its nearest neighbors in the feature space.
- Support Vector Machine (SVM): Finds a hyperplane that best separates data points of different classes, maximizing the margin between them.
- Random Forest (RF): Constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) from all the trees.

- Neural Network (NN): Simulates the human brain using layers of interconnected nodes (neurons). It learns patterns in data by adjusting the weights of these connections based on errors in predictions.

In Appendices IV, V, VI and VII an explanation on each of the three algorithms is shown using pseudocodes. In conclusion, the review provides a comprehensive comparison of the classification algorithms that where explained. KNN, while simple and intuitive, can be computationally inefficient for large datasets due to its need to compare each data point to all others in the training set. SVM, known for its ability to handle high-dimensional data and its robustness to noise, can be computationally expensive to train, especially for complex problems. RF, an ensemble method that combines multiple decision trees, offers a good balance between accuracy and interpretability. However, its complexity can make it challenging to understand the underlying decision-making process. NNs, on the other hand, are highly flexible and can learn complex patterns in the data but require careful tuning of hyperparameters to avoid overfitting. The choice of the best algorithm for a particular task depends on factors such as the size of the dataset, the complexity of the problem, and the desired level of interpretability.

2.2. Synthetic Data Generation

Synthetic data generation has become a critical method in fields such as ML, data analysis, and privacy preservation [20]. It involves the creation of artificial datasets that mimic real-world data distributions while addressing limitations in accessibility, scalability, or privacy concerns. The generation of synthetic data allows researchers and practitioners to train models, test algorithms, and conduct experiments without compromising sensitive information or violating ethical standards. Moreover, synthetic data provides flexibility in designing datasets that may not be available, allowing for controlled experiments and stress testing of ML models. Techniques such as noise injection [21] and deep generative models, such as Generative Adversarial Networks (GANs) [20], are widely used to create these datasets. The synthetic data often mirrors key properties of real-world datasets, making it a valuable tool in fields like healthcare [22], finance [23], and autonomous systems [20].

The use of synthetic data generation has gained significant traction as a means to address the challenges posed by data scarcity and privacy concerns. As highlighted in [22], synthetic data generation is particularly promising in healthcare, where the need for unbiased data and sufficient sample sizes is paramount. Their comprehensive review, focusing on diverse types of medical data, including tabular, imaging, radiomics, time-series, and omics data, emphasizes the growing importance of synthetic data in various applications within the medical field. Through a systematic analysis of studies retrieved from PubMed and Scopus, the authors of [22] categorize synthetic data generation techniques into statistical, probabilistic, ML, and deep learning methods. Particularly, the review [22] reveals that 72.6% of the studies examined utilize deep learning-based synthetic data generators, demonstrating the dominance of these methods in healthcare-related synthetic data generation. Python [24] emerged as the most commonly used programming language, with 75.3% of the generators being implemented in this environment. The findings indicate that synthetic data is primarily used to reduce the cost and time associated with clinical trials for rare diseases, enhance the predictive power of AI models in personalized medicine, and enable fair treatment recommendations across diverse patient populations.

In the financial domain, synthetic data generation has proven to be a valuable tool for improving the performance of ML models, particularly in asset price prediction systems. [23] explored various financial data synthesis methods, with a specific focus on augmented training data and its impact on ML models. Their work addresses the unique characteristics of asset price time sequences, which exhibit patterns in both the temporal and magnitude dimensions that typical statistical measures may fail to capture. One of the primary challenges in financial data synthesis, as noted by [23], is the need for high-fidelity synthetic data that preserves confidentiality while accurately replicating critical temporal patterns, such as financial bubbles or short-lived market changes. By generating augmented datasets that reflect these patterns, [23] demonstrated the potential for improving the predictive capabilities of ML models trained on such data. Their evaluation involved both quantitative and qualitative visual metrics, leading to the selection of SigCWGAN as the best-performing method in terms of fidelity. The financial time series data generated by SigCWGAN were used to train an ARIMA/RNN asset price prediction system, resulting in significant improvements in predictive accuracy, as measured by MAE, MSE, and MAPE metrics. This study highlights the importance of using synthetic data in financial prediction tasks, particularly when real-world data is scarce or sensitive. It also underscores the role of specialized GAN models, such as SigCWGAN, in generating realistic time-series data that enhance the performance of traditional ML models.

The insights from [23] align with broader trends in the use of generative models for synthetic data creation. For example, [23] applied GANs to financial datasets to capture complex dependencies between features, while [22] demonstrated the utility of probabilistic models in generating healthcare datasets. The combination of deep learning techniques and domain-specific knowledge, as illustrated in these studies, underscores the potential for synthetic data to improve predictive modeling across various fields.

In the field of autonomous systems, synthetic data generation plays a critical role in testing and optimizing infrastructure frameworks. [10] presents CODECO, a novel container orchestration framework designed for next-generation Internet applications across a mobile, heterogeneous Edge-Cloud continuum. While the focus of this work is on orchestrating data, network, and computing resources, the underlying approach aligns with decentralized AI techniques, which are increasingly applied in synthetic data generation for autonomous systems. CODECO emphasizes the selection of infrastructure based on user-defined performance profiles, such as resilience and environmental efficiency ("greenness"), making it highly adaptable to the fluctuating environments of autonomous systems. Infrastructural challenges, such as intermittent connectivity and node failure, are addressed through AI-driven orchestration, which ensures the optimal deployment of applications across the network. This dynamic orchestration method mirrors the challenges faced in autonomous systems, where synthetic data is often required to simulate unpredictable and complex environments, enabling robust AI model training and validation.

To address this issue, [10] introduces a synthetic data generator as part of the CODECO orchestration framework. This generator mimics the cross-layer data collection process from key CODECO components. The CODECO Data Generator, generates synthetic Kubernetes (K8s) [25] infrastructure data based on the framework's resource models and metrics monitoring architecture. By simulating the behavior of infrastructure components in a K8s environment, the Data Generator provides critical synthetic data that enables testing and evaluation of the CODECO orchestration framework before real-world deployment. This is especially useful in scenarios where real data is scarce or unavailable. The Data Generator consists of two main components: the Data Generator Collector, which gathers predefined metrics from K8s/Prometheus, and the Data Generator Synthesizer, capable of handling more complex metrics that require synthesis.

2.3. Containerization & Scaling

Containerization has revolutionized the way applications are developed, deployed, and managed in modern computing environments. A system that is built as a single, unified unit, where all its components are interconnected and interdependent has the monolithic architecture [26]. This architecture makes scaling or updating individual part challenging. By encapsulating software and its dependencies into isolated units called containers, this technology enables applications to run consistently across various environments, from development to production. Unlike traditional Virtual Machines (VMs) [27], which virtualize entire Operating Systems, containers share the host system's kernel, making them lightweight and more efficient in terms of resource usage. The rise of containerization, driven by platforms like Docker [28] and K8s [25], has provided developers with greater flexibility, portability, and scalability. Containers allow for the seamless migration of applications between different cloud environments, on-premises data centers, and even edge computing infrastructures. This has been particularly beneficial for microservices architectures, where applications are broken down into smaller, independent services that can be deployed and scaled individually. Furthermore, containerization enhances Continuous Integration and Delivery (CI/CD) pipelines [29], improving the speed and reliability of software development and deployment. It also simplifies version control, security, and resource management, ensuring that each container is isolated and protected while sharing the host system efficiently.

The growing adoption of containerization technologies has sparked significant research and development in both academia and industry, leading to numerous studies that highlight the benefits and challenges of container-based systems. In [30], the differences between hardware virtualization and Operating System-level virtualization are explored, with a particular focus on containers as a key Operating System (OS) virtualization technology. The study compares containers and traditional VMs in large data center environments along the dimensions of performance, manageability, and software development. The results show that while containers offer greater resource efficiency, particularly in multi-tenant environments, they are more susceptible to performance interference between co-located applications for certain workloads. The paper also evaluates management frameworks for both containers and VMs, highlighting the distinct capabilities each technology exposes for application deployment and orchestration. Additionally, the authors propose hybrid approaches that combine hardware and OS virtualization, showing promise for mitigating the performance trade-offs associated with containers.

In a related study, [31] delves into the integration of microservices architectures with containerization technologies like Docker and K8s. The paper highlights the synergistic benefits of microservices—decoupled, independently deployable services—and containerization, which provides lightweight, isolated environments for deploying and managing these services. By breaking down monolithic applications into smaller, loosely coupled services encapsulated within containers, organizations can achieve increased scalability, flexibility, and resilience. Through case studies and real-world examples, the study demonstrates how this architectural paradigm shift enables faster development cycles, more efficient resource utilization, and improved fault tolerance. This work emphasizes that the combination of microservices and containerization allows for more agile and scalable software development in dynamic and complex environments.

Expanding on the above, [32] explores the role of containerization in the resource-constrained environments of edge computing. The study focuses on how edge computing moves computation closer to the data source, reducing network bandwidth usage, latency, and improving response times. In particular, the paper proposes a containerization-based architecture for deploying and managing deep learning models on edge devices, especially in smart home environments. The architecture allows deep learning models to be containerized, offering advantages such as minimal space requirements, device independence, and low latency. Experimental results demonstrate that the performance of the containerized deep learning models in terms of execution time and Central Processing Unit (CPU) load is comparable to native deployments, while providing significant benefits in terms of deployment ease and cross-platform compatibility.

Furthering the discussion on containerization technologies, [33] compares two of the most widely used container engines, Docker and Podman [34]. The paper highlights the role of container engines in modern technological infrastructures, especially in enabling standardized execution runtimes and reducing costs through virtualization techniques. A series of benchmark tests are conducted using a custom-built tool to measure differences in performance between Docker and Podman. The results show a small but not insignificant performance difference in favor of Docker. Additionally, the paper compares real-world metrics and cloud pricing to demonstrate the potential cost differences in deploying these engines at scale, noting that Docker is slightly more cost-effective in cloud environments.

Together, these works reflect the ongoing evolution of containerization and its broad applicability across cloud computing, microservices, edge computing, and container engines. They highlight the significant advantages of container-based systems, as well as the challenges related to performance, resource management, and cost-efficiency that continue to drive innovation in this space.

2.4. Controller – Worker Architecture

A significant advantage of containerization is how well it integrates with the Controller-Worker Architecture. In a containerized environment, the Controller can manage the orchestration of various Worker containers, ensuring optimal task distribution and resource utilization. Platforms like K8s [25] exemplify this by acting as a Controller, scheduling and distributing workloads across containers (Workers) based on available resources, while ensuring that each service runs in isolation and remains highly scalable. This synergy between containerization and the Controller-Worker architecture allows for the efficient scaling of services in cloud-native applications, enabling automated scaling, fault tolerance, and resource management.

The Controller-Worker architecture has become a fundamental model for distributed computing, enabling efficient task allocation and resource management in large-scale systems. This architecture is commonly used in systems where tasks need to be divided into smaller sub-tasks (handled by workers) and coordinated through a central controller that monitors and manages the execution. In recent years, research has focused on optimizing the communication, scalability, and resilience of this architecture, especially in domains such as cloud computing [10][35], Big Data [35][36], and ML infrastructure [35][36].

One notable project using the Controller-Worker architecture in a cloud computing scenario is the CODECO framework introduced by [10]. CODECO adopts a decentralized AI-driven approach to manage tasks across a mobile, heterogeneous Edge-Cloud continuum. The framework relies on the Controller-Worker model to define the infrastructure for next-generation Internet applications, where the controller dynamically orchestrates resources based on performance profiles like resilience and environmental efficiency. This system addresses infrastructural challenges, including intermittent connectivity and node failure, by utilizing AI to select the most suitable infrastructure for application deployment. CODECO's reliance on decentralized intelligence is a key advancement in the flexibility of the Controller-Worker architecture, allowing for real-time adaptation to changing conditions.

The Controller-Worker architecture has found extensive applications in managing and processing Big Data. This architecture's ability to efficiently allocate tasks and resources makes it well-suited for handling the complexities of large-scale data environments. In the domain of Big Data, [36] introduces EverAnalyzer, a self-adjustable Big Data management platform that leverages various frameworks such as Hadoop MapReduce, Mahout, Spark, and MLlib. EverAnalyzer utilizes a Controller-Worker model to manage data collection, processing, and analysis. The platform collects data in both streaming and batch modes and employs a Controller to analyze metadata from user processes to recommend the most suitable data processing framework. This recommendation system is particularly effective, achieving optimal framework selection in 80% of tested scenarios. The Controller-Worker architecture in EverAnalyzer allows for dynamic adjustment and optimization of data processing tasks, ensuring that the most appropriate framework is utilized based on real-time data and user needs.

The project DIASTEMA [35] provides a single-entry point for both technical and non-technical users to process and analyze large datasets. The Controller in Diastema manages the orchestration of various components, facilitating horizontal scaling of processes across cloud environments and computing clusters. This approach not only improves time efficiency but also ensures energy efficiency by optimizing resource utilization. The Workers in this architecture handle specific tasks related to data processing, building analytical procedures, and visualizing results, demonstrating the architecture's ability to manage end-to-end Big Data scenarios effectively. This implementation of the Controller-Worker architecture in Diastema aligns with other applications of the model in Big Data environments. For example, the EverAnalyzer platform also employs a Controller-Worker approach to recommend optimal data processing frameworks based on metadata analysis, further illustrating the architecture's flexibility and effectiveness. Additionally, in autonomous systems, frameworks like CODECO utilize the Controller-Worker model to handle resource orchestration and task management across distributed environments, showcasing its adaptability to various data-intensive applications. The advancements presented in Diastema highlight the ongoing evolution of the Controller-Worker architecture in addressing the complexities of Big Data analytics. By providing scalable, user-friendly solutions and optimizing resource utilization, Diastema contributes to the broader effort to democratize data analytics and enhance its applicability in rapidly growing data domains such as healthcare.

2.5. Network Topologies

Network topologies are fundamental to the design and functionality of both physical [37] and virtual networks [37], influencing their efficiency, scalability, and resilience. The study of network topologies encompasses a range of structures and configurations. In this section, there is a quick overview of the different possible network configurations that can be achieved.

In Meador Brett's survey [37], different network topologies designed to connect various network nodes are discussed. The survey explains that a network topology, which includes the different nodes, routers, and links used, is referred to as a Physical Network Topology. When the focus is not on the hardware used, but rather on representing the topology solely with its nodes and links, it is referred to as a Logical Network Topology (also known as Signal Topology). The survey describes various network topologies based on each one's Physical Topology. It provides an overview of how these topologies differ in their physical arrangements of nodes, routers, and links, and discusses their advantages and limitations in terms of connectivity, scalability, and fault tolerance. The analysis includes the following topologies: (i) bus, (ii) ring, (iii) star, (iv) tree, (v) tree-bus and (vi) mesh. By examining the physical layout of each topology, the survey helps to understand how the arrangement of components impacts overall network performance and reliability.

The choice of network topology has significant implications for various domains, including telecommunications, computer networks, and distributed systems. Traditional topologies, such as star and ring, offer straightforward and manageable configurations but may lack the flexibility required for modern, high-performance systems. As network demands grow, newer and more sophisticated topologies, such as hypercubes [38], torus [39], and hierarchical structures [40], are being explored to meet the needs of large-scale and distributed environments.

2.6. Network Focused Projects

The following Research Projects focus on using Deep Reinforcement Learning (DRL) for network optimization, particularly in routing problems, but with different approaches and specific focuses.

The work of Gyungmin, Yohan and Hyuk [41] explores how DRL can be applied to optimize routing in Software-Defined Networks (SDNs), focusing on reducing end-to-end delay and packet losses. The proposed solution uses a Deep Deterministic Policy Gradient (DDPG) algorithm combined with an M/M/1/K queue-based network model to overcome the long learning process of DRL and mitigate performance degradation during network topology changes. The agent is trained offline to generate an optimal routing policy, and simulations show that this DRL-based method outperforms conventional hop-count routing methods and traffic demand-based RL algorithms. The core contribution lies in the use of offline learning to avoid disrupting real network performance during training

The paper of Paul et al. [42] addresses a major challenge in DRL-based networking: the lack of generalization across different network topologies. The authors propose integrating DRL with Graph Neural Networks (GNNs) to enable generalization to unseen network topologies. GNNs can handle graph-structured data like network topologies, which traditional NNs struggle with. The DRL+GNN agent was evaluated on various real-world and synthetic network topologies, showing better generalization and performance compared to state-of-the-art DRL methods that do not incorporate GNNs

Both projects aim to improve network routing using DRL but tackle different challenges: the first focuses on reducing training times and avoiding real-time network degradation, while the second addresses the issue of generalization across different topologies using GNNs.

2.7. Recommendations Focused Projects

Recommendation systems are a critical tool in modern data-driven environments, especially for optimizing analytics and personalizing user experiences across various domains. These systems analyze large datasets to provide tailored content, product suggestions, or actionable insights, enhancing decision-making and user satisfaction. As data continues to grow in volume and complexity, the importance of refining recommendation systems for efficiency and accuracy cannot be overstated.

In collaborative filtering, one of the most widely used techniques, [43] revisits the effectiveness of traditional embedding-based models, focusing on how simple dot product operations can outperform more complex Neural Collaborative Filtering (NCF) approaches. The findings of this research suggest that, despite the rise of deep learning methods, optimizing fundamental operations like dot products can yield better performance, particularly in resource-constrained environments where retrieval efficiency is paramount.

Similarly, [44] investigates the growing influence of deep learning in recommendation systems. This work highlights how deep learning models, which automatically learn feature representations, have revolutionized the field by significantly improving the ability to handle complex and vast datasets. The paper reviews the recent developments showing how deep learning models have become foundational in web-based recommendation systems, enhancing scalability, personalization, and real-time adaptability.

Tackling the persistent challenge of data sparsity, [45] presents an in-depth review of methods designed to mitigate the impact of sparse datasets on recommendation accuracy. Various similarity measures, innovative approaches are explored, as well as the use of side information to address the issue, offering valuable insights into improving the robustness of recommendation systems when data is limited. The authors underscore the necessity of enhancing system performance in environments where user-item interactions are minimal.

These contributions showcase the versatility and critical impact of recommendation systems, from optimizing traditional collaborative filtering models to integrating advanced deep learning techniques and addressing domain-specific challenges like data sparsity and healthcare personalization.

3. Implemented Tools & Methodologies

This section provides a comprehensive overview of the diverse tools and methodologies employed in the development of the platform. The construction of EverCluster necessitated the seamless integration of various components, including multiple engines, databases, programming languages, frameworks, libraries, and analytical algorithms. The implementation of these elements was executed with meticulous precision to ensure the robustness and efficiency of the platform. For the sake of clarity and coherence, this section is structured into distinct subsections, each dedicated to a specific technical aspect essential for comprehending the platform's architecture. These technical intricacies are further elaborated upon in the subsequent "Platform Architecture Analysis" section.

3.1. Engines

3.1.1. Docker

Docker [28] is an open-source platform designed to automate the deployment, scaling, and management of applications using containerization. Containers are lightweight, portable, and self-contained units that package an application's code along with its dependencies, libraries, and environment settings. This allows applications to run consistently across different computing environments, whether it's on a developer's local machine, a test server, or in the cloud. Docker containers isolate applications from one another and from the underlying system, making them highly efficient in terms of resource utilization. This contrasts with traditional virtualization methods, where each application runs in its own VM with a full OS, leading to greater resource consumption.

Key features of Docker [46]:

- *Portability*: Docker containers can run consistently on different environments.
- *Isolation*: Each container runs in its own isolated environment.
- *Efficiency*: Containers are lightweight and require fewer resources compared to VMs.
- *Version Control*: Docker images allow versioning and can be updated, rolled back, or shared.

A native tool of Docker is Docker Swarm [47]. With Docker Swarm, orchestrating and managing a cluster of Docker containers is achieved. It allows scaling up applications by distributing containers across multiple machines, while maintaining centralized management. In Swarm mode, multiple Docker hosts can work together as a single virtual system, creating a more robust environment for deploying and managing containers.

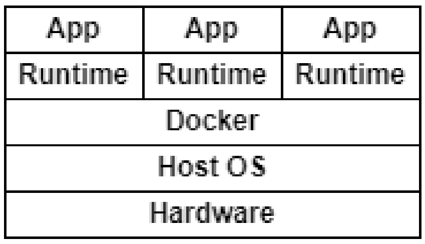


Figure 2 – Docker Architecture

Key features of Docker Swarm [48]:

- *Cluster management*: Turns multiple Docker hosts into a single, unified cluster.
- *Scaling*: Easily scale services up or down by adding or removing containers.
- *Load balancing*: Distributes network traffic across different containers to ensure smooth performance.
- *Service discovery*: Automatically detects and configures services running in the cluster.

A notable mention that goes hand in hand with Docker Swarm, is K8s [25]. K8s is an open-source platform for automating container orchestration, including deployment, scaling, and management of containerized applications. Developed by Google [49] and now maintained by the Cloud Native Computing Foundation [50], K8s is known for its robust features and ability to handle large-scale, complex environments. K8s organizes containers into groups called pods [51], which represent the smallest deployable units in the system. It manages the scheduling and lifecycle of these pods, ensuring that they are running smoothly, and automating tasks like scaling, failover, and rolling updates.

Key features of K8s [52]:

- *Automated deployment and scaling*: K8s automatically manages the desired state of an application, scaling it based on resource usage.
- *Self-healing*: Automatically restarts failed containers, replaces them, and reschedules them to maintain the desired state.
- *Service discovery and load balancing*: Manages networking between containers and distributes traffic for optimal performance.
- *Rolling updates and rollbacks*: Updates applications without downtime and rolls back if something goes wrong.

3.1.2. Spark

Apache Spark [53] is an open-source, distributed computing system used for big data processing and analytics. It provides a fast and general-purpose engine that allows developers to process large datasets across a cluster of computers. Spark is designed to handle data processing tasks in a highly parallel and efficient manner, making it ideal for tasks such as data transformation, ML, and real-time stream processing. Spark's ability to process data both in-memory and on disk allows it to outperform traditional batch-processing systems like Hadoop MapReduce [54] by orders of magnitude, particularly for iterative tasks. It supports a variety of programming languages, including Java [55], Scala[56], Python [24], and R [57].

The core of Apache Spark [53] is handled by the Spark Engine and is responsible for executing the various tasks involved in distributed data processing. It abstracts away the complexity of parallelization, fault tolerance, and task distribution, making it easier for developers to write applications that can scale across multiple machines. The Spark Engine breaks down jobs into smaller tasks and distributes them across a cluster of machines for parallel execution. It manages the scheduling, execution, and coordination of these tasks. One of the key components of the Spark Engine is its ability to handle in-memory processing [58], which significantly speeds up iterative algorithms and interactive data analysis tasks.

As described above, Spark Can be extended by using multiple machines at the same time, creating a Spark Cluster [59]. A Spark Cluster is a group of machines working together to execute Spark jobs in parallel. It typically consists of a master node and multiple worker nodes.

- Master Node: The master node is responsible for managing the cluster and coordinating the execution of Spark jobs. It breaks down the job into smaller tasks, distributes them to the worker nodes, and tracks the progress of these tasks.
- Worker Nodes: The worker nodes execute the tasks assigned by the master node. Each worker node has its own set of resources (CPU, memory) and is capable of running multiple tasks in parallel.

The communication between the master node and the worker nodes allows the cluster to process data in parallel, significantly speeding up data processing for large datasets. The cluster can be scaled by adding more worker nodes, making it suitable for handling massive data volumes in distributed environments.

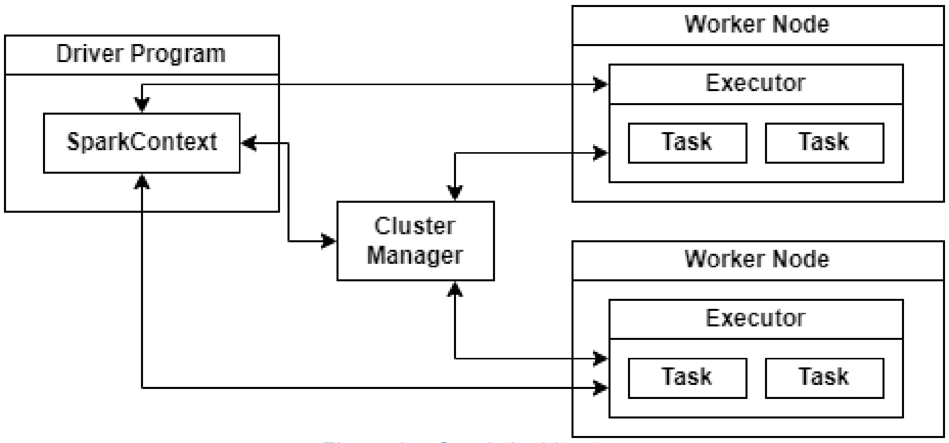


Figure 3 – Spark Architecture

3.2. Databases & Object Storages

The platform utilizes information provided by the end-user, alongside the supplied datasets and computed data, to facilitate its operational capabilities. To implement this functionality, the integration of various databases was required. This subsection offers an analysis of the Mongo [60] and MinIO [61] Database Management Systems (DBMSs).

3.2.1. SQL & NoSQL

A database is a structured collection of data that is stored electronically, managed, and accessed efficiently. Databases enable users and applications to store, retrieve, update, and manage large volumes of data. They play a crucial role in modern applications, ranging from personal use to enterprise-level systems. Databases are categorized based on how they structure data and how they manage access, with two broad types being relational (Structured Query Language – SQL) [62] and non-relational (Non-Structured Query Language – NoSQL) [62] databases.

SQL databases are structured in a tabular form, where data is stored in rows and columns. They are based on a predefined schema, which enforces strict data integrity and relationships between data tables through foreign keys. Key aspects of an SQL Database are the following [62]:

- *Schema*: SQL databases require a well-defined schema (structure) before data is entered.
- *ACID Compliance*: SQL databases adhere to the Atomicity, Consistency, Isolation, Durability (ACID) properties, ensuring high data reliability and transactional integrity.

- *Query Language*: SQL (Structured Query Language) is the standard language used to query and manipulate relational databases.

Some examples of such Databases are the following: MySQL [63], PostgreSQL [64], Oracle [65], and Microsoft SQL Server [66]. Appendix VIII showcases the pros and cons of using an SQL Database based on [62].

NoSQL databases break away from the rigid table structures of SQL and offer flexible, schema-less storage [62]. These databases are designed to handle unstructured [62] or semi-structured data [62] and scale horizontally easily [62]. Key aspects of an SQL Database are the following:

- Schema: NoSQL databases can handle data without requiring a predefined schema.
- BASE: These systems often follow the Basically Available, Soft state, Eventually consistent (BASE) model, which allows for flexibility but sacrifices immediate consistency for better scalability and availability.
- Data Models: NoSQL databases use various models, including key-value pairs, document stores, column families, and graph structures.

Some examples of NoSQL Databases are the following: MongoDB [60], Cassandra [67], Redis [68], Neo4j [69]. Appendix IX showcases the pros and cons of using a NoSQL Database based on [62]. For the development of the platform described in this document, the Mongo DBMS is being used to handle data in a NoSQL manner.

3.2.2. Object Storages

Object Storages [70] are Databases for managing and storing data as objects, as opposed to traditional file storage [71] (which organizes data in a hierarchical structure) or block storage [72] (which stores data in fixed-sized blocks). Each object consists of the data itself, metadata about the data, and a unique identifier. Object storage is ideal for handling large amounts of unstructured data such as multimedia files, backups, and large datasets. It is highly scalable and suited for cloud-native applications that require efficient management of vast quantities of data. For the development of the platform described in this document, MinIO [61] is being used as an Object Storage.

Characteristics of Object Storage [73]:

- *Scalability*: Object storage can scale to accommodate enormous amounts of data.
- *Metadata-Rich*: Objects can have customizable metadata to provide context and make searching easier.
- *Flat Structure*: Unlike traditional file systems, object storage uses a flat namespace without directories or hierarchies, which improves accessibility and scalability.

3.3. APIs

An Application Programming Interface (API) [74] is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests and responses should be structured, enabling programs to share data or functionality without needing to understand the internal workings of each other. Essentially, an API acts as a bridge between systems, allowing them to work together seamlessly. There are two common types of APIs: Representational State Transfer (REST) [75] and Simple Object Access Protocol (SOAP) [76]. APIs are a fundamental aspect of the intercommunication of modern Information

Systems. In the architecture of the developed platform, the usage of Spark APIs [77] and Flask APIs [78] are described in detail.

3.3.1. REST

REST [75] is an architectural style for designing networked applications. It relies on a stateless, client-server communication model where requests are made via standard HTTP methods like GET, POST, PUT, and DELETE. Key characteristics of REST APIs include:

- *Stateless*: Each request from a client to a server must contain all the information needed to process the request. The server does not store the client's previous state.
- *Resource-based*: REST APIs focus on resources (such as users, files, or posts), which are represented via Uniform Resource Locators (URLs) (e.g., https://api.example.com/users).
- *Lightweight*: REST APIs commonly exchange data in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format, with JSON being more popular due to its simplicity and readability.
- *Scalability*: REST is easy to scale due to its stateless nature, making it a preferred choice for web services and microservices.

3.3.2. SOAP

SOAP [76] is a protocol that defines how to communicate between different services over a network. It uses XML to encode its messages and relies on a more rigid and formal structure than REST. Key characteristics of SOAP APIs include:

- *Strict standards*: SOAP uses standardized XML messaging and has built-in error handling, making it ideal for complex, enterprise-level applications.
- *Protocol-based*: Unlike REST, which uses HTTP, SOAP can operate over different protocols like HTTP, SMTP, or TCP.
- *Security*: SOAP has built-in standards for security (WS-Security) and transactions, which makes it a popular choice in applications requiring high security and reliability, such as banking and payment systems.
- *Stateful or stateless*: SOAP can operate in both stateful and stateless modes, depending on the requirements of the interaction.

3.4. Programming Languages

For the development of the platform analyzed in this document, there are some programming languages that have been used in order to create the back-end and the front-end. The front-end is the part of a website or app that users interact with directly, like buttons, text, and images. It's what the user sees on their screen and includes the design and layout. The back-end is everything behind the scenes that powers the front-end. It handles the logic, database, and server, making sure data is processed and sent correctly to the front-end. In the back-end of the platform, Python and its Libraries are being exploited. In the front-end of the platform many common programming and structure languages are being used in other order to provide a User Interface (UI) to the end-users. In this section, the different Programming Languages are described to the reader.

3.4.1. HTML

HyperText Markup Language (HTML) [79] is the foundational markup language used for creating the structure of web pages. It serves as the backbone of all websites, allowing developers to define and organize content into various elements such as headings, paragraphs, images, links, and more. HTML utilizes a system of tags to indicate different content types, such as <h1> for headings or <p> for paragraphs, with attributes often used to provide additional context or functionality. While HTML structures the content, it does not dictate its appearance or behavior, making it essential to pair with other technologies like CSS and JavaScript (JS) for a complete web experience as explained in the following subsections.

3.4.2. CSS

Cascading Style Sheets (CSS) [80] is the language used to control the visual appearance and layout of web pages. By applying styles to the HTML structure, CSS allows developers to define how elements such as text, images, and containers are displayed. Through properties like colors, fonts, spacing, and positioning, CSS helps create visually appealing websites. Additionally, CSS is responsible for responsive design, enabling web pages to adapt to various screen sizes and devices. It can be included directly within an HTML document or linked as an external stylesheet, making it a flexible tool for maintaining consistent design across multiple pages.

3.4.3. JS

JS [81] is a scripting language primarily used to add interactivity and dynamic behavior to websites. Unlike HTML and CSS, which focus on structure and design, JS enables developers to make web pages interactive by responding to user inputs, manipulating the Document Object Model (DOM), and handling events like mouse clicks or form submissions. JS also supports asynchronous operations, allowing web applications to load data in the background without requiring a page refresh. While originally developed for client-side execution in web browsers.

3.4.4. Python

Python is a high-level, general-purpose programming language known for its simplicity and versatility. It's clear, readable syntax makes it a popular choice for beginners and experienced developers alike. Python is widely used in fields such as web development [82], data science [83], AI [83], automation [10], and more. It supports multiple programming paradigms, including object-oriented [84], procedural [84], and functional programming [84], and has an extensive standard library as well as third-party packages that extend its functionality. Python's adaptability makes it suitable for a broad range of applications, from small scripts to complex ML systems.

3.5. Frameworks & Libraries

To implement the architecture detailed in the next section, a variety of frameworks and libraries were employed to enhance code readability and leverage their built-in functionalities. This section provides an overview of the most significant frameworks and libraries utilized in the development of EverCluster.

3.5.1. jQuery

jQuery [85] is a JS library designed to simplify the process of scripting HTML documents, handling events, and creating animations. JQuery aims to reduce the amount of code developers need to write and manage, making it easier to achieve complex interactions and effects with fewer lines of code. One of jQuery's most notable features is its ability to handle DOM manipulation effortlessly. Developers can select elements from a web page, modify their content, and apply styles with concise and readable code. Additionally, jQuery provides a straightforward API for handling events, such as user clicks or keyboard inputs, and supports animations like fading and sliding.

jQuery is also known for its cross-browser compatibility, meaning it addresses inconsistencies between different web browsers, ensuring that code runs smoothly across various platforms. Despite the rise of modern JS frameworks, jQuery remains popular for its simplicity and ease of use, particularly in legacy projects and for developers seeking a lightweight solution.

3.5.2. Bootstrap

Bootstrap [86] is an open-source front-end framework. Bootstrap provides a comprehensive set of design templates and components that facilitate the creation of responsive and visually appealing web interfaces. Bootstrap's core feature is its responsive grid system, which allows developers to create layouts that automatically adjust to different screen sizes and device orientations. This grid system is complemented by a variety of pre-designed components, including navigation bars, buttons, forms, and modals, which streamline the development process and ensure consistency in design.

In addition to its grid and component library, Bootstrap includes a set of customizable CSS and JS utilities for styling and interactivity. The framework supports modern web standards and is designed to be easily extendable and adaptable to various design needs.

3.5.3. Flask

Flask [78] is a micro web framework for Python. It is designed to be lightweight and flexible, providing developers with the core tools needed to build web applications without imposing a lot of constraints or requiring a predefined project structure. Flask's minimalist approach means it does not include built-in components like database abstraction layers or form validation, allowing developers to choose and integrate their preferred tools and libraries. Despite its simplicity, Flask is highly extensible and supports various extensions that can add functionality such as authentication, database integration, and form handling.

One of Flask's key features is its built-in development server and debugger, which makes it easy to test and debug applications during development. Flask uses Jinja2 [87] as its templating engine and Werkzeug [88] as its Web Server Gateway Interface (WSGI) toolkit, providing a solid foundation for creating dynamic web applications. Its straightforward design and ease of use make it a popular choice for both beginners and experienced developers working on small to medium-sized projects.

3.5.4. PySpark

PySpark [89] is the Python API for Apache Spark. With PySpark, users can perform data transformations, run complex queries, and build ML models on large datasets distributed across a cluster of machines. PySpark provides DataFrame and Resilient Distributed Dataset (RDD) APIs, which enable efficient data manipulation and querying. The DataFrame API offers a high-level abstraction for working with structured data [62], while the RDD API provides more control over low-level data operations.

PySpark integrates with various data sources, including Hadoop Distributed File System (HDFS) [13], Amazon Simple Storage Service (S3) [90], and various databases, making it a versatile tool for data engineering and data science. It also includes a ML library called MLlib [91], which provides algorithms for classification, regression and clustering.

3.5.5. Pandas

Pandas [92] is a powerful and flexible data manipulation library for Python. It provides two primary data structures—Series and DataFrame—that are designed for working with structured data in a tabular format [93]. The Series data structure represents one-dimensional labeled arrays, while the DataFrame is a two-dimensional table with labeled axes (rows and columns). These structures enable efficient data manipulation, including operations such as filtering, grouping, merging, and reshaping. Pandas also offers a range of functions for reading from and writing to various data formats, including Comma-Separated Value (CSV) files, Excel, SQL databases, and more.

3.5.6. Scikit-learn

Scikit-learn [94] is a robust ML library for Python. It is built on top of NumPy [95], SciPy [96], and Matplotlib [97], providing a consistent and easy-to-use interface for various ML algorithms and techniques. Scikit-learn includes a comprehensive collection of tools for data preprocessing, model selection, and evaluation. It supports a wide range of algorithms for classification, regression and clustering.

One of the key strengths of Scikit-learn is its emphasis on user-friendly APIs and clear documentation, which makes it accessible for both beginners and experienced practitioners. Its integration with other scientific libraries in Python, such as NumPy and Pandas, allows for seamless data manipulation and analysis.

3.6. ML Algorithms Used

In this subsection, a comprehensive description of the ML algorithms employed in the development of EverCluster is presented. These algorithms were essential to the platform's evolution and functionality. Furthermore, the clustering algorithms detailed are made available to the end-users of the platform, allowing them to leverage these tools for their own analytical needs.

3.6.1. Random Forest Classification

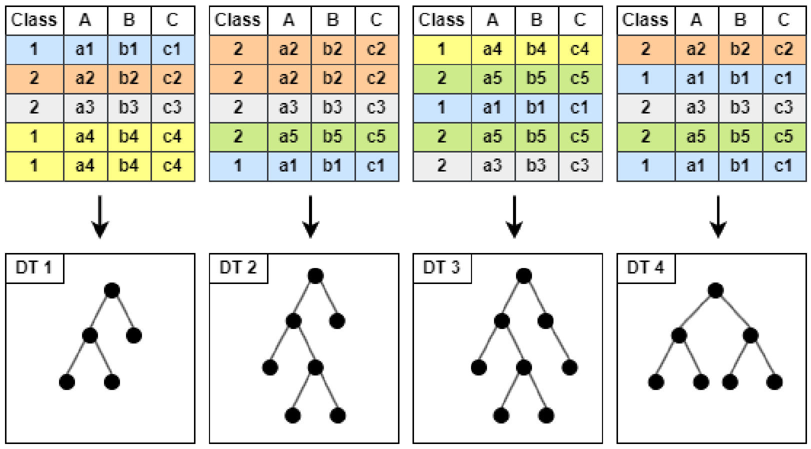
As described in *Section 2*, the RF classification algorithm is a supervised ML algorithm. To create a RF Classifier and predict the class of a specific instance (record or dataset) there is a need of training dataset. The algorithm of this classification method is also shown as pseudocode in Appendix VI. To make this process understood by the reader, below is an example of this classification method with figures.

As explained, the algorithm needs a Training Dataset, like the one shown in Table 1. The training dataset must be annotated, meaning that the class of each row mast be noted beforehand (Class). For each row the row is presented with a different color for better explainability regarding the next step. In this example four (4) trees are selected to be created and used to perform the RF. The first step of the algorithm is to create four (4) bootstrapped datasets. Bootstrapping [98] is a statistical technique that involves creating multiple samples from a single dataset. Each sample is generated by randomly selecting data points from the original dataset with replacement. This means that some data points may appear multiple times in a sample, while others may not appear at all. After creating the four (4) samples, a decision tree [99] is created for each one. In Figure 4, the above process is shown. After creating the decision trees the classifier is ready to be used. On a new instance (a new unlabeled row or rows) all the different decision trees output a class. Depending on the Use

Case and in the parameters that the algorithm is using, the classifier may choose to make to answer as the instance's label, the one with the most votes by the decision trees or using some other kind of statistical formula. The process for the decision on a new instance is shown in Figure 5.

Table 1 – RF Training Dataset

Class	Feature A	Feature B	Feature C
1	a1	b1	c1
2	a2	b2	c2
2	a3	b3	c3
1	a4	b4	c4
2	a5	b5	c5



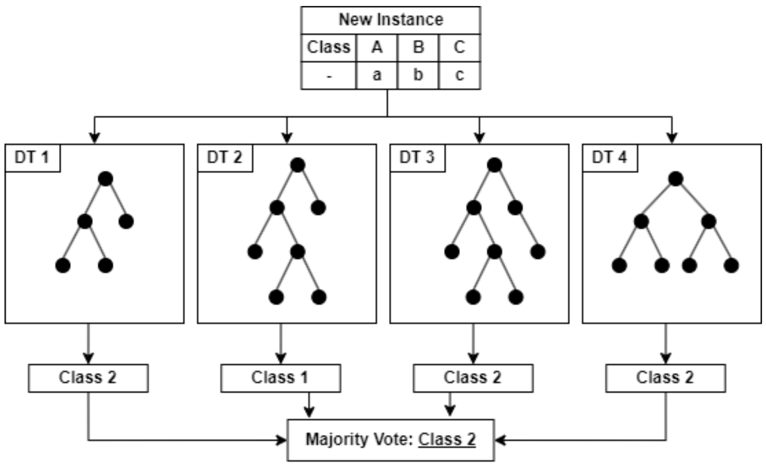
Class	A	B	C
1	a1	b1	c1
2	a2	b2	c2
2	a3	b3	c3
1	a4	b4	c4
1	a4	b4	c4

Class	A	B	C
2	a2	b2	c2
2	a2	b2	c2
2	a3	b3	c3
2	a5	b5	c5
1	a1	b1	c1

Class	A	B	C
1	a4	b4	c4
2	a5	b5	c5
1	a1	b1	c1
2	a5	b5	c5
2	a3	b3	c3

Class	A	B	C
2	a2	b2	c2
1	a1	b1	c1
2	a3	b3	c3
2	a5	b5	c5
1	a1	b1	c1

Figure 4 – Decision Trees



New Instance

Class	A	B	C
-	a	b	c

Figure 5 – RF Decision

3.6.2. K-means Clustering

As already mentioned, K-means clustering is used to partition a dataset into K distinct, non-overlapping groups or clusters. The goal is to group data points such that points within the same cluster are more similar to each other than to those in other clusters. The algorithm of this clustering method is also shown as pseudocode in Appendix I. To make this process understood by the reader, below is an example of this method with figures.

In Table 2 a dataset of five (5) two dimensional points are shown. These points are also represented in Figure 6. The color of each point is represented both in the table and the figure.

Table 2 – Example Clustering Dataset

ID	X	Y
0	0,0	0,0
1	1,0	1,0
2	9,0	8,0
3	8,0	9,0
4	0,5	0,5

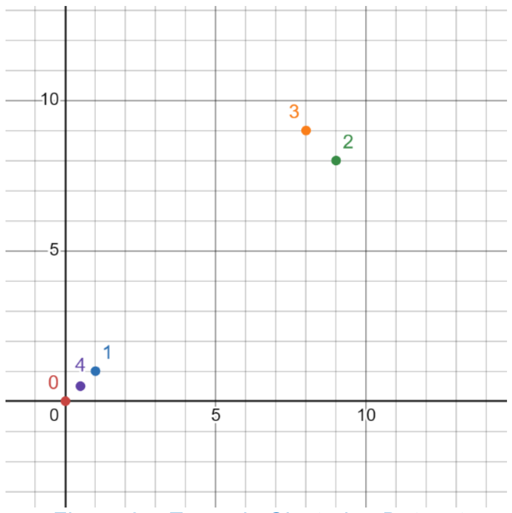


Figure 6 – Example Clustering Dataset

The K-means algorithm begins with K initial centroids. These centroids can be either two new spots, or two preexisting spots from the dataset. These two centroids could also be random or not, depending on the use case that the K-means algorithm is performed. For simplicity, the spots with the identifications (IDs) 1 and 3 are chosen. Then, the distance of each spot in the dataset from the centroids is calculated (Table 3) using Euclidian distance [100]. The spots nearest to each centroid are described as being a cluster. Thus, the first cluster is going to be the identifications (IDs): *0,1,4* and the second on the IDs: *2, 3*. The current clusters and centroids can be seen in Figure 7a.

Table 3 – K-means Centroid Distances

Spot ID \ Centroid Distance	Centroid 1 (1, 0)	Centroid 3 (8, 9)
0	1.4	12.04
1	0	10.63
2	10.63	1.41
3	10.63	0
4	1.18	12.04

The current centroids will now be updated with new X and Y values for each cluster. The two (2) new centroids will have features equal to the mean of the corresponding features of the spots being in their cluster. Thus, the new centroid values will be the following:

$$c1 = (x_{c1}, y_{c1}) = \left(\frac{x_0 + x_1 + x_4}{3}, \frac{y_0 + y_1 + y_4}{3}\right) = (0.5, 0.5)$$

$$c2 = (x_{c2}, y_{c2}) = \left(\frac{x_2 + x_3}{2}, \frac{y_2 + y_3}{2}\right) = (8.5, 8.5)$$

The new centroids with their current clusters can be seen in Figure 7b. After the creation of the new centroids, the process of measuring the distances of each spot from the centroids begins again, until either no changes are made in the clusters, or any other terminating condition is met.

One metric to describe if a clustering algorithm performed well, is the Within-Cluster Sum of Squares (WCSS). If this metric is low, the algorithm did its job. In the other hand, if this metric is high, the algorithm performed badly. In this example the algorithm performed well, but this does not mean that if we run the algorithm again, then we are going to have the same result.

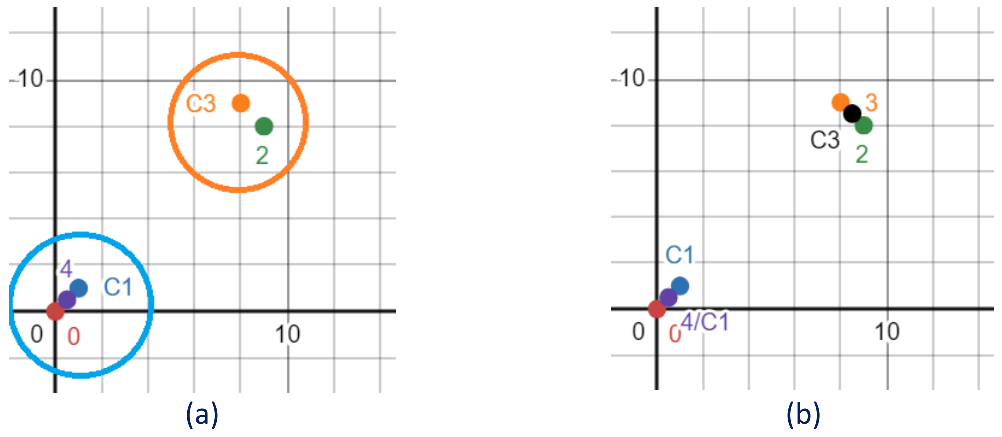


Figure 7 – K-means Clustering Process

3.6.3. Bisecting K-means Clustering

Bisecting K-Means is a hybrid clustering algorithm that combines elements of K-Means and hierarchical clustering [101]. Unlike traditional K-Means, which randomly initializes centroids and iteratively assigns data points to the nearest cluster, Bisecting K-Means adopts a top-down approach, starting with a single cluster and progressively dividing it into smaller, more refined clusters. The algorithm of this clustering method is also shown as pseudocode in Appendix II. To make this process understood by the reader, below is an example of this clustering method with figures.

For this example, the same dataset already shown in Table 2 will be used. Bisecting K-means, performs the clustering procedure in a hierarchical way. This means that the algorithm begins theorizing that all the dataset belongs in one cluster (Figure 8a). The algorithm finds two (2) clusters using K-means or any other needed method that is described from the user (Figure 8b). If the user gave the parameter K as more than two (2), then the algorithm does the same procedure to the cluster with the highest WCSS (Figure 8c). The algorithm will stop when the number K is achieved or if any other terminating condition is raised.

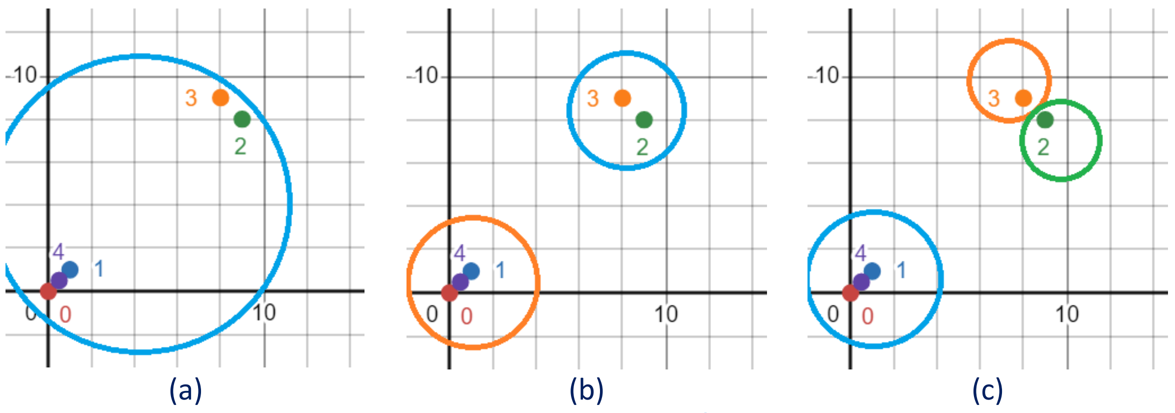


Figure 8 – Bisecting K-means Clustering

3.6.4. Gaussian Mixture Model Clustering

GMM Clustering is a probabilistic clustering algorithm that models the data as a mixture of several Gaussian distributions. Unlike K-Means, which assigns each data point to a single cluster based on the nearest centroid, GMM uses a probabilistic approach to assign data points to clusters with varying degrees of membership. Each cluster is represented by a Gaussian distribution, characterized by its mean and covariance. The algorithm of this clustering method is also shown as pseudocode in Appendix III.

The algorithm begins by initializing the parameters of the Gaussian distributions [102], such as their means, covariances, and mixture weights. It then iterates through two main steps: Expectation (E-step) and Maximization (M-step). In the E-step, the algorithm calculates the probability of each data point belonging to each Gaussian distribution. In the M-step, it updates the parameters of the Gaussian distributions based on these probabilities to maximize the likelihood of the observed data.

To illustrate this process, a single dimension dataset is being used in Figure 9. GMM starts by initializing Gaussian distributions (Figure 10). During the E-step, it calculates the probabilities of data points belonging to each Gaussian distribution. In the M-step, the parameters of the Gaussian distributions are updated based on these probabilities to fit the data better. This iterative process continues until convergence, where the changes in the likelihood of the data become minimal. In a dataset with more dimensions, the same process is performed, using all the different features of each data point.

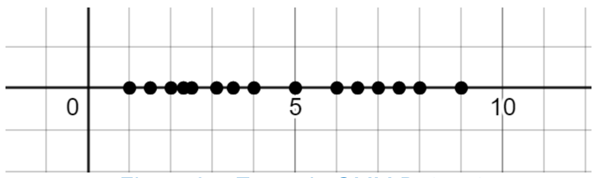


Figure 9 – Example GMM Dataset

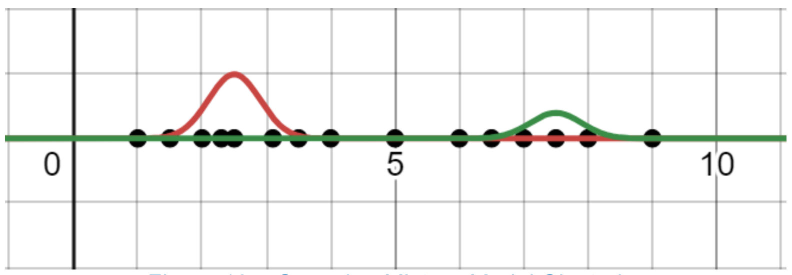


Figure 10 – Gaussian Mixture Model Clustering

4. Platform Implementation

This section presents an in-depth analysis of the platform's implementation, reflecting its objectives and identifying the various end-user groups with a vested interest in EverCluster. Continuing the objectives and the concerned user-groups, the architecture of the platform is analyzed in detail. The application of the tools and methodologies outlined in Section 3 is illustrated through figures and elaborated upon in the subsequent subsections. The analysis begins with an overview of the EverCluster architecture at a high level, followed by detailed representations of the lower-level architectural components. After the architecture is explained in detail, deployment diagrams are shown to represent the different environments that the platform can be function on. Continuing the deployment diagrams of the platform, the code of the platform is explained and analyzed in detail using different Sequence and Class Diagrams. The section comes to close by providing an execution manual as well as a user manual to achieve reproducibility from the readers.

4.1. Objectives & End-Users

In order for EverCluster to have added value to its end-users and give a solution to the problem statement of this document as well, there is a need for the following six (6) objectives to be achieved:

1. *Dataset Management*: This objective concerns the implementation of a way to handle user defined datasets in the platform. The end-user should be able to upload and handle files of data from their local environment to EverCluster. Also, the platform must be able to gather metadata for these datasets to give an easy way to the end-user to understand the datasets that they have already uploaded. Finally, the end-user should be able to download again their dataset from the platform in case that their data got lost from their local environment.
2. *Dataset Clustering*: This objective regards the functionality of EverCluster to perform clustering algorithms with parameters defined by the end-user. A set of ready to be used algorithms should be given to the end-user. These algorithms must be able to have as input the datasets that the end-user have already uploaded.
3. *Results Management*: The results of the clustering procedures should be available to the end-users. The predicted clustering results should be able to be downloaded from the platform's User Interface. Also, the results should be able to be removed from the platform in case the end-user wants to recreate them.
4. *Authentication System*: In a platform such as EverCluster, it is essential that numerous end-users consistently have access to the system. Consequently, the platform must support account management through a robust authentication system.
5. Scalable Architecture: EverCluster must be capable of scaling in response to end-user demands and account requirements. crucial is also considered the platform's ability to dynamically scale horizontally by integrating additional resources through container orchestration systems such as Docker Swarm, K8s, or similar engines. This parallel utilization of multiple resources ensures enhanced efficiency by accelerating the execution of procedures.
6. Algorithm Recommendation: In the rest of the objectives, the necessity for a platform capable of managing datasets and results, executing clustering operations, and offering scalable solutions is outlined. This objective entails developing a system that can provide recommendations reflecting the most suitable algorithm from those

available to the end-user. This recommendation should consider the metadata from the dataset that the end-user intends to handle as well as the already existing resources that the platform can utilize.

A platform designed with the outlined objectives primarily caters to groups of end-users in the ML domain. On one hand, academia, including students, can utilize the platform to experiment with clustering algorithms for educational purposes or to complete various projects, enhancing their understanding of ML techniques. Moreover, this group also includes researchers who can conduct experiments on a platform with the architecture described in subsequent sections. On the other hand, professionals such as data scientists and data engineers can leverage the platform to quickly generate results or replicate the architecture to incorporate self-adaptability into their own systems.

4.2. EverCluster High-Level Architecture

Figure 11 illustrates the High-Level Architecture of EverCluster. EverCluster consists of four main components: (i) Server, (ii) User Interface, (iii) Data Management and (iv) Spark Cluster. Each of the components are an important part of the platform and they are orchestrated in order to provide a seamless platform to the end-user than can provide all the needed functionalities that were described in previous sections. The responsibility of each component is the following:

- *Server*: This component is responsible for being the centre of the functionality of EverCluster. It comprises a Python Flask REST API server, which is further divided into several subcomponents. This component can provide the User Interface component with all the needed information that must be presented to the end-users as well as handle the information that is stored in using the Mongo DBMS and the MinIO Object Storage. Except from the above, this component is the one that handles the recommendation processes of EverCluster, using the scikit-learn Python library. Finally, this component is the one that contacts the Spark Cluster for execution, using the Spark REST API. This component is also divided into the following subcomponents:
    - *Cluster Information Scraper*: This subcomponent is used to gather information by the Spark Master's Dashboard.
    - *Authenticator*: The Authenticator is used to handle the credentials of the end-users of the platform.
    - *Dataset Handler*: This subcomponent is used in order to access the end-user's datasets as well as to co-operate with the Validity Controller in order to manage the uploaded files in a seamless manner.
    - *Validity Controller*: The Validity Controller is used in order to verify validity of end-user uploaded files.
    - *Spark Handler*: This subcomponent is the one responsible to contact the Spark Cluster component explained below. It is the one that can submit analytic works for execution.
    - *Recommendator*: This is the component responsible for making recommendations to the end-users based on the clustering algorithms that they should probably use in order to achieve the best performance regarding on accuracy or speed.
- *User Interface*: The User Interface is a series of jinja2 Templates used by the Server component. The User Interface has direct communication with the Server component

only, and it is responsible to provide responsive and understandable interfaces to the end-users. In this component, the use of Bootstrap and jQuery provides a visibly appealing interface.

- *Data Management*: In this component the NoSQL database of EverCluster handled by the Mongo DBMS exists. Also, the buckets for the upload of end-user datasets and the storing and downloading of the clustering results is handle in this component by MinIO Object Storage. The two tools that are used, are both cloud-centric in nature, making them easily scalable in containerised environments.
- *Spark Cluster*: The Spark Cluster component of EverCluster resides at the bottom of its architecture and is the most important scaling aspect of the platform. In this component the Spark Master and many Spark Workers reside. All the different Spark nodes are coming with PySpark and Pandas preinstalled in order to handle Spark ML procedures in a distributed manner.

The figure of the Paltform's high-level architecture illustrates clearly the utilization of Docker containers across all components. The containerization of the entire system enables the seamless deployment of EverCluster in any environment that supports Docker container initialization.

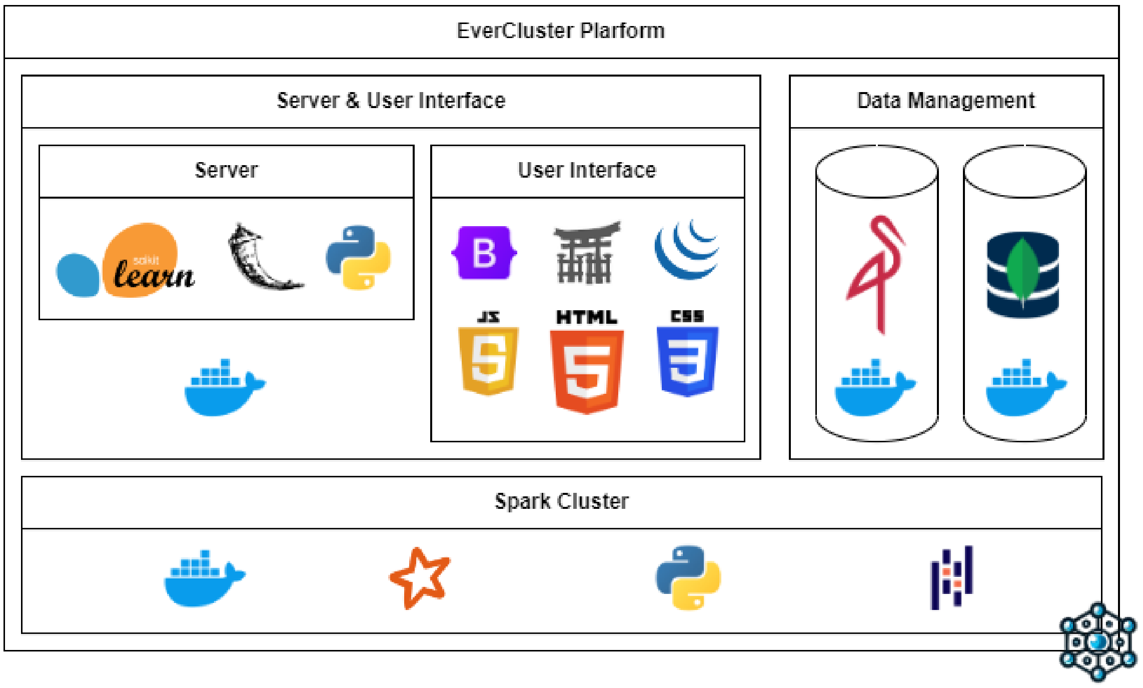


Figure 11 – High-Level Architecture

4.3. EverCluster Low-Level Architecture

The current subsection is divided into two subsections, each one responsible for analyzing the lower levels of EverCluster's architecture explained in the high-level figure that was presented. The lower-level architecture of EverCluster showcased though the inner communications performed on the Server, User Interface & Data Management components as well as the inner communications between the Spark Cluster and the Data Management components. The Spark Cluster and the Server components are seen communicating between the two following subsections. The Data Management component does not have a low-level communication figure since it consists only by the Mongo DBMS and the MinIO Storage containers.

4.3.1. Server & User Interface

In this subsection, low-level architecture figures are used to present the communication of the different components and their inner functionalities.

Starting with Figure 12, three of the platform’s components are represented. This figure illustrates how the platform continuously tracks the number of nodes being utilized by the Spark Cluster. The Server is responsible for scraping information from the Spark Master’s dashboard. By doing so, it can detect though the HTML elements of the dashboard the number of online nodes and if the master node is also alive. Following the information scraping, it collects this information, the Server provides it as an input to the interfaces of the User Interface component.

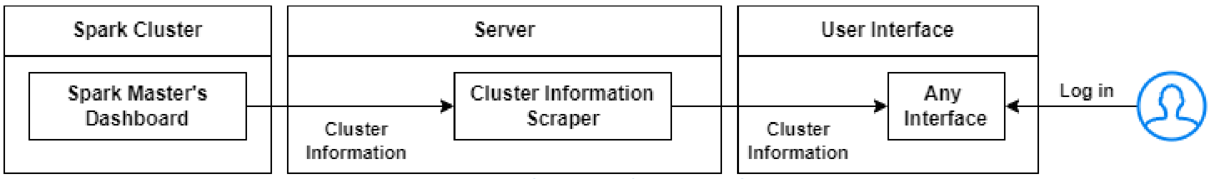


Figure 12 – Cluster Information Scraper

In Figure 13, the way that the authentication of the platform is handled is visualized. Also, the functionality of the platform’s dataset handling is presented. The process begins with the authentication mechanism, where the end-user interacts with the User Interface component to transmit information to the server. The server is responsible to exploit the Data Management component and validate the information given by the end-user. Once validation is completed, the server responds with either a confirmation or a rejection of the end-user's credentials. In the same figure the Server can be seen using the Dataset Handler and Validity Controller subcomponents. These subcomponents are needed to handle the datasets that the end-user is uploading on the platform. When an end-user is uploading a dataset from a Dataset Interface, the label of the dataset, the dataset itself and different metadata of it are transmitted to the Server component. The Server uses the Dataset Handler in combination with the Validity Controller to check for mistakes in the given dataset and raise any meaningful warning to the end-user about their file. If the file is valid and the provided label is unique to the end-user's datasets, the server notifies the object storage and the EverCluster database about the new dataset in the end-user's collection and storage bucket.

One more functionality that can be seen, is that when an end-user is uploading a dataset successfully, a download link is created by MinIO and passed through the flow to the EverCluster database. In this way when the Dataset Handler passes all the datasets of the end-user to the Dataset Interfaces, the download link is provided as well. This functionality gives to the end-user the ability to download dataset right from the platform, by contacting the MinIO object storage directly using their browser.

Lasty, the end-user can delete an uploaded dataset from the platform. With all the provided datasets passed to the User Interface component, the end-user can select a dataset by its label, passing it back to the Dataset Handler subcomponent and triggering the deletion of the dataset and its metadata from the platform’s Data Management component.

In case of an error occurrence, like mismatched passwords during registration, the server manages the issue by sending helpful information to the User Interface component. The User

Interface component then uses this information to display a user-friendly warning message to the end-user.

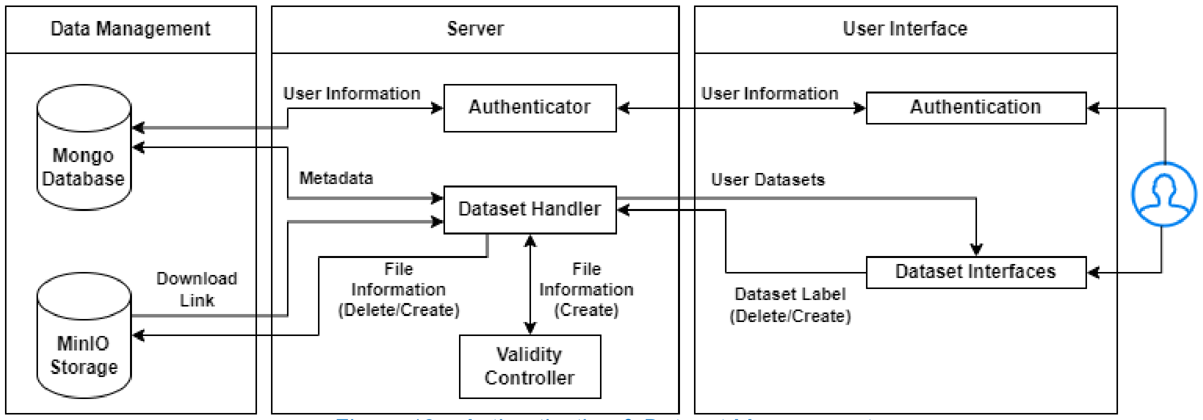


Figure 13 – Authentication & Dataset Management

Following the low-level architecture analysis of the dataset handling, the execution flow for the clustering algorithms by the end-user is shown in Figure 14. In this figure the end-user is seen interacting with the Analytics Interface, passing their dataset information and their preferred execution information to the Spark Handler on the Server's side. The Spark Handler is using the number of current working Spark Nodes by the scraper that has been explained above, the clustering that the end-user wants to perform and the dataset they wish to use to perform a clustering procedure. More specifically the Spark Handler communicates with the Recommendator subcomponent with the following parameters as input: (i) the max clustering iteration number from the end-user, (ii) the amount of different clusters that the end-user wants to find in the dataset, (iii) the number of online Spark Workers, (iv) the byte size of the dataset that the end-user wants to analyze, (v) the amount of records existing in the dataset that the end-user wants to analyze and (vi) the amount of different features that are going to be used on the clustering. The parameters are provided to two classifiers: one selects the best algorithm based on analysis speed, while the other focuses on accuracy. The creation of these classifiers is detailed in the Experimentation Methodology section, as they can be replicated and improved for different environments using various classification methods. After the Spark Handler gathers the two proposals, the subcomponents return them to the Analytics Interface, allowing the end-user to choose one of the suggested options or their original preference.

When the final execution choice has been selected the Spark Handler is responsible to submit the clustering job to the Spark Master using the Spark API. This API returns the job's status, and the Server component performs HTTP polling to be notified when the job is completed. The clustering execution and distribution is handled completely by the Spark Master. Appendixes X, XI and XII are showing the PySpark code that was developed to perform the three clustering procedures. The Spark Master is also responsible for communicating with the Data Management in order to get the end-user's dataset for analysis, leave metadata information in the EverCluster database as well as, leave the results in the object storage. In the next subsection the communication with the Data Management component is shown in a more detailed manner.

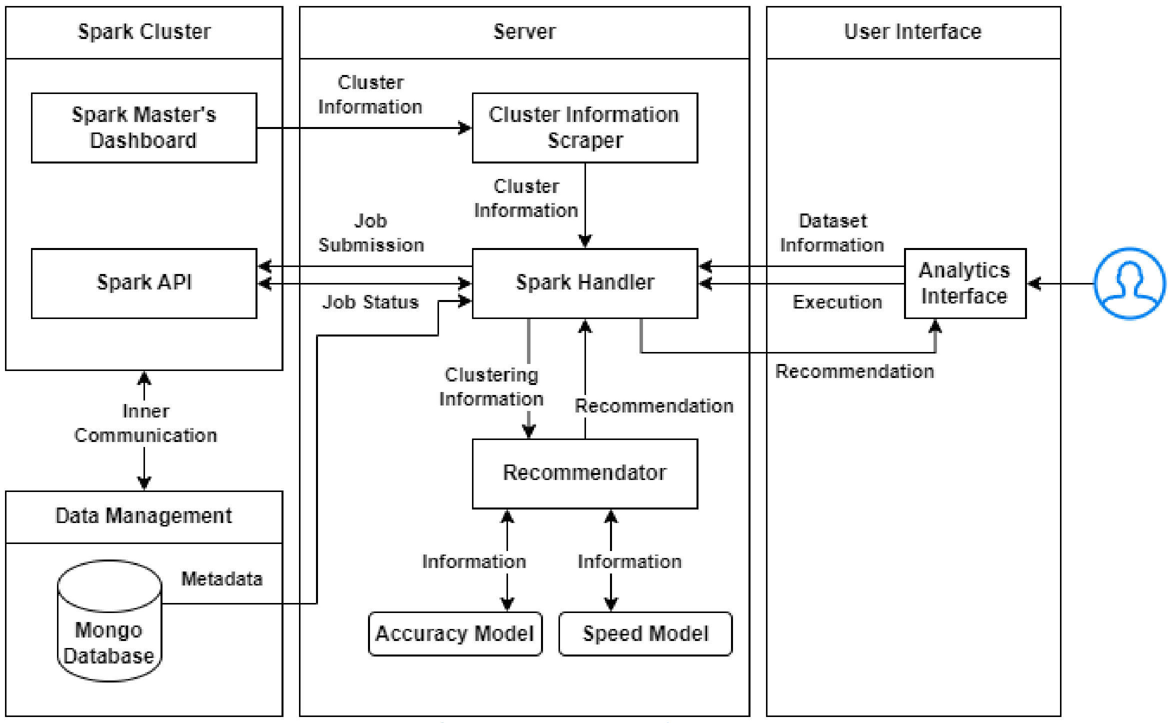


Figure 14 – Clustering Execution & Recommendation

The last low-level architecture figure for the Server and User Interface components is shown in Figure 15. In this figure, the way that the results are handled is shown. The end-user is contacting the User Interface component by the Results Interfaces. The Server can be seen already providing all the results of the end-user. The end-user can perform downloading of the results by a download link that has been saved previously by the Spark Master in the EverCluster database and is shown with the results on the interface. Also, the end-user can delete results by using their label like the dataset deletion functionality previously explained.

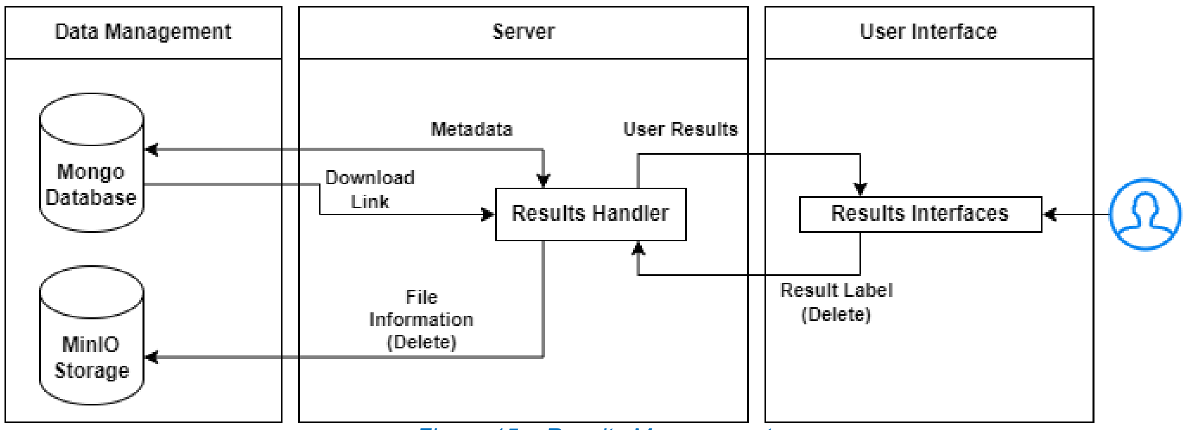


Figure 15 – Results Management

4.3.2. Spark Cluster Communication

In this subsection the low-level architecture of the Spark Cluster Communication with the Data Management components is analyzed. This architecture can be seen in Figure 16. The figure continues the low-level architecture described in Figure 14, having as inputs the job submission and Job Status of the end-user's execution. As it is shown, the Spark API is responsible of creating a Spark Driver in the Spark Master node. This Spark Driver itself is then creating one Spark Executor on each Spark Worker node of the Spark cluster, distributing the execution of the analytics workflow. By contacting the Spark Executors, the Driver is also passing metadata information by the EverCluster database and the dataset's information by the object storage. Once the clustering procedure is complete, the Spark Driver recontacts the EverCluster database to store the results metadata and also reaches out to the object storage to save the actual results.

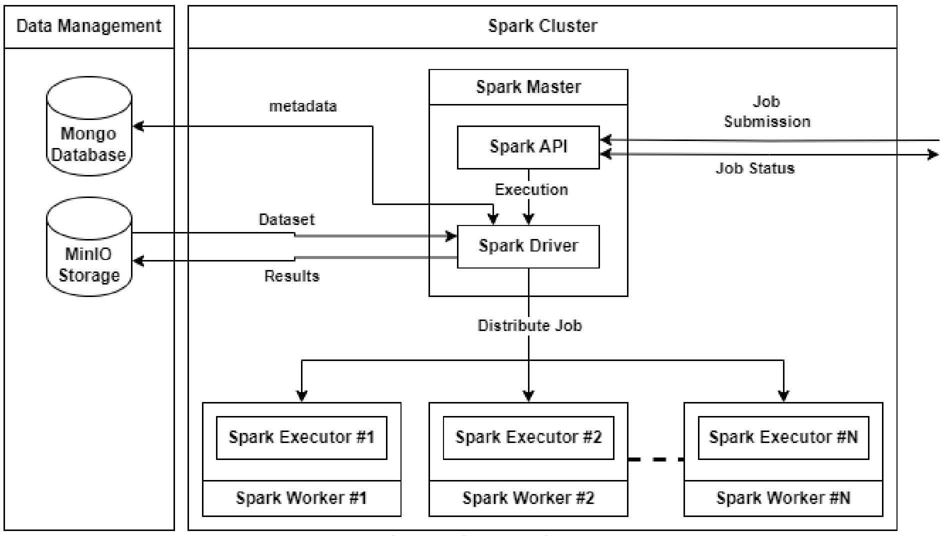


Figure 16 – Spark Cluster Communication

4.4. Deployment Diagrams

Continuing the architecture high- and low-level figures of EverCluster, this section is providing to the reader possible deployment methods for the platform. As shown in the high-level, all the components of the platform are containerized. This means that the platform can be deployed in environments that support containerized applications. This feature makes EverCluster scalable in environments using Docker Swarm, K8s and other related engines. The development and testing of this project was performed using the first deployment shown below (*Single Node Deployment*) by exploiting the capabilities of a docker-compose [103] file. Using this file simplifies the installation process and ensures it can be easily replicated across any environment utilizing the aforementioned engines. In the case of the deployment of the platform in a K8s environment, the use of the kompose [104] tool is heavily advised on the docker-compose file. In Appendixes XIII – XIX, all the versions of the tools used to develop and deploy EverCluster are named.

4.4.1. Single Node Deployment

The fastest and easiest way to deploy EverCluster, is using its single node deployment. In Figure 17, a deployment diagram of the platform being deployed on a single node environment is showcased. The environment depicted represents a laptop having the Docker Engine installed. By using Docker, six (6) containers are shown to be running. The Data Management component is seen to be using two containers to separate the Mongo DBMS and the MinIO object storage. Also, in one container the User Interface component and the Server component can be seen interacting directly with each other as shown in the high-level view of the platform's architecture. These two components are the only ones that can't be separated from each other due to their way that they have been developed. Also, at the base of the deployment, the Spark Cluster component can be seen with each Spark node in a different container.

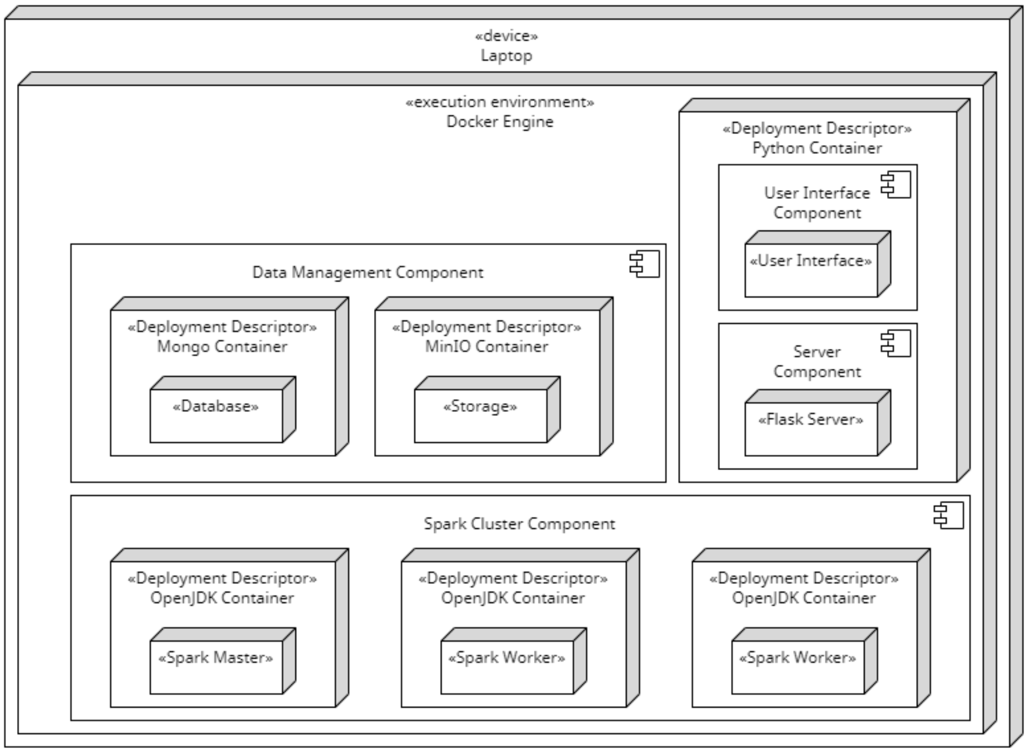


Figure 17 – Single Node Deployment

4.4.2. Two-Node Deployment

This subsection outlines a deployment diagram featuring two nodes. This approach allows for a step-by-step analysis of the platform's scalability by breaking it down into additional devices. In Figure 18, EverCluster is visualized having its workflow shared into two personal computers. This deployment is achieved in the figure conceptually by using Docker Swarm. The first computer is shown with three deployed components: the Data Management, the User Interface and the Server. The second computer is shown with the Spark Cluster deployed on it, being the analytics dedicated resource of the cluster.

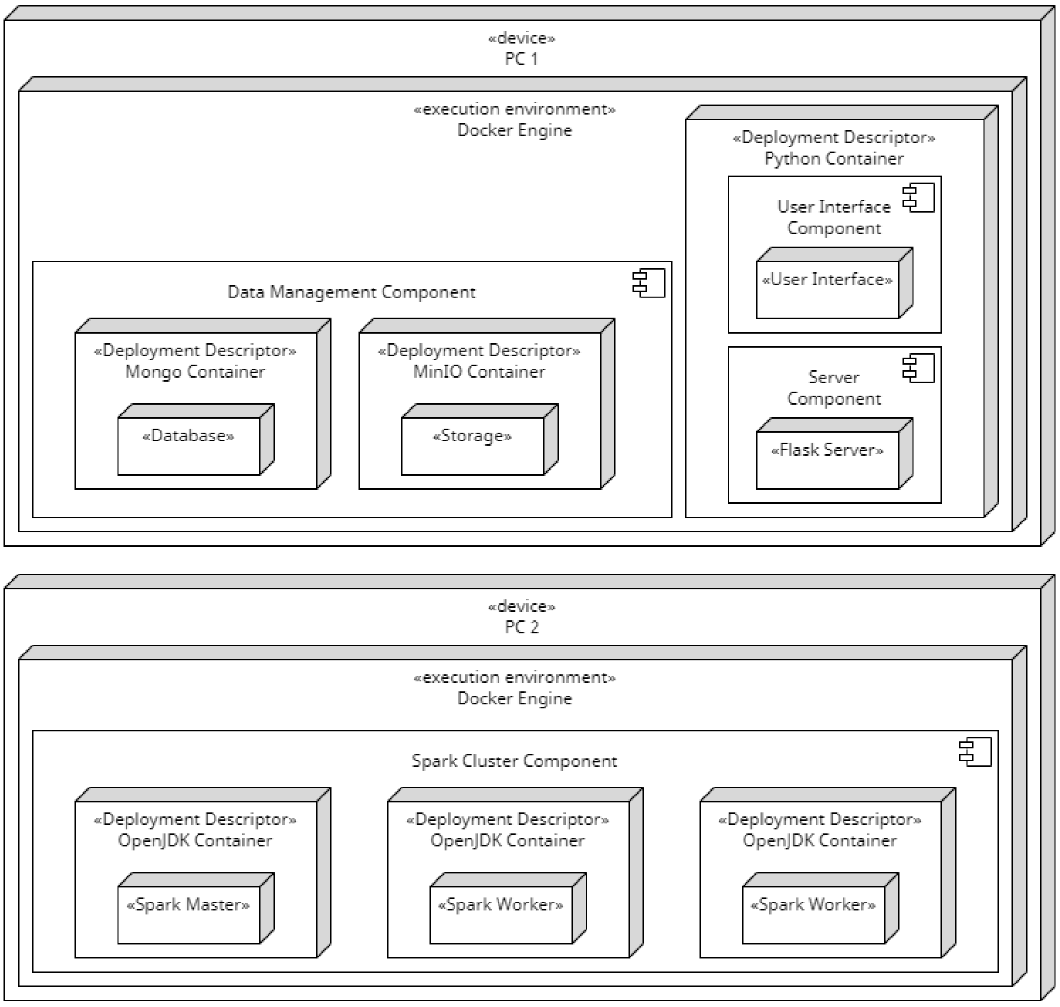


Figure 18 – Two-Node Deployment

4.4.3. Multi-Node Deployment

Continuing the rest of the deployment diagrams. This subsection showcases a scenario better explaining EverCluster's distribution ability. In Figure 19, a multi-node deployment is shown. It can be seen that the same number of containers with the rest of the above deployment diagrams are being used, but this time five (5) different personal computers are being exploited. The components Server and User Interface are both deployed in the first computer since they cannot be separated. The Data Management component is visualized in the second computer. The Spark Cluster component is represented scaling itself across three different computers, with each one executing one Spark Worker. It is important to mention that the Data Management component itself can be partitioned into two computers. Having the MinIO object storage in one computer and the Mongo DBMS in the other.

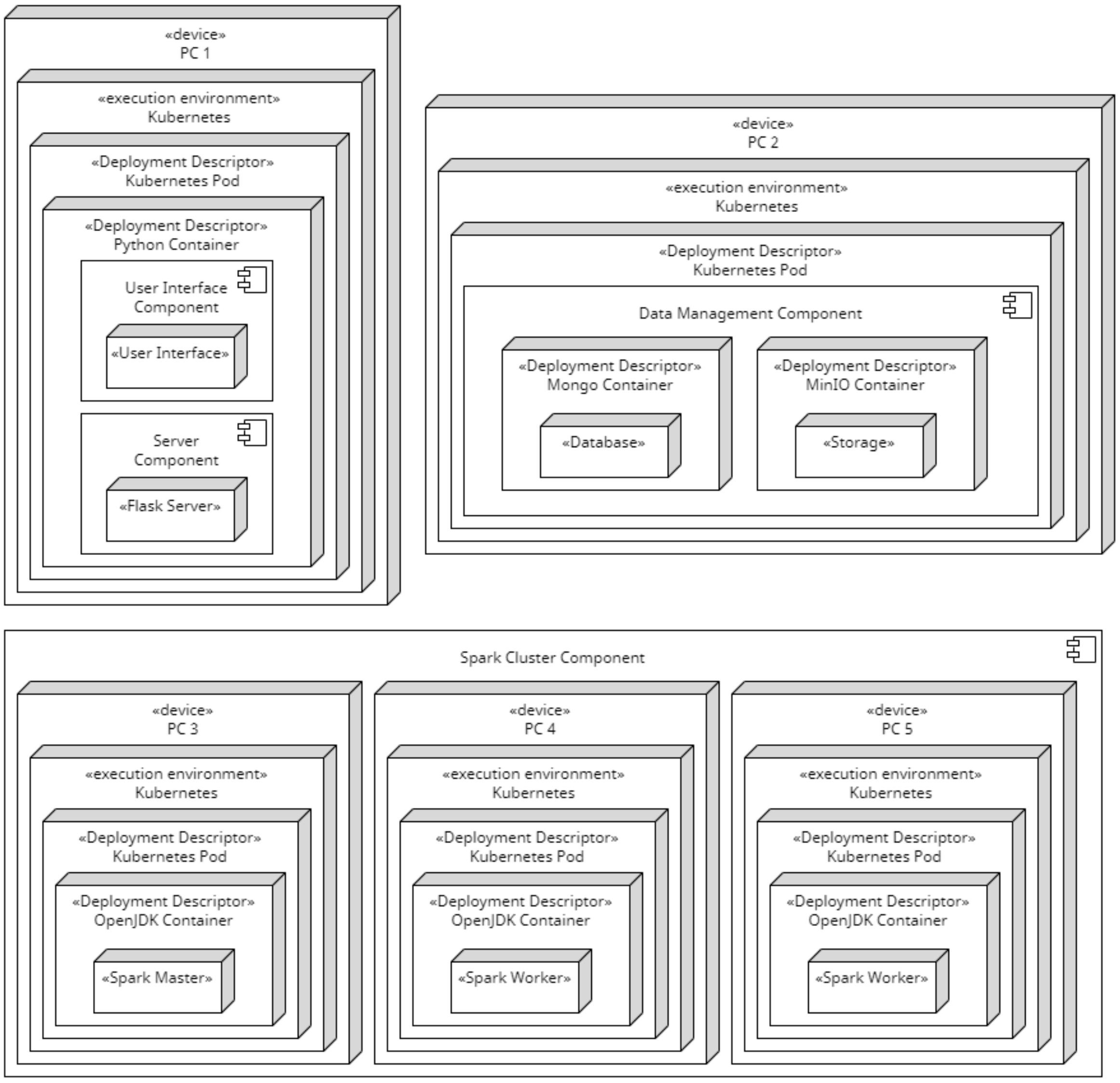


Figure 19 – Multi-Node Deployment

4.5. Activity Diagrams

To assist the architecture described in the previous subsections, different activity diagrams are provided below, showcasing a high-level view of the decision making of EverCluster. For each of the activity diagrams provided, the processes of the end-user and the platform's automated actions are described. The following activity diagrams are also representing each of the main User Interface functionalities of EverCluster.

4.5.1. Authentication

The first activity diagram in Figure 20 illustrates the information flow related to the end-user authentication process. It begins with user login and registration. During registration process, the end-user provides new credentials, which are stored in the database. For the login, the end-user enters their credentials, and the system verifies their validity, as shown in the corresponding low-level architecture diagram. If the login attempt is successful, the end-user is redirected to the EverCluster home page. Throughout the process, as well as in other activity diagrams, the system generates warnings for the end-user in cases of invalid data or unsuccessful action attempts.

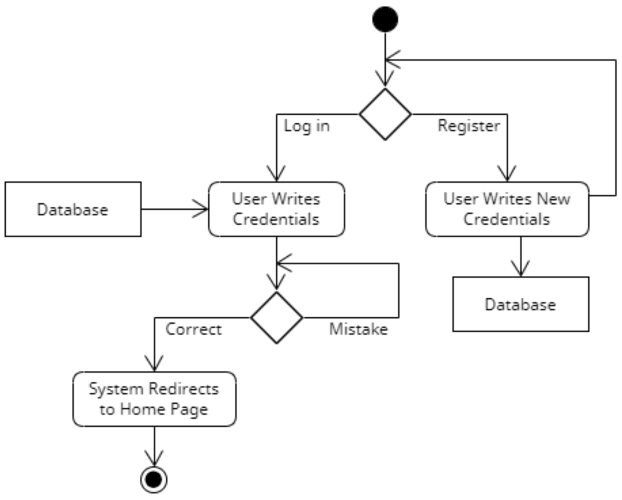
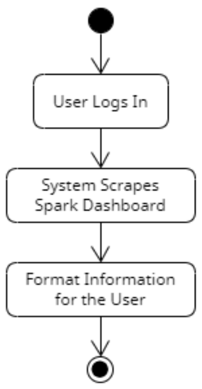


Figure 20 – Authentication Activity Diagram

4.5.2. Cluster Information Gathering

The activity diagram presented in Figure 21, is visualizing the procedure of the platform's system for gathering the Spark Cluster Information. When the end-user is logged in, the system performs the scraping of the needed information by the Spark Dashboard and formats it for the end-user's interfaces.



User Logs In
System Scrapes Spark Dashboard
Format Information for the User

Figure 21 – Cluster Information Gathering Activity Diagram

4.5.3. Dataset Handling

Figure 22 visualizes all the different procedures that an end-user can perform within the Dataset Interface. First, the end-user can see their uploaded datasets. By doing so, the system is gathering all the needed metadata information by the EverCluster database and then presents it to them. Another ability that is given to the end-user is to delete an already uploaded dataset by the dataset's label. By doing so, the system is removing all the existing information labeled below the given dataset and then this process is terminated. The last ability that is given to the end-user is to upload a dataset, in this procedure, the end-user is providing a label and a file as their dataset. The system is responsible for validating the uniqueness of the given label across the rest of the uploaded datasets of the end-user, as well as to check for possible errors in the file. If there are no errors, the file and its metadata are saved in the database and the object storage. Before the dataset uploading is complete, the metadata of the dataset are updated with a download link for the uploaded file. It is important to note that by viewing a dataset, an end-user can also download it, without performing any task in the system. This is done by the download link provided as metadata from the system to the User Interface component.

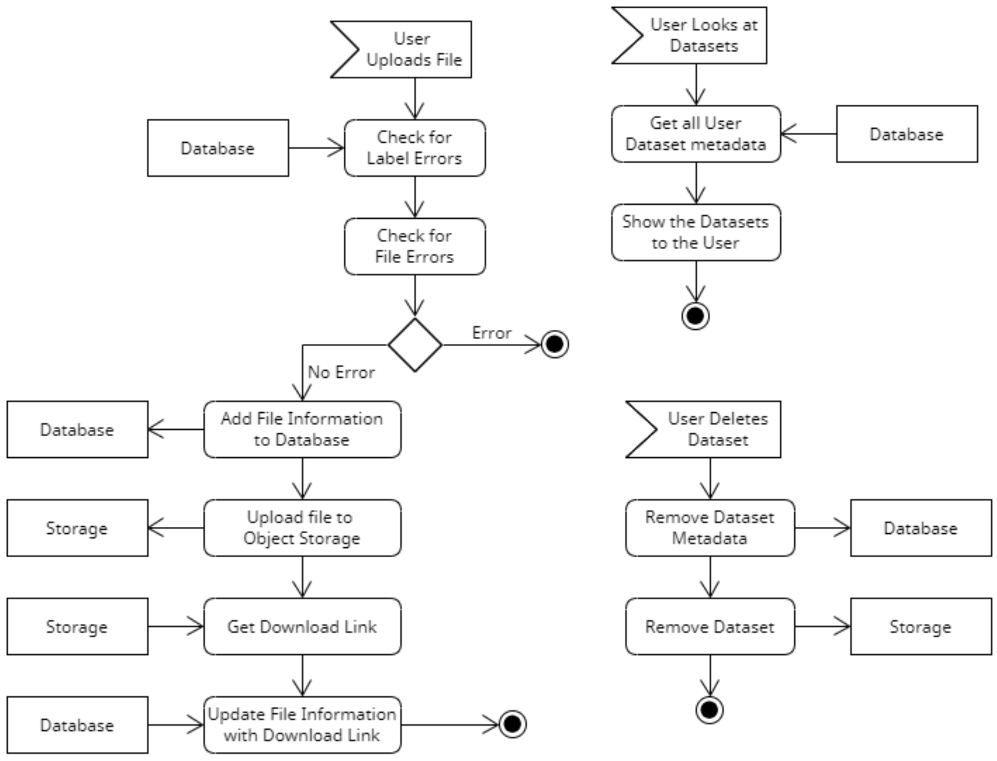


Figure 22 – Dataset Handling Activity Diagram

4.5.4. Clustering Execution

The most important aspect of the platform is presented in Figure 23. In the mentioned figure, the end-user can request a clustering on one of the uploaded datasets. Next, the system gathers all the necessary information to recommend potentially better algorithms, as detailed in the corresponding low-level architecture figure. Then the system is requesting for the most suitable algorithm for the current clustering procedure in terms of speed and accuracy. Following to that the end-user makes a choice between three algorithms that are presented on their interface. By making a choice the system submitted the end-user's job to the Spark cluster continuously checking if the job is finished. When the job is completed, the end-user can proceed into viewing their results. Also, at the finishing execution steps of the job, the results and their useful metadata are saved in the EverCluster's saving systems.

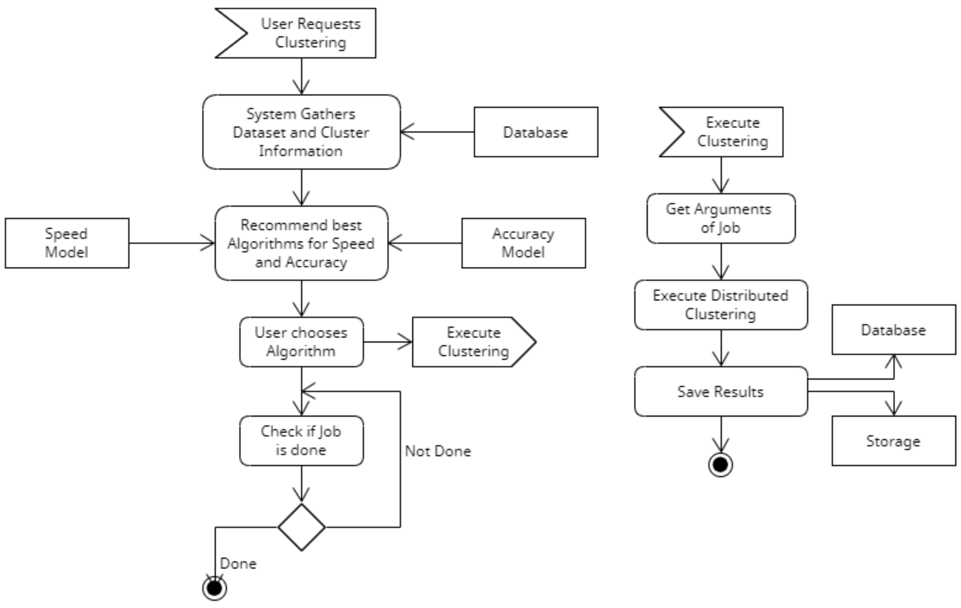


Figure 23 – Clustering Execution Activity Diagram

4.5.5. Results Handling

At any point, by interacting the Results Interfaces of EverCluster, the end-user can view the clustering results as shown in Figure 24. For the end-user to view their results of any analytic procedure, the system is getting all the saved results metadata from the EverCluster database. Then the metadata are formatted and presented on the User Interface component. If the end-user has results available for viewing, they can delete them. In such a scenario, the end-user requests the deletion of results based on their label, and the system removes them in the same manner as it does for an end-user's dataset. It's important to report that by viewing a dataset, the end-user can also download it without needing to take any further action in the system. This is achieved by a download link provided as metadata from the system to the User Interface component.

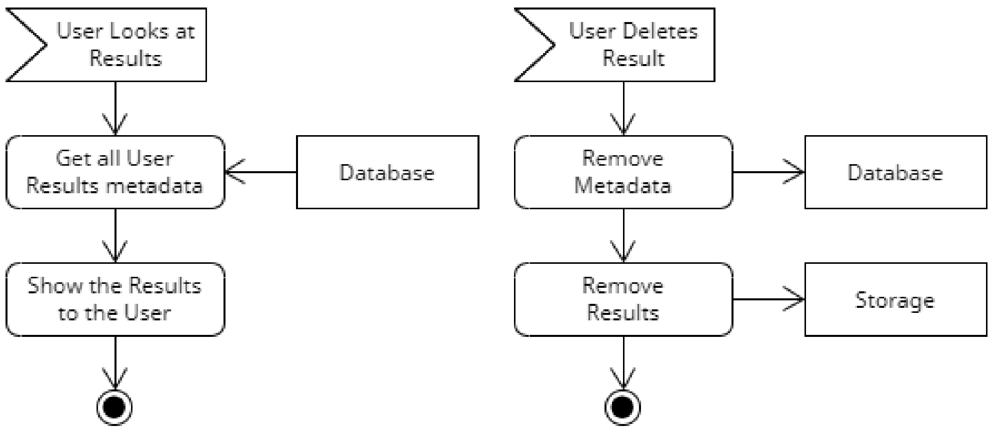


Figure 24 – Results Handling Activity Diagram

4.6. EverCluster Manuals

The above subsections have described explicitly all the functionalities, the deployment methods and how the platform's system interacts between its components to offer a seamless experience for the end-user. This platform is capable to handle uploading and removal of datasets, execute clustering procedures, recommend algorithms as well as remove unwanted results or download them by the User Interface. This subsection is focused on how the end-user can use this platform. To explain the way that an end-user and any concerned party can use EverCluster two guides are given in the following subsections. Firstly, an Installation Guide is described and secondly a User Manual is given visualizing the platform’s interfaces and explaining in a simple manner their usage.

4.6.1. Installation Guide

In this subsection the way to install and execute EverCluster is described. In order to do this, the concerned party who is going to deploy EverCluster must have one device the following tools installed: (i) Docker, (ii) docker-compose, (iii) Git [105]. In the following example a single-node deployment is performed. By making changes and using Docker Swarm or K8s and kompose the rest of the deployment methods can be achieved as well.

Download the Code Repository

In a directory that is preferred open a terminal and type the following command to download the code repository of EverCluster:

```
git clone https://github.com/karamolegkos/EverCluster.git
cd EverCluster
```

Download Images & Execute EverCluster

After verifying that Docker Deamon is running in the system, the following command must be executed in the same terminal:

```
docker-compose up -d
```

After the above command, all the needed images will start to download into the system’s storage and then Docker containers will be executed. After some time, the following containers will be running:

- 1 MinIO Object Storage Container
- 1 Mongo DBMS Container
- 1 Flask Server Container
- 1 Spark Master Container
- 5 Spark Worker Containers

In Appendix XX, the docker-compose.yml file is given. If needed a concerned party can adjust this file to install more, or less, Spark Worker containers.

Download Images & Execute EverCluster

To access the platform, a preferred browser can be opened to access the following URL:

```
localhost:5000
```

Download Images & Execute EverCluster

In the case of needing to remove the platform from the system the following command must be execute from the same directory:

```
docker-compose down
```

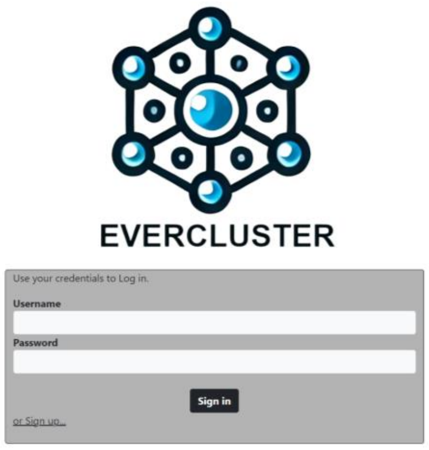
4.6.2. User Manual

Following the execution guide, this subsection provides the user manual of the platform. Below, different screenshots can be found after the above execution is performed. Each screenshot showcases a different interface and explains functionalities that are visible to the end-user.

Log In Interface

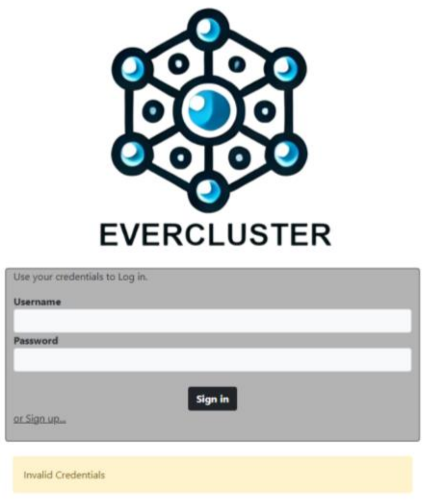
Starting with the *User Manual*, the log in Interface is presented in Figure 25a. This interface is accessible by any browser in the URL: localhost:5000. In this interface, an already registered end-user can Sign in by providing their username and password. In the case of wrong credentials, it can be seen in Figure 25b that the interface is providing a warning to the end-user, informing them about their mistake.

By clicking on the "or Sign up…" link the end-user will be redirected to the Registration Interface.



Use your credentials to Log in.

(a)



Use your credentials to Log in.

Invalid Credentials

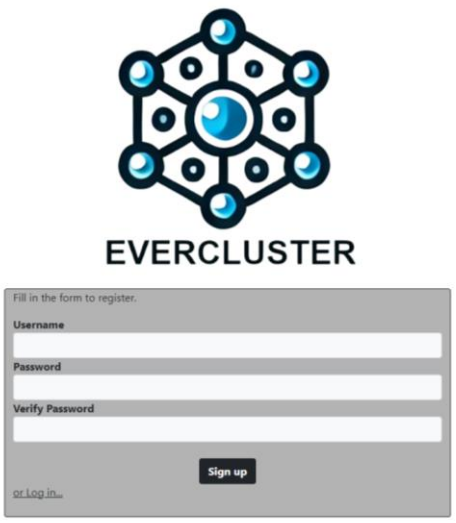
(b)

Figure 25 – Log in Interface

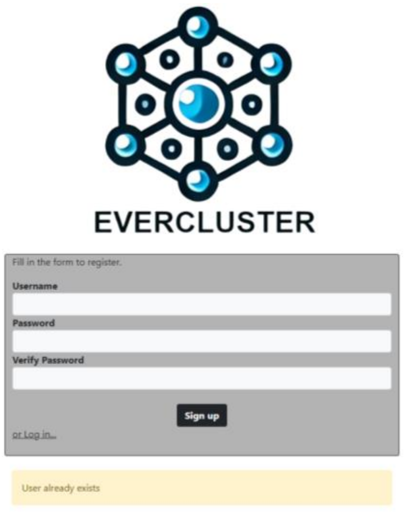
Registration Interface

When an end-user is in the Registration Interface (Figure 26a), they can use a unique username to create an account. In the case of a mismatch in the password and its verification, or in the case of the usage of an already existing username, a user-friendly warning will pop up, as shown in Figure 26b.

By clicking on the "or Sign in…" link the end-user will be redirected to the Log in Interface.



(a)



(b)

Figure 26 – Registration Interface

Home Page Interface

By logging in, or registering, the system will redirect the end-user to EverCluster's Home Page (Figure 27). From this page the Logo of the platform is visible, with information on the nodes of the Spark Cluster. In the screenshot that is provided, it can be seen that currently EverCluster is using one Spark Master and five (5) Spark Workers. Also, from this page and every page following one, the end-user has access to the "Upload Dataset Interface", the "Clustering Interface" and the "Results Interface". Also, the "Log out" button is now an option.

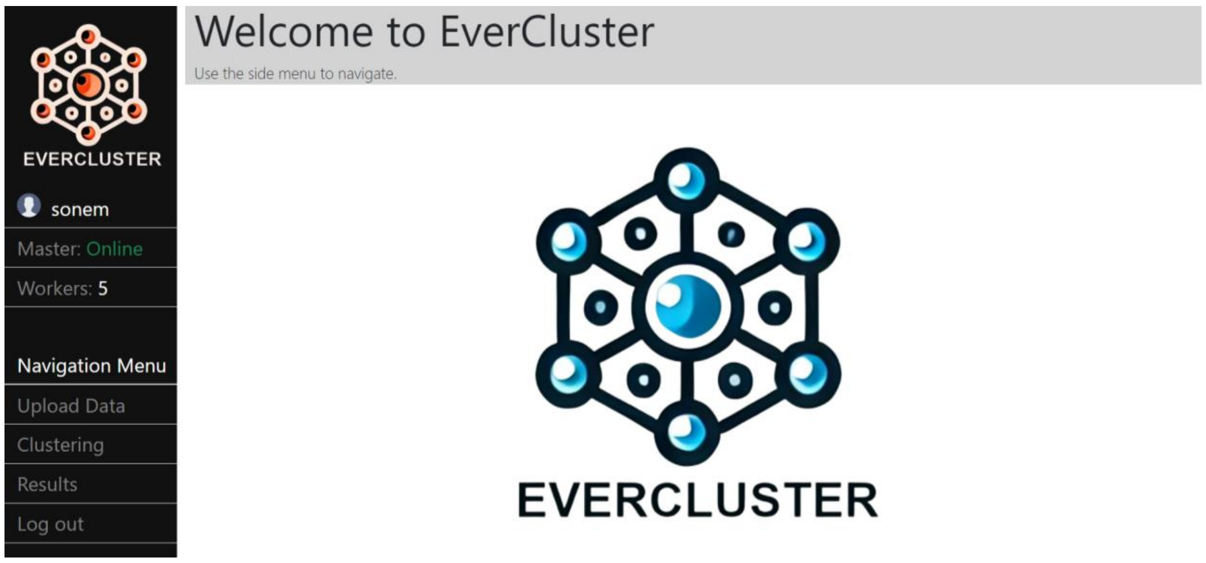
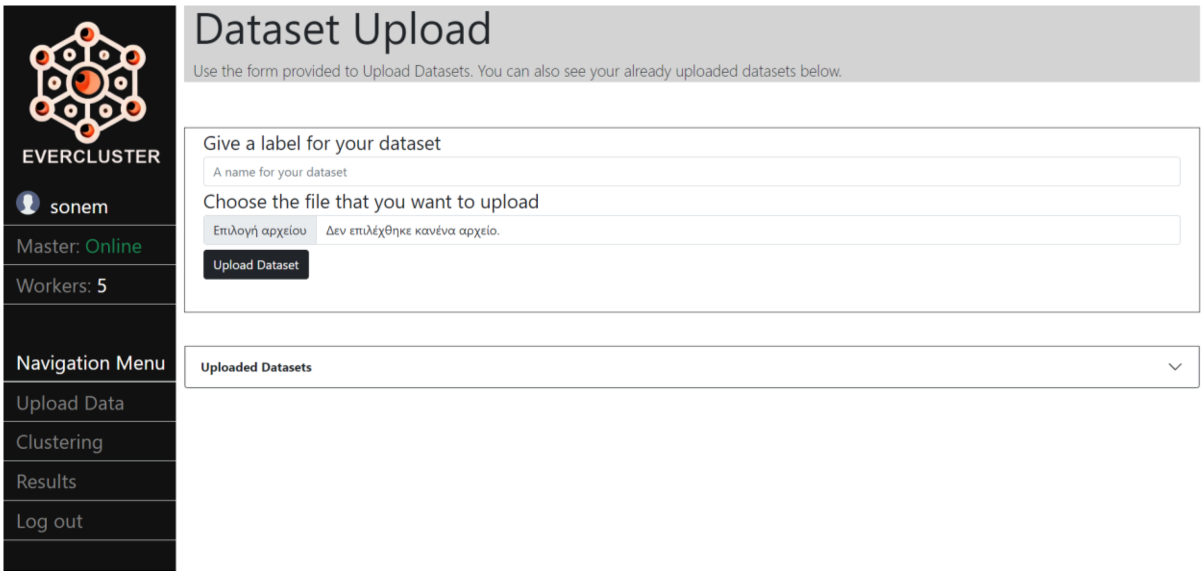


Figure 27 – EverCluster Home Page Interface

Upload Dataset Interface

By pressing the button "Upload Data" an end-user will be able to view the interface presented in Figure 28. In this figure, a label can be given to upload a dataset with its corresponding file from the local environment. When this is done, the end-user can view their datasets by clicking on the "Uploaded Datasets" button. Figure 29 shows two uploaded datasets together. For any uploaded dataset its metadata can be viewed. This metadata consists of: its label, number of records, upload date, byte size and headers. Also, the end-user can use the red button to delete the dataset from the platform, or the black button to download it again from the platform's object storage.



Dataset Upload

Use the form provided to Upload Datasets. You can also see your already uploaded datasets below.

Give a label for your dataset

A name for your dataset

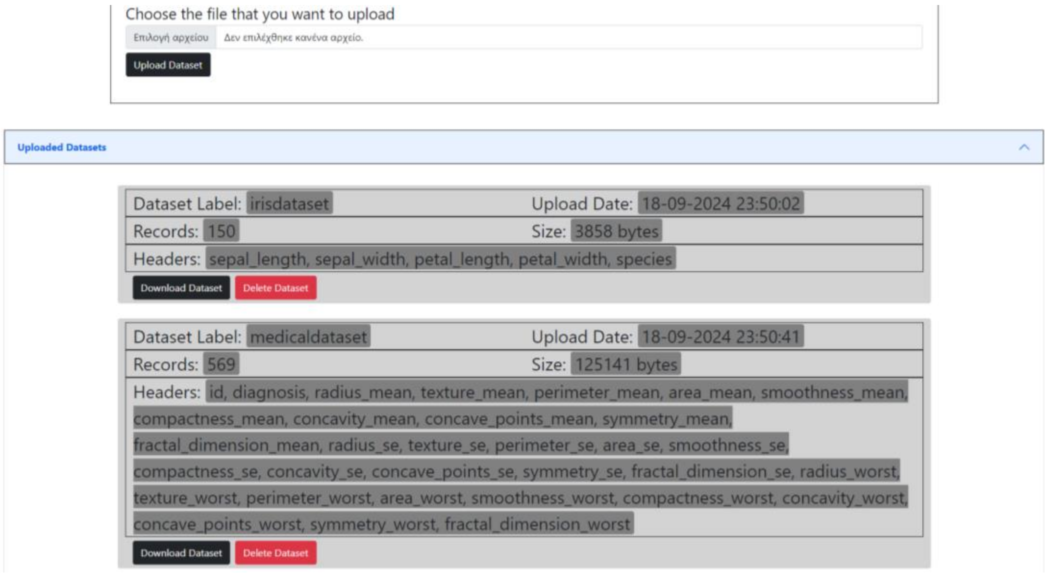
Choose the file that you want to upload

Επιλογή αρχείου Δεν επιλέχθηκε κανένα αρχείο.

Upload Dataset

Uploaded Datasets

Figure 28 – Upload Dataset Interface



Choose the file that you want to upload

Επιλογή αρχείου Δεν επιλέχθηκε κανένα αρχείο.

Upload Dataset

Uploaded Datasets

Dataset Label: irisdataset	Upload Date: 18-09-2024 23:50:02
Records: 150	Size: 3858 bytes
Headers: sepal_length, sepal_width, petal_length, petal_width, species	

Download Dataset Delete Dataset

Dataset Label: medicaldataset	Upload Date: 18-09-2024 23:50:41
Records: 569	Size: 125141 bytes
Headers: id, diagnosis, radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, concave_points_mean, symmetry_mean, fractal_dimension_mean, radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave_points_se, symmetry_se, fractal_dimension_se, radius_worst, texture_worst, perimeter_worst, area_worst, smoothness_worst, compactness_worst, concavity_worst, concave_points_worst, symmetry_worst, fractal_dimension_worst	

Download Dataset Delete Dataset

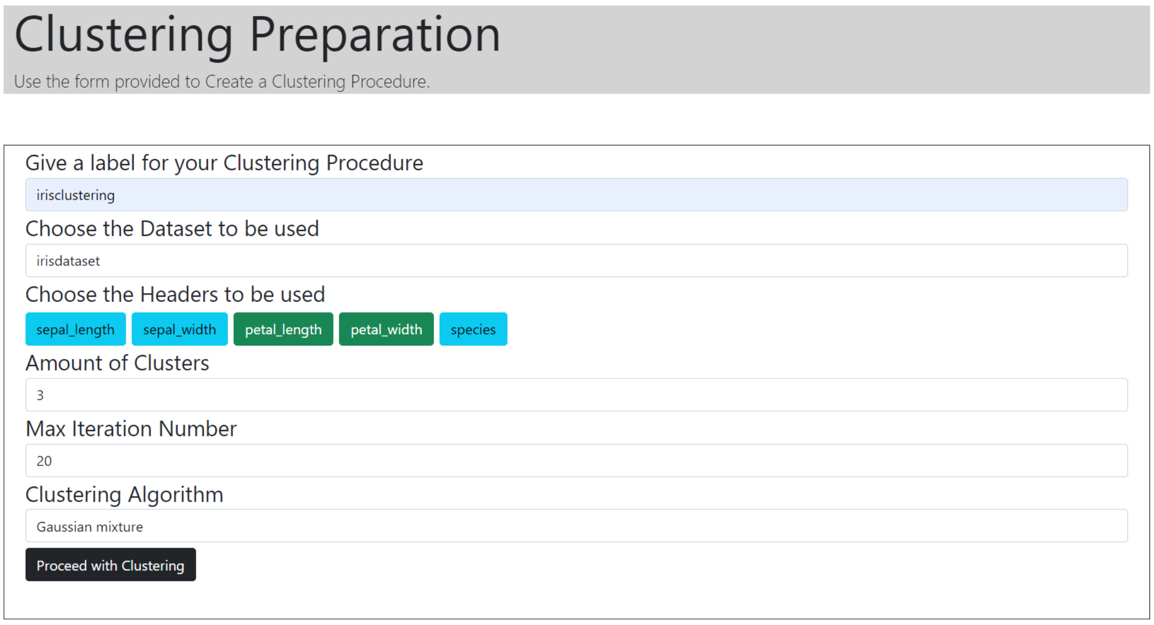
Figure 29 – Uploaded Dataset Example

Clustering Interface

By pressing the button "Clustering" an end-user will be able to view the form presented in Figure 30. In this interface, the end-user can give as input to the platform: the label for a clustering job, the dataset to be used (here the "irisdataset" is chose), the headers to be used for the clustering as features (the selected ones are green), the number of clusters to be found, the max iteration number for the algorithm and the clustering algorithm to be used. All the information is presented using listed items except of the clustering label. If the end-user changes the dataset for the clustering, the headers are updated with the ones of the new dataset. The system is handling in this interface as well any warning related scenario, such as the end-user trying to find more clusters than the records of the datasets, the usage of an already existing clustering label and more.

By pressing the "Proceed with Clustering" button, the end-user can view for a last time the clustering that is going to be performed (Figure 31). This time, two recommendations are also shown in the bottom of the form. The first recommendation regards the best algorithm in speed for this clustering procedure and the second recommendation regards the best algorithm for accuracy. For example, in the same figure, the end-user chose to execute the Gaussian Mixture algorithm, but the system recommends that for both cases the K-means algorithm is better.

By pressing any of the three buttons we select the algorithm that is written on it. Then the execution begins in the Spark Cluster component. From here we can view the progress of the job in the Results Interface.


Clustering Preparation

Use the form provided to Create a Clustering Procedure.

Give a label for your Clustering Procedure: irisclustering

Choose the Dataset to be used: irisdataset

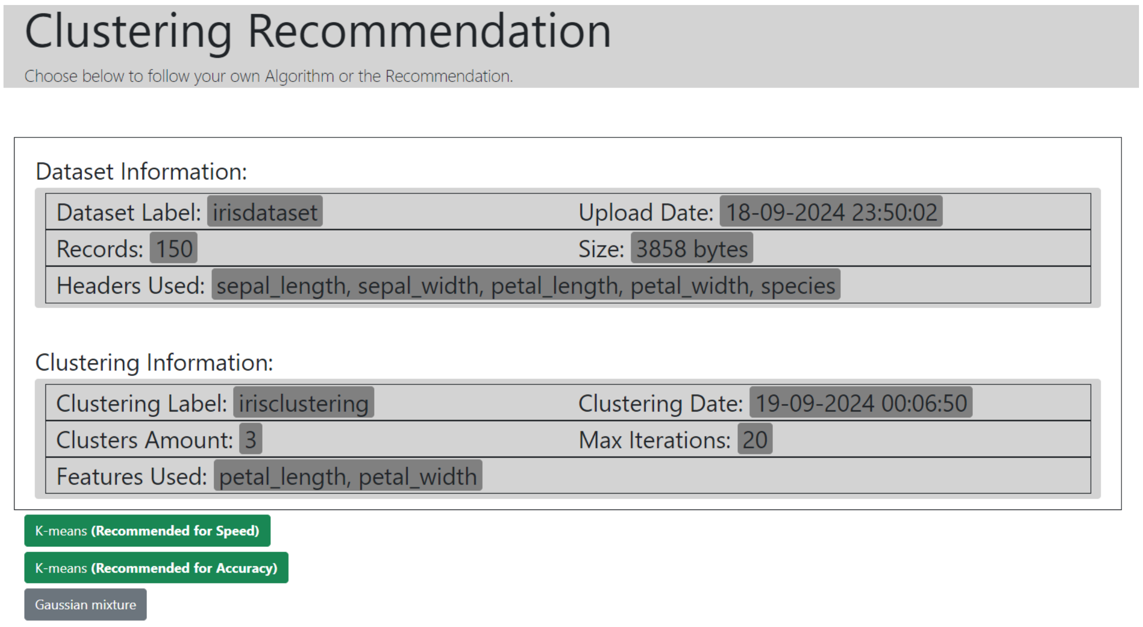
Choose the Headers to be used: sepal_length, sepal_width, petal_length, petal_width, species

Amount of Clusters: 3

Max Iteration Number: 20

Clustering Algorithm: Gaussian mixture

Figure 30 – Clustering Interface


Clustering Recommendation

Choose below to follow your own Algorithm or the Recommendation.

Dataset Information:

Dataset Label: irisdataset	Upload Date: 18-09-2024 23:50:02
Records: 150	Size: 3858 bytes
Headers Used: sepal_length, sepal_width, petal_length, petal_width, species	

Clustering Information:

Clustering Label: irisclustering	Clustering Date: 19-09-2024 00:06:50
Clusters Amount: 3	Max Iterations: 20
Features Used: petal_length, petal_width	

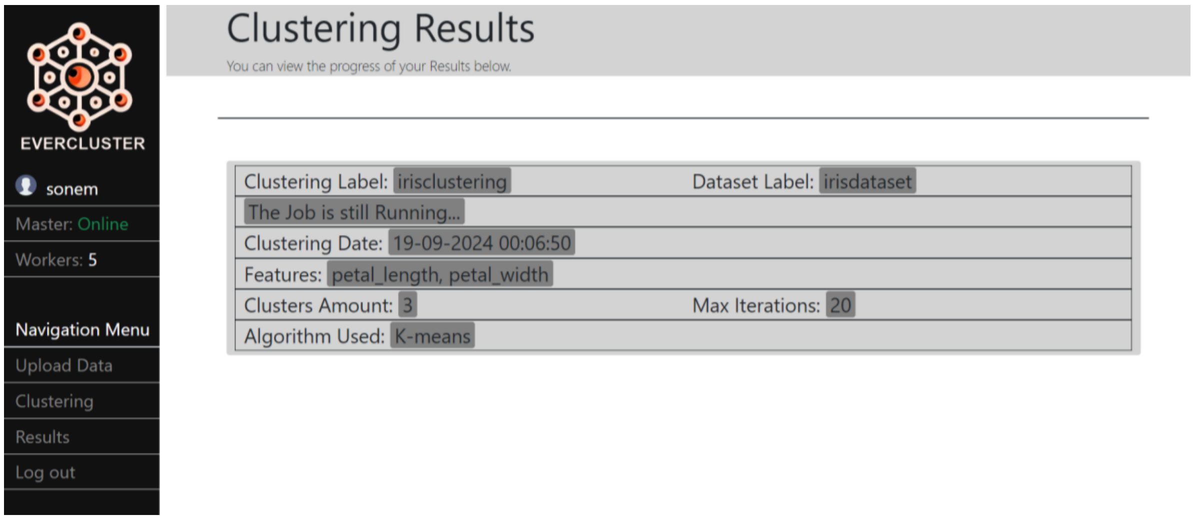
K-means **(Recommended for Speed)**
K-means **(Recommended for Accuracy)**
Gaussian mixture

Figure 31 – Clustering Recommendation

Results Interface

By selecting the button "Results" the interface that shows the results of the clustering procedures of an end-user can be accessed. Continuing the previous interface, in Figure 32, a running job is shown in the webpage of the end-user. On the fields of this job, metadata are shown regarding: the clustering label, the used dataset's label, the job's status, the features used, the number of clusters found, the max iteration used and the algorithm that was performed. The more clustering procedures the end-user is performing, the more results are shown in this interface. After refreshing the page for a couple of seconds the end-user will result in the Figure 33. In this figure, it is visible that the job was completed with results based on its execution speed and accuracy. The end-user can now remove completely from the platform the result if needed or download the results locally as well.

In the case that the end-user wants to view this page before they execute any clustering procedure, the message visualized in Figure 34 shown.

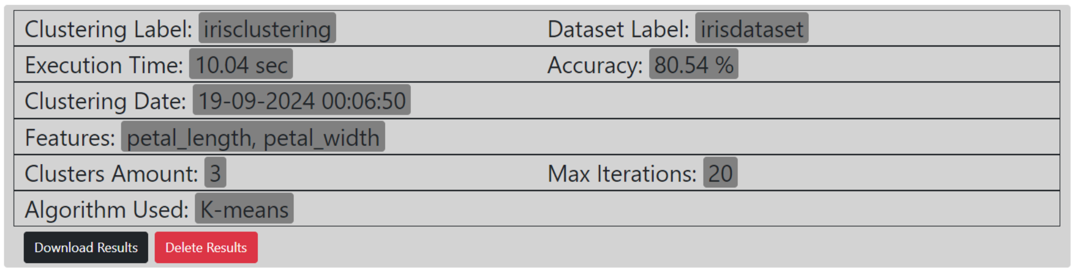


Clustering Results

You can view the progress of your Results below.

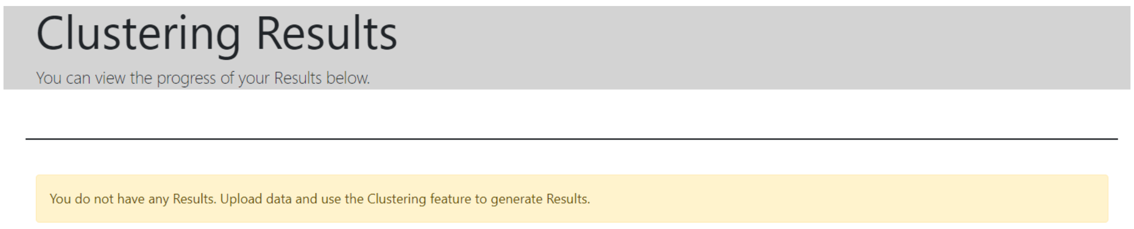
Clustering Label: irisclustering Dataset Label: irisdataset
The Job is still Running...
Clustering Date: 19-09-2024 00:06:50
Features: petal_length, petal_width
Clusters Amount: 3 Max Iterations: 20
Algorithm Used: K-means

Figure 32 – Results Interface



Clustering Label: irisclustering Dataset Label: irisdataset
Execution Time: 10.04 sec Accuracy: 80.54 %
Clustering Date: 19-09-2024 00:06:50
Features: petal_length, petal_width
Clusters Amount: 3 Max Iterations: 20
Algorithm Used: K-means

Figure 33 – Results Example



Clustering Results

You can view the progress of your Results below.

You do not have any Results. Upload data and use the Clustering feature to generate Results.

Figure 34 – No Results Example

5. Experimentation Methodology

This section provides a comprehensive analysis of the steps followed to conduct experiments on EverCluster. The experimentation use case is outlined as well as the reasoning behind it. The resources used for experimentation are outlined, giving to the readers a clear understanding of the environment used for experimentation. It's important to note that readers can replicate these experiments by following the deployment diagrams discussed in the previous section. This ensures that the experiments are reproducible, while also allowing interested parties to adapt the analysis to other deployment methods and design their own experiments.

Following the description of resources, the structure of the datasets used in the experiments is analyzed. The experimentation procedure is then detailed, beginning with the synthesization of datasets used to train two classifiers, as explained in the platform's architecture. The training process for these classifiers is also discussed, with code provided to offer transparency. Finally, the preprocessing steps for the datasets and their application within the platform are demonstrated.

5.1. Experimentation Use Case

The experimentation performed on EverCluster tries to validate the ability of the platform to make clustering recommendations to its end-users, by combining knowledge from the dataset given and the infrastructure that the platform is deployed upon. For this to be achieved, different datasets were normalized, partitioned and then used in the platform. Each time the clustering procedure was created in the platform, its recommendations were noted. By doing so, the end-user could verify if EverCluster was correct in its recommendation or not.

5.2. Experimentation Resources

To perform the experiment that are analyzed in the following subsection, the single-node deployment method was performed using Docker, and the resources described in Table 4 were given to each container of the Spark Cluster. The containers for the MinIO, Mongo and Flask Server containers had Docker's default values. In the other hand, all the Spark Cluster's containers were given exactly one (1) Gigabyte of Random-Access Memory (RAM) and one (1) CPU core.

Table 4 – Experimentation Resources

Spark Node \ Resources	RAM	CPU
Spark Master	1024 mb (1 gb)	1 core
Spark Worker 1	1024 mb (1 gb)	1 core
Spark Worker 2	1024 mb (1 gb)	1 core
Spark Worker 3	1024 mb (1 gb)	1 core
Spark Worker 4	1024 mb (1 gb)	1 core
Spark Worker 5	1024 mb (1 gb)	1 core

5.3. Experimentation Datasets

To limit the bias aspect from the experimentation results, five (5) different datasets have been found on Kaggle [106]. Each dataset regards a different field in the industry. Table 5 presents all the information of the aforementioned datasets. In the following subsections, each dataset is explained, by providing their different features, as well as any needed measurements provided by their Kaggle users.

Table 5 – Experimentation Datasets

Dataset ID	Topic	Bitesize	Records Amount	Features Amount
DS1	Breast Cancer	125141 bytes	569	32
DS2	Covid 19 Cases	18083 bytes	225	10
DS3	Credit Cards	16483 bytes	660	7
DS4	Iris	3858 bytes	150	5
DS5	Mall Customers	3981 bytes	200	5

5.3.1. Breast Cancer Dataset

The dataset provided in [107] regards the Breast Cancer domain, providing the following features: id, diagnosis, radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, concave_points_mean, symmetry_mean, fractal_dimension_mean, radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave_points_se, symmetry_se, fractal_dimension_se, radius_worst, texture_worst, perimeter_worst, area_worst, smoothness_worst, compactness_worst, concavity_worst, concave_points_worst, symmetry_worst, fractal_dimension_worst.

As described by the user who uploaded the dataset, the values of the above features were created Synthetically in order to be studied and create good classification paradigms regarding predictions of Breast Cancer on patients. Appendix XXI shows three random records of this dataset in order for any concerned reader to have a clear view of the raw dataset's form.

5.3.2. Covid 19 Cases Dataset

The dataset provided in [108] regards the Covid-19 pandemic that happened few years before the release of this document. It regards mortality rates based on different countries. The features of this dataset are the following: Country, Other names, ISO 3166-1 alpha-3 CODE, Population, Continent, Total Cases, Total Deaths, Tot Cases/1M pop, Tot Deaths/1M pop, Death percentage

For this dataset to be used, there was a need for a preprocessing procedure. The dataset contained the comma character in some values that EverCluster was able to understand and not allow the dataset to be uploaded. To resolve the mentioned issue, the comma characters that existed between two quotation marks where removed. Appendix XXII shows three random records of this dataset in order for any concerned reader to have a clear view of the preprocessed dataset's form.

5.3.3. Credit Cards Dataset

The dataset provided in [109] concerns information that was gathered by credit card users in order to identify Loyal Customers, Customer Segmentation, Targeted Marketing and other such use cases in the Marketing Industry. The features of this dataset are the following: Sl_No, Customer Key, Avg_Credit_Limit, Total_Credit_Cards, Total_visits_bank, Total_visits_online, Total_calls_made. Appendix XXIII shows three random records of this dataset in order for any concerned reader to have a clear view of the raw dataset's form.

5.3.4. Iris Dataset
The Iris dataset [110] is a well-known dataset in ML and statistics. It contains one hundred and fifty (150) samples of iris flowers from three species: Iris setosa, Iris versicolor, and Iris virginica. The features of the dataset are the following: sepal_length, sepal_width, petal_length, petal_width, species. Appendix XXIV shows three random records of this dataset in order for any concerned reader to have a clear view of the raw dataset's form.

5.3.5. Mall Customers Dataset
Dataset [111] is centered around customer segmentation for a supermarket mall, where the goal is to analyze customer data to identify different customer groups. The data is gathered from membership cards, and it contains the following features: CustomerID, Gender, Age, Annual Income (k$), Spending Score (1-100).

In essence, this dataset enables a supermarket to better understand its customer base and create more effective marketing campaigns by identifying key customer segments. Appendix XXV shows three random records of this dataset in order for any concerned reader to have a clear view of the raw dataset's form.

5.4. Experimentation Procedure
Having described the different datasets that were used, in the following procedure, the analysis of this process can now begin. In the following subsections, the experimentation procedure is broken down to four (4) steps. The steps are: (i) Synthetic Generation, (ii) Speed & Accuracy Classifiers, (iii) Dataset Normalization & Partitioning and (iv) Experimentation on EverCluster. By reading the aforementioned subsections, the understanding of the experimentation procedure is achieved. The results of the following experimentation steps are presented in the next section of the document.

5.4.1. Synthetic Generation
For EverCluster to function, two classifiers need to be trained and be used as input in the Recommendator component that was analyzed in the architecture of the platform. In this scenario, the two classifiers were created using the RF Classification algorithm. For this to be achieved, there is a need for a training dataset. In one hand, there are many ways of finding datasets in places like Kaggle and generally the internet. In the other hand, for this experimentation a synthetic dataset was created. This decision was taken in order to use a dataset that regards specifically the infrastructure used and described in the *Experimentation Resources* subsection. In Appendix XXVI the code for the synthetic data creation is shown.

In this subsection, the way that the synthetic dataset was created is explained. Figure 35 visualizes, by the use of an activity diagram, the process of extracting the synthetic dataset. As it is shown in the figure, the Python code in the above Appendix is executed five (5) times, each time for a different amount of Spark Workers being online in the Spark Cluster component. More specifically the synthetic dataset is distributed across five (5) different CSV files. In the activity diagram, it is shown that for each execution of the aforementioned code, first of all, the script gets as an argument the number of online workers by the Spark Cluster. Then, the script goes over a loop for different datasets. Each dataset has a different combination of its number of features and records used. For this experimentation the features could be the following amounts: three, five, ten or twenty-five (3, 5, 10 or 25) and the records could be the following amounts: ten, twenty-five, one hundred, five hundred, one thousand or four thousand (10, 25, 100, 500, 1000 or 4000). The values in the datasets are generated randomly using the uniform [112] and the gaussian distributions trying to randomize values from zero to one, for the dataset to be normalized. After a dataset is created the script is executing a new loop that regards different clustering procedures on the same dataset. For each dataset, twenty sever (27) different clustering procedures are performed. More specifically for every generated dataset a clustering is being created with a number of clusters to be found in the span of two, three or five (2, 3 or 5) and the maximum iteration number is set in the span of ten, twenty-five, or fifty (10, 25, 50).For each clustering setup and for every dataset the setup is being executed three times, once with the k-means algorithm, once with the bisecting k-means algorithm and once with the GMM algorithm.

As an example, the first time the script is going to create a dataset with three (3) features and ten (10) records. From this information, the script will also create an average amount of bitesize for such a dataset. Then, it’s going to create a clustering setup with two (2) clusters to be found and a max iteration number of ten (10). For this dataset, the script is going to execute this setup with all the three algorithms. After the three clustering procedures are performed, the speed and accuracy results are saved in the CSV dataset for the current number of workers. Then, the clustering setup will be updated to two (2) clusters with a max iteration number of twenty-five (25) and execute again the inner loop until the end. After all the clustering setups are executed for this dataset, the dataset will be updated with a new generated random dataset of five (5) features and twenty-five (25) records. The process explained will end when there are no more datasets or clustering setup combinations to execute. As previously mentioned, the file must be executed with a different number of workers. For this experimentation procedure it was executed five (5) times. An example of the values generated and written in one of the files can be found in Appendix XXVII.

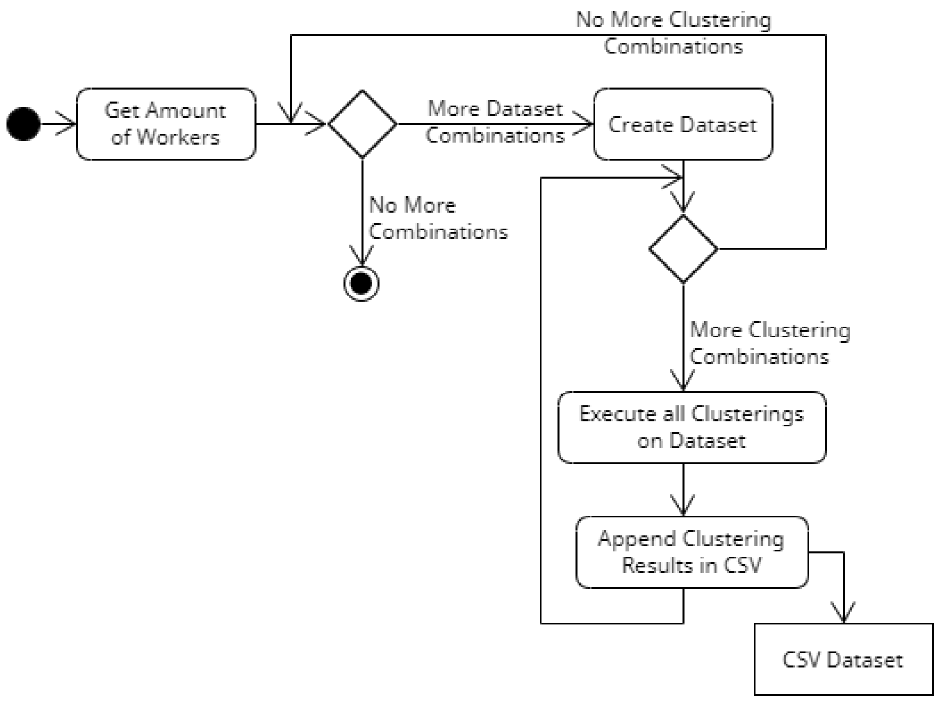

No More Clustering Combinations
Get Amount of Workers
More Dataset Combinations
Create Dataset
No More Combinations
More Clustering Combinations
Execute all Clusterings on Dataset
Append Clustering Results in CSV
CSV Dataset

Figure 35 – Synthetic Dataset Extraction

5.4.2. Speed & Accuracy Experimentation Models

As explained in the previous subsection, five CSV files are synthetically generated in order to create the two models for the EverCluster's Recommendator component. The procedure of the production of the experimentation models is visualized in Figure 36. As it is shown the creation of the models starts by combining the information of the different CSV files into one dataset. After the creation of the combined dataset, this is broken into the speed and accuracy datasets. The speed dataset has as information all the clustering executions performed in the previous step with the features: maxIteration, Clusters, Workers, ByteSize, RecordCount, FeaturesCount, BestBySpeed. From the other hand, the accuracy dataset has as information all the clustering executions performed in the previous step with the features: maxIteration, Clusters, Workers, ByteSize, RecordCount, FeaturesCount, BestByAccuracy. These datasets use their last feature to perform the RF Classification technics. A split was done in the two datasets with 80% (eighty present) of each being a training dataset and the rest the test dataset. Continuing the training procedure, the datasets were evaluated based on their WCSS metric, and their performance was 80.2% (eighty and two tenths precent) each. Appendix XXVIII presents the Python code used to create the two models, following the procedures explained in this subsection. After the creation of the two models, they were saved as Pickle (PKL) files. Finally, the mentioned files were given as input to the Recommendator component of the platform, making it able to function with its newly found "brain".

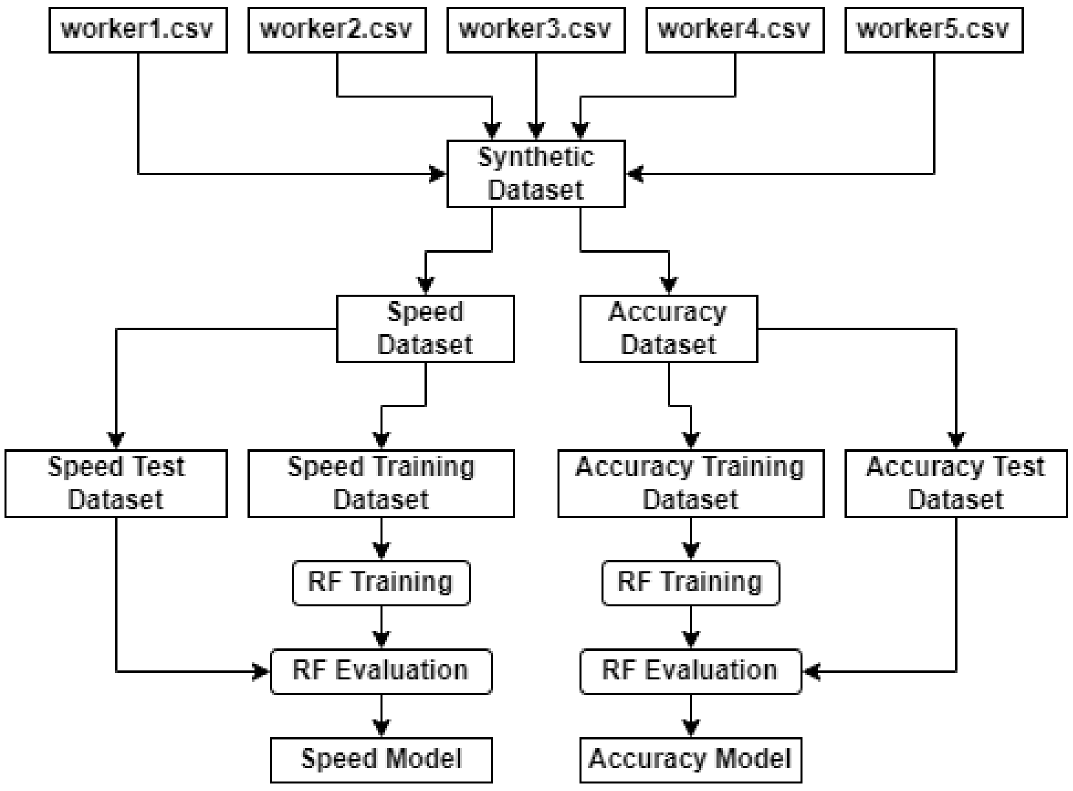


Figure 36 – Speed & Accuracy Models Creation

5.4.3. Dataset Normalization & Partitioning

As mentioned in the previous subsection, for the means of this experimentation EverCluster is exploiting two models that were trained on normalized data from zero (0) up to one (1). To ensure the meaningfulness of the results, the datasets obtained from Kaggle, which were utilized in the subsequent recommendation experimentation process, required normalization to be performed uniformly across all datasets. Appendix XXIX presents the code that was used in order to create the new normalized datasets. The numerical values of the datasets where normalized using the following function:

$$new\_value = \frac{current\_value - min\_value}{\mathrm{max\_malue} - min\_value}$$

Additionally, the string values were standardized by assigning a unique numerical value to each distinct string. Following this transformation, the mathematical formula presented before was applied to these values as well. As a result, all dataset values were normalized and prepared for processing in clustering procedures using EverCluster. An example of the normalized datasets can be found in Appendices XXX - XXXIV.

5.4.4. Experimentation on EverCluster

In this step of the experimentation procedure, EverCluster can generate recommendations based on its Recommendator component and there are five (5) different datasets to perform the clustering algorithms. To test the accuracy of EverCluster's recommendation ability, all the datasets were uploaded on the platform. For each of the datasets three clustering processes with all the supported clustering algorithms, that are provided, were executed. By doing so, the recommended algorithms for speed and accuracy from EverCluster were noted. All the three algorithms were used multiple times for each dataset to generate results.

More specifically, two hundred and seventy (270) different clustering procedures were performed on EverCluster as experiments. Each clustering procedure was named in the following manner:

```
DS<dataset_number>a<algorithm_number>c<clusters_found>m<max_iteration>w<nodes>
```

The above naming rule provides detail about the clustering that was performed. The parameters of each clustering procedure are shown in Table 6. As an example, the label "DS1a1c2m10w2" means that the DS1 dataset was used by exploiting the K-means algorithm, found two (2) clusters with a max iteration number of ten (10) and the number of nodes used in the Spark Cluster was two (2). The results of the experimentation that was described are analyzed in detail in the next section.

Table 6 – Experimentation Parameter Values

Parameter	Description	Values
Dataset	The dataset used for the clustering.	DS1, DS2, DS3, DS4, DS5
Algorithm	The algorithm used for the clustering.	1) K-means 2) Gaussian Mixture 3) Bisecting K-means
Clusters	The number of clusters that were searched for in the dataset.	2, 3, 5
Max Iteration	The max iteration number for the clustering procedure.	10, 20, 50
Workers	The Spark Workers used for the clustering.	2, 5

6. Experimentation Results

Building upon the Experimentation Methodology outlined in the previous section, this section presents a comprehensive analysis of the results and insights gathered from the experimentation process. It focuses on the accuracy of EverCluster's recommendations and includes key metrics for the three algorithms utilized. Detailed results of the experimentation can be found in Appendices XXXV – LII.

These appendices display the outcomes of each execution, with tables categorized by speed and accuracy. For each clustering combination, EverCluster's recommendations are noted in the right column, while the speed and accuracy metrics for each algorithm are presented in the remaining columns. To enhance clarity, correct recommendations are highlighted in green, while incorrect ones are indicated in orange. Additionally, the tables identify the best-performing algorithm for each case in green and the least effective one in orange.

6.1. Recommendation Performance

This subsection addresses the recommendations generated by EverCluster, outlining the platform's advantages and disadvantages. This information is provided by summarizing the information of showcased in the tables of the appendix. To enhance clarity for the reader, the results are divided into two subsections: the first focuses on recommendations related to the platform's processing speed, whereas the second addresses recommendations concerning the accuracy.

6.1.1. Speed Recommendations Performance

As mentioned, in this subsection the information of the results provided in the appendices of this document is summarized. The summarization is being done based on: workers used, dataset used, clusters found and the max iteration number limitation that was set. The following information that is presented in Tables 7 – 10 regards only the speed recommendations.

Based on the conducted experiments, it was reported that the speed ML model, created using the generated synthetic data, achieved an accuracy of 80%. As shown in Table 7, in the experimentation done, the performance of the speed recommendations was 65.5%. It is evident that this result falls significantly short of the model's accuracy percentage. To identify the core cause, it is necessary to perform an examination to the remaining result tables explicitly.

Table 7 presents the speed recommendations summary based on the nodes used in the Spark Cluster. The table presents, with the labels "Correct" and "Wrong", two (2) rows, counting the correct and wrong recommendations of EverCluster. In this way, the accuracy percentage based on the nodes used is calculated in the final column. It is apparent that the accuracy across the nodes ranged between 60% and 70%.

Table 7 – Speed Recommendations Summary based on Nodes

Worker Amount	Correct	Wrong	Total	Percentage
2 Nodes	32	13	45	71.1 %
5 Nodes	27	18	45	60.0 %
Total	59	31	90	65.5 %

Table 8 represents the same evaluation metrics as described above based on the dataset that were used. By cross validating this table with Table 5, the platform recommends the correct algorithm mostly on datasets with a small number of features, having a high failure rate on the dataset DS1. On Table 9 the speed recommendation success rate is shown based on the number of clusters that were searched for. It is apparent that on two clusters, has a high-performance rate, but as the clusters to be found are increasing the performance is slowly dropping. Table 10 reveals a low-performance rate of the platform with lower maximum iteration numbers but demonstrates a high-performance rate (96.6%) as the maximum iteration count increases to fifty percent (50 %).

Table 8 – Speed Recommendations Summary based on Datasets

Dataset Name	Correct	Wrong	Total	Percentage
DS1	8	10	18	44.4 %
DS2	12	6	18	66.6 %
DS3	13	5	18	72.2 %
DS4	13	5	18	72.2 %
DS5	13	5	18	72.2 %
Total	59	31	90	65.5 %

Table 9 – Speed Recommendations Summary based on Clusters Found

Clusters Found	Correct	Wrong	Total	Percentage
2 Clusters	25	5	30	83.3 %
3 Clusters	18	12	30	60.0 %
5 Clusters	16	14	30	53.3 %
Total	59	31	90	65.5 %

Table 10 – Speed Recommendations Summary based on Max Iteration

Max Iteration	Correct	Wrong	Total	Percentage
10 Iterations	9	21	30	30.0 %
20 Iterations	21	9	30	70.0 %
50 Iterations	29	1	30	96.6 %
Total	59	31	90	65.5 %

The most crucial part of the speed related tables is that, as mentioned above, the trained accuracy of the speed recommendation model was 80%, but the performance of the platform was 65.5%. These values show significant statistical disparity. By examining in detail, the results of the tables, on Table 7, there are a total of thirty-one (31) wrong recommendations throughout the experimentation. A significant portion of the failed recommendations is recorded in Table 6.1.1-4, with twenty-one (21) incorrect recommendations resulting when the maximum iteration number was set to ten (10). According to the data, 67.7% of the incorrect recommendations were due to the low iteration count. A low iteration number, such as ten, makes the results highly susceptible to interference from background system processes. Furthermore, it is evident that the accuracy of the speed recommendations improves as the maximum iteration number increases.

An alternative option to express the total accuracy of EverCluster based on its speed recommendations, is removing the experiments of a max iteration number equal to the value of ten (10), as outliers due to the reasoning that was referred in the previous paragraph. By doing so, the new speed recommendation performance is equal to fifty correct recommendations out of sixty (50 out of 60), bringing the new speed recommendation performance as high as 83.3%, being much closer to the accuracy found in the ML model that was created for the corresponding classifier.

6.1.2. Accuracy Recommendations Performance

As mentioned, in this subsection the information of the results provided in the appendices of this document is summarized. The summarization is being done based on workers used, dataset used, clusters found and the max iteration number limitation that was set. The following information that is presented in Tables 11 – 14 regards only the accuracy recommendations.

As presented in Table 11, the experimentation demonstrated an accuracy performance of 81.1% for the recommendations. This percentage aligns with the accuracy of the trained model implemented in the Recommendator component.

Table 11 presents the accuracy recommendations summary based on the nodes used in the Spark Cluster. The table counts two (2) rows, counting the correct and wrong recommendations of EverCluster. In this way, the accuracy percentage based on the nodes used is calculated in the final column. It is evident that the accuracy across the nodes ranged between 70% and 90%.

Table 11 – Accuracy Recommendations Summary based on Nodes

Worker Amount	Correct	Wrong	Total	Percentage
2 Nodes	33	12	45	73.3 %
5 Nodes	40	5	45	88.8 %
Total	73	17	90	81.1 %

Table 12 represents the same calculations based on the dataset that were used. By cross validating this table with Table 5, the platform recommends the correct algorithm most of the times with a higher success rate on datasets with a small number of features. On Table 13 the accuracy recommendation success rate is shown based on the number of clusters that were searched for. It can also be seen that on two clusters there is a high-performance rate, but as the clusters to be found are increasing the performance is slowly dropping. Table 14 showcases in the same way with the above tables the results based on the max iteration limitation that was provided for each clustering procedure. In the last table, there seems to be a success rate of above 70% on all the cases with an indication of the success rate to be dropping as the max iteration number is increasing.

Table 12 – Accuracy Recommendations Summary based on Datasets

Dataset Name	Correct	Wrong	Total	Percentage
DS1	12	6	18	66.6 %
DS2	13	5	18	72.2 %
DS3	17	1	18	94.4 %
DS4	17	1	18	94.4 %
DS5	14	4	18	77.7 %
Total	73	17	90	81.1 %

Table 13 – Accuracy Recommendations Summary based on Clusters Found

Clusters Found	Correct	Wrong	Total	Percentage
2 Clusters	29	1	30	96.6 %
3 Clusters	26	4	30	86.6 %
5 Clusters	18	12	30	60.0 %
Total	73	17	90	81.1 %

Table 14 – Accuracy Recommendations Summary based on Max Iteration

Max Iteration	Correct	Wrong	Total	Percentage
10 Iterations	25	5	30	83.3 %
20 Iterations	25	5	30	83.3 %
50 Iterations	23	7	30	76.6%
Total	73	17	90	81.1 %

The accuracy related tables, seem to be consistent across all the parameters. An interesting aspect of the above results could be that on the lower number of clusters, max iteration number and features used of the exploited dataset the platform's recommendation ability creates a statistical peak.

6.2. Algorithm Performances

This section corresponds to the results found as a comparison to the exploited algorithms from EverCluster. In the clustering experimentations the usage of K-means algorithm was most of the times the best algorithm to be used in the datasets that were exploited. In this section, K-means' performance is shown. This algorithm is also compared with the other two algorithms used. K-means algorithm is examined across the experimentation that was performed on EverCluster. To achieve this, Tables 15 – 18 present the number of times that K-means was the best and the worst algorithm for the executed clustering procedure based on speed and accuracy. Similar tables can be found in Appendices LII – LV for the Gaussian Mixture algorithm and in Appendices LVI – LIX for the Bisecting K-means algorithm.

In their structure, all the aforementioned tables have the same structure with the rest that were provided for the platform's performance analysis. In Table 15 K-means outperforms in comparison with the rest of the algorithms in the two deployment setups. Also, Table 16 provides measurements that correspond to the mentioned algorithm's usage per dataset. In both speed and accuracy measurements, the algorithm appears to perform optimally when applied to datasets with fewer features, while also avoiding being the worst choice among the algorithms used in each case.

Furthermore, K-means' simpler approach on the clustering appears to make it a preferable choice over the other algorithms in terms of speed as the number of clusters to be identified increases (Table 17). In the other hand, K-means is losing each accuracy while the rest of the cluster to be found are increasing. Lastly, Table 18 shows that the different limitations on the number of iterations does not have a notable impact on its speed or accuracy.

Table 15 – K-means Summary based on Nodes

K-means Speed Summary based on Nodes					
Worker Amount	Best	Worst	None	Total	Percentage (Best)
2 Nodes	37	0	8	45	82.2 %
5 Nodes	36	1	8	45	80.0 %
Total	73	1	16	90	81.1 %
K-means Accuracy Summary based on Nodes					
Worker Amount	Best	Worst	None	Total	Percentage (Best)
2 Nodes	41	1	3	45	91.1 %
5 Nodes	40	0	5	45	88.8 %
Total	81	1	8	90	90.0 %

Table 16 – K-means Summary based on Datasets

K-means Speed Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	13	0	5	18	72.2 %
DS2	14	0	4	18	77.7 %
DS3	15	0	3	18	83.3 %
DS4	15	1	2	18	83.3 %
DS5	16	0	2	18	88.8 %
Total	73	1	16	90	81.1 %
K-means Accuracy Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	12	0	6	18	66.6 %
DS2	18	0	0	18	100 %
DS3	18	0	0	18	100 %
DS4	18	0	0	18	100 %
DS5	15	1	2	18	83.3 %
Total	81	1	8	90	90.0 %

Table 17 – K-means Summary based on Clusters Found

K-means Speed Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	21	0	9	30	70.0 %
3 Clusters	25	0	5	30	83.3 %
5 Clusters	27	1	2	30	90.0 %
Total	73	1	16	90	81.1 %
K-means Accuracy Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	29	1	0	30	96.6 %
3 Clusters	28	0	2	30	93.3 %
5 Clusters	24	0	6	30	80.0 %
Total	81	1	8	90	90.0 %

Table 18 – K-means Summary based on Max Iteration

K-means Speed Summary based on Max Iterations					
Max Iteration	**Correct**	**Worst**	**None**	**Total**	**Percentage (Best)**
10 Iterations	15	0	15	30	50 %
20 Iterations	29	1	0	30	96.6%
50 Iterations	29	0	1	30	96.6 %
Total	73	1	16	90	81.1 %
K-means Accuracy Summary based on Max Iterations					
Max Iteration	**Correct**	**Worst**	**None**	**Total**	**Percentage (Best)**
10 Iterations	27	0	3	30	90.0 %
20 Iterations	27	1	2	30	90.0 %
50 Iterations	27	0	3	30	90.0 %
Total	81	1	8	90	90.0 %

7. Discussion & Next Steps

In this concluding section, the document is reviewed in its entirety, with emphasis placed on summarizing the experimental findings, addressing challenges encountered during the research process, and offering guidance for future investigations in related areas.

This document introduces EverCluster, a comprehensive platform designed to manage dataset uploads, metadata organization, execution of clustering procedures, and the provision of results to end-users. The platform's architecture is examined in detail, incorporating both high- and low-level descriptions, supported by deployment diagrams and activity diagrams that elucidate the architectural flow.

As outlined in prior sections, the experiments conducted are reproducible, and it is strongly recommended that readers attempt to replicate them. This will facilitate the verification of results or offer opportunities for critical analysis by other researchers and relevant stakeholders. Furthermore, the ML models developed can be reconstructed by users and utilized within the EverCluster recommendation subcomponent, thereby generating new or potentially more accurate results.

The platform is cloud-centric in design, allowing for seamless deployment across a variety of container orchestration systems. This versatility enables EverCluster to be employed not only for experimental purposes, as previously discussed, but also for broader applications by interested parties.

7.1. Conclusions

The experimental results showcase several important findings regarding the performance and accuracy of the recommendations provided by EverCluster. The following insights have been observed concerning its ability to recommend optimal algorithms:

- The platform’s recommendations for the best algorithm based on speed exhibit an overall success rate of 65.5%. This success rate improves as the algorithm is allotted more time to form clusters, reaching a success rate of 83.3% with a higher number of iterations.
- Additionally, the platform demonstrates a consistent success rate in its accuracy-based recommendations, maintaining an average of 81.1% across various scenarios.

Further analysis shows that EverCluster performs more effectively on datasets with fewer features, resulting in increased success rates for both speed and accuracy recommendations. However, as the maximum iteration limit is extended, the success rate for speed-based recommendations improves, while the success rate for accuracy-based recommendations declines. These observations suggest that EverCluster excels in providing high success rates for speed recommendations, particularly when applied to datasets with a limited number of features but involving clustering procedures with numerous iterations.

Another notable conclusion drawn from the experimentation pertains to the consistently high success rates of the K-means algorithm compared to Bisecting K-means and Gaussian Mixture models. It appears that the synthetic data provided to the Recommendator models may have significantly influenced the platform’s frequent recommendation of the K-means algorithm.

Nonetheless, even accounting for this potential bias, K-means was most often identified as the best-performing algorithm across the majority of experiments.

7.2. Difficulties Faced

This subsection is used in order to provide to the reader the difficulties that have been faced in the deployment and experimentation described in this document.

The following were the difficulties that any concerned party that wants to recreate the experimentation performed or in general wants to deploy EverCluster needs to know about:

- *Single-Node Deployment*: The platform's architecture includes three deployment diagrams that demonstrate EverCluster's capability to be deployed in a multi-node environment. However, due to resource constraints, this setup could not be implemented for experimentation within this document, which was instead conducted using a single-node deployment. A multi-node deployment is feasible with EverCluster, and readers with access to the necessary resources are encouraged to perform such a deployment and generate additional results.
- *Training Datasets*: As it was expressed in the architecture analysis of the platform, the Recommendator component needs to ML models in order to achieve its recommendation abilities. For the models to be created training datasets are needed. In this document, the training datasets proposed are synthesized based on logic given in the appendices.

7.3. Future Work

In this subsection, points are given to the concerned readers as next steps for the current work that was performed in this document.

- *Synthetic to real data for modes*: As mentioned, the Recommendator subcomponent that was analyzed in this document is trained on synthetic data. These data try to simulate the real data on different clustering scenarios. One next step of this work would be to get more results like the ones provided in the appendices for two and five Spark Workers. In that manner, a real dataset of such execution results xan be generated and combined to train more accurate classifiers for the aforementioned component.

- *Deploy EverCluster on Multi-Node Deployment*: In this research, EverCluster was deployed as a single-node installation. Future work for this document would be to deploy the platform on a multi-node infrastructure and collecting again the results by performing once more the experimentation of this document.

- *Expand the range of clustering parameters as recommendation metrics*: Incorporating additional clustering parameters, such as the number of clusters and maximum iterations, into the recommendation process of EverClusters is essential to ensure results are tailored to the specific context of each clustering execution. A potential future enhancement to this work would involve refactoring the Recommendator subcomponent to accommodate a broader set of parameters, tailored to the unique characteristics of the various algorithms supported by the platform.

- *Experiment additional classification algorithms for the Recommendator classifiers*: In this research, the usage of RF Classification algorithm is being utilized to create ML models for the recommendation purposes of the platform. This algorithm can be changed and by doing so the accuracy of the mentioned models will change as well. A comparison study on these algorithms on the EverCluster's recommendations would be another interesting next step.

- *Research on non-star-shaped networks*: The installation explained and the deployment diagrams that have been provided, regard a star-shaped network. One next step of this document would be to have a comparison study on the use of such non-star-shaped networks for the deployment of the platform.

- *Dynamic recommendations based on infrastructure*: More future work can be found on the dynamic adaptability of such systems. More specifically, EverCluster can present the best algorithms based on the cluster's number of nodes. It would be of value to find out if something similar can be achieved based on the topology of the infrastructure nodes.

- *Dynamic algorithm adaptation on clustering execution*: One of the most challenging and compelling next steps for this work involves enhancing the platform's decision-making capabilities. Currently, the platform provides algorithm recommendations, allowing users to choose whether to follow them. The proposed advancement would enable the platform to automatically identify the optimal algorithm for any ongoing clustering procedure. In this scenario, the platform should be capable of adjusting its recommendation if changes occur in the infrastructure topology, such as the addition or removal of nodes. Crucially, this dynamic adjustment would allow the platform to switch the clustering algorithm without losing the progress of the ongoing process.

References

[1]. "Data growth worldwide 2010-2025 | Statista," Statista, Nov. 16, 2023. https://www.statista.com/statistics/871513/worldwide-data-created/

[2]. Kulkarni, R. (2019, February 7). Big data goes big. Forbes. https://www.forbes.com/sites/rkulkarni/2019/02/07/big-data-goesbig/?sh=5b985d0920d7

[3]. V. Rohilla, M. S. S. kumar, S. Chakraborty, and Ms. S. Singh, "Data Clustering using Bisecting K-Means," 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), vol. 1. IEEE, pp. 80–83, Oct. 2019. doi: 10.1109/icccis48478.2019.8974537.

[4]. K. Das and R. N. Behera, "A Survey on Machine Learning: Concept,Algorithms and Applications," International Journal of Innovative Research in Computer and Communication Engineering, vol. 5, no. 2, Jan. 2017, [Online]. Available: https://www.rroij.com/open-access/a-survey-on-machine-learning-conceptalgorithms-and-applications-.pdf

[5]. M. Crawford, T. M. Khoshgoftaar, J. D. Prusa, A. N. Richter, and H. Al Najada, "Survey of review spam detection using machine learning techniques," Journal of Big Data, vol. 2, no. 1. Springer Science and Business Media LLC, Oct. 05, 2015. doi: 10.1186/s40537-015-0029-9.

[6]. M. Vespe, I. Visentini, K. Bryan, and P. Braca, "Unsupervised learning of maritime traffic patterns for anomaly detection," 9th IET Data Fusion &amp; Target Tracking Conference (DF&amp;TT 2012): Algorithms &amp; Applications. IET, pp. 14–14, 2012. doi: 10.1049/cp.2012.0414.

[7]. J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," Machine Learning, vol. 109, no. 2. Springer Science and Business Media LLC, pp. 373–440, Nov. 15, 2019. doi: 10.1007/s10994-019-05855-6.

[8]. K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," IEEE Signal Processing Magazine, vol. 34, no. 6. Institute of Electrical and Electronics Engineers (IEEE), pp. 26–38, Nov. 2017. doi: 10.1109/msp.2017.2743240.

[9]. F. Zhuang et al., "A Comprehensive Survey on Transfer Learning," Proceedings of the IEEE, vol. 109, no. 1. Institute of Electrical and Electronics Engineers (IEEE), pp. 43–76, Jan. 2021. doi: 10.1109/jproc.2020.3004555.

[10]. R. C. Sofia et al., "A Framework for Cognitive, Decentralized Container Orchestration," IEEE Access, vol. 12. Institute of Electrical and Electronics Engineers (IEEE), pp. 79978–80008, 2024. doi: 10.1109/access.2024.3406861.

[11]. T. Mitchell et al., "Never-ending learning," Communications of the ACM, vol. 61, no. 5. Association for Computing Machinery (ACM), pp. 103–115, Apr. 24, 2018. doi: 10.1145/3191513.

[12]. W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, and E. Xing, "High-Performance Distributed ML at Scale through Parameter Server Consistency Models," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, no. 1. Association for the Advancement of Artificial Intelligence (AAAI), Feb. 09, 2015. doi: 10.1609/aaai.v29i1.9195.

[13]. "Apache Hadoop." https://hadoop.apache.org/

[14]. M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means Algorithm: A Comprehensive Survey and Performance Evaluation," Electronics, vol. 9, no. 8. MDPI AG, p. 1295, Aug. 12, 2020. doi: 10.3390/electronics9081295.

[15]. S. Banerjee, A. Choudhary, and S. Pal, "Empirical evaluation of K-Means, Bisecting K-Means, Fuzzy C-Means and Genetic K-Means clustering algorithms," 2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE). IEEE, Dec. 2015. doi: 10.1109/wiecon-ece.2015.7443889.

[16]. H. Wan, H. Wang, B. Scotney, and J. Liu, "A Novel Gaussian Mixture Model for Classification," 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). IEEE, Oct. 2019. doi: 10.1109/smc.2019.8914215.

[17]. E. Patel and D. S. Kushwaha, "Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model," Procedia Computer Science, vol. 171. Elsevier BV, pp. 158–167, 2020. doi: 10.1016/j.procs.2020.04.017.

[18]. V. Jirkovsky, M. Obitko, and V. Marik, "Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration," IEEE Transactions on Industrial Informatics, vol. 13, no. 2. Institute of Electrical and Electronics Engineers (IEEE), pp. 660–667, Apr. 2017. doi: 10.1109/tii.2016.2596101.

[19]. E. Y. Boateng, J. Otoo, and D. A. Abaye, "Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review," Journal of Data Analysis and Information Processing, vol. 08, no. 04. Scientific Research Publishing, Inc., pp. 341–357, 2020. doi: 10.4236/jdaip.2020.84020.

[20]. A. Figueira and B. Vaz, "Survey on Synthetic Data Generation, Evaluation Methods and GANs," Mathematics, vol. 10, no. 15. MDPI AG, p. 2733, Aug. 02, 2022. doi: 10.3390/math10152733.

[21]. R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, "Noise injection for training artificial neural networks: A comparison with weight decay and early stopping," Medical Physics, vol. 36, no. 10. Wiley, pp. 4810–4818, Sep. 25, 2009. doi: 10.1118/1.3213517.

[22]. V. C. Pezoulas et al., "Synthetic data generation methods in healthcare: A review on open-source tools and methods," Computational and Structural Biotechnology Journal, vol. 23. Elsevier BV, pp. 2892–2910, Dec. 2024. doi: 10.1016/j.csbj.2024.07.005. D

[23]. S. Xuereb, "Generation of synthetic data to improve financial prediction models," 2023. https://www.um.edu.mt/library/oar/handle/123456789/120596

[24]. "Welcome to Python.org," Python.org, Sep. 24, 2024. https://www.python.org/

[25]. "Production-Grade Container Orchestration," Kubernetes. https://kubernetes.io/

[26]. L. De Lauretis, “From Monolithic Architecture to Microservices Architecture,” 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, Oct. 2019. doi: 10.1109/issrew.2019.00050.
[27]. J. E. Smith and Ravi Nair, “The architecture of virtual machines,” Computer, vol. 38, no. 5. Institute of Electrical and Electronics Engineers (IEEE), pp. 32–38, May 2005. doi: 10.1109/mc.2005.173.
[28]. “Docker: Accelerated Container Application Development,” Docker, Jul. 08, 2024. https://www.docker.com/
[29]. M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” IEEE Access, vol. 5. Institute of Electrical and Electronics Engineers (IEEE), pp. 3909–3943, 2017. doi: 10.1109/access.2017.2685629.
[30]. P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and Virtual Machines at Scale,” Proceedings of the 17th International Middleware Conference, vol. 11. ACM, pp. 1–13, Nov. 28, 2016. doi: 10.1145/2988336.2988337.
[31]. J. Doe, "Leveraging Micro services and Containerization for Scalable Software Solutions," International Journal of Advanced Engineering Technologies and Innovations, vol. 10, no. 2, pp. 451-470, 2024.
[32]. N. Gupta, K. Anantharaj, and K. Subramani, “Containerized Architecture for Edge Computing in Smart Home : A consistent architecture for model deployment,” 2020 International Conference on Computer Communication and Informatics (ICCCI), vol. 3. IEEE, pp. 1–8, Jan. 2020. doi: 10.1109/iccci48352.2020.9104073.
[33]. K. Voulgaris et al., “A Comparison of Container Systems for Machine Learning Scenarios: Docker and Podman,” 2022 2nd International Conference on Computers and Automation (CompAuto), vol. 17. IEEE, pp. 114–118, Aug. 2022. doi: 10.1109/compauto55930.2022.00029.
[34]. “Podman.” https://podman.io/
[35]. A. Kiourtis et al., “An Autoscaling Platform Supporting Graph Data Modelling Big Data Analytics,” Studies in Health Technology and Informatics. IOS Press, Jun. 29, 2022. doi: 10.3233/shti220743.
[36]. P. Karamolegkos, A. Mavrogiorgou, A. Kiourtis, and D. Kyriazis, “EverAnalyzer: A Self-Adjustable Big Data Management Platform Exploiting the Hadoop Ecosystem,” Information, vol. 14, no. 2. MDPI AG, p. 93, Feb. 03, 2023. doi: 10.3390/info14020093.
[37]. B. Meador, A Survey of Computer Network Topology and Analysis Examples, 2008. [Online]. Available: http://www.cs.wustl.edu/~jain/cse567-08/ftp/topology/index.html. [Accessed: 26-Sep-2024].
[38]. M. Mohsin, A. AL Rammahi, and F. Rabee, “An Innovative Topologies Based on Hypercube Network Interconnection,” European Journal of Information Technologies and Computer Science, vol. 3, no. 4. European Open Science Publishing, pp. 7–13, Sep. 18, 2023. doi: 10.24018/compute.2023.3.4.92.
[39]. K.-T. Chen et al., “Reading Strategies for Graph Visualizations that Wrap Around in Torus Topology,” 2023 Symposium on Eye Tracking Research and Applications, vol. 28. ACM, pp. 1–7, May 30, 2023. doi: 10.1145/3588015.3589841.
[40]. J. ZHU, H. ZHOU, C. WANG, L. ZHOU, S. YUAN, and W. ZHANG, “A review of topology optimization for additive manufacturing: Status and challenges,” Chinese Journal of Aeronautics, vol. 34, no. 1. Elsevier BV, pp. 91–110, Jan. 2021. doi: 10.1016/j.cja.2020.09.020.
[41]. G. Kim, Y. Kim, and H. Lim, “Deep Reinforcement Learning-Based Routing on Software-Defined Networks,” IEEE Access, vol. 10. Institute of Electrical and Electronics Engineers (IEEE), pp. 18121–18133, 2022. doi: 10.1109/access.2022.3151081.
[42]. P. Almasan, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, “Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case,” Computer Communications, vol. 196. Elsevier BV, pp. 184–194, Dec. 2022. doi: 10.1016/j.comcom.2022.09.029.
[43]. S. Rendle, W. Krichene, L. Zhang, and J. Anderson, “Neural Collaborative Filtering vs. Matrix Factorization Revisited,” Fourteenth ACM Conference on Recommender Systems. ACM, Sep. 22, 2020. doi: 10.1145/3383313.3412488.
[44]. S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep Learning Based Recommender System,” ACM Computing Surveys, vol. 52, no. 1. Association for Computing Machinery (ACM), pp. 1–38, Feb. 25, 2019. doi: 10.1145/3285029.
[45]. N. Idrissi and A. Zellou, “A systematic literature review of sparsity issues in recommender systems,” Social Network Analysis and Mining, vol. 10, no. 1. Springer Science and Business Media LLC, Feb. 11, 2020. doi: 10.1007/s13278-020-0626-2.
[46]. I. Docker, Docker, 2020. [Online]. Available: https://www.docker.com/what-docker. [Accessed: 26-Sep-2024].
[47]. “‘Swarm mode,’” Docker Documentation, Sep. 10, 2024. https://docs.docker.com/engine/swarm/
[48]. N. Singh et al., “Load balancing and service discovery using Docker Swarm for microservice based big data applications,” Journal of Cloud Computing, vol. 12, no. 1. Springer Science and Business Media LLC, Jan. 07, 2023. doi: 10.1186/s13677-022-00358-7.
[49]. “Google - About Google, Our Culture & Company News.” https://about.google/
[50]. CNCF, “Cloud Native Computing Foundation,” CNCF. https://www.cncf.io/
[51]. D. Balla, C. Simon, and M. Maliosz, “Adaptive scaling of Kubernetes pods,” NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, Apr. 2020. doi: 10.1109/noms47738.2020.9110428.
[52]. C. Carrión, “Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges,” ACM Computing Surveys, vol. 55, no. 7. Association for Computing Machinery (ACM), pp. 1–37, Dec. 15, 2022. doi: 10.1145/3539606.
[53]. “Apache SparkTM - Unified Engine for large-scale data analytics.” https://spark.apache.org/
[54]. K. Kalia and N. Gupta, “Analysis of hadoop MapReduce scheduling in heterogeneous environment,” Ain Shams Engineering Journal, vol. 12, no. 1. Elsevier BV, pp. 1101–1110, Mar. 2021. doi: 10.1016/j.asej.2020.06.009.
[55]. “Download the Latest Java LTS Free.” https://www.oracle.com/java/technologies/downloads/
[56]. “The Scala Programming Language.” https://www.scala-lang.org/

[57]. "R: The R Project for Statistical Computing." https://www.r-project.org/
[58]. S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications," IEEE Transactions on Knowledge and Data Engineering. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–1, 2020. doi: 10.1109/tkde.2020.2975652.
[59]. N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, "A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench," Journal of Big Data, vol. 7, no. 1. Springer Science and Business Media LLC, Dec. 2020. doi: 10.1186/s40537-020-00388-5.
[60]. "MongoDB: The Developer Data Platform," MongoDB. https://www.mongodb.com/
[61]. MinIO, Inc., "MinIO | S3 Compatible Storage for AI," MinIO. https://min.io/
[62]. W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review," Big Data and Cognitive Computing, vol. 7, no. 2. MDPI AG, p. 97, May 12, 2023. doi: 10.3390/bdcc7020097.
[63]. "MySQL." https://www.mysql.com/
[64]. "PostgreSQL," PostgreSQL, Sep. 26, 2024. https://www.postgresql.org/
[65]. "AI-Enhanced Data Solutions with Database 23ai." https://www.oracle.com/database/
[66]. "SQL Server Downloads | Microsoft." https://www.microsoft.com/en-us/sql-server/sql-server-downloads
[67]. "Apache Cassandra | Apache Cassandra Documentation," Apache Cassandra. https://cassandra.apache.org/_/index.html
[68]. "Redis - The Real-time Data Platform," Redis, Sep. 04, 2024. https://redis.io/
[69]. Neo4j, "Neo4j Graph Database & Analytics | Graph Database Management System," Graph Database & Analytics, Jul. 31, 2024. https://neo4j.com/
[70]. Y. Kim and T. Kim, "HOSS: Hybrid Object Storage System for Performance Acceleration," IEEE Systems Journal, vol. 16, no. 1. Institute of Electrical and Electronics Engineers (IEEE), pp. 1483–1486, Mar. 2022. doi: 10.1109/jsyst.2021.3097383.
[71]. J.-F. Lai and S.-H. Heng, "Secure File Storage On Cloud Using Hybrid Cryptography," Journal of Informatics and Web Engineering, vol. 1, no. 2. Multimedia University, pp. 1–18, Sep. 15, 2022. doi: 10.33093/jiwe.2022.1.2.1.
[72]. J. Li, Q. Wang, P. P. C. Lee, and C. Shi, "An In-depth Comparative Analysis of Cloud Block Storage Workloads: Findings and Implications," ACM Transactions on Storage, vol. 19, no. 2. Association for Computing Machinery (ACM), pp. 1–32, Mar. 06, 2023. doi: 10.1145/3572779.
[73]. M. Armbrust et al., "Delta lake," Proceedings of the VLDB Endowment, vol. 13, no. 12. Association for Computing Machinery (ACM), pp. 3411–3424, Aug. 2020. doi: 10.14778/3415478.3415560.
[74]. Y. Hao, M. C. Schmidt, Y. Wu, and N. C. Knutson, "Portal dosimetry scripting application programming interface (PDSAPI) for Winston-Lutz test employing ceramic balls," Journal of Applied Clinical Medical Physics, vol. 21, no. 11. Wiley, pp. 295–303, Oct. 24, 2020. doi: 10.1002/acm2.13043.
[75]. A. Ehsan, M. A. M. E. Abuhaliqa, C. Catal, and D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," Applied Sciences, vol. 12, no. 9. MDPI AG, p. 4369, Apr. 26, 2022. doi: 10.3390/app12094369.
[76]. I. Ahmad, E. Suwarni, R. I. Borman, Asmawati, F. Rossi, and Y. Jusman, "Implementation of RESTful API Web Services Architecture in Takeaway Application Development," 2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS). IEEE, Oct. 15, 2021. doi: 10.1109/ice3is54102.2021.9649679.
[77]. "Apache Spark Hidden REST API," Gist. https://gist.github.com/arturmkrtchyan/5d8559b2911ac951d34a
[78]. "Welcome to Flask — Flask Documentation (3.0.x)." https://flask.palletsprojects.com/en/3.0.x/
[79]. "W3Schools.com." https://www.w3schools.com/html/
[80]. "W3Schools.com." https://www.w3schools.com/css/
[81]. "W3Schools.com." https://www.w3schools.com/js/DEFAULT.asp
[82]. D. Ghimire, "Comparative study on Python web frameworks: Flask and Django," 2020. [Online]. Available: https://www.theseus.fi/handle/10024/339796
[83]. S. Raschka, J. Patterson, and C. Nolet, "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," Information, vol. 11, no. 4. MDPI AG, p. 193, Apr. 04, 2020. doi: 10.3390/info11040193.
[84]. B. C. Pierce, Advanced Topics in Types and Programming Languages. 2004. doi: 10.7551/mitpress/1104.001.0001.
[85]. OpenJS Foundation - openjsf.org, "jQuery." https://jquery.com/
[86]. M. O. J. T. Contributors and Bootstrap, "Bootstrap." https://getbootstrap.com/
[87]. "Jinja — Jinja Documentation (3.1.x)." https://jinja.palletsprojects.com/en/3.1.x/
[88]. "Werkzeug," PyPI, Aug. 21, 2024. https://pypi.org/project/Werkzeug/
[89]. "PySpark Overview — PySpark 3.5.3 documentation." https://spark.apache.org/docs/latest/api/python/index.html
[90]. D "Amazon S3 - Cloud Object Storage - AWS," Amazon Web Services, Inc. https://aws.amazon.com/s3/
[91]. M. M. Saeed, Z. Al Aghbari, and M. Alsharidah, "Big data clustering techniques based on Spark: a literature review," PeerJ Computer Science, vol. 6. PeerJ, p. e321, Nov. 30, 2020. doi: 10.7717/peerj-cs.321.
[92]. "pandas - Python Data Analysis Library." https://pandas.pydata.org/
[93]. R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," Information Fusion, vol. 81. Elsevier BV, pp. 84–90, May 2022. doi: 10.1016/j.inffus.2021.11.011.
[94]. "scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation." https://scikit-learn.org/stable/

[95]. "NumPy." https://numpy.org/

[96]. "SciPy." https://scipy.org/

[97]. "Matplotlib — Visualization with Python." https://matplotlib.org/

[98]. K. Phinzi, D. Abriha, and S. Szabó, "Classification Efficacy Using K-Fold Cross-Validation and Bootstrapping Resampling Techniques on the Example of Mapping Complex Gully Systems," Remote Sensing, vol. 13, no. 15. MDPI AG, p. 2980, Jul. 28, 2021. doi: 10.3390/rs13152980.

[99]. Y. Izza, A. Ignatiev, and J. Marques-Silva, "On Explaining Decision Trees," 2020, arXiv. doi: 10.48550/ARXIV.2010.11034.

[100]. R. Suwanda, Z. Syahputra, and E. M. Zamzami, "Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K," Journal of Physics: Conference Series, vol. 1566, no. 1. IOP Publishing, p. 012058, Jun. 01, 2020. doi: 10.1088/1742-6596/1566/1/012058.

[101]. P. Govender and V. Sivakumar, "Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019)," Atmospheric Pollution Research, vol. 11, no. 1. Elsevier BV, pp. 40–56, Jan. 2020. doi: 10.1016/j.apr.2019.09.009.

[102]. J. Chen, Y. Yuan, H. Gao, and T. Zhou, "Gaussian distribution-based modeling of cutting depth predictions of kerf profiles for ductile materials machined by abrasive waterjet," Materials &amp; Design, vol. 227. Elsevier BV, p. 111759, Mar. 2023. doi: 10.1016/j.matdes.2023.111759.

[103]. "'Docker Compose,'" Docker Documentation, Sep. 19, 2024. https://docs.docker.com/compose/

[104]. "Kompose - Convert your Docker Compose file to Kubernetes or OpenShift." https://kompose.io/

[105]. "Git." https://git-scm.com/

[106]. "Kaggle: Your Machine Learning and Data Science Community." https://www.kaggle.com/

[107]. "Breast Cancer Dataset," Kaggle, Jun. 17, 2022. https://www.kaggle.com/datasets/nancyalaswad90/breast-cancer-dataset?resource=download

[108]. "COVID-19 Coronavirus Pandemic," Kaggle, Apr. 05, 2022. https://www.kaggle.com/datasets/rinichristy/covid19-coronavirus-pandemic

[109]. "Credit Card Customer Data," Kaggle, May 15, 2021. https://www.kaggle.com/datasets/aryashah2k/credit-card-customer-data

[110]. "Iris dataset," Kaggle, Jul. 20, 2022. https://www.kaggle.com/datasets/himanshunakrani/iris-dataset

[111]. "Mall Customer Segmentation Data," Kaggle, Aug. 11, 2018. https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python?select=Mall_Customers.csv

[112]. J. Rosenblatt and M. K. Roychowdhury, "UNIFORM DISTRIBUTIONS ON CURVES AND QUANTIZATION," Communications of the Korean Mathematical Society, vol. 38, no. 2, pp. 431–450, Apr. 2023, doi: 10.4134/CKMS.C210434.

Appendices

Appendix I – K-means Algorithm

1. Initialization: Choose k initial cluster centroids (randomly or by another method).
2. Assignment: Assign each data point to the nearest centroid, based on the distance (typically Euclidean distance).
3. Update: Recalculate the centroid of each cluster based on the mean of all data points assigned to that cluster.
4. Repeat: Repeat the **Assignment (2)** and **Update (3)** steps until the centroids no longer change or until a predefined number of iterations is reached.

Appendix II – Bisecting K-means Algorithm

1. Initialization: Start with all data points in a single cluster.
2. Bisect: Split the cluster into two sub-clusters using the K-Means algorithm (Appendix I) with k=2.
3. Repeat: Select the cluster with the highest variance or highest Sum of Squared Errors (SSE) and **bisect (2)** it again using K-Means. Stop when the desired number of clusters is reached.

Appendix III – Gaussian Mixture Model Algorithm (GMM)

1. Initialization: Initialize the parameters (mean, variance, and mixing coefficients) for k Gaussian distributions (clusters).
2. Expectation Step (E-Step): Calculate the probability that each data point belongs to each Gaussian cluster using the current parameters.
3. Maximization Step (M-Step): Update the parameters (mean, variance, and mixing coefficients) of the Gaussian distributions to maximize the likelihood of the observed data.
4. Repeat: Repeat the **E-Step (2)** and **M-Step (3)** until convergence (the change in parameters between iterations becomes small).

Appendix IV – K-Nearest Neighbor (KNN)

1. Determine the value of k (number of neighbors).
2. For each data point in the test set:
  a. Calculate the distance between the test point and all data points in the training set.
  b. Find the k nearest neighbors to the test point.
  c. Assign the test point to the class that is most common among its k neighbors.

Appendix V – Support Vector Machine (SVM)

1. Choose a kernel function (linear, polynomial, radial basis function, etc.).
2. Find the hyperplane that maximizes the margin between the two classes in the training set.
3. For each data point in the test set:
  a. Calculate the distance between the test point and the hyperplane.
  b. Assign the test point to the class based on its distance from the hyperplane.

Appendix VI – Random Forest (RM)

```
1. Determine the number of trees in the forest.
2. For each tree in the forest:
  a. Create a bootstrap sample of the training data.
  b. Build a decision tree on the bootstrap sample.
  c. Prune the decision tree to prevent overfitting.
3. For each data point in the test set:
  a. Let each tree in the forest make a prediction.
  b. Assign the test point to the class that is most common among the predictions.
```

Appendix VII – Neural Network (NN)

```
1. Initialize the weights and biases of the neural network randomly.
2. For each epoch:
  a. For each data point in the training set:
    i. Forward propagate the data point through the network to calculate the predicted output.
    ii. Calculate the error between the predicted output and the actual output.
    iii. Backpropagate the error to update the weights and biases.
3. Repeat step 2 until the error converges or a maximum number of epochs is reached.
4. For each data point in the test set:
  a. Forward propagate the data point through the trained network to calculate the predicted output.
  b. Assign the test point to the class with the highest predicted probability.
```

Appendix VIII – Pros & Cons of SQL Databases

Pros	Cons
Strong consistency and data integrity.	Scaling SQL databases horizontally (across multiple servers) can be difficult.
Ideal for structured data with complex relationships.	Schema rigidity makes it less flexible for dynamic or unstructured data.
Well-suited for transactional applications, such as financial systems.	

Appendix IX – Pros & Cons of NoSQL Databases

Pros	Cons
Horizontal scaling capabilities make them excellent for large-scale data applications.	Lack of strict ACID compliance may result in eventual consistency, which can be unsuitable for all use cases.
Highly flexible for storing diverse and unstructured data.	May require more complex data modelling for certain applications.
Suitable for high-velocity, high-volume, and high-variability data (e.g., IoT, real-time applications).	

Appendix X – K-means Clustering Code

```
import sys

# Get the arguments
args = sys.argv[1:]

arguments = {}
for arg in args:
    key, value = arg.split("=")
    arguments[key] = value

# Parse the arguments
MONGO_HOST_PORT = arguments["MONGO_HOST_PORT"]
MINIO_HOST_PORT = arguments["MINIO_HOST_PORT"]
MINIO_USER = arguments["MINIO_USER"]
MINIO_PASS = arguments["MINIO_PASS"]
MINIO_BUCKET = arguments["MINIO_BUCKET"]
DATASET_PATH = arguments["DATASET_PATH"]
RESULT_PATH = arguments["RESULT_PATH"]
FEATURES = arguments["FEATURES"]
K = int(arguments["K"])
MAX_ITER = int(arguments["MAX_ITER"])
MONGO_DATABASE = arguments["MONGO_DATABASE"]
CLUSTER_LABEL = arguments["CLUSTER_LABEL"]

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pymongo import MongoClient
from minio import Minio
import pandas as pd
from io import BytesIO
import time

""" Minio Initializations """
minio_host = MINIO_HOST_PORT
client = Minio(
    minio_host,
    access_key=MINIO_USER,
    secret_key=MINIO_PASS,
    secure=False
)

response = client.get_object(MINIO_BUCKET, DATASET_PATH)

# Load the data into a Pandas DataFrame
df = pd.read_csv(BytesIO(response.data))

""" Spark Code """
# Initialize a Spark session
spark = SparkSession.builder \
    .appName("K-Means Application")  \
    .master("spark://spark-master:7077")   \
    .getOrCreate()

# Create an RDD from the Pandas DataFrame
df = spark.createDataFrame(df)

# Print the DataFrame schema
df.printSchema()

# Print the first 5 rows
df.show(5)

# Clustering Example
feature_columns = FEATURES.split(",")
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_df = assembler.transform(df)

# Train a KMeans model
# kmeans = KMeans().setK(K).setMaxIter(MAX_ITER).setSeed(1) # Seed for reproducibility
kmeans = KMeans().setK(K).setMaxIter(MAX_ITER)
start_time = time.time()    # Start the timer
model = kmeans.fit(assembled_df)
total_seconds = time.time() - start_time    # Calculate the time taken to cluster

# Make predictions
predictions = model.transform(assembled_df)

# Evaluate clustering by computing Within Set Sum of Squared Errors
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print(f"Silhouette with squared euclidean distance = {silhouette}")

# Show the results (excluding the 'features' column)
# predictions.select('id', 'feature1', 'feature2', 'prediction').show()
columns_to_show = [col for col in predictions.columns if col != 'features']
predictions.select(columns_to_show).show()

# Removed the 'features' column from the predictions DataFrame
predictions = predictions.drop('features')

# Make predictions RDD back to Pandas DataFrame
```

```
predictions = predictions.toPandas()

# Print the first 5 rows
# predictions.show(5)

# Write the resulting DataFrame with predictions to a new CSV file in Minio
csv_bytes = predictions.to_csv().encode('utf-8')
csv_buffer = BytesIO(csv_bytes)

client.put_object(MINIO_BUCKET,
        RESULT_PATH,
        data=csv_buffer,
        length=len(csv_bytes),
        content_type='application/csv')

# Update MongoDB with the clustering results
mongoClient = MongoClient('mongodb://'+MONGO_HOST_PORT+'/')
db = mongoClient[MONGO_DATABASE]
collection = db['users']
# Update the user's clustering results with the time taken to cluster and the accuracy
# collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL: {"time_taken": total_seconds, "accuracy": silhouette}}})
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".time_taken": total_seconds}})
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".accuracy": silhouette}})

# Stop the Spark session
spark.stop()
```

Appendix XI – Bisecting K-means Clustering Code

```
import sys

# Get the arguments
args = sys.argv[1:]

arguments = {}
for arg in args:
   key, value = arg.split("=")
   arguments[key] = value

# Parse the arguments
MONGO_HOST_PORT = arguments["MONGO_HOST_PORT"]
MINIO_HOST_PORT = arguments["MINIO_HOST_PORT"]
MINIO_USER = arguments["MINIO_USER"]
MINIO_PASS = arguments["MINIO_PASS"]
MINIO_BUCKET = arguments["MINIO_BUCKET"]
DATASET_PATH = arguments["DATASET_PATH"]
RESULT_PATH = arguments["RESULT_PATH"]
FEATURES = arguments["FEATURES"]
K = int(arguments["K"])
MAX_ITER = int(arguments["MAX_ITER"])
MONGO_DATABASE = arguments["MONGO_DATABASE"]
CLUSTER_LABEL = arguments["CLUSTER_LABEL"]

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import BisectingKMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pymongo import MongoClient
from minio import Minio
import pandas as pd
from io import BytesIO
import time

""" Minio Initializations """
minio_host = MINIO_HOST_PORT
client = Minio(
   minio_host,
   access_key=MINIO_USER,
   secret_key=MINIO_PASS,
   secure=False
)

response = client.get_object(MINIO_BUCKET, DATASET_PATH)

# Load the data into a Pandas DataFrame
df = pd.read_csv(BytesIO(response.data))

""" Spark Code """
# Initialize a Spark session
spark = SparkSession.builder \
   .appName("Bisecting K-Means Application") \
   .master("spark://spark-master:7077") \
   .getOrCreate()

# Create an RDD from the Pandas DataFrame
df = spark.createDataFrame(df)

# Print the DataFrame schema
df.printSchema()

# Print the first 5 rows
df.show(5)

# Clustering Example
```

```
feature_columns = FEATURES.split(",")
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_df = assembler.transform(df)

# Train a BisectingKMeans model
# bisecting_kmeans = BisectingKMeans().setK(K).setMaxIter(MAX_ITER).setSeed(1)  # Seed for reproducibility
bisecting_kmeans = BisectingKMeans().setK(K).setMaxIter(MAX_ITER)
start_time = time.time()    # Start the timer
model = bisecting_kmeans.fit(assembled_df)
total_seconds = time.time() - start_time    # Calculate the time taken to cluster

# Make predictions
predictions = model.transform(assembled_df)

# Evaluate clustering by computing Silhouette Score with the squared Euclidean distance
evaluator = ClusteringEvaluator(metricName="silhouette", distanceMeasure="squaredEuclidean")
silhouette = evaluator.evaluate(predictions)
print(f"Silhouette with squared euclidean distance = {silhouette}")

# Show the results (excluding the 'features' column)
columns_to_show = [col for col in predictions.columns if col != 'features']
predictions.select(columns_to_show).show()

# Remove the 'features' column from the predictions DataFrame
predictions = predictions.drop('features')

# Convert predictions RDD back to Pandas DataFrame
predictions = predictions.toPandas()

# Write the resulting DataFrame with predictions to a new CSV file in Minio
csv_bytes = predictions.to_csv().encode('utf-8')
csv_buffer = BytesIO(csv_bytes)

client.put_object(MINIO_BUCKET,
                RESULT_PATH,
                data=csv_buffer,
                length=len(csv_bytes),
                content_type='application/csv')

# Update MongoDB with the clustering results
mongoClient = MongoClient('mongodb://'+MONGO_HOST_PORT+'/')
db = mongoClient[MONGO_DATABASE]
collection = db['users']
# Update the user's clustering results with the time taken to cluster and the accuracy
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".time_taken": total_seconds}})
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".accuracy": silhouette}})

# Stop the Spark session
spark.stop()
```

Appendix XII – Gaussian Mixture Model Clustering Code

```
import sys

# Get the arguments
args = sys.argv[1:]

arguments = {}
for arg in args:
    key, value = arg.split("=")
    arguments[key] = value

# Parse the arguments
MONGO_HOST_PORT = arguments["MONGO_HOST_PORT"]
MINIO_HOST_PORT = arguments["MINIO_HOST_PORT"]
MINIO_USER = arguments["MINIO_USER"]
MINIO_PASS = arguments["MINIO_PASS"]
MINIO_BUCKET = arguments["MINIO_BUCKET"]
DATASET_PATH = arguments["DATASET_PATH"]
RESULT_PATH = arguments["RESULT_PATH"]
FEATURES = arguments["FEATURES"]
K = int(arguments["K"])
MAX_ITER = int(arguments["MAX_ITER"])
MONGO_DATABASE = arguments["MONGO_DATABASE"]
CLUSTER_LABEL = arguments["CLUSTER_LABEL"]

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import GaussianMixture
from pyspark.ml.evaluation import ClusteringEvaluator
from pymongo import MongoClient
from minio import Minio
import pandas as pd
from io import BytesIO
import time

""" Minio Initializations """
minio_host = MINIO_HOST_PORT
client = Minio(
    minio_host,
    access_key=MINIO_USER,
    secret_key=MINIO_PASS,
    secure=False
)
```

```
response = client.get_object(MINIO_BUCKET, DATASET_PATH)

# Load the data into a Pandas DataFrame
df = pd.read_csv(BytesIO(response.data))

""" Spark Code """
# Initialize a Spark session
spark = SparkSession.builder \
    .appName("Gaussian Mixture Application")  \
    .master("spark://spark-master:7077")   \
    .getOrCreate()

# Create an RDD from the Pandas DataFrame
df = spark.createDataFrame(df)

# Print the DataFrame schema
df.printSchema()

# Print the first 5 rows
df.show(5)

# Clustering Example
feature_columns = FEATURES.split(",")
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_df = assembler.transform(df)

# Train a GaussianMixture model
# gmm = GaussianMixture().setK(K).setMaxIter(MAX_ITER).setSeed(1) # Seed for reproducibility
gmm = GaussianMixture().setK(K).setMaxIter(MAX_ITER)
start_time = time.time()    # Start the timer
model = gmm.fit(assembled_df)
total_seconds = time.time() - start_time    # Calculate the time taken to cluster

# Make predictions
predictions = model.transform(assembled_df)

# Evaluate clustering by computing Silhouette Score with the squared Euclidean distance
evaluator = ClusteringEvaluator(metricName="silhouette", distanceMeasure="squaredEuclidean")
silhouette = evaluator.evaluate(predictions)
print(f"Silhouette with squared euclidean distance = {silhouette}")

# Show the results (excluding the 'features' column)
columns_to_show = [col for col in predictions.columns if col != 'features']
predictions.select(columns_to_show).show()

# Remove the 'features' column from the predictions DataFrame
predictions = predictions.drop('features')

# Convert predictions RDD back to Pandas DataFrame
predictions = predictions.toPandas()

# Write the resulting DataFrame with predictions to a new CSV file in Minio
csv_bytes = predictions.to_csv().encode('utf-8')
csv_buffer = BytesIO(csv_bytes)

client.put_object(MINIO_BUCKET,
                RESULT_PATH,
                data=csv_buffer,
                length=len(csv_bytes),
                content_type='application/csv')

# Update MongoDB with the clustering results
mongoClient = MongoClient('mongodb://'+MONGO_HOST_PORT+'/')
db = mongoClient[MONGO_DATABASE]
collection = db['users']
# Update the user's clustering results with the time taken to cluster and the accuracy
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".time_taken": total_seconds}})
collection.update_one({"username": MINIO_BUCKET}, {"$set": {"clustering_results."+CLUSTER_LABEL+".accuracy": silhouette}})

# Stop the Spark session
spark.stop()
```

Appendix XIII – Server Deployment Version

```
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
blinker==1.8.2
certifi==2024.7.4
cffi==1.17.0
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
dnspython==2.6.1
```

```
Flask==3.0.3
idna==3.8
itsdangerous==2.2.0
Jinja2==3.1.4
joblib==1.4.2
MarkupSafe==2.1.5
minio==7.2.7
numpy==2.1.1
pandas==2.2.2
pycparser==2.22
pycryptodome==3.20.0
pymongo==4.8.0
python-dateutil==2.9.0.post0
pytz==2024.1
requests==2.32.3
scikit-learn==1.5.1
scipy==1.14.1
six==1.16.0
threadpoolctl==3.5.0
typing_extensions==4.12.2
tzdata==2024.1
urllib3==2.2.2
Werkzeug==3.0.3
```

Appendix XIV – Engines Deployment Version

```
Docker Engine: 20.10.24
Spark Engine: 3.5.2
```

Appendix XV – PySpark Libraries Deployment Version

```
numpy==2.1.1
pandas==2.2.2
minio==7.2.7
pymongo==4.8.0
```

Appendix XVI – Data Management Databases Deployment Version

```
MinIO: RELEASE.2024-08-03T04-33-23Z
Mongo: 7.0.9
```

Appendix XVII – User Interface Deployment Version

```
HTML: HTML5
Javascript: ES2023
CSS: CSS3
Bootstrap: 5.0.2
jQuery: 1.11.0
Jinja2: 3.1.4
```

Appendix XVIII – Docker Images Deployment Version

```
EverCluster Server Component: sonem/cluster-flask-app:1.0.0
Spark Node: sonem/cluster-apache-spark:3.5.2-v4
MinIO: quay.io/minio/minio:RELEASE.2024-08-03T04-33-23Z
MongoDB: mongo:7.0.9
```

Appendix XIX – Classification Versions

```
joblib==1.4.2
numpy==2.1.1
pandas==2.2.2
python-dateutil==2.9.0.post0
pytz==2024.1
scikit-learn==1.5.1
scipy==1.14.1
six==1.16.0
threadpoolctl==3.5.0
tzdata==2024.1
```

Appendix XX

```
version: "3.3"
services:
  # Spark Master
  spark-master:
    # build:
    #   context: ./spark-code/.
    image: sonem/cluster-apache-spark:3.5.2-v4
    container_name: master
    ports:
      - "9090:8080"
      - "7077:7077"
      - "6066:6066"
    # volumes:
    #   - ./spark-code/apps:/opt/spark-apps
    #   - ./spark-code/data:/opt/spark-data
    environment:
      - SPARK_LOCAL_IP=spark-master
      - SPARK_WORKLOAD=master

  # Spark Worker A
  spark-worker-a:
    # build:
    #   context: ./spark-code/.
    image: sonem/cluster-apache-spark:3.5.2-v4
    container_name: worker-a
    ports:
      - "9091:8080"
      - "7001:7000"
    depends_on:
      - spark-master
    environment:
      - SPARK_MASTER=spark://spark-master:7077
      - SPARK_WORKER_CORES=1
      - SPARK_WORKER_MEMORY=1G
      - SPARK_DRIVER_MEMORY=1G
      - SPARK_EXECUTOR_MEMORY=1G
      - SPARK_WORKLOAD=worker
      - SPARK_LOCAL_IP=spark-worker-a
    # volumes:
    #   - ./spark-code/apps:/opt/spark-apps
    #   - ./spark-code/data:/opt/spark-data

  # Spark Worker B
  spark-worker-b:
    # build:
    #   context: ./spark-code/.
    image: sonem/cluster-apache-spark:3.5.2-v4
    container_name: worker-b
```

```
   ports:
    - "9092:8080"
    - "7002:7000"
   depends_on:
    - spark-master
   environment:
    - SPARK_MASTER=spark://spark-master:7077
    - SPARK_WORKER_CORES=1
    - SPARK_WORKER_MEMORY=1G
    - SPARK_DRIVER_MEMORY=1G
    - SPARK_EXECUTOR_MEMORY=1G
    - SPARK_WORKLOAD=worker
    - SPARK_LOCAL_IP=spark-worker-b
   # volumes:
   #    - ./spark-code/apps:/opt/spark-apps
   #    - ./spark-code/data:/opt/spark-data

  # Spark Worker C
  spark-worker-c:
   # build:
   #   context: ./spark-code/.
   image: sonem/cluster-apache-spark:3.5.2-v4
   container_name: worker-c
   ports:
    - "9093:8080"
    - "7003:7000"
   depends_on:
    - spark-master
   environment:
    - SPARK_MASTER=spark://spark-master:7077
    - SPARK_WORKER_CORES=1
    - SPARK_WORKER_MEMORY=1G
    - SPARK_DRIVER_MEMORY=1G
    - SPARK_EXECUTOR_MEMORY=1G
    - SPARK_WORKLOAD=worker
    - SPARK_LOCAL_IP=spark-worker-c
   # volumes:
   #    - ./spark-code/apps:/opt/spark-apps
   #    - ./spark-code/data:/opt/spark-data

  # Spark Worker D
  spark-worker-d:
   # build:
   #   context: ./spark-code/.
   image: sonem/cluster-apache-spark:3.5.2-v4
   container_name: worker-d
   ports:
    - "9094:8080"
    - "7004:7000"
   depends_on:
    - spark-master
   environment:
    - SPARK_MASTER=spark://spark-master:7077
    - SPARK_WORKER_CORES=1
    - SPARK_WORKER_MEMORY=1G
    - SPARK_DRIVER_MEMORY=1G
    - SPARK_EXECUTOR_MEMORY=1G
    - SPARK_WORKLOAD=worker
    - SPARK_LOCAL_IP=spark-worker-d
   # volumes:
   #    - ./spark-code/apps:/opt/spark-apps
   #    - ./spark-code/data:/opt/spark-data

  # Spark Worker E
  spark-worker-e:
   # build:
   #   context: ./spark-code/.
   image: sonem/cluster-apache-spark:3.5.2-v4
   container_name: worker-e
   ports:
    - "9095:8080"
    - "7005:7000"
   depends_on:
    - spark-master
   environment:
    - SPARK_MASTER=spark://spark-master:7077
    - SPARK_WORKER_CORES=1
    - SPARK_WORKER_MEMORY=1G
    - SPARK_DRIVER_MEMORY=1G
    - SPARK_EXECUTOR_MEMORY=1G
    - SPARK_WORKLOAD=worker
    - SPARK_LOCAL_IP=spark-worker-e
   # volumes:
   #   - ./spark-code/apps:/opt/spark-apps
   #   - ./spark-code/data:/opt/spark-data

  # Mongo DBMS
  mongo:
   image: mongo:7.0.9
   container_name: mongo-dbms
   # volumes:
   #  - ./data:/data/db
   ports:
    - "27017:27017"

  # MinIO Storage
  minio:
   image: quay.io/minio/minio:RELEASE.2024-08-03T04-33-23Z
   container_name: minio-storage
   # volumes:
   #  - ./data:/data
   ports:
    - 9000:9000
    - 9001:9001
   environment:
    MINIO_ROOT_USER: 'admin12345'
    MINIO_ROOT_PASSWORD: 'admin12345'
    MINIO_ADDRESS: ':9000'
    MINIO_CONSOLE_ADDRESS: ':9001'
   command: minio server /data

  # Main Flask Application
  flask-app:
   # build:
   #   context: ./flask-server/.
   image: sonem/cluster-flask-app:1.0.0
   container_name: server
   ports:
    - "5000:5000"
   depends_on:
    - mongo
    - minio
    - spark-master
   # volumes:
   #  - ./flask-server:/app
   environment:
    APP_HOST: '0.0.0.0'
    APP_PORT: 5000
    MONGO_HOST: mongo
    MONGO_PORT: 27017
    MONGO_DATABASE: "EverCluster"
    MINIO_HOST: minio
    MINIO_PORT: 9000
    MINIO_SERVER_PORT: 9001
    MINIO_USER: "admin12345"
    MINIO_PASS: "admin12345"
    SPARK_HOST: spark-master
    SPARK_PORT: 6066
```

Appendix XXI – Breast Cancer Dataset / Example Records

Feature	Record 1	Record 2	Record 3
id	842302	842517	858477
diagnosis	M	M	B
radius_mean	17.99	20.57	8.618
texture_mean	10.38	17.77	11.79
perimeter_mean	122.8	132.9	54.34
area_mean	1001	1326	224.5
smoothness_mean	0.1184	0.08474	0.09752
compactness_mean	0.2776	0.07864	0.05272
concavity_mean	0.3001	0.0869	0.02061
concave_points_mean	0.1471	0.07017	0.007799

symmetry_mean	0.2419	0.1812	0.1683
fractal_dimension_mean	0.07871	0.05667	0.07187
radius_se	1.095	0.5435	0.1559
texture_se	0.9053	0.7339	0.5796
perimeter_se	8.589	3.398	1.046
area_se	153.4	74.08	8.322
smoothness_se	0.006399	0.005225	0.01011
compactness_se	0.04904	0.01308	0.01055
concavity_se	0.05373	0.0186	0.01981
concave_points_se	0.01587	0.0134	0.005742
symmetry_se	0.03003	0.01389	0.0209
fractal_dimension_se	0.006193	0.003532	0.002788
radius_worst	25.38	24.99	9.507
texture_worst	17.33	23.41	15.4
perimeter_worst	184.6	158.8	59.9
area_worst	2019	1956	274.9
smoothness_worst	0.1622	0.1238	0.1733
compactness_worst	0.6656	0.1866	0.1239
concavity_worst	0.7119	0.2416	0.1168
concave_points_worst	0.2654	0.186	0.04419
symmetry_worst	0.4601	0.275	0.322
fractal_dimension_worst	0.1189	0.08902	0.09026

Appendix XXII – Covid 19 Cases Dataset / Example Records

Feature	Record 1	Record 2	Record 3
Country	Afghanistan	Greece	Lithuania
Othernames	Afghanistan	Greece	Lithuania
ISO3166-1alpha-3CODE	AFG	GRC	LTU
Population	40462186	10333930	2655811
Continent	Asia	Europe	Europe
TotalCases	177827	3077711	1030966
TotalDeaths	7671	27684	8907
TotCases//1Mpop	4395	297826	388193
TotDeaths/1Mpop	190	2679	3354
Deathpercentage	4.313743132	0.899499661	0.863947017

Appendix XXIII – Credit Cards Dataset / Example Records

Feature	Record 1	Record 2	Record 3
Sl_No	1	186	533
CustomerKey	87073	98969	47848
Avg_Credit_Limit	100000	18000	48000
Total_Credit_Cards	2	3	4
Total_visits_bank	1	0	3
Total_visits_online	1	4	2
Total_calls_made	0	4	2

Appendix XXIV – Iris Dataset Example / Records

Feature	Record 1	Record 2	Record 3
sepal_length	5.1	5.5	5.6
sepal_width	3.5	2.3	2.8
petal_length	1.4	4.0	4.9
petal_width	0.2	1.3	2.0
species	setosa	versicolor	virginica

Appendix XXV – Mall Customers Dataset / Example Records

Feature	Record 1	Record 2	Record 3
CustomerID	4	130	159
Gender	Female	Male	Male
Age	23	38	34
AnnualIncome(k$)	16	71	78
SpendingScore(1-100)	77	75	1

Appendix XXVI – Synthetic Dataset Extraction

```
# /opt/spark/bin/spark-submit --master spark://spark-master:7077 /opt/spark-apps/clustering_datasets.py WORKERS=1 DATASET_NUMBER_DONE=0
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans, BisectingKMeans, GaussianMixture
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import rand, randn, expr, min, max, col
import time, sys, random

# Get the WORKERS from the arguments
args = sys.argv[1:]
arguments = {}
for arg in args:
    key, value = arg.split("=")
    arguments[key] = value
WORKERS = int(arguments["WORKERS"])
DATASET_NUMBER_DONE = int(arguments["DATASET_NUMBER_DONE"])

# Workaround to execute the job many times and collect the results
if DATASET_NUMBER_DONE == 0:
    file_path = "/opt/spark-data/"+"workers-"+str(WORKERS)+".csv"
    f = open(file_path, "w")
    f.write("maxIteration,Clusters,Workers,ByteSize,RecordCount,FeaturesCount,ExecutionSpeed,Accuracy,Algorithm\n")
    f.close()

# Initialize Spark session
spark = SparkSession.builder.appName("Clustering Datasets Creator").getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

def generate_dataset(num_rows, num_features):
    df = spark.range(0, num_rows)

    for i in range(num_features):
        # Choose a random distribution type
        distribution_type = random.choice([0, 1, 2])

        if distribution_type == 0:
            # Uniform distribution
            df = df.withColumn(f'feature_{i}', rand())
        elif distribution_type == 1:
            # Normal distribution
            df = df.withColumn(f'feature_{i}', randn())
        else:
            # Exponential distribution (by taking the log of a uniform distribution)
            df = df.withColumn(f'feature_{i}', -expr("log(rand())"))

        # Normalize the column to be between 0 and 1
        col_name = f'feature_{i}'
        min_val = df.agg(min(col(col_name))).first()[0]
        max_val = df.agg(max(col(col_name))).first()[0]
        df = df.withColumn(col_name, (col(col_name) - min_val) / (max_val - min_val))

    return df
```

```
def run_clustering_algorithms(dataset, k, maxIter):
    print("APPLICATION LOG: Dataset is being processed in kmeans, bisecting kmeans, and gaussian mixture algorithms")
    # dataset.show()
    results = []

    # Assemble features into a single column
    feature_columns = [col for col in dataset.columns if col != "id"]  # Assuming 'id' is not a feature
    assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
    dataset = assembler.transform(dataset)

    # KMeans v3
    kmeans = KMeans().setK(k).setMaxIter(maxIter).setFeaturesCol("features")
    start_time = time.time()
    kmeans_model = kmeans.fit(dataset)
    kmeans_time = time.time() - start_time
    kmeans_wssse = kmeans_model.summary.trainingCost
    results.append(('KMeans', kmeans_time, kmeans_wssse))

    # Bisecting KMeans
    bisecting_kmeans = BisectingKMeans().setK(k).setMaxIter(maxIter)
    start_time = time.time()
    bisecting_kmeans_model = bisecting_kmeans.fit(dataset)
    bisecting_kmeans_time = time.time() - start_time
    bisecting_kmeans_wssse = bisecting_kmeans_model.computeCost(dataset)
    results.append(('BisectingKMeans', bisecting_kmeans_time, bisecting_kmeans_wssse))

    # Gaussian Mixture
    gaussian_mixture = GaussianMixture().setK(k).setMaxIter(maxIter)
    start_time = time.time()
    gaussian_mixture_model = gaussian_mixture.fit(dataset)
    gaussian_mixture_time = time.time() - start_time
    gaussian_mixture_logLikelihood = gaussian_mixture_model.summary.logLikelihood
    results.append(('GaussianMixture', gaussian_mixture_time, gaussian_mixture_logLikelihood))

    return results

# loop for multiple datasets
print("APPLICATION LOG: Loop for multiple datasets and configurations (Synthetic Generation & Clustering)")
training_data = []

dataset_to_check = 1
# for num_rows in [4000, 1000, 500, 100, 25, 10]:
for num_rows in [10, 25, 100, 500, 1000, 4000]:
    # for num_features in [25, 10, 5, 3]:
    for num_features in [3, 5, 10, 25]:
        # Scip if the dataset has already been processed
        if dataset_to_check <= DATASET_NUMBER_DONE - 9:
            print("APPLICATION LOG: Datasets already processed. Skipping 9...")
            dataset_to_check += 9
            continue
        dataset = generate_dataset(num_rows, num_features)
        # for k in [5, 3, 2]:
        for k in [2, 3, 5]:
            # for maxIter in [50, 25, 10]:
            for maxIter in [10, 25, 50]:
                print(f"APPLICATION LOG: ---> dataset with {num_rows} rows, {num_features} features, {k} clusters, and {maxIter} maxIter (Workers: {WORKERS})")

                # Checking if this dataset has already been processed
                if dataset_to_check <= DATASET_NUMBER_DONE:
                    print("APPLICATION LOG: Dataset already processed. Skipping...")
                    dataset_to_check += 1
                    continue
                dataset_to_check += 1

                results = run_clustering_algorithms(dataset, k=k, maxIter=maxIter)
                byte_size = dataset.count() * len(dataset.columns) * 8  # Rough estimate
                record_count = dataset.count()
                for result in results:
                    file_path = "/opt/spark-data/"+"workers-"+str(WORKERS)+".csv"
                    f = open(file_path, "a")
                    f.write(f"{maxIter},{k},{WORKERS},{byte_size},{record_count},{num_features},{result[1]},{result[2]},{result[0]}\n")
                    f.close()
                # Increment the dataset number to avoid overwriting the results
                print(f"APPLICATION LOG: DATASETS DONE: {DATASET_NUMBER_DONE + 1}")
                DATASET_NUMBER_DONE += 1
print("APPLICATION LOG: End of Clustering (Synthetic Generation & Clustering)")
```

Appendix XXVII – CSV Experimentation File Example

Feature	Record 1	Record 2	Record 3
maxIteration	10	25	50
Clusters	2	3	5
Workers	1	1	1
ByteSize	320	2200	16000
RecordCount	10	25	500
FeaturesCount	3	10	3
ExecutionSpeed	4.0894	0.819	2.8598
Accuracy	2.4466	2.446	1.997
Algorithm	KMeans	BisectingKMeans	GaussianMixture

Appendix XXVIII – Random Forest Model Creation

```
# pip install scikit-learn
# pip install pandas
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Load the datasets
df_training_1 = pd.read_csv("./workers-1.csv")
df_training_2 = pd.read_csv("./workers-2.csv")
df_training_3 = pd.read_csv("./workers-3.csv")
df_training_4 = pd.read_csv("./workers-4.csv")
df_training_5 = pd.read_csv("./workers-5.csv")

# Concatenate the datasets
df_training = pd.concat([df_training_1, df_training_2, df_training_3, df_training_4, df_training_5])
# print(df_training)

# Create empty Dataset to store the values of the best algorithm per three rows
df_clean_training = pd.DataFrame(columns=["maxIteration", "Clusters", "Workers", "ByteSize", "RecordCount", "FeaturesCount"])
# Create empty Dataset to store the best algorithm per three rows
df_best_execSpeed = pd.DataFrame(columns=["BestBySpeed"])
df_best_execAccuracy = pd.DataFrame(columns=["BestByAccuracy"])

# Loop the df_training per three rows and print the algorithm feature
for i in range(0, len(df_training), 3):
    alg_1 = df_training.iloc[i]["Algorithm"]
    alg_2 = df_training.iloc[i+1]["Algorithm"]
    alg_3 = df_training.iloc[i+2]["Algorithm"]

    # Find the algorithm with the best speed
    if df_training.iloc[i]["ExecutionSpeed"] < df_training.iloc[i+1]["ExecutionSpeed"] and df_training.iloc[i]["ExecutionSpeed"] < df_training.iloc[i+2]["ExecutionSpeed"]:
        best_speed = alg_1
    elif df_training.iloc[i+1]["ExecutionSpeed"] < df_training.iloc[i]["ExecutionSpeed"] and df_training.iloc[i+1]["ExecutionSpeed"] < df_training.iloc[i+2]["ExecutionSpeed"]:
        best_speed = alg_2
    else:
        best_speed = alg_3

    # Find the algorithm with the best accuracy
    if df_training.iloc[i]["Accuracy"] < df_training.iloc[i+1]["Accuracy"] and df_training.iloc[i]["Accuracy"] < df_training.iloc[i+2]["Accuracy"]:
        best_accuracy = alg_1
    elif df_training.iloc[i+1]["Accuracy"] < df_training.iloc[i]["Accuracy"] and df_training.iloc[i+1]["Accuracy"] < df_training.iloc[i+2]["Accuracy"]:
        best_accuracy = alg_2
    else:
        best_accuracy = alg_3

    # Append the best algorithm to the df_best_execSpeed and df_best_execAccuracy
    # df_best_execSpeed = df_best_execSpeed.append({"BestBySpeed": best_speed}, ignore_index=True)
    df_best_execSpeed.loc[len(df_best_execSpeed.index)] = [best_speed]
    # df_best_execAccuracy = df_best_execAccuracy.append({"BestByAccuracy": best_accuracy}, ignore_index=True)
    df_best_execAccuracy.loc[len(df_best_execAccuracy.index)] = [best_accuracy]

    # Append the features of the three algorithms to the df_clean_training (They are the same for the three algorithms)
    # df_clean_training = df_clean_training.append(df_training.iloc[i][["maxIteration", "Clusters", "Workers", "ByteSize", "RecordCount", "FeaturesCount"]], ignore_index=True)
    df_clean_training.loc[len(df_clean_training.index)] = df_training.iloc[i][["maxIteration", "Clusters", "Workers", "ByteSize", "RecordCount", "FeaturesCount"]]



# Prepare features and labels
X = df_clean_training[["maxIteration", "Clusters", "Workers", "ByteSize", "RecordCount", "FeaturesCount"]]
y_speed = df_best_execSpeed["BestBySpeed"]
y_accuracy = df_best_execAccuracy["BestByAccuracy"]


# Split the data into training and testing sets
X_train, X_test, y_speed_train, y_speed_test = train_test_split(X, y_speed, test_size=0.2, random_state=42)
_, _, y_accuracy_train, y_accuracy_test = train_test_split(X, y_accuracy, test_size=0.2, random_state=42)   # The same random state so the split is the same
```

```
# Train the Speed Classifier
speed_classifier = RandomForestClassifier()
speed_classifier.fit(X_train, y_speed_train)

# Train the Accuracy Classifier
accuracy_classifier = RandomForestClassifier()
accuracy_classifier.fit(X_train, y_accuracy_train)

# Evaluate Speed Classifier
y_speed_pred = speed_classifier.predict(X_test)
speed_accuracy = accuracy_score(y_speed_test, y_speed_pred)
print(f"Speed Classifier Accuracy: {speed_accuracy:.2f}")

# Evaluate Accuracy Classifier
y_accuracy_pred = accuracy_classifier.predict(X_test)
accuracy_accuracy = accuracy_score(y_accuracy_test, y_accuracy_pred)
print(f"Accuracy Classifier Accuracy: {accuracy_accuracy:.2f}")

""" Saving of the Models """
# Save the models
joblib.dump(speed_classifier, "speed_classifier.pkl")       # Save the model to PKL file (serialized Python object)
joblib.dump(accuracy_classifier, "accuracy_classifier.pkl")  # Save the model to PKL file (serialized Python object)

# To load the models
# speed_classifier = joblib.load("speed_classifier.pkl")
# accuracy_classifier = joblib.load("accuracy_classifier.pkl")

"""
# Example new data
new_data = pd.DataFrame({
    "maxIteration": [25],
    "Clusters": [3],
    "Workers": [6],
    "ByteSize": [5000],
    "RecordCount": [1000],
    "FeaturesCount": [8]
})

# Predict the best algorithm for speed
best_by_speed = speed_classifier.predict(new_data)
print(f"Recommended Algorithm for Speed: {best_by_speed[0]}")

# Predict the best algorithm for accuracy
best_by_accuracy = accuracy_classifier.predict(new_data)
print(f"Recommended Algorithm for Accuracy: {best_by_accuracy[0]}")
"""
```

Appendix XXIX – Experimentation Normalization

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Function to scale all columns, including strings
def scale_csv(input_file, output_file):
    df = pd.read_csv(input_file)

    # Create a copy of the DataFrame to store encoded string columns
    encoded_df = df.copy()

    # Initialize label encoder
    label_encoders = {}

    # Apply Label Encoding to string columns
    for col in encoded_df.select_dtypes(include=['object']).columns:
        le = LabelEncoder()
        encoded_df[col] = le.fit_transform(encoded_df[col])
        label_encoders[col] = le

    # Apply MinMax scaling to all columns
    scaler = MinMaxScaler()
    scaled_df = pd.DataFrame(scaler.fit_transform(encoded_df), columns=encoded_df.columns)

    # Save the scaled data to a new CSV file
    scaled_df.to_csv(output_file, index=False)
    print(f"File saved: {output_file}")
```

```
# Scale and save each CSV
scale_csv('Breast_Cancer_Dataset.csv', 'Breast_Cancer_Dataset_scaled_all.csv')
scale_csv('COVID-19_Coronavirus.csv', 'COVID-19_Coronavirus_scaled_all.csv')
scale_csv('Credit_Card_Customer_Data.csv', 'Credit_Card_Customer_Data_scaled_all.csv')
scale_csv('Iris_dataset.csv', 'Iris_dataset_scaled_all.csv')
scale_csv('Mall_Customer_Segmentation_Data.csv', 'Mall_Customer_Segmentation_Data_scaled_all.csv')
```

Appendix XXX – Normalized Breast Cancer Dataset / Example Records

Feature	Record 1	Record 2	Record 3
id	0.000914	0.000936	0.000938
diagnosis	1.0	0.0	0.0
radius_mean	0.521037	0.382365	0.167021
texture_mean	0.022658	0.342238	0.354413
perimeter_mean	0.545988	0.390505	0.171722
area_mean	0.363732	0.238430	0.080890
smoothness_mean	0.593752	0.462850	0.537780
compactness_mean	0.792037	0.416906	0.340224
concavity_mean	0.703139	0.398313	0.151733
concave_points_mean	0.731113	0.438121	0.152485
symmetry_mean	0.686363	0.401515	0.435353
fractal_dimension_mean	0.605518	0.271272	0.586773
radius_se	0.356147	0.132283	0.080427
texture_se	0.120469	0.133530	0.331462
perimeter_se	0.369033	0.138953	0.060500
area_se	0.273811	0.075342	0.024482
smoothness_se	0.159295	0.256212	0.345616
compactness_se	0.351398	0.262099	0.264502
concavity_se	0.135681	0.122904	0.115934
concave_points_se	0.300625	0.350634	0.292479
symmetry_se	0.311645	0.099876	0.210896
fractal_dimension_se	0.183042	0.090695	0.203736
radius_worst	0.620775	0.366417	0.114905
texture_worst	0.141524	0.325426	0.285980
perimeter_worst	0.668310	0.364012	0.110613
area_worst	0.450697	0.206399	0.046500
smoothness_worst	0.601135	0.554249	0.388496
compactness_worst	0.619291	0.381300	0.172318
concavity_worst	0.568610	0.415575	0.103434
concave_points_worst	0.912027	0.726804	0.210859
symmetry_worst	0.598462	0.250147	0.161245
fractal_dimension_worst	0.418863	0.179063	0.231011

Appendix XXXI – Normalized Covid 19 Cases Dataset / Example Records

Feature	Record 1	Record 2	Record 3
Country	0.026785	0.575892	0.683035
Othernames	0.044642	0.580357	0.674107
ISO3166-1alpha-3CODE	0.040178	0.651785	0.723214
Population	6.846482	0.000196	1.211680
Continent	0.600000	0.0	1.0
TotalCases	9.154553	0.000450	4.937740
TotalDeaths	0.000133	0.000185	5.951070
TotCases//1Mpop	0.108346	0.186395	0.318275
TotDeaths/1Mpop	0.216194	0.104677	0.052338
Deathpercentage	0.099256	0.027925	0.008177

Appendix XXXII – Normalized Credit Cards Dataset / Example Records

Feature	Record 1	Record 2	Record 3
Sl_No	0.004552	0.127465	0.309559
CustomerKey	0.330002	0.772528	0.248955
Avg_Credit_Limit	0.137055	0.030456	0.071065
Total_Credit_Cards	0.444444	0.222222	0.111111
Total_visits_bank	0.2	0.4	0.4
Total_visits_online	0.066666	0.266666	0.333333
Total_calls_made	0.4	0.5	0.5

Appendix XXXIII – Normalized Iris Dataset Example / Records

Feature	Record 1	Record 2	Record 3
sepal_length	0.194444	0.222222	0.166666
sepal_width	0.666666	0.208333	0.208333
petal_length	0.067796	0.338983	0.593220
petal_width	0.041666	0.416666	0.666666
species	0.0	0.5	1.0

Appendix XXXIV – Normalized Mall Customers Dataset / Example Records

Feature	Record 1	Record 2	Record 3
CustomerID	0.030150	0.402010	0.396984
Gender	0.0	1.0	0.0
Age	0.326923	0.750000	0.596153
AnnualIncome(k$)	0.024590	0.319672	0.319672
SpendingScore(1-100)	0.051020	0.510204	0.418367

Appendix XXXV – Experimentation Results for labels DS*a*c2m10w2

Labels: DS*a*c2m10w2				
Speed Recommendation Results for labels ending with: c2m10w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	5.04	8.54	4.81	Bisecting K-means
DS2	5.65	7.72	4.65	Bisecting K-means
DS3	5.36	8.09	4.94	Bisecting K-means
DS4	5.06	7.9	4.66	Bisecting K-means
DS5	4.96	7.8	4.85	Bisecting K-means
Accuracy Recommendation Results for labels ending with: c2m10w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	52.45	70.11	K-means
DS2	50.06	10.55	50.06	K-means
DS3	57.91	27.79	57.91	Bisecting K-means
DS4	81.01	62.9	81.01	K-means
DS5	65.55	24.35	65.55	K-means

Appendix XXXVI – Experimentation Results for labels DS*a*c2m20w2

Labels: DS*a*c2m20w2				
Speed Recommendation Results for labels ending with: c2m20w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	5.22	9.35	5.76	Bisecting K-means
DS2	5.69	8.68	5.71	K-means
DS3	5.33	8.77	5.76	K-means
DS4	4.94	8.88	5.71	K-means
DS5	4.89	8.89	5.72	K-means
Accuracy Recommendation Results for labels ending with: c2m20w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	58.62	70.11	Bisecting K-means
DS2	50.06	10.55	50.06	Bisecting K-means
DS3	57.91	34.21	57.91	Bisecting K-means
DS4	81.01	81.01	81.01	K-means
DS5	65.55	81.01	65.55	K-means

Appendix XXXVII – Experimentation Results for labels DS*a*c2m50w2

Labels: DS*a*c2m50w2				
Speed Recommendation Results for labels ending with: c2m50w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	5.28	10.2	7.71	K-means
DS2	5.59	8.27	7.68	K-means
DS3	5.14	8.95	7.75	K-means
DS4	4.95	8.85	7.69	K-means
DS5	4.89	11.09	7.96	K-means
Accuracy Recommendation Results for labels ending with: c2m50w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	61.3	70.11	Bisecting K-means
DS2	50.06	10.55	50.06	Bisecting K-means
DS3	57.91	34.21	57.91	Bisecting K-means
DS4	81.01	81.01	81.01	K-means
DS5	65.55	20.15	65.55	K-means

Appendix XXXVIII – Experimentation Results for labels DS*a*c3m10w2

Labels: DS*a*c3m10w2				
Speed Recommendation Results for labels ending with: c3m10w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	6.16	8.16	5.65	Gaussian Mixture
DS2	5.89	7.78	5.53	Gaussian Mixture
DS3	5.12	7.94	5.8	Gaussian Mixture
DS4	5.07	7.99	5.73	Gaussian Mixture
DS5	6.12	7.78	5.84	Gaussian Mixture
Accuracy Recommendation Results for labels ending with: c3m10w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	37.49	41.05	K-means
DS2	43.66	0.92	42.65	Bisecting K-means
DS3	66.2	33.56	66.2	K-means
DS4	81.49	41.19	81.49	K-means
DS5	55.26	17.88	52.64	K-means

Appendix XXXIX – Experimentation Results for labels DS*a*c3m20w2

Labels: DS*a*c3m20w2				
Speed Recommendation Results for labels ending with: c3m20w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	6.15	9.31	7.02	Gaussian Mixture
DS2	5.68	8.93	7.1	K-means
DS3	5.12	9.07	7.05	K-means
DS4	4.86	8.83	6.95	K-means
DS5	6.18	8.66	7.12	K-means
Accuracy Recommendation Results for labels ending with: c3m20w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	39.39	41.05	K-means
DS2	43.66	0.61	42.65	Bisecting K-means
DS3	66.2	32.41	66.2	K-means
DS4	81.49	42	81.49	K-means
DS5	55.26	18.01	52.64	K-means

Appendix XL – Experimentation Results for labels DS*a*c3m50w2

Labels: DS*a*c3m50w2				
Speed Recommendation Results for labels ending with: c3m50w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	5.95	10.61	10.57	K-means
DS2	5.84	8.74	10.72	K-means
DS3	5.24	8.54	10.46	K-means
DS4	5.04	9.84	10.76	K-means
DS5	6.08	8.28	10.87	K-means
Accuracy Recommendation Results for labels ending with: c3m50w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	40.35	41.05	K-means
DS2	43.66	0.61	42.65	K-means
DS3	66.2	32.41	66.2	K-means
DS4	81.49	41.71	81.49	K-means
DS5	55.26	18.01	52.64	K-means

Appendix XLI – Experimentation Results for labels DS*a*c5m10w2

Labels: DS*a*c5m10w2				
Speed Recommendation Results for labels ending with: c5m10w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	6.3	8.43	6.55	Gaussian Mixture
DS2	5.33	7.94	6.29	Gaussian Mixture
DS3	5.55	8.1	6.45	Gaussian Mixture
DS4	5.21	7.89	6.28	Gaussian Mixture
DS5	5.05	7.63	6.48	Gaussian Mixture
Accuracy Recommendation Results for labels ending with: c5m10w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.34	34.71	23.98	K-means
DS2	40.44	-0.16	35.81	Bisecting K-means
DS3	44.8	16.45	44.77	K-means
DS4	62.61	43.61	60.81	K-means
DS5	60.2	37.31	40.73	K-means

Appendix XLII – Experimentation Results for labels DS*a*c5m20w2

Labels: DS*a*c5m20w2				
Speed Recommendation Results for labels ending with: c5m20w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	6.69	9.34	8.39	Gaussian Mixture
DS2	5.5	8.78	8.2	K-means
DS3	5.51	8.78	8.31	K-means
DS4	5.3	8.63	8.27	K-means
DS5	5.31	8.89	8.39	K-means
Accuracy Recommendation Results for labels ending with: c5m20w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.65	35.18	23.98	K-means
DS2	40.44	0.12	35.81	Bisecting K-means
DS3	44.8	16.83	44.77	K-means
DS4	62.61	28.27	60.81	K-means
DS5	60.2	1.2	40.73	K-means

Appendix XLIII – Experimentation Results for labels DS*a*c5m50w2

Labels: DS*a*c5m50w2				
Speed Recommendation Results for labels ending with: c5m50w2 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	6.77	11.01	13.57	K-means
DS2	5.47	9.01	13.08	K-means
DS3	5.6	11.5	13.43	K-means
DS4	5.01	11.18	13.32	K-means
DS5	5.48	10.13	13.1	K-means
Accuracy Recommendation Results for labels ending with: c5m50w2 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.65	35.71	23.98	K-means
DS2	40.44	0.12	35.81	Bisecting K-means
DS3	44.8	27.84	44.77	K-means
DS4	62.61	46.01	60.81	K-means
DS5	60.2	2.65	40.73	Bisecting K-means

Appendix XLIV – Experimentation Results for labels DS*a*c2m10w5

Labels: DS*a*c2m10w5				
Speed Recommendation Results for labels ending with: c2m10w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	10.16	13.81	9.3	Bisecting K-means
DS2	10.16	13.19	9.97	Bisecting K-means
DS3	9.28	12.98	7.57	Bisecting K-means
DS4	7.41	10.14	5.99	Bisecting K-means
DS5	6.37	9.8	9.91	Bisecting K-means
Accuracy Recommendation Results for labels ending with: c2m10w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	52.45	70.11	K-means
DS2	50.06	10.55	50.06	K-means
DS3	57.91	27.79	57.91	K-means
DS4	81.01	62.9	81.01	K-means
DS5	65.55	24.35	65.55	K-means

Appendix XLV – Experimentation Results for labels DS*a*c2m20w5

Labels: DS*a*c2m20w5				
Speed Recommendation Results for labels ending with: c2m20w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	9.68	16.06	11.04	Bisecting K-means
DS2	10.09	14.09	10.84	Gaussian Mixture
DS3	9.38	15.01	11.3	Bisecting K-means
DS4	7.6	9.53	8.41	K-means
DS5	8.76	11.14	9.87	K-means
Accuracy Recommendation Results for labels ending with: c2m20w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	58.62	70.11	K-means
DS2	50.06	10.55	50.06	K-means
DS3	57.91	34.21	57.91	K-means
DS4	81.01	81.01	81.01	K-means
DS5	65.55	22.84	65.55	K-means

Appendix XLVI – Experimentation Results for labels DS*a*c2m50w5

Labels: DS*a*c2m50w5				
Speed Recommendation Results for labels ending with: c2m50w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	9.53	17.05	12.91	K-means
DS2	10.69	14.33	12.91	K-means
DS3	8.36	15.27	11.84	K-means
DS4	5.13	9.74	9.93	K-means
DS5	4.88	16.55	13.03	K-means
Accuracy Recommendation Results for labels ending with: c2m50w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	70.11	61.3	70.11	K-means
DS2	50.06	10.55	50.06	K-means
DS3	57.91	34.21	57.91	K-means
DS4	81.01	81.01	81.01	K-means
DS5	65.55	20.15	65.55	K-means

Appendix XLVII – Experimentation Results for labels DS*a*c3m10w5

Labels: DS*a*c3m10w5				
Speed Recommendation Results for labels ending with: c3m10w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	12.18	13.95	11.31	Gaussian Mixture
DS2	10.7	13.08	10.99	Gaussian Mixture
DS3	9.73	10.58	9.12	Gaussian Mixture
DS4	5.72	8.43	8.59	Gaussian Mixture
DS5	6.21	12.77	9.85	Gaussian Mixture
Accuracy Recommendation Results for labels ending with: c3m10w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	37.49	41.05	K-means
DS2	43.66	0.92	42.65	K-means
DS3	66.2	33.56	66.2	K-means
DS4	81.49	41.19	81.49	K-means
DS5	55.26	17.88	52.64	K-means

Appendix XLVIII – Experimentation Results for labels DS*a*c3m20w5

Labels: DS*a*c3m20w5				
Speed Recommendation Results for labels ending with: c3m20w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	11.59	15.39	13.22	Gaussian Mixture
DS2	11.23	11.9	12.51	K-means
DS3	6.99	11.29	8.54	K-means
DS4	5.45	9.29	7.23	K-means
DS5	7.7	13.41	12.65	K-means
Accuracy Recommendation Results for labels ending with: c3m20w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	39.39	41.05	K-means
DS2	43.66	0.61	42.65	K-means
DS3	66.2	32.41	66.2	K-means
DS4	81.49	42	81.49	K-means
DS5	55.26	18.01	52.64	K-means

Appendix XLIX – Experimentation Results for labels DS*a*c3m50w5

Labels: DS*a*c3m50w5				
Speed Recommendation Results for labels ending with: c3m50w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	10.51	15.94	15.38	K-means
DS2	9.7	12.08	11.54	K-means
DS3	7.28	13.44	17.2	K-means
DS4	11.18	12.04	14.39	K-means
DS5	8.23	10.44	17.46	K-means
Accuracy Recommendation Results for labels ending with: c3m50w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	41.05	40.35	41.05	K-means
DS2	43.66	0.61	42.65	K-means
DS3	66.2	32.41	66.2	K-means
DS4	81.49	41.71	81.49	K-means
DS5	55.26	18.01	52.64	K-means

Appendix L – Experimentation Results for labels DS*a*c5m10w5

Labels: DS*a*c5m10w5				
Speed Recommendation Results for labels ending with: c5m10w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	13.24	14.62	10.97	Gaussian Mixture
DS2	9.88	10.88	12.19	Gaussian Mixture
DS3	8.17	10.08	9.85	Gaussian Mixture
DS4	6.42	8.15	6.88	Gaussian Mixture
DS5	7.1	10	12.9	Gaussian Mixture
Accuracy Recommendation Results for labels ending with: c5m10w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.34	34.71	23.98	K-means
DS2	40.44	-0.16	35.81	K-means
DS3	44.8	16.45	44.77	K-means
DS4	62.61	43.61	60.81	K-means
DS5	60.2	37.31	40.73	K-means

Appendix LI – Experimentation Results for labels DS*a*c5m20w5

Labels: DS*a*c5m20w5				
Speed Recommendation Results for labels ending with: c5m20w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	13.53	15.42	14.41	Gaussian Mixture
DS2	9.44	14.41	11.8	K-means
DS3	6	10.2	9.21	K-means
DS4	11.41	11.26	11.3	K-means
DS5	7.85	14.78	15.03	K-means
Accuracy Recommendation Results for labels ending with: c5m20w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.65	35.18	23.98	K-means
DS2	40.44	0.12	35.81	K-means
DS3	44.8	16.83	44.77	K-means
DS4	62.61	28.27	60.81	K-means
DS5	60.2	1.2	40.73	K-means

Appendix LII – Experimentation Results for labels DS*a*c5m50w5

Labels: DS*a*c5m50w5				
Speed Recommendation Results for labels ending with: c5m50w5 (in sec)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	12.83	16.64	20.77	K-means
DS2	11.35	11	19.67	K-means
DS3	11.05	17.75	17.37	K-means
DS4	7.39	16.8	17.65	K-means
DS5	9.26	16.09	17.22	K-means
Accuracy Recommendation Results for labels ending with: c5m50w5 (in %)				
Dataset	**K-means**	**Gaussian Mixture**	**Bisecting K-means**	**Recommendation**
DS1	32.65	35.71	23.98	K-means
DS2	40.44	0.12	35.81	K-means
DS3	44.8	27.84	44.77	K-means
DS4	62.61	46.01	46.01	K-means
DS5	60.2	2.65	40.73	K-means

Appendix LII – Gaussian Mixture Summary based on Nodes

Gaussian Mixture Speed Summary based on Nodes					
Worker Amount	**Best**	**Worst**	**None**	**Total**	**Percentage (Best)**
2 Nodes	0	36	9	45	00.0 %
5 Nodes	2	30	13	45	04.4 %
Total	2	66	22	90	02.2 %
Gaussian Mixture Accuracy Summary based on Nodes					
Worker Amount	**Best**	**Worst**	**None**	**Total**	**Percentage (Best)**
2 Nodes	6	39	0	45	13.3 %
5 Nodes	5	40	0	45	11.1 %
Total	11	79	0	90	12.2 %

Appendix LIII – Gaussian Mixture Summary based on Datasets

Gaussian Mixture Speed Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	0	16	2	18	00.0 %
DS2	1	13	4	18	05.5 %
DS3	0	15	3	18	00.0 %
DS4	1	11	6	18	05.5 %
DS5	0	11	7	18	00.0 %
Total	2	66	22	90	02.0 %
Gaussian Mixture Accuracy Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	6	12	0	18	33.3 %
DS2	0	18	0	18	00.0 %
DS3	0	18	0	18	00.0 %
DS4	4	14	0	18	22.2 %
DS5	1	17	0	18	05.5 %
Total	11	79	0	90	12.2 %

Appendix LIV – Gaussian Mixture Summary based Clusters Found

Gaussian Mixture Speed Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	0	28	2	30	00.0 %
3 Clusters	0	21	9	30	00.0 %
5 Clusters	2	17	11	30	06.6 %
Total	2	66	22	90	02.2 %
Gaussian Mixture Accuracy Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	5	25	0	30	16.6 %
3 Clusters	0	30	0	30	00.0 %
5 Clusters	6	24	0	30	20.0 %
Total	11	79	0	90	12.2 %

Appendix LV – Gaussian Mixture Summary based Max Iteration

Gaussian Mixture Speed Summary based on Max Iterations					
Max Iteration	Correct	Worst	None	Total	Percentage (Best)
10 Iterations	0	26	4	30	00.0 %
20 Iterations	1	27	2	30	03.3 %
50 Iterations	1	13	16	30	03.3 %
Total	2	66	22	90	02.0 %
Gaussian Mixture Accuracy Summary based on Max Iterations					
Max Iteration	Correct	Worst	None	Total	Percentage (Best)
10 Iterations	2	28	0	30	06.6 %
20 Iterations	5	25	0	30	16.6 %
50 Iterations	4	26	0	30	13.3 %
Total	11	79	0	90	12.2 %

Appendix LVI – Bisecting K-means Summary based on Nodes

Bisecting K-means Speed Summary based on Nodes					
Worker Amount	Best	Worst	None	Total	Percentage (Best)
2 Nodes	8	9	28	45	17.7 %
5 Nodes	7	14	24	45	15.5 %
Total	15	23	52	90	16.6 %
Bisecting K-means Accuracy Summary based on Nodes					
Worker Amount	Best	Worst	None	Total	Percentage (Best)
2 Nodes	23	4	18	45	51.1 %
5 Nodes	26	4	15	45	57.7 %
Total	49	8	33	90	54.4 %

Appendix LVII – Bisecting K-means Summary based on Datasets

Bisecting K-means Speed Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	5	2	11	18	27.7 %
DS2	3	5	10	18	16.6 %
DS3	3	3	12	18	16.6 %
DS4	2	6	10	18	11.1 %
DS5	2	7	9	18	11.1 %
Total	15	23	52	90	16.6 %
Bisecting K-means Accuracy Summary based on Datasets					
Dataset Name	Best	Worst	None	Total	Percentage (Best)
DS1	12	6	0	18	66.6 %
DS2	6	0	12	18	33.3 %
DS3	12	6	0	18	66.6 %
DS4	12	1	5	18	66.6 %
DS5	7	1	10	18	38.8 %
Total	49	8	33	90	54.4 %

Appendix LVIII – Bisecting K-means Summary based Clusters Found

Bisecting K-means Speed Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	9	2	19	30	30.0 %
3 Clusters	5	9	16	30	16.6 %
5 Clusters	1	12	17	30	03.3 %
Total	15	23	52	90	16.6 %
Bisecting K-means Accuracy Summary based on Clusters Found					
Clusters Found	Correct	Worst	None	Total	Percentage (Best)
2 Clusters	29	1	0	30	96.6 %
3 Clusters	20	0	10	30	66.6 %
5 Clusters	0	7	23	30	0.00 %
Total	49	8	33	90	54.4 %

Appendix LIX – Bisecting K-means Summary based Max Iteration

Bisecting K-means Speed Summary based on Max Iterations					
Max Iteration	Correct	Worst	None	Total	Percentage (Best)
10 Iterations	15	4	11	30	50.0 %
20 Iterations	0	2	28	30	00.0 %
50 Iterations	0	17	13	30	00.0 %
Total	15	23	52	90	16.6 %
Bisecting K-means Accuracy Summary based on Max Iterations					
Max Iteration	Correct	Worst	None	Total	Percentage (Best)
10 Iterations	17	2	11	30	56.6 %
20 Iterations	15	3	12	30	50.0 %
50 Iterations	17	3	10	30	56.6 %
Total	49	8	33	90	54.4 %