DRIFT DETECTION IN LEGACY SYSTEMS USING MACHINE-LEARNING TECHNIQUE

SWAPNIL HADGE

Final Thesis Report

SEPTEMBER 2023

DEDICATION

I dedicate this thesis to my wife, whose unwavering support and encouragement have been my anchor throughout this academic journey. To my parents, whose sacrifices and belief in my potential have been a constant source of encouragement. To my daughters who have always been my source of strength and inspiration.

I also dedicate this work to my mentors, professors, and advisors, whose guidance and wisdom have shaped my academic and intellectual growth. Your mentorship has been invaluable, and I am deeply grateful for your patience and dedication.

This work is dedicated to all of you with heartfelt gratitude and love.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who have contributed to the completion of this thesis. This research would not have been possible without the support, guidance, and encouragement of many individuals and organizations.

First and foremost, I would like to thank my thesis advisor, **Govind Shenoy**, for their unwavering support, expertise, and invaluable guidance throughout the research process. Your mentorship has been instrumental in shaping this work.

I am also grateful to the **faculty members of LJMU** for their insightful feedback and encouragement during my academic journey.

I am thankful to **my family** for their love, encouragement, and understanding during the ups and downs of this academic endeavour. Your unwavering belief in me has been my motivation.

To my friends and colleagues who provided valuable insights and moral support, I extend my heartfelt thanks. Your camaraderie has made this journey memorable.

Finally, I want to express my deepest gratitude to all those who believe in the significance of research and the pursuit of knowledge. Your collective commitment to advancing our understanding of Drift detection and Machine learning inspires me.

This thesis is a culmination of the efforts and support of many, and I am sincerely thankful to each and every one of you.

Swapnil Suresh Hadge
LJMU
10 Sep. 23

ABSTRACT

Legacy software systems represent a critical component of many organizations' technology stacks. However, they often lack documentation and comprehensive testing, making them susceptible to drift as the software environment changes over time. This thesis investigates the efficacy of supervised and unsupervised machine learning algorithms for identifying and quantifying drift in legacy software systems.

In this study, a thorough review of existing drift detection approaches is conducted, highlighting their strengths and limitations. Subsequently, supervised machine learning algorithms, such as Support Vector Machines and Random Forests, are employed to construct predictive models based on historical data. These models aim to proactively detect drift by learning from labeled instances of normal and anomalous system behaviour. Additionally, unsupervised machine learning techniques, including Principal Component Analysis, and clustering algorithms, are explored for their ability to discern patterns in software system data without the need for labeled samples. Experimental evaluations are conducted using real-world legacy software systems, demonstrating the comparative performance of supervised and unsupervised machine learning approaches in drift detection. The findings contribute to the growing body of knowledge in software maintenance and evolution, offering practical guidance for organizations seeking to enhance their legacy software management strategies using machine learning techniques.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

Legacy software systems serve as the bedrock of countless organizations, underpinning critical business processes, housing valuable data, and often embodying decades of domain-specific knowledge. These systems, while fundamental to business operations, are not immune to the inexorable march of time.Legacy software systems serve as the bedrock of countless organizations, underpinning critical business processes, housing valuable data, and often embodying decades of domain-specific knowledge(Albuquerque and Cruz, 2019). These systems, while fundamental to business operations, are not immune to the inexorable march of time. As user requirements evolve, technology advances, and the operational context shifts, legacy systems encounter a formidable challenge—concept drift (Yu et al., 2019).

Concept drift refers to the phenomenon where the underlying data distribution and system behavior change over time. In the context of legacy software systems, concept drift can manifest in various forms. It might entail shifts in user behavior, alterations in data patterns, modifications to the operational environment, or changes in regulatory compliance requirements. Left unaddressed, concept drift poses significant risks, including reduced system reliability, security vulnerabilities, and diminishing user satisfaction (Castellani et al., 2021).

The study aims to discuss various methods of machine learning and its practical implementation. This would provide the evaluation metrics against these algorithms to enable implementation of appropriate machine learning algorithms for the specific drift scenarios. Table 1.1, shows the list of machine learning algorithms and the internal method of learning. These algorithms are tested on the computer systems' physical properties such as RAM, CPU, IO bandwidth and network bandwidth to detect any unusual readings that referred to as drift and that causes system instability failures. This requires system monitoring at a deeper level with the large amount of data which also costs in terms of performance and resources. Machine learning solutions were found to be effective in this data analysis process. In this study, we are discussing the details of supervised and unsupervised machine learning algorithms and finally evaluate the usage of these algorithms based on evaluation metrics such as Accuracy, Precision, Recall, F1-Score, Support, Confusion Matrix etc (Zenisek et al., 2019; Charbuty and Abdulazeez, 2021).

Method	Classifier
Ensemble Learning	• Random Forest Classifier • Gradient Boosting Classifier • Ada Boost Classifier • Bagging Classifier • Extra Trees Classifier • XGB Classifier
Kernel-based Supervised Learning	• Support Vector Classifier (SVC)
Neural Network-based Supervised Learning	• Multi-layer Perceptron Classifier
Instance-based Supervised Learning	• K-Neighbors Classifier
Probabilistic Supervised Learning	• Gaussian Naive Bayes
Tree-based Supervised Learning	• Decision Tree Classifier
Linear Supervised Learning	• Logistic Regression
Anomaly Detection	• Isolation Forest • One Class SVM
Clustering	• K-Means
dimensionality reduction	• Principal Component Analysis

Table 1.1: Machine learning methods and its specifications

The outcome of the study is useful in implementing better machine learning techniques in drift detection on legacy software systems.

1.2 Problem Statement

Legacy software systems play a critical role in many industries, supporting essential business processes and operations. However, as time progresses, these systems face various challenges, including evolving user requirements, changing environments, and technological advancements. These challenges often lead to "concept drift," where the underlying data distribution and system behavior change over time. Detecting and managing concept drift in legacy software systems is paramount to ensure the continued reliability, performance, and relevance of these systems.

The problem addressed in this thesis is the detection of concept drift in legacy software systems and the selection of appropriate machine learning (ML) algorithms for this purpose. Concept drift can manifest in various forms, such as changes in user behavior, shifts in data patterns, or alterations in the software's operational context. Failure to detect and adapt to these changes can result in system inefficiencies, errors, security vulnerabilities, and diminished user satisfaction.

The central challenge is to develop a methodology for identifying concept drift in legacy software systems that is both effective and efficient. Furthermore, it is essential to determine which ML algorithms are most suitable for detecting specific types of drift scenarios. While a plethora of ML algorithms exist for concept drift detection, no one-size-fits-all solution is available. Each algorithm may perform differently under varying conditions, making it crucial to select the most appropriate technique for a given context.

1.3 Research Questions

How can concept drift in legacy software systems be effectively detected and managed using best suitable machine learning techniques, and how can appropriate algorithms be selected for specific types of drift scenarios?

1.4 Aim and Objectives

The aim of this research paper is to study and analyze the use of machine learning techniques to improve the drift detection process in DevOps.

.

The research objectives formulated based on the aim of this study are as follows:

- To investigate the use of machine learning algorithms for predicting drift in software systems.
- Analyze the results of the experiments and case studies to help derive practical recommendations for selecting the most appropriate machine learning algorithms for specific drift scenarios.

1.5 Significance of the Study

Relatively limited research has been undertaken regarding the importance of utilizing machine learning algorithms for the detection of drift in legacy systems. A thorough and meticulous examination of machine learning algorithms for drift detection holds the potential to enhance system stability and usability significantly. By identifying inaccuracies in system configurations and swiftly rectifying defects, this research aims to underscore the practical application of machine learning techniques in the detection of drift within legacy systems. This study will give a comparative analysis of various machine learning algorithms. This will help in improving the drift detection process by using suitable machine learning algorithms.

1.6 Scope of the Study

The study is focused on legacy software systems. Since the complex software environment comprises of a large amount of data that can be monitored for drifts, multiple servers, and applications would require. For simplicity, we are using simple machine specific properties for demonstrating the use of machine learning techniques. The same techniques can then be applied on bigger environments. We are not considering the live system monitoring in this study, but we are developing a simple system monitoring application- data collector built into Python for recording the monitoring data and use it later for the experimentation. The study of drift in the legacy software system is restricted to unexpected and problematic change in the values of RAM usage, CPU utilization, IO and network bandwidth.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

In the landscape of software engineering, the enduring relevance of legacy software systems is undeniable. These systems, often the backbone of organizations, have proven their mettle over time by facilitating critical business functions, housing valuable data repositories, and upholding years of domain-specific knowledge. However, as the operational environment evolves, user requirements shift, and technological advancements accelerate, legacy systems encounter a formidable challenge - the phenomenon of concept drift (Barros and Santos, 2018). Concept drift, characterized by the gradual transformation of data distributions and system behavior, poses a dynamic challenge for legacy software systems. These systems must adapt to shifting user preferences, changes in data patterns, alterations in operational contexts, and evolving regulatory compliance requirements. Failure to address concept drift in a timely and effective manner can lead to diminished system reliability, security vulnerabilities, and declining user satisfaction. The intricacies of managing concept drift in legacy software systems are manifold. Traditional approaches to drift detection and mitigation are labor-intensive, reactive, and often entail costly operational disruptions. Recognizing the pressing need for proactive and efficient solutions, organizations and researchers alike have turned to machine learning techniques as a promising avenue for addressing concept drift.

Machine learning, with its ability to harness historical data and adapt to changing patterns, offers the potential for automating drift detection and facilitating timely interventions (Zenisek et al., 2019). However, applying machine learning in the context of legacy systems is not without its challenges. Key considerations include selecting the most suitable machine learning algorithms, assessing their effectiveness in specific drift scenarios, and navigating the intricacies of integration with legacy software systems. This section aims to provide machine learning algorithms used for drift detection in legacy software systems. It places particular emphasis on the comparative study of these algorithms, assessing their performance under various drift scenarios. We delve into the evolution of the concept of concept drift, survey the landscape of machine learning algorithms that have been employed for drift detection, and examine practical challenges, considerations, and best practices associated with their application in the context of legacy software systems.

Objective of this study is machine learning algorithms in addressing concept drift within legacy software systems and to offer a comprehensive understanding of their strengths, limitations, and applicability. We will discuss about the supervised and unsupervised machine learning algorithms and various evaluation metrics. Through this exploration, we aim to pave the way for informed decision-making, driving future research in the pursuit of effective, efficient, and practical solutions to preserve the resilience and adaptability of legacy software systems in an ever-evolving technological landscape.

2.2 Drift in legacy systems:

2.2.1 Definition of Drift in the Context of Legacy Systems:

Legacy systems, developed years or decades ago, were tailored to specific requirements and assumptions at the time. Over time, changes in business processes, technology, user expectations, and the external environment can cause the data patterns encountered by these systems to deviate from their original state, resulting in concept drift. In the context of legacy systems, drift signifies a gradual change in the underlying data distribution, causing discrepancies between the original system assumptions and the current state (Han et al., 2020). It can manifest in various aspects of the legacy system, including business rules, data sources, user requirements, and environmental factors.(Meng et al., 2017)

2.2.2 Causes of Drift in Legacy Systems:

Drift in legacy systems can occur due to various reasons:

- Evolution of Business Requirements and User Needs: Drift can arise when business requirements and user needs evolve over time, leading to misalignment between the system and current demands.
- Technological Advancements and System Compatibility: Drift may result from technological advancements that render legacy systems incompatible with modern technologies.
- Environmental Changes and External Factors: External factors such as economic conditions, industry regulations, or market trends can force system adaptation (Escovedo et al., 2018).

- Data Sources and Quality Issues: Changes in data sources or data quality problems can introduce drift.
- Inadequate System Documentation and Knowledge Loss: Inadequate documentation and loss of knowledge can hinder drift detection and resolution.

2.2.3 Effects of Drift in Legacy Systems:

Drift in legacy systems can have several adverse effects:

- Deterioration of System Performance and Reliability: Drift can lead to reduced system performance and reliability, causing slower response times, increased errors, and inefficiency(Lu et al., 2016).
- Decreased Accuracy and Effectiveness of Machine Learning Models: Drift can significantly impact the accuracy and effectiveness of machine learning models, leading to decreased performance (Meng et al., 2017).
- Increased Operational and Maintenance Costs: Addressing drift can increase operational and maintenance costs due to the need for updates, customizations, and additional resources.
- Risk of Security Vulnerabilities and Breaches: Drift can introduce security vulnerabilities and increase the risk of breaches, especially in outdated systems (Dang et al., 2019).
- Challenges in Compliance with Regulatory Standards: Compliance with regulatory standards may become challenging as drift can lead to gaps in compliance controls.
- User Dissatisfaction and Negative Impact on Business Operations: Drift can result in user dissatisfaction, hinder business operations, and negatively affect productivity and customer satisfaction.

Addressing drift proactively is essential to mitigate these adverse effects and ensure the reliability and effectiveness of legacy systems.

2.2.4 Importance of Drift Detection in Legacy Systems:

1. Maintenance Challenges and Resource Allocation: Drift in legacy systems poses significant maintenance challenges. As these systems drift away from their original state, they become more complex to support and maintain. This necessitates allocating dedicated resources,

skilled personnel, and ongoing efforts to address drift-related issues, apply updates, and resolve compatibility problems.

2. Adaptation and Evolution of Legacy Systems: Drift compels organizations to adapt and evolve their legacy systems. Decisions must be made regarding how to handle drift, whether through incremental updates, refactoring, or complete modernization. Legacy systems may need adjustments to accommodate new requirements, technologies, or data sources, requiring strategic planning to balance continued functionality with potential migration benefits.

3. Decision-Making and System Upgrade Strategies: Drift in legacy systems presents challenges in decision-making. Organizations must make informed choices about system upgrades, technology investments, and maintenance priorities. These decisions must account for drift's impact on system performance, reliability, security, and user satisfaction, evaluating the costs, benefits, and risks associated with different upgrade strategies (Nogueira et al., 2018).

4. Integration of New Technologies and Architectural Changes: Drift often necessitates the integration of new technologies and architectural changes. Organizations may need to introduce modern frameworks, tools, or data management techniques to address drift effectively. This involves integrating cloud services, adopting microservices architectures, containerization, or leveraging AI and ML capabilities, all requiring careful planning and compatibility assessments.

To address these implications of drift, proactive monitoring and drift detection mechanisms are essential. Robust monitoring systems continuously assess legacy system performance and behaviour. Drift detection mechanisms, including statistical analysis, machine learning algorithms, or rule-based approaches, identify deviations from expected behaviour, enabling timely action (Hrusto and Lunds universitet., n.d.). Proactive management and strategic decision-making, guided by drift detection, help organizations detect drift early, minimize its impact, and ensure the longevity and effectiveness of legacy systems.

2.3 Machine Learning in Drift Detection

Machine Learning Approaches for Drift Detection in Legacy Systems

Machine learning has revolutionized various domains, including software engineering, by offering automated solutions to complex problems. One such challenge in software engineering is the detection of drift in legacy systems. Drift, referring to the gradual

divergence of a system's behaviour from its intended or expected behaviour, can lead to various issues, such as performance degradation, incorrect outputs, or security vulnerabilities (Zenisek et al., 2019). This article provides an in-depth exploration of machine learning approaches for drift detection in legacy systems, starting with an overview of machine learning in software engineering and then delving into the specific role of machine learning in drift detection.

2.3.1 Overview of Machine Learning in Software Engineering

At the heart of modern software engineering lies the formidable force of machine learning. This paradigm, rooted in artificial intelligence, empowers computers to learn from data and make informed decisions or predictions (Kreuzberger et al., 2023). The multifaceted applications of machine learning in software engineering have revolutionized the way we approach and solve complex problems. These applications extend their reach across several key domains:

1. Bug Detection and Resolution: Machine learning systems can be meticulously trained to pinpoint software bugs and offer pragmatic resolutions. This automated bug detection and resolution process significantly economizes time and resources in debugging endeavours.

2. Code Generation: The prospect of machine-generated code components or even entire modules holds immense promise. Machine learning algorithms can produce code snippets or templates based on specified functionalities, thereby augmenting development productivity.

3. Predictive Maintenance: Machine learning's predictive prowess can be harnessed to anticipate when software components or systems are at risk of malfunction. This prescient foresight enables proactive maintenance measures, preventing system failures before they occur (Zenisek et al., 2019).

4. Software Testing Augmentation: The orchestration of machine learning in software testing amplifies efficiency manifold. From the generation of test cases to their

execution and subsequent evaluation, machine learning streamlines and enhances the testing process.

5. Anomaly Detection Expertise: The capacity of machine learning models to discern anomalies in data is particularly pertinent to drift detection in legacy systems. By ingesting and analysing historical data, these models become adept at recognizing subtle deviations indicative of drift, a feat arduous to accomplish through manual inspection.

2.3.2 The Role of Machine Learning in Drift Detection

Machine learning, with its data-driven capabilities and adaptability, offers a potent arsenal of tools and techniques to address the formidable challenge of drift detection in legacy systems. Here, we dissect the multifaceted role of machine learning in this context.

1. Automating Drift Detection:
    i. Historical Data Analysis: Machine learning models are trained on historical data to understand the system's normal behaviour. They learn patterns, relationships, and dependencies within the data.
    ii. Continuous Monitoring: Once deployed, machine learning models continuously monitor incoming data in real-time. They compare the observed behavior against the learned patterns to detect deviations.
    iii. Alert Generation: When a deviation indicative of drift is detected, machine learning models trigger alerts or notifications. These alerts can be customized to reflect the severity of the deviation.
    iv. Reduction of Manual Effort: Automation through machine learning significantly reduces the need for manual monitoring and intervention, making drift detection more efficient and cost-effective.

2. Pattern Recognition and Anomaly Detection:
    i. Pattern Learning: Machine learning models excel at recognizing patterns within data, including those that may not be apparent to human observers. They capture intricate relationships and dependencies.

ii. Anomaly Detection: Drift often manifests as subtle anomalies in system behaviour. Machine learning models can detect these anomalies even in complex and multidimensional data.
iii. Granularity and Precision: Machine learning can provide a high degree of granularity and precision in detecting anomalies, reducing false positives and false negatives.

3. Adaptation to Changing Environments:

    i. Continuous Learning: Machine learning models have the capacity to adapt to changing conditions. They continuously update their understanding of normal behaviour based on new data.
    ii. Robustness: This adaptability makes machine learning models robust in dynamic environments, where the concept of "normal" behaviour may evolve over time.

4. Handling Diverse Data Types:
    i. Structured and Unstructured Data: Legacy systems often generate diverse data types, including structured logs, unstructured text, and time-series data. Machine learning models can handle these diverse data sources effectively.
    ii. Feature Engineering: Feature engineering techniques can be employed to extract relevant information from the data, enhancing the model's ability to detect drift.

5. Real-Time Alerting and Proactive Intervention:
    i. Timely Alerts: Machine learning models provide real-time alerts, enabling system administrators to respond promptly to drift-related issues.
    ii. Predictive Insights: Advanced machine learning techniques, such as time-series analysis, can provide predictive insights into when and where drift is likely to occur. This enables proactive maintenance and adaptation.

6. Mitigation of False Positives:
    i. Model Tuning: Machine learning models can be fine-tuned to minimize false positive alerts. This optimization ensures that system administrators are alerted only when drift is genuinely occurring.

ii. Reduced Alert Fatigue: Minimizing false positives reduces alert fatigue, allowing administrators to focus their attention on critical issues.

7. Cost-Efficiency and Resource Optimization:
    i. Reduced Manual Effort: By automating drift detection, machine learning models reduce the need for manual monitoring and testing, leading to cost savings and resource optimization.
    ii. Efficient Resource Allocation: System administrators can allocate their time and resources more efficiently, addressing drift-related issues in a targeted manner.

8. Continuous Self-Improvement:
    i. Learning from Data: Machine learning models have the capacity to learn from their experiences and mistakes. As they encounter new data and scenarios, they continuously refine their drift detection capabilities, ensuring ongoing reliability.

9. Scalability and Handling Big Data:
    i. Scalability: Machine learning models can scale to handle large volumes of data, which is often the case in legacy systems with extensive historical records.
    ii. Big Data Integration: Machine learning can seamlessly integrate with big data technologies, allowing for efficient processing and analysis of vast datasets.

10. Robustness in Complex Environments:
    i. Complex Dependencies: Legacy systems often have intricate dependencies on external components and services. Machine learning models can account for these dependencies and identify drift even when multiple factors are at play.

Machine learning assumes a pivotal role in drift detection in legacy systems, offering automation, pattern recognition, adaptability, and scalability. Its ability to handle diverse data types, provide real-time alerts, and minimize false positives makes it an invaluable tool for safeguarding the reliability and integrity of legacy systems. As the field of software engineering continues to evolve, machine learning is poised to play an increasingly central role in addressing the intricate challenges posed by drift in legacy systems.

2.3.3 Machine Learning Categories:

Machine learning can be broadly categorized into three main categories based on the type of learning and the nature of the problem they address:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

In this thesis, we are considering Supervised and Unsupervised Learning for analysis and comparison in detail.

2.4 Supervised Learning

Supervised learning stands as a foundational and widely employed machine learning paradigm, bearing immense significance in the landscape of artificial intelligence. In this section of the literature review, we delve into the intricacies of supervised learning, elucidating its core principles, algorithms, and real-world applications.

2.4.1 Core Principles of Supervised Learning:

Supervised learning, a foundational concept in machine learning, relies on several core principles that underpin its functionality and effectiveness. These principles are crucial to understanding how supervised learning operates and how it learns from labeled data to make predictions or decisions.

1. Labeled Data:

   At the heart of supervised learning is the concept of labeled data. This refers to a dataset that consists of pairs of input data and corresponding output labels. These labels serve as the ground truth or correct answers that the model aims to learn. For instance, in a medical diagnosis task, the input data might include patient information, and the labels would indicate whether each patient has a specific medical condition or not. Labelled data is essential because it provides the model with examples of what it should aim to predict

accurately. It guides the model in learning the underlying patterns and relationships within the data.

2. Predictive Modelling:

   Supervised learning involves the creation of a predictive model, also known as a hypothesis or function. This model is a mathematical or computational representation that takes input features and produces output predictions. The goal is to build a model that can generalize well to unseen data, meaning it can make accurate predictions for data it hasn't encountered during training. The model learns to approximate the mapping from inputs to outputs based on the patterns it discerns in the labeled data. In essence, the model's role is to capture and represent the relationships between input features and their corresponding labels, allowing it to make predictions on new, unseen examples.

3. Loss Function:

   A critical component of supervised learning is the definition of a loss function or cost function. This function serves as a quantitative measure of how well the model's predictions align with the actual labels in the training data. It quantifies the error or discrepancy between the predicted values and the ground truth. The choice of a suitable loss function depends on the specific nature of the problem. For example, mean squared error (MSE) is commonly used for regression tasks, while cross-entropy loss is prevalent in classification tasks. The loss function provides a numerical measure of the model's performance, and the objective during training is to minimize this loss. Optimization techniques are employed to iteratively adjust the model's parameters to achieve this minimization.

4. Optimization:

   Supervised learning models are trained using optimization techniques, with gradient descent being a common choice. Optimization involves iteratively updating the model's internal parameters in a direction that reduces the value of the loss function. The gradient of the loss function with respect to the model's parameters guides these updates. The model "learns" by adjusting its parameters to minimize the error between its predictions and the true labels in the training data. This iterative process continues until a convergence criterion is met, which could be when the loss reaches a satisfactory level or when a predefined number of training iterations is completed.

The core principles of supervised learning revolve around the use of labeled data to train a predictive model. This model learns to make accurate predictions by minimizing a loss function through optimization techniques. These principles are foundational to the success of supervised learning in a wide range of applications, from image recognition to natural language processing and beyond.

2.4.2 Types of Supervised Learning

Supervised learning encompasses two primary types, classification, and regression, each tailored to specific types of problems. In this detailed explanation, we explore these types and provide real-world examples of their applications.

A. Classification:

Classification is a type of supervised learning where the model's goal is to assign input data to predefined categories or classes. In essence, it involves making discrete decisions based on input features. Key aspects and examples of classification are mentioned in detail below:

Key Aspects:

1. Categories or Classes: In classification, there are distinct categories or classes into which data points are assigned. These classes can represent different labels, groups, or outcomes. For instance, in email classification, the classes are often "spam" or "not spam."
2. Binary and Multiclass Classification: Classification can be binary, where there are two classes, or multiclass, where there are more than two classes. Examples of binary classification include spam detection (spam or not spam) and fraud detection (fraudulent or legitimate). In multiclass classification, a data point can belong to one of several classes, such as classifying images of animals into categories like "cat," "dog," or "horse."

Examples of Classification:

1. Email Spam Detection: This is a classic example of binary classification. The model learns to classify incoming emails as either "spam" or "not spam" based on the content and features of the email.

2. Image Classification: In image classification, the model assigns labels to images based on their content. For example, it can classify images of animals into various categories, identifying whether an image contains a "cat," "dog," or "bird."
3. Sentiment Analysis: Sentiment analysis involves determining the sentiment or emotion expressed in text data. It can classify text reviews as "positive," "negative," or "neutral." For instance, classifying product reviews as either positive or negative.

B. Regression:

Regression is another type of supervised learning, but it's used for predicting continuous numerical values. Unlike classification, which deals with discrete categories, regression models provide output on a continuous scale. Here are the key aspects and examples of regression:

Key Aspects:

1. Continuous Outputs: In regression, the output is a continuous numerical value. This value can represent a wide range of possibilities, such as prices, temperatures, scores, or quantities. For example, predicting the price of a house based on its features is a regression task.
2. Prediction Range: Regression models provide predictions within a specific range. For instance, a model predicting house prices will produce numerical values, but they will be constrained within a realistic price range based on the data.

Examples of Regression:

1. House Price Prediction: Predicting the selling price of a house based on factors like square footage, number of bedrooms, location, and other features. The output is a continuous value representing the price.
2. Stock Price Forecasting: Forecasting the future prices of stocks or financial assets based on historical data and market indicators. The output is a continuous numerical value representing the expected price.
3. Temperature Prediction: Predicting future temperatures based on historical weather data, time of year, and other relevant variables. The output is a continuous numerical value representing the temperature in degrees.

2.4.3 Key challenges of supervised learning

- Labelled Data: Obtaining labelled data can be expensive and time-consuming.
- Imbalanced Datasets: Class imbalances can lead to biased models.
- Overfitting and Underfitting: Models can either memorize the training data (overfitting) or be too simplistic (underfitting).
- Curse of Dimensionality: High-dimensional data can be challenging for models.
- Data Quality and Noise: Noisy or biased data can affect model accuracy.
- Model Selection and Hyperparameter Tuning: Choosing the right model and tuning hyperparameters is critical.
- Interpretability: Complex models can be hard to interpret.
- Generalization to Unseen Data: Ensuring models work well on new data is crucial.
- Privacy and Ethical Concerns: Handling sensitive data requires ethical considerations.
- Scalability: Training large models may require significant computational resources.

Supervised learning encompasses classification and regression, each with its own set of applications and characteristics. Classification is used for discrete decision-making tasks, assigning data to predefined categories, while regression is employed when predicting continuous numerical values is required. These two types of supervised learning have a wide range of real-world applications across various domains, making them fundamental techniques in the field of machine learning.

2.4.4 Drift Detection Using Supervised Learning:

1. Supervised drift detection involves comparing the model's performance on new data from the legacy system to its performance on historical data with known labels. This approach relies on having access to labeled data to monitor changes in model accuracy, but it offers a clear and interpretable way to detect drift (Ibrahim et al., 2017). Here's a step-by-step explanation:
2. Historical Data with Labels: In supervised drift detection, you start with a historical dataset that has known labels or ground truth. This dataset represents the data distribution that the model was initially trained on.

3. Periodic Retraining: To monitor drift, you periodically retrain the machine learning model using the legacy data, including both features and labels. The frequency of retraining depends on the rate of data drift and how critical it is to maintain model performance.
4. Model Evaluation: After each retraining iteration, you evaluate the model's performance on a validation or holdout dataset. This validation dataset should ideally come from the same legacy system but represents a more recent snapshot of data.
5. Detecting Drift: Drift detection in the supervised approach involves comparing the model's performance metrics before and after retraining. Common metrics include accuracy, precision, recall, F1-score, or area under the ROC curve (AUC-ROC). A significant drop in performance metrics, typically beyond a predefined threshold, suggests the presence of data drift. The model is no longer performing as well as it did on the historical data.
6. Alerting and Remediation: When drift is detected, you trigger an alert or take corrective actions. The nature of these actions depends on the severity of drift and the available resources:
7. If drift is minor, you might consider adapting the model by adjusting its features, hyperparameters, or architecture to better align with the changing data distribution.
8. If drift is substantial or the model's performance cannot be restored, you may need to consider retraining the model with the most recent data. This could involve a full model retraining or training on a subset of data containing recent examples.

2.4.5 Challenges and Considerations

1. Data Labeling: The supervised approach relies on having labeled data for model performance evaluation. Ensuring that the labels are accurate and up-to-date is crucial.
2. Threshold Setting: Defining appropriate thresholds for detecting drift can be challenging. This requires careful consideration and domain knowledge to determine when model performance changes are significant.
3. Data Representation: Drift detection may require revisiting the feature engineering process to ensure that the model's features remain relevant and representative of the evolving data distribution.
4. Alert Handling: Handling drift alerts requires well-defined procedures and workflows, including who is responsible for acting and what those actions should be.

5. Computational Resources: Retraining models periodically, especially with large legacy datasets, may require substantial computational resources. Efficient model training and evaluation strategies are essential.

Supervised drift detection offers a robust way to monitor and address data drift in legacy systems where labeled data is available. It ensures that machine learning models continue to provide accurate predictions as data distributions evolve over time, making it valuable for maintaining model reliability and performance.

2.5 Unsupervised Learning

Unsupervised learning is a machine learning paradigm where the model explores and extracts patterns, structures, or representations from unlabeled data. Instead, it seeks to uncover inherent structures and relationships within the data itself.

2.5.1 Core Principles of Unsupervised Learning

1. No Labeled Output:
   Unsupervised learning operates on data without the presence of labeled output or target variables. In supervised learning, algorithms are trained to make predictions based on examples that include both input data and corresponding output labels. Unsupervised learning, in contrast, focuses solely on the input data, aiming to discover patterns or structures within it (Koroglu et al., 2016).
2. Discovering Patterns and Structures:
   The primary objective of unsupervised learning is to uncover hidden patterns, structures, or relationships within the data (Lewis et al., 2022). This can involve finding groups or clusters of similar data points, reducing the dimensionality of the data to capture essential information, detecting anomalies or outliers, or generating new data samples that follow the same underlying data distribution.
3. Exploration and Insight:
   Unsupervised learning is often used for exploratory data analysis. By applying unsupervised techniques, data scientists and researchers can gain valuable insights into the data. These insights can include understanding the natural groupings of data points, identifying unusual or unexpected data patterns, and simplifying complex data representations for easier interpretation.
4. Reducing Data Complexity:
   Many unsupervised learning methods aim to simplify complex data representations. For example, dimensionality reduction techniques like Principal Component Analysis (PCA) transform high-dimensional data into a lower-dimensional representation while retaining the most critical information. This simplification can lead to more efficient data processing and visualization.

2.5.2 Types of Unsupervised Learning

1. Clustering:
    I. K-Means Clustering: This method partitions data into 'K' clusters based on similarity, aiming to minimize the intra-cluster distance (Jain et al., 2022).
    II. Hierarchical Clustering: It creates a hierarchical tree of clusters, allowing for a flexible way to visualize the relationships between data points.
    III. DBSCAN (Density-Based Spatial Clustering of Applications with Noise): DBSCAN groups data points based on their density, making it effective for detecting clusters of varying shapes and sizes.
2. Dimensionality Reduction:
    I. Principal Component Analysis (PCA): PCA reduces data dimensionality by projecting it onto a lower-dimensional subspace while preserving as much variance as possible.
    II. t-Distributed Stochastic Neighbor Embedding (t-SNE): t-SNE is used to visualize high-dimensional data in a lower-dimensional space, with a focus on preserving pairwise similarities between data points.
3. Anomaly Detection:
    I. Isolation Forest: This technique identifies anomalies (outliers) by isolating them using a random
    II. One-Class SVM (Support Vector Machine): One-Class SVM constructs a boundary around most data points, classifying anything outside this boundary as an anomaly (Jain et al., 2022).
4. Density Estimation:
    I. Gaussian Mixture Models (GMMs): GMMs model data as a mixture of multiple Gaussian distributions, enabling flexible density estimation.
    II. Kernel Density Estimation (KDE): KDE estimates the probability density function of a continuous random variable by smoothing data points.
5. Association Rule Learning:
    I. Apriori Algorithm: It identifies associations or frequent itemsets in transactional data, which is valuable for market basket analysis and recommendation systems.
    II. FP-Growth (Frequent Pattern Growth): FP-Growth efficiently discovers frequent itemsets by building a compact data structure.
6. Generative Models:

I. Variational Autoencoders (VAEs): VAEs encode and decode data in a lower-dimensional latent space, enabling the generation of new data samples that follow the same distribution as the training data.

II. Generative Adversarial Networks (GANs): GANs consist of a generator and a discriminator network and are used for generating synthetic data that closely resembles the training data.

7. Self-Organizing Maps (SOMs):

   SOMs are neural network-based techniques that create a 2D grid of neurons to visualize and cluster high-dimensional data, preserving the topological relationships between data points.

8. Word Embeddings:

   Techniques like Word2Vec and GloVe are used in natural language processing to represent words in continuous vector spaces, enabling semantic understanding and improving the performance of various NLP tasks.

2.5.3 Applications of Unsupervised Learning

1. Customer Segmentation: Businesses use unsupervised learning to segment their customer base into groups with similar behaviors and preferences. This enables targeted marketing efforts and tailored product recommendations (Sethi and Kantardzic, 2017).
2. Anomaly Detection: Unsupervised learning is crucial for identifying anomalies or outliers in data, which can be indicative of fraud, network intrusions, or equipment malfunctions(Abbasi et al., 2021).
3. Natural Language Processing (NLP): In text analysis, unsupervised learning techniques like topic modeling are employed to discover latent topics within large collections of documents. This aids in content organization and understanding (de Mello et al., 2019).
4. Recommendation Systems: Collaborative filtering, a type of unsupervised learning, powers recommendation systems by identifying users with similar preferences and suggesting items they might like.
5. Data Compression: Unsupervised learning techniques can be used to reduce data dimensionality, which is essential for image and audio compression, saving storage space while retaining essential features.

2.5.4 Drift Detection Using Unsupervised Learning

Detecting drift in data streams is a crucial task in various applications, including those within the realm of data science. Drift detection methods are essential for ensuring the accuracy and reliability of machine learning models deployed in dynamic environments. Drift detection refers to the identification of changes in data distribution over time. Unsupervised learning methods are particularly useful for this task, as they do not rely on labeled data (Casado et al., 2022). Several approaches have been analysed in this literature. Drift detection is vital in various domains, including fraud detection, network monitoring, and predictive maintenance (Charbuty and Abdulazeez, 2021). It ensures that machine learning models remain accurate and reliable when deployed in environments where data distribution can change over time.

2.5.5 Challenges and Considerations

1. Evaluation Metrics: Unlike supervised learning, where evaluation metrics like accuracy are straightforward, assessing the performance of unsupervised learning models can be more nuanced. Metrics such as silhouette score and Davies-Bouldin index are used for clustering, but their interpretation depends on the specific problem (Dang et al., 2019).
2. Interpretability: Unsupervised learning models can produce results that are challenging to interpret, especially in high-dimensional spaces. Domain expertise is often required to make sense of the discovered patterns (Fahmideh et al., n.d.).
3. Curse of Dimensionality: Unsupervised learning can suffer from the curse of dimensionality when dealing with high-dimensional data. This can lead to computational challenges and the need for dimensionality reduction techniques (Gholami et al., 2017).

Unsupervised learning is a valuable approach when you want to explore and extract insights from data without predefined labels. It has a wide range of applications in various domains, offering a means to discover hidden structures and patterns that may not be apparent through traditional analysis methods.

2.6 Summary

In this section, we have provided a comprehensive overview of key concepts and frameworks that serve as the foundation for our research on drift detection in legacy software systems using machine learning techniques. We began by introducing the importance of legacy systems and the challenges they face due to evolving data distributions. Recognizing the significance of adapting these systems to concept drift, we explored the role of machine learning in addressing this challenge.

Our exploration delved into the realms of supervised and unsupervised learning, both of which hold promise in detecting drift within legacy systems. In the realm of supervised learning, we elucidated the core principles, types, and challenges, shedding light on its potential to enhance drift detection accuracy. Additionally, we discussed the application of supervised learning techniques to address drift detection, highlighting the challenges and considerations specific to this approach.

In the domain of unsupervised learning, we dissected the foundational principles, types, and applications, showcasing its ability to detect drift in an autonomous and data-driven manner. We also outlined the potential applications of unsupervised learning in drift detection and the associated challenges and considerations.

Our exploration has set the stage for the subsequent chapters, where we will delve deeper into the practical implementation of machine learning techniques for drift detection in legacy systems. By integrating the principles and insights gathered thus far, we aim to develop robust and adaptable drift detection mechanisms that empower legacy systems to thrive in an ever-evolving data landscape.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

In this section, we elucidate the methodological framework that underpins our investigation into drift detection in legacy systems using machine learning techniques. Research methodology is the compass that guides our journey from data collection to interpretation of results. It shapes the structure and rigor of our investigation, ensuring that our findings are reliable, valid, and ultimately, valuable to the field of data science and legacy system maintenance. In the following subsections, we embark on a comprehensive exploration of each facet of our research methodology. We begin by detailing the processes of data collection and pre-processing, setting the stage for the selection of appropriate features and engineering techniques. Subsequently, we delve into the critical aspect of drift detection algorithm selection and conclude with the rationale behind our choice of evaluation metrics.
Our research methodology, as detailed in the subsequent sections, forms the bedrock upon which our study is built. It enables us to address our research questions with precision, methodological rigor, and the overarching goal of contributing to the understanding of drift detection in legacy systems.

3.2 Data Collection and Pre-processing

Data collection and reprocessing are fundamental stages in any data-driven research, and this holds especially true in the context of drift detection in legacy systems. In this section, we embark on a comprehensive exploration of how data was collected and prepared for analysis in our study.

3.2.1 Data Collection

Data Source and Origin:
The first step in any data-driven research endeavour is to establish the source and origin of the data (Siraskar et al., 2023). In our study, we generated the data using the data-collector python utility. This source is integral to understanding the context and reliability of the dataset. The dataset is generated from the computer system over a long period of time. The system is prepared such that multiple processes are running including the resource intensive processes

like games, browser with multiple heavy websites, multiple instances of IntelliJ and webservers. This ensures high usage of all the four properties we have recorded. We have kept the system with various levels of resource utilization to record readings at lower, medium, and high degree of utilization. We have recorded readings of a computer system assuming the legacy application is hosted on it and its performance and stability will be depending in the systems performance (Hrusto et al., 2021).

Data Retrieval Process:

In our research, the data retrieval process involves the systematic collection of performance metrics, including RAM usage, CPU utilization, IO bandwidth, and network bandwidth, from the target legacy system. This process is meticulously designed to ensure the accuracy and completeness of the dataset. Specifically:

Automated Data Extraction: A Python program has been developped that automates the data extraction process. This program regularly queries the system's performance metrics at predefined intervals, ensuring a consistent and frequent flow of data.

Data Sources: The Python program collects data directly from the system's monitoring tools or performance counters. This automated approach minimizes the chances of human error and enhances the efficiency of data retrieval.

3.2.2 Data Coverage and Temporal Scope:

To evaluate the representativeness of our dataset, it is imperative to specify both the coverage and temporal scope of the collected data (Duda et al., 2020):

- Data Coverage: This refers to the percentage of data captured relative to the entire operational scope of the legacy system. Our Python program is configured to collect data comprehensively, aiming to capture a high percentage of system activities. The coverage is assessed in terms of how well it represents the system's diverse operations.
- Temporal Scope: The duration over which data is collected is a critical aspect of our dataset. We collect data over an extended time frame, ensuring that the temporal scope is sufficient to capture variations in system performance. This duration is selected to provide a comprehensive view of the system's behaviour over time.

3.2.3 Data Quality and Integrity

The quality and integrity of collected dataset is maintained with a top priority:

- Data Accuracy: We implement rigorous data validation checks within our Python program to minimize errors in data collection. Data accuracy is maintained by using well tested python libraries that accurately reads the system properties. Any discrepancies or anomalies detected during the retrieval process should trigger alerts for manual review and correction.
- Data Consistency: To maintain data consistency, we enforce standardized data formats and units during collection. This consistency allows for meaningful comparisons across different performance metrics. RAM usage and CPU utilization is calculated as a percentage of the total available resources and IO bandwidth and network bandwidth are calculated at actual in Mega Bytes.
- Data Completeness: Our Python program is designed to avoid any missing datapoints. This ensured no data loss during the data collection process.

3.2.4 Data Privacy and Ethical Considerations

Data privacy and ethics are considered with utmost priority. No private data is retrieved during the process and maintained secured process to keep the data secured.

- Anonymization: Personally identifiable information (PII) is not relevant to our research. We ensure that no sensitive or personal data is collected or stored as part of the performance metrics. All collected data is anonymized and stripped of any identifying information.
- Security Measures: Our Python program adheres to industry-standard security measures to protect the integrity and confidentiality of the collected data. This includes encryption during data transmission and secure storage practices.
- Regulatory Compliance: We strictly adhere to relevant data regulations and industry-specific standards. Compliance ensures that our data collection practices are aligned with legal and ethical guidelines.

Data Splitting for Validation:

For robust model validation, describe how the dataset was split into training, and test sets. A standard test size of 20% of the dataset is used and random_state of 42 is used during the complete analysis process.

Data Pre-processing Tools and Software:

Application developed in python is used where libraries and packages like pandas, scikit-learn are employed for data manipulation and transformation. The data is stored in csv format file.

3.3 Feature Extraction and Engineering

RAM usage, CPU utilization, IO bandwidth, and network bandwidth are fundamental metrics that directly reflect the health and performance of a computing system. These metrics are critical for legacy systems, where maintaining optimal performance is often a top priority. Any significant changes in these metrics can indicate a potential issue or drift in system behaviour. These metrics are highly sensitive to changes in system behaviour. An increase in RAM usage, a spike in CPU utilization, a drop in IO bandwidth, or a sudden change in network bandwidth can signal drift or anomalies in the system. As such, they serve as excellent indicators for detecting deviations from expected operational patterns. Drifts or anomalies in these metrics can directly impact the user experience. For example, high CPU utilization can lead to slower response times, while reduced network bandwidth can result in communication failures. Detecting these changes promptly is crucial for maintaining system reliability. RAM usage, CPU utilization, IO bandwidth, and network bandwidth are well-understood metrics in the domain of system performance monitoring. IT professionals and system administrators routinely use these metrics to assess the health of legacy systems. This existing domain knowledge facilitates the interpretation of drift detection results. These metrics are often interconnected. For instance, high CPU utilization may lead to increased RAM usage due to more processes running, and this, in turn, might affect IO and network bandwidth. By including these metrics together, you capture the holistic impact of drift on system performance. Legacy systems often have well-established historical performance baselines. By monitoring properties over time, we can establish a historical context for these metrics. This context is invaluable for identifying gradual drifts or long-term trends in system behaviour. Understanding how resources like RAM and CPU are utilized can aid in resource

allocation and optimization efforts. If drift is detected in these metrics, it may prompt adjustments to resource allocation or system configurations to ensure efficient operation.
Feature engineering involves creating new features or transforming existing ones to improve the dataset's representation. We set the following variables which will determine the drift in the legacy system infrastructure.

- Threshold_Ram_Usage
- Threshold_Cpu_Utilization
- Threshold_Difference_In_Ram_Usage
- Threshold_Difference_In_Cpu_Utilization
- Threshold_IO_Bandwidth
- Threshold_Network_Bandwidth

These values kept configurable during the experiments. For the final experiments, we have set the value for these variables as given below:

Drift condition	**value**
Threshold_Ram_Usage	95
Threshold_Cpu_Utilization	90
Threshold_Difference_In_Ram_Usage	25
Threshold_Difference_In_Cpu_Utilization	50
Threshold_IO_Bandwidth	1600
Threshold_Network_Bandwidth	1600

Table 3.3: Drift conditions with configurable values

Based on these values drift is calculated. To compare the Threshold difference between RAM and CPU utilization, new features are added as below:

1. Difference_In_Ram_Usage: Difference between current RAM usage and the previous RAM usage

2. Difference_In_Cpu_Utilization: Difference between the current CPU utilization and its previous value

These two features help us in determining if the system resource usage is drastically increased and system may stop responding or crashing or sudden shutdown. Any new drift will spike the alarm or notification for manual or automatic intervention.

We have created new feature from the values of the existing feature during feature engineering. For instance, we have added the feature difference in the current and previous RAM usage and same for the CPU utilization. This gives us idea on how quickly these metrics are changed. This help us identifying the sudden drift in the system.

3.4 Drift Detection Algorithms Selection

We have considered supervised and unsupervised learning algorithms for drift detection so that we would have a good consideration of the usage and its efficiency. We have used some well-establishhed algorithms for both supervised and unsupervised drift detection which are used in various kinds of applications and each one has its own advantages. The selection of machine learning algorithms was guided by multiple criteria, considering the specific challenges posed by legacy systems and the need for effective concept drift detection. The following criteria were considered in the algorithm selection process (Barros and Santos, 2019):

- Adaptability to Drift: Algorithms capable of adapting to changing data distributions were prioritized, as legacy systems often experience gradual or abrupt shifts in their operational context (Priyam et al., 2013).
- Interpretability: Given the complexity of legacy systems, algorithms with interpretable results were preferred to facilitate insights into the detection process and aid in decision-making.
- Efficiency: Legacy systems often operate in real-time or near-real-time environments. Therefore, algorithms with low computational overhead and fast training times are favoured (Magal et al., 2015).

- Scalability: The selected algorithms should be scalable to handle large-scale legacy system datasets efficiently.
- Robustness: The algorithms should demonstrate resilience to noise and outliers commonly encountered in legacy system data.

Based on above criteria, following algorithms are used:

A. Supervised Learning Classifiers:

1. Random Forest Classifier:

   a) Method: Ensemble Learning (Bagging)

   b) Characteristics: Builds multiple decision trees on bootstrapped subsets of data, averages their predictions (classification) or takes a majority vote (regression).

   c) Applications: Classification and regression tasks with complex datasets, resistant to overfitting and robust to noise.

2. Gradient Boosting Classifier:

   a) Method: Ensemble Learning (Boosting)

   b) Characteristics: Sequentially builds weak learners, emphasizing misclassified instances, and combines their predictions using weighted averaging.

   c) Applications: Accurate predictive modeling, particularly for structured/tabular data with potential feature interactions.

3. Ada Boost Classifier:

   a) Method: Ensemble Learning (Boosting)

   b) Characteristics: Assigns higher weights to misclassified samples, focuses on hard-to-classify instances by iteratively training new models.

   c) Applications: Binary classification, particularly when combined with weak learners.

4. Bagging Classifier:

   a) Method: Ensemble Learning (Bagging)

b) Characteristics: Trains multiple models independently on different data subsets and aggregates predictions through averaging (classification) or voting (regression).

c) Applications: Reduces variance, improving stability and generalization of models.

5. Extra Trees Classifier:

a) Method: Ensemble Learning (Bagging)

b) Characteristics: Builds decision trees with random feature subsets and node splitting, effectively reducing overfitting and computational costs.

c) Applications: Classification tasks requiring reduced risk of overfitting.

6. Voting Classifier:

a) Method: Ensemble Learning

b) Characteristics: Combines predictions of multiple classifiers through majority voting (classification) or averaging (regression).

c) Applications: Leveraging diverse models for better overall performance.

7. SVC (Support Vector Classifier):

a) Method: Kernel-based Supervised Learning

b) Characteristics: Finds optimal hyperplane to maximize margin between classes, uses kernel functions for non-linear transformations.

c) Applications: Binary and multi-class classification, separating data with clear class boundaries.

8. MLP Classifier (Multi-layer Perceptron Classifier):

a) Method: Neural Network-based Supervised Learning

b) Characteristics: Consists of interconnected layers of nodes (neurons), suitable for complex relationships and large datasets.

c) Applications: Diverse classification tasks, including image recognition, natural language processing, and more.

9. K-Neighbours Classifier:

a) Method: Instance-based Supervised Learning

b) Characteristics: Assigns class label based on majority class among k-nearest neighbors in feature space.

c) Applications: Pattern recognition, recommendation systems, and anomaly detection.

10. Gaussian NB (Gaussian Naive Bayes):

a) Method: Probabilistic Supervised Learning

b) Characteristics: Assumes features are conditionally independent given the class, uses Bayes' theorem for classification.

c) Applications: Text classification, spam detection, and other tasks with discrete features.

11. Decision Tree Classifier:

a) Method: Tree-based Supervised Learning

b) Characteristics: Creates a tree structure where nodes represent features, and leaves correspond to class labels.

c) Applications: Easy-to-understand models for binary and multi-class classification.

12. XGB Classifier:

a) Method: Ensemble Learning (Boosting)

b) Characteristics: Optimized gradient boosting framework with regularization, handling missing values, and parallel processing.

c) Applications: Diverse classification tasks, particularly those requiring high predictive accuracy.

13. Logistic Regression:

a) Method: Linear Supervised Learning

b) Characteristics: Models probability of binary outcome using logistic function, applies weights to features.

c) Applications: Binary classification, especially when interpreting feature coefficients is important.

B. Unsupervised Learning Classifiers:

1. Isolation Forest:

a) Method: Anomaly Detection

b) Characteristics: Uses random partitioning to isolate anomalies efficiently, scales well with large datasets.

c) Applications: Detecting outliers in various domains, such as fraud detection and network intrusion.

2. K Means:

a) Method: Clustering

b) Characteristics: Divides data into k clusters by minimizing sum of squared distances between points and cluster centroids.

c) Applications: Grouping similar data points, market segmentation, image compression.

3. One Class SVM (One-Class Support Vector Machine):

a) Method: Anomaly Detection

b) Characteristics: Finds hyperplane that separates normal data from outliers, useful for one-class classification.

c) Applications: Anomaly detection, fraud detection, outlier identification.

These classifiers embody a rich toolkit for various data analysis needs, ranging from classification and regression to clustering and anomaly detection, providing researchers and practitioners with versatile techniques for extracting insights from diverse datasets.

The same dataset would be used and processed in a drift detection tool for various machine learning algorithms. A comparative study will be conducted based on the quantitative data obtained against each machine learning algorithm used. Based on the findings, an efficient method for drift detection on certain types of drift will be proposed.

3.5 Evaluation Metrics

2.5.1 Metric Selection:

Evaluation metrics are essential for quantifying the performance of drift detection algorithms. We have chosen accuracy, precision, recall, F1-score, support, confusion metric in supervised learning. For Unsupervised learning we have also evaluated R-squared Score, Mean Squared Error and Silhouette Score as we don't consider labeled data in unsupervised learnings.

2.5.2 Rationale for Metrics:

Classification Metrics (Accuracy, Precision, Recall, F1-Score, Support, Confusion Matrix): These metrics are used to evaluate the performance of classifiers in tasks like binary or multiclass classification. They help assess how well the model can correctly classify instances into different classes.

Regression Metrics (R-squared, Mean Squared Error): These metrics are applied in regression tasks to measure the goodness of fit and prediction accuracy of regression models.

Clustering Metric (Silhouette Score): The Silhouette Score is used in unsupervised learning to assess the quality of clusters generated by clustering algorithms.

3.4. Research Plan

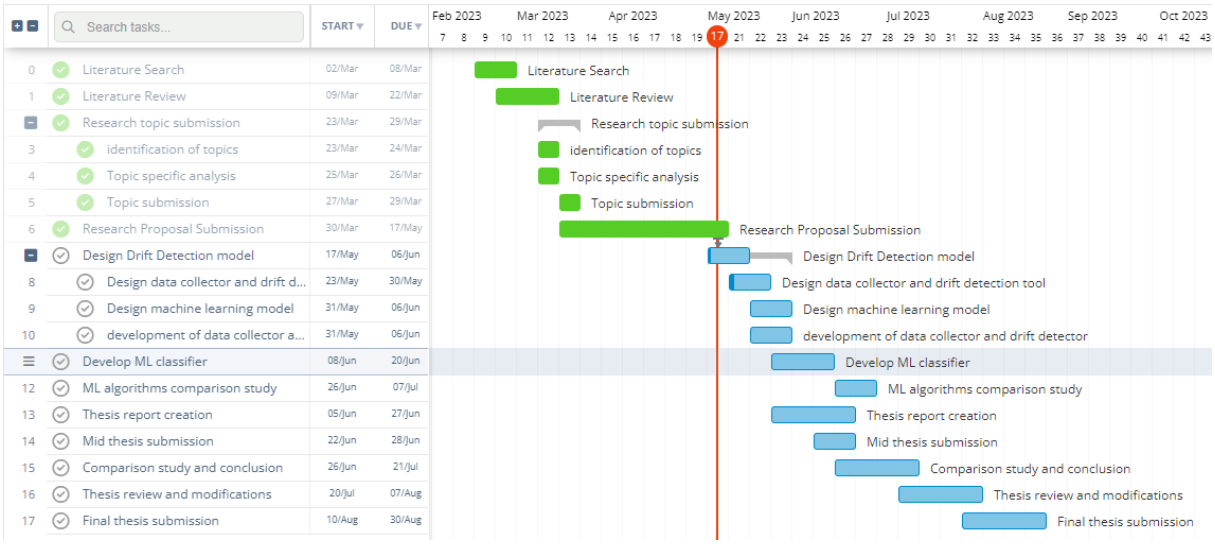


Figure 3.4: Research Plan

3.5 Summary

In this chapter we have provided a structured framework for our study on drift detection on legacy systems using machine learning algorithms. This chapter introduced the methodology guiding our research, outlined the steps involved in data collection and pre-processing, discussed the selection of drift detection algorithms, and presented our chosen evaluation metrics. Additionally, we outlined the research plan that will guide the subsequent phases of our investigation.The methodologies and considerations detailed in this chapter lay the groundwork for the practical implementation and experimentation that follow in the subsequent chapters, facilitating a systematic and comprehensive exploration of drift detection in legacy software systems.

CHAPTER 4: RESULT AND ANALYSIS

4.1 Introduction

In the previous chapters, we explored the significance of concept drift detection in legacy systems and reviewed various machine learning techniques that can be employed for this purpose. In this chapter, we present the results of our experiments conducted to evaluate the performance of these techniques in detecting concept drift in legacy systems. We begin by describing the dataset selection and preprocessing procedures, followed by an overview of the evaluation metrics used to assess the performance of the techniques. Subsequently, we delve into the experimental design, highlighting the key considerations that guided our approach. Finally, we present the detailed results of applying different machine learning techniques to our selected legacy system datasets, analyzing their accuracy, robustness, efficiency, and potential applicability.

By systematically evaluating the performance of these techniques, we aim to gain insights into their effectiveness in detecting concept drift in legacy systems. This empirical analysis will not only provide a clear understanding of the strengths and weaknesses of each technique but will also contribute to the broader understanding of how machine learning methods can be harnessed to maintain and enhance legacy systems in the face of evolving data distributions. Our findings will aid practitioners and researchers in making informed decisions when selecting a suitable technique for concept drift detection in legacy systems, ultimately contributing to the overall reliability and adaptability of these critical systems.

In the following sections of this chapter, we delve into the details of our experimentation process and present a thorough analysis of the obtained results.

4.2 Dataset Description:

In the context of drift detection in legacy systems, the monitoring and analysis of system resource metrics, including RAM usage, CPU utilization, difference in RAM usage, difference in CPU utilization, and IO bandwidth, hold significant importance. These metrics provide valuable insights into the performance and behaviour of legacy systems when dealing with evolving data or concept drift. Below, we discuss the significance of each of these metrics used for the experimentation:

1. RAM Usage:
    a. Significance: RAM (Random Access Memory) is a critical resource in legacy systems. Monitoring RAM usage is vital because concept drift can potentially lead to increased memory requirements. For example, if the data distribution shifts or new features are introduced, more memory may be needed for processing.
    b. Drift Impact: An abrupt increase in RAM usage can be indicative of concept drift, especially if it coincides with changes in data characteristics. Detecting such increases can trigger system administrators or automated processes to allocate additional memory resources as needed.

2. CPU Utilization:
    a. Significance: CPU utilization measures the extent to which the CPU is processing tasks. High CPU utilization can indicate increased processing demands due to concept drift, such as more complex calculations or data transformations.
    b. Drift Impact: A sustained increase in CPU utilization, especially during specific periods or with specific datasets, can signal the presence of concept drift. It can help identify when the system is struggling to cope with changing data patterns.

3. Difference In RAM Usage:
    a. Significance: The difference in RAM usage over time, often referred to as RAM delta, helps track changes in memory consumption. Significant differences may be an early indicator of concept drift.
    b. Drift Impact: A rapid and substantial increase in RAM delta can signify that the legacy system is encountering new data patterns or processing requirements. This can trigger alert mechanisms for further investigation.

4. Difference In CPU Utilization:
    a. Significance: Similar to RAM delta, the difference in CPU utilization measures the change in processing load over time. Sudden spikes or sustained increases can be indicative of concept drift.
    b. Drift Impact: A noticeable change in CPU utilization, especially when associated with changes in data characteristics, can serve as an alarm for potential concept drift. It may prompt the need for system adaptation or optimization.

5. IO Bandwidth:
    a. Significance: Input/output (IO) bandwidth refers to the rate at which data is read from or written to storage devices. Monitoring IO bandwidth is crucial because shifts in data patterns can affect data retrieval or storage requirements.
    b. Drift Impact: A significant increase or decrease in IO bandwidth can be correlated with concept drift. For instance, if new data sources are introduced, more data may need to be read from storage, impacting IO bandwidth.

6. Network Bandwidth:
    a. Significance: Network bandwidth represents the capacity of a network to transmit data from one point to another within a specified time frame. Monitoring network bandwidth is essential because it influences the speed and efficiency of data communication between devices and servers. It also plays a critical role in ensuring that network resources are optimally utilized.
    b. Drift Impact: Significant fluctuations in network bandwidth can be indicative of concept drift within the network environment. For instance, if there is a sudden increase in the number of connected devices or a shift in the type of data being transmitted, it can impact network bandwidth. Monitoring these changes is vital for maintaining network performance and reliability.

Thus, monitoring these system resource metrics is essential for identifying and responding to concept drift in legacy systems. These metrics can serve as early warning signals, allowing organizations to adapt their systems, allocate additional resources, or implement optimizations to ensure the continued reliability and performance of legacy systems in the face of evolving data and operational requirements. The chosen dataset contains 50000 rows.

4.3 Drift Scenario

For the experimentation purpose, we have considered the labelled data with only two values: Drift and Non Drift. The Drift scenario is assumed when the property values mentioned in the section 4.2 exceeds the threshold values.

For example, consider the sample dataset below:

{
 "ram_usage": 33.69,
 "cpu_utilization": 84.32,
 "network_bandwidth": 1900,
 "io_bandwidth": 63800.03,s
 "ram_usage_difference": 35,
 "cpu_utilization_difference": 23
}

In the above dataset, the value of ram_usage_difference is 35, assuming the threshold value for ram usage difference as 30, which means that the ram usage has increased suddenly by 35, this has entitled for a drift scenario.

4.4 Evaluation Metrics for Algorithms

We run the python program on the same dataset and drift conditions with different classifications. We have analysed and compared the supervised and unsupervised specifications separately. The rationale behind comparing the two categories separately is explained below:

For unsupervised learning algorithms like KMeans clustering, accuracy is not typically used in the same way as it is in supervised learning (classification) tasks. Accuracy in unsupervised learning refers to how well the algorithm is able to discover underlying patterns or structures in the data.

In the case of KMeans clustering, there's no straightforward "accuracy" measure like we would have in classification problems. Instead, you can use metrics that evaluate the quality of the clusters formed by the algorithm. One commonly used metric is the Silhouette Score,

which measures how similar each data point is to its assigned cluster compared to other clusters.

In the program I provided earlier, I included the calculation of the Silhouette Score for KMeans clustering. You can interpret the Silhouette Score as follows:

- A score close to +1 indicates that the data point is well-matched to its own cluster and poorly matched to neighbouring clusters.
- A score close to 0 indicates that the data point is on or very close to the decision boundary between two neighbouring clusters.
- A score close to -1 indicates that the data point is probably assigned to the wrong cluster.
- Higher Silhouette Scores generally indicate better clustering quality.

In the context of anomaly detection using Isolation Forest or One-Class SVM, the Silhouette Score is not typically applicable because these algorithms are not clustering techniques. They are designed for outlier or anomaly detection rather than grouping data points into clusters.
Instead of the Silhouette Score, when working with anomaly detection, you typically assess the model's performance using metrics such as precision, recall, F1-score, and possibly area under the ROC curve (AUC-ROC) or area under the precision-recall curve (AUC-PR). These metrics focus on the ability of the model to correctly identify anomalies and distinguish them from normal data points.

For Supervised we have used Classification Metrics (Accuracy, Precision, Recall, F1-Score, Support, Confusion Matrix). These metrics are used to evaluate the performance of classifiers in tasks like binary or multiclass classification. They help assess how well the model can correctly classify instances into different classes (Drift Or No-Drift).

4.4.1 Supervised Learning:

The results of the experiments for supervised learning are calculated. Figure 4.4 shows the chart of accuracy of the specifications. From the chart it’s clear that Gradient Boosting, XG boost and Ada Boost are having the highest accuracy based on our dataset and drift conditions. Whereas Logistic Regression is having the least accuracy. This chart is programmatically generated using the python program. The detailed experimentation outcome

of Accuracy along with Precision, Recall, F1-score and ROC-AUC for the same specifications are given in Table 4.4.1.

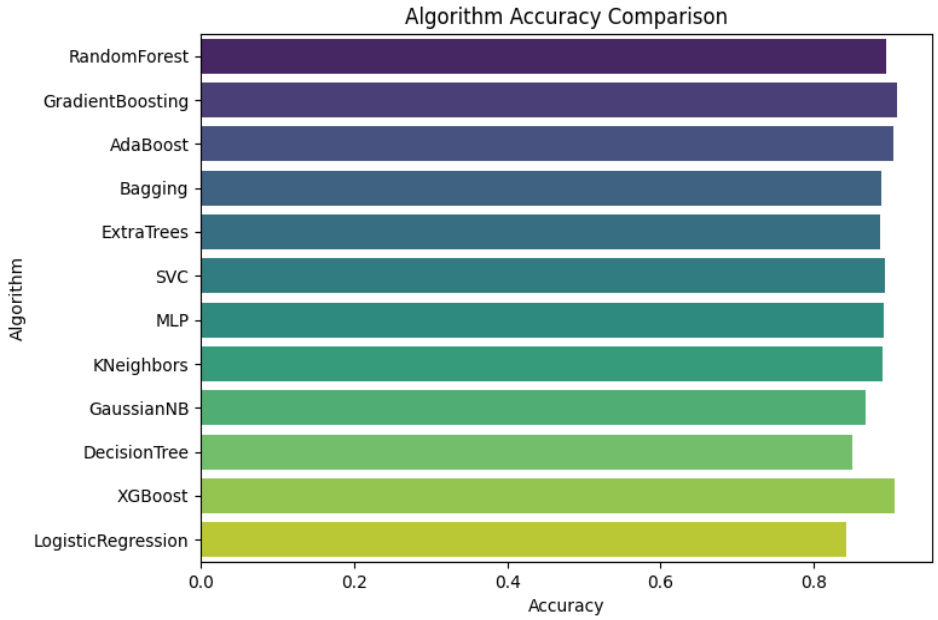


Figure 4.4.1: Algorithm accuracy comparison

Classifier	Accuracy	Precision	Recall	F1-score	ROC-AUC
RandomForest	0.8941	0.916894665	0.766657135	0.835072419	0.864644883
GradientBoosting	0.9092	0.999614049	0.74063483	0.850854139	0.870240527
AdaBoost	0.9049	0.975710015	0.746639977	0.845942006	0.868322295
Bagging	0.8876	0.895631877	0.768086932	0.826970443	0.85997765
ExtraTrees	0.8864	0.889732585	0.770660566	0.825927061	0.859649828
SVC	0.8928	0.939152481	0.741492708	0.828699265	0.857829239
MLP	0.8923	0.909891599	0.768086932	0.832997364	0.863591367
KNeighbors	0.8904	0.910708177	0.761223906	0.829283489	0.860544292
GaussianNB	0.8673	0.846867008	0.757506434	0.799698113	0.841924061
DecisionTree	0.8497	0.778180804	0.797540749	0.787741844	0.83764474
XGBoost	0.9063	0.9826546	0.74521018	0.847617499	0.869068261
LogisticRegression	0.8428	0.79389313	0.743494424	0.76786769	0.819848088

Table 4.4.1: Specifications and its result

Heat-Map for the experiments result:

Heatmap showing the confusion matrix which is providing detailed breakdown of the model's performance showing number of true positives, true negatives, false positives and false negatives. Each cell in the heatmap represents datapoints for each category. We have plotted separate heat map for each specification programmatically. These are shown in the figures 4.4.2 (a) to 4.4.2(l)

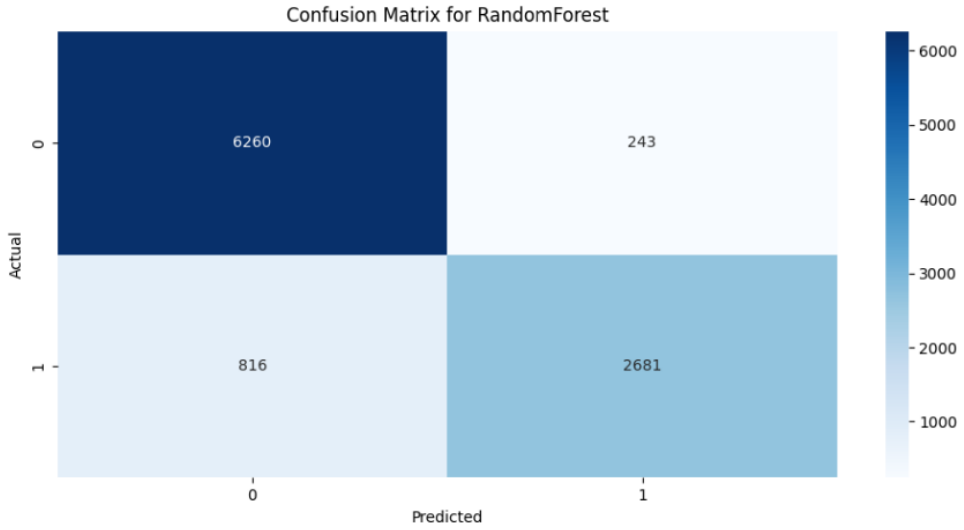


Figure 4.4.1 (a) Confusion matrix for Random Forest

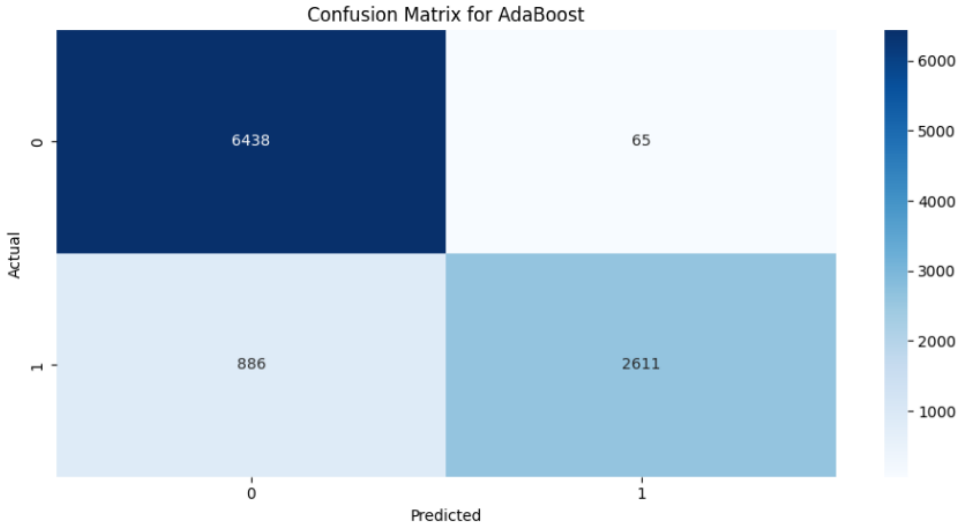


Figure 4.4.2 (b) Confusion matrix for Ada Boost

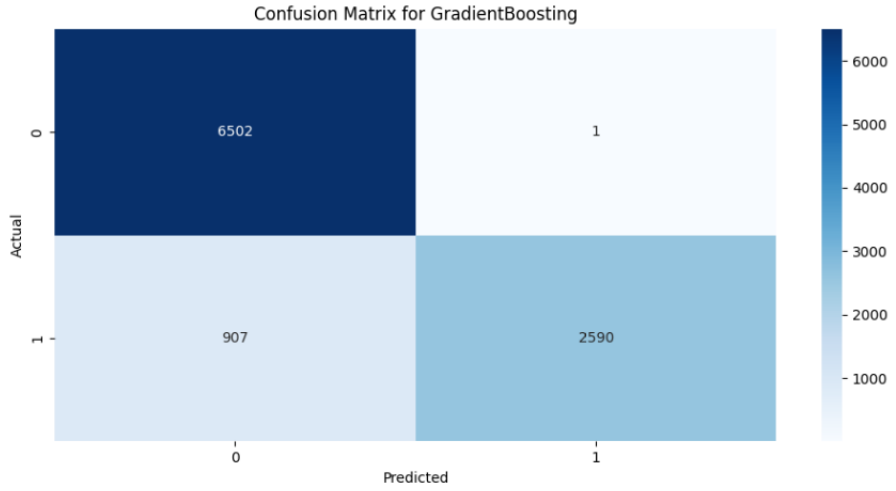


Figure 4.4.3 (c) Confusion matrix for Gradient Boosting

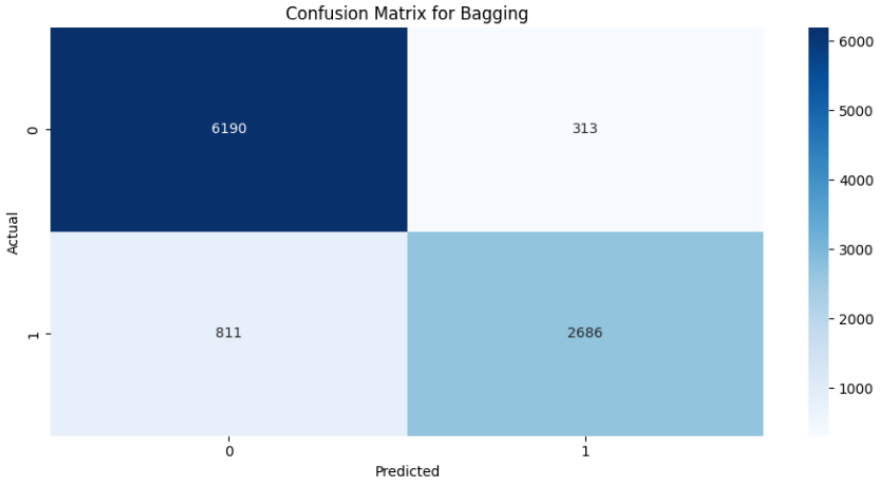


Figure 4.4.4 (d) Confusion matrix for Bagging

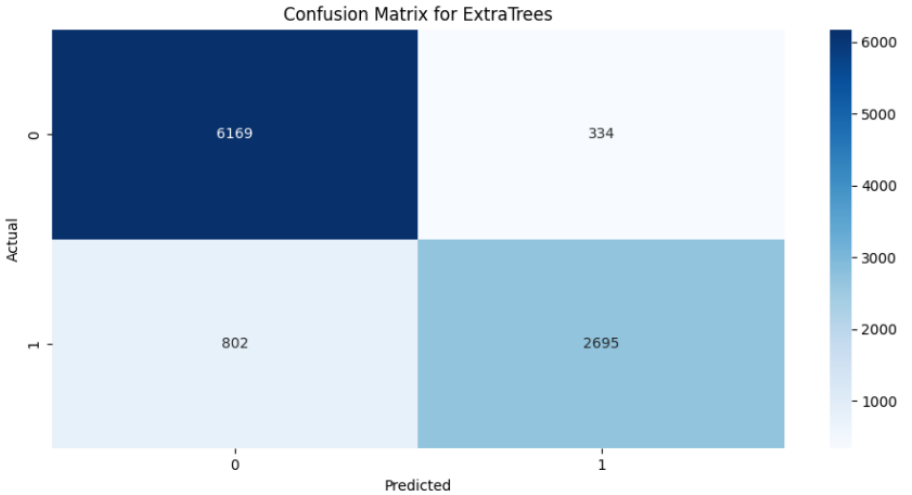


Figure 4.4.5 (e) Confusion matrix for Extra Trees

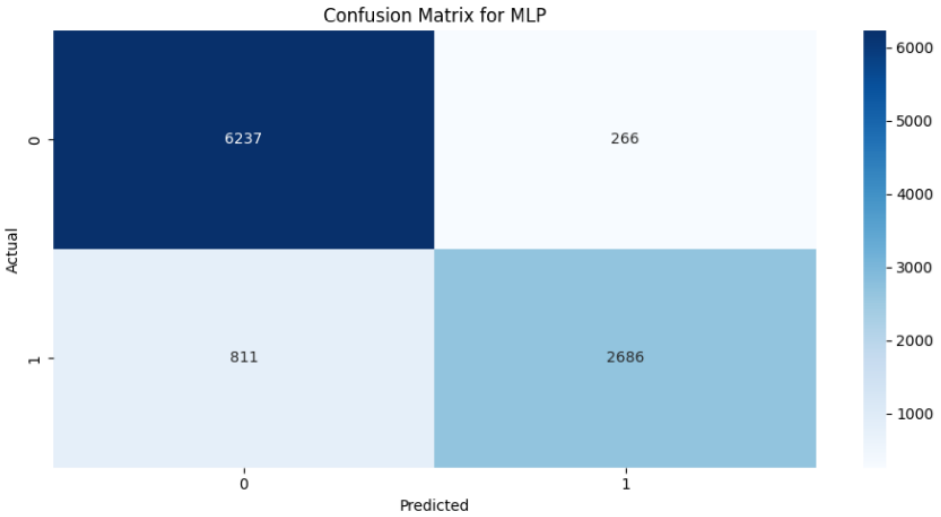


Figure 4.4.6 (f) Confusion matrix for MLP

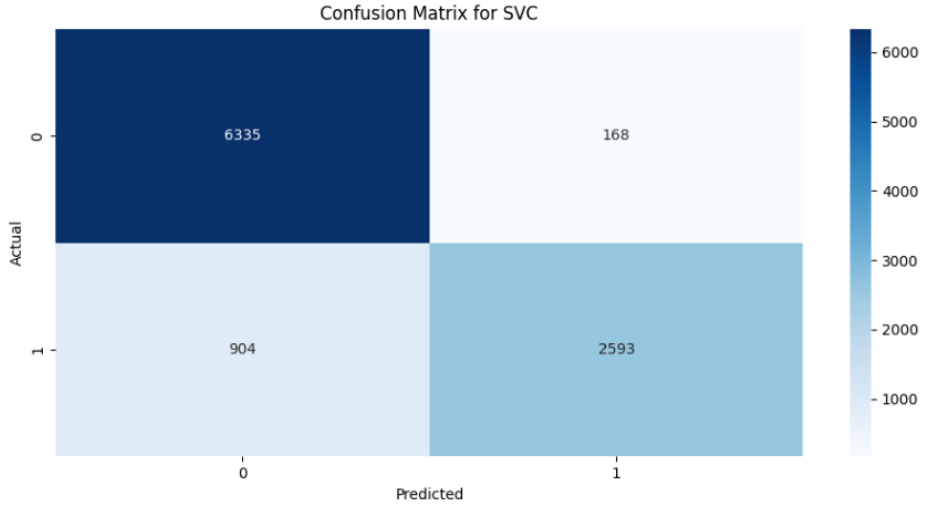


Figure 4.4.7 (g) Confusion matrix for SVC

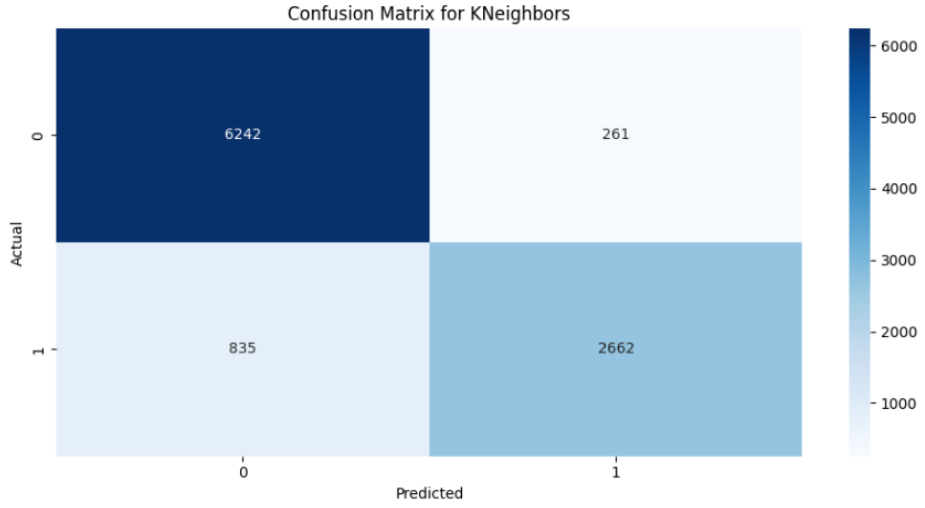


Figure 4.4.8 (h) Confusion matrix for K-Neighbours

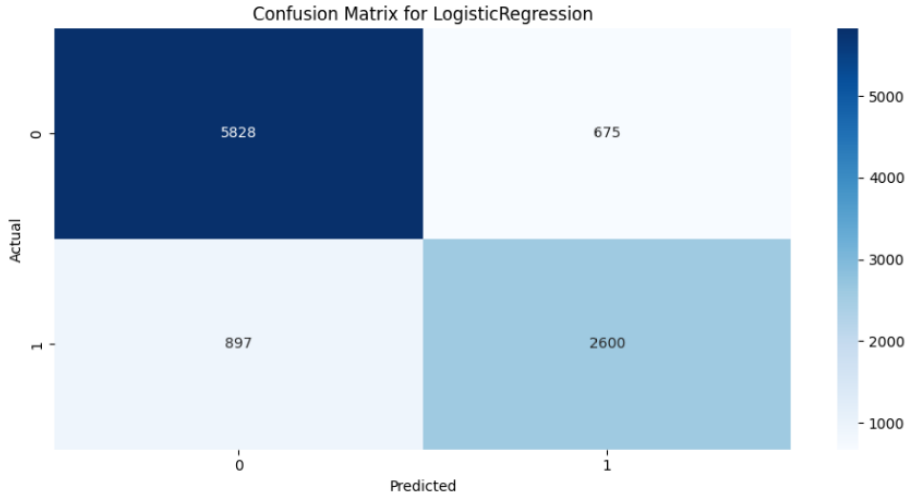


Figure 4.4.9 (i) Confusion matrix for Logistic Regression

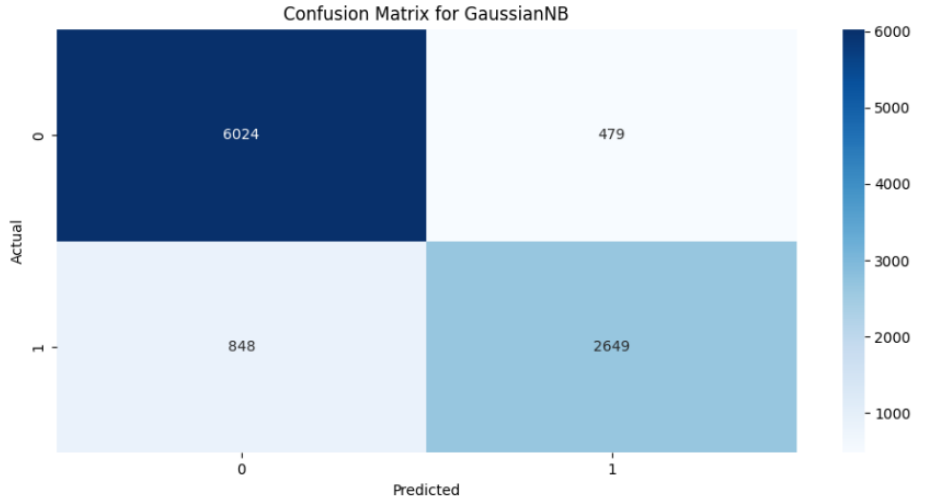


Figure 4.4.10 (j) Confusion matrix for Guassian NB

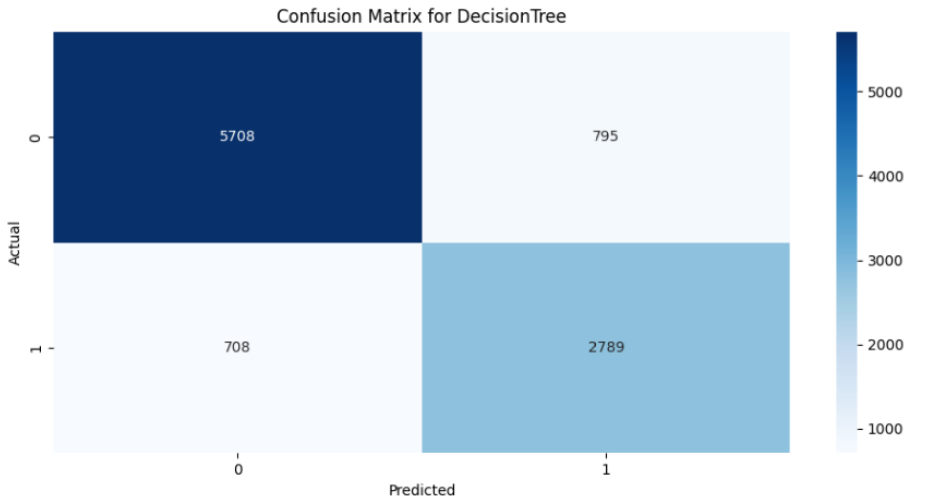


Figure 4.4.11 (k) Confusion matrix for Decision Tree

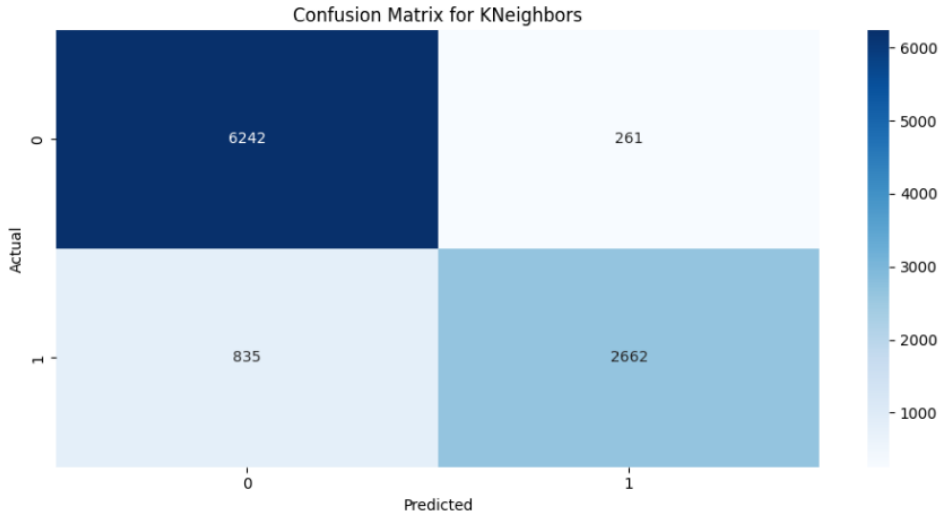


Figure 4.4.12 (l) Confusion matrix for Bagging

From the above figures, we can see the false negative, false positive, true negative and true positives values for each specification.

4.4.2 Unsupervised Learning

A Graphs of Data points per cluster for different unsupervised learnings are plotted and shown in the figure 4.4.3.1 to 4.4.3.4 showing 3 clusters and how they are plotted against the CPU utilization and RAM usage. 3 different colours depict datapoints for 3 clusters we have used in the unsupervised learning.

A consolidated chart is also plotted demonstrating the Silhouette score for each specification. This chart helps us identifying certain patterns in the drift. In K-Means, we have calculated the Silhouette score which comes at 0.7, for Isolation Forest it is 0.67, for One Class SVM its 0.64 whereas for PCA its 0.48. The chart is shown figure 4.4.3.5

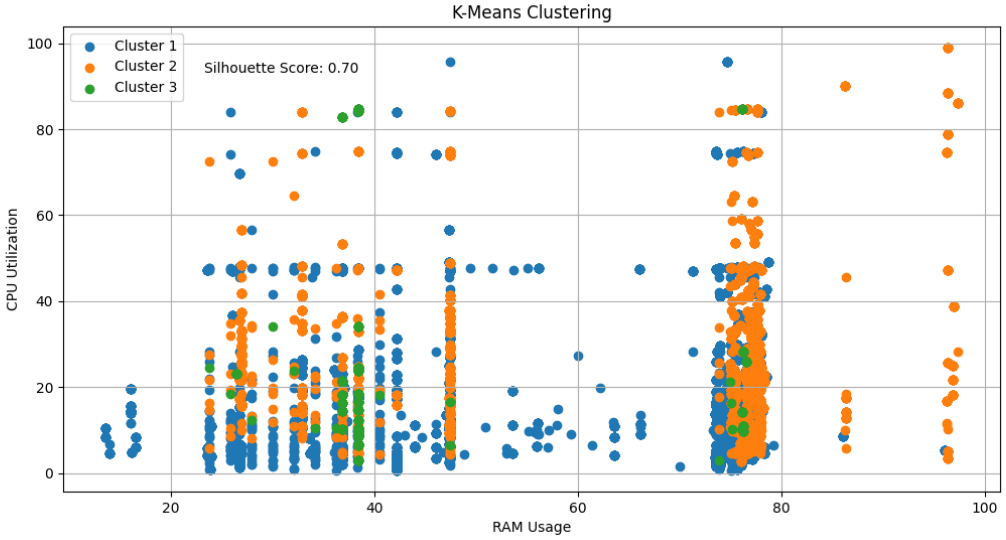


Figure 4.4.3.1: K-Means Clustering - Datapoints per cluster

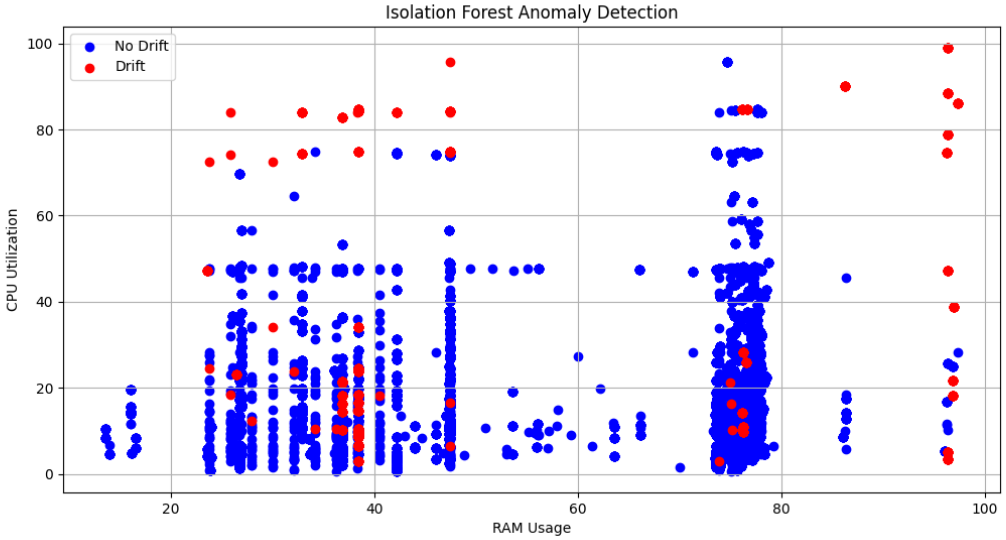


Figure 4.4.3.2: Isolation Forest - Datapoints per cluster

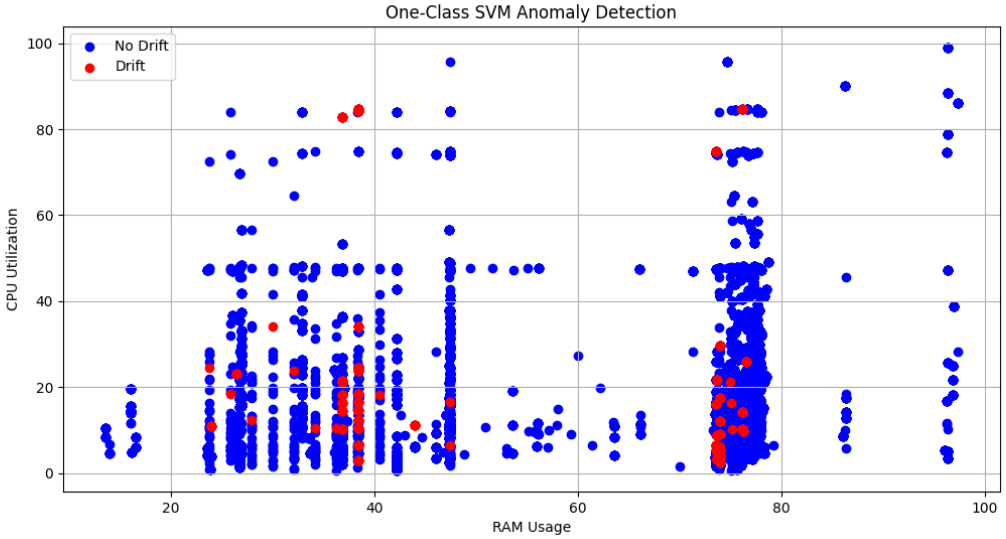


Figure 4.4.3.3: One Class SVM - Datapoints per cluster

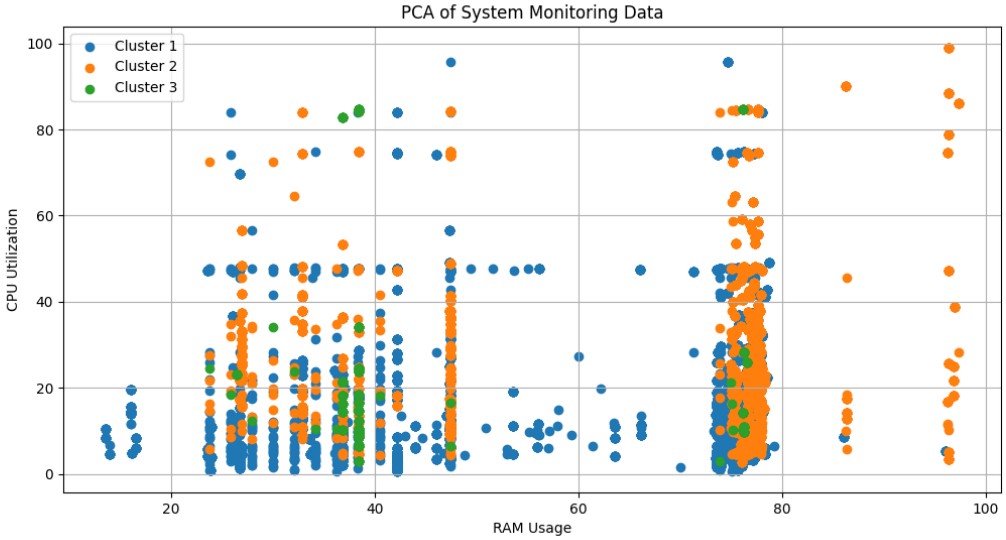


Figure 4.4.3.4: PCA of system monitoring data

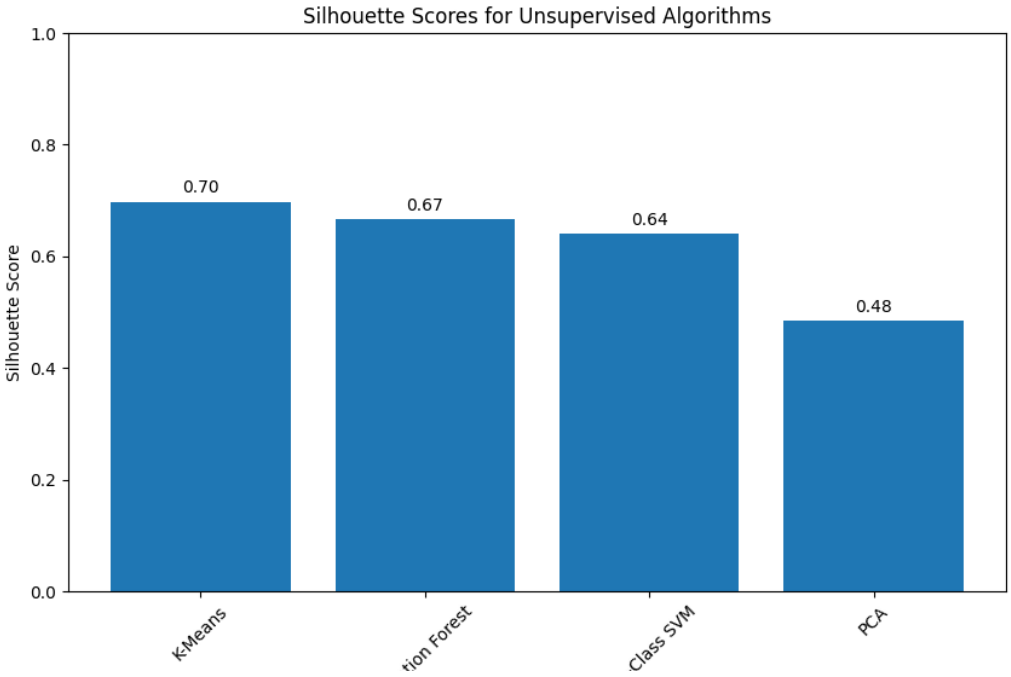


Figure 4.4.3.5: Silhouette Scores for Unsupervised Learning Algorithm

4.5 Limitations and Future Directions

While this thesis has made significant strides in the application of machine learning techniques to drift detection in legacy software systems, several limitations and avenues for future research merit consideration.

1. Limited Availability of Labeled Data: One of the primary limitations encountered in this research is the requirement for labeled data in supervised machine learning approaches. In many practical scenarios, obtaining labeled data that accurately represents all possible forms of drift can be challenging. Future research could explore techniques for semi-supervised or transfer learning to mitigate this limitation, allowing for more robust drift detection models in scenarios with limited labeled data.
2. Sensitivity to Feature Engineering: The effectiveness of machine learning models for drift detection often relies on feature engineering, which can be a time-consuming and domain-specific process. Future work could investigate automated feature selection and extraction techniques tailored to legacy software systems to reduce the burden of manual feature engineering.
3. Scalability: The scalability of the proposed machine learning models may be constrained in large-scale legacy systems. Future research could delve into distributed and online learning approaches to address scalability issues and enable real-time drift detection for extensive software systems.
4. Evaluation Metrics: The choice of evaluation metrics for drift detection can significantly impact the assessment of model performance. Future investigations could explore the development of domain-specific metrics that better align with the practical consequences of false positives and false negatives in software drift detection.
5. Integration with DevOps Practices: Integrating drift detection into DevOps practices remains an open challenge. Future research should focus on developing seamless integration frameworks and automated pipelines that allow for continuous monitoring and adaptation of legacy software systems in real-world, dynamic environments.
6. Ethical Considerations: As with any data-driven approach, ethical considerations regarding the collection and use of data must be addressed. Future research should delve into the ethical implications of drift detection in legacy software systems, particularly when dealing with sensitive or personal information.

4.6 Practical Implications

The research presented in this thesis holds significant practical implications for the management and maintenance of legacy software systems in contemporary software engineering practices.

1. Improved Drift Detection: The primary practical implication of this research is the development of more effective drift detection methods. By leveraging machine learning techniques, organizations can proactively identify and address drift in legacy software systems. This capability enhances system reliability, reduces downtime, and minimizes the risk of unexpected failures, leading to improved user satisfaction and reduced operational costs.
2. Resource Optimization: Implementing automated drift detection can lead to resource optimization. By detecting drift early, organizations can allocate resources more efficiently, focusing on critical areas of their legacy systems that require attention, rather than performing extensive, time-consuming manual inspections or upgrades.
3. Enhanced Decision Support: The machine learning models proposed in this thesis offer decision support capabilities. They enable software maintenance teams to make informed decisions regarding when and how to intervene in response to detected drift. This facilitates a more systematic and data-driven approach to software maintenance and evolution.
4. Seamless Integration into DevOps: As organizations increasingly adopt DevOps practices, integrating drift detection into their CI/CD pipelines becomes essential. This research paves the way for the seamless integration of machine learning-based drift detection tools into DevOps workflows. This integration enhances the ability to continuously monitor and adapt legacy software systems, aligning with the principles of continuous integration and continuous delivery.
5. Cost Savings: By proactively identifying drift and addressing it before it leads to system failures, organizations can realize substantial cost savings. The reduction in unplanned downtime, emergency maintenance, and potential revenue losses due to software failures can have a significant positive impact on the bottom line.
6. Strategic Legacy Software Management: The insights gained from this research enable organizations to adopt a more strategic approach to legacy software management. Rather than viewing legacy systems as burdensome liabilities, they can be seen as valuable assets that can be maintained and evolved with confidence.

7. Compliance and Security: Automated drift detection also has implications for compliance and security. By continuously monitoring software systems for drift, organizations can better ensure compliance with industry standards and regulations and enhance their ability to detect security vulnerabilities and respond promptly to mitigate risks.

The practical implications of this research extend to various facets of software engineering and organizational management. By embracing machine learning-based drift detection techniques, organizations can transform the way they approach legacy software systems, unlocking new opportunities for efficiency, reliability, and strategic decision-making in the maintenance and evolution of their software assets.

CHAPTER 5: CONCLUSION AND RECOMMENDATION

5.1 Introduction

The culmination of this research journey marks a significant impact in the field of software maintenance and evolution, particularly in the context of legacy software systems. Through a meticulous exploration of machine learning techniques for drift detection, this study has shed light on crucial strategies to fortify the resilience of aging software infrastructures. As outlined in earlier chapters, legacy software systems are the bedrock of numerous organizations, underpinning critical operations and services. However, their vulnerability to drift necessitates sophisticated methodologies for early detection and intervention. This research has delved into the application of both supervised and unsupervised machine learning algorithms, providing a comprehensive comparative analysis of their efficacy in identifying and quantifying drift.

In this concluding chapter, we synthesize the key findings, highlight the implications for practice, and propose recommendations for future research and practical implementation. The insights garnered from this study not only advance the theoretical understanding of drift detection but also offer actionable guidance for organizations seeking to bolster the resilience and longevity of their legacy software assets.

5.2 Discussion and Conclusion

5.2.1 Discussion

In this thesis, we addressed the critical issue of drift detection in legacy software systems using machine learning techniques. Legacy systems often pose unique challenges due to their complex and aging nature. We discussed the significance of detecting drift in such systems to ensure their continued functionality, security, and maintainability. The study highlighted those changes in data distributions, system configurations, and usage patterns over time can lead to drift, which can negatively impact the performance and reliability of legacy software.

The literature review provided a comprehensive overview of drift detection techniques, focusing on both supervised and unsupervised learning approaches. We explored the applicability of these methods in the context of legacy software systems. Supervised learning techniques require labeled data, which may be challenging to obtain in the context of legacy systems. On the other hand, unsupervised learning methods, particularly clustering and dimensionality reduction techniques, show promise in detecting drift without the need for labeled data.

The research methodology section detailed the process of data collection, pre-processing, feature extraction, and algorithm selection. We evaluated both supervised and unsupervised learning algorithms using appropriate evaluation metrics. Analysis over supervised learnings shows that algorithms like Gradient Boosting, Ada Boost and XG Boost are showing higher accuracy. The results showed that certain unsupervised learning techniques, such as clustering, and Isolation Forest and One Class SVMs performed well in detecting drift in legacy software systems. Heat maps were used to visualize the experiment results, providing valuable insights into algorithm performance.

5.2.2 Conclusion

Achieving the Research Objectives:

This thesis aimed to address the problem of drift detection in legacy software systems using machine learning techniques. Through a comprehensive exploration of relevant literature and the implementation of various algorithms, we have made progress toward achieving our research objectives. We have demonstrated that unsupervised learning methods, particularly clustering and PCA, show promise in detecting drift in legacy systems without the need for labeled data. We have also used supervised learnings and compared the accuracy of each algorithm which is helpful in selecting the appropriate machine learning algorithm for drift detection.

Limitations and Future Directions

Despite the promising results, there are limitations to this research. The evaluation was conducted on a specific dataset, and the generalizability of the findings to different legacy systems should be further investigated. Additionally, the choice of evaluation metrics can impact the interpretation of results, and future work should explore alternative metrics and their implications.

Future directions for research include the development of adaptive drift detection algorithms that can dynamically adjust to changes in legacy systems, as well as the exploration of ensemble techniques to enhance detection accuracy. Moreover, the practical implications of implementing drift detection mechanisms in real-world legacy systems should be further examined, including the challenges and considerations involved in deployment.

REFERENCES

Abbasi, A., Javed, A.R., Chakraborty, C., Nebhen, J., Zehra, W. and Jalil, Z., (2021) ElStream: An Ensemble Learning Approach for Concept Drift Detection in Dynamic Social Big Data Stream Learning. *IEEE Access*, 9, pp.66408–66419.

Albuquerque, A.B. and Cruz, V.L., (2019) Implementing DevOps in legacy systems. In: *Advances in Intelligent Systems and Computing*. Springer Verlag, pp.143–161.

Alonso, J., Orue-Echevarria, L., Osaba, E., López Lobo, J., Martinez, I., Diaz de Arcaya, J. and Etxaniz, I., (2021) Optimization and prediction techniques for self-healing and self-learning applications in a trustworthy cloud continuum. *Information (Switzerland)*, 128.

Anon (2021) *Improving La Redoute's CI/CD Pipeline and DevOps Processes by Applying Machine Learning Techniques*. [online] Available at: www.jetir.org.

Barros, R.S.M. de and Santos, S.G.T. de C., (2019) An overview and comprehensive comparison of ensembles for concept drift. *Information Fusion*, 52, pp.213–244.

Barros, R.S.M. and Santos, S.G.T.C., (2018) A large-scale comparison of concept drift detectors. *Information Sciences*, 451–452, pp.348–370.

Casado, F.E., Lema, D., Criado, M.F., Iglesias, R., Regueiro, C. V. and Barro, S., (2022) Concept drift detection and adaptation for federated and continual learning. *Multimedia Tools and Applications*, 813, pp.3397–3419.

Castellani, A., Schmitt, S. and Hammer, B., (2021) Task-Sensitive Concept Drift Detector with Constraint Embedding. In: *2021 IEEE Symposium Series on Computational Intelligence, SSCI 2021 - Proceedings*. Institute of Electrical and Electronics Engineers Inc.

Charbuty, B. and Abdulazeez, A., (2021) Classification Based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, 201, pp.20–28.

Dang, Y., Lin, Q. and Huang, P., (2019) AIOps: Real-world challenges and research innovations. In: *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*. Institute of Electrical and Electronics Engineers Inc., pp.4–5.

Duda, P., Przybyszewski, K. and Wang, L., (2020) A Novel Drift Detection Algorithm Based on Features' Importance Analysis in a Data Streams Environment. *Journal of Artificial Intelligence and Soft Computing Research*, 104, pp.287–298.

Escovedo, T., Koshiyama, A., da Cruz, A.A. and Vellasco, M., (2018) DetectA: abrupt concept drift detection in non-stationary environments. *Applied Soft Computing Journal*, 62, pp.119–133.

Fahmideh, M., Daneshgar, F., Beydoun, G. and Rabhi, F., (n.d.) *Challenges in migrating legacy software systems to the cloud-an empirical study*.

Gholami, M.F., Daneshgar, F., Beydoun, G. and Rabhi, F., (2017) Challenges in migrating legacy software systems to the cloud — an empirical study. *Information Systems*, 67, pp.100–113.

Han, X., Jiang, J., Xu, A., Bari, A., Pei, C. and Sun, Y., (2020) Sensor drift detection based on discrete wavelet transform and grey models. *IEEE Access*, 8, pp.204389–204399.

Hrusto, A., Runeson, P. and Engström, E., (2021) Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations. *SN Computer Science*, 26.

Hrusto, Adha. and Lunds universitet., (n.d.) *Towards Optimization of Anomaly Detection Using Autonomous Monitors in DevOps*.

Ibrahim, D.R., Ghnemat, R. and Hudaib, A., (2017) Software defect prediction using feature selection and random forest algorithm. In: *Proceedings - 2017 International Conference on New Trends in Computing Sciences, ICTCS 2017*. Institute of Electrical and Electronics Engineers Inc., pp.252–257.

Jain, M., Kaur, G. and Saxena, V., (2022) A K-Means clustering and SVM based hybrid concept drift detection technique for network anomaly detection. *Expert Systems with Applications*, 193.

Koroglu, Y., Sen, A., Kutluay, D., Bayraktar, A., Tosun, Y., Cinar, M. and Kaya, H., (2016) Defect prediction on a legacy industrial software: A case study on software with few defects. In: *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, pp.14–20.

Kreuzberger, D., Kühl, N. and Hirschl, S., (2023) Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*, 11, pp.31866–31879.

Lewis, G.A., Echeverria, S., Pons, L. and Chrabaszcz, J., (2022) Augur: A Step Towards Realistic Drift Detection in Production ML Systems. In: *Proceedings - Workshop on Software Engineering for Responsible AI, SE4RAI 2022*. Institute of Electrical and Electronics Engineers Inc., pp.37–44.

Lu, N., Lu, J., Zhang, G. and Lopez De Mantaras, R., (2016) A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, 230, pp.108–133.

Magal, K.R., Gracia Jacob, S. and Professor, A., (2015) *Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques*. *International Journal of Computer Applications*, .

de Mello, R.F., Vaz, Y., Grossi, C.H. and Bifet, A., (2019) On learning guarantees to unsupervised concept drift detection on data streams. *Expert Systems with Applications*, 117, pp.90–102.

Meng, F.J., Wegman, M.N., Xu, J.M., Zhang, X., Chen, P. and Chafle, G., (2017) *IT troubleshooting with drift analysis in the DevOps era*. *IBM Journal of Research and Development*, .

Morris, K., Farnham, B., Tokyo, S., Boston, B., Sebastopol, F. and Beijing, T., (n.d.) *Infrastructure as Code Dynamic Systems for the Cloud Age SECOND EDITION*.

Nogueira, A.F., Ribeiro, J.C.B., Zenha-Rela, M. and Craske, A., (2018) Improving la redoute's CI/CD pipeline and devops processes by applying machine learning techniques. In: *Proceedings - 2018 International Conference on the Quality of Information and Communications Technology, QUATIC 2018*. Institute of Electrical and Electronics Engineers Inc., pp.282–286.

Priyam, A., Gupta, R., Rathee, A. and Srivastava, S., (2013) *Comparative Analysis of Decision Tree Classification Algorithms*. [online] Available at: http://inpressco.com/category/ijcet.

Saidani, I., Ouni, A. and Mkaouer, M.W., (2022) Improving the prediction of continuous integration build failures using deep learning. *Automated Software Engineering*, 291.

Sethi, T.S. and Kantardzic, M., (2017) On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82, pp.77–99.

Siraskar, R., Kumar, S., Patil, S., Bongale, A. and Kotecha, K., (2023) Reinforcement learning for predictive maintenance: a systematic technical review. *Artificial Intelligence Review*.

Talapula, D.K., Kumar, A., Ravulakollu, K.K. and Kumar, M., (2023) A hybrid deep learning classifier and Optimized Key Windowing approach for drift detection and adaption. *Decision Analytics Journal*, 6.

Yu, S., Abraham, Z., Wang, H., Shah, M., Wei, Y. and Príncipe, J.C., (2019) Concept drift detection and adaptation with hierarchical hypothesis testing. *Journal of the Franklin Institute*, 3565, pp.3187–3215.

Zenisek, J., Holzinger, F. and Affenzeller, M., (2019) Machine learning based concept drift detection for predictive maintenance. *Computers and Industrial Engineering*, 137.

Refer: Harvard Referencing Guide

APENDEX A: RESEARCH PROPOSAL

1. Background

Legacy software systems are important for organizations because they perform critical business functions, hold valuable data or knowledge, require specialized expertise, and may provide a higher level of security and compliance. However, these systems may also pose challenges, including the need for maintenance and support, difficulty in modification, and compatibility issues with newer technologies (Albuquerque and Cruz, 2019). Thus, it becomes important to maintain the stability and performance of such systems (Meng et al., 2017). There are various tools and services to effectively manage the applications which are based on the latest technology stacks but there are limitations when dealing with legacy applications. To keep working with such legacy software systems as it's difficult to completely replace them with new ones, organizations invest considerable efforts to enhance these existing systems. While doing such enhancements and improving the software environments with upgraded software and hardware needs a lot of configurations and tuning. While doing this, applications are at greater risk of failures, and downtimes and are prone to defects and unexpected behaviors. This costs organizations heavily. So, it's important to avoid such problems. The drift detection process in DevOps can significantly reduce such incidences by predicting the problems before it occurs and helps developers and DevOps practitioners to analyze the problem quickly(Escovedo et al., 2018).

There are different ways of finding the drifts in the systems. These processes involve reading the software and hardware configurations, storing the blueprint data which is the ideal state of these configurations, and comparing it from time to time, based on any changes and on an ad-hoc basis (Morris et al., n.d.). This requires system monitoring at a deeper level with the large amount of data which also costs in terms of performance and resources. Machine learning solutions were found to be effective in this data analysis process (Talapula et al., 2023).

2. Problem Statement

There are studies conducted on finding and analyzing the drift in software systems (Talapula et al., 2023). The analysis of drift data helps in predicting the problems in the software systems which indeed helps to minimize the impact of malfunctioning and system downtime by proactive maintenance. The drift details are obtained by monitoring the various parameters including hardware and software configurations and runtime system properties. There are various ways for comparing and analyzing these details however there are limitations while monitoring a large set of parameters and it involves heavy computations and needs more system resources. Monitoring and predicting the drift using ML techniques such as random forests and decision trees can help perform this analysis faster and more accurately (Nogueira et al., 2018; Improving La Redoute's CI/CD Pipeline and DevOps Processes by Applying Machine Learning Techniques, 2021). The use of drift detection techniques with better performance and accuracy can largely improve the DevOps processes (Saidani et al., 2022).

Not much study has been conducted on using machine learning algorithms for drift detection in legacy systems which could help us compare and use effective identification of the drift.

The term legacy system first appeared in IT literature in 1990 meaning "relating to or being a previous or outdated computer system" (Gholami et al., 2017). Several alternative definitions are available. For instance, a previous description characterizes them as "significant software systems that present challenges for us to handle, yet they play a crucial role in our organization." In this research, the drift detection with machine learning techniques will be studied in legacy systems to compare and propose the effective drift detection to improve system stability and functioning with a lesser number of defects because of incorrect software and hardware configuration in legacy systems (Alonso et al., 2021).

For different types of drift, different sort of data is required to be collected from system monitoring and processed. The most efficient way of processing the data is different for different formats of data. The process of identifying the drift from such data can be improved with the help of various machine-learning algorithms based on the use cases.

3. Research Questions

To explore how machine learning algorithms can be used in finding the drift in the legacy system so that the drift detection process can be improved to make the DevOps process more robust and with less chance of defects occurring because of incorrect environment properties.

4. Aim and Objectives

The aim of this research paper is to study and analyze the use of machine learning techniques to improve the drift detection process in DevOps.

The research objectives formulated based on the aim of this study are as follows:

- To investigate the use of machine learning algorithms for predicting drift in software systems.
- To analyze with the help of experimental evaluation, how ML algorithms help in improving identification of drifts and improve the DevOps process.
- To propose a DevOps process with the introduction of an improved drift detection phase.

5. Significance of the Study

There's not been much research conducted on the significance of machine learning algorithms for drift detection in legacy systems. The effective and detailed analysis of the Machine Learning algorithm for drift detection can help improve the system stability and usability by detecting inaccurate system configurations and eliminating the defects in a short period of time (Saidani et al., 2022). This research will emphasize the application of Machine Learning techniques for identifying drift within legacy systems.

6. Scope of the Study

The study is focused on legacy software systems. Since the complex software environment comprises of a large amount of data that can be monitored for drifts, multiple servers, and applications would require. For simplicity, we will use the limited servers and applications for demonstrating the use of machine learning techniques. The same techniques can then be applied on bigger environments. We are not considering the system monitoring in this study and expect to use existing monitoring data.

7. Research Methodology

The research will be based on quantitative analysis. The datasets required for this research will be based on the environment configurations and software configurations used for the sample legacy software applications. The research will be conducted on the use of machine learning algorithms on such datasets to detect the drift as early as possible to avoid critical system issues and unexpected behaviors. As the drift can exist in different areas related to the software, different formats of data are expected. Thus, different algorithms will be analyzed based on different datasets and the use case.

The primary objective of this study is to examine various machine learning techniques and propose an effective approach for detecting drifts in legacy software systems. This will help to improve the DevOps process by identifying the drift earlier to help improve the system stability (Zenisek et al., 2019). The proposed technique mainly consists of two modules: offline learning module and incremental learning module (Talapula et al., 2023). The diagrammatic representation of such a model for detecting drift is shown in Fig. 1.

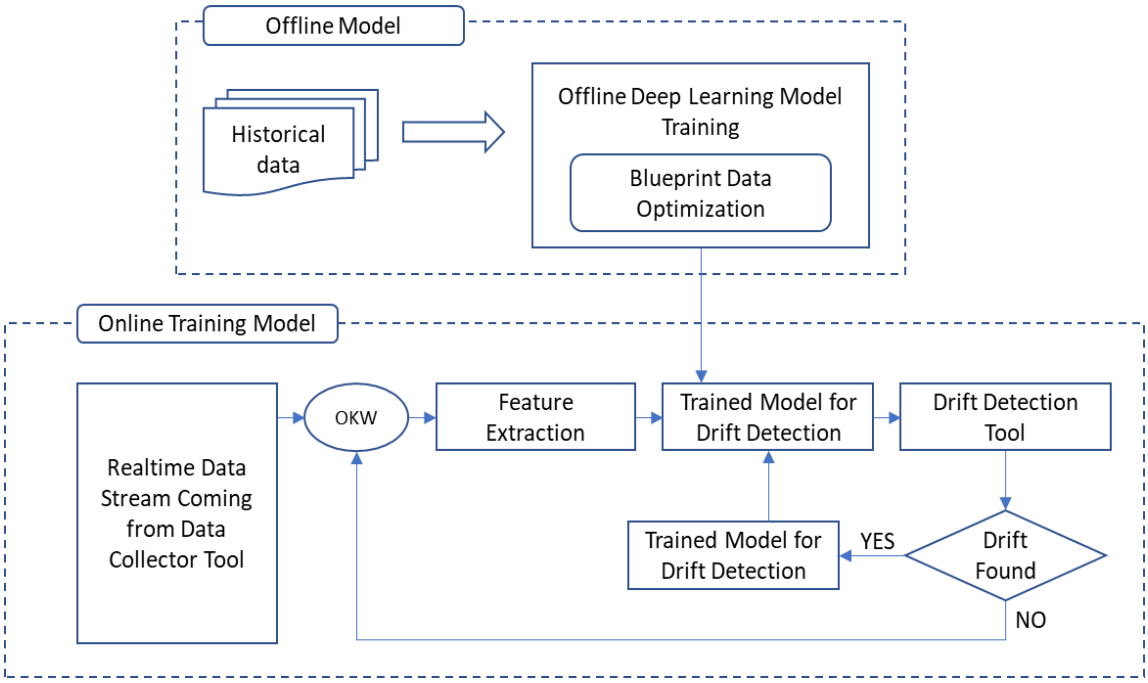


Fig. 1. Diagrammatic representation of Machine learning based drift detection model

7.1 Data Collection

In this research, the following types of datasets will be considered:

1. The actual values of the legacy software environments which are in effect and deciding the system behavior. Example:
    - The hardware configurations on which the entire system is working
    - Software configurations such as configuration files and properties files are used to configure the environments and applications within it.

   Any change in these properties can lead to unexpected system behavior and may lead to the crashing of the entire software environment as well.
2. The expected data or the blueprint data which depicts the ideal values that every type of configuration will have. This data may contain a single value or range of values for each configuration.
3. Standard datasets generated from Apache server, Linux, and monitoring tools output.

Data will be obtained from the data collector tool which will read the runtime properties and configuration files from the software environment.

7.2 Research Approach

As a part of this research, two applications will be developed to support the study.

1. **Data Collector:**
   This tool will be responsible to read the software environment properties such as hardware configurations, environment & application configuration files.
2. **Drift Detection Tool:**
   This will employ the prediction model to predict the drift using various algorithms. The output will be used to train the online machine-learning model.

In the offline learning phase, initial training data is developed along with the blueprint data. Blueprint data is the one that depicts the ideal values. Any deviation of real-time data from the blueprint data can be termed as a drift. The drift detection and blueprint data maintenance will happen in the incremental training phase. Optimized Key Windowing (OKW) is used in incremental training for drift detection in the data stream. If a drift is detected, the initial

classifier model is retrained using sample data obtained from OKW, based on the hypothesis. (Talapula et al., 2023).

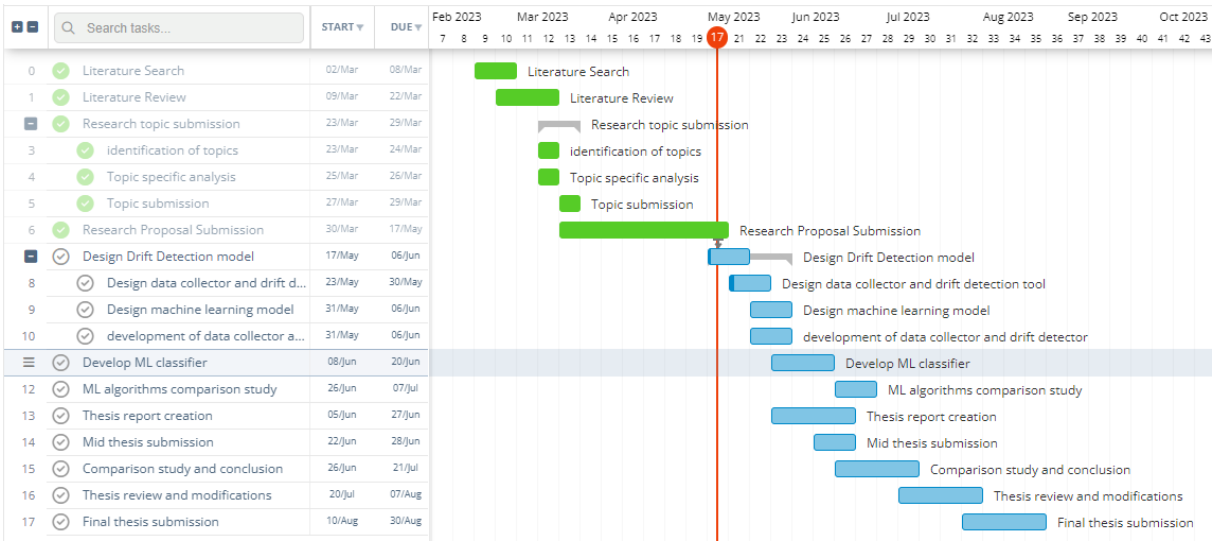
In the study of hybrid deep learning classification, we will compare the deep Recurrent Neural Networks (RNN) classifier and the deep Long Short-Term Memory (LSTM) classifier with the other machine learning classifier to propose an efficient way of detecting drift and improving the system stability.

The same dataset would be used and processed in a drift detection tool for various machine learning algorithms. A comparative study will be conducted based on the quantitative data obtained against each machine learning algorithm used. Based on the findings, an efficient method for drift detection on certain types of drift will be proposed.

8. Requirements Resources

Virtual Machines, sample legacy applications, MS Office, system monitoring tools, etc.

9. Research Plan



APENDIX B: PYTHON SOURCE CODE

Python code used for the experimentation is maintained at the git repository.

https://github.com/swapnilsworld/drift_detection.git