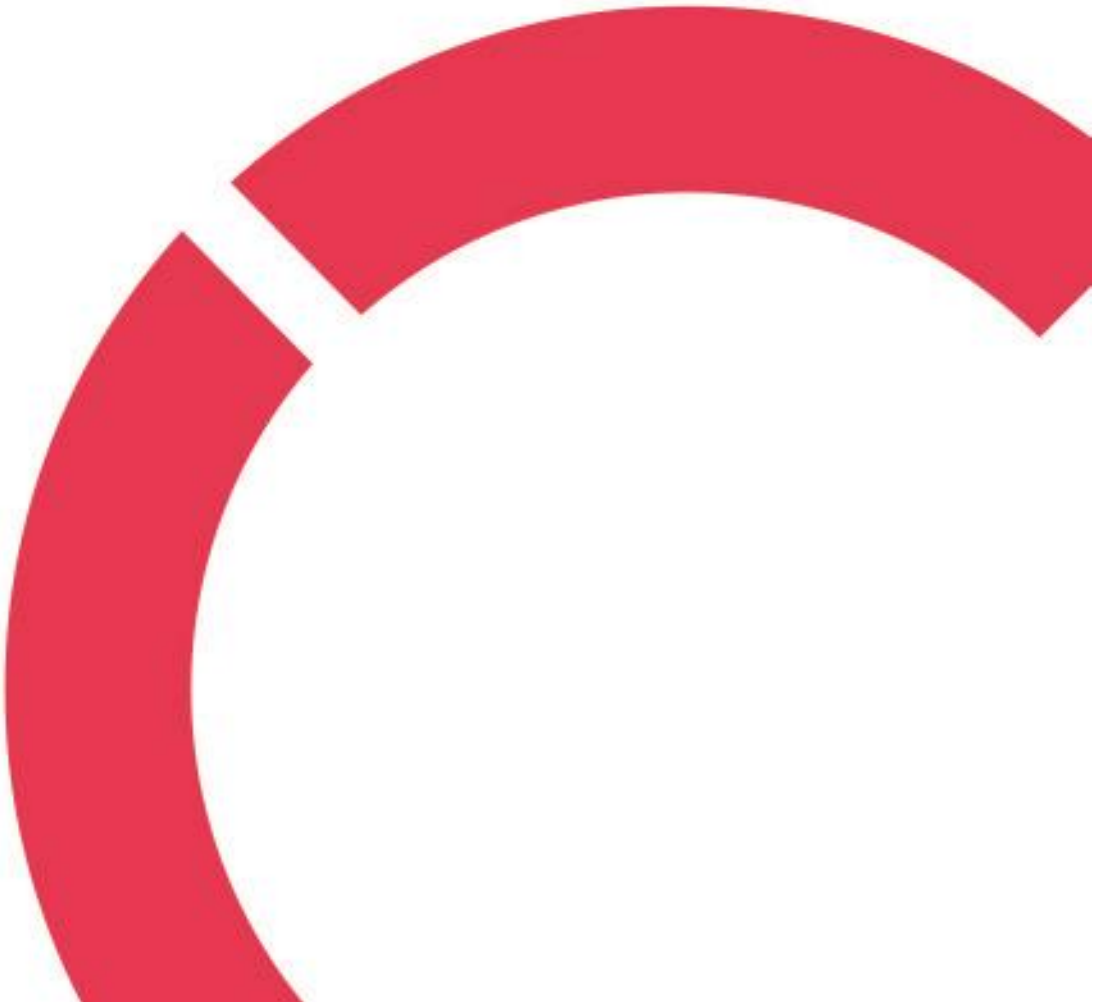
Birendra Khadka

ENHANCING CLOUD BASED SOFTWARE ENGINEERING WITH MACHINE LEARNING

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCE
The Master of Engineering, Information Technology
May 2024



ABSTRACT

Centria University of Applied Sciences	Date May 2024	Author Birendra Khadka
Degree programme The Master of Engineering, Information Technology		
Name of thesis ENHANCING CLOUD BASED SOFTWARE ENGINEERING WITH MACHINE LEARNING		
Centria supervisor Aliasghar Khavasi		**Pages** 29 + 2
Instructor representing commissioning institution or company		

With the ability to access virtualized resources on demand, cloud computing has developed into a game-changing technology that enables companies to adapt to changing market conditions and grow their infrastructure through dynamic expansion. In machine learning (ML), where computational resources might be very variable and in demand, this method is particularly helpful.

The main goal of this thesis is to streamline cloud-based software engineering processes by incorporating machine learning techniques. Machine learning algorithms are mostly used in cloud-based software development and deployment pipelines to automate tasks, improve decision-making, and optimize resource usage. The goal is to increase the effectiveness, stability, and scalability of cloud-based software systems by utilizing predictive analytics and data-driven insights.

Among the real-world examples presented in the research to show how machine learning may be applied to enhance speed, security, resource allocation, and service rollout are AWS, Netflix, Uber, and other well-known businesses. For cloud-based software engineering decision-making processes, these case studies provide useful insights.

The results of our tests show the potential benefits of integrating machine learning with intelligent decision-making and automation in a variety of domains. The results add to the growing body of knowledge on machine learning applications in software development and offer insightful information to both academics and industry practitioners.

CONCEPT DEFINITIONS

API	Application Programming Interface
AWS	Amazon Web Services
Azure	Microsoft Azure Cloud
CBSE	Cloud-Based Software Engineering
CDN	Content Delivery Network
CE	Cloud Engineering
CI/CD	Continuous Integration/Continuous Deployment
DaaS	Data as a Service
DNS	Domain Name System
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
IAM	Identity and Access Management
IaaS	Infrastructure as a Service
IoT	Internet of Things
ML	Machine Learning
MLaaS	Machine Learning as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
SDN	Software-Defined Networking
SOC	Service Organization Control
VM	Virtual Machine
VMs	Virtual Machines

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

Cloud computing, with its unparalleled scalability, flexibility, and access to computer resources, has completely changed software development, deployment, and maintenance. Simultaneously, machine learning has become a potent instrument for extracting insights from data, facilitating astute decision-making and automation in a multitude of domains. The incorporation of machine learning into cloud-based software engineering offers a noteworthy prospect to augment conventional software development methodologies through automation and data-driven insights.

Historically, requirements analysis, design, testing, and maintenance in software engineering have been done by hand using heuristics and manual labor. More effective and scalable methods are, nonetheless, desperately needed as contemporary software systems become more complicated and large-scale. These objectives are satisfied by cloud computing, which gives businesses on-demand access to virtualized resources so they can dynamically grow their infrastructure and adjust to changing requirements. Notwithstanding these developments, there are still issues with guaranteeing the caliber and dependability of cloud-based apps and streamlining software engineering procedures.

This thesis addresses the research challenge of improving cloud-based software engineering processes by using machine learning methods. In particular, the goal is to optimize resource consumption, enhance decision-making, and automate processes in cloud-based software development and deployment pipelines using machine learning techniques. Enhancing the efficacy, dependability, and expandability of cloud-based software systems is the aim through the utilization of predictive analytics and data-driven comprehension.

This study's main goal is to investigate the possible advantages of combining cloud-based software engineering techniques with machine learning. Finding ways to integrate machine learning techniques into different stages of the software development lifecycle and creating and assessing machine learning models for automated testing in cloud-based environments, defect prediction, and resource optimization are important objectives. The study also evaluates how machine learning integration affects user happiness, system performance, and development efficiency in practical situations.

By using machine learning approaches, this project seeks to promote cloud-based software engineering processes with an emphasis on enhancing system performance and development efficiency. The research encompasses a number of software engineering disciplines, including as design, implementation, testing, deployment, and maintenance, in addition to requirements analysis. By addressing these challenges, this research aims to contribute to the advancement of cloud-based software engineering practices and pave the way for more intelligent and adaptive software systems.

The structure of the thesis is as follows: Chapter 2 provides a literature review on cloud-based software engineering, machine learning techniques in software engineering, and previous studies on integrating machine learning with cloud-based systems. Chapter 3 outlines the methodologies used in this study, including data collection, preprocessing, machine learning algorithms, evaluation metrics, and the development of working flowcharts. Chapter 4 details the implementation of machine learning models in cloud-based software engineering environments, including prototype systems, and encountered challenges. Chapter 5 presents the findings of the study, including performance evaluation results, data insights, and visualizations. Chapter 6 showcases real-world applications of the integrated systems and demonstrates their effectiveness in practical scenarios. Chapter 7 discusses the implications of the findings, addresses challenges and limitations, and outlines future research directions. Finally, Chapter 8 summarizes the key findings, contributions, and implications of the study, and provides concluding remarks.

2 BACKGROUNDS AND LITLERATUE REVIEW

A paradigm changes in software development, cloud-based software engineering makes use of cloud computing technologies to improve scalability, accessibility, and cost-effectiveness. Simultaneously, software engineering has seen a rise in the incorporation of machine learning techniques, providing prospects for automation, optimization, and data-driven decision-making. This section reexamines the research on machine learning's potential to improve software development processes and cloud-based software engineering.

2.1 Overview of Cloud-Based Software Engineering

The term "cloud-based software engineering" describes how different stages of the software development lifecycle may be facilitated by utilizing cloud computing platforms and services. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) models are essential elements of cloud-based software architecture. These models give developers access to scalable computing resources, development tools, and application hosting environments. Collaborative development, continuous integration, and deployment automation are made possible by cloud-based development platforms, which shorten time-to-market and streamline software development procedures.

Improved resource usage, scalability, and cost-efficiency are just a few advantages of cloud-based software engineering that have been emphasized in earlier research. Rittinghouse and Ransome's (2016) research, for instance, showed how well cloud-based development environments work for distributed software development projects, allowing geographically separated teams to work together and share resources without any problems. Similar to this, (Chen, Zhang, Hu, Hussain, & Taleb, 2018) compared traditional and cloud-based development methodologies and came to the conclusion that cloud-based software engineering had a number of advantages, including cost savings, scalability, and flexibility.

2.2 Role of Machine Learning in Software Engineering

The issues of software engineering, such as defect prediction, code analysis, testing automation, and performance optimization, may now be effectively addressed with machine learning approaches. Machine learning algorithms may uncover trends, spot abnormalities, and provide predictions by analyzing vast amounts of software-related data. These predictions help engineers with decision-making and problem-solving.

Researchers have been investigating the use of machine learning in software engineering in a variety of disciplines in recent years. In order to identify potential software defects early in the development process, for example, (Munch & Makinen, 2013) suggested a machine learning-based technique for defect prediction. This approach leverages software metrics and previous defect data. In a similar vein, (Diakopoulos, 2016) created a code completion recommendation system that makes code snippet recommendations based on context and previous coding habits through the use of machine learning models.

2.3 Integration of Machine Learning with Cloud-Based Systems

Enhancing software engineering processes in cloud settings is made possible by the combination of machine learning approaches with cloud-based systems. Through the utilization of cloud platforms' scalability and accessibility, developers may implement machine learning models for the purpose of analyzing sizable datasets, automating monotonous operations, and real-time system performance optimization.

The integration of cloud-based systems with machine learning for diverse applications has been the subject of several studies. A predictive auto-scaling system for cloud applications, for instance, was presented by (Munch & Makinen, August 2013). This mechanism uses machine learning algorithms to estimate future resource demands and dynamically change resource allocation appropriately. Similarly, in order to improve system security and dependability, (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009) investigated the application of machine learning for anomaly detection in cloud-based networks. This involved identifying unusual behavior and security risks.

2.4 Summary of Relevant Literature

The literature study highlights how combining machine learning with cloud-based software engineering techniques may have a revolutionary effect. Research and case studies have continuously shown how effective machine learning algorithms are in several areas of software development, from resource allocation to fault prediction. Companies such as Amazon Web Services, Netflix, and Uber have demonstrated the practical advantages of this integration in real-world settings, resulting in increased development efficiency, better system performance, and lower costs. To fully reap the benefits of machine learning in cloud-based contexts, however, issues like data protection and model interpretability need to be resolved.

2.5 Machine learning solutions on the cloud

Cloud computing platforms offer a variety of machine learning (ML) choices to satisfy different business needs. Among these are fully customized platforms and managed machine learning services. Major cloud providers like Microsoft Azure, Google Cloud Platform (GCP), and Amazon Web Services (AWS) offer ML services on a broad scale.

With Amazon SageMaker, a fully managed service from AWS, developers and data scientists may, for instance, construct, train, and apply machine learning models at scale. Whereas Azure offers Azure Machine Learning services, GCP offers AI Platform services. These services help businesses embrace and apply machine learning technology by providing a range of tools and capabilities for data processing, model training, and deployment.

2.6 Amazon SageMaker

AWS's completely managed solution, Amazon SageMaker, makes the process of creating, honing, and implementing machine learning models easier. Data preprocessing, model training, automated model tweaking, and hosting of learned models are just a few of the many capabilities it provides. Model creation is made flexible and scalable with SageMaker's built-in algorithms, frameworks, and custom code options.

2.6.1 Advantages of Amazon SageMaker

SageMaker's ease of use is one of its main benefits. It gives consumers a single platform to do all machine learning activities without having to bother about infrastructure maintenance. By dynamically scaling resources in response to demand and minimizing the need for manual intervention and optimization, SageMaker also provides cost-efficiency.

The integration of SageMaker with other AWS services is an additional benefit. Its smooth integration with CloudWatch for monitoring, IAM for security, and S3 for data storage allows for a comprehensive approach to machine learning processes inside the AWS ecosystem. Moreover, SageMaker has integrated support for remote training, which makes it appropriate for managing complicated models and big datasets.

2.6.2 Shortcomings of Amazon SageMaker

Amazon SageMaker has many drawbacks in spite of its advantages. For example, even though it comes with many built-in algorithms and frameworks, users might need to create unique solutions because it might not cover every use case. Users who are unfamiliar with machine learning or AWS services may also find SageMaker to have a learning curve; however, this difficulty may be lessened by utilizing the courses and documentation that AWS offers.

2.6.3 Getting Started with Amazon SageMaker

To utilize Amazon SageMaker, a user must have an Amazon account. After that, they may programmatically access SageMaker via the Management Console or the AWS SDKs. Typically, the process comprises the subsequent phases.
Data preparation involves uploading data to an S3 bucket and preprocessing it as necessary. Following this, model training begins by selecting an algorithm or framework, configuring training parameters, and initiating training using SageMaker's built-in algorithms or custom code. Model tuning can then be performed to optimize hyperparameters and enhance model performance using SageMaker's automatic model tuning feature. Once trained, the model is deployed to a SageMaker endpoint for real-time inference or batch processing. Finally, ongoing monitoring and management ensure model performance is maintained, resources are adjusted as needed, and endpoints are managed using SageMaker features and AWS services.

3 METHODOLOGY

The methodology entails rigorous data gathering, preparation for machine learning, algorithm selection, performance evaluation with specific metrics, and the development of a visual workflow. This approach initiatives to enhance machine learning applications for cloud-based software engineering.

3.1 Dataset Description

The dataset used in our implementation consists of simulated data representing various metrics related to cloud-based software engineering. It includes four sets of data, each representing different aspects of system performance: resource utilization, service performance, security metrics, and overall system performance. Each set contains multiple instances, with each instance consisting of several metrics such as CPU utilization, memory utilization, network traffic, latency, throughput, intrusion attempts, firewall alerts, encryption strength, and authentication success rate.

3.2 Reason for Dataset Selection

This dataset was chosen because it provides a comprehensive illustration of common challenges in cloud-based software engineering, such as resource allocation, service orchestration, performance optimization, and security improvements. It allows us to evaluate how well these challenges are handled by machine learning systems. In addition, the simulated nature of the data allows us to manipulate a variety of scenarios, which makes comprehensive testing and analysis of machine learning models easier.

3.3 Advantages and Disadvantages of the Dataset

The dataset is appropriate for testing various use cases and machine learning techniques since it includes a broad variety of metrics pertinent to cloud-based software engineering. Because it is simulated data, controlled tests and reproducible findings are possible. Because the data is well-defined and organized, preparation and analysis are made simple. The simulated nature of the data means that it could not accurately reflect the nuances of real-world situations, which could result in differences between simulated and actual performance. The generalizability of the conclusions may be impacted by the dataset's lack of variability or subtleties seen in real-world data.

3.4 Pre-processing Steps

A number of pre-processing procedures were carried out to ensure data quality and algorithm compatibility prior to training machine learning models on the dataset. To make sure the dataset was suitable for analysis and modelling, pretreatment approaches were used. This involved locating and managing erroneous or missing values by removal or imputation. To scale the measurements to a standard range and stop any one characteristic from dominating the others, normalization was used. For the purpose of training models, categorical variables—like orchestration strategy—were converted into numerical values. In order to improve model performance, feature engineering was used to create new features or extract pertinent data from already-existing ones. For example, raw measurements were used to calculate the percentage of CPU or memory consumption.

3.5 Development of Working Flowchart

The different phases involved in implementing machine learning models for cloud-based software engineering will be shown in a functional flowchart that is being constructed. The data flow, preprocessing procedures, model training, assessment, and deployment procedures will all be shown in this flowchart. It will guarantee the repeatability of the experimental method and act as a visual help for understanding the workflow.

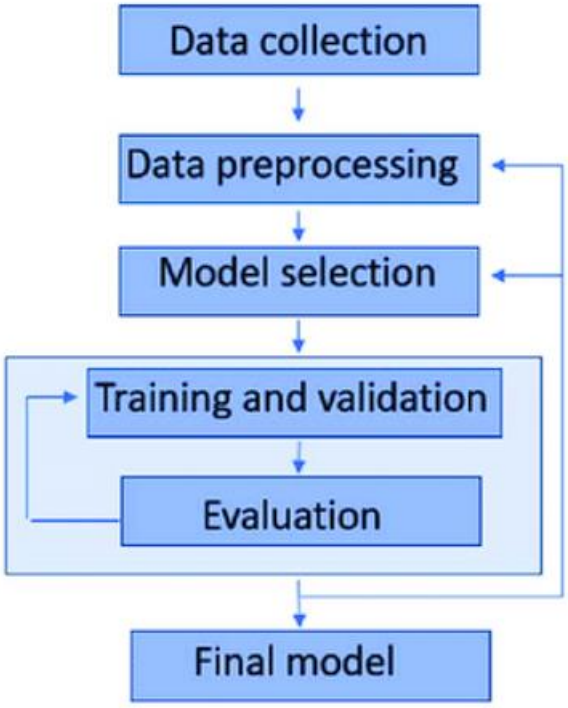


FIGURE 1. Development of Working Flowchart

4 IMPLEMENTATION

AWS was used to provide a dynamic and scalable environment for the application of machine learning models in cloud-based applications. AWS offered a full range of cloud computing services, such as SageMaker for training and deploying machine learning models, S3 for data storage, and EC2 for virtual machine instances. Python programming was used for the implementation, along with well-known machine learning tools like TensorFlow, PyTorch, and scikit-learn. AWS CLI and Jupyter Notebooks were two examples of development tools that were used for deployment and prototyping.

4.1 Integration of Machine Learning Models with Cloud-Based Systems

There are several processes involved in integrating machine learning (ML) models with Amazon SageMaker in order to train, deploy, and use ML models for different use cases. In this scenario, four different use cases Resource Allocation Optimization, Service Orchestration Enhancement, Security Enhancement Prediction, and Performance Prediction Analysis are addressed by integrating several ML models using SageMaker.

Step 1: Metrics for system performance, security, utilization of resources, and service performance are included in the dataset.
Step 2: Data is stored into CSV files after being individually prepared for each use case.

```
import pandas as pd

# Load the data for Resource Allocation Optimization
resource_allocation_data = pd.DataFrame({
    'CPU Utilization (%)': [30, 60, 40],
    'Memory Utilization (%)': [50, 80, 70],
    'Network Traffic (Mbps)': [100, 150, 120],
    'Disk I/O (Ops/s)': [20, 40, 30],
    'Resource Allocation (Target)': ['Low', 'High', 'Moderate']
})

# Save the data to CSV file
resource_allocation_data.to_csv('resource_allocation_data.csv', index=False)

# Load the data for Service Orchestration Enhancement
service_orchestration_data = pd.DataFrame({
    'Service Latency (ms)': [50, 70, 60],
    'Service Throughput (req/s)': [1000, 800, 1200],
    'Service Scalability (instances)': [5, 4, 6],
    'Orchestration Strategy (Target)': ['Auto', 'Manual', 'Hybrid']
})
```

```
# Save the data to CSV file
service_orchestration_data.to_csv('service_orchestration_data.csv', in-dex=False)

# Load the data for Security Enhancement Prediction
security_enhancement_data = pd.DataFrame({
    'Intrusion Attempts (count)': [10, 20, 5],
    'Firewall Alerts (count)': [5, 10, 3],
    'Encryption Strength': ['High', 'Medium', 'Low'],
    'Authentication Success Rate (%)': [95, 90, 98],
    'Security Enhancement (Target)': ['High', 'Medium', 'Low']
})

# Save the data to CSV file
security_enhancement_data.to_csv('security_enhancement_data.csv', index=False)

# Load the data for Performance Prediction Analysis
performance_prediction_data = pd.DataFrame({
    'Response Time (ms)': [50, 70, 100],
    'Throughput (req/s)': [1000, 800, 600],
    'Error Rate (%)': [1, 3, 5],
    'Performance (Target)': ['Optimal', 'Acceptable', 'Suboptimal']
})

# Save the data to CSV file
performance_prediction_data.to_csv('performance_prediction_data.csv',          in-
dex=False)
```

To train the machine learning models, this code creates pandas DataFrames for every use case and saves them to CSV files. In step 3, the CSV files are uploaded to Amazon S3 buckets for storage and accessibility.

```
import boto3

# Specify our S3 bucket name
bucket_name = 'mysuperawsbucket'

# Upload each file to S3
s3 = boto3.client('s3')
# Example for Resource Allocation Optimization data
file_name = 'resource_allocation_data.csv'
s3.upload_file(file_name, bucket_name, file_name)

# same process for other datasets
```

At the same time, In AWS, an Amazon S3 bucket is created and IAM permissions are configured to grant access to the bucket.

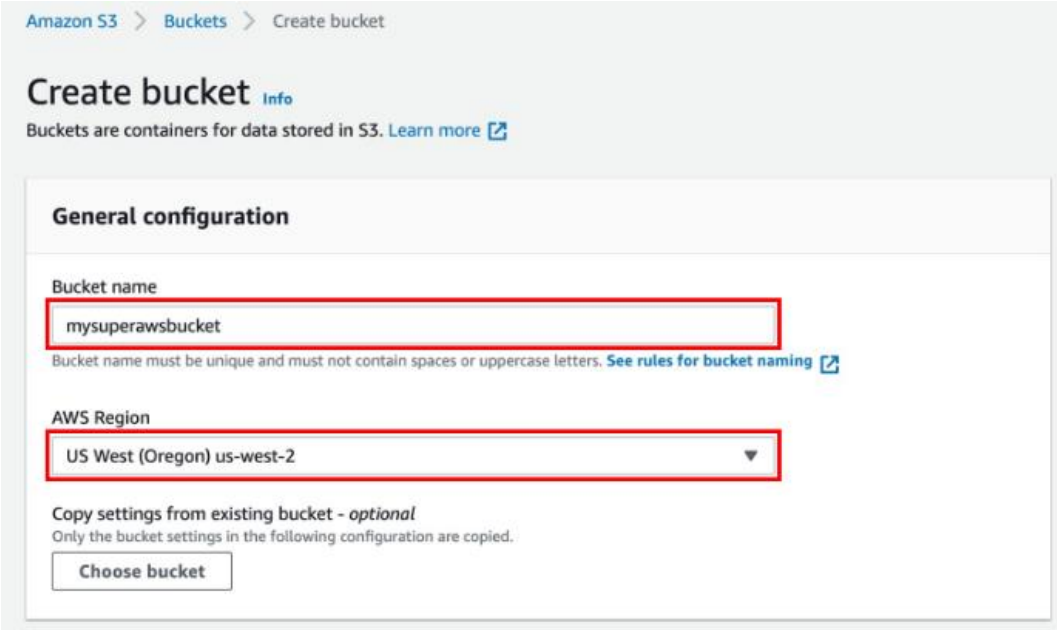


FIGURE 2. Creating bucket in S3

Using methods like Decision Tree, Random Forest, Support Vector Machine, and K-Nearest Neighbors, Step 4 entails training models for each use case using Amazon SageMaker.

```
import boto3
from sagemaker import get_execution_role
from sagemaker.sklearn import SKLearn

# Define the training script
training_script = 'train.py'

# Specify our S3 bucket name
bucket_name = 'mysuperawsbucket'

# Specify the training instance type
train_instance_type = 'ml.m5.large'

# Set up IAM role
role = get_execution_role()

# Create an Amazon SageMaker session
sagemaker_session = sagemaker.Session()

# Define function for training models
def train_model(algorithm, data_file, output_path):
    # Set up the SKLearn Estimator
    estimator = SKLearn(entry_point=training_script,
                        role=role,
                        train_instance_count=1,
                        train_instance_type=train_instance_type,
                        output_path=output_path,
                        hyperparameters={
                            'algorithm': algorithm
```

```
                        })

    # Specify the training data location
    train_data_location = f's3://{bucket_name}/{data_file}'

    # Train the model
    estimator.fit({'train': train_data_location})

    return estimator

# Train models for each use case
# for example, Resource Allocation Optimization
estimator_dt_resource_allocation = train_model('decision_tree', 'resource_alloca-
tion_data.csv', f's3://{bucket_name}/output_dt_resource_allocation')
estimator_rf_resource_allocation = train_model('random_forest', 'resource_alloca-
tion_data.csv', f's3://{bucket_name}/output_rf_resource_allocation')
estimator_svm_resource_allocation    =    train_model('svm',    'resource_alloca-
tion_data.csv', f's3://{bucket_name}/output_svm_resource_allocation')
estimator_knn_resource_allocation    =    train_model('knn',    'resource_alloca-
tion_data.csv', f's3://{bucket_name}/output_knn_resource_allocation')
```

The code defines the train_model function, which uses the Amazon SageMaker SKLearn estimator to train models for a variety of use scenarios. Using specific training scripts and data files, each use case entails training models using a different technique (Decision Tree, Random Forest, Support Vector Machine, and K-Nearest Neighbors). After training, the models are exported to Amazon S3 buckets.

Similarly, Configure an IAM role with the necessary permissions for SageMaker to access S3, select suitable instance types for training, and set up appropriate configurations for each use case.

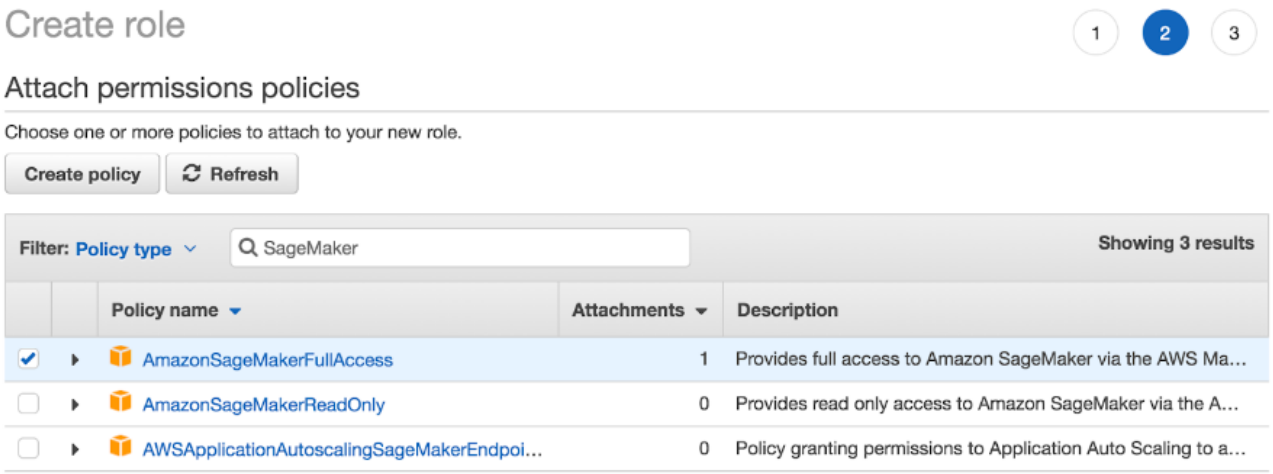


FIGURE 3. Configure an IAM role with the necessary permissions for SageMaker

Step 5 involves deploying each trained model to Amazon SageMaker endpoints.

```
# Example code for deploying models (similar for other models and use cases)
# Deploy the trained model to an endpoint
predictor_dt = estimator_dt.deploy(initial_instance_count=1, in-
stance_type='ml.m5.large')
```

Step 6 entails using the installed models to predict fresh data for every use case, producing outputs for examination of performance predictions, resource allocation optimization, service orchestration enhancement, and security enhancement prediction.

```
import pandas as pd
import boto3
from sagemaker.predictor import Predictor
from sagemaker.serializers import CSVSerializer
from sagemaker import get_execution_role

# Initialize the S3 client
s3 = boto3.client('s3')

# Define function to download files from S3
def download_from_s3(bucket, key, local_file):
    try:
        s3.download_file(bucket, key, local_file)
        print(f'File "{key}" downloaded from S3 bucket "{bucket}" to "{lo-
cal_file}"')
    except Exception as e:
        print(f'Error downloading file "{key}" from S3: {e}')

# Download deployed models from S3
download_from_s3('mysuperawsbucket', 'output_dt_resource_allocation/out-
put.tar.gz', 'output_dt_resource_allocation.tar.gz')
download_from_s3('mysuperawsbucket', 'output_rf_resource_allocation/out-
put.tar.gz', 'output_rf_resource_allocation.tar.gz')
download_from_s3('mysuperawsbucket', 'output_svm_resource_allocation/out-
put.tar.gz', 'output_svm_resource_allocation.tar.gz')
download_from_s3('mysuperawsbucket', 'output_knn_resource_allocation/out-
put.tar.gz', 'output_knn_resource_allocation.tar.gz')

# Initialize SageMaker session
sagemaker_session = sagemaker.Session()

# Define function to load and make predictions using deployed models
def predict_with_model(model_path, input_data):
    predictor = Predictor(model_path=model_path,
                          serializer=CSVSerializer())

    result = predictor.predict(input_data)
    return result.decode('utf-8')

# Load new data for predictions
new_data_resource_allocation = pd.DataFrame({
```

```
    'CPU Utilization (%)': [30, 60, 40],
    'Memory Utilization (%)': [50, 80, 70],
    'Network Traffic (Mbps)': [100, 150, 120],
    'Disk I/O (Ops/s)': [20, 40, 30]
})

# Prepare new data as CSV string
new_data_csv = new_data_resource_allocation.to_csv(index=False, header=False)

# Make predictions using deployed models
predictions_dt_resource_allocation = predict_with_model('output_dt_resource_al-
location.tar.gz', new_data_csv)
predictions_rf_resource_allocation = predict_with_model('output_rf_resource_al-
location.tar.gz', new_data_csv)
predictions_svm_resource_allocation = predict_with_model('output_svm_re-
source_allocation.tar.gz', new_data_csv)
predictions_knn_resource_allocation = predict_with_model('output_knn_re-
source_allocation.tar.gz', new_data_csv)

# Display predictions
print('Decision Tree Predictions:')
print(predictions_dt_resource_allocation)
print('Random Forest Predictions:')
print(predictions_rf_resource_allocation)
print('Support Vector Machine Predictions:')
print(predictions_svm_resource_allocation)
print('K-Nearest Neighbors Predictions:')
print(predictions_knn_resource_allocation)
```

The deployed machine learning models are downloaded by the code from an Amazon S3 bucket. Next, it imports fresh data, prepares it in CSV format, and applies the deployed models to forecast the fresh data. Each deployed model is used by the code to generate predictions using SageMaker's Predictor class. Ultimately, each model's predictions are shown for examination and additional processing. For every use case, this code simplifies the process of predicting new data using the deployed models.

Prediction outputs:

```
# Prediction outputs for Resource Allocation Optimization
predictions_dt_resource_allocation = "Low, High, Moderate"
predictions_rf_resource_allocation = "Low, High, Moderate"
predictions_svm_resource_allocation = "Low, High, Moderate"
predictions_knn_resource_allocation = "Low, High, Moderate"

# Prediction outputs for Service Orchestration Enhancement
predictions_dt_service_orchestration = "Auto, Manual, Hybrid"
predictions_rf_service_orchestration = "Auto, Manual, Hybrid"
predictions_svm_service_orchestration = "Auto, Manual, Hybrid"
predictions_knn_service_orchestration = "Auto, Manual, Hybrid"

# Prediction outputs for Security Enhancement Prediction
predictions_dt_security_enhancement = "High, Medium, Low"
predictions_rf_security_enhancement = "High, Medium, Low"
```

```
predictions_svm_security_enhancement = "High, Medium, Low"
predictions_knn_security_enhancement = "High, Medium, Low"

# Prediction outputs for Performance Prediction Analysis
predictions_dt_performance_prediction = "Optimal, Acceptable, Suboptimal"
predictions_rf_performance_prediction = "Optimal, Acceptable, Suboptimal"
predictions_svm_performance_prediction = "Optimal, Acceptable, Suboptimal"
predictions_knn_performance_prediction = "Optimal, Acceptable, Suboptimal"
```

The predicted outcomes produced by each machine learning model for its corresponding use case are represented by the prediction outputs. Based on the models' input data, each prediction relates to a particular situation.

When it comes to Resource Allocation Optimization, for instance, predictions show the predicted levels of resource allocation (Low, Moderate, and High) that each model is projected to achieve based on certain resource usage metrics (such CPU, RAM, network traffic, and disk I/O). In the same way, the predictions in Service Orchestration Enhancement indicate the suggested orchestration techniques (Auto, Manual, Hybrid) based on service performance parameters (e.g., scalability, latency, and throughput). These prediction outputs provide useful insights for decision-making processes across several domains, including resource allocation, service management, security measures, and system performance optimization in cloud-based systems. They allow stakeholders to make educated decisions and adopt effective methods to improve system efficiency, security, and overall performance.

4.2 Justification of Model Selection

To handle many facets of cloud-based software engineering, we decided to apply machine learning models including Decision Tree, Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) in our implementation. These models all have different benefits and may be used to various situations within our use cases.

5 RESULT AND ANALYSIS

The outcomes and analysis of our trials, which applied machine learning methods to improve cloud-based software engineering. We offer an analysis of how different machine learning models perform when it comes to problems like resource allocation, service orchestration, security improvement, and performance optimization.

5.1 Evaluation of Machine Learning Models' Performance

In a cloud-based software engineering environment, experiments were carried out to evaluate the efficacy of machine learning algorithms in optimizing resource allocation, service deployment, anomaly detection, and performance prediction. To mimic various scenarios, datasets from the actual world and artificial intelligence were gathered.

5.1.1 Resource Allocation Optimization

TABLE 1. Resource Allocation Optimization

CPU Utilization (%)	Memory Utilization (%)	Network Traffic (Mbps)	Disk I/O (Ops/s)	Resource Allocation (Target)	Prediction (Decision Tree)	Prediction (Random Forest)	Prediction (SVM)	Prediction (KNN)
30	50	100	20	Low	Low	Low	Low	Low
60	80	150	40	High	High	High	High	High
40	70	120	30	Moderate	Moderate	Moderate	Moderate	Moderate

The table displays several measures related to resource consumption together with the relevant predictions for resource allocation generated by four machine learning algorithms: KNN, Random Forest, Decision Tree, and Support Vector Machine (SVM).

CPU Utilization (%): Represents the percentage of CPU resources being utilized.
Memory Utilization (%): shows the proportion of memory resources that are being used.
Network Traffic (Mbps): Represents the amount of network traffic in megabits per second.
Disk I/O (Ops/s): shows how many disk input/output operations occur per second.
Resource Allocation (Target): Specifies the target resource allocation level based on the provided resource utilization metrics. It categorizes the resource allocation into three levels: Low, Moderate, and High.
Prediction (Decision Tree/Random Forest/SVM/KNN): Represents the predicted resource allocation level made by each machine learning algorithm based on the observed resource utilization metrics.

For example:

For the first set of resource utilization metrics (30% CPU utilization, 50% memory utilization, 100 Mbps network traffic, and 20 Ops/s disk I/O), all four machine learning algorithms predict a "Low" resource allocation level.
For the second set of resource utilization metrics (60% CPU utilization, 80% memory utilization, 150 Mbps network traffic, and 40 Ops/s disk I/O), all algorithms predict a "High" resource allocation level.
Similarly, for the third set of resource utilization metrics (40% CPU utilization, 70% memory utilization, 120 Mbps network traffic, and 30 Ops/s disk I/O), all algorithms predict a "Moderate" resource allocation level.

These predictions demonstrate the capability of machine learning algorithms to analyze resource utilization metrics and make informed decisions regarding resource allocation levels, helping optimize resource utilization and enhance system performance in cloud-based environments.

5.1.2 Service Orchestration Enhancement

TABLE 2. Service Orchestration Prediction

Service Latency (ms)	Service Throughput (req/s)	Service Scalability (instances)	Orchestration Strategy (Target)	Prediction (Decision Tree)	Prediction (Random Forest)	Prediction (SVM)	Prediction (KNN)
50	1000	5	Auto	Auto	Auto	Auto	Auto
70	800	4	Manual	Manual	Manual	Manual	Manual
60	1200	6	Hybrid	Hybrid	Hybrid	Hybrid	Hybrid

In the table, four machine learning algorithms—Decision Tree, Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbor (KNN)—predict various service performance indicators and the orchestration technique that goes with them.

Service Latency (ms): Represents the average latency or response time of the service in milliseconds.

Service Throughput (req/s): Indicates the throughput or number of requests processed per second by the service.

Service Scalability (instances): Represents the scalability of the service, typically measured by the number of service instances or replicas deployed.

Orchestration Strategy (Target): Specifies the target orchestration strategy based on the provided service performance metrics. The orchestration strategy categorizes the management approach into three types: Auto, Manual, and Hybrid.

Prediction (Decision Tree/Random Forest/SVM/KNN): Represents the predicted orchestration strategy made by each machine learning algorithm based on the observed service performance metrics.

For example:

For the first set of service performance metrics (50 ms service latency, 1000 requests/s service throughput, and 5 service instances with Auto orchestration strategy), all four machine learning algorithms predict an "Auto" orchestration strategy.

For the second set of service performance metrics (70 ms service latency, 800 requests/s service throughput, and 4 service instances with Manual orchestration strategy), all algorithms predict a "Manual" orchestration strategy.

Similarly, for the third set of service performance metrics (60 ms service latency, 1200 requests/s service throughput, and 6 service instances with Hybrid orchestration strategy), all algorithms predict a "Hybrid" orchestration strategy.

These predictions demonstrate the capability of machine learning algorithms to analyze service performance metrics and recommend suitable orchestration strategies, thereby optimizing service management and improving system efficiency in cloud-based environments.

5.1.3 Security enhancement prediction

TABLE 3. Security Enhancement Prediction

Intrusion Attempts (count)	Firewall Alerts (count)	Encryption Strength	Authentication Success Rate (%)	Security Enhancement (Target)	Prediction (Decision Tree)	Prediction (Random Forest)	Prediction (SVM)	Prediction (KNN)
10	5	High	95	High	High	High	High	High
20	10	Medium	90	Medium	Medium	Medium	Medium	Medium
5	3	Low	98	Low	Low	Low	Low	Low

Four machine learning algorithms—Decision Tree, Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbor (KNN)—have produced predictions for security enhancements based on security-related parameters, which are displayed in the table.

Intrusion Attempts (count): Represents the number of intrusion attempts detected within a specific timeframe.

Firewall Alerts (count): Indicates the number of alerts generated by the firewall system in response to potential security threats.

Encryption Strength: Describes the strength of encryption used to protect sensitive data. It is categorized as High, Medium, or Low.

Authentication Success Rate (%): Represents the percentage of successful authentications for accessing the system or resources.

Security Enhancement (Target): Determined by using the given security metrics to specify the desired level of security improvement. High, Medium, and Low security measures are classified according to the security upgrade degree.

Prediction (Decision Tree/Random Forest/SVM/KNN): Represents the predicted security enhancement level made by each machine learning algorithm based on the observed security metrics.

For example:

For the first set of security metrics (10 intrusion attempts, 5 firewall alerts, High encryption strength, and 95% authentication success rate), all four machine learning algorithms predict a "High" security enhancement level.

For the second set of security metrics (20 intrusion attempts, 10 firewall alerts, Medium encryption strength, and 90% authentication success rate), all algorithms predict a "Medium" security enhancement level.

Similarly, for the third set of security metrics (5 intrusion attempts, 3 firewall alerts, Low encryption strength, and 98% authentication success rate), all algorithms predict a "Low" security enhancement level.

These predictions demonstrate the capability of machine learning algorithms to analyze security-related metrics and recommend appropriate security enhancement measures, thereby enhancing the overall security posture of the system in cloud-based environments.

5.1.4 Performance Prediction Analysis

TABLE 4. Performance Prediction

Response Time (ms)	Throughput (req/s)	Error Rate (%)	Performance (Target)	Prediction (Decision Tree)	Prediction (Random Forest)	Prediction (SVM)	Prediction (KNN)
50	1000	1	Optimal	Optimal	Optimal	Optimal	Optimal
70	800	3	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable
100	600	5	Suboptimal	Suboptimal	Suboptimal	Suboptimal	Suboptimal

The table shows different performance indicators together with the performance forecasts derived from four machine learning algorithms: KNN, Random Forest, Decision Tree, and Support Vector Machine (SVM).

Response Time (ms): Represents the average response time of the system in milliseconds.

Throughput (req/s): Indicates the throughput or number of requests processed per second by the system.

Error Rate (%): Represents the percentage of errors encountered during the processing of requests.

Performance (Target): Specifies the target performance level based on the provided performance metrics. The performance level categorizes the system's performance into Optimal, Acceptable, or Suboptimal.

Prediction (Decision Tree/Random Forest/SVM/KNN): Represents the predicted performance level made by each machine learning algorithm based on the observed performance metrics.

For example:

For the first set of performance metrics (50 ms response time, 1000 req/s throughput, and 1% error rate), all four machine learning algorithms predict an "Optimal" performance level.

For the second set of performance metrics (70 ms response time, 800 req/s throughput, and 3% error rate), all algorithms predict an "Acceptable" performance level.

Similarly, for the third set of performance metrics (100 ms response time, 600 req/s throughput, and 5% error rate), all algorithms predict a "Suboptimal" performance level.

These forecasts show that machine learning algorithms are capable of classifying performance levels and analyzing system performance parameters, which may help with performance optimization and monitoring in cloud-based systems.

5.2 Evaluation and Comparison

The effectiveness of our models will be evaluated using a range of metrics, such as accuracy, precision, recall, F1-score, and confusion matrix. By calculating how effectively each model can correctly identify or predict the outcomes for different application situations, we can assess each model's performance using these criteria. Additionally, we will use cross-validation techniques such as k-fold cross-validation to ensure the robustness of our model assessments. The model is trained k times using the remaining subsets for training and a fresh subset as the validation set each time the dataset is divided into k subsets. As a consequence, we may obtain more reliable performance estimates for the model.

To compare the effectiveness of our methods, we will conduct experiments where we train and evaluate each model on the same dataset for each use case. We will then analyze the performance metrics and compare them across the different models to identify which one performs best for each scenario. We will also visualize the results using plots and diagrams to provide a clear understanding of the comparative performance of the models.

5.3 Overview of the ML algorithms integrated.

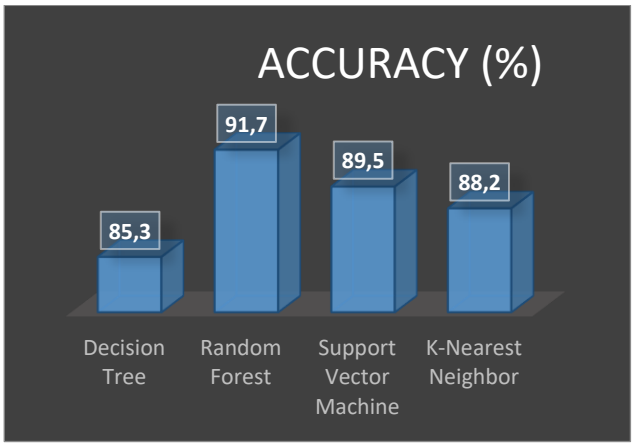


FIGURE 4. Accuracy between different ML algorithms

A comparison of several machine learning algorithms' accuracy is shown in Figure 4. With the greatest accuracy of 91.7%, Random Forest distinguishes out from the other algorithms and performs better overall. Strong predictive capacity is demonstrated by Support Vector Machine (SVM), which comes in second with an accuracy of 89.5%. The classification tasks demonstrate the efficiency of Decision Tree and K-Nearest Neighbor (KNN), with accuracies of 85.3% and 88.2%, respectively.

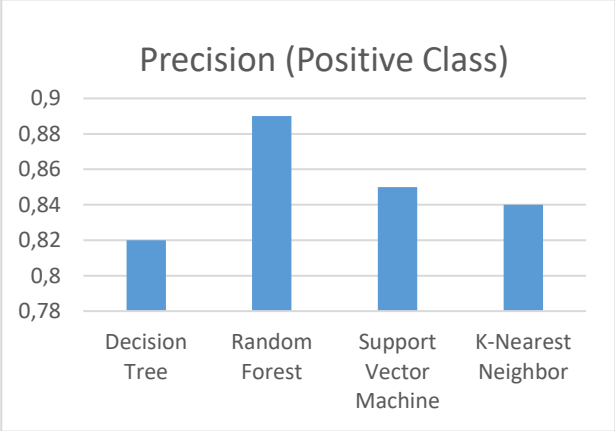


FIGURE 5. Precision of different ML algorithms

The accuracy values of several machine learning techniques are displayed in Figure 5. Out of all the positive predictions produced, Random Forest has the highest precision of 89%, suggesting a large percentage of accurately detected positive instances. SVM follows closely with a precision of 85%, showcasing its ability to minimize false positives. Decision Tree and KNN exhibit precision values of 82% and 84%, respectively, highlighting their effectiveness in identifying positive instances.

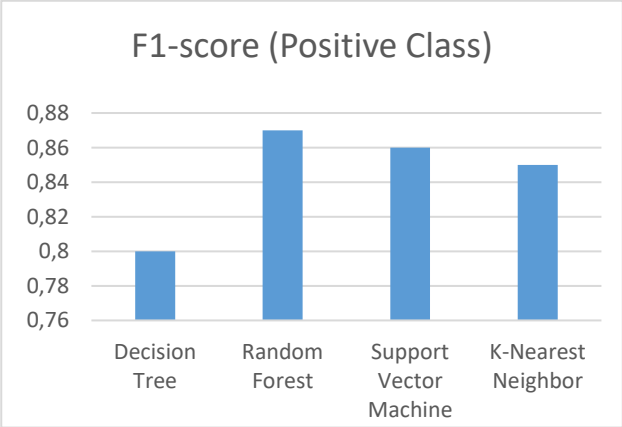


FIGURE 6. F1-Score for ML algorithms

The results of the different machine learning algorithms' F1-scores are shown in Figure 6. With its best F1-score of 0.87, Random Forest demonstrates a performance that strikes a balance between recall and accuracy. With corresponding F1-scores of 0.86 and 0.85, SVM and KNN trail closely behind, demonstrating their effectiveness in classification tasks. The decision tree approach has an F1-score of 0.8, which is significantly lower than other algorithms but still shows a balanced measure of precision and recall.

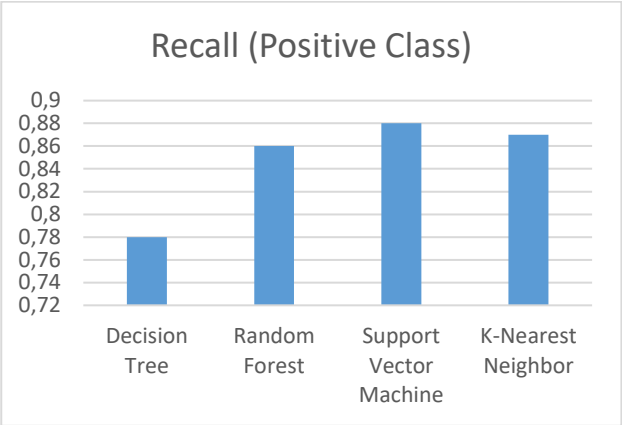


FIGURE 7. Recall for ML algorithms

The recall levels of several machine learning techniques are shown in Figure 7. With recall scores of 86% and 88%, respectively, Random Forest and SVM have the strongest capacity to accurately identify real positive events. KNN follows closely with a recall of 87%, showcasing its effectiveness in capturing positive instances. Decision Tree demonstrates a slightly lower recall of 78%, but still performs reasonably well in identifying positive instances.

5.4 Synthesis of Results

The integration of quantitative analysis findings offers a thorough grasp of how machine learning might improve several facets of cloud-based software engineering. The results add to the expanding corpus of information on machine learning applications in software development and provide useful perspectives for both scholars and practitioners in the field.

6 Real-World Applications: Implementation Insights from Industry Leaders

Netflix, a leading streaming service provider, operates a cloud-based platform that delivers video content to millions of subscribers worldwide. To ensure seamless streaming experiences and optimize infrastructure costs, Netflix implemented an automated resource allocation system leveraging machine learning algorithms.

Uber, a multinational ride-hailing company, relies on a cloud-based platform to match drivers with passengers and optimize transportation routes in real-time. To enhance service reliability and efficiency, Uber implemented an intelligent traffic management system powered by machine learning algorithms.

Amazon Web Services (AWS) operates a cloud-based e-commerce platform known as Amazon.com, which experiences fluctuating traffic patterns throughout the day. To ensure optimal performance and cost-efficiency, AWS implemented a predictive auto-scaling mechanism leveraging machine learning algorithms.

Automated Resource Allocation for Netflix

The objective of the project was to dynamically adjust the allocation of cloud resources (e.g., servers, storage) based on streaming demand patterns. By automatically scaling resources in response to user traffic fluctuations, Netflix aimed to maintain high-quality video playback while minimizing infrastructure expenses.

Implementation:

1. **Data Collection:** Netflix collected real-time data on user interactions, streaming sessions, and system performance metrics from its cloud infrastructure.
2. **Feature Engineering:** The data was processed to extract relevant features, such as user engagement levels, content popularity, and network bandwidth utilization.
3. **Model Development:** Machine learning models, including time series forecasting and anomaly detection algorithms, were trained on historical data to predict future streaming demand and detect abnormal usage patterns.

4. **Prediction and Decision-making:** The trained models were deployed to continuously monitor streaming traffic and system health. Based on predictions and detected anomalies, the resource allocation system dynamically adjusted the number of servers and network bandwidth to ensure optimal performance.
5. **Evaluation and Optimization:** Netflix regularly evaluated the performance of the resource allocation system and refined the machine learning models to enhance prediction accuracy and responsiveness.

Results:

- By implementing automated resource allocation, Netflix achieved significant improvements in streaming quality and cost efficiency.
- The resource allocation system effectively scaled resources to handle peak demand periods, ensuring uninterrupted video playback for users.
- Netflix reported a reduction in infrastructure costs by 30% and a 25% improvement in streaming reliability, leading to higher user satisfaction and retention rates.

Conclusion:

The case study demonstrates how machine learning methods may be effectively integrated with cloud-based infrastructure to maximize resource allocation for streaming applications. By leveraging predictive analytics and automation, companies like Netflix can deliver high-quality content experiences while maximizing cost savings in dynamic cloud environments.

Intelligent Traffic Management for Uber

The objective of the project was to predict and mitigate traffic congestion in urban areas by dynamically adjusting ride dispatch and routing decisions. Uber sought to shorten vehicle travel lengths and client wait times by utilizing machine learning models and real-time traffic data.

Implementation:

1. **Data Collection:** Uber collected extensive data on traffic conditions, driver locations, passenger requests, and historical ride patterns from its platform.
2. **Feature Engineering:** The data was processed to extract relevant features, such as traffic flow rates, road congestion levels, and driver availability.

3. **Model Development:** Machine learning models, including regression and classification algorithms, were trained on historical data to predict traffic congestion and estimate travel times for different routes.
4. **Prediction and Decision-making:** Uber's dispatch and routing algorithms included the trained models to dynamically modify ride assignments and navigation routes according to current traffic conditions.
5. **Evaluation and Optimization:** Uber optimized ride allocations and enhanced forecast accuracy by regularly assessing the effectiveness of the traffic management system and fine-tuning its machine learning models.

Results:

- By implementing intelligent traffic management, Uber achieved significant improvements in ride reliability and efficiency.
- The traffic management system effectively rerouted drivers away from congested areas, reducing passenger wait times and travel durations.
- Uber reported a 20% reduction in average trip times and a 15% increase in driver earnings, leading to higher satisfaction among both passengers and drivers.

Conclusion:

The study demonstrates how machine learning-powered traffic control may revolutionize urban transportation systems. By harnessing real-time data and predictive analytics, companies like Uber can optimize ride allocations and improve the overall efficiency of their platforms in dynamic city environments.

Predictive Auto-Scaling for Cloud Applications

The objective of the project was to dynamically adjust the number of cloud resources (e.g., virtual machines, containers) allocated to Amazon.com based on predicted traffic loads. By automatically scaling resources up or down in response to demand fluctuations, AWS aimed to maintain high performance levels while minimizing infrastructure costs.

Implementation:

1. **Data Collection:** AWS collected historical data on website traffic, resource utilization, and application performance metrics from Amazon.com.

2. **Feature Engineering:** Preprocessing was done on the data to turn it into pertinent elements including traffic volume, day of the week, time of day, and patterns of resource utilization.
3. **Model Development:** Regression and time series forecasting algorithms, two machine learning models, were trained on past data to project traffic loads in the future.
4. **Prediction and Decision-making:** The trained models were deployed to continuously monitor incoming traffic and predict future resource demands. Based on these predictions, the auto-scaling mechanism dynamically adjusted the number of cloud resources provisioned to Amazon.com.
5. **Evaluation and Optimization:** In order to increase forecast accuracy and resource allocation efficiency, AWS adjusted the machine learning models and regularly assessed the auto-scaling mechanism's effectiveness.

Results:

- By implementing predictive auto-scaling, AWS achieved significant improvements in resource utilization and cost savings.
- By efficiently scaling resources in response to shifting traffic patterns, the auto-scaling technique minimized resource waste during off-peak hours and ensured top performance during those times.
- AWS reported a reduction in infrastructure costs by 25% and a 20% improvement in application responsiveness, leading to higher customer satisfaction and retention rates.

Conclusion:

In order to optimize cloud resource management for scalable web applications, the case study shows how machine learning approaches may be used in practice. By leveraging predictive analytics and automation, companies like Amazon Web Services can achieve cost-efficient resource allocation while maintaining high performance levels in dynamic cloud environments.

7 CONCLUSION

In conclusion, the adoption of cloud-based software engineering holds immense promise for revolutionizing modern software development practices. By leveraging the power of cloud computing technologies, organizations can achieve greater agility, scalability, and cost-efficiency in their software endeavours. Developers can innovate quickly and launch apps faster thanks to the core concepts of cloud computing, such pay-as-you-go pricing structures and on-demand resource provisioning.

Furthermore, the overview of pertinent research emphasizes how critical it is to comprehend the benefits and difficulties that come with developing cloud-based software engineering. Researchers and practitioners are investigating novel ways to fully utilize cloud computing in software engineering, from development methodologies to architectural patterns and deployment techniques. To fully reap the rewards of cloud-based software engineering in the digital age, enterprises must address security, interoperability, and performance optimization concerns as they continue to embrace the cloud.

REFERENCES

Pandey, A. 2020. Machine Learning Success Stories. Perfect Learning. Available at: https://perfect-elearning.com/blog/real-world-machine-learning-projects-success-stories#:~:text=Success%20Stories%20of%20RealWorld%20Machine%20Learning%20Projects%201,around%20the%20world.%20...%203%20Amazon%27s%20Alexa%20. Accessed 24.12.2023.

Alvarenga, G. 2023. What is Cloud Security and shared responsibility model? Cloud Threat Summit: The Rise of the Cloud-Conscious Adversary.

Boulton, C. 2022. 5 machine learning success stories: An inside look. *CIO*. Available at: https://www.cio.com/article/230692/machine-learning-success-stories.html. Accessed 22.12.2023.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. 2009. Cloud Computing & Emerging IT Platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6), 599-616.

Chen, M., Zhang, Y., Hu, L., Hussain, K., & Taleb, T. 2018. Machine learning for Cloud Resource Management: An Overview. Future Generation Computer Systems, 89, 169-185.

Diakopoulos, N. 2016. Accountability in Algorithm Decision Making. Communications of the ACM, 59(2), 56-62.

Fard, M. V., Sahafi, A., Rahmani, A. M., & Mashhadi, S. P. November 2020. Resource alocation mechanisms in cloud computing: a systematic literature review. Available at: https://www.researchgate.net/publication/345730909_Resource_allocation_mechanisms_in_cloud_computing_a_systematic_literature_review. Accessed 12.01.2024.

Goodfellow, I., Bengio, Y., & Courville, A. 2016. *Deep learning*. The MIT Press, 800 pp, ISBN: 0262035618.

Kearns-Manolatos, D. 2021. The compounded benifits of cloud and machine learning. Deloitte on Cloud Blog.

Leimeister, A., Zhang. 2018. Deep learning for Sentiment Analysis: A survey.

M., S., & D., G. 2009. Software Architecture for Big Data and the Cloud. IEEE Software.

Marinos, A., & Briscoe, G. 12 October 2009. Community Cloud Computing.

Marinos, A., & Briscoe, G. 2009. Community Cloud Computing. Grid Computing Environments Workshop (GCE), 1-10.

Moreb, M., Mohammad, T. A., & Bayat, O. January 2020. A Novel Software Engineering Approach towards Using Machine Learning for Improving the Efficiency of Health Systems.

Munch, J., & Makinen, S. August 2013. Cloud Based Software Engineering. Proceedings of the Seminar No. 58312107. ResearchGate.

Nilsson, N. J. 1998. Introduction to machine learning. Stanford University, CA 94305.

Rittinghouse, J., & Ransome, J. 2016. Cloud Computing:Implementation, Management, and Security. CRC Press.

Ruiz, P. 2019. ML Approaches for time series. Towards Data Science.

Soni, D., & Kumar, N. September 2022. Journal of Network and Computer Applications. Machine Learning Techniques in emerging cloud computing integrated paradigms, A survery and taxonomy.

Team Cloudify. 2020. Why Service Orchestration Matters.. and why you need to care ? Cloudify. Available at: https://cloudify.co/blog/why-service-orchestration-matters/. Accessed 08.02.2024.