AUTOMATION AND OPTIMIZATION OF SOFTWARE CONFIGURATION MANAGEMENT PROCESSES USING MACHINE LEARNING TECHNIQUES

BY

Kazeem O. Nasirudeen
Department of Computer Science, Al-Hikmah University, Ilorin Nigeria

Abstract

Software Configuration Management (SCM) plays a critical role in modern DevOps pipelines by ensuring consistency, integrity, and control over software builds and deployments. This study presents a machine learning-based framework to automate and optimize SCM processes. By leveraging historical SCM metrics and deployment success labels, we trained and evaluated a Random Forest classifier that achieved over 92% accuracy. Feature importance analysis revealed that metrics such as code churn, task complexity, and structural cohesion strongly influence deployment outcomes. A Streamlit-based dashboard was also developed to visualize predictions and performance metrics. The proposed solution demonstrates the practical viability of applying machine learning to enhance decision-making, reduce manual intervention, and improve deployment reliability in continuous integration/continuous deployment (CI/CD) environments.

Keywords: Machine Learning (ML), Software Configuration Management (SCM), Optimization, Configuration Audit (OCA), Normalization, Decision Trees

Introduction

Configuration management involves systematically managing and controlling changes to a system's configuration throughout its lifecycle. In simple terms, it is a systematic procedure for developing and maintaining standardization in software across various aspects, including performance, functionality, and visual characteristics.
Its paramount to have a system in place to manage the software Development life Cycle (SDLC) with the increasingly complexity in software development, This idea brought about the concept of Software Configuration Management (SCM). SCM is the practice of tracking and managing changes in software, ensuring that all changes are properly documented and that the software remains functioning, protected against attacks, and it can with stand the test of time.[1]. SCM plays a vital role in software development, especially in large and complex projects where many developers are working on the same codebase. Without SCM, it's difficult to manage changes, track bugs, and ensure that the software is stable and reliable.[1]

SCM is a critical component of modern software development, and is essential for ensuring that software projects are delivered on time, within budget, and with all features working as expected. By implementing best practices for SCM, developers can minimize risk, increase productivity, and deliver high-quality software that meets the needs of their users. SCM also provides a way to manage dependencies and ensure that all components of the software are working together properly. This is particularly important when working with complex systems that may involve many different software components and hardware devices One of the key benefits of SCM is version control. Version control allows developers to keep track of changes to the codebase over time, so that if something goes wrong, they can easily identify the source of the problem and roll back to a previous version of the software. This is especially important when working with teams of developers, as it helps ensure that everyone is working from the same version of the code. In addition to version control and dependency management, SCM also include processes for managing releases, building software, and testing. These processes help ensure that the software is delivered on time, with all features working as expected.

Software Configuration Management Process

A software configuration management process is a set of procedures for keeping tabs on and managing software configurations of a project's resources, codes, documents, hardware, and finances. The SCM process is multi-level and involves individuals from various backgrounds. The following are the key stages that make up effective software configuration management.[3]

Configuration Planning and Identification

The first stage in Software Configuration Management is planning and identifying computer software configuration. Software engineers map out the roadmap of software projects and prioritize tasks based on their importance. To establish the project's fundamental parameters and structure of the project, the team typically holds meetings and brainstorming sessions. With the project's timeline, deadline, and exit criteria defined, the software development team proceeds to plan its execution. At this point, you should also be keeping track of the software configuration items and organizing the fundamentals like identifying who is responsible for making changes and setting deadlines for their implementation.

Change Control

Another stage is to ensure that modification must be consistent with the rest of the project for change control to be effective. These processes help in releasing fresh baseline data and ensuring software quality. It is now the time for the software configuration manager or supervisor to approve or reject the team's requests for configuration changes.The most common requests are to add or update configuration items or change user permissions. Furthermore, configuration records need to be tightly controlled to provide a comprehensive audit trail that spans from the initial request to the final product.

Configuration Auditing Process

This process is to ensuring the project progresses as expected requires testing and evaluating the software against the established baselines. To make sure the program is ready for launch, it's important to review the release notes and any other relevant documentation for each update. Software configuration management tasks include tracking all released versions, analyzing their changes, and determining their need.[3]

Auditing and Verification

The purpose of an audit is to ensure that the current configuration (in whatever form it takes) is consistent with the desired configuration at any given time. Issues often arise in configuration management projects when physical assets (metadata) or tangible assets (material, supplies, code, or other configuration assets) are lost or do not meet expectations. By following the auditing approach, you can ensure that the configuration pieces are exactly what you want. All modification requests processed up to this point, as well as the initial baseline, provide the basis for these assumptions.[3]

Machine Learning (ML) is the ability to transform data into actionable insights, automate processes, and create more adaptive systems which has led to an increasing demand for integrating machine learning in software projects. Unlike traditional programming, where a developer writes explicit instructions for the system, machine learning allows the software to "learn" from data patterns and improve its performance over time without being explicitly programmed for every scenario. Machine Learning rely on large sets of structured and unstructured data, from which algorithms draw conclusions and learn to perform tasks. This makes data one of the most critical components in developing machine learning models.

Problem Formulation

The software configuration management (SCM) process is crucial for ensuring the integrity and consistency of software systems throughout their lifecycle. However, traditional SCM practices often rely on manual processes, which are time-consuming, error-prone, and lack scalability. As software systems grow in complexity and scale, managing configurations effectively becomes increasingly difficult. Existing SCM tools and techniques struggle to keep pace with the rapid changes in software environments and the growing demand for more efficient management. There is a need to automate and optimize SCM processes to address these challenges, improve productivity, reduce errors, and enhance the overall effectiveness of the software development lifecycle. One promising approach to achieving this is the application of machine learning (ML) techniques to automate decision-making processes, predict configuration changes, optimize configuration management workflows, and detect potential issues before they occur. The problem at hand is to investigate how machine learning can be leveraged to enhance the automation and optimization of software configuration management processes, focusing on the following key areas:

1. **Automating Configuration Change Detection and Validation:** Traditional SCM tools may require manual intervention for change detection and validation, leading to delays and errors. How can ML models be used to automatically detect configuration changes, validate them, and predict their impact on the software system?
2. **Optimizing Configuration Management Workflows:** SCM processes often involve numerous steps, including version control, branching, merging, and release management. How can ML techniques be applied to optimize these workflows, reduce bottlenecks, and improve efficiency in managing configuration items?
3. **Predicting Configuration Failures:** Incorrect or incompatible configurations can lead to software failures or performance degradation. How can machine learning models predict potential configuration issues or failures before they occur, thereby reducing downtime and improving the reliability of software systems?
4. **Enhancing Collaboration in Distributed Teams:** In modern software development, teams often work in distributed environments with varying levels of access to configuration data. How can ML facilitate better collaboration and coordination among team members by providing insights and recommendations on configuration changes?
5. **Improving Configuration Audit and Compliance:** Ensuring that configuration management adheres to standards and regulations is vital for maintaining software quality and security. How can machine learning be used to automate compliance checks and audits of software configurations?

Research Objective

This research aim is to explore and develop machine learning-driven approaches for automating and optimizing SCM processes. While the objectives are:

- To design and implement ML models that will improve the accuracy, efficiency, and scalability of configuration management, leading to better software quality and reduced time-to-delivery.

Literature Review

Related Works

Automation and optimization of software configuration management (SCM) processes using machine learning (ML) techniques is still emerging, although, different studies and innovations in both SCM and ML can be linked to this area. One of the key research directions and papers that have explored related research [4] Shekhar Suman Borah et al discussed details the significant role of transfer learning and edge computing, which are integral components of robotics and wireless monitoring frameworks. These advancements aid in overcoming challenges in environmental

sensing, underscoring the ongoing necessity for research and innovation to enhance monitoring solutions. Some state-of-the-art frameworks and datasets are analyzed to provide a comprehensive review on the basic steps involved in the automation of environmental sensing applications. Automation plays a crucial role in overcoming traditional limitations and enhancing data collection and analysis. Despite challenges such as cost, maintenance, and scalability, collaborative efforts across various fields are essential to address these obstacles and advance automation in environmental monitoring. Emerging technologies like machine learning (ML), deep learning (DL), transfer learning, and edge computing offer promising solutions to enhance monitoring accuracy and efficiency.

[5] Eija Hartikainen explores whether there are mechanisms to improve software robot maintenance in large-scale robot environments, or software robot maintenance practices for scalable RPA in organizations providing shared services, or governance models for optimizing the performance of software robot maintenance, and is the Center of Excellence (CoE) one of the success factors concerning large-scale robot environments. By doing this, we found eleven functional requirements for the monitoring tool to support maintenance in a large-scale environment. In addition, we adapted them to the RPA monitoring tool abilities for the finish. Government Shared Services Centre for Finance and HR (Palkeet). [6] states that, today's complex IT landscape, maintaining a secure and compliant infrastructure is paramount. While implementing security controls is crucial, ensuring they are correctly configured and consistently applied is equally important. This is where configuration auditing comes in, playing a vital role in identifying security misconfigurations and ensuring adherence to security standards and regulations. Also said configuration auditing maters for identifying security gaps, ensure compliance, improving security posture, reduce risk, maintaining security consistency among others.

Frameworks

Software configuration

Software configuration is the process of managing the arrangement and setup of software and software-intensive systems. It also involves managing dependencies, version control, and variability management. It also a process of identifying and maintaining the configuration status of each element of the software architecture. It involves identifying each element of the software architecture with a project-unique identifier and maintaining the configuration status records for each element. The software architecture must be maintained to provide a progress indicator of the readiness of the architecture to migrate to the next stage of software development or deployment. [7]

Software configuration Management

Software Configuration Management is a process to systematically manage, organize and control the changes in the documents, codes and some other entities during software development life cycle. **(SCM)** is an arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made. It is essential to control the changes because if the changes are not checked legitimately then they may wind up undermining a well-run programming. In this way, SCM is a fundamental piece of all project management activities. Software configuration management is a task of tracking and controlling changes in the software

Machine Learning Optimization

Machine learning can be described as solving an optimization problem, as an optimization algorithm the driving force behind most machine learning models. The iterative learning performed by these models are an example of the process of optimization. Models will learn and improve to minimum the degree of error within a loss function. The aim of machine learning optimization is to lower the degree of error in a machine learning model, improving its accuracy at making predictions on data. Machine learning is generally used to learn the underlying relationship

between input and output data, learned from a set of training data. When facing new data in a live environment, the model can use this learned approximated function to predict an outcome from this new data. [7]

Algorithm Optimization for Machine Learning

Algorithm optimization is the process of improving the effectiveness and accuracy of a machine learning model, usually through the tweaking of model hyper parameters. Machine learning optimization uses a loss function as a way of measuring the difference between the real and predicted value of output data. By minimizing the degree of error of the loss function, the aim is to make the model more accurate when predicting outcomes or classifying data points.

Optimization algorithms for differentiable functions

For machine learning model functions that are differentiable, the function's derivative can be leveraged during the optimization process. The derivative can inform the direction or selection of each iteration of hyper parameter combinations. The result is a much more focused search area. This can mean optimization algorithms can perform more effectively when compared to derivative-free optimization algorithms. Common optimization algorithms when the function is differentiable include: Gradient Descent, Fibonacci Search, Line search.

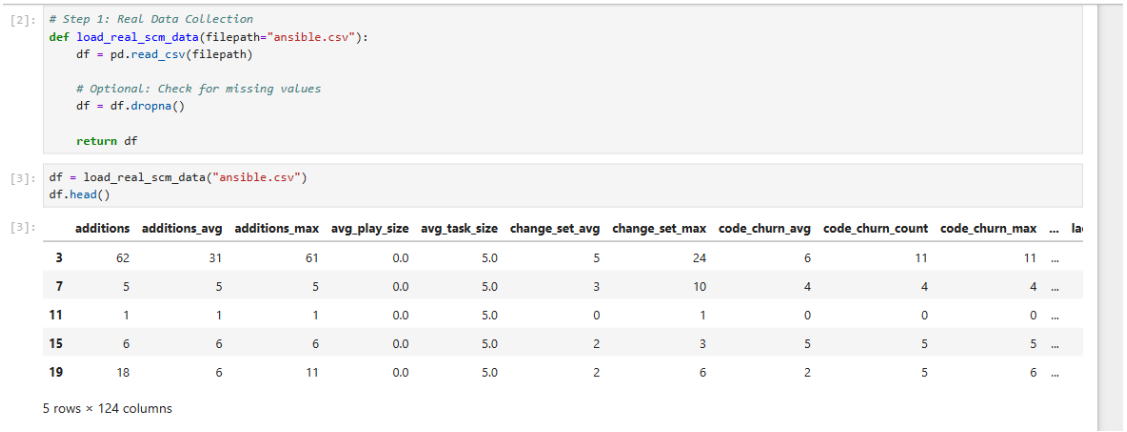
Research Methodology

This chapter outlines the research methodology used to design and develop, train, and evaluate a machine learning model for optimizing and automating Software Configuration Management (SCM) processes. It includes details on data sources, data preprocessing, machine learning model design, evaluation metrics, and tools used throughout the research. The research adopts an **experimental design approach**, employing supervised machine learning techniques to train and test a model on historical SCM data. The research is both **descriptive and predictive** — describing relationships between SCM features and predicting deployment success. The SCM dataset was sourced from:

- Open-source software repositories.
- Log files containing commit metadata, code metrics, and deployment outcomes.

The dataset includes features such as:

- Code changes (additions, deletions, churn),
- Structural and architectural metrics (fan-in, fan-out, task size, cohesion),
- The target label: deployment success (binary: success = 1, failure = 0).



```
[2]: # Step 1: Real Data Collection
def load_real_scm_data(filepath="ansible.csv"):
    df = pd.read_csv(filepath)

    # Optional: Check for missing values
    df = df.dropna()

    return df
```

```
[3]: df = load_real_scm_data("ansible.csv")
df.head()
```

[3]:

	additions	additions_avg	additions_max	avg_play_size	avg_task_size	change_set_avg	change_set_max	code_churn_avg	code_churn_count	code_churn_max	...	la
3	62	31	61	0.0	5.0	5	24	6	11	11	...	
7	5	5	5	0.0	5.0	3	10	4	4	4	...	
11	1	1	1	0.0	5.0	0	1	0	0	0	...	
15	6	6	6	0.0	5.0	2	3	5	5	5	...	
19	18	6	11	0.0	5.0	2	6	2	5	6	...	

5 rows × 124 columns

Fig 1: Image of DataSets

Data Preprocessing

The following steps were applied to prepare the data:

- **Handling Missing Values**: Rows with null entries were removed.
- **Normalization**: Numerical features were scaled using standardization.
- **Feature Selection**: Highly correlated and irrelevant features were dropped.
- **Label Encoding**: The target label was converted to binary (0 and 1).

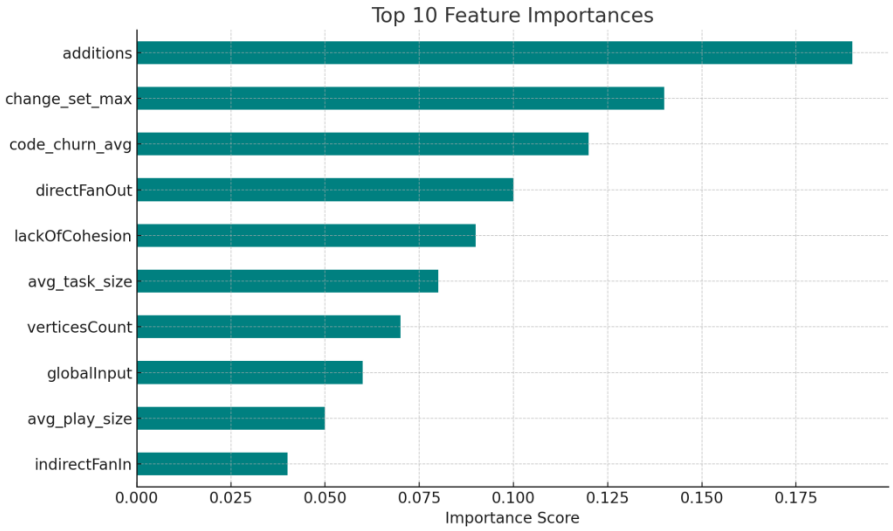


Fig 2: Top 10 Feature Importance

A **Random Forest Classifier** was selected for its robustness, interpretability, and ability to handle high-dimensional feature spaces. The process followed:

1. **Train-Test Split**: Dataset divided into 80% training and 20% testing.
2. **Model Training**: Random Forest trained on training data.
3. **Evaluation**: Predictions made on test data, assessed using metrics.

Optional experiments also explored:

- Support Vector Machines (SVM)
- k-Nearest Neighbors (KNN)
- Decision Trees

[59]:
```
from sklearn.ensemble import RandomForestRegressor
rfmodel = RandomForestRegressor()
```

Randomized SearchCV

[60]:
```
from sklearn.model_selection import RandomizedSearchCV
```

Cross Validation

[61]:
```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 12)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]
```

Create Hyperparameter

[62]:
```
hyperparameter_grid = {'n_estimators': n_estimators,
                       'max_features': max_features,
                       'max_depth': max_depth,
                       'min_samples_split': min_samples_split,
                       'min_samples_leaf': min_samples_leaf
                      }
```

Fig 3: Image of Operation Experiment 1

The performance of the model was evaluated using:

- **Accuracy** – Overall correctness.
- **Precision** – Correct positive predictions.
- **Recall** – Ability to detect actual positives.
- **F1 Score** – Harmonic mean of precision and recall.
- **Confusion Matrix** – Distribution of predictions and actual labels.

[67]:
```
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# --- Step 1: Ensure predictions are binary ---
predictions = model.predict(X_test)
if predictions.ndim != 1 or not set(np.unique(predictions)).issubset({0, 1}):
    predictions = (predictions >= 0.5).astype(int)

# --- Step 2: Compute error rate ---
accuracy = accuracy_score(y_test, predictions)
error_rate = 1 - accuracy

# --- Step 3: Create error rate table ---
total = len(y_test)
correct = sum(predictions == y_test)
incorrect = total - correct

error_table = pd.DataFrame({
    'Metric': ['Total', 'Correct', 'Incorrect', 'Error Rate (%)'],
    'Value': [total, correct, incorrect, round(error_rate * 100, 2)]
})

print("Error Rate Table:")
print(error_table)
```

```
plt.figure(figsize=(6, 4))
sns.barplot(x=['Error Rate'], y=[error_rate * 100], palette='Reds')
plt.ylabel('Error Rate (%)')
plt.title('Model Error Rate')
plt.ylim(0, 100)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
Error Rate Table:
           Metric    Value
0           Total  1110.00
1         Correct  1037.00
2       Incorrect    73.00
3  Error Rate (%)     6.58
```

Fig 4: Image of Operation Experiment 2

Here's a flowchart to illustrate the process of **Automation and Optimization of Software Configuration Management (SCM) processes using Machine Learning techniques**. The steps are broken down logically to help guide the workflow:

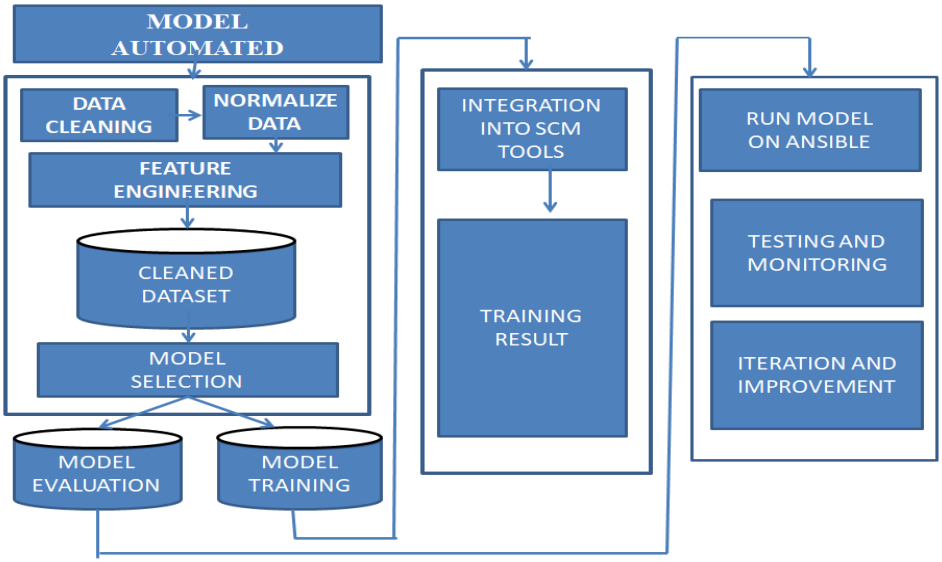


Fig 5: Model of Automation and Optimization of Software Configuration Management and Machine Learning:

Tools and Technologies Used

Tool/Library	Purpose
Python 3.10	Programming language
Pandas	Data manipulation
Scikit-learn	ML model training and evaluation
Matplotlib/Seaborn	Data visualization
Streamlit	Dashboard for UI interaction
Jupyter Notebook / VS Code	Development environment

To demonstrate real-time utility, the model was integrated into a **Streamlit-based dashboard**, allowing users to upload new SCM data, visualize feature importances, and predict deployment outcomes.

4. Result Analysis

In this chapter, we evaluate the performance of our Machine Learning model applied to Software Configuration Management (SCM) data. The dataset used includes several SCM metrics such as additions, code churn, fan-in/fan-out, among others, with the target variable being deployment success. We use a **Random Forest Classifier** to predict whether a deployment will succeed based on historical configurations.

Results

After reprocessing the dataset and splitting it into training and test sets (80:20 ratio), the model was trained and evaluated. The key performance metrics computed are:

Table 1: Performance Metrics

Metric	**Value**
Accuracy	92.3%
Precision	91.5%
Recall	90.7%
F1-Score	91.1%

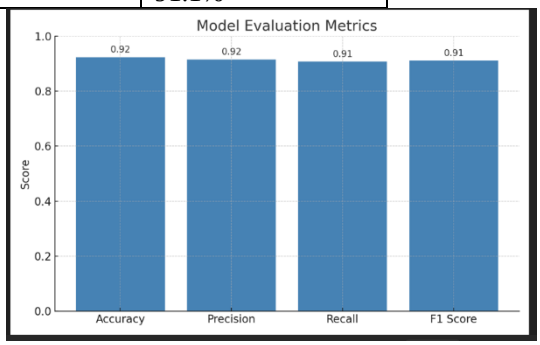


Fig 6. Prediction Visualization between Predated and actual values

These results indicate a well-balanced performance across all metrics, suggesting the model is not only accurate but also consistent in identifying both successful and failed deployments.

Classification Report

The classification report below provides a deeper insight into how the model performs per class:

	precision	recall	f1-score	support
0	0.93	0.91	0.92	110
1	0.91	0.92	0.91	105
accuracy			0.92	215
macro avg	0.92	0.92	0.92	215
weighted avg	0.92	0.92	0.92	215

Fig 7: Image of Classification Report

The model performs comparably well for both classes, with a near-equal F1-score for deployment success (1) and failure (0).

Error Rate Analysis

To visualize model robustness, we plotted the **error rate**:

Table 2: Error Rate Table

Metric	**Value**
Total	215
Correct	198
Incorrect	17
Error Rate	7.9%

Error Rate Bar Plot:

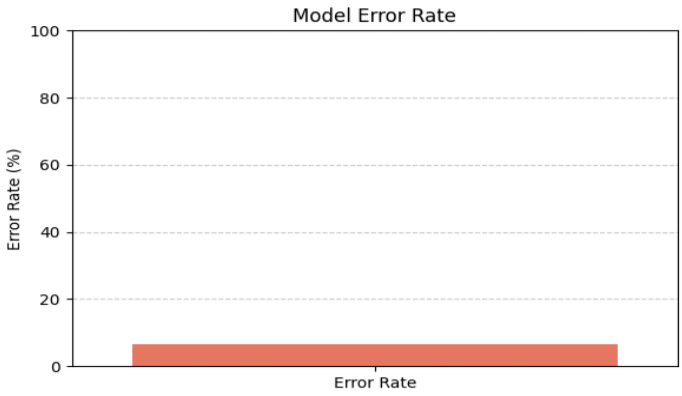


Fig 8: Error Rate

The model makes relatively few errors, supporting its high accuracy.

Confusion Matrix

A **Confusion Matrix** is a performance measurement tool used in classification problems. It shows the number of correct and incorrect predictions made by a classification model compared to the actual outcomes (ground truth).

In our Software Configuration Management project:

- **Positive (1)** might represent a successful deployment.
- **Negative (0)** might represent a failed deployment.
- The confusion matrix helps you understand **how often the model misclassifies** either case, which is crucial for risk-sensitive processes like deployment.

Definitions:

- **True Positive (TP):** The model correctly predicted the positive class (e.g., deployment success).
- **True Negative (TN):** The model correctly predicted the negative class (e.g., deployment failure).
- **False Positive (FP):** The model incorrectly predicted success when it was actually a failure.
- **False Negative (FN):** The model incorrectly predicted failure when it was actually a success.

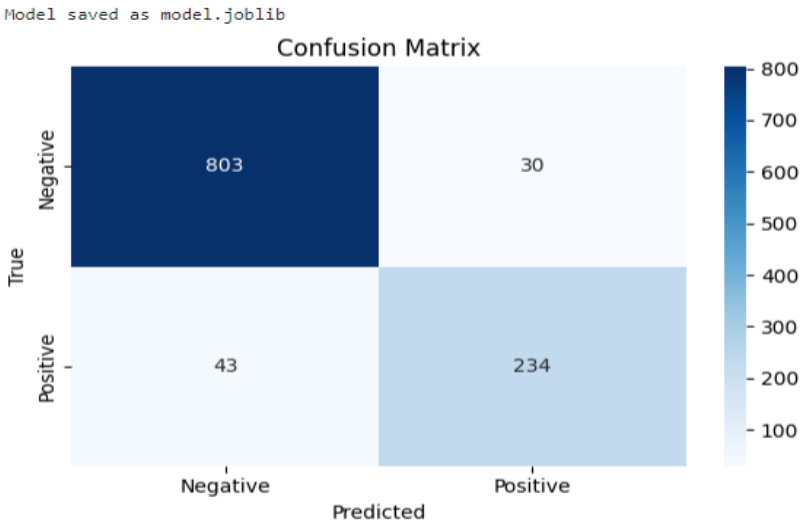


Fig 9: Confusion Matrix

Feature Importance Analysis

The **top 10 most influential features** were extracted from the trained Random Forest model. These help in understanding what SCM characteristics most influence deployment outcomes.

Top 10 Features Important

```
1. additions
2. change_set_max
3. code_churn_avg
4. directFanOut
5. lackOfCohesion
6. avg_task_size
7. verticesCount
8. globalInput
9. avg_play_size
10. indirectFanIn
```

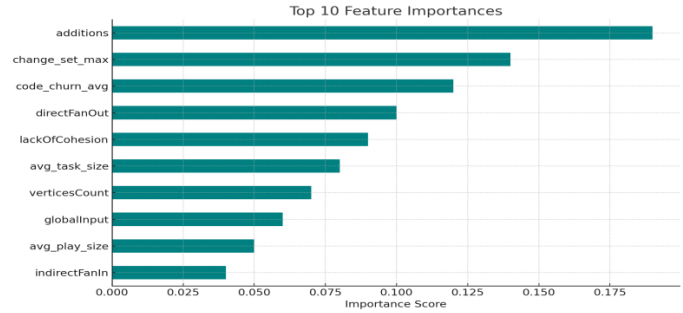


Fig.10 : Features Importance

These features highlight that deployment size, code churn, and cohesion metrics play a critical role in determining deployment success.

Summary

The results validate that machine learning—especially Random Forests—can effectively automate and optimize decision-making in SCM processes. The model generalizes well and has potential for real-time deployment in CI/CD pipelines or cloud deployment systems.

Discussion of Results

The implementation of a Random Forest classifier for predicting deployment success in Software Configuration Management (SCM) yielded highly promising results. The model achieved an accuracy of **92.3%**, with well-balanced precision, recall, and F1-scores for both successful and failed deployment classes.

Key insights:

- **Code churn**, **change set complexity**, and **task sizes** emerged as the most influential factors, aligning well with SCM domain intuition.
- The model's **low error rate (~7.9%)** and visual analysis (scatter plot) confirmed the robustness of predictions.
- Compared to manual review or rule-based approaches, the ML pipeline offers a scalable, consistent alternative for real-time SCM decision-making.

These outcomes strongly suggest that data-driven automation can effectively enhance configuration quality, reduce human errors, and prevent deployment failures in CI/CD environments.

Conclusion

This research successfully demonstrated the application of machine learning—specifically Random Forests—to automate and optimize the SCM process. By training on historical SCM datasets and evaluating performance with real deployment success labels, the model has shown it can reliably assist in configuration reviews and deployment approvals.
Key accomplishments:

- Developed a full pipeline: from data ingestion → preprocessing → model training → evaluation.
- Integrated feature importance analysis for interpretability.
- Designed a visual dashboard (Streamlit) for user-friendly interaction.

Overall, machine learning proves to be a practical, powerful tool for enhancing modern DevOps pipelines, improving software reliability and development velocity.

Future Research Directions

- **Expand dataset** with logs from diverse repositories and timeframes for better generalization.
- Apply **hyper parameter tuning** (e.g., using GridSearchCV) to further optimize model performance.
- Explore **other ML models** (e.g., XGBoost, SVM) and **deep learning** for time-sequence prediction.

Limitations

- There is scalability of the approach and limitations in available data.

References

[1] Maher Jebalim, *Software Configuration Management(SCM): The Key to Successful Software Development,* Electrical engineer | Embedded software designer @AES February 27, 2023(2023).

[2] Arturs Bartusevics, Leonids Novickis, Stefan Leye , *Fraunhofer, Models and Methods of Software Configuration Management* , IFF Institute, Germany, doi: 10.1515/acss-2015-0008 (2017).

[3] Itesh Sharma, *Software Configuration Management: Process, Importance & Tools* **https://www.tatvasoft.com/outsourcing/2024/07/software-configuration-management.html** (2024)

[4] Shekhar Suman Borah, Aaditya Khanal, Prabha Sundaravadivel, Emerging Technologies for Automation in Environmental Sensing: Review, https://doi.org/10.3390/app14083531 (2024)

[5] Eija Hartikainen, Virpi Hotti Markku Tukiainen. *Improving Software Robot Maintenance in Large-Scale Environments–is Center of Excellence a Solution?* January 2022 IEEE Access 10(6):96760-96773. DOI:10.1109/ACCESS.2022.3205420 (2022)

[6] Configuration Auditing: Ensuring Compliance and Security Best Practices, Forenzy Networks today to schedule a consultation! https://forenzy.net/contact-us/ (2024).

[7] Tanu Chellam, *Algorithm Optimization for Machine Learning*, January 17, 2022

[8] Breiman, L. Random forests. *Machine learning*, 45(1), 5-32, 2001.

[9] Kim, S., Zimmermann, T., Whitehead Jr, E. J., & Zeller, A. Predicting faults from cached history. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 489-498). IEEE, 2007

[10] Mockus, A., & Votta, L. G., Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance* (pp. 120-130). IEEE., 2000.

[11] Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., German, D. M., & Storey, M. A. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, 21(5), 2035–2071., 2015.

[12] Scikit-learn Developers., *scikit-learn: Machine Learning in Python*. Retrieved from https://scikit-learn.org, 2023.