Just-in-Time Software Defect Prediction using Unsupervised Methods for Detecting and Handling Concept Drift

Major Project Part-I 2024

Integrated B.Tech and M.Tech

in

Information Technology

Submitted By

Aditya Pote: 2020-IMT069

Under the Supervision of

Dr.Santosh Singh Rathore

Department of Computer Science and Engineering

and

Prof. Joydip Dhar

Department of Engineering Sciences

ABV-INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT GWALIOR

GWALIOR, INDIA

DECLARATION

I hereby certify that the work, which is being presented in the report entitled Just-in-Time Software Defect Prediction using Unsupervised Methods for Detecting and Handling Concept Drift, in fulfilment of the requirement for Major Project Part-I and submitted to the institution is an authentic record of my/our own work carried out during the period August-2024 to October-2024 under the supervision of Dr. Santosh Singh Rathore and Prof. Joydip Dhar. I also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

Dated: **Signature of the candidate**

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dated: **Signature of supervisor**

Acknowledgements

I would like to express my sincere gratitude and appreciation to Dr. Santosh Singh Rathore and Prof. Joydip Dhar for their exceptional mentorship throughout the course of this project titled "Just-in-Time Software Defect Prediction using Unsupervised Methods for Detecting and Handling Concept Drift" Dr. Santosh's guidance and unwavering support have been instrumental in shaping the direction and successful completion of this research.

I am genuinely thankful for the time and effort that both Dr. Santosh and Prof. Joydip dedicated to this project. Their encouragement and dedication to creating a stimulating academic environment have inspired me to push my boundaries, think critically, and maintain a high standard of work in all aspects of the research. Their mentorship has not only enhanced my technical skills but has also deepened my appreciation for the field of software engineering and the potential impact it can have on real-world applications.

Aditya Pote

Abstract

Software defect prediction helps improve the reliability and maintainability of software by identifying defect-prone areas in the code. Traditional supervised models rely on labeled data, but in many real-world cases, labeled datasets are not available, making unsupervised models more suitable. However, a key challenge in unsupervised software defect prediction is dealing with concept drift, where the data distribution changes over time due to evolving development practices or frameworks. Concept drift can degrade model performance, leading to inaccurate predictions. This report explores the integration of an unsupervised drift detection framework designed to detect concept drift, with unsupervised software defect prediction models. The framework monitors data distribution changes and helps adapt the prediction models dynamically as new data arrives. By detecting shifts in software metric distributions, the approach ensures the model remains accurate over time, providing a more robust solution for predicting software defects in changing development environments.

Keywords: Unsupervised Software Defect Prediction, Concept Drift Detection, Unsupervised drift detection, Software Quality, Machine Learning, Adaptive Models

Contents

List of Figures

1

Introduction

This chapter provides an introduction to the thesis topic including Software Defect Prediction and Concept Drift.

1.1 Introduction

Software defect prediction is crucial for ensuring software quality, especially as systems become more complex. By identifying defect-prone code early, developers can focus their testing and maintenance efforts effectively. Traditional supervised models for defect prediction rely on labeled data, but in many real-world cases, such data is limited or unavailable, making supervised methods difficult to apply. As a result, there has been a growing interest in unsupervised models, which do not require labeled data [1].

Unsupervised software defect prediction models analyze various software metrics (such as code complexity, churn, and dependencies) to group code modules and predict potential defects. These models work by recognizing patterns within the data without needing predefined labels. However, a key challenge for unsupervised methods is concept drift [3].

1.1.1 Just-in-Time Defect Prediction

Just-in-Time (JIT) software defect prediction is an approach that focuses on predicting defect-inducing software changes as they occur. Rather than predicting defects at a release level, JIT models operate at the commit level, providing real-time insights into the likelihood of a software change introducing a defect. This is especially valuable in agile and continuous integration environments where immediate feedback is critical for maintaining high code quality [11].

JIT defect prediction typically uses features such as commit messages, code churn, and developer activity to assess the risk of each change. By detecting potential defects early in the development process, JIT approaches allow developers to take corrective actions before the code is integrated, reducing the cost of defect fixes [11]. However, like other defect prediction models, JIT approaches are also susceptible to concept drift, as the nature of changes and development practices evolve over time.

1.1.2 Concept Drift in Software Defect Prediction

Concept drift occurs when the data's characteristics change over time. In software defect prediction, this can be caused by changes in coding practices, new frameworks, or team dynamics [3]. As a result, models that once accurately predicted defects may become less effective as the data evolves. Concept drift causes models to degrade in performance over time, making them unreliable in continuously evolving environments.

In unsupervised defect prediction, this issue is even more challenging since there are no labels to guide the model's adjustments to these changes. Without a mechanism to detect and respond to concept drift, models may produce inaccurate results, decreasing their overall usefulness [5].

1.1.3 Unsupervised Framework for Addressing Concept Drift

To address concept drift, this report explores the use of an unsupervised framework for detecting drift in real-time. The framework monitors changes in data distributions by using deep learning representations to compute distance metrics, such as the Fréchet distance, which signals when the data has shifted [3].

When applied to defect prediction, the framework tracks key software metrics to detect shifts in defect-prone code characteristics. By identifying these shifts, the framework allows the defect prediction model to adapt and maintain its accuracy as the software evolves [3]. This dynamic adaptation ensures that the model remains effective even in continuously updated software environments.

1.1.4 Integrating an Unsupervised Drift Detection Framework with Unsupervised Defect Prediction

The integration of an unsupervised drift detection framework into unsupervised software defect prediction models provides a promising solution to the problem of concept drift. This framework continuously monitors data for drift and informs the defect prediction model when adjustments are needed. This ensures that the model remains accurate and

reliable even as the software development process introduces new patterns or practices [8].

By integrating these two approaches, we can create a more robust and adaptive framework for unsupervised software defect prediction, capable of handling the dynamic nature of software development [9].

.

1.2 Motivation

The need to handle evolving software systems and concept drift motivates the integration of an unsupervised drift detection framework with unsupervised software defect prediction. This approach aims to enhance model adaptability and accuracy.

(i) **Evolving Software:** As software systems evolve with new updates and practices, defect-prone patterns also change. Models must adapt to remain effective.

(ii) **Limited Labeled Data:** Many projects lack labeled defect data, and while unsupervised models address this, their accuracy can decline due to concept drift.

(iii) **Real-Time Adaptation:** Concept drift can degrade model performance over time. Real-time drift detection ensures the model stays accurate by adapting to changing data.

(iv) **Software Quality:** Reliable defect prediction improves software quality by identifying issues early, reducing time and cost for testing and maintenance.

Integrating an unsupervised drift detection framework helps create a system that adapts to ongoing changes, providing accurate predictions that evolve with the software.

2

Literature Review

This chapter responds to the significant amount of research assigned to understanding the efficient methods for software defect prediction, particularly unsupervised approaches and concept drift detection. We examine some of the literature and briefly review the development of former proposed methods and the research gaps.

2.1 Literature Survey

This section provides a detailed examination of the literature on unsupervised software defect prediction (SDP) and concept drift detection techniques. The aim is to understand existing methods and identify research gaps that justify the proposed solution.

2.1.1 Unsupervised Software Defect Prediction

Unsupervised software defect prediction (SDP) has emerged as an important field within software engineering, focusing on identifying defect-prone modules without relying on labeled datasets. Various methods have been proposed to tackle this challenge. For example, Kumar, Chaturvedi, and Kailasam (2022) introduced an unsupervised approach using threshold derivation to identify faulty code based on software metrics [2]. This approach, TCL/TCLP enhances defect detection by deriving metric thresholds through a logarithmic transformation.

Unsupervised methods like clustering and threshold-based models aim to predict defects using software metrics such as code complexity, churn, and dependency metrics [2]. These methods allow defect prediction models to work even when labeled data is unavailable. However, a significant limitation of such models is that their performance may degrade over time due to the phenomenon of concept drift, where the statistical properties of the data change [2].

2.1.2 Just-in-Time Defect Prediction

Just-in-Time (JIT) software defect prediction focuses on identifying defect-prone software changes at the time of commitment, providing immediate feedback to developers. JIT models analyze features such as code churn, commit messages, and developer activity to assess whether a specific code change might introduce defects. This real-time feedback is particularly valuable in modern development practices like continuous integration, allowing developers to address potential issues before they propagate into the release [11].

Recent advancements in JIT-SDP include deep learning models such as DeepJIT,

which utilize neural networks for automatic feature extraction from commit data, improving predictive performance. However, JIT models, like traditional defect prediction models, are susceptible to concept drift as software development practices and code changes evolve. To address this, various learning techniques have been explored to allow models to adapt over time without requiring retraining from scratch, ensuring the model remains relevant as new changes are introduced [11].

2.1.3 Concept Drift Detection Techniques

Concept drift refers to changes in the statistical properties of data over time, which can reduce the performance of machine learning models. Detecting concept drift is critical in any domain where data evolves. Various techniques have been developed to detect and handle drift, including statistical tests and unsupervised methods.

Statistical-based methods, such as the Kolmogorov-Smirnov test or the Maximum Mean Discrepancy (MMD) test, compare the distributions of data over time [3]. DriftLens, for instance, employs deep learning representations to detect shifts in unstructured data like images, text, and speech. It has been shown to outperform other drift detectors in scenarios by measuring changes in data distribution through distance metrics [3].

Other methods, such as the Maximum Concept Discrepancy (MCD) introduced by Wan, Liang, and Yoon (2024), utilize contrastive learning to adaptively detect drift in high-dimensional data streams without relying on labeled data [5]. Additionally, the QuadCDD framework introduced by Wang et al. (2024) provides a deeper analysis of drift by characterizing the start, end, severity, and type of drift, making it useful for managing the impact of drift on predictive models [6]. These advanced techniques allow for more nuanced drift detection in evolving environments.

2.1.4 Concept Drift in Software Defect Prediction

Concept drift is particularly challenging in the domain of software defect prediction. As software systems evolve, the patterns of defect-prone code also change. This means that

models trained on historical software metrics may no longer be accurate in predicting defects in current or future versions of the software.

Gangwar and Kumar (2023) have shown that concept drift can lead to a degradation in the performance of defect prediction models [4]. Their study emphasized the need for models that can detect and adapt to drift. Methods like paired learners have been proposed, where models continuously evaluate recent data and adjust accordingly [4]. However, while such methods can detect drift, they often lack robust mechanisms for addressing the consequences of drift [4].

2.2 Research Gaps

Despite the advancements in unsupervised software defect prediction and concept drift detection, several gaps remain:

- Many unsupervised software defect prediction models do not account for concept drift, leading to poor long-term performance [4].
- Current concept drift detection techniques, while effective, often struggle with scalability, especially in real-time applications with high-dimensional data [3].
- Although some methods detect the onset of drift, they lack comprehensive strategies for characterizing and mitigating the drift [6].

The proposed solution integrates DriftLens with unsupervised SDP to provide a more scalable and adaptive model that can handle both the detection and mitigation of concept drift in software defect prediction.

2.3 Objectives

- To develop an unsupervised software defect prediction model that can accurately predict defect-prone code without the need for labeled data.

- To integrate DriftLens for detecting and handling concept drift, ensuring the model adapts to changes in software data over time.
- To test and evaluate the model's performance, ensuring it provides accurate real-time predictions in JIT-SDP and adapts effectively after concept drift.

3

Tentative Methodology

This chapter provides a comprehensive discussion of the tentative methodology employed in the proposed architecture, including the entities and different libraries implemented in the project.

3.1 Proposed Methodology

The proposed methodology integrates a dynamic concept drift detection mechanism with an unsupervised software defect prediction (SDP) model. This framework aims to address the challenges posed by evolving software metrics, ensuring that defect prediction models remain accurate and adaptive over time.

3.1.1 Overview of the Methodology

The methodology consists of two core components: the unsupervised SDP model and a concept drift detection system. The unsupervised SDP model is responsible for analyzing software metrics and predicting defect-prone modules, while the drift detection system continuously monitors data distributions to detect changes. Upon detecting drift, the SDP model adapts to maintain accuracy.

3.1.2 Unsupervised SDP Model

- **Model Approach:**
  - The unsupervised SDP model utilizes software metrics to predict defects, without the need for labeled data. Multiple approaches are viable, depending on the metrics and the nature of the project data.
  - The model groups software modules based on their characteristics, relying on patterns derived from historical software data.
- **Model Optimization:** Optimization techniques are applied to fine-tune the SDP model based on historical data, ensuring its ability to generalize across various software datasets. The model's performance is evaluated before and after concept drift detection to assess the impact of data evolution on prediction accuracy.

3.1.3 Concept Drift Detection Mechanism

A concept drift detection system, driftlens is used to monitor changes in the underlying data distribution. The system works alongside the unsupervised SDP model and identifies when the characteristics of the data deviate from the expected patterns, signaling the need for model adjustment.

3.1.3.1 Workflow

The drift detection system functions in along with the SDP model, continuously monitoring software metrics and detecting any shifts in the data distribution. The workflow follows these steps:

- **Data Monitoring:**
  - The system monitors the software metrics as they evolve over time and identifies potential changes in the data distribution.
  - By tracking these metrics, the system can detect significant shifts that may indicate the presence of concept drift.
- **Drift Detection:**
  - The system applies statistical tests Frechét distance to detect drift by analyzing the differences in the distribution of incoming data.
  - Once drift is detected, an alert is triggered, indicating that the unsupervised SDP model may require updating or adjustment.

3.1.3.2 Model Update Process

Upon detecting drift, the unsupervised SDP model is updated as follows:

- **Retraining:**

– The model is retrained with the updated data distribution, ensuring that it adapts to the new characteristics of the data and continues to predict defects accurately.

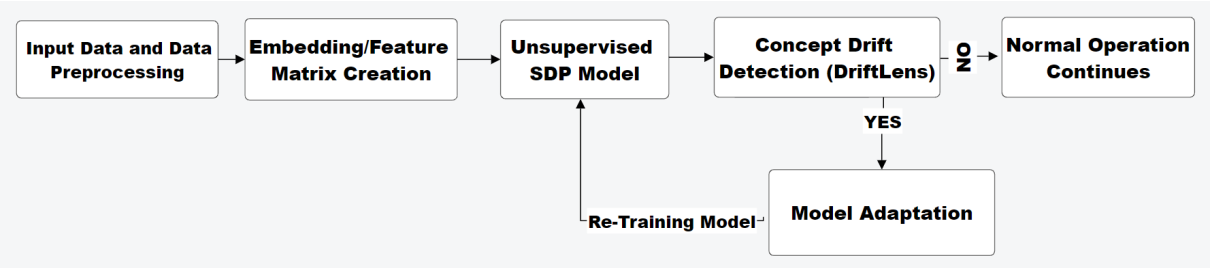

Figure 3.1: Architecture of the proposed framework integrating concept drift detection with unsupervised SDP model

3.1.3.3 Continuous Monitoring and Adaptation

The system continuously monitors the data distribution and adjusts the SDP model as needed. This ensures that the model remains adaptive to new patterns in the software metrics and maintains its performance over time.

3.1.4 Dataset Overview

The dataset used for this study is a comprehensive collection of software change revisions extracted from popular Apache projects, known as the ApacheJIT dataset.This dataset includes various change metrics that are crucial for defect prediction, such as lines added (la), lines deleted (ld), files touched (nf), directories modified (nd), subsystems involved (ns), and change entropy (ent). These metrics help identify patterns associated with defect-prone code.

Additional key features of the dataset include:

- **ndev**: The number of developers who previously worked on the file.
- **age**: The time since the last change to the file, measured in weeks.
- **nuc**: The number of unique changes made to the file.

- **aexp**: The experience of the author with the project.
- **arexp**: The author's experience with the affected subsystem.
- **asexp**: The author's experience with similar files in the past.

These features provide a detailed profile of each commit, contributing to a more granular understanding of which changes are more likely to introduce defects. The dataset covers a wide range of project histories from 2003 to 2019, making it suitable for training machine learning models aimed at defect prediction. Its size and diversity are particularly beneficial for deep learning models, which typically require large datasets to generalize effectively across different software versions and development patterns [10].

Table 3.1: Summary of ApacheJIT Dataset Features and Characteristics

Characteristic	**Description**
Total Commits	106,674 total commits from 14 Apache projects
Bug-Inducing Commits	28,239 labeled bug-inducing commits
Clean Commits	78,435 clean commits
Time Span	2003 to 2019
Key Features	Lines added (la), lines deleted (ld), files touched (nf), directories modified (nd), subsystems involved (ns), change entropy (ent), number of developers (ndev), time since last change (age), unique changes (nuc), author experience (aexp)
Projects Covered	14 Apache projects
Use Case	Just-In-Time Defect Prediction, software change analysis

4

Progress Summary

This chapter provides progress made so far and the work that remains to be done.

4.1 Progress Made

The project has made significant progress in the following areas:

- **Literature Review:** A comprehensive literature review has been conducted, covering unsupervised software defect prediction (SDP) models and concept drift detection techniques. This review has highlighted the gaps in existing approaches and justified the need for a combined framework for unsupervised SDP with dynamic drift detection.

- **Proposed Methodology:** The methodology has been outlined, with DriftLens selected as the concept drift detection mechanism. The unsupervised SDP model is flexible, with potential approaches identified, including clustering and threshold-based methods.

- **Dataset Collection and Analysis:** The ApacheJIT dataset has been chosen for the project. The dataset consists of a large number of commits from Apache projects, providing a rich source of data for training and evaluating the defect prediction models. Key metrics such as lines added, lines deleted, and change entropy have been extracted and pre-processed for model training.

4.2 Pending Tasks

While substantial progress has been made, several key tasks remain:

- **Model Training and Drift Detection:** Further experimentation is required for training and evaluating unsupervised SDP models using clustering-based and threshold-based approaches. DriftLens has been integrated, but additional tests are needed to thoroughly assess its effectiveness in detecting concept drift and triggering appropriate updates to the defect prediction model.

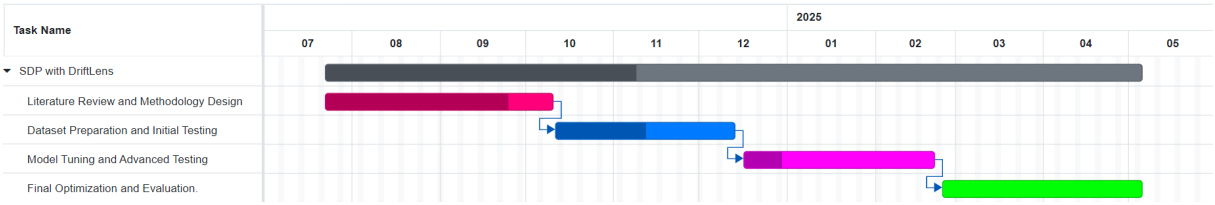

Figure 4.1: Gantt chart showing project progress and pending tasks

- **Evaluation of DriftLens:** Additional testing is required to evaluate the performance of DriftLens in different software development environments. This will involve comparing the model's accuracy before and after drift detection and adjusting the system for minor or significant drifts.

- **Implementation of Real-Time Updates:** The system needs to be configured for real-time updates, where the unsupervised SDP model dynamically adjusts its parameters or retrains based on the drift detected by DriftLens.

- **Final Integration and Testing:** Full integration of the unsupervised SDP model with DriftLens must be completed, followed by comprehensive testing across different datasets to ensure the framework's generalizability and scalability.

- **Documentation and Final Report:** The final report, detailing the methodology, experiments, and results, will need to be prepared, along with system documentation for future work or replication studies.

Bibliography

[1] Z. Xu, L. Li, M. Yan, J. Liu, X. Luo, J. Grundy, Y. Zhang, and X. Zhang, "A comprehensive comparative study of clustering-based unsupervised defect prediction models," *Journal of Systems and Software*, vol. 172, p. 110862, 2021. doi: https://doi.org/10.1016/j.jss.2020.110862.

[2] R. Kumar, A. Chaturvedi, and L. Kailasam, "An Unsupervised Software Fault Prediction Approach Using Threshold Derivation," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 911-932, 2022. doi: 10.1109/TR.2022.3151125.

[3] S. Greco, B. Vacchetti, D. Apiletti, and T. Cerquitelli, "Unsupervised Concept Drift Detection from Deep Learning Representations in Real-time," *arXiv preprint*, arXiv:2406.17813, 2024. Available: https://arxiv.org/abs/2406.17813.

[4] A. K. Gangwar and S. Kumar, "Concept Drift in Software Defect Prediction: A Method for Detecting and Handling the Drift," *ACM Transactions on Internet Technology*, vol. 23, no. 2, Article 31, 28 pages, May 2023. doi: 10.1145/3589342.

[5] K. Wan, Y. Liang, and S. Yoon, "Online Drift Detection with Maximum Concept Discrepancy," *arXiv preprint*, arXiv:2407.05375, 2024. Available: https://arxiv.org/abs/2407.05375.

[6] P. Wang, H. Yu, N. Jin, D. Davies, and W. L. Woo, "QuadCDD: A Quadruple-based Approach for Understanding Concept Drift in Data Streams," *Expert Systems with Applications*, vol. 238, p. 122114, 2024. doi: https://doi.org/10.1016/j.eswa.2023.122114.

[7] K. Malialis, J. Li, C. G. Panayiotou, and M. M. Polycarpou, "Incremental Learning with Concept Drift Detection and Prototype-based Embeddings for Graph Stream Classification," *arXiv preprint*, arXiv:2404.02572, 2024. Available: https://arxiv.org/abs/2404.02572.

[8] S. Khaki, A. A. Aditya, Z. Karnin, L. Ma, O. Pan, and S. M. Chandrashekar, "Uncovering Drift in Textual Data: An Unsupervised Method for Detecting and Mitigating Drift in Machine Learning Models," *arXiv preprint*, arXiv:2309.03831, 2023. Available: https://arxiv.org/abs/2309.03831.

[9] T. Salazar, J. Gama, H. Araújo, and P. H. Abreu, "Unveiling Group-Specific Distributed Concept Drift: A Fairness Imperative in Federated Learning," *arXiv preprint*, arXiv:2402.07586, 2024. Available: https://arxiv.org/abs/2402.07586.

[10] H. Keshavarz and M. Nagappan, "ApacheJIT: A Large Dataset for Just-In-Time Defect Prediction," *arXiv preprint*, arXiv:2203.00101, 2022. Available: https://arxiv.org/abs/2203.00101.

[11] Y. Zhao, K. Damevski, and H. Chen, "A systematic survey of just-in-time software defect prediction," *ACM Computing Surveys*, vol. 55, no. 10, p. 201, 2023. doi: https://doi.org/10.1145/3567550.