



Run Run Shaw Library

香港城市大學
City University of Hong Kong

Copyright Warning

Use of this thesis/dissertation/project is for the purpose of private study or scholarly research only. ***Users must comply with the Copyright Ordinance.***

Anyone who consults this thesis/dissertation/project is understood to recognise that its copyright rests with its author and that no part of it may be reproduced without the author's prior written consent.

CITY UNIVERSITY OF HONG KONG
香港城市大學

Comprehensive Investigation of Concept Drift
on the Performance of Software Defect
Prediction Models
軟體缺陷預測模型性能概念漂移的綜合研究

Submitted to
Department of Computer Science
電腦科學系
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
哲學博士學位

by

KABIR Md Alamgir

August 2021
二零二一年八月

Abstract

Concept Drift (CD) is a phenomenon in which the data distribution in datasets shifts over time, which negatively affects the performance of prediction models in software engineering (SE), such as software defect prediction (SDP) models. Detecting CD in SE datasets and improving the prediction performance of models impacted by CD is a challenge. Over time, a long-running software project experiences several versions, producing high volumes of data and making non-stationary data distribution. Furthermore, the relationship between data variables changes (i.e., concept drift). We can assume that data of project versions appear in the form of temporal data distributions. Predicting defects of such a project is challenging.

In SDP, the prediction models utilize historical data (from which the prediction model is built) and predict the defects for upcoming projects (with which the prediction model is tested). The performance of prediction models is known to be dependent on the datasets used for training the models. To improve the information content of the training datasets that appear over time, the impact of concept drift should be considered, which has not been extensively investigated. Among the SDP techniques, cross-version defect prediction (CVDP) is a process where the historical data is retrieved from the previous version of the software project and tested on the current version of the same project, which is considered more practical and realistic. In such a realistic scenario, the software project version is updated and maintains a chronological order of the project versions. The impact of concept drift on the performance of classification-based SDP models has never been explored

over the chronological defect datasets. Unfortunately, the defect datasets are imbalanced, with the non-defective modules dominating the defective modules. Class rebalancing techniques that modify the distribution of training datasets have been shown to improve the prediction performance of defect prediction models in prior research. Since class rebalancing techniques try to produce a similar representation of the classes, it may eliminate drift and impact the performance of classification-based SDP models, which has not been thoroughly investigated yet. Furthermore, prior works show the effectiveness of feature selection techniques to improve the performance of SDP. Researchers attributed that the prediction results fluctuate due to the changes in the data features. Therefore, feature selection techniques may avoid CD due to the variation of selecting features in each version of the software project, which has not yet been examined.

In this thesis, we investigate the impact of CD on SDP performances. A systematic investigation is conducted to examine whether class rebalancing could help eliminate CD from chronological defect datasets. To deal CD while considering class imbalance issue, we propose a novel concept drift detection (CDD) framework to detect CD in the cross-version (CV) defect datasets and investigate the feasibility of alleviating CD in CVDP models using class rebalancing techniques. The framework consists of four steps, in which the first pre-processes the CV defect datasets and forms a CV data stream, the second constructs the CV defect models, the third calculates the test statistics, and the fourth provides a hypothesis-test based CD detection method. We demonstrated the ability of the CDD framework by conducting experiments on 36 versions of 10 open-source software projects. We found that 50% of the CV defect datasets are drift-prone. Additionally, the class rebalancing techniques had a positive impact on the prediction performance for CVDP by correctly classifying the CV defective modules and detected CD

by up to 31% on the resampled datasets. We observed that the class rebalancing techniques are beneficial when the practitioners wish to increase the ability to correctly classifying the CV defective modules. In addition to this, we formulate CVDP as a temporal chunk-based learning problem in which the data of different versions of a software project appear at each time step. To assess the impact of DP while using feature selection techniques in chunk-based temporal learning for improving the CVDP models, we investigated whether applying feature selection techniques separately on each version of the project when building classification-based CVDP models could improve prediction performance and the robustness of these trained CVDP models to CDs. The Friedman and Nemenyi Post-hoc test results indicate that there were statistical differences between the prediction results with and without feature selection techniques when evaluated with the AUC, Recall, and pf . CVDP models trained on the most recent versions are not always the best cross-version defect predictors. When the prediction models were updated by selecting the features from each version, the feature selection techniques could help reduce drift from the defect datasets.

Keywords: *Software Defect Prediction, Concept Drift, Class Rebalancing Techniques, Feature Selection Techniques, Empirical software engineering*

CITY UNIVERSITY OF HONG KONG
Qualifying Panel and Examination Panel

Surname: KABIR

First Name: Md Alamgir

Degree: PhD

College/Department: Department of Computer Science

The Qualifying Panel of the above student is composed of:

Supervisor(s)

Dr. KEUNG Wai Jacky Department of Computer Science
City University of Hong Kong

Qualifying Panel Member(s)

Dr. CHAN Wing Kwong Department of Computer Science
City University of Hong Kong

Dr. YU Yuen Tak Department of Computer Science
City University of Hong Kong

This thesis has been examined and approved by the following examiners:

Dr. KEUNG Wai Jacky Department of Computer Science
City University of Hong Kong

Dr. WONG Hau San Department of Computer Science
City University of Hong Kong

Dr. LEUNG Wing Ho Howard Department of Computer Science
City University of Hong Kong

Dr. LAI Richard Department of Computer Science and Information
Technology
La Trobe University

Acknowledgement

First, I thank ALLAH for blessing me with this wonderful opportunity to pursue my PhD.

My greatest expression of gratitude goes to my supervisor Dr. Jacky Wai Keung, for all his direction, encouragement, and continuous support throughout my PhD journey, which has been very key to my PhD completion. His valuable advice, constructive comments, scholarly guidance, and support helped me to make it possible to complete my PhD research studies.

I have been amazingly fortunate to work closely with and mentored by Dr. Burak Turhan and Dr. Kwabena Ebo Bennin. Their technical and editorial advice was essential to the completion of this dissertation. I would like to thank my qualifying panel members, who have reviewed my research progress reports over the past three years and provided essential and crucial feedback that has helped improve my research work. I wish to thank my labmates at City University of Hong Kong. My PhD studies would not have been possible without any financial support. I am thus very grateful to the Postgraduate Studentship, which has supported my student life in Hong Kong.

This thesis would not have been possible without the support of my family. I couldn't have the peace of mind and comfort to complete my PhD without the unflinching support and love of my dear wife Nishat Kabir, who cared for me. You deserve more than what I say here. I am very grateful to you and care for our beloved daughter, Inaaya. Finally and most importantly, my gratitude goes to my parents. I am also grateful to my colleagues

and friends, I say a big thank you for your support and love.

Contents

List of Figures	xvii
-----------------	------

List of Tables	xxi
----------------	-----

1	Introduction	1
1.1	Introduction	1
1.1.1	Problem Statement	4
1.1.2	Thesis Contributions	6
1.1.3	Thesis Organization	7
2	Background	9
2.1	Concept Drift	9
2.2	Class Rebalancing Approaches for Defect Prediction Models .	12
2.2.1	Random Over Sampling (OVER)	12
2.2.2	Random Under Sampling (UNDER)	12
2.2.3	Synthetic Minority Oversampling Technique (SMOTE)	13
2.2.4	Bootstrap Random Oversampling Examples (ROSE) . .	14
2.3	Feature Selection Methods for Defect Prediction Models	14
2.3.1	Correlation-based Feature Selection	14
2.3.2	Consistency-based Feature Selection	15
2.3.3	AutoSpearman	15
2.4	Evaluation of Defect Prediction Models	16
2.4.1	Performance Measures	16
	False Positive Rate	16

	Recall	17
	Gmean	17
	F-measure	17
	Area Under the ROC Curve (AUC)	17
2.5	Chapter Summary	18
3	A Motivating Study: Assessing the Significant Impact of Concept Drift in Defect Prediction	19
3.1	Introduction	19
3.1.1	Chapter Organization	22
3.2	Related Work	22
3.3	Experimental Setting	24
3.3.1	Datasets	24
3.3.2	Classifiers	26
3.3.3	Evaluation Measures	27
3.3.4	Statistical Test	28
3.3.5	Experimental Procedure	29
3.4	Experimental Results	31
3.4.1	Discussion	36
	Implication of Results	36
	Threats to Validity	37
3.5	Chapter Summary	38
3.6	Concluding Remarks	38
4	A Drift Propensity Detection Technique for Improving Cross-Version Software Defect Prediction Performance	41
4.1	Introduction	41
4.1.1	Motivation	42
4.1.2	Contributions	44
4.1.3	Chapter Organization	45

4.2	Related Work	45
4.3	Drift Propensity	47
4.4	DP Detector	49
4.5	Fisher’s based Statistical Significant DP Detection	52
4.6	Experimental Methodology	54
4.6.1	Benchmark datasets	55
4.6.2	Evaluation setup	55
4.6.3	Effect size computation for two-sample case	59
4.7	Experimental results and discussion	60
4.8	Threats to validity	64
4.9	Chapter Summary	65
5	The role of Class Rebalancing Techniques in Eliminating Concept Drift in Chronological Defect Datasets	67
5.1	Introduction	67
5.1.1	Chapter Organization	71
5.2	Related work	71
5.2.1	Cross-version defect prediction	71
5.2.2	CVDP considering distribution differences	73
5.3	Preliminaries	74
5.3.1	Concept of dataset shift	74
5.3.2	Data drift detection methods	75
5.4	The CDD framework	76
5.5	Methodology	80
5.5.1	Studied Datasets	80
5.5.2	Apply Class Rebalancing Approaches	82
5.5.3	Construct Cross-Version Defect Models	83
5.5.4	Calculate Model Performance	84
5.5.5	Statistical Analysis of the Experiment	85

Statistical Comparison	85
Effect size Computation	86
5.5.6 Experimental Setup	86
5.6 Results	88
5.7 Discussion	96
5.8 Threats to validity	98
5.8.1 Construct and internal validity	98
5.8.2 External validity	99
5.9 Chapter Summary	100
5.9.1 Concluding Remarks	101
6 Cross-Version Defect Prediction with Feature Selection using Tem-	
poral Chunk-based Learning	103
6.1 Introduction	103
6.1.1 Chapter Organization	107
6.2 Related Work	107
6.3 Method	109
6.3.1 Problem Formulation	109
6.3.2 Framework	112
6.4 Methodology	116
6.4.1 Benchmark datasets	117
6.4.2 Apply feature selection techniques	118
6.4.3 Construct cross-version defect models	121
6.4.4 Calculate model performance	122
6.4.5 Statistical tests	123
6.4.6 Experimental setup	124
6.5 Results	127
6.6 Discussion	140
6.6.1 Is continuously retraining a model actionable enough?	140

6.6.2	General discussion	142
6.7	Threats to validity	145
6.8	Chapter Summary	146
7	Conclusion	151
7.1	Introduction	151
7.2	Findings and Implications	152
7.3	Research Contributions	155
7.4	Future Research Opportunities	156
7.4.1	Investigating the impact on other class rebalancing techniques	157
7.4.2	Understanding the impact of configuration parameters	157
7.4.3	Investigating the impact of other feature selection techniques	158
7.4.4	Investigating the impact of deep learning models . . .	158
7.4.5	Revisiting the impact of concept drift in various setting of defect prediction models	158
A	Publications	159
	Bibliography	163

List of Figures

1.1	Impact of classification model: circle ($\circ$) and diamond ($\diamond$) represent the two class labels.	3
2.1	Decomposition of versions of the chronological datasets. . . .	10
2.2	Sources of CD. At time t , empty circle($\circ$) and empty diamond ($\diamond$) are the two class labels; At time $(t + 1)$, circle ($\bullet$) and diamond ($\blacklozenge$) filled with black color indicate the class labels of the two dimensional data distributions $\{x_1, x_2\}$	11
2.3	An example of class over sampling technique.	12
2.4	An example of class under sampling technique.	13
2.5	An example of synthetic minority oversampling technique. . .	13
2.6	An example of class bootstrap random oversampling examples technique.	14
2.7	ROC plot [150].	18
3.1	Performance curve across the twenty-five (25) versions of the <i>prop</i> datasets. 5(A) and 5(B) portrait the error-rate fluctuation in different versions. By using the Naive Bayes and Decision Tree classifier, 5(A) and 5(B) identify five and four drifts respectively that marked by vertical lines.	33
3.2	Error-rate fluctuation for the different training examples for the same model. By using Naive Bayes and Decision Tree classifiers, we identified two drifts in <i>jm1</i> dataset by the statistical test where drifts are marked by vertical lines.	36

4.1	DP detection in target feature space by changing the decision boundary. White circle and square are the two class labels at time t where black circle and square of class labels at time $(t + 1)$.	48
4.2	Adopted No-memory data management approach to manage the versions of the chronological defect datasets.	58
4.3	Evaluation process	58
4.4	Performance curve and portrait the error-rate fluctuation of the versions of the considered datasets, using NB. Significant DP's are marked by vertical lines of the chronological datasets.	62
5.1	Two-window based data distribution at timestamp $t + 7$	75
5.2	An overview of CDD framework.	77
5.3	Version-based moving window, where window size refer to one version's defect data.	77
5.4	CD detection in the studied benchmark datasets while considered all metric sets (default datasets). A green symbol indicates that there is no drift between two consecutive versions of the project (p -value <0.05 in Fisher's Exact Test); while a red symbol indicates otherwise (p -value ≥ 0.05 in Fisher's Exact Test).	89
5.5	The percentage of CD detection across the prediction models of the 36 chronological versions of 10 open-source software projects. Here, a high value defines the worst score, i.e., the CD detection rate is high.	89
5.6	CD detection in the chronological defect datasets while considered class rebalancing techniques. A red symbol indicate a drift between two consecutive versions of the project (p -value <0.05 in Fisher's Exact Test); while a green symbol indicates otherwise (p -value ≥ 0.05 in Fisher's Exact Test).	90

5.7	The percentage of CD elimination difference when applying class rebalancing techniques to 36 cross-version defect datasets. Here, a high value represents the best score, i.e., the CD elimination rate is high.	91
5.8	The CVDP performance difference, when employing class rebalancing techniques. A red line indicates no improvement (i.e., the performance difference, zero).	92
5.9	Mann-Whitney-U test win-tie-loss comparison with Cliff's $ \delta $ effect size magnitudes of without class rebalancing vs. with class rebalancing techniques across cross-version defect prediction models. The complete green box indicates significant values.	93
5.10	Scenario for periodically updated models [116].	97
6.1	An example of version-wise chunk data with different classes.	111
6.2	Framework for cross-version defect prediction using temporal chunk-based learning.	113
6.3	The performance difference when employing FS techniques to cross-version defect prediction models. A red line indicates no performance improvement.	130
6.4	CVDP models performance across the versions of the considered datasets for Recall.	131
6.5	CVDP models performance across the versions of the considered datasets for AUC.	132
6.6	CVDP models performance across the versions of the considered datasets for pf	132

6.7	Concept drift detection in the studied benchmark datasets. Figure A shows the detection of drift while considered all metric sets (all features) and Figure B exhibits the drift detection while considered AUTOS, CFS, and CONFS FS techniques. A red symbol indicate a drift between two consecutive versions of the project ($p\text{-value} < 0.05$ in Fisher's Exact Test); while a green symbol indicates otherwise ($p\text{-value} \geq 0.05$ in Fisher's Exact Test).	138
6.8	The percentage of concept drift identification when applying FS techniques. Figure A shows the detection of drift in percentage while considered all metric sets and Figure B exhibits the drift detection in percentage while considered AUTOS, CFS, and CONFS. Here, a low value represents the best score. . . .	139
6.9	CVDP-based moving window, where window size refer to one version's defect data.	141
6.10	Comparison of FS-based CVDP classification models with DS3, HALKP, and WGNCS state-of-the-art methods using F-measure and G-mean evaluation indicators. Here, horizontal red lines indicate the mean value obtained from the state-of-the-art methods. The vertical red lines separate the FS-based methods from the state-of-the-art methods.	142

List of Tables

2.1	Confusion Matrix for Two-Class Problem	16
3.1	Summary of studied 28 datasets collected from SEACRAFT repository	25
3.2	Confusion Matrix	27
3.3	Performance observation to detect the <i>concept drift</i> in terms of error-rate and p-value across twenty-five (25) versions of the <i>prop</i> datasets. The significant changes of error-rate indicate the changes of concept. These error-rates were calculated by using the Naive Bayes and Decision Tree classifier. The significant p-values are highlighted in bold.	34
3.4	Performance observation to detect the <i>concept drift</i> in terms of error-rate and p-value across three (3) versions of the <i>jm1</i> datasets. These error-rates were calculated by using the Naive Bayes and Decision Tree classifier. Significant p-values are highlighted in bold.	35
4.1	For recent chunk	52
4.2	For old chunk	53
4.3	Adapted generalized contingency table for recent and old chunk	53
4.4	Summary of Selected Benchmark Datasets	56
4.5	Description of Metrics Used in the Chronological Defect Datasets	57
4.6	Confusion matrix setup for two-class problem	57

4.7	Performance observation of the DP detection technique in the chronological defect datasets. Two-window based Fisher's Exact test results and effect sizes of the pairwise comparison for two consecutive data distributions. Large effect sizes point to a practical significance in favor of DP detection.	61
5.1	The test statistics of a pair of classifiers at time t and $t + 1$. . .	79
5.2	A statistical summary of selected chronological defect datasets.	81
5.3	Description of the static code metrics [38] used in this study. .	82
5.4	Confusion matrix.	84
6.1	The test statistics of a pair of classifiers at time t and $t + 1$. . .	114
6.2	A statistical summary of selected chronological defect datasets.	119
6.3	List of the static code metrics used in this empirical study [60].	120
6.4	Confusion matrix.	122
6.5	Summary of prediction results of cross-version defect prediction across the algorithms.	128
6.6	The Friedman means ranks of the with and without FS-based predictive models based on pf , Recall, AUC-score. The lower position of rank becomes, the better model is. ALL refers to all feature sets are considered to develop the models.	133
6.7	The result of the post-hoc analysis. 0 refers to there is no statistical difference between the pair of classifiers otherwise, there is a statistical difference in terms of P for pf and R for Recall. ALL refers to all feature sets considered to develop the predictive models.	135

Chapter 1

Introduction

1.1 Introduction

Software system is spreading through with all aspects of our daily lives, with the online banking system, automatic vehicles, and AI personal assistants such as Alexa that have involvement into finances and safety. Over time, the software system is being mature and undergoes multiple changes by incorporating new requirements, and the system becomes more complex. The increasing demands of using such a complex software system make it extremely difficult to ensure quality. Based on the US National Institute of Standards and Technology (NIST) survey, the software failure cost the national economy ranges from \$22.2 to \$59.5 billion per year [142]. Other studies reported that the enormous loss of revenue for the software system causes due to software bugs [138]. In 2018 and 2019, two subsequent fatal plane crashes happened on Boeing 737 MAX airplane killing everyone on board. Investigations reported that there was a defect in the Boeing simulation software used by the pilots [79]. The two deadly crashes damaged colossal property and cost of human life [65]. Such fatal cases arise a realization that quality improvement of such critical software systems has become more and more crucial. Software Quality Assurance (SQA) assures such a critical software system that satisfies a specific quality level by maintaining the SQA efforts [154].

A large number of recent research has focused on the proper utilization of SQA efforts. Out of the research works, software defect prediction (SDP) has received significant attention in the past decade. In SDP, the classification models are used to classify the classes or modules that are likely to be faulty in the future (e.g., [77] and [66]). Efficient identification (prediction) of defective or faulty modules helps to prioritize the SQA efforts [155]. In particular, the classification models are built using machine learning (ML) techniques by utilizing the historical data (from which the prediction model is built) that assists software developers to focus on the modules, which need special attention concerning allocate the scarce testing resources, code inspection and others [15, 47].

Recently, several classification-based SDP models based on historical defect datasets [31, 72, 103] have been developed. These defect datasets, however, do change over time [16]. A long-running software project experiences several versions after a stable period, appearing in temporal order and making non-stationary data distributions. Over time, the relationship between data variables changes, i.e., referred to as *concept drift*. *Concept drift* appears in the data distribution that changes after a stable period by changing the prior probabilities of classes or class conditional probabilities [118]. Figure 1.1 shows the impacts of concept drift in a classification model. It can be seen that there is a clear shift in the decision boundary after changes in the incoming data in Figure 1.1 (b) and (c), implying that *concept drift* has a significant effect on the classification performance. Based on Bayesian decision theory [32], as an example, the classification model can be framed for a given instance A based on the prior probabilities $p(c|A)$ for the classes $c \in 1, \dots, n$ as $p(c|A) = p(c)p(A|c)/p(A)$ where $p(A) = \sum_1^n p(c)p(A|c)$. Figure 1.1 (b) shows the shifts in the decision boundary, exhibiting that concept drift occurs in the prior probabilities $p(c|A)$ by changing the feature space as the decision boundary shifts. In Figure 1.1 (c), the decision boundary remains the same

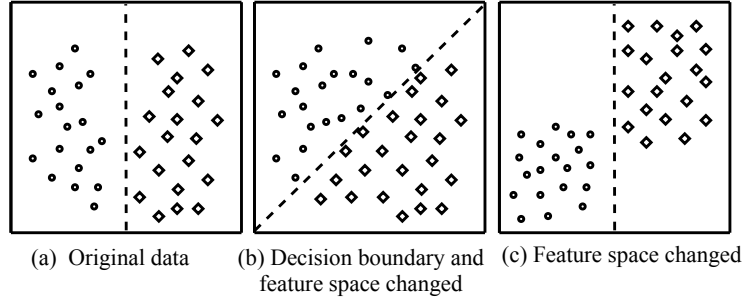


FIGURE 1.1: Impact of classification model: circle (○) and diamond (◇) represent the two class labels.

as in the original form (Figure 1.1 (a)) and concept drift occurs wherever the class conditional probabilities $p(A|c)$ are prone to change with the feature space.

The obvious fact is that if the historical data changes over time, the classification model becomes outdated [113]. The research efforts that studied the long-running software projects for SDP [31, 72, 78, 95, 98, 103, 153, 156, 179, 181, 183] considered the learning distributions are stationary (i.e., no *concept drift*). Ekanayake et al. [35, 36] observed that changes (fixes) to software files tend to change the concept in data distribution [36] that consequently impacting the performance of software defect predictions. Their studies suggest considering *concept drift* that can improve the performance of prediction models. This thesis focuses on the effect of *concept drift* and the performance of prediction models.

Before we examine in details, we initially conducted an empirical study to detect drift in the chronological defect datasets by adopting the drift detection strategy DDM (Drift Detection Method) [43] and to evaluate its statistical significance using the chi-square test with the Yates' continuity correction [135]. Two projects with different versions are extracted from NASA [2] and Jureczko defect datasets [82]. In details, three versions of *jm1* and 25 versions of *prop* projects were used. The *jm1* project contains 25,157 modules and *prop* contains 67,346 modules, respectively. From our experiment,

we find the existence of *concept drift* in the defect datasets affects the prediction performance. The models are trained with the same settings to monitor the performance curves. We observe the fluctuation of the error-rates. Our findings suggest that the performance of defect prediction models is highly influenced by concept drifts in defect datasets that impact the quality of prediction.

1.1.1 Problem Statement

We formulate the thesis statement based on the experimental results from our preliminary investigation (See Chapter 3).

Thesis Statement: *Concept drift influences the prediction performance of SDP models. Empirical studies on the impact of concept drift on the performance of SDP models are needed to achieve more practical implications from SDP models.*

Due to the rapid development of the software industry, the software development environment becomes an evolving process where a mature software project undergoes multiple changes by incorporating new requirements. Over time, the software project's version is updated and maintains a chronological order of the project versions [183]. The updated version becomes more complicated during this process due to refactoring or module deletion from the previous version. This causes *concept drift* i.e., distribution changes among the versions of the same software project over time. Few studies [16, 35, 36, 58] observed that prediction accuracy fluctuates over the chronological datasets that are used for model training. The impact of concept drift on the performance of classification-based SDP models has never been explored over the chronological defect datasets. In addition to this, Menzies et al. [124] suggested improving the training datasets' information content for improving prediction models. To improve the information content of the

training datasets that appear over time, the impact of concept drift should be considered, which has not been extensively investigated.

Furthermore, several researchers have raised concerns about the impact of datasets used for defect prediction models in software engineering prediction research. Researchers have suggested that the datasets should be pre-processed properly before training the prediction models [52]. For processing the defect datasets, several SDP models have been developed considering the imbalance nature of class distribution [15, 161, 172] of the defect datasets. Researchers have tried to show performance improvement for SDP models by adopting rebalancing techniques [15, 161]. Tantithamthavorn et al. [161] showed the impact of class rebalancing techniques on SDP models. They pointed out that drift develops when the class distributions of training and testing sets are not similar. Moreover, the change of prior probabilities of classes is responsible for drift [118]. Since class rebalancing techniques try to produce a similar representation of two classes, it may eliminate drift and impact the performance of classification-based SDP models, which has not been thoroughly investigated yet.

In an attempt to improve the performance of SDP models, feature selection techniques are known to have a significant impact on SDP. Prior works [46, 93, 180] showed that carefully selected features could achieve acceptable prediction performance in SDP. In the studies [16, 35, 36], the authors attributed that the prediction results fluctuate due to the changes in the data features. Therefore, feature selection techniques may avoid concept drift due to the variation of selecting features in each version of the software project, which has not yet been examined.

Thus, this thesis aims to investigate and assess the impact of concept drift on SDP performances. A systematic investigation is conducted to examine whether class rebalancing could help eliminate concept drift from chronological defect datasets. In addition to this, an empirical study is conducted to

investigate whether applying feature selection techniques separately on each version of the software project when building the classification-based SDP models could improve prediction performance and examine the robustness of the model to concept drift.

1.1.2 Thesis Contributions

This thesis investigates the impact of concept drift on the performance of SDP models. A comprehensive investigation of class rebalancing and feature selection techniques on the performance of SDP models and examine the robustness of the prediction models to concept drift. We present a variety of contributions to the field. We highlight the key contributions as follows:

1. We empirically detect drift in the chronological defect datasets by adopting the strategy of the drift detection method. The statistical significance test, the chi-square test with the Yates continuity correction, enables us to understand better the adopted drift detection technique that identifies the types of concept drift in software defect datasets. The empirical study suggests that the classification-based SDP models need to be updated after the drift to obtain better prediction performances.
2. We develop a drift detection method that potentially identified the drift's where the prediction accuracy varies in the software versions' during the learning process and observes that an exponential increase of the error-rate addresses the drift's in the chronological defect datasets.
3. We propose a concept drift detection framework to detect drift in the chronological defect datasets and investigates the feasibility of alleviating drifts in the chronological defect datasets.

4. We empirically check whether class rebalancing can help reduce the detection rate of concept drift from chronological defect datasets and impact the prediction performance for cross-version defect prediction.
5. We contribute by providing recommendations on which feature selection method to use that is robust for predicting defects for cross-version software projects.

1.1.3 Thesis Organization

The thesis is organized as follows. Chapter 2 summarizes several related works and state-of-art literature relevant to drift detection techniques. Data management techniques for a chunk-based learning environment are also presented in this chapter. In Chapter 3, a motivating study investigates the existence of concept drift and its associated effects on software defect prediction performance. Since the software project versions appear as chunks by changing the class distribution and data features of the consequent version of software projects, we are driven to examine and conduct more experiments to assess the impact and benefits of class rebalancing techniques and feature selection techniques on the performance of defect prediction models. In Chapter 4, a drift detection method is proposed that monitors the learning algorithms error-rate by employing a no-memory data management technique. In Chapter 5, we investigated whether class rebalancing techniques can eliminate drift from the chronological defect datasets and impact the prediction performance of defect prediction models. We found that up to 50% of the chronological defect datasets are drift-prone while applying the most popular classifiers used from the defect prediction literature. Additionally, the

class rebalancing techniques had a positive impact on the prediction performance for cross-version defect prediction by correctly classifying the cross-version defective modules and detected less drifts (i.e., 31%) on the resampled datasets. In Chapter 6, we assessed the impact of feature selection techniques on the performance of cross-version defect prediction using temporal chunk-based learning. We applied feature selection techniques separately on each version of the project when creating cross-version defect prediction models to check whether it improved prediction performance and examine the robustness of the model to concept drift. The empirical results indicated that there were statistical differences between the prediction results with and without feature selection techniques when evaluated with the AUC, Recall, and *pf*. Cross-version defect prediction models trained on the most recent versions are not always the best cross-version defect predictors. The AutoSpearman outperformed among the studied feature selection techniques. When the prediction models were updated by selecting the features from each version, the feature selection techniques could help reduce drift from the defect datasets. We conclude the thesis in Chapter 7 with a discussion of findings from all the experiments conducted in the previous Chapters and highlight some promising future directions.

Chapter 2

Background

2.1 Concept Drift

The concept of dataset shift or drift was first identified in the data-stream mining community [112, 167]. Detecting dataset drift is very challenging in non-stationary environments because the data change over time. Lu et al. [112] pointed out that the uncertainty of a dataset is an inherent property of streaming data. Change is unavoidable and inevitably deteriorates the prediction results. The concept of dataset shift in software engineering-prediction models was introduced by Turhan [167], who cautioned about its effects and attempted to account for changes in data distributions. However, drift-identification techniques used for analysis of chronological software datasets remain suboptimal. As shown by [35, 36, 86], the accuracy of such prediction models degrades over time due to the changes in software defect datasets.

The concept of dataset shift refers to the changes of data distributions. More formally, the problem is formulated as follows. Suppose that v is the version index of a chronological dataset at a discrete time index.

$$\Xi = [v_1, v_2, v_3, \dots, v_N] \tag{2.1}$$

$$\Phi = \begin{bmatrix} m_{11} & m_{21} & \dots & m_{N1} \\ m_{12} & m_{22} & \dots & m_{N2} \\ \vdots & \vdots & \dots & \vdots \\ m_{1p} & m_{2p} & \dots & m_{Np} \end{bmatrix} \quad (2.2)$$

The total dataset consists of the number of N in versions Ξ of the dataset. By Equations 2.1 and 2.2, we can define the dataset as $\{\Xi, \Phi\}$ where each version v contains the number of p in modules $m = 1, 2, 3, \dots, p$. In Equation 2.2, matrix Φ is constructed to describe the modules m of the versions Ξ . As an example, suppose there are two versions, v_1 and v_2 , at the discrete time t . The data of these two distributions change after a certain period of time. If the joint probability changes in the data of versions Ξ , it can be defined as $v_1 \neq v_N$ where $(1, N) = 1, 2, 3, \dots, N$.

As seen in Figure 2.1, we decompose the chronological datasets according to version v that contains the number of p in modules m at time points t . We separate the corresponding versions with their modules. CD occurs between the versions Ξ if their distributions change after a stable period.

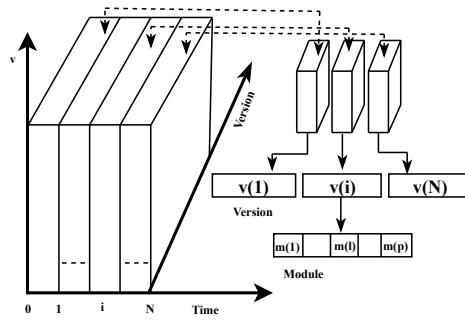


FIGURE 2.1: Decomposition of versions of the chronological datasets.

The CD in distributions can manifest in various forms by changing the probability functions A and b at time t .

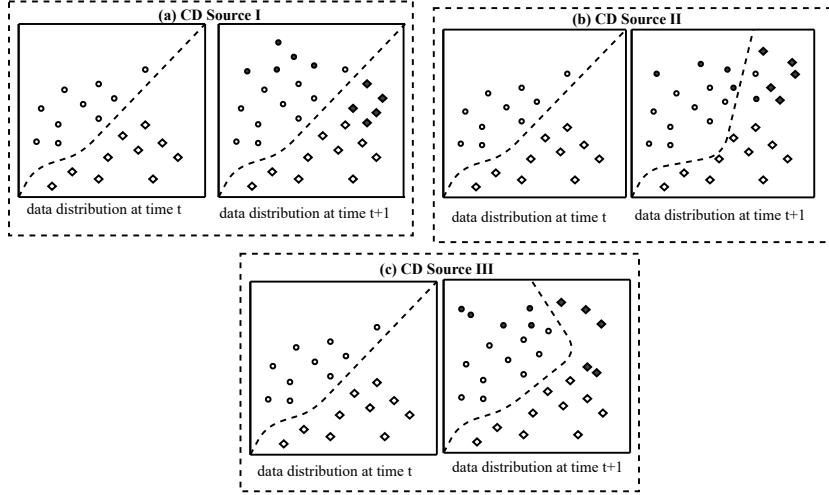


FIGURE 2.2: Sources of CD. At time t , empty circle($\circ$) and empty diamond ($\diamond$) are the two class labels; At time $(t + 1)$, circle ($\bullet$) and diamond ($\blacklozenge$) filled with black color indicate the class labels of the two dimensional data distributions $\{x_1, x_2\}$.

In Figure 2.2, we portray how CD can occur for classification problem that contains two class labels. For this, we assume a two-dimensional data distribution $A = \{x_1, x_2\}$ with the class labels $b = \{y_1, y_2\}$, where empty circles ($\circ$) and diamonds ($\diamond$) indicate the data distributions at time t . Black circles ($\bullet$) and black diamonds ($\blacklozenge$) indicate the data distributions at time $t + 1$. CD has the effect of changing the joint probability, which corresponds to *iff* $\exists t : P_t(A, b) \neq P_{t+1}(A, b)$. The joint probability $P_t(A, b)$ can be defined as $P_t(A, b) = P_t(A) \times P_t(b|A)$. Figure 2.2 depicts the following ways in which CD can occur:

1. **CD Source I:** In Figure 2.2(a), CD occurs at time $t + 1$ when $P_t(A) \neq P_{t+1}(A)$. In this case, the CD does not influence the decision boundary. Therefore, $P_t(A)$ and $P_{t+1}(b|A)$ are the same.
2. **CD Source II:** In Figure 2.2(b), CD changes the decision boundary at time $t + 1$ when $P_t(A) \neq P_{t+1}(b|A)$. In this case, $P_t(A)$ and $P_{t+1}(A)$ are the same.
3. **CD Source III:** In Figure 2.2(c), CD results in changes in both cases, $P_t(A)$ and $P_t(b|A)$, at time $t + 1$. This alters the feature space and in turn

the decision boundary when $P_t(A) \neq P_{t+1}(A)$ and $P_t(A) \neq P_{t+1}(b|A)$.

2.2 Class Rebalancing Approaches for Defect Prediction Models

2.2.1 Random Over Sampling (OVER)

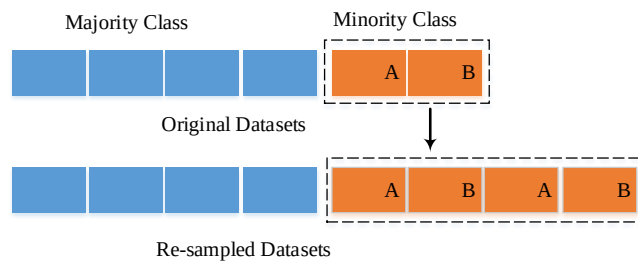


FIGURE 2.3: An example of class over sampling technique.

The random over sampling technique is called as up-sampling that randomly increases the samples by replicating the number of minority class examples. In the re-sampled datasets, the minority class will be the same size as the majority class examples. This class rebalancing technique doesn't loss information. With the same time, it may lead to overfitting due to the replication of the original minority class examples. This technique has been widely implemented in the empirical studies of software engineering [15, 161]. Figure 2.3 illustrates an example of random over sampling technique.

2.2.2 Random Under Sampling (UNDER)

The random under sampling (UNDER) technique reduces the number of majority instances of the original datasets by randomly removing the instances to be the same number of minority instances. In other words, the minority instances are kept intact whilst the majority instances are decreased. The UNDER is also called as down-sampling. UNDER is more appropriate because

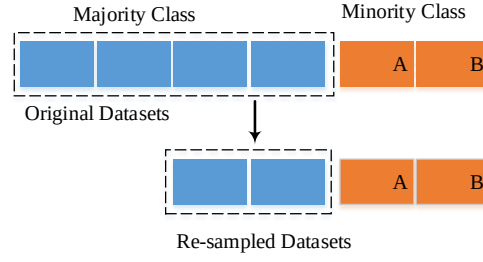


FIGURE 2.4: An example of class under sampling technique.

the datasets have a relatively large number of majority instances to avoid huge training time. A significant disadvantage of UNDER is that it removes the majority class instances that may cause loss of information, influencing the model's performance.

2.2.3 Synthetic Minority Oversampling Technique (SMOTE)

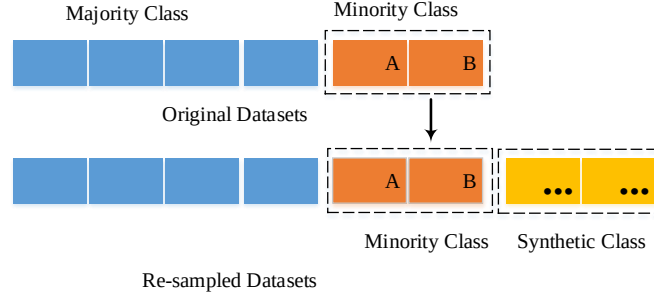


FIGURE 2.5: An example of synthetic minority oversampling technique.

Synthetic Minority Oversampling Technique (SMOTE) was proposed by Chawla et al. [22] that creates artificial instances (synthetic instances) in the area of minority class region. SMOTE was proposed to combat the drawback of OVER and UNDER class rebalancing techniques. For each minority class instance, SMOTE executes the following steps:

- Select a minority class instance and find the k minority class instances that are nearest neighbors instances, where $k=5$.

- Create new synthetic instances that join the each neighbour of that selected minority class instance.

2.2.4 Bootstrap Random Oversampling Examples (ROSE)

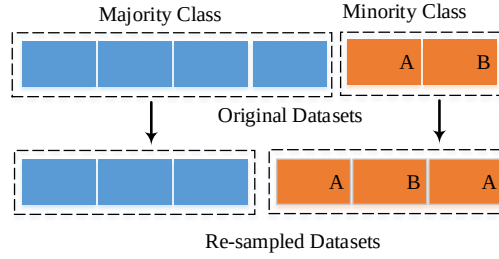


FIGURE 2.6: An example of class bootstrap random oversampling examples technique.

Bootstrap Random Oversampling Examples (ROSE) [115] generates synthetic balanced instances in the feature space. ROSE creates augmented instances by combining OVER and UNDER techniques according to the approach of smoothed-bootstrap. It handles instances for categorical and continuous data and generates synthetic instances [161].

2.3 Feature Selection Methods for Defect Prediction Models

2.3.1 Correlation-based Feature Selection

In prior studies [46, 76, 93, 146, 180] of SDP, as a filter-based subset selection technique, correlation-based feature selection [53] is reported as best performing technique. This feature selection technique selects a subset of features based on correlation, meaning that the selected features have strong correlations with the class levels. In the other word, the selected feature sets share the most substantial relationship with the outcome variable and having

a low correlation with the other feature sets. In our study, we consider this technique as one of the benchmark method, which searches the best subset of features (i.e., metrics set) among the considered metrics.

2.3.2 Consistency-based Feature Selection

Consistency-based feature selection [26], a deterministic technique, selects the feature sets by using consistency measure. This technique aims at selecting the consistent features whose rate is equal to all the features. It uses consistency of the class levels of the features and selects the best subset of features from the original datasets. This technique utilizes search techniques i.e., BestFirst, Genetic, and Rank search for selecting minimal feature sets. Based on the studies of Ghotra et al. [46], BestFirst search performed best in selecting the best feature sets in PROMISE data sets.

2.3.3 AutoSpearman

AutoSpearman is an automated feature selection approach (i.e., hybrid-based feature selection approach) based on Spearman rank correlation and variance inflation factor (VIF) analysis [74]. Recently, Jiarpakdee et al. [75] proposed this approach for interpreting defect models. It alleviates correlated metrics [76]. It elects a feature that achieves the least correlation from the feature sets to select one feature from the feature space. This technique is performed in unsupervised fashion where outcome levels are not required. We use the implementation of AutoSpearman using the AutoSpearman function as provided by the AutoSpearman R package [74].

TABLE 2.1: Confusion Matrix for Two-Class Problem

	Positive Prediction	Negative Prediction
Actual Positive	TP	FN
Actual Negative	FP	TN

2.4 Evaluation of Defect Prediction Models

2.4.1 Performance Measures

Confusion matrix is considered for the evaluation of the most prediction models [14]. In defect prediction literature, this matrix is widely used for defective modules classification and prioritization [143]. In this matrix, the defect-prone class is considered as the positive instance and non defect-prone class is classified as the negative instance. From Table 2.1, the outcomes are described as follows:

- True Positives (TP): defective modules predicted as defective and classified correctly.
- False Positives (FP): non-defective modules predicted as defective and also classify incorrectly as defective.
- True Negatives (TN): non-defective modules predicted as non-defective and classified correctly.
- False Negatives (FN): defective modules classify incorrectly as non-defective.

False Positive Rate

False Positive Rate is referred as to probability of false alarms (pf). pf is measured by the total number of non-defective modules predicted as defective

among all the non-defective modules.

$$pf = \frac{FP}{FP + TN} \quad (2.3)$$

Recall

Recall is referred as probability of detection (pd) or true positive rate. Recall is calculated by the ratio of correctly classified the predictive modules to the total number of total number of predicted defective modules.

$$Recall(pd) = \frac{TP}{TP + FN} \quad (2.4)$$

Gmean

Gmean is the geometric mean of recall and precision where recall is $\frac{TP}{TP+FN}$ and precision is $\frac{TP}{TP+FP}$. Gmean is calculated by the following equation:

$$Gmean = \sqrt{precision \times recall} \quad (2.5)$$

F-measure

F-measure is the harmonic mean of recall and precision. F-measure is the trade-offs between recall and precision. For example, we will get high values of recall if the prediction of test data as defective, but we will get low precision. We calculate F-measure by the following equation:

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.6)$$

Area Under the ROC Curve (AUC)

Area Under the ROC Curve (AUC) is the curve measured the under ROC (receiver operating characteristic) curve area. ROC curve is plotted by using true positive rate (pd) and false positive rate (pf) (Figure 2.7). The value of

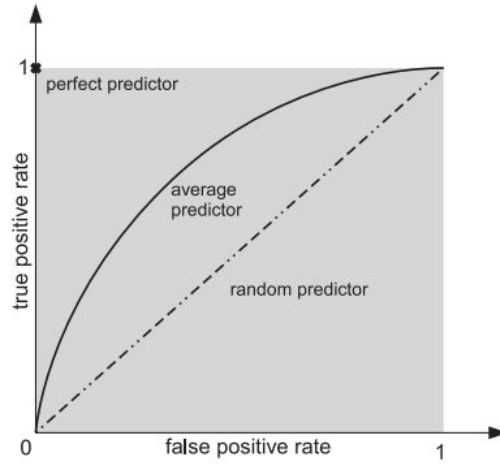


FIGURE 2.7: ROC plot [150].

curve that is close to top left corner considered as better value. A model is perfect when the pd value of 1 and pf value of 0. The curve in the Figure 2.7 implies that the probability of detecting defective or non-defective modules is equal, which has been recommended in the studies of SDP. The higher value of AUC refers to better prediction performance and when the AUC value is 0.5, the model is considered as a random prediction model [145].

2.5 Chapter Summary

In this chapter, definition and source of concept drift are presented. Popular class rebalancing techniques and feature selection methods are also explained. In the next chapter, we conduct a motivation study to assess the impact of concept drift in software defect prediction. The empirical study indicates that the concept drift occurs in software defect datasets, and its existence subsequently degrades the performance of software defect prediction models. We suggest concept drift should be considered by software quality assurance teams when building the defect prediction models.

Chapter 3

A Motivating Study: Assessing the Significant Impact of Concept Drift in Defect Prediction

3.1 Introduction

Software defect prediction models are developed from historical software data to improve the quality by fixing the bugs of the software [95, 98]. These types of models have been developed over years to identify bugs in the source code to reduce the development cost [174]. If the historical data (fixed distributions) changes over time, the prediction model becomes outdated [31]. Rahul et al. [95] caution that if the prediction model changes because of the changing historical data, managers will get less faith in the potential of the trained prediction model leading to inefficient use of scarce testing resources.

The characteristics of data distribution changes over time i.e. referred to as *concept drift*. In another way, the changes in probability distribution in high-speed data streams is also referred to as *concept drift*. Recently, several defect prediction models based on historical data (fixed distribution) [31, 72, 78, 95, 98, 103, 153] have been proposed and developed. These defect datasets

do change over time [95]. Rahul et al. studied over 15 defect datasets (i.e., chronological datasets). Their results showed that the accuracy of defect prediction model varies in these chronological datasets. According to Dong et al. [31], a well-trained prediction model could give more inaccurate results if the incoming data starts to drift over time. Inconsistencies between the historical data and current incoming data makes thus makes the model training process more challenging [132, 157] and learning process more complicated [103]. This leads to wrong prediction which disrupts the goal of achieving quality software [95].

Several software defect prediction models [31, 72, 78, 95, 98, 153] developed over the years are constructed with C&K (Chidamber & Kemerer object-oriented) [24] and process metric datasets. To significantly improve prediction performance, these studies consider different factors such as feature selection methods, data preprocessing techniques, sampling strategy and others. The impact of the data distribution changes over time on prediction performance has not been considered by many. Few studies [16, 35, 36] observed that prediction accuracy fluctuates when chronological datasets are used for model training. They attribute the fluctuating results to changes in the data features. In the data mining literature, changes to data distribution over time is referred to as *concept drift*. The first to investigate the notion of *concept drift* in software defect prediction was Ekanayake et al. [35, 36]. By examining the bug fixing process and files of software programmers, they observed that changes (fixes) to software files tend to change the concept in data distribution [36] consequently impacting the performance of defect predictions.

According to *concept drift* literature, it is a very complex task to learn under it due to the streaming data [113]. There have been developed many methods to detect the *concept drift* over the last years as well as to know when and how the data starts to drift that affects in the accuracy of prediction

models [6, 43]. Drift detection methods are applied in many domains such as fraud detection (where the target variable is fraudulent) and weather prediction (where the target variable is temperature). The studies by Ekanayake et al. [35, 36] provides the foundation to investigate for *concept drift* in software defect prediction studies and to examine and assess the impact of *concept drift* on prediction performance. The studies of Ekanayake et al. [35, 36] is limited and not comprehensive considering most defect prediction models are built using static code metrics. Additionally, evaluation of the statistical significance of *concept drift* effects on prediction performance using robust statistical tests are yet to be done. To our best knowledge, this is the first attempt to identify drift in software engineering domain (i.e. detecting the defect in the source code where the target variable is 'defect') by applying with *concept drift* detection method and statistically favored chi-square test with the Yates' continuity correction.

In this chapter, we conduct empirical experiments to detect drift in the chronological defect datasets by adopting the drift detection strategy DDM (Drift Detection Method) [43] and to evaluate its statistical significance using the chi-square test with the Yates' continuity correction [135]. Two projects with different versions are extracted from NASA [2] and Jureczko datasets [82]. In details, three versions of *jm1* and 25 versions of *prop* projects were used. The *jm1* project contains 25,157 modules and *prop* contains 67,346 modules respectively. Since DDM works on the two-time windows-based data distributions [43], for the *jm1* dataset, we increase 500 modules in each trial as the recent window (i.e. new data) and compare it with the previous trial (i.e. historical data or previous window) to find the changes between the two trials. For the *prop* datasets, we add the instances of the previous versions (historical data or previous window) and compare with the added version (new window or new data). In summary, this work makes the following contribution:

- investigate the existence and types of *Concept Drift* in software defect datasets
- investigate the significance of *Concept Drift* in software defect prediction

3.1.1 Chapter Organization

The remaining parts of this chapter are structured as follows. Section 3.2 discusses the related studies on defect prediction. In section 3.3, we describe our experimental methodology and evaluation measures. Section 3.4 presents the empirical results. Discussion together with the threats to validity of this empirical study is presented in Section 3.4.1. Finally, section 3.5 concludes the summary of our work and future directions.

3.2 Related Work

Several studies have been conducted in the domain of Software Defect Prediction (SDP). Most often, researchers are concerned with the prediction performance instability in SDP. Bernstein et al. [16] argued that prediction performance depends on the temporal features of the data. They conducted an experiment by using non-linear models to find the hidden relationship between the features of the data that controls the performance of predictions models. In their experiment, they extracted the temporal features of the data from the Eclipse projects obtained from the bug reporting repository, Bugzilla¹ and version control system, CVS². Temporal features such as revision numbers and reported issues within a time frame were extracted for the experiment. By using the tree-based induction algorithms to compare the features and monitoring the changes of the features, the authors assumed that

¹<https://www.bugzilla.org/>

²<https://www.nongnu.org/cvs/>

the prediction performance varies because of the changes in the bug generation process. After their first experiment (i.e., finding the hidden relationship between the data features) [16], Ekanayake et al. [35] extended their previous study and investigated why the prediction quality varies in time in terms of accuracy. They increased their experimental datasets by extracting other three open source projects, Mozilla, Open Office, and Netbeans. They noticed that the defect prediction model becomes unstable and unsuitable due to the changes of the features. Based on our literature, this is the first work in software defect prediction where *concept drift* is adapted to find the changes in the concept (i.e., bug generation process). Their experiment illustrates that the concept changes due to the changing of the bug fixing process.

Furthermore, Ekanayake et al. [36] investigated the instability of the accuracy of the defect prediction model by observing the features of the projects that cause the changes of the quality of prediction model as well as to find the period of consistency of the quality of defect prediction. They proposed an accuracy estimation approach that suggests the changes in the accuracy of the prediction model. To assess their proposed model, the features that are extracted from the streaming data are used to produce the accuracy of the model by comparing the predicted values of the different trials. They took a little amount of data at a time on each run to obtain the accuracy for comparing the accuracies of the previous run and recent run. Thereby, the predicted model is evaluated to find the stability of the prediction performance. In their experiments, they observed that the data vary over time that affects the quality of the prediction model. By analyzing the historical data for different time periods, they observed that the number of authors changes to a file in fixing bugs does affect the quality of the prediction model. Finally, they conclude that the quality of the prediction model significantly varies over time due to feature changes of software projects.

Note that these studies [16, 35, 36] investigated the reason why the quality

of the prediction model changes over time and to identify the stability of the prediction model on different versions of open source bug datasets. Aforementioned that these datasets are formed by extracting the data features such as versioning system features, bug tracker features, component features, and project level features.

These previous works examined the open source projects' features (e.g., bug tracker features) to find the hidden relations that influence the quality of the prediction model and for the reason of the stability and variability of the accuracy of the prediction models as well. By conducting their experiments, they got the notion of *concept drift* in software projects affecting the prediction quality.

Over the years, several prediction model [31, 72, 78, 95, 98, 103, 153] using the C&K metrics based defect datasets have been developed. In our study, we investigate the C&K metrics-based defect datasets to identify drifts and its types by adapting the strategy of the drift detection method, DDM and to evaluate its statistical significance by adopting the statistical favoured test, the chi-square test with the Yate's Continuity Correction.

3.3 Experimental Setting

In this section, we discuss the datasets used in this experiment which are publicly available in the SeaCraft repository. Experiments are conducted using the statistical tool R [162]. The packages, Caret [96] and e1071 [29] were used in the experiment.

3.3.1 Datasets

In this experimental study, we considered two projects (i.e., *jm1* and *prop*) that contain in total 28 versions collected by Menzies et al.[2] and Marian

TABLE 3.1: Summary of studied 28 datasets collected from SEACRAFT repository

Studied Dataset	Project	Version	# of Modules	# of Faulty Modules	% Faulty Modules
NASA	jm1	1.0	7782	1672	21.49%
		1.2	9593	1759	18.34%
		1.3	7782	1672	21.49%
Jureczko	prop	4	3022	264	8.74%
		9	4255	149	3.50%
		40	4053	466	11.50%
		44	4620	389	8.42%
		85	3077	948	30.81%
		92	3557	1269	35.68%
		121	2998	425	14.18%
		128	3527	220	6.24%
		157	2496	367	14.70%
		164	3457	319	9.23%
		185	2825	268	9.49%
		192	3598	85	2.36%
		225	1810	147	8.12%
		236	2231	76	3.41%
		245	1962	103	5.25%
		256	1964	625	31.82%
		285	1694	177	10.45%
		292	2285	209	9.15%
		305	2344	89	3.80%
		318	2395	365	15.24%
		347	2871	162	5.64%
		355	2791	924	33.11%
		362	2854	213	7.46%
		452	256	33	12.89%
		453	192	20	10.42%

Jureczko [82]. These datasets are collected from the SeaCraft repository³. Table 3.1 provides a general summary of our selected datasets. These datasets are comprised of C&K (Chidamber & Kemerer object-oriented) metrics [158]. More explanations of the metrics could be found in [24]. These datasets were widely used by several researchers in software defect prediction [95, 131, 132]. Three (3) versions of *jm1* and twenty-five (25) versions of *prop* projects were used in this experiment. Although these datasets consist very high imbalanced percentage of faulty modules. For each experimental run, we increased 500 instances of *jm1* datasets and added the next version with the previous version of *prop* datasets to monitor the prediction performance.

³<https://zenodo.org/communities/seacraft/>

3.3.2 Classifiers

In software defect prediction, it has been a challenge as to which classifier is better for building efficient predictions? Agrawal et al. [1] conducted a rank-based study to select better classifiers in their experiments. They argued that the classifier is "good" if it compares with multiple classifiers considering data imbalance problem. According to their study, the Naive Bayes and Decision Tree classifiers were ranked as "good". The classifiers were trained on C&K metrics. Worth mentioning that we selected C&K metrics-based software defect datasets since it is widely used than process metrics [1].

In our experiment, we used Naive Bayes and Decision Tree for detecting *concept drift* in the C&K metrics-based datasets. In the drift detection literature (i.e., error-rate based drift detection methods), the classifiers, Naive Bayes and Decision Tree, were used [43, 71, 135]. Nishida et al. [136] conducted an experiment to detect *concept drift* by leveraging prior knowledge of the context. In their experiment, they used decision tree as a base learner to produce the accuracy measure. Gama et al. [43] proposed DDM by studying the learning problem in non-stationary environments. They produced on-line error-rate for monitoring the changes in data streams. They also used the decision tree classifier to produce the error-rate. Cabral et al. [100] implemented efficient Fisher's Exact test for detecting *concept drift* in streaming datasets. In their experiments, they used Naive Bayes as a learner to test the methods. In addition, these classifiers have been frequently used [100] in the drift detection MOA framework⁴.

⁴<https://moa.cms.waikato.ac.nz/>

3.3.3 Evaluation Measures

Confusion matrix is considered for the evaluation of the most prediction models [14]. In defect prediction literature, this matrix is widely used for defective modules classification and prioritization [143]. In this matrix, the defect-prone class is considered as the positive instance and non defect-prone class is classified as the negative instance. From Table 3.2, the outcomes are described as follows:

- True Positives (TP): defective modules predicted as defective and classified correctly.
- False Positives (FP): non-defective modules predicted as defective and also classify incorrectly as defective.
- True Negatives (TN): non-defective modules predicted as non-defective and classified correctly.
- False Negatives (FN): defective modules classify incorrectly as non-defective.

TABLE 3.2: Confusion Matrix

	Positive Prediction	Negative Prediction
Actual Positive	TP	FN
Actual Negative	FP	TN

In this study, the error-rate is computed to assess the performance of the learners and monitor the fluctuation to detect the changes in data distributions. The detailed description of these measures can be found in [143] and [10]. The mathematical definitions of these measures are presented in equation 3.1 and 3.2.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (3.1)$$

$$Error - rate = 1 - Accuracy \quad (3.2)$$

Nishida et al. [135] mentioned in their drift detection method that if the accuracy of the two-time window-based data distributions significantly decreases over time, that indicates the concept is changing. On the other hand, if the accuracy remains the same across both distributions, the concept is not changing (i.e., stationary). According to Gama et al. [43], the error-rate decreases if the distribution is stationary. Significant increases of error-rate reveal that the distribution changes.

3.3.4 Statistical Test

The statistical test, chi-square test with Yates' continuity correction was recommended by Nishida et al. [135], Gama et al. [43], Minku et al. [126], and Lu et al [113] for comparing two data distributions for identifying the drift data point. This robust test was also implemented and suggested for the detection of *concept drift* by [135].

We took into consideration the statistically favored, chi-square test with Yates' continuity correction [152] to discover the existence and its statistically significant difference between two distributions that change over time. Since, our evaluation measures were illustrated in 2×2 probability table. This test suited for 2×2 contingency table used by [152]. The statistically significant value was computed using $\alpha = 0.05$. Nishida et al. [135] conducted an experiment for detecting *concept drift* in two-time window-based data distributions named as STEPD (Statistical Test of Equal Proportions Detection) where the warning level for drift detection was defined as (p-value) as 0.05 (i.e., the observed significant level) [135]. Later, this determined significant level was used as the warning level in our experiment that was executed by different researchers [100, 113] for drift detection.

In our experiment, we applied the following hypothesis:

- H_0 : There is no difference between two distributions.
- H_1 : There is a difference between two distributions.

If the observed significant level is less than the significant level (0.05), the null hypothesis (H_0) is rejected and alternative hypothesis (H_1) is accepted that is to say, *concept drift* is detected.

The test statistic is calculated as follows:

$$T(CC_o, CC_r, OD_o, RD_r) = \frac{|a| - 0.5b}{\sqrt{\hat{p}(1 - \hat{p})b}} \quad (3.3)$$

where,

$$a = |CC_o/OD_o - CC_r/RD_r|,$$

$$b = (1/OD_o + 1/RD_r),$$

and

$$\hat{p} = (CC_o + CC_r)/(OD_o + RD_r).$$

In equation 3.3, the CC_o is the number of correct classifications among the overall, OD_o distributions and CC_r = number of correct classifications among the recent, RD_r distributions.

3.3.5 Experimental Procedure

We investigated the existence and types of *Concept Drift* in software defect datasets. For this purpose, we used two open source defect datasets, *jm1* and *prop* as described in Section 3.3.1. The *jm1* dataset includes three versions from where initially we took 500 instances in the first experimental run and increased it by 500 instances for subsequent runs. For each run, the model was trained on the data by taking 20% as the training set and rest of the data

was used as test data to compute the performance measures. Afterwards, the next batch of data (500 instances) was added with the old (previous) instances and maintained the same procedure to build the model to compute the error-rate. This process was iterated until the datasets have been used for this experiment. By computing error-rate, it was monitored to detect the changes (significant increases or decreases of error-rate) for drift detection as followed in the drift detection approaches by [18, 43, 68]. As mentioned, we repeated the same procedure to build the model that helped to produce the error-rate of each run. By using the equation 3.3, the p-values were calculated of each trial to find the statistical significance. Algorithm 1 illustrates the process followed for the experiments. From the algorithm 1, steps 1 to 2 initializes the parameter s to take the inputs. These parameters are the input variables for the experiments. From steps 3 to 8, the process continues by increasing 500 instances where classifiers are constructed on the training data (D_{train}) and then tested on the test data (D_{test}). Afterwards, error-rates computed in step 6 and stored it in S_i in step 7. From the *prop* datasets, we included 25 versions in our experiments as described in the Table 3.1. We took the initial version to conduct the first trial and added the next version for subsequent runs. This procedure was iterated until the versions of this datasets have been used. We monitored the significant changes in error-rates to detect changes in the distributions. For computing the error-rate, like *jm1*, the classifiers are construed using the 20% of the training data and rest as the testing data. In Algorithm 1, the procedure continues for computing the error-rate from the step 10 through to 17 where classifiers are generated and computed the error-rate (E_b). The error-rates are stored in S_j for the further investigation. Algorithm 2 presents the process to evaluate the statistical significance using the statistical test presented in equation 3.3. This algorithm inputs the error-rates, (S_i, S_j) computed by executing the classifiers Naive Bayes and Decision Tree. From the algorithm 2, lines 1 and 2 calculate the

p-values and store in i . Line 3 initializes the warning threshold for detecting the significance value as 0.05. From the line 4 through to 9, the significance values ($\hat{i}$) are identified by comparing with the warning threshold (α) 0.05 used as warning level by the authors of the drift detection methods [43, 113].

Algorithm 1 Outline of the Experiment

Input: The historical data, OD_n, VD_n

Output: drift point, d^*

```

1:  $Data \leftarrow OD_n; Data_v \leftarrow VD_n,$ 
2: Initialize  $r = [20]; C = [NB, DT]$  repeat where  $r =$  is the split ratio of the
   data into 20:80 ratio as training and test set
3: for  $i = 500; i = i + 500;$  while  $i \leq Data$  do
4:    $exData \leftarrow Data[1 : i]$  // Adding 500 instances in each trial
5:   Generate: Training sets,  $D_{train}$  and test sets,  $D_{test}$  from  $exData$  by fol-
     lowing  $r$ 
6:   Compute: error-rate,  $E_a$  for the classifier  $C_i$  on  $D_{test}$  data after the train
     the data by the classifier,  $C_i$ 
7:    $S_i \leftarrow E_a$ 
8: end for
9: until  $OD_n$  is empty
10: if ( $Data_v$ ) then
11:   for each item  $v$  in  $Data_v$  do
12:     Generate: training sets,  $D_{train}$  and test sets,  $D_{test}$  from  $exData$  by fol-
       lowing split ratio,  $r$ 
13:     Compute: error-rate,  $E_b$  for the classifier  $C_i$  on  $D_{test}$  data after the
       train the data by the classifier,  $C_i$ 
14:      $S_j \leftarrow E_b$ 
15:      $v \leftarrow$  adding the next version of  $Data_v$  for each trial
16:   end for
17: end if

```

3.4 Experimental Results

In this section, we describe the empirical results for finding the drift points and its type in chronological datasets using the drift detection method discussed in Section 3.3.

We present the prediction performance in terms of error-rate and its statistical significance to identify the drift points in our considered software defect datasets. Considering the 25 versions of the *prop* datasets, Table 3.3 presents

3.4. Experimental Results

Algorithm 2 Significant drift point identification by using the equation 3.3

Input: The obtained error-rate, $[S_i, S_j]$ of the classifiers, $C = [NB, DT]$

Output: significant drift point, $\hat{i}$

- 1: Calculate: p-value by the equation 3.3
 - 2: Store the calculated p-value in i
 - 3: Initialize warning threshold, $\alpha = 0.05$
 - 4: **if** $i \geq 0.05$ **then**
 - 5: Return True
 - 6: Significant value, $\hat{i} \leftarrow i$; // H_1 : Alternative Hypothesis is accepted
 - 7: **else**
 - 8: Return False; // H_0 : Null Hypothesis is accepted
 - 9: **end if**
-

the error-rates and its corresponding p-values. We demonstrated the error-rates of the overall distributions of the trials to monitor the changes in the next trial. Here, the bold highlighted values are the significant values of its corresponding observed versions.

To illustrate the results, we discuss the first drifting point introduced by using the Naive Bayes classifier when the overall distributions or instances (i.e., called as reference window) are 29,109. In this trial, the instances of previous versions are added. After adding the 2998 instances of the target version (i.e., 128), the number of instances becomes 29,109. We monitored the changes between the overall instances of the previous trial's instances 25,582 with the subsequent trial's instances 29,109. The error-rates of these two trials are 0.222 and 0.233. To statically evaluate, the obtained p-value of the trial (where the overall instances are 29,109) is 0.046. It reaches the below of the warning level (i.e., 0.05) for drift point identified as used by [43]. It is also less than the significant value meaning that *concept drift* is detected i.e., the alternative hypothesis is accepted and null hypothesis is rejected. By using the Naive Bayes classifier, we observed the significant fluctuation of the error-rates at the data points, 29109, 41485, 45526, 55775, and 61041. The corresponding versions of these data points are 121, 185, 225, 292, and 318 where

the error-rates are 0.233, 0.241, 0.238, and 0.234 respectively noticed maximum changes of error-rate compared to previous error-rates (Table 3.3). The p-values of these data points are 0.046, 0.032, 0.044, 0.045, and 0.046 which are also less than the significant level indicating the warning level (i.e., 0.05 [43]) of drift point. Figure 3.1(A) portraits the five (5) drifting points and its performance curve for Naive Bayes classifier of *prop* datasets. From the drift identification literature [43, 103], we noticed that the drift point is indicated by a vertical line. Gama et al. [43] also indicated the changes of concept by vertical lines. In Figure 3.1, the drifting points are also identified by vertical lines.

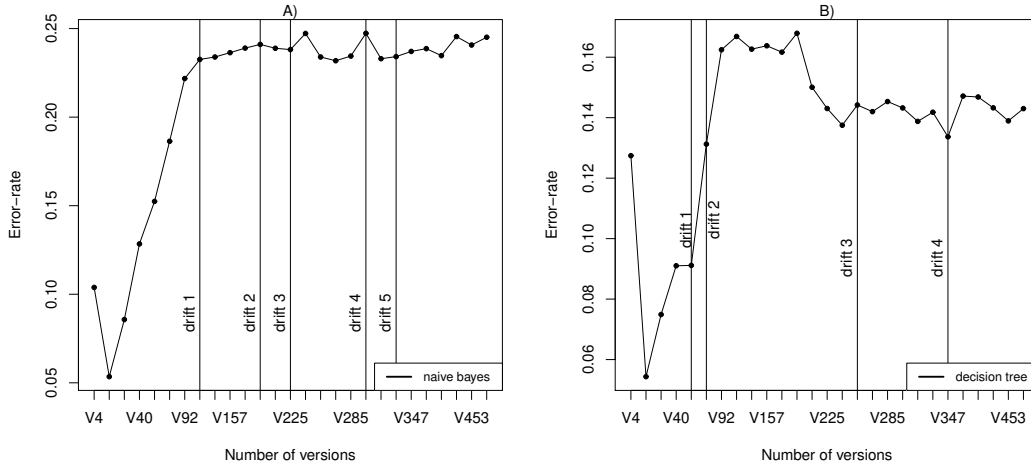


FIGURE 3.1: Performance curve across the twenty-five (25) versions of the *prop* datasets. 5(A) and 5(B) portrait the error-rate fluctuation in different versions. By using the Naive Bayes and Decision Tree classifier, 5(A) and 5(B) identify five and four drifts respectively that marked by vertical lines.

By using the decision tree classifier, we observed the significant fluctuation of the error-rates at the data points 11330, 19027, 49452, 63832 that change the concepts of data distributions. The corresponding observed versions of these data distributions are 9, 44, 245, and 347. The statistically significant values are 0.038, 0.037, 0.043, and 0.037 (Table 3.3) which satisfy the alternative hypothesis and identified four drifting point (Figure 3.1(B)) by

3.4. Experimental Results

TABLE 3.3: Performance observation to detect the *concept drift* in terms of error-rate and p-value across twenty-five (25) versions of the *prop* datasets. The significant changes of error-rate indicate the changes of concept. These error-rates were calculated by using the Naive Bayes and Decision Tree classifier. The significant p-values are highlighted in bold.

# of Overall Instances (ODo)	# of New (Recent) Instances (RDr)	prop Dataset				
		Observed Version	Naive Bayes		Decision Tree	
Historical Data	New Data		P-value (β)	Error-rate (γ)	P-value (β)	Error-rate (γ)
3022	-	-	-	0.104	-	0.127
7277	4255	4	0.202	0.053	0.302	0.054
11330	4053	9	0.658	0.086	0.038*	0.075
15950	4620	40	0.132	0.128	0.264	0.091
19027	3077	44	0.106	0.152	0.037*	0.091
22584	3557	85	0.149	0.186	0.045	0.131
25582	2998	92	0.14	0.222	0.331	0.162
29109	3527	121	0.046*	0.233	0.088	0.167
31605	2496	128	0.135	0.234	0.155	0.163
35062	3457	157	0.054	0.236	0.164	0.164
37887	2825	164	0.057	0.239	0.187	0.162
41485	3598	185	0.032*	0.241	0.052	0.168
43295	1810	192	0.064	0.239	0.225	0.15
45526	2231	225	0.044*	0.238	0.093	0.143
47488	1962	236	0.051	0.247	0.119	0.138
49452	1964	245	0.718	0.234	0.043*	0.144
51146	1694	256	0.095	0.232	0.655	0.142
53431	2285	285	0.059	0.234	0.399	0.145
55775	2344	292	0.045*	0.247	0.081	0.143
58170	2395	305	0.168	0.233	0.273	0.139
61041	2871	318	0.046*	0.234	0.119	0.142
63832	2791	347	0.186	0.237	0.037*	0.134
66686	2854	355	0.056	0.239	0.197	0.147
66942	256	362	0.326	0.235	0.689	0.147
67134	192	452	0.209	0.245	0.183	0.143
67346	-	453	0.134	0.241	0.303	0.139

* Statistical Significance: $*p < 0.05$ and reached the drift level. According to Nishida [135], if the p-value is less than 0.05, it identifies the changes of concept in data distributions called as *concept drift*.

using Decision Tree classifier. From Figure 3.1, we also observed that the error-rate gradually changed and replaced the old concept by the new concept. By definition (Figure 3.1), this type of changes in the concept is called as *gradual drift* identified in *prop* datasets.

The *jm1* project comprises three chronological datasets. In the section 3.3.5, for each experimental run, the distributions were increased by 500 instances and the error-rate computed. We monitored the error-rate of every

TABLE 3.4: Performance observation to detect the *concept drift* in terms of error-rate and p-value across three (3) versions of the *jm1* datasets. These error-rates were calculated by using the Naive Bayes and Decision Tree classifier. Significant p-values are highlighted in bold.

jm1 Dataset						
# of Overall Data (ODo)	Recent Data (RDr)	Observed Version	Naive Bayes		Decision Tree	
Historical Data	New data		P-value (β)	Error-rate (γ)	P-value (β)	Error-rate (γ)
500	-	-	-	0.225	-	0.3
1000	500	1.0	0.249	0.304	0.747	0.28
1500	500	1.0	0.296	0.242	0.475	0.292
2000	500	1.0	0.717	0.284	0.603	0.268
2500	500	1.2	0.102	0.275	0.078	0.282
3000	500	1.2	0.027*	0.38	0.025*	0.419
3500	500	1.2	0.121	0.035	0.222	0.042
4000	500	1.3	0.041*	0.1	0.041*	0.077
4500	-	1.3	0.313	0.301	0.656	0.278

* Statistical Significance: $*p < 0.05$ and reached the drift level. if the p-value is less than 0.05, it identifies the changes of concept in data distributions called as *concept drift*.

run. From the performance curve of the learners in Figure 3.2, we observed the fluctuations in the error-rates across all runs. Form Table 3.4, the bold highlighted values of error-rate and its corresponding p-value indicate the drifting points. For the Naive Bayes and Decision Tree classifiers, the drifting points are identified at the 3000 and 4000 data points of the version 1.2 and 1.3 of the *jm1* datasets respectively. Considering the p-values of these data points, the null hypothesis is rejected and the alternative hypothesis is accepted implying that *concept drift* is detected. By observing the performance curve in Figure 3.2, we noticed that the concept changes suddenly within a short period called as *sudden drift*.

Concept drift identified in the chronological defect datasets. It significantly affected the performance of prediction models that requires to update the classification model for better prediction performance. Results also showed that the concept gradually detected and replaced by the old concept in the *prop* datasets called as *gradual drift*. In the *jm1* datasets, *sudden drift* occurred by changing the old concept within a short period.

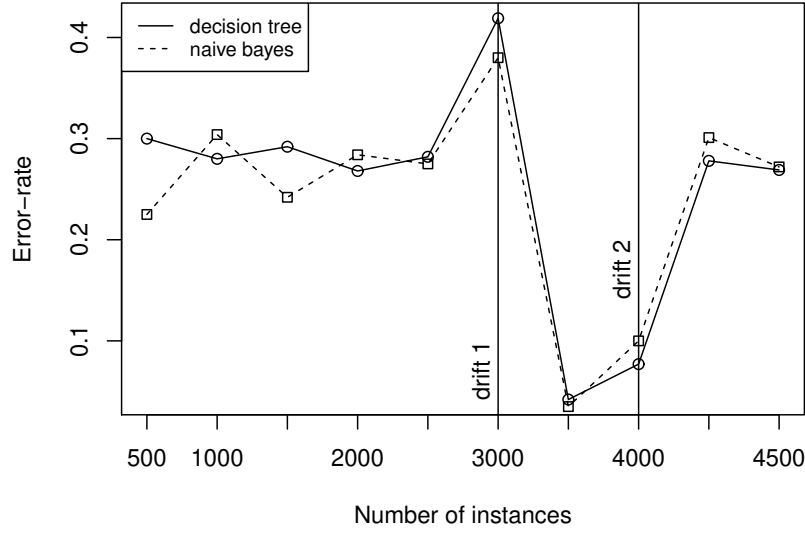


FIGURE 3.2: Error-rate fluctuation for the different training examples for the same model. By using Naive Bayes and Decision Tree classifiers, we identified two drifts in *jm1* dataset by the statistical test where drifts are marked by vertical lines.

3.4.1 Discussion

Implication of Results

Our experiment result shows that the existence of *concept drift* in the defect datasets affects the prediction performance. The models are trained with the same settings to monitor the performance curves. We observe the fluctuation of the error-rates (Figure 3.1 and 3.2) and mark by vertical lines. This indicates that the model needs to be updated after the drift to obtain the better prediction performances. The identification of drift points is helpful for Software Quality Assurance (SQA) Engineer/ teams as drift changes indicate that the prediction model needs to be updated or changed to obtain better prediction performance. It helps to classify the defect and clean modules to test the defect modules. The significant p-value for the corresponding data points identified the chronological datasets and prioritized the versions of those datasets that help the SQA teams to retrain the model again to get the

better prediction performance.

We agree with the previous findings [35, 36] that *concept drift* exists in software defect datasets. Our experiments clearly show that the presence of *concept drifts* in the C&K metrics-based defect datasets impact the quality of prediction. The comparisons between the error-rates with the different versions of *prop* datasets are demonstrated in Table 3.3. Analyzing the drift data points for Naive Bayes and Decision Tree classifier, we identify concept drifts when the versions 9, 44, 121, 185, 225, 245, 292, 318, and 347 introduced into the *prop* chronological datasets thus suggesting the model be updated after the drift to produce better prediction performance.

The concept gradually changes in these versions of *prop* datasets. This type of changes is called as *gradual drift*. In the *jm1* datasets, the sudden drift occurs by changing the concept within a very short period. We suggest considering *concept drift* in the defect prediction studies for better classification.

Threats to Validity

As a preliminary study, we investigate to identify the presence of *concept drift* in software defect datasets. We discuss the possible threats to validity to our empirical results.

In this experiment, we used two chronological software defect datasets, *jm1* and *prop*. These datasets are widely studied [2, 95, 131, 132] in software defect predictions. These studied datasets are however open source projects and the use of commercial projects is left for future studies. Furthermore, empirical outcomes may not be optimistically generalized beyond these studied chronological datasets.

Regarding the statistical test, chi-square test with Yates' continuity correction is adopted to find the statistical significance results. This statistical test

is widely recommended in the *concept drift* detection literature [43]. In addition, we comparatively used large datasets, *jm1* and *prop*, since the statistical test is favored for large sample [43, 113].

In this study, we considered two main learning algorithms namely Naive Bayes and Decision Tree. These learning algorithms have been widely used in *concept drift* detection [43, 135, 136] studies. We choose these learners because they have been used widely in literature and increased the prediction performances as reported by Gama et al. [43].

3.5 Chapter Summary

Previous studies have revealed that *concept drift* may occur in chronological defect datasets. However, the studies did not adopt and apply any drift detection method to identify *concept drift* in those datasets.

This chapter adopts a well-known and widely accepted drift detection method, DDM and investigate the existence of *concept drift* in the NASA and Jureczko datasets.

3.6 Concluding Remarks

From the NASA project, our experimental analyses identified two drifts in the version 1.2 and 1.3 of *jm1* datasets. By monitoring the error-rate of the learning algorithms, *sudden drift* occurred in these datasets indicating that to obtain better prediction performance, the old learner needs to be updated or in some cases replaced by a new one.

From the Jureczko datasets, we considered 25 versions of the *prop* datasets containing 67,346 modules. Among the versions, drifts were identified in the versions of 9, 44, 121, 185, 225, 245, 292, 318, and 347. The drifts occurred gradually in these versions referred to as *gradual drift*.

This experimental study could be enhanced in several ways. As future work, we aim to investigate the *concept drift* by considering the class imbalance issue in defect datasets and how class imbalance and *concept drift* impacts the prediction performance. Additionally, we intend to examine the small software defect datasets with Fisher's Exact test to monitor the data distributions to find the changes of concept that affect prediction performance.

Chapter 4

A Drift Propensity Detection Technique for Improving Cross-Version Software Defect Prediction Performance

4.1 Introduction

Software defect prediction (SDP) is nowadays pervasive in both Software Engineering (SE) academia and industry [15]. The SDP models classify the modules that are likely to be faulty [31]. Efficient identification of defective modules help to allocate the testing resources [121]. It is inevitable to identify the faulty modules for producing a quality software product [45, 67].

In the past decade, several prediction models including machine learning and statistical approaches have been proposed. Traditional SDP models utilize the historical data to predict defects of the unlabeled data [98]. For Within-Project Defect Prediction (WPDP), prediction models are trained on labeled data and tested on the unlabeled data within the same project. In WPDP, cross-validation techniques are used to partition the training and

test datasets within the same project version, and, moreover, is called Inner-Version Defect Prediction (IVDP) [4]. However, the development of software projects are becoming more sophisticated and the development process experiences frequent changes over time [47]. For the projects with multiple versions, the prediction model is more appropriate if the training data is obtained from the previous version and conducts the prediction of the upcoming version. This is called Cross-Version Defect Prediction (CVDP).

4.1.1 Motivation

The software development environment is an evolving process where a mature project undergoes multiple changes by incorporating new requirements when the version is updated [47]. During this process, the updated version becomes more complicated due to refactoring and module deletion from the previous version. This causes distribution differences among the versions of the software project [4, 181]. The studies of [4, 181, 183] argued that the data distribution differences among the versions make the prediction results misleading. Dong et al. [31] affirmed that the prediction results of a well-trained prediction model could be inaccurate if the historical data differs over time. However, in the domain of SE prediction research, not many researchers have investigated it. In the CVDP setting, it is recommended by the studies of [4, 181, 183] to consider the previous version of the contemplated project as a training dataset for getting a better prediction since the prior version yields the related distribution information [156]. These studies noticed a certain degree of differences in the data distributions, which impacts the prediction performance.

Recent studies of Xu et al. [179, 181, 183] considered the issue of data distribution differences of the versions for CVDP. The proposed framework of Xu et al. [179] adapted an active learning method for modules selection

from the current version and merged the selected modules with prior versions to improve the information content of the training set. By doing such, they tried to alleviate the data distribution differences of two consecutive versions for predicting defects of current version. Later, Xu et al. [183] utilized an optimization technique for suitable modules selection and evaluated the performance of CVDP on 67 versions. Furthermore, in the study of [181], Xu et al. preferred to take the advantage of subset selection methods based on dissimilarity and tried to mitigate the differences in data distributions. The subset selection method picked the effective modules from the previous version and applied it to classify the defective modules of the current versions. The recent two studies (i.e., [179, 181]) of Xu et al. considered either the previous version or current version for subset selection using the selection methods. In both ways, useful information could be removed during the reduction process, as claimed by Xu et al. [183].

However, no studies considered that data comes in streaming fashion, taking all the versions into account in the context of CVDP. According to streaming data analytic, the learning process becomes more complicated due to the non-stationary environment leading to changes in data distributions [113]. The changes in data distribution are referred to as concept drift in the field of data streaming mining [42]. Ekanayake et al. [35, 36] investigated the phenomenon of concept drift and conducted an empirical study (i.e., [35]) to assess the reason why the prediction accuracy fluctuates in SDP. However, the studies of [35, 36] mainly focused on finding the location of significant changes in data distribution and not comprehensive in the context of CVDP. To the best of our knowledge, this is the first attempt to consider the data that comes in streaming fashion. We refer to this situation as Drift Propensity (DP): significant changes in data distribution among the versions and if the DP is not addressed in the data distribution, it results in poor prediction performance. Identification of DP is essential to take necessary steps for

improving prediction performance in the cross-version scenario. Since the differences of data across the versions have adverse influences on the prediction performance of CVDP.

In this work, we address the concern as DP and propose a DP detection technique that identify the DP's in the chronological defect datasets. By recognizing DP's in the datasets, practitioners could take necessary steps to enhanced the prediction results. We conduct an empirical study on 26 versions of six projects from the SEACRAFT repository ¹ to evaluate the effectiveness of our proposed DP detection method.

4.1.2 Contributions

We highlight the main contributions as follows:

1. We propose a DP detection method potentially identified the DP's where the prediction accuracy varies in the versions during the learning process. We trigger three flags (i.e., DP, warning, and control flag) to take necessary steps during the DP detection process.
2. We provide analysis by performing experiments on 26 versions of six benchmark datasets. To our best knowledge, this is the first effort to identify DP by conducting experiments in the cross-versions defect datasets.
3. We employ fisher's based statistical test to identify the significant DP values during the detection process. Furthermore, we use the scheme of two-window based data management technique, no-memory, to manage the versions and to create the scenario of cross-version.
4. The effect size computation for the pair-wise comparison, Hedges' g , ensures that the identified DP's are meaningful and practically significant. Based on the results, we observe that an exponential increase of the error-rate addresses the DP's in the chronological defect datasets.

¹<https://zenodo.org/communities/seacraft>

We inspire the use of the DP detection technique for the identification of DP in the chronological defect datasets for CVDP. The DP's are critical to classification problems due to inequality in the decision boundary, thereby influences the model performance. By identifying DP, necessary steps can be taken to improve the prediction performance. The exponential increases of error-rate cause DP in the versions of software defect datasets and consequently exhibit poor prediction performance.

4.1.3 Chapter Organization

The rest of this work is structured as follows. Section 4.2 describes the related work on CVDP. Section 4.3 presents the definition of DP. The two-window based DP detection process is described in section 4.4. We introduce fisher's based statistical significant DP detection process in the section 4.5. In section 4.6, we determine the configuration settings of our experiment. We outline the experimental results in section 4.7. Section 4.8 points the potential threats and section 4.9 concludes this work by outlining future direction.

4.2 Related Work

CVDP has drawn high interest because of its applicability to the real development scenario where the previous version is used to predict defects in the current version of the same project, and distribution differences among the versions influence the prediction performance. However, not many researchers have considered it.

Yang and Wen [184] examined the ridge and lasso regression model for investigating the performance of CVDP. By employing 11 projects with 41 versions, they tried to find the strong correlations among the predictors and found the existence of multicollinearity in the considered datasets. Their

work focused on ranking the defects. They avoided the task of feature selection and specified that it loses the information during the selection process. However, they didn't focus on the issue of data distributions differences, which affects prediction performance.

Shukla et al. [156] considered CVDP in their experiment and explained it as the problem of multi-objective optimization with two objectives: maximizing recall with minimizing the cost of misclassification and quality assurance. They studied 11 projects containing 41 versions and revealed that the multi-objective regression outperformed the single-objective regression. This study didn't focus on the distribution differences among the versions in CVDP, and it's applicability of building prediction models in cross-version settings.

Bennin et al. [11] performed cross-release performance evaluation of 25 software projects comprised of only two releases by considering $\text{Norm}(P_{opt})$ effort-aware performance measure where M5 and k^* outperformed 9 other prediction models. They observed the strong correlation between the accuracies for intra-version and cross-version. However, the conducted 11 models were not statistically significant. This study also didn't consider the distribution differences among the releases that make the prediction accuracies outdated.

Lu et al. [110] considered the data distribution differences and performed feature selection task by using active learning method for CVDP and tried to alleviate distribution differences. However, the selected modules were unable to act as representative modules of the recent version as claimed by Huang et al. [70].

Xu et al. [179] developed a framework by combining principal component analysis and hybrid active learning for selecting informative and representative modules from the current versions and tried to alleviate the distribution differences. By experimenting on ten projects comprising of 31 releases, they attempted to improve the prediction performance and further improved the

work of Lu et al. [110]. In this work, the selected modules were added to previous versions, which required extra effort in labeling the modules.

Later, Xu et al. [181] proposed a subset selection technique based on dissimilarity for CVDP. They selected the representative modules from the previous version and combined them with current versions. In the work of [179], they chose the modules from the current version and added them to the prior release by using a hybrid active learning approach. Similar to the work in [181], they selected the modules from the previous version and added them to the current version. In both ways, they tried to alleviate the distribution differences by combining the selected modules to previous and current version accordingly. However, it requires extra effort to label the selected modules in the combining procedure.

The studies of [110, 156, 184] developed the prediction model for cross-version projects by using all the modules of the previous version without taking the issue of distribution differences into account. Besides, the studies in [179, 181, 183] used a feature selection method to alleviate the distribution differences. For combining the selected modules to the considered versions for training the model, it requires extra effort during the adding process. None of these studies considered that the data across the versions come in streaming fashion, where data drifting within the versions can influence the prediction performance in cross-version settings.

4.3 Drift Propensity

This section provides a formal definition of Drift Propensity (DP), which may manifest in various ways in the data distributions.

DP describes unforeseeable variations in the data distributions by changing the statistical properties over time. The changes might occur for the hidden changes of variables. Formally, DP can be defined as follows: For a classification problem, given in a time period $(0, t)$, with a set of data samples $D_{(0,t)} = \{d_0, d_1, \dots, d_t\}$ in which each instance d contains (X, y) where X is feature with the class label $(y \in Y)$. DP occurs at the time t , if the distribution of $D_{(0,t)}$ is significantly different from the $D_{(t+1),\infty}(X, y)$ as simply defined as $D_{(0,t)}(X, y) \neq D_{(t+1),\infty}(X, y)$. DP is critical for classification problem. Particularly, in CVDP, the data distribution comes over time and trained data is derived from the previous version of the defect datasets where current version is considered as test data. DP creates such differences that may lead to inequality in the decision boundaries, thereby decreasing learning accuracy.

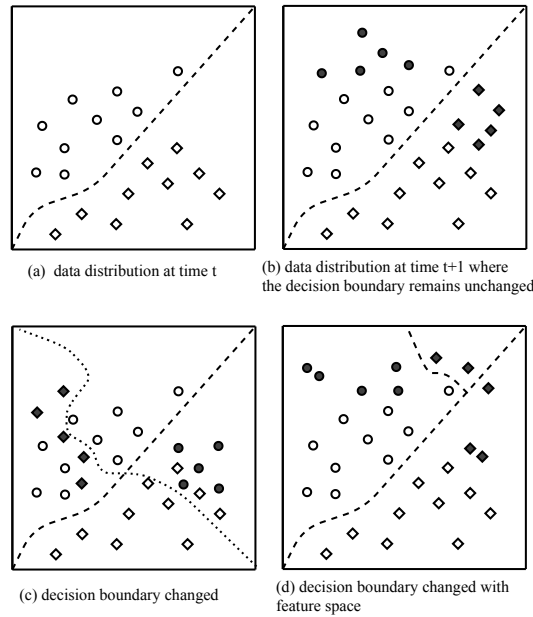


FIGURE 4.1: DP detection in target feature space by changing the decision boundary. White circle and square are the two class labels at time t where black circle and square of class labels at time $(t + 1)$.

DP in the distributions can manifest in multiple forms by changing the probability F of X and y at time t . Figure 4.1 portraits how DP occurs in a feature space. It may occur in the following ways (Figure 4.1):

1. **Feature Space DP:** The decision boundary of Figure 4.1a and Figure 4.1b remain unchanged though $F_t(X) \neq F_{t+1}(X)$. In this case, $F_t(X)$ and $F_{t+1}(y|X)$ are same.
2. **Decision Boundary DP:** In Figure 4.1c, DP causes by changing the decision boundary where $F_t(X) \neq F_{t+1}(y|X)$. In this case, $F_t(X)$ and $F_{t+1}(X)$ are same.
3. **Feature Space with Decision Boundary DP:** In Figure 4.1d, DP occurs for the both cases, changing the feature space with decision boundary where $F_t(X) \neq F_{t+1}(X)$ and $F_t(X) \neq F_{t+1}(y|X)$.

4.4 DP Detector

In this work, we propose a DP detection technique to compare the two consecutive data distributions. As a base learner, we use Naive Bayes (NB) in our experimental study. For detecting DP in the chronological software defect datasets, we assume that the data comes in streaming fashion. In our proposed DP technique, we assume the two consecutive versions of a software project as two-window based data distributions that arrive over time. In the CVDP, the prior version of a software project is considered as historical data, and the model is built on that data to predict defects for the next release or version of the software project. We employ the scheme proposed by Drift Detection Method (DDM) [43] to identify DP in the chronological defect datasets in the CVDP settings. The idea behind the method is to monitor the algorithm's error-rate. If the difference between two adjacent versions of defect datasets is significant, it indicates DP that exists in the examined versions. The remedy is to retrain or enhance the model again to get better performance.

In this method, we assume that the data comes over time (i.e., in chunk fashion). We observe the error-rate (r) from the incoming n data samples that contains the number of instances i .

Suppose, the instances can be presented as a pair of $(\vec{a}, b)$ where $\vec{a}$ is the values with different attributes and b is the label of class. For the i – th instances, it can be presented as $(\vec{a}_i, b_i)$. When the prediction model is built the class level can be either $(b_i = \hat{b}_i)$ or $(b_i \neq \hat{b}_i)$ where the error-rate is a random variable in Bernoulli trails. The general scheme is adopted from the drift detection method, DDM [43]. In binomial distribution, the probability of random variable illustrates the error of i instances. The error-rate is computed for i instances with standard deviation $d_i = \sqrt{r_i(1 - r_i)/i}$ at time t . The DP detector stores the values of error-rate r_i and standard deviation d_i during the learning process. When the new chunk arrives, the model is built and stored the error-rate with its standard deviation. The binomial distribution is considered as a normal distribution for a large number of data distributions, and consequently, it contains approximately the same variance and mean. The probability distribution should not changed unless the changes of the data distributions happens and then the $1 - \frac{\alpha}{2}$ confidence interval for error-rate er_i with i instances is $(er_i \pm \alpha * sd_i)$ where the parameter α relies on the confidence level.

During the learning process, the DP detector manages two registers for minimum values of error-rate r_{low} and standard deviation d_{low} . From the stored error-rate r_i and standard deviation d_i , the detector compares with r_{low} and d_{low} to the target version of datasets. This comparison helps to identify the data distribution differences of the the adjacent defect datasets. For detecting DP in cross-version settings, we set the confidence level for triggering the warning flag to 95% ($\alpha = 0.05$) with the threshold $r_i + d_i \geq r_{low} + 2d_{low}$. The confidence level for triggering the DP flag has been determined to 99% ($\alpha = 0.01$) with the threshold $r_i + d_i \geq r_{low} + 3d_{low}$ as suggested by Gama et

al. [43].

During the detection process, the DP detector triggers three flags to identify the DP in the data distributions: DP flag, warning flag, and control flag. The flag of DP is triggered, if $r_i + d_i \geq r_{low} + 3d_{low}$. Warning flag is triggered, if $r_i + d_i \geq r_{low} + 2d_{low}$. DP flag indicates that DP is detected, where the necessary actions are imminent to achieve more reliable prediction performance. By triggering the warning flag, it implies that the target distributions will tend to be drift. Otherwise, the distribution is stationary by triggering the control flag. If the DP exits on that version, the current model will be outdated, and it needs to be retrained or updated. Thereby, DP is detected by processing the data that arrives in bundles.

Algorithm 3 Outline of DP Detector

Input:

The old window O_t and recent window R_t

Learning algorithm L

The sample of train and test i

Result:

DP F_{dp} , Warning F_w and Control F_c .

Process:

```

1: for  $t = 1, 2, \dots$  do
2:   Build the learner on  $O_t$ 
3:   Learner predicts on  $R_t$ 
4:   Calculate error-rate  $r_i$ 
5:   Calculate  $d_i = \sqrt{r_i(1 - r_i)/i}$ 
6:   Store  $r_i$ ,  $d_i$  and find the minimum error-rate  $r_{low}$  and standard deviation  $d_{low}$  during the process
7:   if  $(r_i + d_i \geq r_{low} + 3d_{low})$  then
8:      $F_{dp}$  detected // DP flag
9:   else if  $(r_i + d_i \geq r_{low} + 2d_{low})$  then
10:     $F_w$  detected // warning flag
11:   else
12:     $F_c$  // control flag
13:   end if
14: end for

```

Algorithm 1 presents the pseudo-code of DP detector. The model is built on the old window O_t and tested on recent window R_t based data distributions. DP detector detects the differences in data distributions by treating the

data as chunk by chunk. However, error-rate is calculated r_i with standard deviations d_i (lines 2 to 5) of the considered versions of the defect datasets. The minimum values of error-rate r_{low} and its standard deviation d_{low} are kept in register to compare it with the recent values. In this DP detection settings, more than two versions are considered for this learning process and also to make the scenario of cross-version of the software project. The DP detector triggers the flags to identify DP in the data distribution by executing lines 7 to 13.

4.5 Fisher's based Statistical Significant DP Detection

Fisher's Exact test is proposed to analyze contingency table and considered as exact test [39]. This test doesn't lose precision because it is not dependent on approximation and is not influenced by data behavior (e.g., imbalanced data) [100]. This test can be applied to analyze $m \times n$ dimensional tables. It is generally used 2×2 contingency tables, which fit the case of two-window based data distribution.

Assume, we have two distributions which indicate two chunks: recent chunk and old chunk. The confusion matrix for the recent chunk and old chunk is given below (e.g., in Table 4.1 and 4.2).

TABLE 4.1: For recent chunk

	Positive Prediction	Negative Prediction
Actual Positive	q_1	r_1
Actual Negative	s_1	t_1

The primary reason for adopting a statistical test is to ensure that DP is not caused by sampling error and the DP value is statistically significant. Fisher's

TABLE 4.2: For old chunk

	Positive Prediction	Negative Prediction
Actual Positive	q_2	r_2
Actual Negative	s_2	t_2

TABLE 4.3: Adapted generalized contingency table for recent and old chunk

Labels	Recent chunk	Old chunk	Total
# of Errors	s	t	s + t
# of Correct Classes	q	r	q + r
Total	q + s	r + t	n = q + r + s + t

Exact test is implemented for the two-window based data distributions using the 2×2 contingency tables presented in Tables 4.1 and 4.2 and explained below.

Definition 1. Given labeled datasets X_i and X_j are transformed into vector space $V_i = (q_1, r_1, s_1, t_1)$ and $V_j = (q_2, r_2, s_2, t_2)$ of the recent and old chunk, accordingly in Table 4.1 and 4.2. Most classification models are assessed employing a confusion matrix. Two-window based data distributions (i.e., recent chunk and old chunk) where four types of outcomes are achieved as follows: q_1, q_2 are the faulty instances that are correctly classified as faulty; s_1, s_2 are non-faulty instances incorrectly classified as faulty; r_1, r_2 are the faulty instances incorrectly classified as not faulty; t_1, t_2 are non-faulty instances classified as not faulty. Therefore, for the recent chunk, the total number of correct classification $q = q_1 + t_1$ and misclassification $s = s_1 + r_1$. For the old chunk, the total number of correct classification $r = q_2 + t_2$ and misclassification $t = s_2 + r_2$.

In the Eq. 4.1 and 4.2, p denotes as probability matrix of any event, $m = 1 - p$, and $!$ is factorial of the data. From the Table 4.3, the t occurrences in the independent events of $s + t$ can be expressed by $((s + t)! / (s! \times t!)) \times p^s \times m^t$.

Similarly, the events of probability q in $q + r$ is determined by $((q + r)! / (q! \times r!)) \times p^q \times m^r$. According to 2×2 contingency table, the probability of the 2 chunks can be presented by the Table 4.3 and determined by the Eq. 4.1. By using the combined analysis, Yates [186] and de Lima Cabral et al. [100] proved that the Eq. 4.1 could be formed by the Eq. 4.2.

$$p = \frac{(q + r) \times (s + t)!}{q! \times r! \times s! \times t!} \times p^{q+s} \times m^{r+t} \quad (4.1)$$

$$p = \frac{\binom{q+r}{q} \times \binom{s+t}{s}}{\binom{n}{q+s}} = \frac{(q + r)! \times (s + t)! \times (q + s)! \times (r + t)!}{q! \times r! \times s! \times t! \times n!} \quad (4.2)$$

Assuming, the null hypothesis is that the distribution on both windows X_i and X_j is not similar if and only if the observed $p - values$ are less than the significant level. Furthermore, our concern is to check if the chunks are affected by the DP or not. The null hypothesis that the error of both chunks should be rejected if and only if the observed values of p (0.05) are less than at 5% significant level.

We calculate the $p - values$ from the Eq. 4.2. The factorial functions of each item from the data distributions $[0...n]$ where n is the seen examples. As suggested by Lu et al. [113], the size of the window can be determined by the user. For simplicity, without discarding any instance, we consider the full data of the two consecutive versions as old and new chunk of the data distributions. Thus, we implement the Fisher's Exact test for two-window based data distributions to identify the significant DP of the considered datasets.

4.6 Experimental Methodology

In this section, we describe the experimental configuration in detail containing the chronological software defect datasets, and the illustration of the DP

detection process with the data management technique.

4.6.1 Benchmark datasets

In this empirical study, we consider 26 versions of six benchmark datasets extracted from the SEACRAFT Repository which were provided by Jureczko and Madeyski [83, 117]. These datasets have been used in the studies of CVDP [179, 181, 183, 184]. The statistics of the considered datasets are shown in Table 4.4 where #Modules, #DefectiveM, and %DefectiveM denote the number of modules, the number of faulty modules, and the ratio of defect, respectively. This study requires datasets that include several versions of the same projects to prepare the scenario of CVDP. The datasets comprise modules where each module is characterized by the label of defective or non-defective with 20 static code software metrics (Table 4.5).

4.6.2 Evaluation setup

In this section, we describe the evaluation process of our empirical study. Learning from such environment where the data comes over time is very challenging task [135]. In streaming mining, the drift detection techniques are limited, in the sense that the methods are performed over every data instance in the streaming and are threshold dependent [171].

However, this study uses the DP detection process (see Section 4.4) to identify the DP in the chronological datasets. Following the DP detector method, error-rate is computed for each version by building the prediction model where train data is used as previous version and tested the classification model on the new version. Most of the models are evaluated from the confusion matrix and error-rate is computed described in Table 4.6 where $error - rate = 1 - accuracy$. Accuracy is computed as $\frac{(TP+TN)}{(TP+TN+FP+FN)}$ where True Positives (TP) are the defective modules that are correctly classified,

TABLE 4.4: Summary of Selected Benchmark Datasets

Project	Version	#Modules	#DefectiveM	%DefectiveM
ant	1.3	126	20	15.9%
	1.4	178	40	22.5%
	1.5	293	32	10.9%
	1.6	352	92	26.1%
	1.7	745	166	22.3%
camel	1.0	339	13	3.8%
	1.2	608	216	35.5%
	1.4	872	145	16.6%
	1.6	965	188	19.5%
jedit	3.2	272	90	33.1%
	4.0	306	75	24.5%
	4.1	312	79	25.3%
	4.2	367	48	13.1%
	4.3	492	11	2.2%
poi	1.5	237	141	59.5%
	2.0	314	37	11.8%
	2.5	385	248	64.4%
	3.0	442	281	63.6%
xalan	2.4	723	110	15.2%
	2.5	803	387	48.2%
	2.6	885	411	46.4%
	2.7	909	898	98.8%
xerces	init	162	77	47.5%
	1.2	440	71	16.1%
	1.3	453	69	15.2%
	1.4	588	437	74.3%

False Positives (FP) are the non-defective modules that are incorrectly classified, True Negatives (TN) are the non-defective modules as non-defective, and False Negatives (FN) are the defective modules that are incorrectly classified as non-defective, respectively.

Our proposed DP detection approach operates the data based on the two-window based data management technique. We adopt the data processing scheme of no-memory data management technique to handle the data and execute it as two-window (see Figure 4.2). In Figure 4.2, versions of the data

TABLE 4.5: Description of Metrics Used in the Chronological Defect Datasets

Abbreviation	Description
WMC	the sum of weighted methods in a class
DIT	the level of depth of inheritance tree for a class
NOC	the number of direct successor of a class
CBO	the number of coupling classes between objects
RFC	the number of methods responded from where received messages
LCOM	the set of methods not related to share the class fields
Ca	the number of other classes uses the specific class
Ce	the number of other classes are used by specific class
NPM	the number of declared public methods
LCOM3	a variant of LCOM
LOC	the number of line of code
DAM	the number of protected attributes declared
MOA	measuring the part-whole relationship
MFA	measuring the number of methods inherited meaning functional abstraction
CAM	cohesion among methods of class
IC	inheritance coupling
CBM	measuring the number of methods where methods are coupled
AMC	average method complexity
MaxCC	the maximum values of cyclomatic complexity of methods in a class
Avg(CC)	the average number of arithmetic mean of cyclomatic complexity of methods in a Class
BUG	bugs in the class

TABLE 4.6: Confusion matrix setup for two-class problem

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

distributions that shifts over time are observed by modifying the version update. Cross-version scenario is designed by considering the two adjacent versions as portrayed in Figure 4.2. As an example, the model is trained by employing poi 1.5 and tested on poi 2.0. Our evaluation process adopts the

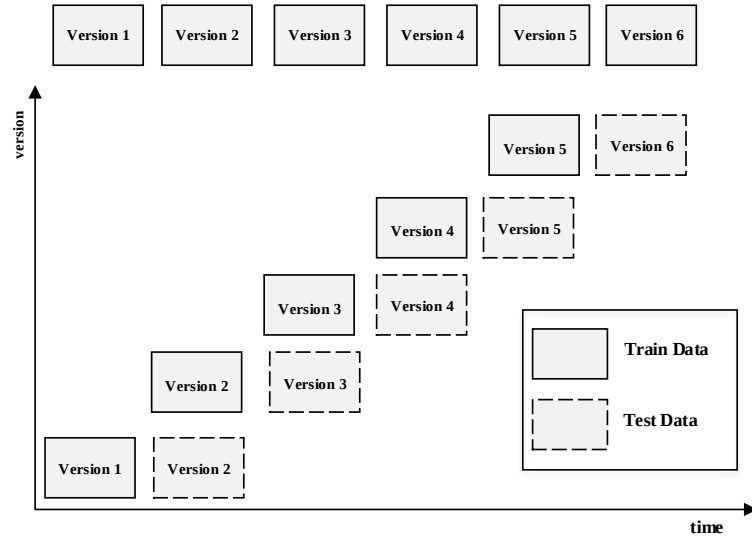


FIGURE 4.2: Adopted No-memory data management approach to manage the versions of the chronological defect datasets.

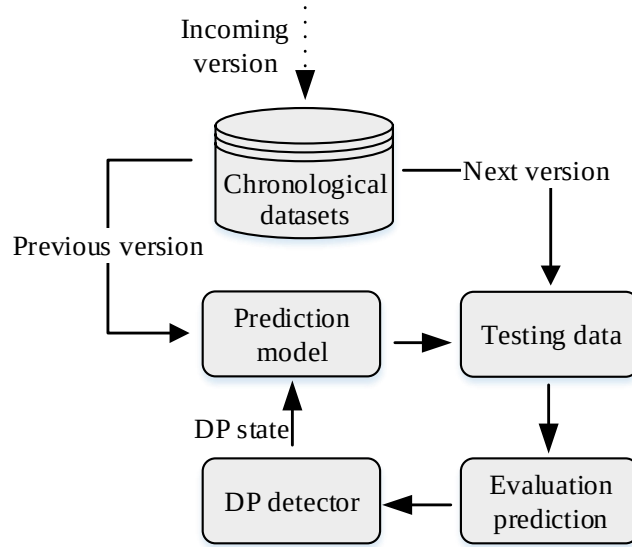


FIGURE 4.3: Evaluation process

single base learner Naive Bayes (NB) to compute the error-rate. The evaluation process is illustrated in Figure 4.3. In the learning process, error-rate is monitored by triggering the flags as proposed in Algorithm 1. As mentioned above, the proposed DP detector triggers three flags values. By triggering

the DP flag, it implies that DP is detected in the examined version, where the necessary steps (e.g., updating or retraining the model) are imminent to obtain more reliable prediction performance. By triggering the warning flag, it suggests that the target distributions will tend to be drift. The control flag determines the stability of the environment (i.e., the distributions do not change over time). By executing fisher's based statistical significant test (Section 4.5), we ensure that the DP's are not caused by sampling error, but they are statistically significant as well.

4.6.3 Effect size computation for two-sample case

Effect size (ES) is considered as a quantitative measure of magnitude that has been applied in many studies of behavioural science [177]. Carl et al. [20] recommended the use ES in empirical studies to obtain practical significant, and meaningful results.

Moreover, the conclusion extracted from hypothesis testing might be imprecise because of the sufficient number of experimental subjects and the $p - values$ may also become inaccurate. Thus, weak inferences are not evaluated as part of the experimental settings [88]. Therefore, it is also recommended by [33, 88, 89, 125] to consider effect sizes in the experimental study of SE along with significant statistical test for getting practical and meaningful results. Kampenes and Sjøberg [88] mentioned that 1% increase in defect detection would be highly practical significance for critical software if the defects belong to more critical defects.

As a robust ES computation standardized method, Hedges'g [63, 170] is considered to affirm the practical significance of our empirical study. This test reveals the significant differences between two consecutive data distributions and indicates meaningful and practical significance. Furthermore, it is an unbiased method and helps to find the identified differences that are

significant for the small size of data distributions [20]. It is calculated by employing the means and the sample based standard deviations of the two data groups where the ES component is indicated as Hedges's g [125]. Suppose two data distributions are considered as two sequential versions of the chronological defect datasets. These two sequential data distributions are treated as control and experimental condition where the ES is defined by the Eq. 4.3.

$$Hedges'g = \frac{X_{ov} - X_{rv}}{S_{pooled}} \quad (4.3)$$

Eq. 4.3 implies the ratio of the two means (X_{ov} and X_{rv}) of the sequential defect datasets considered as old and recent version in the control and experimental condition. The difference between the two means is divided by the pooled standard deviation S_{pooled} . Since the versions of the defect datasets are dissimilar in size from each other, we calculate the pooled standard deviation by the Eq. 4.4.

$$S_{pooled} = \sqrt{\frac{(n_{ov} - 1) * s_{ov}^2 + (n_{rv} - 1) * s_{rv}^2}{n_{ov} + n_{rv} - 2}} \quad (4.4)$$

In Eq. 4.4, n_{ov} , s_{ov} , n_{rv} , and s_{rv} are the number of subjects, measurement of standard deviation of old version, the number of subjects, and measurement of standard deviation of recent version, respectively. We consider the absolute values of Hedges's g that are interpreted as follows: the values of Hedges's $g < 0.20$ indicate as the small effect, the values of about 0.50 interpret as moderate effect, and the values of about 0.80 explain as large effects using Cohen's benchmarks [25] and also adopted in the studies of [44, 48].

4.7 Experimental results and discussion

In this section, we discuss the results of this experimental study. The results are presented based on the proposed DP detection technique and identify

statistically significant values of DP in the chronological defect datasets using Hedges'g effect size and fisher's based statistical test at 5% significant level.

TABLE 4.7: Performance observation of the DP detection technique in the chronological defect datasets. Two-window based Fisher's Exact test results and effect sizes of the pairwise comparison for two consecutive data distributions. Large effect sizes point to a practical significance in favor of DP detection.

Project	OVersion	#InsOWin	RVersion	#InsRWin	DP value	p-value	Pooled effect size Hedges'g
ant	1.3	125	1.4	178	4.91	2.62E-03*	1.526**
	1.4	178	1.5	293	15.10	8.69E-10*	0.856**
	1.5	293	1.6	351	2.40	0.157	0.046
	1.6	351	1.7	745	1.02	0.093	0.092
camel	1.0	339	1.2	608	9.72	6.29E-07*	0.834**
	1.2	608	1.4	872	5.04	6.16E-05*	0.805**
	1.4	872	1.6	965	1.02	0.0864	0.085
jedit	3.2	272	4.0	306	10.20	0.032*	1.902**
	4.0	306	4.1	312	2.42	0.101	0.019
	4.1	312	4.2	367	2.19	0.059	0.317
	4.2	367	4.3	492	0.97	0.061	0.438
poi	1.5	237	2.0	314	1.02	0.285	0.178
	2.0	314	2.5	385	15.85	1.53E-03*	1.262**
	2.5	385	3.0	442	1.51	0.114	0.017
xalan	2.4	723	2.5	803	4.07	1.77E-08*	0.850**
	2.5	803	2.6	885	11.07	1.14E-03*	1.457**
	2.6	885	2.7	909	2.34	0.798	0.035
xerces	init	162	1.2	440	4.45	0.004*	0.868**
	1.2	602	1.3	453	1.00	0.055	0.025
	1.3	1055	1.4	588	27.34	1.07E-05*	1.456**

Old version (OVersion); Recent version (RVersion); number of instances in old window or version (#InsOWin); number of instances in recent window or version (#InsRWin); Drift Propensity value (DP value). "*" donates the values of 5% statistical significance p . "***" denotes the practical significance at $Hedges'g \geq 0.80$ calculated for large effect size and unequal data distributions.

In this experimental study, we consider defect datasets which had at least four versions to establish a solid scenario of cross-version and to perform more generalizable results. The aim of this empirical study is to detect the distribution differences among the versions that are responsible for DP in the defect datasets. Mentioned above, the versions of the defect datasets are managed by the no-memory data management approach (Figure 4.2) and to establish the scenario of CVDP. The predictive error-rates are computed to

4.7. Experimental results and discussion

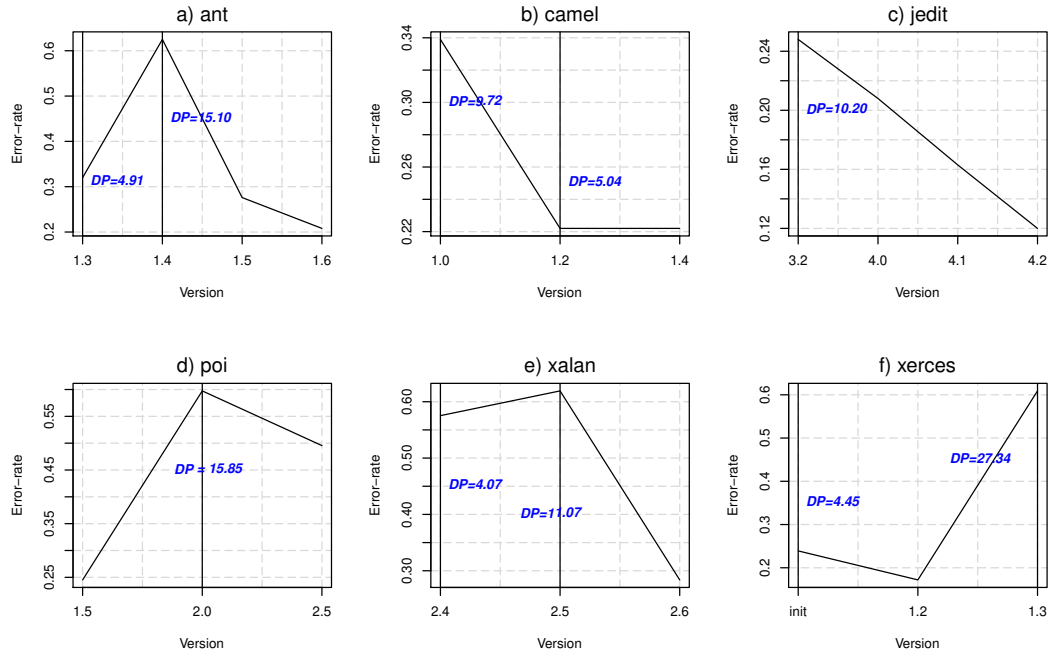


FIGURE 4.4: Performance curve and portrait the error-rate fluctuation of the versions of the considered datasets, using NB. Significant DP's are marked by vertical lines of the chronological datasets.

identity the DP in the datasets based on the proposed DP detector. Please note that the experiment is conducted very carefully specifically in selecting the data distributions of the versions and group comparisons to eliminate any randomness in the experimental settings. In this case, the no-memory data management approach plays a vital role in our study.

The datasets contain several versions considered as chronological where the versions are created over time (Table 4.4). Experimental results are illustrated by adopting a two-window based approach. In Table 4.7, we observe the error-rate attained using the two consecutive versions where IVersion, #InsOWin, and #InsRWin refer the initial version, number of instances of old version, and recent version, respectively. The values of DP obtained from the experiment are illustrated in the Table 4.7. To summarize the results and for easy understanding, we use "*" to denote the significant values of the DP's

in the associated versions. Further, we use "***" to indicate the practical significance using pair-wise comparison with effect size test across the versions of the defect datasets. The effect size Hedges' g test is considered along with significant statistical analysis to ensure practical and meaningful results, and to avoid misleading p – values. Since the experiment includes fewer subjects in the settings, it is recommended by [125] to consider ES test as part of the SE empirical study. The boldface values represent the best results gained from our proposed algorithm and illustrate the identification of DP's in the associated versions of the project. The DP flags are triggered on those versions according to our detection process. Results from the DP detection process by pair-wise comparison among the two adjacent data distributions, we identify DP in the ant 1.3 and 1.4 versions among the seven versions, which achieve the best significant values (p – value $\ll 0.001$). These versions gain large effect size (i.e., ≥ 0.80). In this case, the most guided practice is to update or retrain the model if DP is identified in the data distributions to get more reliable prediction performance. In addition, the flags of DP are triggered in the versions of 1.0 and 1.2 for camel, 3.2 for jedit, 2.0 for poi, 2.4 and 2.5 for xalan, and init and 1.3 for xerces, which are statistically significant and achieve the more substantial effect in pair-wise comparisons.

A depiction of the performance curve of the considered datasets is presented in Figure 4.4. The DP's are marked by vertical lines where the flags of DP's are triggered. It indicates to update the model when DP is identified in a particular version to get better prediction performance for the target data distributions. The error-rate fluctuation can be observed in this figure expressing an observation; the DP's are identified when the error-rate of the learning algorithm exponentially increased.

The proposed DP detection technique identifies the number of DP's and their positions across the versions of the defect datasets. It is worth mentioning that the adaptation of fisher's exact test is computationally costly because

of astute inability of factorial calculation, as also identified as limitation in [100]. Fisher's based statistical test for significant DP detection itself, a worthy contribution, can be utilized for similar types of problems. In addition to this, it can be applied in two-window based data distributions in offline cases as well. Apart from this, the effect size computation for the pair-wise comparison, Hedges' g , ensures that the identified DP's are meaningful and practically significant.

One can speculate how DP can impede the prediction performance for the projects of cross-versions. In order way, how identification of DP can help to improve the prediction performance for CVDP. To illustrate, please consider the following scenario. The poi project contains four versions. If the model is built using the data of poi 2.0 as historical data to predict defects for consecutive versions, the prediction model won't be able to detect defects accurately. Again, The xerces project contains four versions. As the last model, if the model is built using the data of xerces 1.3 as historical data to predict defects for consecutive versions, the prediction model won't be able to detect defects accurately. In these case, the software developers or practitioners could know which models are DP-prone or not. Based on the results of our empirical study, we observe that the model needs to be updated for predicting defects for the version of 2.5 for poi and xerces 1.4 because the difference of the data distributions is statistically significant, causing DP in that version and resulting in the deterioration of model performance. Therefore, we suggest addressing DP in the versions of the defect datasets for defect prediction in cross-version settings.

4.8 Threats to validity

In the empirical study, it is necessary to be aware of threats to validity of the obtained results [8]. Basili et al. [9] argued that it is challenging to make a

general conclusion from an experiment in SE because the experimental process requires a large number of relevant context variables.

We consider 26 cross-versions defect datasets from six benchmarks projects. To make the cross-versions scenario, we take the datasets that have at least four versions. For managing the data, we consider a two-window based data management technique, no-memory approach. Others could have different preferences to make the scenario and for managing the defect datasets of the versions.

In our empirical study, we consider a single base learner to execute our proposed DP detection method. One could explore other classifiers to know the prediction performances. The aim of this study is to recognize the significant distribution differences that make the prediction misleading. Another possible threat is the adopted statistical test. We calculate the magnitudes of differences using Hedges'g. One could consider the other post hoc tests to identify the DP's in the datasets. As a consequence, the results might be different, although our adopted techniques work better on such large datasets in CVDP.

4.9 Chapter Summary

Cross-version defect prediction (CVDP) is a more realistic scenario for predicting defects of software projects compared with cross-project. Very few studies have been conducted in the domain of CVDP specifically data with different distributions. In this work, we formalize the issue of data distributions difference of two consecutive versions of defect datasets as drift propensity (DP) in cross-version settings. In our work, we propose a DP detection method that identifies the distribution differences of two consecutive versions of the defect datasets. We conduct our empirical study on 26 versions of six projects. We observe that significant distribution differences

affect the prediction performance. The considered statistical test ensure that the DP's are statistically significant and obtain large effect size by calculating the magnitudes of the data distribution differences. It is observed that the exponential increase of the error-rate indicates the DP's in the cross-version of chronological defect datasets. Therefore, the identification of DP's in the versions may be useful for building more reliable prediction techniques for CVDP.

In the future, we seek to find the sources of DP's in the defect datasets. We intend to investigate the proposed DP detection technique in other domains.

Chapter 5

The role of Class Rebalancing Techniques in Eliminating Concept Drift in Chronological Defect Datasets

5.1 Introduction

Software defect prediction (SDP) has been one of the most prominent research areas in software engineering (SE) in recent decades [121, 123]. SDP helps developers to perform early identification of faulty modules in software and assists project managers in maintaining a trade-off between quality and time-to-market [49, 98].

In the context of prediction models, if the underlying data distributions change after a stable period for any unforeseeable reason, the performances of models based on these data also degrade [111]. This phenomenon is referred as “Concept Drift” (CD), which can be formally defined as follows. Given two distinct time-points defining the time interval $[t, t + \Delta]$, with feature vector x and the class label l then the distribution is provided as $D_{(t,\Delta)}(x, l)$. CD occurs between two distinct time-points, t and $(t + \Delta)$, *iff* $\exists t :$

$D_{(t)}(x, l) \neq D_{(t+\Delta)}(x, l)$ where $D_t(x, l) = D_t(l|x)D_t(x)$ denotes the joint probability distribution at time t between the attributes in feature vector x and class label l .

Identifying CD is a relatively new research area in data mining and machine learning [31], but is a well-known phenomenon in data-stream mining [51, 111]. When the underlying distribution changes over time in non-stationary environments [51], the data behavior is susceptible to CD. This problem was first reported as “data drift” by Schlimmer and Granger [151] who observed it while working on noisy data that was changing over time [111, 114]. This aspect of SDP has received little attention and also, little is known about how drift-prone the defect datasets are.

To build more effective prediction models, the research community needs to tackle the challenge of data distributions that change after minimal stable period [167]. Therefore, it is necessary to be conscious of dataset shifts and their implications in the SE domain. Dataset shifts can generate biased prediction results, for example due to changes in class prior probabilities. Changes in data probability, including probability distributions, can lead to inaccurate results, and even a well-trained prediction model will become outdated in the face of such drift, as noted by Dong et al. [31]. Furthermore, other researchers [11, 156, 179, 181, 183, 184] have reported that the distribution changes among the versions in chronological defect datasets. Findings from streaming data analytics verify that if the historical data change over time, the prediction models become outdated [111]. Likewise, Ekanayake et al. [35, 36] and Bennin et al. [13] confirmed that the accuracy of SDP models fluctuates if the chronological data changes over time, meaning that the prediction performance varies.

There are many SDP models that use machine learning to predict faulty modules in software [61, 67, 140, 187, 190]. These models use the historical

data (on which a prediction model is trained) and predict defects for upcoming projects. Among these SDP techniques, cross-version defect prediction (CVDP) [183] is a process wherein the historical data from prior versions of a project are used to train models to predict defects in the latest version of the same project. As this reflects the actual process of software development, it is more realistic and effective than cross-project and cross-validation defect prediction [66, 144, 178]. This approach to determining the training and test sets is also referred to as chronological splitting [109, 127] in the area of software-effort prediction.

In addition to the practical relevance of CVDP, two seemingly contradictory key facts motivated us to conduct our study in the CVDP context. 1) The use of long-standing software results in the co-existence of several versions, and consequently the characteristics of development-related data, which are used to build prediction models, might change with each release after a stable period, and 2) at the same time, the defect-inducing patterns of previous versions might be relevant to defect prediction for the latest version. Therefore, a prediction model built upon a prior version of the same project may produce better results than a prediction model that uses data from different projects [156], as the latter would share similar distribution information among releases.

Software defect datasets are usually imbalanced with defective and non-defective modules [15]. Previous studies have tried to show performance improvement for SDP models by adopting rebalancing techniques [15, 161]. For example, Tantithamthavorn et al. [161] show the impact of class rebalancing techniques on SDP models. They point out that drift develops when the class distributions of training and testing sets are not similar. Furthermore, the change of prior probabilities of classes is responsible for drift [118]. Since class rebalancing techniques try to produce a similar representation of two classes, it may eliminate drift and impact the performance of CVDP.

In this work, we conduct a systematic investigation to examine and assess how drift-prone the chronological defect datasets are and examine whether class rebalancing could help eliminate CD from chronological defect datasets and impact the prediction performance for CVDP. In other words, we assess how drift-prone the resampled datasets are and impact the performance of CVDP. Thereby, we design the following two research questions:

(RQ1) How drift-prone are the chronological defect datasets?

(RQ2) Does class rebalancing eliminate CD from chronological defect datasets and impact the performance of CVDP?

Based on the above research questions, we conduct empirical investigation on 36 versions of 10 open-source software projects donated by Madeyski and Jureczko [117]. Our results reveal that 50% of the chronological defect datasets are drift-prone while applying the most popular classifiers used from the defect prediction literature and the class rebalancing techniques impact the prediction performance of CVDP models and eliminate the CD of the chronological defect datasets by up to 31%.

To the best of our knowledge, this study comprises the first endeavor to detect CD in the chronological defect datasets and empirically check whether class rebalancing can help reduce the detection rate of CD and impact the prediction performance.

We encourage the practitioners to identify CD in chronological defect datasets for CVDP and suggest adding class rebalancing in the CD elimination process to improve the prediction performance.

5.1.1 Chapter Organization

The rest of the work is organized as follows: Section 5.2 summarizes the closely related work describing the studies about CVDP. Section 5.3 summarizes the preliminaries by describing the background information and the description for concept drift for the cross-version defect prediction. Section 5.4 presents the concept drift detection (CDD) framework for detecting concept drift in chronological defect datasets and illustrates its fundamental steps. In Section 5.5, we describe the experimental methodology, studied datasets, class rebalancing techniques, defect prediction models, performance evaluation measures, statistical test, and experimental setup. The experimental results and analysis are presented in Section 5.6. Further analysis with discussions are presented in Section 5.7. Section 5.8 highlights the potential threats to validity of our experimental results and finally, we conclude in Section 5.9.

5.2 Related work

5.2.1 Cross-version defect prediction

CVDP has drawn significant attention because of its applicability in practice. Among the CVDP literature, most works using machine learning models assume that the historical data collected from software projects used to build prediction models maintain chronological order.

For a realistic evaluation, prediction models have been carried out for defect prediction using multiple versions of software projects [11, 168, 183]. For example, Bennin et al. [11] carried out a comparative study of 11 density-based prediction models using effort-ware measures. The empirical results show that K-star and M5 as the best performers in the CVDP context. Xu et al. [183] mentioned that the differences of the prediction performance obtained

from the experimental results were not statistically significant among all the methods.

Recent work has either emphasized to enhance the training sets or utilized the cross-project defect prediction (CPDP) approaches for CVDP. Xu et al. [179] proposed a framework for CVDP by leveraging hybrid active learning combined with kernel principal component analysis (KPCA) where features are selected from the current version and merge with the previous version and prepared a representative module sets (i.e., mixed training sets). The proposed framework tried to enrich the training sets. However, considerable effort is required to label selected modules and produce a suitable training set. Another shortcoming of this approach is that it requires one to convert the metrics back to the datasets for an estimator, and it influences the result of estimation while the models are constructed, as described by [184]. Furthermore, the study of Amasaki [3] focused on the effects of CPDP approaches under CVDP. These studies were conducted by utilizing multiple older versions. However, some methods from CPDP were not considered to compare with the proposed methods. Moreover, Shukla et al. [156] addressed the problem of multi-objective optimization in CVDP (i.e., maximizing recall by minimizing cost and misclassification) and conducted experiments on 30 versions from 11 projects.

An insightful work [184] in CVDP is based on the ranking task. In this work, Yang and Wen addressed CVDP as the problem of multicollinearity. They investigated ridge and lasso regression in 41 versions from 11 projects. Ridge regression performed better compared with baseline methods. However, their work focuses on ranking tasks, but not on the defect classification.

The works described above do not consider the practical scenario that the software companies grow with time, and the developed techniques may be obsolete after a certain time point. The works [3, 11, 179, 184] discussed are

likely to be developed for stationary environments. Changes in the distributions, can lead to inaccurate results, and even a well-trained prediction model will become outdated in the face of such CD, as noted by Dong et al. [31].

5.2.2 CVDP considering distribution differences

In this section, we present the works that tried to remove the distribution differences between two adjacent versions of the software system for CVDP.

Xu et al. [181] tried to mitigate the distribution differences between the two versions by utilizing dissimilarity-based sparse subset selection (DS3). The study was conducted on 56 versions of 15 projects for CVDP. They discovered that distribution differences degraded the prediction performance of all the models. They tried to alleviate the distribution difference between two adjacent versions by deleting labeled modules. Moreover, if modules are deleted from the prior version, it hinders an effective prediction model's performance. However, it requires extra effort to level the representative modules. To overcome this issue, Xu et al. [183] proposed a two-stage training subset selection method by conducting a large-scale experiment on 50 versions of 17 projects. In both studies [181, 183], modules were removed from project versions in an attempt to alleviate the distribution differences and produce a suitable training set. Inevitably, this caused loss of data carrying information about the current and prior versions of the projects. These losses of information are a drawback to the performance of such models.

Even though the studies above considered the chronology of the projects' versions, they did not consider the CVDP as an learning scenario where the versions appear after a certain period of time. For this, it may be leading to data drift. Consequently, the prediction performance drop could occur in case concept drift occurs in the associated versions.

Concerning concept drift, very few studies examined it in SDP [13, 35, 36]. For instance, Ekanayake et al. [36] conducted an investigation in open source projects for change-level defect prediction. They observed the impact of concept drift leading to prediction performance deterioration in SDP. In our prior works [85, 86], we also observe the significant impact of concept drift in SDP. To a large extent, these studies guide us to consider the chronological defect datasets as CV data stream to reflect the CVDP in practice. In the current state of CVDP research, the detection of CD is required for improving prediction performance in the CV settings.

5.3 Preliminaries

5.3.1 Concept of dataset shift

The recent prevalence of technology has created an enormous amount of data available in streaming format. In general, streaming data are assumed to be non-stationary [30]. The data-generating process in non-stationary environments is variously characterized by data shift, or concept drift [30, 100]. The phenomenon has been defined by various researchers in data-stream mining [111, 167, 175], and is also known as prior probability shift [175]. Data drift occurs if the probability of X at time t changes. Formally, it can be defined as $P_t(X) \neq P_{t+1}(X)$. Furthermore, data drift can be explained from the perspective of two-window-based data distributions known as historical data and new data. In Figure 5.1, the historical data D_t to D_{t+6} are considered as the old window and the new data D_{t+7} are considered as the recent window, and are defined by the user as specified by Lu et al. [111]. The windows continually move as new data appear over time. To adapt to the changes in data distribution, the prediction model needs to be regularly updated, as widely acknowledged in the literature [31, 111, 114].

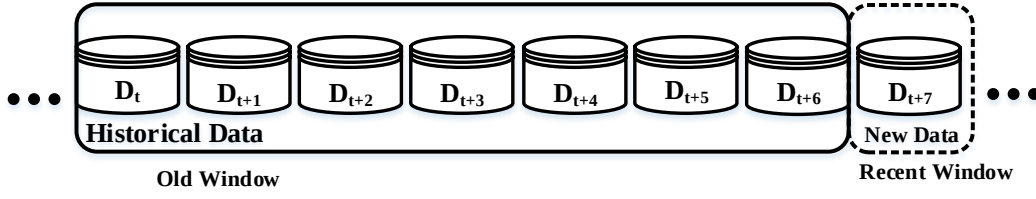


FIGURE 5.1: Two-window based data distribution at timestamp $t + 7$.

5.3.2 Data drift detection methods

The drift detection methods could be categorized in three methods: error estimations, distribution change detection, and statistical-test based methods. In this section, we briefly describe some of them.

Error-rate based methods: These methods focus on monitoring the error-rate of the learning algorithms [114]. Drift Detection Method (DDM) designed for online scenarios is the most traditional technique in the data-streaming literature [173]. DDM was the first model to be developed to define a warning level and a drift level according to the changes in the data distribution. The learning model makes a decision whenever the data become available. Another widely used method is early drift detection method (EDDM) [6]. EDDM considers the average distance between two error-rates and the standard deviation of the classification model. Both DDM and EDDM are threshold-dependent and use a single instance to identify the drift in streaming data. ADaptive WINdow (ADWIN) is a two-time window-based drift detection method [17]. The sliding window is maintained to adapt to the concept drift where window size changes according to the distribution changes. The Hoeffding Drift Detection Method (FHDDM) [139] also utilizes the sliding window. It monitors the changes in the probability of current predictors with the previous predictor of maximum probability.

Distribution-based methods: This type of detection method considers the distance metric to identify dissimilarity between the current and historical

distribution. If the dissimilarity between current and historical distribution is dissimilar enough to cross a (user supplied) threshold, the algorithm triggers an alarm to update the model. The users define historical and recent data to identify drift. Figure 5.1 shows sliding two-window based data distribution at different timestamp where historical data D_t to D_{t+6} are considered as the old window and the new data D_{t+7} are considered as the recent window. More information about distribution-based drift detection methods can be found in the literature [31, 42, 111, 175].

Hypothesis-test based methods: This type of method also considers the sliding window approach with different statistical testing strategies. The Statistical Test of Equal Proportions Detection (STEPD) assesses two-time windows, i.e., the most recent window and overall window, and computes the changes of error-rate for each timestamp. The users define the window size. De Lima Cabral et al. [100] uses Fisher’s Exact test with two windows and monitors prediction results of the two windows. Similar to the work of Cabral et al., in our proposed framework, we adopt the strategy of the sliding window.

5.4 The CDD framework

This section lays out the concept drift detection (CDD) framework for detecting CD in chronological defect datasets. This framework has four primary steps: 1) data processing, 2) construct CV defect models, 3) agreement and disagreement measurement (calculation of test statistics), and 4) significance test (hypothesis test). Figure 5.2 provides an overview of the CDD framework.

Step 1 (Data processing): For implication in practice, we formulate the chronological defect data as a CV data stream (i.e., windowed stream). The adopted version-based moving-window approach (Figure 5.3) used to build the CV

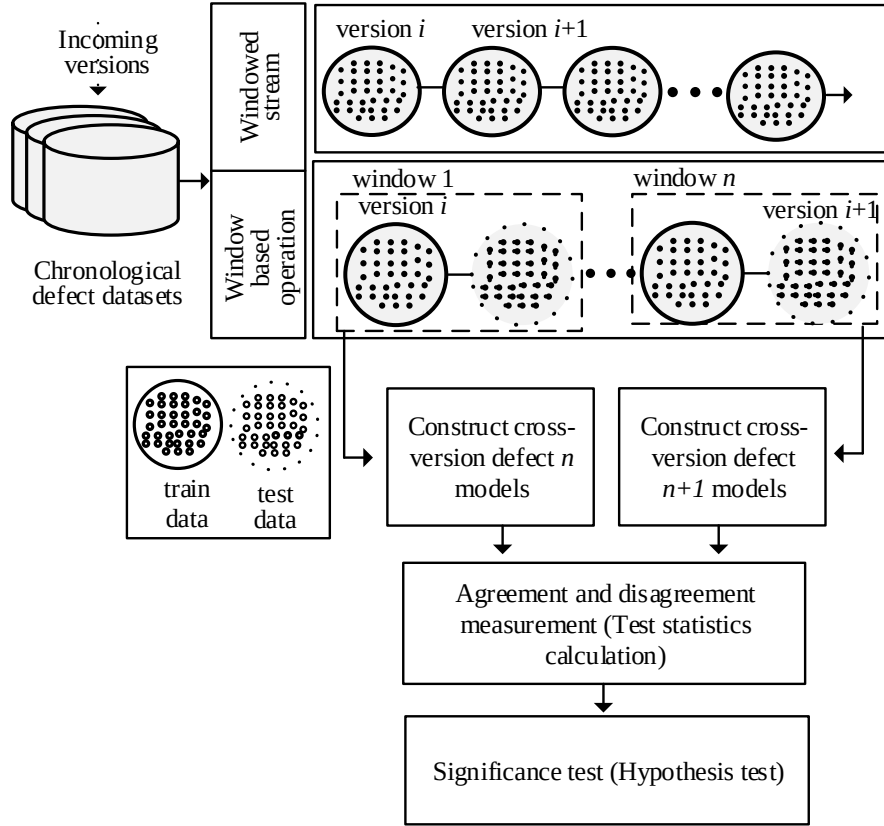


FIGURE 5.2: An overview of CDD framework.

classifiers is the most closely related to actual practice, and thus the most practical strategy is to create the window using the method suggested by Klinkenberg et al. [91]. We collect all the available versions from the sources

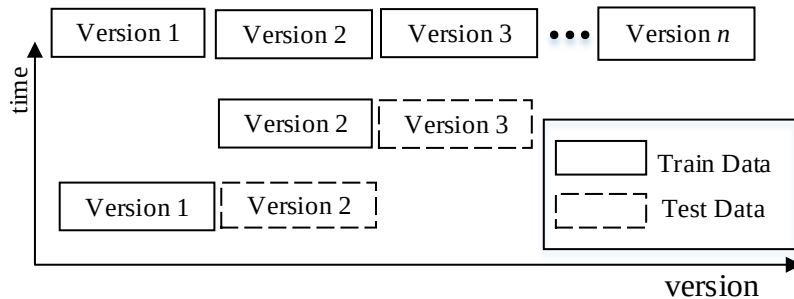


FIGURE 5.3: Version-based moving window, where window size refer to one version's defect data.

and restructure them according to chronological order. The windowing technique manages the versions. The old windows' distributions are discarded by following the moving-window approach. This is the chronological splitting procedure [108, 109, 127] in the window-based operation, known as the data-management approach, used in studies of data-stream mining [91]. The distributions of each version considered as one window.

Step 2 (Construct cross-version defect models): In CVDP, two nearest versions are considered for the defect prediction, bearing in mind that consecutive versions share similar characteristics, which helps to train the model accurately, as suggested by Amasaki [3–5]. Defect prediction models are trained on the earlier version and used to predict the defects of the subsequent version. As discussed above, we utilize the version-based moving window approach to split the versions, as prior work applied a similar strategy (i.e., split the versions). For example, Xu et al. [181] consider splitting the versions of the defect datasets and tried to mitigate the distribution differences between two consecutive versions. For instance, for project ant (see Table 5.2), it contains five versions (1.3 to 1.7). The prediction model is trained on version 1.3 and used to test the model on version 1.4. Afterwards, the prediction model will be trained on the version 1.4 and tested on the version 1.5.

Step 3 (Test statistics calculation): Test statistics are the measurement of pairwise agreement and disagreement (e.g., diversity) measures of two consecutive CV defect predictors. Several studies [100, 118] calculated the agreement and disagreement of the classifiers to detect drift in data streams. For example, De Lima Cabral et al. [100] calculated the diversity of the classifiers using sliding windows. We adopt a similar strategy to calculate the agreement and disagreement of two consecutive CV defective models using a version-based moving window approach. As an example, let $X = x_1, \dots, x_n$ be a distribution and $o_1 = [o_1(x_1), \dots, o_1(x_n)]$ is a binary vector that represents the output of the classifier c . Suppose for a pair of classifiers c_1 and c_2 and n^{ab} is the all

TABLE 5.1: The test statistics of a pair of classifiers at time t and $t + 1$.

Instances	window at time t	window at time $t + 1$
# of correct	$(q_1 + t_1)$	$(q_2 + t_2)$
# of incorrect	$(r_1 + s_1)$	$(r_2 + s_2)$
Total	$(q_1 + t_1 + r_1 + s_1)$	$(q_2 + t_2 + r_2 + s_2)$

possible outcomes of instances $x_j \in X$ for which $c_1(x_i) = a$ and $c_2(x_i) = b$.

We calculate the test statistics for consecutive pairs of CV defect models. Let R_i and O_i be the distributions obtained from two consecutive CV data at time t and $t + 1$. The two feature spaces are transformed into $V_i = (q_1, r_1, s_1, t_1)$ and $V_j = (q_2, r_2, s_2, t_2)$. For t time window, (q_1, t_1) are the correctly classified positive and negative instances, and (s_1, r_1) are the wrongly classified positive and negative instances, respectively. Similarly, for $t + 1$ time window, (q_2, t_2) are the correctly classified positive and negative instances and (s_2, r_2) are the wrongly classified positive and negative instances, respectively. Thereby, the test statistics are calculated. The agreement and disagreement measures are the consistent and inconsistent decisions of the total considered observations. These reflect the variety of responses of the pair of CV predictors that changes at time t and $t + 1$, i.e., probability of diversity between a pair of CV learners. Thus, from 2×2 contingency table (Table 5.1), the test statistic of correctly and incorrectly classified instances at t and $t + 1$ time windows (i.e., variety) are calculated where $(q_1 + t_1)$ (i.e., agreement) are the number of correctly classified instances and $(r_1 + s_1)$ (i.e., disagreement) are the number of incorrectly classified instances among the $(q_1 + t_1 + r_1 + s_1)$ total number of instances at t time window, respectively. Furthermore, $(q_2 + t_2)$ (i.e., agreement) are the number of correctly classified instances and $(r_2 + s_2)$ (i.e., disagreement) are the number of incorrectly classified instances among the $(q_2 + t_2 + r_2 + s_2)$ total instances at $t + 1$ time window, respectively.

Step 4 (Hypothesis test): Prior research effort on concept drift detection [91, 100, 116, 135] assumes that a prediction model trained on a time window (past data) would have no statistical difference in the prediction performance of the model of consecutive time window (data of next period) when there is no CD. We follow the same hypothesis to measure the CDs. If a defect prediction model trained from the previous version's data exhibits a statistically significant difference of prediction performance for identifying defective or non-defective modules of the models of consecutive version's data, then a CD exists. We conduct Fisher's Exact test at a 5% significant level to assess the statistical significance of the changes observed in two consecutive versions data obtained from step 3.

5.5 Methodology

5.5.1 Studied Datasets

To choose chronological defect datasets, we consider a corpus of publicly-available datasets containing 36 versions of 10 benchmark open-source software projects. This corpus was gathered by Jureczko and Madeyski [117] and has been well used in studies of CVDP [86, 179, 181, 183]. We collect the corpus from the SEACRAFT repository¹ [164] (which was formerly known as the PROMISE repository [163]). The statistics of these datasets are presented in Table 5.2 including the versions with their release date, the number of defective modules, defects, and the percentages of defects, respectively. The release date is in the form of day-month-year. Our selected open-source projects contain specified versions that appear in chronological order. The release date of the versions is collected from version-control repositories, which

¹<https://zenodo.org/communities/seacraft/>

TABLE 5.2: A statistical summary of selected chronological defect datasets.

Project	Version	Release Date	#Modules	#Defects	Defects(%)
ant	ant-1.3	12-08-2003	125	20	15.90%
	ant-1.4	12-08-2003	178	40	22.50%
	ant-1.5	12-08-2003	293	32	10.90%
	ant-1.6	18-12-2003	351	92	26.10%
	ant-1.7	13-12-2006	745	166	22.30%
camel	camel-1.0	19-01-2009	339	13	3.80%
	camel-1.2	19-01-2009	608	216	35.50%
	camel-1.4	19-01-2009	872	145	16.60%
	camel-1.6	17-02-2009	965	188	19.50%
poi	poi-1.5	24-06-2007	1988	141	59.50%
	poi-2.0	24-06-2007	9277	37	11.80%
	poi-2.5	24-06-2007	1988	248	64.40%
	poi-3.0	24-06-2007	125	281	63.60%
log4j	log4j-1.0	08-01-2001	135	34	25.20%
	log4j-1.1	20-05-2001	109	37	33.90%
	log4j-1.2	10-05-2002	205	189	92.20%
xerces	xerces-init	08-11-1999	162	77	47.50%
	xerces-1.2	23-06-2000	440	71	16.10%
	xerces-1.3	29-09-2000	453	69	15.20%
	xerces-1.4	26-01-2001	588	437	74.30%
velocity	velocity-1.4	01-12-2006	196	147	75.00%
	velocity-1.5	06-03-2007	214	142	66.40%
	velocity-1.6	01-12-2008	229	78	34.10%
ivy	ivy-1.1	13-06-2005	111	63	56.80%
	ivy-1.4	09-11-2006	241	16	6.60%
	ivy-2.0	18-01-2009	352	40	11.40%
lucene	lucene-2.0	26-05-2006	195	91	46.70%
	lucene-2.2	17-06-2007	247	144	58.30%
	lucene-2.4	08-10-2008	340	203	59.70%
synapse	synapse-1.0	13-06-2007	157	16	10.20%
	synapse-1.1	12-11-2007	222	60	27.00%
	synapse-1.2	09-06-2008	256	86	33.60%
xalan	xalan-2.4	28-08-2002	723	110	15.20%
	xalan-2.5	10-04-2003	803	387	48.20%
	xalan-2.6	27-02-2004	885	411	46.40%
	xalan-2.7	06-08-2005	909	898	98.80%

are used in the studies of Bangash et al. [7]. Each module of these datasets includes 20 static metrics with a labeled feature (BUG) for java classes listed in the Table 5.3. The defects (BUG) are in class-level. In the datasets, if the value of BUG is 0, the module is non-defective; and otherwise defective. Therefore, our experiment is conducted for CVDP on class-level.

TABLE 5.3: Description of the static code metrics [38] used in this study.

Metric	Description
WMC	Weighted methods per class
DIT	Depth of inheritance tree
NOC	Number of children
CBO	Coupling between objects classes
RFC	Response for a class
LCOM	Lack of cohesion in methods
Ca	Afferent coupling
Ce	Effective coupling
NPM	Number of public methods
LCOM3	Lack of cohesion in methods, different from LCOM
LOC	Number of lines of code
DAM	Data access metric
MOA	Measure of aggregation
MFA	Measure of functional abstraction
CAM	Cohesion among methods of class
IC	Inheritance coupling
CBM	Coupling between methods
AMC	Average method complexity
MaxCC	Maximum values of CC methods in a investigated class
Avg(CC)	Arithmetic mean of CC methods in a investigated class
BUG	Indicate the number of fault in a class

5.5.2 Apply Class Rebalancing Approaches

We apply four class rebalancing approaches in this empirical study. These are: Random Over Sampling (OVER), which increases the minority class samples created by randomly selecting and duplicating the minority samples that be the same size as majority class samples. Random Under Sampling (UNDER), which randomly reduces the “*majority class*” to be the same number of the minority class. A significant drawback of the UNDER approach is that it losses important information [161]. Synthetic Minority Oversampling Technique (SMOTE) over samples the minority class by creating “*synthetic*” instances. Bootstrap Random Oversampling Examples (ROSE) generates augmented instances under minority instance. In a practical scenario,

class rebalancing techniques are applied on training sets [161]. In our experiment, we use the techniques on the training sets. To apply OVER, UNDER, and ROSE, we utilize the implementation of the functions, `upSample`, `downSample`, and `ROSE`, respectively, provided by the `caret` package [97]. To apply SMOTE, we utilize the `SMOTE` function of the `R DMwR` package [165] with the default setting, k -neighbour value was set to five. The value of `perc.over=100` that refers the number of how many extra cases from the minority class are generated. The value of `perc.under=200` that refers the the number of how many extra cases from the majority classes are selected for each case generated from the minority class. As future work, I plan to utilize the different number of nearest neighbors to examine whether the results is sensitive to the number of nearest neighbors used.

5.5.3 Construct Cross-Version Defect Models

There is a plethora of methods used in software defect prediction [12, 161]. We adopt a manageable amount of classification techniques for our empirical investigation. For the experiments, we choose six machine learning algorithms to construct the cross-version defect predictors, i.e, Naive Bayes (NB), Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Logistic Regression (LR).

NB, a probability-based method, estimates score by simplifying the Bayes law that has been widely used in the defect prediction literature [121]. RF, a powerful machine learning method, builds multiple decision trees from bootstrap data samples. DT, a decision-based method, produces reasonable prediction results with various input data. SVM, kernel-based technique, determines hyperplane that separates negative and positive instances. LR is a regression-based method which measures the relationship between dependent and independent variables. KNN, an instance-based method, utilizes

the similarity of features to predict the new data based on how closely to the assigned values. Since the considered methods have settings of configurable parameter, we utilize the implementation of caret parameter optimization prior to building the models by the train function with the option of nb, rf, rpart, svmLinear, knn, and glm, respectively.

5.5.4 Calculate Model Performance

TABLE 5.4: Confusion matrix.

	Predicted Positive	Predictive Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

The defect dataset is classified into two groups: non-defective and defective. These performance indicators are calculated from the prediction models' outcome. To that end, the performance results of defect prediction classification models are assessed using the results calculated from a confusion matrix (Table 5.4) where the defective modules are considered as positive and non-defective modules as negative. The outcomes are classified as follows: (1) true positives (TP) refers to correctly classified positive samples as positive; (2) false positives (FP) refers to wrongly classified negative samples as positive samples; (3) true negatives (TN) refers to correctly classified negative samples as negative samples; and (4) false negatives (FN) refers to incorrectly classified positive samples as negative samples. The considered performance indicators have been used to precisely detect defective classes by several defect prediction studies [12, 15, 62, 161]. Using the confusion matrix described in Table 5.4, the measures are calculated and defined to assess the defect prediction models. Metrics such as the probability of false alarms (pf) and Recall (pd) were recommended as stable metrics for imbalanced datasets by Menzies

et al. [121]. To achieve high recall and precision simultaneously is challenging in practice, and such a measure, F-measure, calculates the harmonic mean between these two measures. Menzies et al. [122] also refuted precision as an unstable measure. We do not consider precision and F-measure in our empirical study. Measures such as Recall (pd), pf , and Gmean are considered in our study. A high pd and low pf indicate as best predictor. Higher values of Gmean denote better performance. Eqs. 5.1 to 5.3 define the mathematical definitions of Gmean, pf , and pd .

$$pf = \frac{FP}{FP + TN} \quad (5.1)$$

$$Recall(pd) = \frac{TP}{TP + FN} \quad (5.2)$$

$$Gmean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{FP + TN}} \quad (5.3)$$

5.5.5 Statistical Analysis of the Experiment

Statistical Comparison

To make a statistical evaluation and determine whether the performance differences are statistically significant across the benchmark methods, we assess the prediction results through the Mann-Whitney-U test at a 5% significant level and adopt *Win-Tie-Loss statistic* [92]. Using this statistic, we conduct a pairwise comparison between two predictors, i.e., a predictor without employing the class rebalancing technique with a predictor while utilizing the class rebalancing technique for CVDP. Three counters are computed: If the compared two predictors are not statistically different, they are marked as *tie*; *win* counter refers to the significance of a predictor while the other predictor is marked as *loss*.

Effect size Computation

To examine the practical and statistical significance of the empirical results, we adopt a robust effect-size computation method, (Cliff's δ), to compute the magnitude of the performance differences. The method of computing Cliff's δ effect size was recommended by Kitchenham et al. [90] for its effectiveness in handling tied results. The values of δ ranges from zero to plus one (+1). A high value indicates more practical and significant results, where zero (0) values denotes that the results are identical. We interpret the practical significance using the magnitude thresholds of Romano et al. [148] as follows: negligible effect ($\delta < 0.147$), small effect ($0.147 \leq \delta < 0.33$), medium effect ($0.33 \leq \delta < 0.474$), and large effect ($\delta \geq 0.474$).

5.5.6 Experimental Setup

The main goal of this empirical study is not only focused on applying the class rebalancing techniques on each version for CVDP, but it also evaluates the robustness of the impact on CD. Thus, we conduct an empirical analysis to answer the RQs.

(RQ1) How drift-prone are the chronological defect datasets?

Motivation: CD refers to the data distributions that are being collected may vary after a minimum stable period [41, 42]. Such drift may negatively influence the prediction performance of a model trained on past datasets when applied to the new datasets, as noted by Dong et al. [31]. Based on our previous works [85, 86], we confirm that the accuracy of prediction models negatively affect due to the changes of defect data distributions. However, little is known about how drift-prone the chronological defect datasets are. Therefore, we set out to explore the studied chronological datasets through the RQ1. In particular, we leverage statistical test to identify CD.

Approach: Prior research effort on CD detection [100, 116, 135] assumes that there is no CD if a prediction model trained on a time window (past data) would have no statistical difference in the prediction performance of the model of the consecutive time window (data of next period). We follow the same hypothesis to measure the CDs in the chronological defect datasets. If a defect prediction model trained from the previous version's data exhibits a statistically significant difference of prediction performance for identifying defective or non-defective modules of the models of consecutive version's data, then a CD exists. Section 5.4 provides an overview of CD detection process for chronological defect datasets. We plot the CD detection distributions for each prediction model in Figure 5.4.

(RQ2) Does class rebalancing eliminate CD from chronological defect datasets and impact the performance of CVDP?

Motivation: Class rebalancing approaches aim to rebalance the classes of the training datasets before constructing a model. Such approaches try to produce a similar representation of the defective and non-defective modules (two-class) prior to developing a defect prediction classification model. Turhan [167] alluded class rebalancing approaches may lead to influence for CD. Tantithamthavorn et al. [161] pointed out that drift develops when the class distributions of training and testing sets are not similar. Again, the change of prior probabilities of classes is responsible for CD [118]. Since class rebalancing techniques try to produce a similar representation of two classes, it may eliminate CD and impact the performance of CVDP. Furthermore, the findings of prior research efforts have revealed the benefits of adapting class rebalancing approaches for SDP [15, 161]. Through the RQ2, we assess whether class rebalancing techniques can help reduce CD and impact the performance of CVDP.

Approach: We compute and analysis the CD detection and CVDP performance values between the results of with and without class rebalancing

techniques. As mentioned in Section 5.4, we consider the chronological order of the versions to build the CVDP models, such that we train a model using a version and test it with subsequent version's defect datasets.

To assess whether class rebalancing techniques can help reduce CD, we compare the CD detection results with and without class rebalancing techniques among the classification techniques. We plot the CD detection distributions while utilized class rebalancing techniques in Figure 5.6. Besides, we compute the differences of the CD detection rate by subtracting the results of with and without class rebalancing techniques in Figure 5.7.

To compute whether class rebalancing techniques impact the performance of CVDP models, we compare the results of each performance measure with and without class rebalancing techniques. In this experimental setup, we use each resampling set generated by a class rebalancing technique as input to the studied five defect prediction models. We consider 10 benchmark projects containing 36 versions. As mentioned, the experiment is conducted in the cross-version scenario. Therefore, each model has 26 performance values for 26 consecutive pairs for each performance measure. We adopt three performance measures. For the baseline, our experiment yields 468 performance measure values ($26 \text{ performance values} \times 6 \text{ classifiers} \times 3 \text{ performance measures}$). For 4 class rebalancing techniques, our experiment yields (468×4) 1872 performance measure values. After that, we plot the distribution of performance differences using the boxplots for each performance measure corresponding to each classifier and class rebalancing techniques in Figure 5.8.

5.6 Results

(RQ1) How drift-prone are the chronological defect datasets?

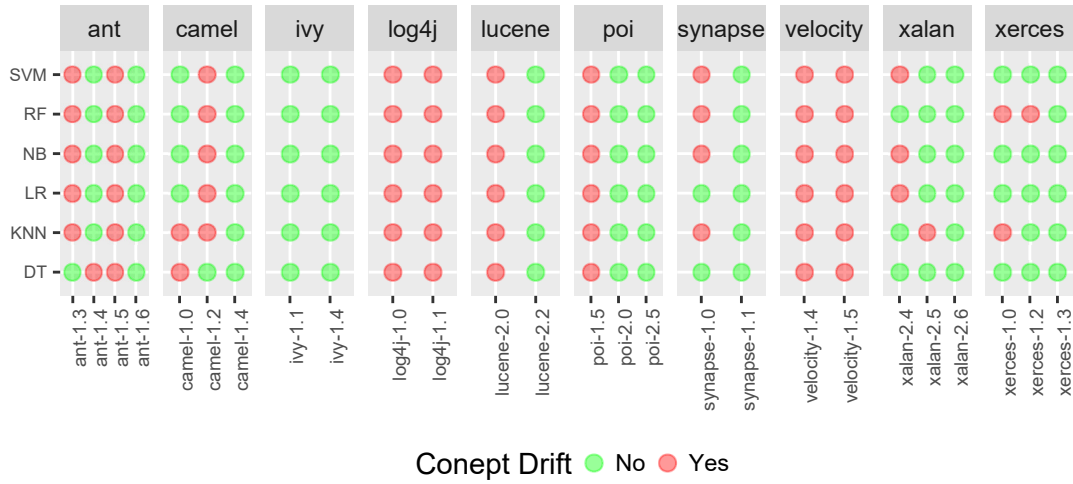


FIGURE 5.4: CD detection in the studied benchmark datasets while considered all metric sets (default datasets). A green symbol indicates that there is no drift between two consecutive versions of the project ($p\text{-value} < 0.05$ in Fisher's Exact Test); while a red symbol indicates otherwise ($p\text{-value} \geq 0.05$ in Fisher's Exact Test).

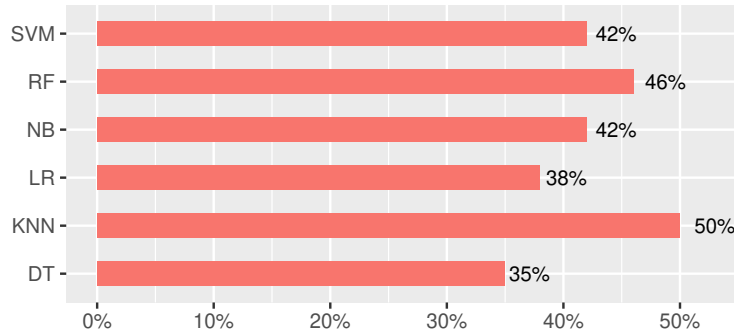


FIGURE 5.5: The percentage of CD detection across the prediction models of the 36 chronological versions of 10 open-source software projects. Here, a high value defines the worst score, i.e., the CD detection rate is high.

Figure 5.4 presents the CDs for each version of the considered chronological defect datasets corresponding to each classification technique. The different number of CDs identified by the prediction models may be due to the different models have different sensitivity to the evaluation of the datasets. We observe no drift for the ivy defect datasets when considered the classification techniques. On the other hand, all the versions of log4j and velocity projects are drift-prone when considered among the classification techniques.

5.6. Results

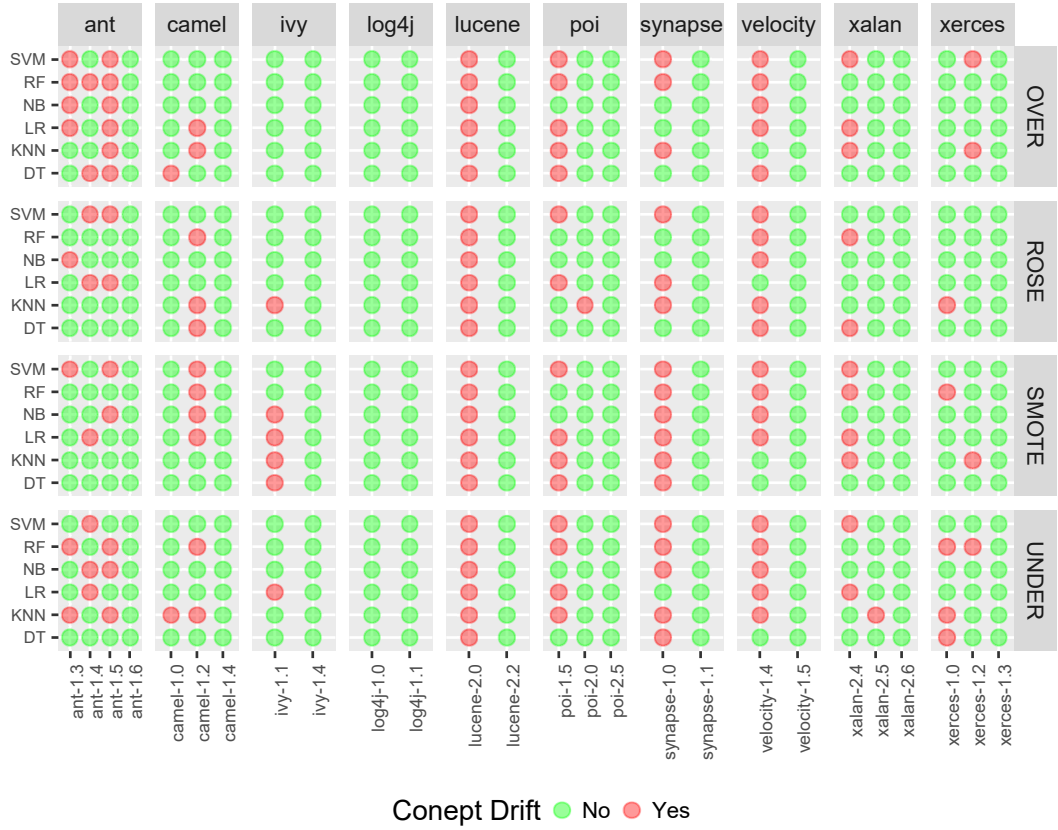


FIGURE 5.6: CD detection in the chronological defect datasets while considered class rebalancing techniques. A red symbol indicate a drift between two consecutive versions of the project ($p\text{-value} < 0.05$ in Fisher's Exact Test); while a green symbol indicates otherwise ($p\text{-value} \geq 0.05$ in Fisher's Exact Test).

We notice that classification techniques attain the different CD percentages among all the CV predictors, even if they are applied on the same chronological defect datasets (Figure 5.5). We find that 50% of datasets are drift-prone when KNN is applied, which is the maximum CD detection. DT achieves the minimum score of drift-proneness, i.e., 35%. **Overall, we observe that a maximum of 50% datasets are drift-prone when the prediction models considered the default data with KNN.** Furthermore, the use of deep learning-based models could improve the CD detection results, which are considered as future steps of this study. From the experiments, we observe that CD detection in chronological defect datasets is *model-aware*. This observation will help the practitioners to produce an effective drift adaptation technique

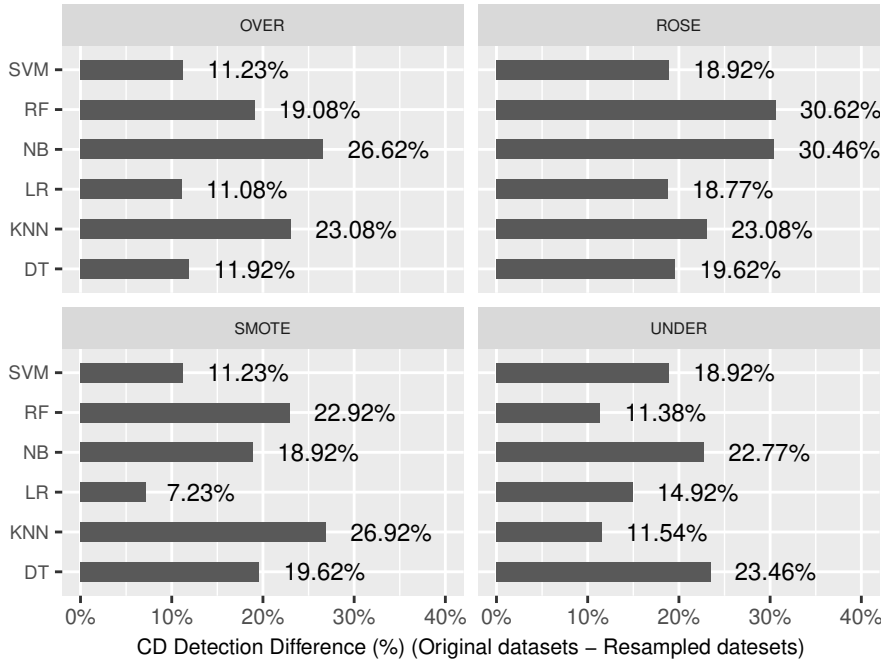


FIGURE 5.7: The percentage of CD elimination difference when applying class rebalancing techniques to 36 cross-version defect datasets. Here, a high value represents the best score, i.e., the CD elimination rate is high.

for detecting defects in such CVDP scenario. Furthermore, this observation agrees with the conclusion of Haug et al. [59] that concept drift detection is model-dependent. The instability of the CD detection rate among the classification models provides practical implications. In such a realistic CVDP scenario where false alarm rates are expensive, model-awareness reduces the sensitivity of CD from chronological defect datasets.

(RQ2) Does class rebalancing eliminate CD from chronological defect datasets and impact the performance of CVDP?

Figure 5.6 plots the CD detection distributions for each version of the considered defect datasets corresponding to each classifier and each class rebalancing technique. Like RQ1, the different number of CDs identified while considered class rebalancing techniques corresponding to each classification model. Figure 5.7 presents the percentage of the CD detection rate of the datasets while applying the class rebalancing techniques. The studied class

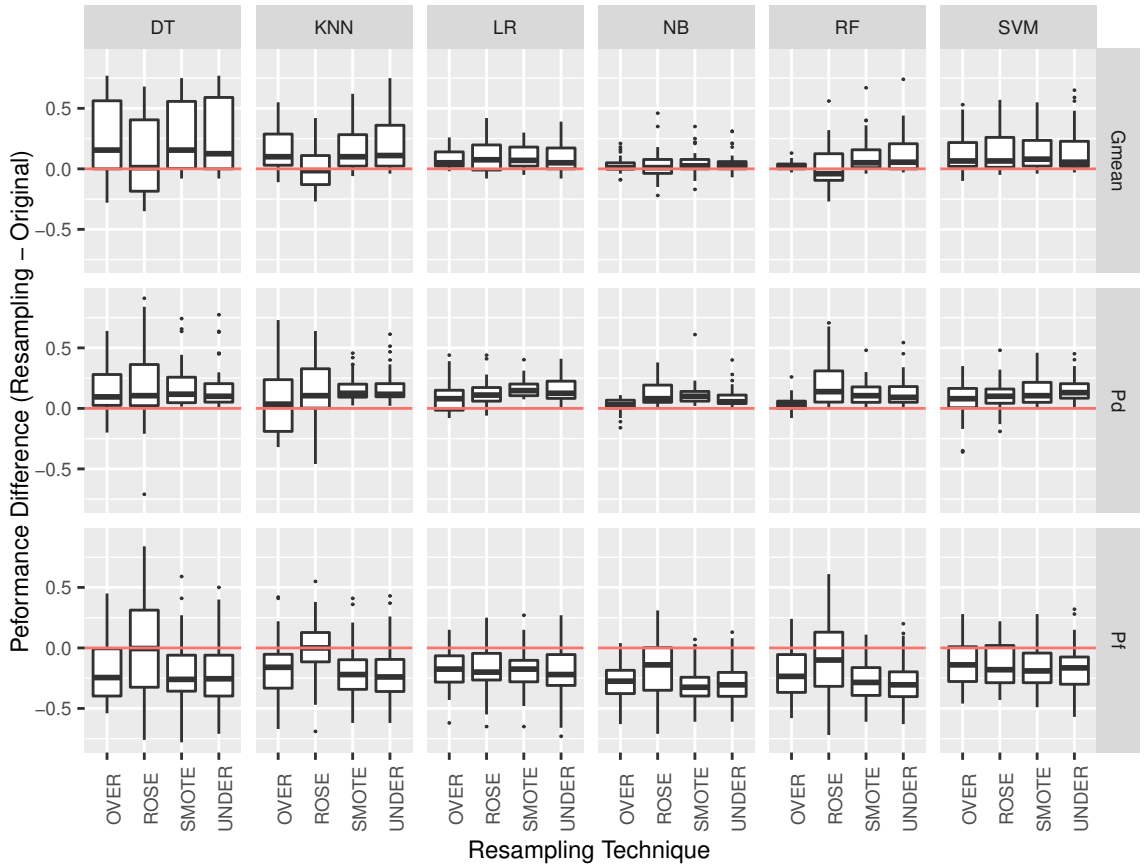


FIGURE 5.8: The CVDP performance difference, when employing class rebalancing techniques. A red line indicates no improvement (i.e., the performance difference, zero).

rebalancing techniques reduce the CDs by balancing the datasets by up to (CD detection for original dataset - CD detection with the resampled dataset, i.e., 46%-15.38%) almost 31% while employing ROSE with RF. It yields the highest average pd (i.e., 84%), which fulfill the prediction performance criteria defined by He et al. [62]. We also observe that ROSE with NB achieves a maximum score on average 84% for pd with (CD detection for original dataset - CD detection with resampled dataset, i.e., 42%-12%) 30% CD elimination rate. In contrast, while utilizing the default datasets, 42% drift-prone of that datasets, while a maximum mean pd of 62% was achieved by NB, followed by DT with an average score of 59%. The noteworthy observation is that ROSE with NB has a positive impact on reducing CD by up to 31%

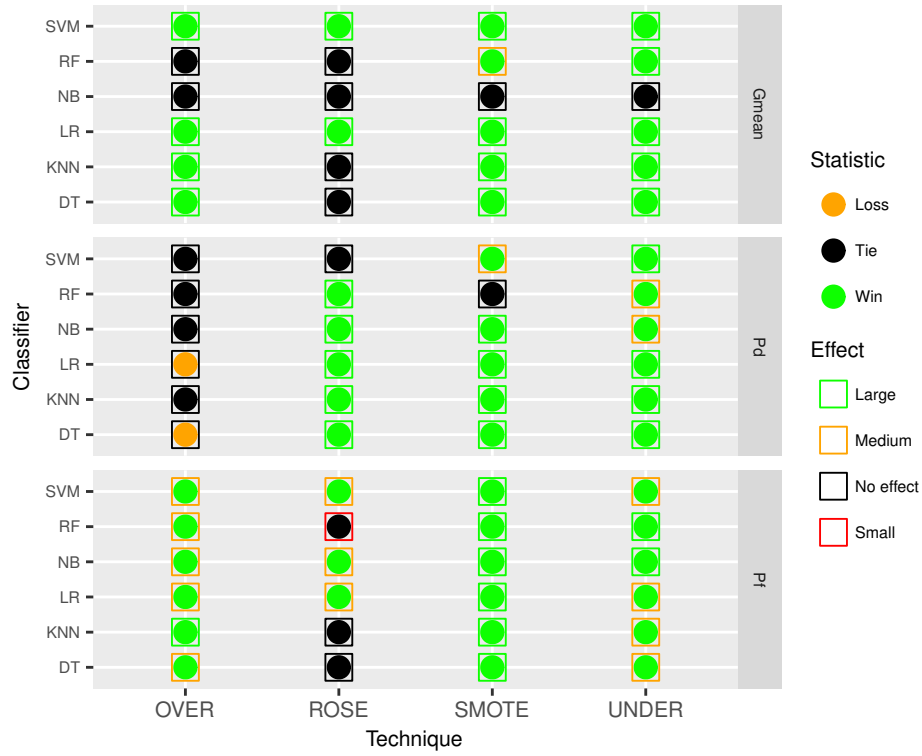


FIGURE 5.9: Mann-Whitney-U test win-tie-loss comparison with Cliff's $|\delta|$ effect size magnitudes of without class rebalancing vs. with class rebalancing techniques across cross-version defect prediction models. The complete green box indicates significant values.

and yields 84% performance improvement for pd of CVDP models. Looking at Figure 5.7, we observe that resampling the chronological datasets with class rebalancing techniques eliminates CD from the considered datasets and improves prediction performance. To avoid CD, this observation guides us to recommend practitioners to maintain the high performance of the prediction models over time for such a practical CVDP scenario where pd rates are costly.

Figure 5.8 presents the results of the performance differences when class rebalancing techniques are applied to CVDP models for each of the three performance measures. We present our experimental findings of each performance measure that are (1) not impacted the prediction performance and (2) impacted the performance by utilizing the class rebalancing techniques.

From Figure 5.8, we observe that the considered class rebalancing techniques impacted positively on Gmean for most of the CVDP models performance improvement. For the ROSE technique, the Gmean measure is less sensitive. Figure 5.8 shows that the ROSE class rebalancing technique yields unstable conclusions when applied to CVDP models. This observation supports the conclusion of Tantithamthavorn et al. [161], who observed that the ROSE technique produces a negative and positive impact on Gmean. From our experiment, we observe that the ROSE technique produces a negative impact when the CVDP models are built with RF. The SMOTE, under-sampling, and oversampling techniques substantially improve Gmean and shows a positive impact on the performance of CVDP models. The performance values of the Gmean for 75% (i.e., the values of 1st-3rd quartiles) CVDP models vary from -5% to 19%. Overall, when employing the SMOTE, under-sampling, and oversampling techniques, we find that the performance difference of Gmean for 75% (i.e., the values of 1st-3rd quartiles) of the CVDP models vary from 0% to 21%. These results indicate that these techniques tend to positively impact Gmean when they are employed for CVDP models.

Across all the CV models, the class rebalancing techniques substantially improve pd (i.e., probability of detection) and achieve a maximum score on average 84%. We find that while employing the ROSE with NB and RF, the proportion of CV defective modules that are accurately classified improves the models' performance on average up to 84%. From Figure 5.8, it can be seen that the SMOTE, undersampling, and oversampling techniques have a positive impact on the performance for correctly classifying the CV defective modules. Overall, we observe that the performance of pd for 75% (i.e., the values of 1st-3rd quartiles) of the CVDP models vary from 4% to 19%. Interestingly, the class rebalancing techniques substantially decrease pf and increase pd , which implies the positive impact of class rebalancing techniques

in classifying the CV defective modules.

To make a statistical evaluation and determine whether the effect of class rebalancing techniques are statistically significant on the CVDP models, we assess the results by comparing the performance of models trained on default imbalanced datasets with the performance of the models trained on the class rebalancing techniques using *Win-Tie-Loss statistic*. For a record in Figure 5.9 for the performance metrics used, the status of a win (solid green circle) and large effect size (empty green colored box) attached to the "circle" indicates the class rebalancing techniques outperformed the default (original) dataset trained models. A solid black box (combination with a black circle and an empty black box) shows otherwise. Class rebalancing techniques with more green circles and embedded green empty boxes indicate better performance. The noteworthy findings are: Undersampling and SMOTE trained models outperform the prediction models for all the performance measures. In some cases, oversampling and SMOTE techniques perform better when considered with Gmean. For all considered performance measures, the application of SMOTE and undersampling techniques based prediction models statistically outperform the prediction models trained on default (original) chronological defect datasets. For the pd performance measure, some losses are listed when oversampling method is applied to the CV prediction models. Based on the empirical analysis and statistical tests, we observe the followings:

1. ROSE technique eliminates CD by balancing the datasets by up to 31%. Overall, the added benefit of class rebalancing techniques is noticeable and reduces CD from the chronological defect datasets.
2. The experiment results allow us to guide practitioners to maintain high prediction performance of CVDP models over time to eliminate CDs.
3. The impact of SMOTE and undersampling techniques on CVDP models are statistically and practically significant regarding the considered

performance metrics.

5.7 Discussion

Is retraining a model actionable enough to remove CD?

To eliminate drift, the most common remedy in such a situation is to update the existing model [42, 111]. In Figure 5.10, we illustrate how a model is updated periodically. In Figure 5.10 (a), a stationary model is trained based on the historical data and tested on the new data. The model is never updated during the learning process. In Figure 5.10 (b), the prediction model is updated periodically when the new data becomes available. Lyu et al. [116] considered this strategy to update the models for artificial intelligence-dependent IT solutions. They observed that periodically updated models help eliminated the drifts. In our experimental setup, we follow the same strategy for defect prediction in the cross-version scenario. The CV models are updated in chronological order by maintaining a version-based moving window (Figure 5.3). Note that we do not use all the historical data of previous versions to retrain the model. Lin et al. [101] alluded that using all historical data may not lead to better performance. Through the RQ1, we update the prediction models by maintaining the sliding window that moves forward in chronological order where each window refers to one version data. Through the RQ2, we adopt the class rebalancing technique while updating the models. We observe that only retaining the model may not actionable enough, and the added class rebalancing techniques improve prediction performance and reduce drift. In summary, the experiment results reveal that when the studied classification techniques are trained on the default datasets, up to 50% of the datasets are drift-prone. The class rebalancing techniques improve the prediction performance of CVDP models and eliminate the drift of the chronological defect datasets by up to 31%.

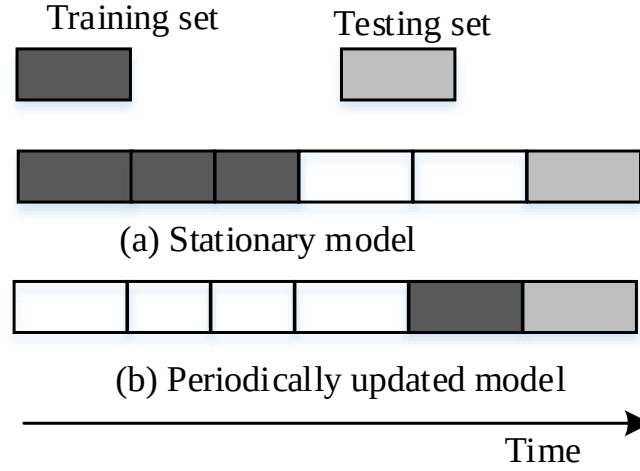


FIGURE 5.10: Scenario for periodically updated models [116].

By conducting this comprehensive and systematic empirical study, we discuss the following two observations from our analysis.

Model-awareness. We observe from the experiment that the classifiers' choice is too sensitive to the drift of the chronological defect datasets. NB achieves the best results without applying any class rebalancing techniques. However, 50% of the datasets are drift-prone with KNN. Among the considered classifiers, DT yields the minimum percentage of drift-proneness, i.e., 35%, while it could not obtain satisfying prediction performances. Our observation shows that model-awareness reduces the sensitivity of drift.

Update the model-not actionable. From the data streaming mining literature, the most guided strategy to adapt the CD is to update the model [42]. Our experiment reveals that updating the model does not always improve the prediction performance and does not reduce the drift from the data. While employing class rebalancing techniques during the drift detection process, we observe that the class rebalancing techniques eliminates drift up to 31% from the chronological defect datasets and improves prediction

performance. Therefore, we suggest adopting and applying class rebalancing techniques in the CD elimination process from the chronological defect datasets and improving prediction performance.

Looking at Figure 5.4 and 5.6, we observe that some versions of the projects have CDs while employed the class rebalancing techniques. However, there were no drifts when the models are updated without applying class rebalancing techniques. For example, for *ivy* project, CDs appear after employing ROSE techniques with KNN. But, there were no drifts when only the models are updated over the versions. This observation guides us to recommend practitioners to consider the class rebalancing technique as an attention technique for CD adaptation in chronological defect datasets. Overall, the class rebalancing technique can be thought of as one of the properties of drift adaptation in such a CVDP practical environment. From this empirical investigation, we provide a guideline to software engineering researchers and practitioners - If they wish to eliminate CD from the chronological defect datasets, they should be aware of the models and consider class rebalancing as an attention technique.

5.8 Threats to validity

5.8.1 Construct and internal validity

From the available benchmark defect datasets, we select those that maintain the chronological order of the software development cycle. We consider the chronological-split approach to make the CV scenario. Others could make different choices to conduct experiments in CVDP. Furthermore, the adopted empirical approach for this study does have an impact on the CD-detection and elimination results for CVDP. It should be noted that our CDD framework relies on the available class labels of testing data. Detection of CD in

the absence of class labels is beyond this empirical study, although this is quite an important limitation in practice.

One possible threat is about the ground truth information. Chronological defect data usually does not provide any ground truth information about the location of CD. From the considered datasets, Xu et al. [183] observed that 40% of the modules of the next versions were defective, which indicates the distribution differences. Such differences can make the versions drift-prone if it is considered as in a CV data stream. Based on our previous works [85, 86], we affirm the existences of CD in the software defect datasets.

In our comprehensive and systematic empirical study, we characterise the software with static code metrics, which may not capture other aspects captured by process metrics. This is a limitation imposed by the choice of datasets used. However, the considered static code metrics are widely-used in the CVDP literature [179, 181, 183].

5.8.2 External validity

We consider a limited number of cross-version systems. As a result, the conclusions may not be generalizable beyond the experimental environment and datasets used. However, to the best of our knowledge, our empirical study considers the chronological defect datasets that have been used in the studies of CVDP [11, 156, 179, 181, 183, 184]. Furthermore, Basili et al. [9] stated that it is difficult to make a general conclusion from an experiment in SE because the experimental process requires a very large number of relevant context variables.

In addition, the results of this empirical study rely on cross-version defect prediction. However, there are various scenarios of defect prediction in the literature (e.g., just-in-time defect prediction [37] and cross-project defect

prediction [64, 144]). Therefore, the lessons learned from the experimental results may differ from other defect prediction scenarios. We studied three performance measures. Our results may vary to threshold-independent performance measures (e.g., Area Under the receiver operator characteristic Curve (AUC) and Brier score) that can be explored in future work.

5.9 Chapter Summary

In this work, we conducted an investigation on concept drift (CD) and impact of four class rebalancing techniques, i.e., SMOTE, ROSE, undersampling, and oversampling on CD for chronological defect datasets. We conduct the experiment on 36 versions of 10 software projects where project versions appear in chronological order after a certain period. We train the cross-version (CV) models using six classification techniques and evaluate the performance using three performance measures. A version-based moving window approach is adopted to investigate the CD in the CV data stream. Through a comprehensive and systematic empirical study, we observe the following:

- Up to 50% of the chronological defect datasets are drift-prone while applying the most popular classifiers used from the defect prediction literature.
- The class rebalancing techniques improve the prediction performance of CVDP models and eliminate the CD of the chronological defect datasets by up to 31%.
- The class rebalancing techniques exhibit noteworthy significant and practical performance improvement when considering Recall and Gmean. Additionally, the models yield better performance enhancement when employing the SMOTE and undersampling techniques.

5.9.1 Concluding Remarks

Based on the findings of the experimental results, we make the following suggestions:

1. The class rebalancing techniques are beneficial when the practitioners wish to increase the ability of correctly classifying the CV defective modules.
2. We suggest to add class rebalancing techniques in the drift elimination process if practitioners wish to alleviate CD from chronological defect datasets.

Chapter 6

Cross-Version Defect Prediction with Feature Selection using Temporal Chunk-based Learning

6.1 Introduction

Software defect prediction (SDP) is considered as one of the predominant research topics by the software engineering (SE) community [66, 119, 160, 166]. In SDP, the prediction models are built using machine learning (ML) techniques to identify defective modules or classes that are likely to be defective [15, 119, 121, 137, 159]. This assists software developers to focus on those modules, which need special attention concerning allocate the scarce testing resources or code inspection [15, 93]. Over the years, many ML techniques have been proposed to build the classification models for SDP [56, 64, 66, 80, 120]. The SDP classification models utilize historical data (from which the prediction model is built) and predict the defects (with which the prediction model is tested). Among the SDP classification studies, within-project SDP is a scenario that trains a classification model on labeled modules and tests the model on the unlabeled modules within the project. This scenario can

be categorized into two schemes [56], which are inter-project SDP and intra-project SDP. Intra-project SDP utilizes the current version to build the prediction model and testing data from the same version of the software project, whereas cross-version defect prediction (CVDP) is a process where the historical data is retrieved from the previous version of the software project and tested on the current version of the same project, which is considered as more practical and realistic [156, 179, 181].

In the past decade, SE researchers pay attention to study in the CVDP context due to some unique aspects [156, 179, 181, 183]. For example, Shukla et al. [156] mentioned that prior version of a software project is more appropriate for building the prediction models for better results. Shukla et al. alluded that the project characteristics regarding architectural design will be more or less the same across the versions. Therefore, prior version becomes the more suitable training set for predicting the defects. We were able to identify the cited works (i.e., [3–5, 11, 56, 80, 156, 179, 181, 183, 184]) that studied long-running software projects for predicting defects in the CVDP context through our SDP literature review. Xu et al. [179, 181, 183] tried to mitigate the distribution difference among two consecutive versions. The studies of Amasaki [3–5] focused on the effects of cross-projects SDP approaches under CVDP. The studies [11, 56, 80, 156, 184] discussed are likely to be developed for stationary environments.

Due to the rapid development of the software industry, the prevalence of software systems has led to massive data volumes, making as a critical system versions over time [58]. Data generated in these environments are non-stationary [86]. The effects of non-stationary environments (NSE) are detrimental, and it poses challenges to learn from such an environment [176]. For example, data probabilistic properties change over time [30] which consequently can make a previously well-performing prediction model trained in a stationary environment become obsolete [94]. Furthermore, software

projects produce several versions that appear in temporal order and become complex due to the frequent changes made in each version [179]. For instance, the current version of a software project becomes more complicated due to the frequent changes made in some modules in the prior version, which leads to the changes in data distribution among the versions. If the relationship between variables of underlying data distribution changes (i.e., concept drift [30, 167]) due to unforeseeable reasons after a stable period, the prediction model would not be reliable [111]. In such a scenario, the learning environment is unlikely to be stationary. The researchers who studied CVDP seem reluctant to consider cross-version defect datasets as non-stationary data distribution and investigate further by considering the temporal order of the distributions. Since each version's data appears as a chunk in temporal order, the learning environment becomes temporal chunk-based learning. We formulate CVDP as a temporal chunk-based learning problem in which the data of different versions of a software project appear at each time step.

In an attempt to enhance the performance of SDP models, feature selection (FS) techniques are known to have a significant impact on SDP. Prior works [46, 93, 180] show that carefully selected features could achieve acceptable prediction performance in SDP. Therefore, it is needed to appraise the usability of FS techniques in the CVDP context while considering temporal chunk-based learning over the versions of a software project. Furthermore, FS techniques may avoid concept drift due to the variation of selecting features in each version of cross-version defect datasets. Therefore, our study investigates whether applying FS techniques separately on each version of the project when building CVDP classification models could improve prediction performance and the robustness of these trained CVDP models to concept drifts. Moreover, in temporal chunk-based learning, this could potentially improve performance of the prediction models and reduce the negative effect of concept drift on the defect data. Specifically, we perform

an empirical study aimed at answering the following two research questions (RQs):

- (RQ1) *How do the FS techniques impact the performance of CVDP?*
- (RQ2) *Which FS method is more robust to concept drift?*

Based on the above RQs, we conduct the experiments on 36 versions of 10 software projects provided by Madeyski and Jureczko [83, 117]. These datasets are selected because they have several versions and are widely used in the studies of SDP. The datasets contain static code metrics and are used for predicting defects for cross-version at the class levels. We consider: three best-performing FS techniques (correlation-based FS [26], consistency-based FS [27], and AutoSpearman [75], five prediction models (DT, KNN, LR, NB, and RF) and three evaluation measures (AUC, Recall, and *pf*). To the best of our knowledge, this study is the first endeavor to assess the impact of FS techniques on the performance of CVDP using temporal chunk-based learning. Thus, the contributions of our empirical study can be summarized as follows.

- Findings and insights which are useful to the software engineering researchers interested in improving the prediction performance of CVDP.
- Identify FS techniques that help to reduce the concept drifts and improve the prediction performance of CVDP.

The empirical study reveals that there are statistically significant differences in performance values between the results with and without FS techniques when evaluated with AUC, Recall, and *pf*. We find that 50% of the cross-version defect datasets are drift-prone while applying the considered classifiers. The FS techniques improve the prediction performance of CVDP models and reduce the drifts from the defect datasets.

6.1.1 Chapter Organization

The rest of the work is organized as follows: Section 6.2 summarizes the closely related work describing the studies about CVDP. Section 6.3.1 formulates the problem for cross-version software defect prediction. Section 6.4 describes the methodological procedure applied to conduct the empirical study. We report the results obtained from the experiments in Section 6.5. Section 6.6 discusses the impacts. Section 6.7 highlights the potential threats to validity of our experimental results and finally, we conclude in Section 6.8.

6.2 Related Work

CVDP has drawn significant attention because of its applicability in practice. Among the CVDP literature, most work using ML models assumes that the historical data collected from software projects is used to build prediction models. In this section, we discuss some of them.

Xu et al. [181] tried to mitigate the distribution differences between the two versions by utilizing dissimilarity-based sparse subset selection (DS3). The study was conducted on 56 versions of 15 projects for CVDP. They discovered that distribution differences degraded the prediction performance of all the models and attempted to mitigate the distribution difference between two adjacent versions by sparse subset selection. However, it needs further effort to level the representative modules, as claimed by Xu et al. [183]. To overcome this issue, Xu et al. [183] proposed a two-stage training subset selection method by conducting a large-scale experiment on 50 versions of 17 projects. In the first stage, the modules are selected by leveraging a sparse modeling approach. In the second stage, the modules are refined by using a dissimilarity-based sparse subset selection method. The selected modules are used for defect prediction by using a weighted extreme machine learning classifier.

For a realistic evaluation, prediction models have been carried out for SDP using multiple versions of software projects. For example, Bennin et al. [11] conducted an empirical study with a pair of open-source software systems using effort-ware measures. The empirical results show that K* and M5 achieved the best results among the 11 models, but they are affected by the size and defect ratio. Furthermore, Xu et al. [183] affirmed that the results obtained are not statistically significant among all prediction models.

Recent work has either focused on enhancing the training sets or utilizing the inter-project defect prediction (CPDP) approaches for CVDP. Xu et al. [179] proposed a two-phase framework by combining hybrid learning strategy and kernel principal component analysis (KPCA) where features are selected from the current version and merged with the previous version to obtain a representative module set (i.e., mixed training sets). By leveraging KPCA, mapped training sets are selected, and a linear regression model is applied to predict defects. The proposed framework tried to enrich the training sets; however, it required extra effort to produce suitable training sets.

The studies of Amasaki [3–5] focused on the effects of CPDP approaches under CVDP. These studies were conducted by utilizing multiple older versions. Furthermore, Shukla et al. [156] considered CVDP as a multi-objective problem (i.e., maximizing recall by minimizing cost and misclassification) and conducted experiments on 30 versions from 11 projects. However, these studies may not be sufficient to provide comprehensive guidelines on how to learn from NSE for CVDP.

An insightful work [184] in this area is based on the ranking of the predicted number of defects. In this work, Yang and Wen addressed CVDP as the problem of multicollinearity. They investigated ridge and lasso regression in 41 versions from 11 projects. Ridge regression performed better compared with the linear and negative binomial regression. However, this study focuses on ranking tasks but not on defect classification.

Harman et al. [58] conducted an investigation on 8 Hadoop versions for SDP by considering the learning environment as a temporal problem. They developed the prediction models by a tuned SVM classifier. They found that the prediction models developed with the recent version are not always the best predictors.

The methods described above do not consider the fact that software companies grow with time, and the developed techniques may be obsolete after a certain time point. The works [3–5, 11, 156, 179, 184] discussed are likely to be developed for stationary environments. For instance, Dong et al.[31] affirmed that a well-trained prediction model could be deceptive if the distribution changes over time. Even though the studies above considered the chronology of the projects' versions, they did not consider CVDP as a temporal chunk-based learning problem, although the versions appear after a certain period. This may lead to data drift [167]. Consequently, the prediction performance degradation could occur in case concept drift occurs in the associated versions. Concerning concept drift, very few studies examined it in SDP [13, 35, 36]. For instance, Ekanayake et al. [36] investigated several open-source projects and observed the impact of concept drift leading to model performance deterioration. In our previous studies [85, 86], we also observe the notable impact of concept drift in SDP. To a large extent, these studies guide to consider the temporal chunk-based learning to reflect the CVDP in practice.

6.3 Method

6.3.1 Problem Formulation

In this work, we formulate CVDP as the problem of temporal learning in which each version of the project is considered as a training dataset at each

time step. The training dataset is used to build CVDP models. At each time step, the model is updated when the subsequent version arrives. Since each version's data appears as a chunk in temporal order, the learning environment becomes temporal chunk-based learning. An example of version-wise data chunk is shown in Figure 6.1. As the first data chunk at time t_1 for version r_1 , there are two classes denoted by circles and diamonds. After some duration at time t_k , a data chunk enters in the chunk-based learning process where the feature space and decision boundary have changed for the version r_n . The term "chunk-based learning" refers to learning the data as chunk based on the "discard-after-learn" concept [81]. Here, the data chunk is used for learning only once and then discarded from the learning process to keep the memory space available for the next data chunk. In contrast, batch learning refers to only one subset of data presented for each epoch. The epoch numbers are not manageable because the data is trained repeatedly for the weights until they reach the stopping condition, affecting the datum order during the learning (i.e., the sensitivity of the learning data sequence) [81]. Junsawang et al. [81] suggested learning data as a chunk at a time to overcome the impact of sensitivity of the learning data sequence. Overall, we formulate CVDP as a temporal chunk-based learning problem in which the data of different versions of a project appear at each time step. Here, the data of each version is considered as a chunk. We consider the learning method based on the concept of discard-after-learn for temporal data chunks, in which the data of each chunk is utilized one single time.

In CVDP, two nearest versions are considered for the defect prediction. At each time step, the defects are predicted in the next version in CVDP, bearing in mind that consecutive versions share similar characteristics, which helps to train the model accurately, as suggested by Amasaki [3–5]. Based on this setting, we conduct 26 pairs of cross-versions of the considered defect datasets.

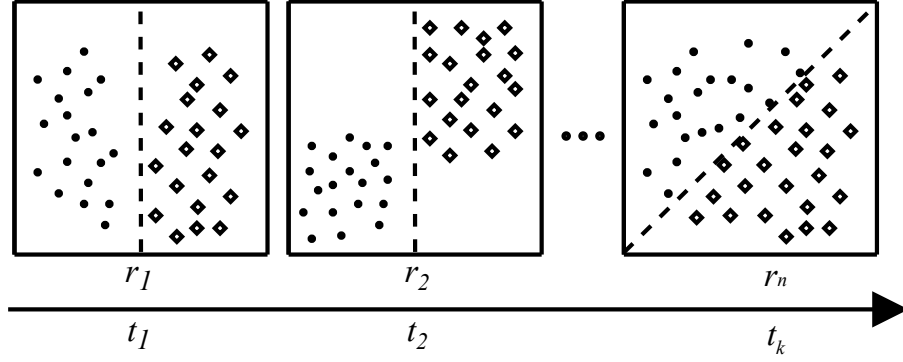


FIGURE 6.1: An example of version-wise chunk data with different classes.

To manage the data in temporal learning environment, windowing techniques [188] have been developed to manage the data [94]. In our case, the data of each version arrives as a chunk. Therefore, a chunk-based learning strategy is more appropriate. Moreover, the version-based moving window is validated and recommended by Harman et al. [58], where window size refers to one version's data (i.e., chunk). In our study, we consider the same strategy to build the prediction models for CVDP. In particular, the defect data is generated in such a manner so that the sequence of examples (a_t, b_t) arrives at time step t . The examples a_t belong to feature vector and b_t represents the discrete class label at time point t , in which each example is generated with the joint probability distribution $p_t(a, b)$. The examples can be produced by chunk-based (CB) settings [30, 94]. In each setting, a tuple $D_t = \{(a_t, b_t)\}$ is provided to the learning algorithm. The training examples could be generated by CB settings and form data $D = \langle D_1, D_2, D_3, \dots \rangle$, in which a key issue could occur, i.e., concept drift, for which the predictive model may become outdated, resulting in the performance of predictive models degrading, as noted by Ditzler et al. [30]. Minku [128] defines concept drift and chunk-based (CB) learning as follows.

Concept drift: Given at each point in time t , the joint probability distribution (i.e., concept) can be defined for every example as $p_t(a, b) = p_t(b|a)p_t(a)$ where $p(a)$ is the class prior probabilities and $p(b|a)$ is the class conditional

probability. Concept drift occurs at two distinct time-points, t and $(t + \Delta)$, iff $\exists t : D_{(t)}(a, b) \neq D_{(t+\Delta)}(a, b)$.

CB learning: In CB learning, the prediction learner lg is processed for the CB learning, where data $D_t = \{(a_t^{(i)}, b_t^{(i)})\}_{i=1}^{pm_t} \sim_{iid} p_t(a, b); pm_t > 1$ where pm_t is the training data size at time t which is greater than 1 [128]; $(a_t^{(i)}, b_t^{(i)}) \in A \times B$; and $p_t(a, b)$ is a joint probability distribution at time t . At time t , we train the model $f_t : A \times B$ based on the data chunk.

6.3.2 Framework

As shown in Figure 6.2, the framework contains six steps: data pre-processing, instance selection by feature processing, construct cross-version defect prediction models, statistical test calculation, hypothesis test, and analyse model performance.

① *Data pre-processing:* For implication in practice, we formulate the chronological defect data as a cross-version data stream (i.e., windowed stream). The adopted version-based moving-window approach used to build the cross-version classifiers is the most closely related to actual practice, and thus the most practical strategy is to create the window using the method. We collect all the available versions from the sources and restructure them according to chronological order. The windowing technique manages the versions. The old windows' distributions are discarded by following the moving-window approach. The distributions of each version considered as one window. Each version of the software project collectively constitute a window.

② *Instance selection:* This step is for instance selection by applying feature selection techniques. In this step, we adopt correlation and consistency-based feature selection, and AutoSpearman feature selection technique. Details of feature selection techniques are illustrated in the Section 6.4.2.

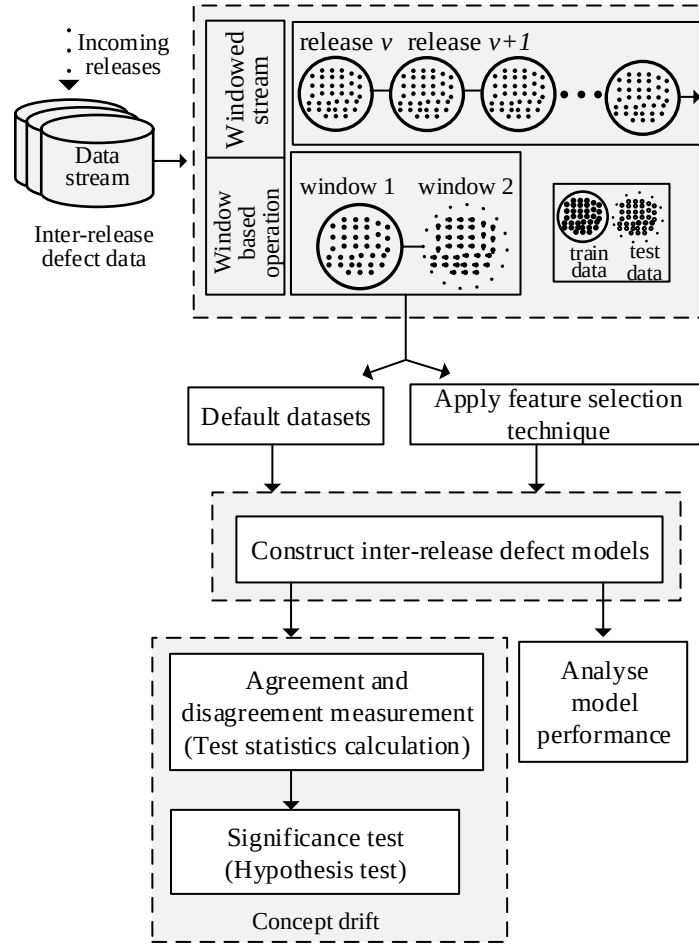


FIGURE 6.2: Framework for cross-version defect prediction using temporal chunk-based learning.

③ *Construct cross-version defect models*: In CVDP, two nearest versions are considered for the defect prediction, bearing in mind that consecutive versions share similar characteristics, which helps to train the model accurately, as suggested by Amasaki [3–5]. Defect prediction models are trained on the earlier version and used to predict the defects of the subsequent version. As discussed above, we utilize the version-based moving window approach to split the version, as prior work applied a similar strategy (i.e., split the version). For instance, for project ant, it contains five versions (1.3 to 1.7). The prediction model is trained on version 1.3 and used to test the model on version 1.4. Afterwards, the prediction model will be trained on the version 1.4

TABLE 6.1: The test statistics of a pair of classifiers at time t and $t + 1$.

Instances	window at time t	window at time $t + 1$
# of correct	$(q_1 + t_1)$	$(q_2 + t_2)$
# of incorrect	$(r_1 + s_1)$	$(r_2 + s_2)$
Total	$(q_1 + t_1 + r_1 + s_1)$	$(q_2 + t_2 + r_2 + s_2)$

and tested on the version 1.5.

④ *Test statistics calculation:* Test statistics are the measurement of pairwise agreement and disagreement (e.g., diversity) measures of two consecutive cross-version defect predictors. Several studies [100, 118] calculated the agreement and disagreement of the classifiers to detect drift. For example, De Lima Cabral et al. [100] calculated the diversity of the classifiers using sliding windows. We adopt a similar strategy to calculate the agreement and disagreement of two consecutive cross-version defective models using a version-based moving window approach. As an example, let $X = x_1, \dots, x_n$ be a distribution and $o_1 = [o_1(x_1), \dots, o_1(x_n)]$ is a binary vector that represents the output of the classifier c . Suppose for a pair of classifiers c_1 and c_2 and n^{ab} is the all possible outcomes of instances $x_j \in X$ for which $c_1(x_i) = a$ and $c_2(x_i) = b$. We calculate the test statistics for consecutive pairs of cross-version defect models. Let R_i and O_i be the distributions obtained from two consecutive cross-version data at time t and $t + 1$. The two feature spaces are transformed into $V_i = (q_1, r_1, s_1, t_1)$ and $V_j = (q_2, r_2, s_2, t_2)$. For t time window, (q_1, t_1) are the correctly classified positive and negative instances, and (s_1, r_1) are the wrongly classified positive and negative instances, respectively. Similarly, for $t + 1$ time window, (q_2, t_2) are the correctly classified positive and negative instances and (s_2, r_2) are the wrongly classified positive and negative instances, respectively. Thereby, the test statistics are calculated. The agreement and disagreement measures are the consistent and inconsistent decisions of the total considered observations. These reflect the

variety of responses of the pair of cross-version predictors that changes at time t and $t + 1$, i.e., probability of diversity between a pair of cross-version learners. Thus, from 2×2 contingency table (Table 6.1), the test statistic of correctly and incorrectly classified instances at t and $t + 1$ time windows (i.e., variety) are calculated where $(q_1 + t_1)$ (i.e., agreement) are the number of correctly classified instances and $(r_1 + s_1)$ (i.e., disagreement) are the number of incorrectly classified instances among the $(q_1 + t_1 + r_1 + s_1)$ total number of instances at t time window, respectively. Furthermore, $(q_2 + t_2)$ (i.e., agreement) are the number of correctly classified instances and $(r_2 + s_2)$ (i.e., disagreement) are the number of incorrectly classified instances among the $(q_2 + t_2 + r_2 + s_2)$ total instances at $t + 1$ time window, respectively.

⑤ *Hypothesis test*: Prior research effort on concept drift detection [100, 116] assumes that a prediction model trained on a time window (past data) would have a statistical difference in the prediction performance of the model of consecutive time window (data of next period) when there is a concept drift. We follow the same hypothesis to detect drifts. If a defect prediction model trained from the previous version's data exhibits a statistically significant difference of prediction performance for identifying defective or non-defective modules of the models of consecutive version's data, then a CD exists. We conduct Fisher's Exact test at a 5% significant level to assess the statistical significance of the changes observed in two consecutive versions data obtained from step 4. As mentioned in the previous chapter, this statistical test has recently been adapted for concept drift detection and recommended by de Lima Cabral et al. [100]. Nishida et al. [135] suggested employing this test when the sample sizes are small. The rationale behind employing this test is that the sample size (i.e., number of modules) of each version is small (see Section 6.4.1). In particular, we conduct the statistical analysis on the test statistics acquired from the two windows.

⑥ *Analyse model performance*: We collect performance report of the models of using all feature sets (default datasets). We report prediction performance values for the models constructed by feature selection techniques for cross-version defect prediction using chunk-based learning.

6.4 Methodology

To answer the RQs and empirically validate the results, we conduct a systematic investigation by selecting appropriate cross-version benchmark datasets (Section 6.4.1), choosing best-performing FS techniques from SDP literature (Section 6.4.2), selecting widely-used learning algorithms (Section 6.4.3), evaluating the results using several performance metrics (Section 6.4.4), applying the robust statistical tests to obtain reliable experimental results (Section 6.4.5), and conducting experimental setup (Section 6.4.6). To gain a comprehensive understanding of the impact of FS techniques for CVDP using a temporal chunk-based learning environment, we conduct several experiments. First, we investigate the impact of each FS technique to answer RQ1. For each FS technique, we calculate the prediction results of each performance measure. We train the CVDP model on the previous version and test the subsequent version within the same software project as a practical scenario. Before preparing the model, we first preprocess the data based on chronology. We split the versions from the main datasets into train and test sets based on the version's order. Thus, the datasets behave as a windowed stream containing v versions (see Figure 6.2). The train and test sets are processed to build the model by following the window-based operation, where each version is considered as a window. We build the prediction models by utilizing the defaults datasets and calculate the model performance. To understand the impact of the FS technique, we then apply the FS techniques on the default datasets. After that, the prediction models are built on the

selected features and calculate the model performance of each performance measure. As mentioned in Section 6.3.1, we consider chunk-based learning where the data of each version is considered as chunk and processed based on the concept of discard-after-learn in the temporal order. This strategy ensures the memory space available for the subsequent data chunk. The prediction model development and evaluation were conducted in R Core Team [162]. To answer RQ2, we conduct a hypothesis test to detect concept drift. We calculate the test statistics of two consecutive CVDP models and compute the diversity of the two consecutive prediction models. We consider Fisher’s Exact test to assess the statistically significant differences of two consecutive time windows that identify the drifts. Section 6.4.6 discusses the motivation behind these RQs and illustrates the experimental setup. Furthermore, Figure 6.2 provides an overview of our empirical study.

6.4.1 Benchmark datasets

We conduct our empirical investigation on a corpus of publicly available datasets containing 36 versions of 10 benchmark open-source software projects provided by Jureczko and Madeyski [83, 117], Jureczko and Spinellis [84]. These cross-version defect datasets have been widely utilized in the studies of CVDP [3, 4, 11, 156, 179, 181, 183, 184]. We collect the corpus from the SEACRAFT repository¹ [164] (which was formerly known as the PROMISE repository [163]). A summary of the datasets, including versions and number of modules, is shown in Table 6.2. Each instance or module of these datasets includes 20 static code metrics with a labeled feature (BUG) listed in Table 6.3. The original datasets contain the number of defects of each module. We convert it for binary classification, where a class is defective if the BUG number is one (1). Otherwise, a class is defined as non-defective by annotating it as zero (0). From the considered datasets, Xu et al. [183] observed that 40% of

¹<https://zenodo.org/communities/seacraft/>

the modules of the subsequent versions were defective, which indicates the distribution differences. Such differences can make the versions drift-prone.

Our selected open-source software projects contain specified version that appears in chronological order. The version date of the versions is collected from version-control repositories, which are used in the study of Bangash et al. [7]. As mentioned in the Section 6.3.1, each version's data appears as a chunk in temporal order, and the learning environment becomes chunk-based learning. Suppose that the ant project has five versions from 1.3 to 1.7 (Table 6.2). It appears in temporal order and creates temporal chunk-based learning environments. Furthermore, the project considered in our empirical study has at least three versions for constructing the scenario of CVDP.

6.4.2 Apply feature selection techniques

In prior studies of SDP, several feature selection (FS) techniques have been adopted to improve the prediction performance. Since it is unrealistic to consider all the utilized FS methods, we select those that have demonstrated better performance than others. In particular, filter-based (FS) techniques are widely adopted for classification models in SDP [46, 93, 146, 180]. We apply three FS techniques in this empirical study. These are - correlation and consistency-based FS techniques and the AutoSpearman FS technique.

Several recent investigations have revealed the impact of FS techniques on SDP [46, 76, 93, 146, 180]. Ghotra et al. [46] conducted a large-scale empirical study by considering 30 FS techniques applied on software defect datasets with 21 classification models. They found that correlation-based FS technique performed based among the others, which is recommended to consider while building classification-based prediction models for SDP. Another empirical investigation containing 32 FS techniques conducted by Xu et al. [180] on software defect datasets. Based on their experiment, they observed that

TABLE 6.2: A statistical summary of selected chronological defect datasets.

Project	version	version Date	#Modules	#Defects	Defects(%)
ant	ant-1.3	12-Aug-2003	125	20	15.90%
	ant-1.4	12-Aug-2003	178	40	22.50%
	ant-1.5	12-Aug-2003	293	32	10.90%
	ant-1.6	18-Dec-2003	351	92	26.10%
	ant-1.7	13-Dec-2006	745	166	22.30%
camel	camel-1.0	19-Jan-2009	339	13	3.80%
	camel-1.2	19-Jan-2009	608	216	35.50%
	camel-1.4	19-Jan-2009	872	145	16.60%
	camel-1.6	17-Feb-2009	965	188	19.50%
poi	poi-1.5	24-Jun-2007	1988	141	59.50%
	poi-2.0	24-Jun-2007	9277	37	11.80%
	poi-2.5	24-Jun-2007	1988	248	64.40%
	poi-3.0	24-Jun-2007	125	281	63.60%
log4j	log4j-1.0	08-Jan-2001	135	34	25.20%
	log4j-1.1	20-May-2001	109	37	33.90%
	log4j-1.2	10-May-2002	205	189	92.20%
xerces	xerces-init	08-Nov-1999	162	77	47.50%
	xerces-1.2	23-Jun-2000	440	71	16.10%
	xerces-1.3	29-Nov-2000	453	69	15.20%
	xerces-1.4	26-Jan-2001	588	437	74.30%
velocity	velocity-1.4	01-Dec-2006	196	147	75.00%
	velocity-1.5	06-Mar-2007	214	142	66.40%
	velocity-1.6	01-Dec-2008	229	78	34.10%
ivy	ivy-1.1	13-Jun-2005	111	63	56.80%
	ivy-1.4	09-Nov-2006	241	16	6.60%
	ivy-2.0	18-Jan-2009	352	40	11.40%
lucene	lucene-2.0	26-May-2006	195	91	46.70%
	lucene-2.2	17-Jun-2007	247	144	58.30%
	lucene-2.4	08-Oct-2008	340	203	59.70%
synapse	synapse-1.0	13-Jun-2007	157	16	10.20%
	synapse-1.1	12-Nov-2007	222	60	27.00%
	synapse-1.2	09-Jun-2008	256	86	33.60%
xalan	xalan-2.4	28-Aug-2002	723	110	15.20%
	xalan-2.5	10-Apr-2003	803	387	48.20%
	xalan-2.6	27-Feb-2004	885	411	46.40%
	xalan-2.7	06-Aug-2005	909	898	98.80%

among the filter-based FS techniques, correlation and consistency-based FS techniques performed best compared to other techniques. The study of Kondo et al. [93] on FS techniques for feature reduction on SDP models demonstrated that correlation and consistency-based FS techniques outperformed other techniques. We give a brief description of these two techniques. Correlation-based FS searches the best subset of features among the considered features. The selected features share the most substantial relationship with the outcome variable and have a low correlation with the other feature sets [53].

TABLE 6.3: List of the static code metrics used in this empirical study [60].

Abbreviation	Description
CK suite (6)	
WMC	Weighted methods per class
DIT	Depth of inheritance tree
NOC	Number of children
CBO	Coupling between objects
RFC	Response for classes
LCOM	Lack of cohesion in methods
Martins metric (2)	
CE	Efferent couplings
CA	Afferent couplings
QMOOM suite (5)	
MOA	Measure of aggregation
CAM	Cohesion among methods
MFA	Measure of functional abstraction
DAM	Data access metric
NPM	Number of public methods
Extended CK suite (4)	
LCOM3	Normalized version of LCOM
IC	Inheritance coupling
AMC	Average method complexity
CBM	Coupling between methods
McCabe's CC (2)	
AVG_CC	Mean values of methods in the same class
MAX_CC	Maximum values of methods in the same class
LOC	Lines of code
BUG	Bugs or no-bug

Consistency-based FS, a deterministic technique, selects the feature sets by using consistency measures. This technique aims at selecting the consistent features whose rate is equal to all the features [26]. We denote Correlation-based FS method as **CFS** and Consistency-based FS method as **CONFS** throughout our experimental analysis. We utilize the implementation of the functions, `cfs` and `consistency`, respectively, provided by the R `FSselector` package [149].

Recently, Jiarpakdee et al. [75] proposed an approach called AutoSpearman for interpreting defect models. They conducted an investigation to assess the impact of FS techniques on the interpretation of SDP models. They considered 11 FS techniques that are commonly used in the studies of SDP

to compare with their proposed one, AutoSpearman, an automated FS approach based on Spearman rank correlation, and variance inflation factor analysis [74]. From their experimental study, they observe that AutoSpearman alleviates the correlated metrics better than other FS techniques [76]. In our empirical study, we adopt this automated FS technique. We use the implementation of AutoSpearman using the AutoSpearman function as provided by the AutoSpearman R package [74]. We refer to this automated FS method as **AUTOS** in this study. To compare the performance of the considered FS method in the scenario of CVDP using temporal chunk-based learning, we consider **all metric sets** as a baseline method that is widely used in the existing studies.

6.4.3 Construct cross-version defect models

There are a plethora of classification methods used in SDP. We adopt a manageable amount of classification techniques for our empirical investigation. Hall et al. [54] show that Random Forest (RF) and Logistic Regression (LR) are the most widely used techniques found in SDP. Naive Bayes is also used for classification in SDP and shows better performance. For example, He et al. [60] conduct an empirical study on 34 versions of defect data sets from the PROMISE repository for cross and within-project defect prediction. They observed that NB was able to show better performance compared to other classifiers. Furthermore, Menzies et al. [121] affirm the effectiveness of NB on NASA defect data sets. For our experimental study, we choose commonly-used five machine learning algorithms to construct the cross-version defect predictors, i.e., Naive Bayes (NB), Random Forest (RF), Decision Tree (DT), K-Nearest Neighbor (KNN), and Logistic Regression (LR).

Since the considered algorithms have settings of configurable parameter, we utilize the implementation of caret parameter optimization prior to

building the models by the `train` function with the option of `nb`, `rf`, `rpart`, `knn`, and `glm`, respectively.

6.4.4 Calculate model performance

TABLE 6.4: Confusion matrix.

	Predicted Positive	Predictive Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

For a comprehensive evaluation of the prediction models, three performance indicators are employed: Recall, pf (probability of false alarm), and AUC. The defect dataset is classified into two groups: non-defective and defective. These performance indicators are calculated from the prediction models' outcomes. When the model is built to predict the defective modules, the prediction comes with the result i.e., either defective or not. To that end, the performance results of the classification models are computed from a confusion matrix (Table 6.4) where the defective modules are considered as positive and non-defective modules as negative. The outcomes are classified as follows:

1. True Positives (TP) refers to correctly classified positive samples as positive samples;
2. True Negatives (TN) refers to correctly classified negative samples as negative samples;
3. False Positives (FP) refers to wrongly classified negative samples as positive samples; and
4. False Negatives (FN) refers to incorrectly classified positive samples as negative samples.

Using the confusion matrix described in Table 6.4, the measures are calculated and defined as follows: Recall is $\frac{TP}{(TP+FN)}$ and pf is $\frac{FP}{(TN+FP)}$. Metrics such as the probability of false alarms (pf) and Recall were recommended as stable metrics for imbalanced datasets by Menzies et al. [121]. Menzies et al. [122] also refuted precision as an unstable measure. We do not consider precision and F-measure in our empirical study. The Recall is the completeness of the prediction performance that achieves a high value of 100% if the false negatives are zero. AUC refers to *Area Under the receiver operator characteristic Curve* that is used to measure the discrimination power of the prediction models [55]. It computes the region under the curve that outlines the true and false-positive rate for x and y-axis, respectively. The value of AUC ranges between 0 and 1. A high Recall and low pf indicate as the best predictor. Higher values of AUC denote better performance.

6.4.5 Statistical tests

The authenticity of empirical results can only be verified using statistical tests. Demsar [28] recommended using a non-parametric statistical test for comparing the prediction results of multiple classifiers obtained from an experiment, which is validated and recommended by Malhotra [120] and Lessmann et al. [98]. Haouari et al. [56] advocated the Friedman test to determine performance differences statistically significant. Later, the pairwise comparisons of the ML techniques are completed the Post-hoc analysis utilizing Nemenyi test [134].

To make a statistical evaluation and determine whether the performance differences of the predictors are statistically significant across the methods over the datasets, we assess the prediction results through the non-parametric statistical test, Friedman test [40], at a 5% significant level. Two types of predictors are developed in our study: (1) a predictor with only the classification

model (i.e., without FS-based predictor) and (2) a predictor is a combination of the prediction model and FS technique (i.e., with FS-based predictor). Using the Friedman test, we find the mean ranks based on the evaluated Recall, pf , and AUC performance measure of the methods (predictors). We determine the null hypothesis (H_0) that all the methods have the same performance results. We reject the null hypothesis (H_0) if the performance results are statistically different at a 5% significant level. The lower position of rank becomes, the better method is. If the prediction performances are statistically significant, we perform the Post-hoc analysis utilizing the Nemenyi test to examine whether the performance differences between any two methods (predictors) are significant, which is widely used in the previous studies of SDP [34, 50, 56, 64, 73, 98, 99, 120, 133, 182, 183]. We present the mean ranks of the methods for each performance measure in Table 6.6, post-hoc analysis using the Nemenyi test presented in Tables 6.7 and ??.

6.4.6 Experimental setup

The main goal of this empirical study is not only focused on applying the FS techniques separately on each version for CVDP, but it also evaluates the robustness of the impact on concept drift. Thus, we conduct an empirical analysis to answer the RQs.

(RQ1) How do the FS techniques impact the performance of CVDP?

Motivation: The findings of prior research have revealed the benefits of adapting FS techniques to predict software defects [46, 76, 147, 180]. For example, Ghotra et al. [46] studied the impact of FS techniques on SDP. Still, the conclusion derived from their study is not compatible with the scenario of CVDP concerning temporal chunk-based learning. At the same time, the

main goal of our empirical study is to investigate whether FS techniques performed separately for each version of the project while creating prediction models that can improve prediction performance. We set out to explore the impact of the best-performing FS techniques retrieved from the literature on the performance of CVDP models through the RQ1.

Approach: To assess whether FS techniques impact CVDP, we compute and analyze the prediction performance of CVDP models that are created using the subsets of metrics generated by the FS techniques and a baseline (without applying FS technique, i.e., all metrics of an cross-version defect dataset). In the CVDP scenario, we utilize a temporal chunk-based learning strategy using the versions' order. Then we train a model using a version and test it with subsequent version defect datasets. We use each metric set generated by a FS technique as input to the studied five defect prediction models. We adopt three performance measures. We consider 10 benchmark open-source software projects containing 36 versions. As mentioned, the experiment is conducted in the cross-version scenario. Therefore, each model has 26 performance values for 26 consecutive pairs for each performance measure. For the baseline, our experiment yields 390 performance measure values ($26 \text{ performance values} \times 5 \text{ classifiers} \times 3 \text{ performance measures}$). For 3 FS techniques, our experiment yields (390×3) 1170 performance measure values. We compare the results of each performance measure with baseline (default datasets) and considered FS techniques. After that, we plot the distribution of performance differences using the boxplots for each performance measure corresponding to each classifier and FS techniques in Figure 6.3.

(RQ2) Which FS method is more robust to concept drift?

Motivation: In data mining and machine learning, concept drift refers to the changes of data distributions over time [41, 94]. Such drift may negatively influence the prediction performance of a model trained on past datasets

when applied to the new datasets, as noted by Dong et al. [31]. Based on our previous works [85, 86], we confirm that the accuracy of prediction models negatively affected due to the changes of defect data over time. Due to the variation of FS for different versions, the FS technique may avoid concept drift. Therefore, in this research question, we investigate the studied benchmark datasets and evaluate which FS techniques are robust to concept drift. In particular, we leverage the statistical test to identify the concept drift.

Approach: Prior research effort on concept drift detection [100, 118] assumes that a prediction model trained on a time window (past data) would have a statistically significant difference in the prediction performance of the model of the consecutive time window (data of next period) when there is a concept drift. We follow the same hypothesis to measure the concept drift. If a defect prediction model trained from the previous version's data exhibits a statistically significant prediction performance difference for identifying defective or non-defective modules of the models of consecutive version data, then a concept drift exists.

In this experimental study, we use the version-based moving window to split the versions, as prior work applied a similar strategy (i.e., split the versions). For example, Xu et al. [181] consider splitting the versions of the defect datasets and tried to mitigate the distribution differences between two consecutive versions. The experimental setup is as follows: (1) We collect all the available versions from the sources and restructure them into a time frame. The windowing technique manages the versions whereas two time period contains two consecutive versions. (2) We train the prediction models using the version's data from the period and test the model using the version's data for the next period. (3) Similar to prior work [100, 118], we calculate the test statistics of two consecutive CVD models. We calculate the diversity of the two prediction models, i.e., agreement (number of correctly

classified instances at two-time windows) and disagreement (number of incorrectly classified instances at two-time windows) between two prediction models. (4) We conduct Fisher’s Exact test at a 5% significant level to assess the statistical significance of the difference based on diversity observed in two consecutive time windows. If a prediction model trained on the previous version’s defect data shows a significant difference on the new version’s data, then a drift exists. This statistical test has recently been adapted for concept drift detection and recommended by de Lima Cabral et al. [100]. Nishida et al. [135] suggested employing this test when the sample sizes are small. The rationale behind employing this test is that the sample size (i.e., number of modules) of each version is small (see Table 6.2). In particular, we conduct the statistical analysis on the test statistics acquired from the two windows. We portray the drift detection results in Figure 6.7 A for default datasets. After that, we conduct the test again on metric sets obtained by performing the FS techniques separately for every version and show the results in Figure 6.7 B.

6.5 Results

In this section, we present the results of our empirical study with respect to the following two RQs.

(RQ1) How do the FS techniques impact the performance of CVDP?

Results: To assess the abilities of FS techniques on CVDP in the temporal learning environment, we consider three performance measures. These are Recall, pf , and AUC. Table 6.5 compile the prediction results in which include max, min, and average performance of each performance measure. The bold font values describe the best values reported upon all the methods.

TABLE 6.5: Summary of prediction results of cross-version defect prediction across the algorithms.

	Recall			pf			AUC		
	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN
NB	0.62	0.92	0.04	0.75	1.00	0.04	0.54	0.78	0.20
RF	0.57	0.94	0.04	0.67	1.00	0.07	0.48	0.77	0.00
DT	0.61	0.94	0.03	0.70	1.00	0.00	0.30	0.73	0.00
KNN	0.55	0.94	0.04	0.66	1.00	0.03	0.39	0.67	0.00
LR	0.57	0.93	0.04	0.68	1.00	0.06	0.46	0.74	0.16
CFS-NB	0.80	1.00	0.11	0.52	0.90	0.03	0.72	0.90	0.54
CFS-RF	0.73	1.00	0.04	0.47	0.99	0.04	0.70	0.89	0.53
CFS-DT	0.71	1.00	0.00	0.44	0.89	0.00	0.61	0.78	0.50
CFS-KNN	0.71	1.00	0.03	0.48	0.98	0.03	0.64	0.88	0.46
CFS-LR	0.72	1.00	0.04	0.46	0.97	0.00	0.70	0.87	0.55
CONFS-NB	0.75	1.00	0.00	0.50	0.94	0.00	0.66	0.87	0.48
CONFS-RF	0.67	1.00	0.03	0.49	0.97	0.03	0.68	0.88	0.54
CONFS-DT	0.70	1.00	0.00	0.42	0.93	0.00	0.61	0.78	0.50
CONFS-KNN	0.68	1.00	0.00	0.48	0.98	0.00	0.65	0.85	0.50
CONFS-LR	0.68	1.00	0.00	0.51	1.00	0.00	0.69	0.87	0.55
AUTOS-NB	0.81	1.00	0.38	0.53	0.91	0.06	0.72	0.89	0.49
AUTOS-RF	0.71	1.00	0.15	0.45	0.99	0.02	0.71	0.90	0.52
AUTOS-DT	0.73	1.00	0.00	0.38	0.92	0.00	0.70	0.90	0.50
AUTOS-KNN	0.68	1.00	0.01	0.44	1.00	0.00	0.65	0.78	0.48
AUTOS-LR	0.72	1.00	0.23	0.51	0.98	0.02	0.70	0.84	0.56

From Table 6.5, AUTOS with NB achieves maximum average Recall, the probability of defective modules that are correctly classified by 81% and outperforms others in the range from 1% to 26%. In terms of Recall values, KNN achieves the lowest mean values than the FS-based models by 55%. Among all the models, the FS-based models reach a perfect Recall score of 100%, resulting in a high probability for detecting defective modules correctly; for example, the FS-based models classify all defect-prone modules when the models are trained with camel-1.0 and tested with the consecutive version camel-1.2. Overall, the results demonstrate that FS-based models are competitive for correctly classifying the defective modules in the temporal learning environment and produce better CVDP.

In terms of probability of false alarm pf , the prediction models that do not consider FS achieves high pf . Among all of the methods, the models trained with NB achieve maximum pf by 75%. In contrast, among all the methods, those methods that achieve 0 pf score mean that the models can identify all

defective software modules, resulting in less testing effort. Among all of the methods, ATUOS-DT has the lowest average pf by 38%, meaning that it detects the minimum amount of software modules that are not defective. In practice, we want to achieve low pf with high Recall that reduces the high testing effort for a critical software system [21, 56, 73]. Especially for a critical software system, where each version changes over time, the proper identification (prediction) will be more effective in practice in such a temporal chunk-based learning environment.

The outcomes of the AUC measure range from 0.30 to 0.72 score, with the worst average prediction performance for DT and the best one for AUTOS-NB, and CFS-NB, followed by AUTOS-NB with an average highest Recall score of 0.81. This means that AUTOS-NB has the ability to detect defective modules for CVDP and reduces the testing effort. The FS-based models' outcome ranges from 0.61 to 0.72 AUC score. Overall, CFS-NB and AUTOS-NB achieve a maximum average AUC of 72% and outperform others in the range from 2% to 42%. The best value of AUC is recorded by 90% for CFS-NB, ATUOS-RF, and AUTOS-DT, respectively. However, in Table 6.5, we observe that RF, DT, and KNN gain the AUC values of 0, meaning that the models whose predictions achieve maximum errors (100% wrong prediction). From these empirical results, we observe that the choice of classifier employed with FS techniques is as important as the temporal defect data used to build the CVDP models in such a realistic environment.

Figure 6.3 presents the performance difference values between the results with and without FS techniques for each of the three performance measures. We present our experimental findings that (1) are not impacted the prediction performance and (2) impacted the performance by utilizing the FS techniques. In the figure, the distributions centered at zero indicate that the performance measures are not impacted negatively or positively by FS techniques for CVDP. Figure 6.3 shows that when applying FS techniques,

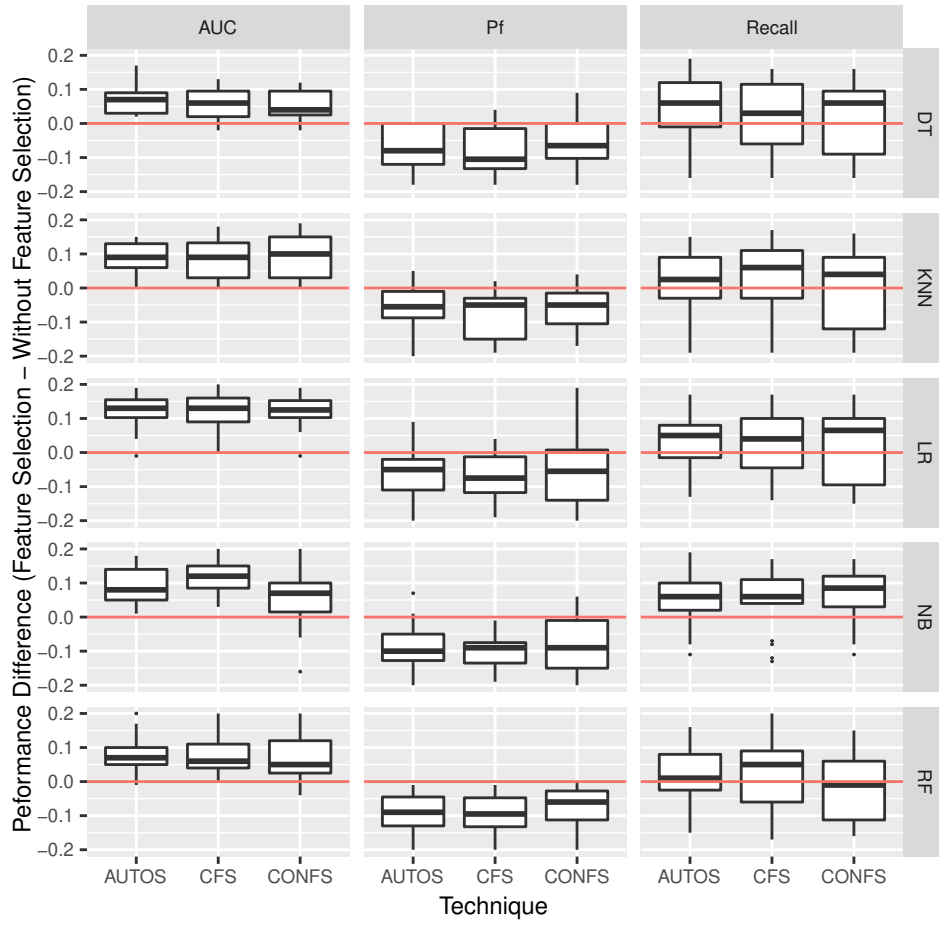


FIGURE 6.3: The performance difference when employing FS techniques to cross-version defect prediction models. A red line indicates no performance improvement.

we find that the performance difference of AUC measure varies from -16% to 90% (i.e., min-max). We observe that the performance distribution of the AUC measure for 75% CVDP models varies from 8% to 37% (i.e., the values of 1st-3rd quantiles). This result indicates that FS techniques tend to have a positive impact on AUC when they are employed to CVDP models. In terms of Recall values, the performance distributions that are centered at zero indicate that the Recall is not impacted positively or negatively by FS techniques for CVDP models. As mentioned above, FS-based CVDP models are effective for correctly classifying the defective modules in this temporal learning environment. We observe that the performance distribution of the Recall measure for 75% CVDP models varies from -5% to 97% (i.e., min-max). We also observe

that the performance distribution of the Recall measure for 75% CVDP models varies from -3% to 22% (i.e., the values of 1st-3rd quantiles). Overall, from the Figure 6.3, it can be seen that the FS techniques have a positive impact on the performance for correctly classifying the cross-version defective modules. pf is measured by the total number of non-defective modules predicted as defective. Interestingly, AUTOS, CFS, and CONFIS FS techniques decrease pf and increase Recall, which implies the positive impact of FS techniques in classifying the cross-version defective modules.

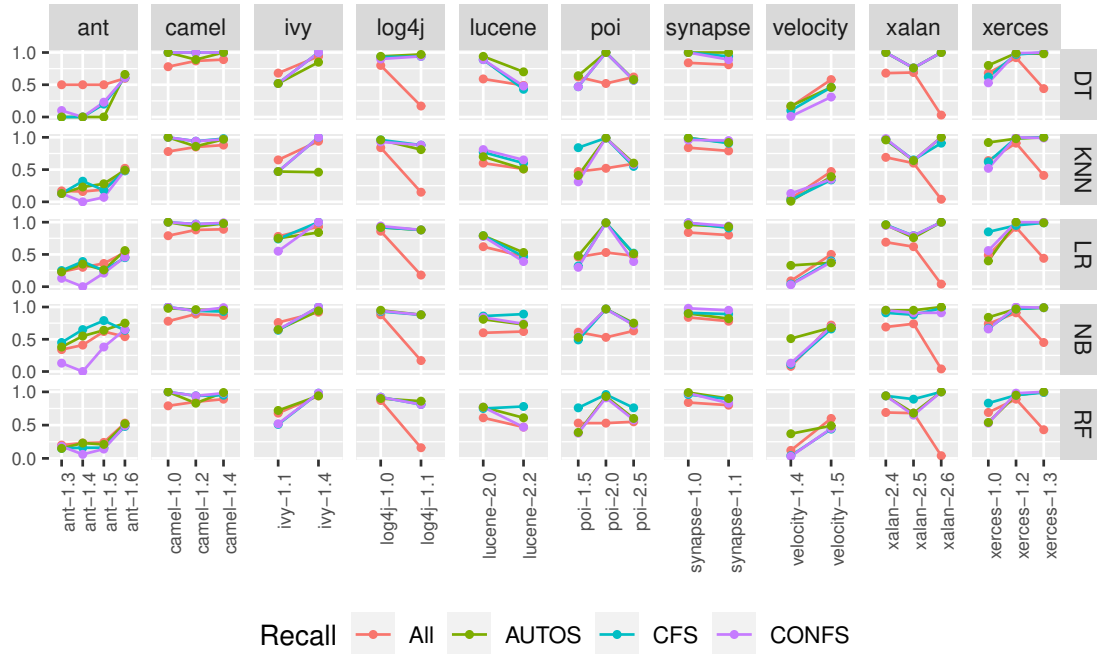


FIGURE 6.4: CVDP models performance across the versions of the considered datasets for Recall.

The prediction performance values across the versions of the considered datasets are depicted in the Figures 6.4 to 6.6. We observe that the Recall and AUC scores do not always increase over the versions with time in the temporal learning environment (Figure 6.4 and 6.5), which means that the latest version does not always give the height values of Recall and AUC. In some

6.5. Results

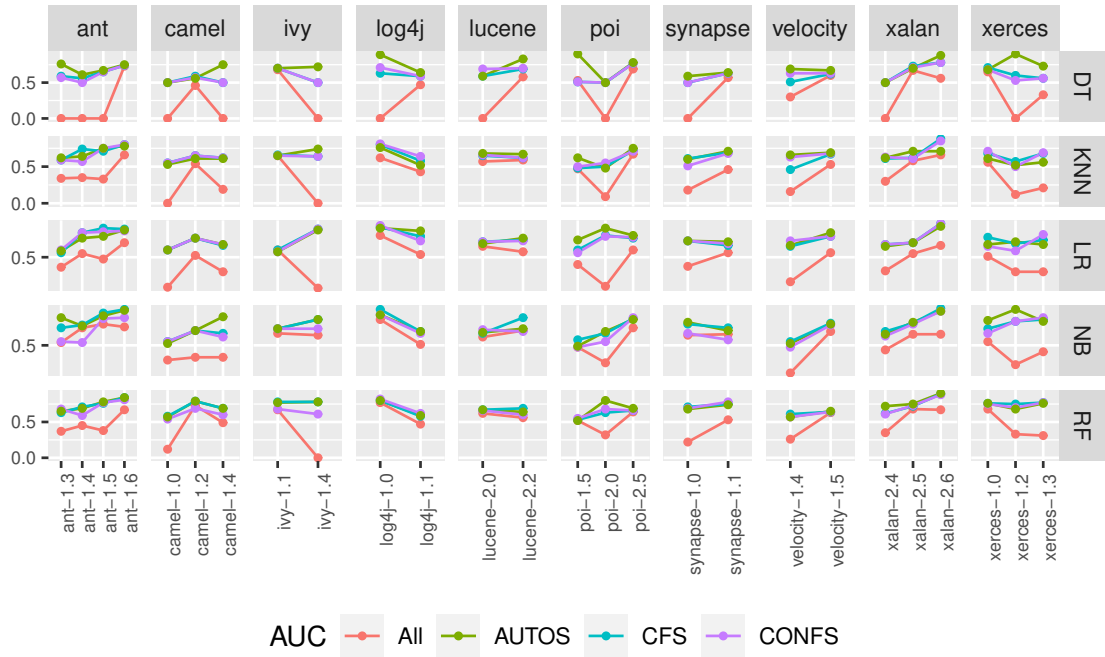


FIGURE 6.5: CVDP models performance across the versions of the considered datasets for AUC.

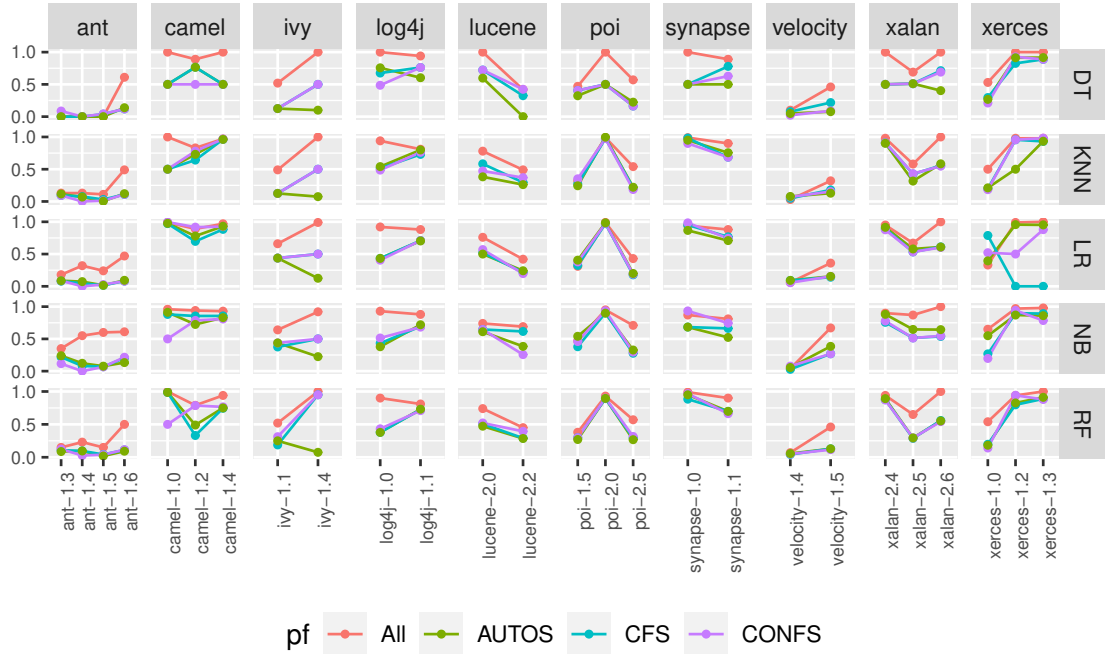


FIGURE 6.6: CVDP models performance across the versions of the considered datasets for pf .

cases, the initial versions are more suitable for correctly classifying the defective modules. For example, as a training set for an CVDP model, synapse-1.0 is able to classify the defective modules correctly (i.e., high Recall) than

synapse-1.1. For pf values, initial versions across the considered datasets are more effective than have low rate of pf (Figure 6.6). We also see that the CVDP models remain competitive throughout the versions in the temporal chunk-based learning environment. Another observation can be drawn from the Figures 6.4 to 6.6: the CVDP models trained on the most recent versions do not always yield the best cross-version defect predictors. This observation agrees with the results of Harman et al. [58]. They found that the latest version is not always the best predictor when considered SDP as a temporal learning problem suggested by Harman et al. [58], dynamic adaptive prediction systems [57] could be employed to recognize the best model. Point to be noted that the ability to underline the best models among the CVDP models is out of the scope of our study.

TABLE 6.6: The Friedman means ranks of the with and without FS-based predictive models based on pf , Recall, AUC-score. The lower position of rank becomes, the better model is. ALL refers to all feature sets are considered to develop the models.

	ALL					CFS					CONFS					AUTOS				
	NB	RF	DT	LR	KNN	NB	RF	DT	KNN	LR	NB	RF	DT	KNN	LR	NB	RF	DT	KNN	LR
Recall	8.9	7.1	8.5	6.6	5.6	13.3	10.4	12.6	10.8	11.7	13.8	4.8	11.8	10.2	10.6	14.0	10.5	13.2	10.0	11.6
pf	16.4	16.5	16.9	16.4	16.3	9.7	7.3	8.4	9.1	8.7	9.7	7.8	7.0	8.9	9.2	10.7	6.7	5.4	8.8	10.1
AUC	5.5	5.2	3.6	3.3	2.3	16.2	14.8	8.7	10.3	14.3	11.7	12.9	9.2	10.8	13.9	15.1	15.0	12.9	10.8	13.5

To achieve reliability and statistical validation, Demsar [28] alluded not to depend only on mean values of performance measures while compared with multiple classifiers over datasets and recommended considering an additional statistical test to verify the comparison of the prediction results. In this section, we show the results for pf and Recall. In particular, we assess the prediction results using the non-parametric Friedman statistical test amongst the methods. The null hypothesis (H_0) is that all the methods have the same performance. We reject H_0 if and only if the test is statistically significant. Here, we set the p -value to reject H_0 at 5% significant level (p -value<0.05). We compare twenty classification methods over the defect datasets in our selection. Therefore, the degree of freedom is 19. The computed Friedman test

6.5. Results

statistics for the Recall are 84.243, 202.533 for pf , finally 272.565 for AUC. The p -value for recall, pf , and AUC measures are 3.42E-10, 1.07E-32, and 8.07E-47, respectively.

TABLE 6.7: The result of the post-hoc analysis. 0 refers to there is no statistical difference between the pair of classifiers otherwise, there is a statistical difference in terms of P for pf and R for Recall. ALL refers to all feature sets considered to develop the predictive models.

	ALL					CFS					CONFS					AUTOS			
	NB	RF	DT	KNN	LR	NB	RF	DT	KNN	LR	NB	RF	DT	KNN	LR	NB	RF	DT	KNN
RF	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DT	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
KNN	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LR	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CFS-NB	PRA	PRA	PRA	PRA	PRA	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CFS-RF	PRA	PRA	PRA	PRA	PRA	0	-	-	-	-	-	-	-	-	-	-	-	-	-
CFS-DT	PR	PR	PR	PRA	PR	A	A	-	-	-	-	-	-	-	-	-	-	-	-
CFS-KNN	PR	PR	PRA	PRA	PRA	0	0	0	-	-	-	-	-	-	-	-	-	-	-
CFS-LR	PRA	PRA	PRA	PRA	PRA	0	0	0	0	-	-	-	-	-	-	-	-	-	-
CONFS-NB	PRA	PRA	PRA	PRA	PRA	0	0	0	0	0	-	-	-	-	-	-	-	-	-
CONFS-RF	PRA	PRA	PRA	PRA	PRA	0	0	0	0	0	0	-	-	-	-	-	-	-	-
CONFS-DT	PR	PR	PR	PRA	PR	A	0	0	0	0	0	0	-	-	-	-	-	-	-
CONFS-KNN	PR	PR	PRA	PRA	PRA	0	0	0	0	0	0	0	0	-	-	-	-	-	-
CONFS-LR	PRA	PRA	PRA	PRA	PRA	0	0	A	0	0	0	0	0	0	-	-	-	-	-
AUTOS-NB	A	A	PRA	A	A	0	0	A	0	0	0	0	A	0	0	-	-	-	-
AUTOS-RF	PRA	PRA	PRA	PRA	PRA	0	0	0	0	0	0	0	0	0	0	0	-	-	-
AUTOS-DT	PRA	PRA	PRA	PRA	PRA	0	0	0	0	0	0	0	0	0	0	0	0	-	-
AUTOS-KNN	PR	PR	PRA	PRA	PRA	0	0	0	0	0	0	0	0	0	0	0	0	0	-
AUTOS-LR	PRA	PRA	PRA	PRA	PRA	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After observing the statistical test results of the Friedman test, we reject the null hypothesis, meaning that there is a statistically significant difference between the compared methods. Therefore, we can now adopt a Post-hoc test using Nemenyi's Post-hoc test to verify if each pair of methods or classifiers is different statistically. The mean ranks obtained from the Friedman test are shown in Table 6.6, regarding the Post-hoc test results displayed in Table 6.7. In Table 6.7, 0 refers to there is no statistical difference between the pair of classifiers otherwise, there is a statistical difference in terms of P for pf and R for Recall. From the pairwise comparison, 71 is found to be statistically significant for pf and Recall.

Results drawn from Tables 6.6 and 6.7, in terms of Recall, CONFS-RF achieved the best average rank among the classifiers, followed by KNN with an average rank of 5.6. The Post-hoc test results manifest that there is a statistically significant difference among the without FS-based classifiers. All the FS-based classifiers are not statistically significant from each other, which makes it hard to determine which FS-based classifier is best based on Recall. In terms of pf , AUTOS-DT achieved the best average rank of 5.4 among the classifiers, followed by AUTOS-RF with an average rank of 6.7. From the Post-hoc results, they are the statistically significant difference among the without FS-based classifiers. When it comes to FS-based classifiers, they are not statistically significant from each other. On the other hand, without FS-based classifiers ranked the best average ranks among all the classifiers when it comes to AUC (see Table 6.7). From the results, KNN achieved the best average rank with an average rank of 2.3, followed by LR with an average rank of 3.3. But, they are not statistically significant based on the Post-hoc test results. Overall, we observe that the performance differences between with and without FS-based classifiers are statistically significant. However, the performance differences are not statistically significant among the FS-based classifiers, confirming the observation that FS-based classifiers are suitable

for CVDP when the learning environment is temporal chunk-based learning.

Summary

When there is a need to identify (predict) the most defect-prone modules for a critical software system, we require to choose the prediction models that achieve high Recall [21, 56, 73]. Especially for CVDP, where the software system becomes more critical by changing its functions in each version, we need to choose the models with high Recall scores. When we consider exploring the impact of FS-based classification models in comparison without FS-based classification models for a chunk-based temporal learning environment in an CVDP scenario, the FS-based prediction models surpassed all the without FS-based models. Our experiment shows that the AUTOS-NB achieves the highest average Recall, followed by CFS-NB with an average score of 0.80. They also achieve the highest average AUC values with low *pf* scores. From the Post-hoc analysis, the performance differences are statistically significant compared with the without FS-based classification models. We observe that the CVDP models trained on the most recent versions do not always yield the best cross-version defect predictors (Figures 6.4 to 6.6).

(RQ2) Which FS method is more robust to concept drift?

Results: Figure 6.7 represents the status of concept drift for each version of the defect datasets. Figure 6.7 A depicts the detection of drift while considered all feature sets, and Figure B exhibits drifts while employed FS techniques. The various number of drift identified by the prediction models may be due to the different models have different sensitivity to the evolution of the cross-version defect data. For example, the NB and RF models are probably more sensitive to the specific pattern in each version of the cross-version

6.5. Results

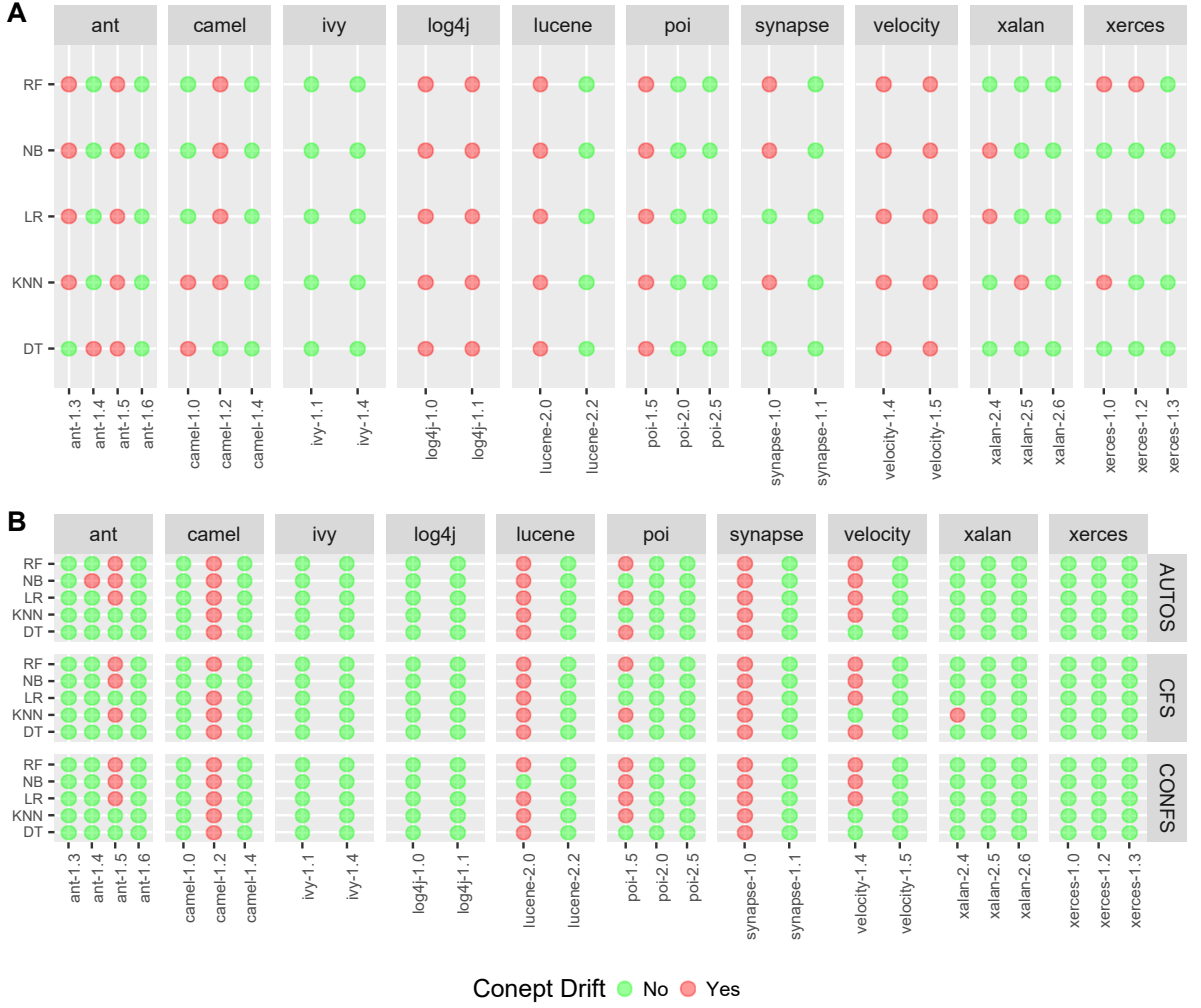


FIGURE 6.7: Concept drift detection in the studied benchmark datasets. Figure A shows the detection of drift while considered all metric sets (all features) and Figure B exhibits the drift detection while considered AUTOS, CFS, and CONFS FS techniques. A red symbol indicate a drift between two consecutive versions of the project ($p\text{-value} < 0.05$ in Fisher’s Exact Test); while a green symbol indicates otherwise ($p\text{-value} \geq 0.05$ in Fisher’s Exact Test).

defect data. We observe the maximum drift while considered the full feature sets (i.e., default data) with KNN by 50%. When FS techniques are applied to each version, the considered techniques help eliminate drift from the datasets. Figure 6.8 shows the drift detection in percentage when FS techniques are applied to the datasets. We observe that CONFS with DT is the robust technique to concept drift that detects only 12% drift among the considered datasets. We also observe that while applying the CFS technique with

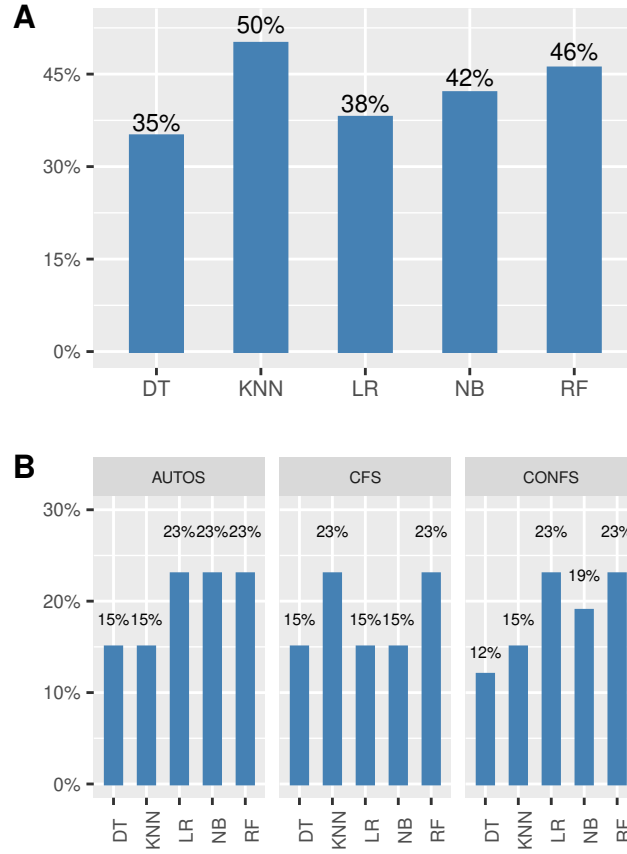


FIGURE 6.8: The percentage of concept drift identification when applying FS techniques. Figure A shows the detection of drift in percentage while considered all metric sets and Figure B exhibits the drift detection in percentage while considered AUTOS, CFS, and CONFS. Here, a low value represents the best score.

DT, it achieves noteworthy improvement on average for Recall 71% and for AUC 61%. By considering the CVDP as temporal chunk-based learning, we observe that while the FS techniques are applied separately on each version of the project, it eliminated the drift from all of the versions of the ivy, log4j, and xerces projects, whereas most of the versions contain drift in the default datasets. This observation guides us to recommend practitioners to consider the FS technique as an *attention technique* for concept drift adaptation [42] in the defect datasets that arrive in temporal order. Overall, the findings suggest that FS techniques play an important role in concept drift reduction and improve the performance of CVDP models. Note that elimination of concept

drift entirely in the defect datasets is beyond of this empirical study.

Summary

From the experiment, we observe that concept drift exists in the cross-version defect datasets, which can be described by the fact that the relationship between the features in the datasets changes over time. We find that almost 50% of the cross-version defect datasets are drift-prone while employed the classifiers only. The FS techniques help mitigate the drift up to 88% and improve the CVDP scenario's prediction performance. Researchers and practitioners should consider concept drift for CVDP while learning environment as chunk-based temporal learning. For the concept drift adaptation process [42], the FS technique could be used as an attention technique.

6.6 Discussion

In this section, we first discuss results obtained from the RQ2 related to concept drift and then present a comparative discussion to verify the discriminating capacity of the FS techniques with the state-of-the-art methods for the CVDP performance concerned with RQ1.

6.6.1 Is continuously retraining a model actionable enough?

Prior research effort affirms that concept drift exists in the defect data and deteriorates the performance of prediction models, i.e., the models trained on past data may become obsolete over time [35, 86]. To eliminate concept drift, the most common remedy in such a situation is to update the existing model [42]. However, from the RQ2, the results obtained from our empirical study are in disagreement with the common findings from data mining and machine learning, i.e., continuously retraining the model may not help

reduce concept drifts and improve the prediction performance [42]. We elab-

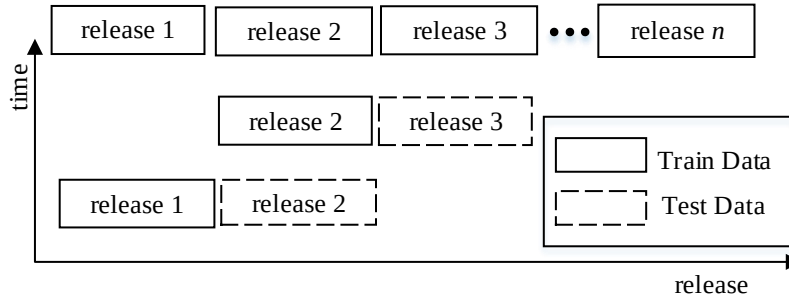


FIGURE 6.9: CVDP-based moving window, where window size refer to one version's defect data.

orate on how continuously a model is updated and how an FS technique helps eliminate the drift while updating the model and calibrate the model to be more actionable using the version-based moving window strategy. This strategy can be called 'learning based on project chronology.' In software effort estimation (SEE), the use of project chronology has demonstrated as a valuable method for improving prediction performance [104–109, 127, 129, 130]. In SDP, Harman et al. [58] conducted a study on 8 Hadoop versions where chronology played an important role in predicting defect over time. Like Harman et al. [58] work, in our empirical study, a model is retrained using the next version's defect data. As mentioned above, a version-based moving window technique is adopted to update the model (see Figure 6.9). We do not use all the historical data of previous versions to retrain the model. Furthermore, updating the prediction models using all the data of previous versions may not lead to better prediction performance [102]. Please note that we split the versions from the main datasets into train and test sets based on the version's order (i.e., version-based splitting). In Figure 6.7 A, we exhibit the drifts detection for each version of the datasets. Figure 6.7 B shows the drift detection when we applied the considered FS techniques during the model retraining process. From the experiment, we observe that adopting

FS techniques during retraining the models helps reduce concept drifts and improve the prediction performance. We also notice that detection of drift in software cross-version defect datasets is *model-aware*, the percentage of drift detection varies from model to model. Figure 6.8 describes the drift detection in percentages per classifiers, i.e., sensitive to the models. Even if we adopt the CFS, CONFS, and AUTOS FS techniques in the drift detection process, it shows the different rate of drift detection percentages. Accordingly, we may conclude that drift detection in defect datasets is model-aware, and model-awareness may reduce the sensitivity of drift detection and improve prediction performance. Based on our experiment results, we suggest that software engineering researchers and practitioners should consider FS techniques if they wish to eliminate drift from the defect datasets. They could also consider adding FS techniques as an attention technique in concept drift adaptation.

6.6.2 General discussion

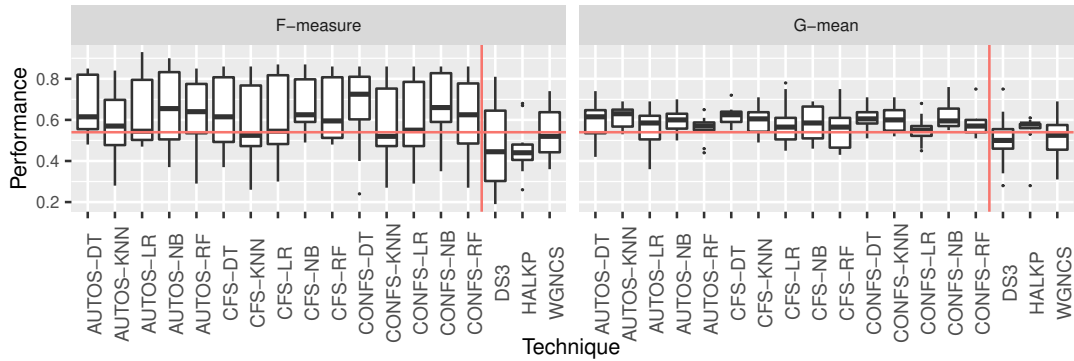


FIGURE 6.10: Comparison of FS-based CVDP classification models with DS3, HALKP, and WGNCS state-of-the-art methods using F-measure and G-mean evaluation indicators. Here, horizontal red lines indicate the mean value obtained from the state-of-the-art methods. The vertical red lines separate the FS-based methods from the state-of-the-art methods.

To verify the discriminating capacity of the considered FS techniques in

the temporal chunk-based learning environment, we compare the CVDP performance using F-measure and G-mean. For comparison, we assess three state-of-the-art methods obtained from the CVDP studies, in which two sparse subset selection methods, DS3 [181] and HALKP [179], and one hybrid method, WGNCS [189], considered. Briefly, Xu et al. [181] proposed a dissimilarity-based sparse subset selection (DS3) method to enhance the CVDP performance. Based on their experiments, DS3 outperformed two baseline methods, Turhan Filter (TF) [169] and Peter Filter (PF) [141]. In [179], Xu et al. proposed hybrid active learning and kernel PCA (HALKP) method that selects representative feature sets to enhance CVDP performance. HALKP outperformed its five downgraded variations. Zhang et al. [189] proposed WGNCS (wasserstein GAN with gradient penalty and CNN-SVM), a deep learning-based hybrid method that outperformed seven baselines CVDP models. We report the results using the box-plots of the F-measure and G-mean performance indicators for CVDP as depicted in Figure 6.10. To cope with chunk-based learning, we consider the average value of each inter-version pair for each project as the indicator value of the project. For instance, camel project contains three inter-version pairs (camel-1.0 $\rightarrow$ camel-1.2, camel-1.2 $\rightarrow$ camel-1.4, camel-1.4 $\rightarrow$ camel-1.6) (see Table 6.2). For the F-measure indicator value of the camel project, we take the average value. Thereby, we report all of the average values of all projects. From the F-measure and G-mean, we observe that most of the FS-based methods surpassed all the state-of-the-art methods, as depicted in Figure 6.10. Compared with the state-of-the-art methods, CFS with NB and CONFS with NB achieve average F-measure by 67% and outperform others in the range from 1% to 21% on the evaluated software projects. Among the methods, HALKP attains the lowest average F-measure by 46%. In Figure 6.10, we draw a horizontal red line by the highest average F-measure value of 0.537 obtained by the WGNCS state-of-the-art method. We notice that FS-based methods performed best among the state-of-the-art

methods. In terms of the G-mean, CFS with DT and CONFS with NB gain an average value of 62% and outperform others in the range from 1% to 12%. On average, we observe a similar G-mean of HALKP, CFS with RF, and CONFS with LR. In this figure, the average G-mean of 0.546 is obtained by HALKP outlined by a horizontal red line. Most prediction performance of the FS-based CVDP models performed better than the considered state-of-the-art methods, as shown by the red line in Figure 6.10. Furthermore, RF, LR, and NB were used as baseline methods in the studies of SDP (e.g., [133, 187]). In RQ1, these baseline methods are compared with the FS-based methods that performed best in the CVDP chunk-based learning environment. Results computed for F-measure and G-mean are made available in additional material [87]. From the results of RQs, we conclude that FS techniques do improve the performance (Recall, *pf*, and AUC) of CVDP. The performance differences between the results with and without FS techniques are statistically significant among the prediction models. By conducting an empirical analysis with acceptable performance measures and statistical tests, we present the overall findings.

Findings In a temporal learning environment where the data appears as chunks, FS techniques do improve for correctly classifying the defective modules by achieving high Recall scores. The Post-hoc analysis exhibits that the performance differences are statistically significant while compared the prediction results with FS and without FS-based models in the temporal learning environment. The recently developed FS technique AUTOS validated with software defect data performs best to identify (predict) the most defect-prone modules for the considered critical software systems. From the results, we observe that the CVDP models trained on the most recent versions do not always yield the best predictors. Even the CVDP models are more competitive throughout the versions. In some cases, the prediction results obtained by using first-version data are more suitable. These observations

agree with the findings of Harman et al. [58], suggested to adaptive prediction systems to find and utilize the best predictors in such temporal learning environment. Furthermore, concept drift is a widespread problem in such a learning environment. The use of FS techniques could eliminate drift and its associate effects from the temporal learning environment.

6.7 Threats to validity

In our comprehensive and systematic empirical study, we consider the benchmark projects that maintain the temporal order of the software development cycle. We conduct our experiment on 36 versions of 10 benchmark projects. In addition, we consider only static code metrics. We acknowledge that our experimental results may not generalize for other metrics. Other researchers could have different choices to conduct the experiments in CVDP. However, the considered benchmark projects were adopted by prior CVDP studies where static code metrics performed well [179, 181, 183]. Besides, Basili et al. [9] affirmed that it is challenging to gain a comprehensive judgment from an investigation in software engineering because the experimental process requires a vast number of relevant context variables. As a result, the conclusions may not generalize beyond the experimental environment and datasets. In addition, we consider three best-performing FS techniques in temporal chunk-based learning. We carefully selected those that performed best for SDP [46, 76, 93, 146, 180]. The impacts of other FS techniques deserve further investigation. Five commonly used classifiers are used to build cross-version prediction models. There are other classifiers that may be sensitive to concept drift and performance results. We intend to extend our experiment in the future. For the performance measures, we consider three performance measures, one threshold-independent measure (i.e., AUC) and two threshold-dependent measures (i.e., Recall and pf) that are widely adopted

as valid measures in the study of SDP. However, the choice of measures could influence the results. In the future, we plan to adopt more measures to generalize our experimental results.

6.8 Chapter Summary

This study focuses on the empirical assessment of the effectiveness of feature selection (FS) methods for cross-version defect prediction (CVDP). The impact of FS techniques on CVDP has not been investigated in detail, considering temporal chunk-based learning where the relation between data variables changes (i.e., concept drift) in the inter-version defect datasets. Additionally, the effect of concept drift in CVDP is unknown. This empirical study employs three best-performing FS techniques obtained from software defect prediction (SDP) literature: correlation-based (CFS) and consistency-based FS (CONFS) and AutoSpearman (AUTOS) FS methods. The study utilizes five machine learning (ML) classifiers, namely Naive Bayes (NB), Random Forest (RF), Decision Tree (DT), K-Nearest Neighbor (KNN), and Logistic Regression (LR). The primary analysis was conducted on the CVDP performance results using three recommended performance measures, pf , Recall, and AUC. Additionally, three state-of-the-art methods (DS3, HALKP, and WGNCS) were adopted to compare the prediction results obtained from the FS-based CVDP models.

We conducted a comprehensive empirical study on 36 versions, distributed in 10 benchmark open-source software projects, and used statistical tests to obtain a more reliable conclusion using a non-parametric Friedman test and the Nemenyi test to compare the prediction results. The CVDP performances of the models that use no FS techniques were compared with the prediction performances of the models, which utilized FS techniques using temporal

chunk-based learning. We assessed the robustness of the FS methods to concept drift in the temporal chunk-based learning environment. The results obtained from the empirical study are summarized as follows:

- From our empirical results, FS techniques significantly improved the CVDP performance (AUC, Recall, and pf) of the CVDP models across all the considered datasets. The CVDP predictor, trained with AUTO FS technique with NB classifier, achieves the highest average Recall, followed by the prediction model, CFS FS technique with NB, with an average score of 0.80. They also achieve the highest average AUC values with low pf scores. For a software system that experiences versions due to refactoring and organization demands, a prediction model with high Recall and low pf is recommended [21, 56, 73]. For such a scenario to predict defects, FS-based prediction models are suitable for better prediction performance in the temporal chunk-based learning environment.
- Assessing the prediction results statistically, the non-parametric Friedman test and the post-hoc Nemenyi test confirm that the performance differences are statistically significant between the prediction results with and without FS-based CVDP models.
- In the temporal chunk-based learning environment, the CVDP models are build based on the version-based moving window and discard-after-learn concept. For this reason, we are able to observe the prediction results for each pair. From the results, we observe that the CVDP models trained on the most recent versions do not always yield the best inter-version defect predictors. This result agrees with the observation of Harman et al. [58]. However, finding the best models in the CVDP scenario is beyond this empirical study.

- Compared with the state-of-the-art methods, CFS with NB and CONFS with NB achieve average F-measure by 67% and outperform others in the range from 1% to 21% on the evaluated software projects. Among the methods, HALKP attains the lowest average F-measure by 46%. In terms of the G-mean, CFS with DT and CONFS with NB gain an average value of 62% and outperform others in the range from 1% to 12%. On average, we observe a similar G-mean of HALKP, CFS with RF, and CONFS with LR. The most prediction performance of the FS-based CVDP models performed better than the considered state-of-the-art methods.
- The result obtained from our empirical study regarding concept drift elimination is in disagreement with the common finding, retaining the prediction model when drift occurred [42]. We observe that continuously retraining the model may not help reduce concept drift and improve the prediction performance. Furthermore, we notice that different drift rates are identified for the considered classifiers, which indicates the model-awareness to concept drift. We recommend adopting the FS technique as an attention technique while retraining the CVDP models to eliminate concept drifts and improve prediction performance in a chunk-based temporal learning environment for software engineering researchers and practitioners.

Concluding Remarks

The empirical results indicate that there were statistical differences between the prediction results with and without feature selection techniques when evaluated with the AUC, Recall, and *pf*. CVDP models trained on the most recent versions are not always the best cross-version defect predictors. The AutoSpearman outperformed among the studied feature selection techniques.

When the prediction models were updated by selecting the features from each version, the feature selection techniques could help reduce drift from the defect datasets.

Further extensions of our work involve other open-source software projects with more versions, utilizing the results obtained from this empirical study. We also intend to assess the prediction performance of the cross-project defect prediction scenario. By the experimental evaluation of Artificial Immune Systems conducted by Haouari et al. [56], we plan to investigate such systems for concept drift detection and elimination.

Chapter 7

Conclusion

7.1 Introduction

Due to the rapid development of the software industry, the prevalence of software systems has led to massive data volumes. Software system experiences several releases after a stable period that appear in a temporal order. It becomes more complex for a long-running software system due to the frequent changes made in each release, which leads to changes in data distribution among the releases. Software defect datasets generated in such an environment are typically characterized by non-stationary data distributions. The effects of such data distributions are detrimental. If the relationship between variables of underlying data distribution changes (i.e., *concept drift*) due to unforeseeable reasons after a stable period, the prediction model would not be reliable. However, very little is known about the impact of concept drift for achieving the better performance of the prediction models in a more practical scenario. The researchers who studied software defect prediction (SDP) by considering long-running critical software systems seem reluctant to consider the datasets as non-stationary and require further investigation by considering the temporal order of the datasets. Prior studies of SDP demonstrate the impact of class rebalancing techniques for improving software defect prediction models. Since class rebalancing techniques try to produce a similar representation of classes, it may eliminate drift and impact

the performance of classification-based SDP models. There are still inconclusive conclusions regarding the impact of class rebalancing techniques regarding concept drift for defect prediction performance, which has not been investigated in detail. This thesis aims to examine class rebalancing techniques and assess the impact of concept drift on SDP performances. Furthermore, we propose a drift detection framework to detect concept drift in the chronological defect datasets. Moreover, this thesis examines the feature selection techniques that could avoid concept drift due to the variation of selecting features in each version of the software project. The main research findings and implications of this thesis are discussed in detail in the sections below.

7.2 Findings and Implications

By conducting extensive empirical experiments and robust statistical tests, we discuss the overall findings from our analysis.

Finding 1: Our experiment results indicate that concept drift occurs in software defect datasets and its existence subsequently degrades the performance of prediction models.

Implications: The identified drift in the chronological datasets would help the SQA teams to re-train the model to get better prediction performance. Therefore, we suggest considering concept drift for improving the performance of classification-based defect prediction models.

Finding 2: *If the error-rate exponentially increases, it causes drift, resulting in prediction performance deterioration.* The proposed drift detection method potentially identified drifts where the prediction accuracy varies in the versions during the learning process by triggering drift, warning, and control flags. By adopting fisher's based statistical test, we observe that the identified drifts are statistically significant.

Implications: We inspire software engineering researchers and practitioners to use the proposed drift detection technique to detect drift in the chronological defect datasets. Further, the drift 's are critical to classification problem due to inequality in the decision boundary, thereby influences the classification performance. By identifying drifts, the classification performance could be improved. The exponential increases of error-rate cause drifts in the versions of software defect datasets and consequently exhibit poor prediction performance.

Finding 3: *Concept drift detection is model-aware for software defect datasets.* We observe from the experiment that the classifiers' choice is too sensitive to the drift of the chronological defect datasets. NB classifier achieves the best results without applying any class rebalancing techniques. We find that 50% of datasets are drift-prone when KNN classifier is applied, which is the maximum. DT achieves the minimum score of drift-proneness, i.e., 35%.

Implications: This observation will help the practitioners to produce an effective drift adaptation technique for detecting defects in the cross-version defect prediction scenario. Furthermore, this observation agrees with the conclusion of Haug et al. [59] that concept drift detection is model-dependent. The instability of the concept drift detection rate among the classification models provides practical implications. In such a realistic CVDP scenario where false alarm rates are expensive, model-awareness reduces the sensitivity of concept drift from chronological defect datasets.

Finding 4: *Update the model is not actionable.* Our experiment reveals that updating the model does not always improve the prediction performance and does not reduce the drift from the defect data. While employing class rebalancing techniques during the drift detection process, we observe that the class rebalancing techniques eliminates drift up to 31% from the chronological defect datasets and improves prediction performance. ROSE class rebalancing technique eliminates drifts by balancing the datasets by up to 31%.

The impact of SMOTE and undersampling class rebalancing techniques on cross-version defect prediction models are statistically and practically significant of the performance metrics (Gmean, pd , and pf). Overall, the added benefit of class rebalancing techniques is noticeable.

Implications: We suggest adopting class rebalancing techniques in the concept drift elimination process from the chronological defect datasets and improve SDP performance. The experiment results allow us to guide practitioners to maintain high prediction performance of cross-version defect prediction models over time to eliminate drifts.

Finding 5: *Inter-release defect prediction models trained on the most recent releases do not always yield the best inter-release defect predictors.* We observe that the inter-release defect prediction models remain competitive throughout the releases in the temporal chunk-based learning environment. This observation agrees with the conclusion of Harman et al. [58]. They found that the latest release is not always the best predictor when considered SDP as a temporal learning problem suggested by Harman et al. [58], dynamic adaptive prediction systems [57] could be employed to recognize the best model.

Implications: When there is a need to identify (predict) the most defect-prone modules for a critical software system, we require to choose the prediction models that achieve high Recall [21, 56, 73]. Especially for inter-release defect prediction, where the software system becomes more critical by changing its functions in each release, we need to choose the models with high Recall scores. When the prediction models were updated by selecting the features from each release, the feature selection techniques could help reduce drift from the defect datasets.

Finding 6: *There were statistical differences between the prediction results with and without feature selection techniques when evaluated with the performance measures (AUC, Recall, and pf).* When we consider exploring the impact of feature

selection-based classification models in comparison without feature selection-based classification models for a chunk-based temporal learning environment in an inter-release defect prediction scenario, the feature selection-based prediction models surpassed all the without feature selection-based models. Our experiment shows that the AUTOS-NB achieves the highest average Recall, followed by correlation-based feature selection with NB attained an average score of 0.80. They also achieve the highest average AUC values with low pf scores.

Implications: The feature selection techniques help mitigate the drift up to 88% and improve the inter-release defect prediction scenario's prediction performance. Researchers and practitioners should consider concept drift for inter-release defect prediction while learning environment as chunk-based temporal learning. The findings and insights of our empirical study can be helpful in the software engineering researchers interested in improving the prediction performance of inter-release defect prediction. Overall, feature selection techniques could help to reduce the concept drifts and improve the prediction performance of inter-release defect prediction.

7.3 Research Contributions

The main contributions resulting from the empirical investigations carried in this thesis are listed in the section below.

1. Empirical investigation conducted to detect concept drift in the chronological defect datasets. Adopted statistical test enabled us to understand the drifts in the software defect datasets. Empirical studies suggest that classification-based SDP models need to be updated after the drift to obtain better prediction performances.

2. We provide drift detection method that identify drifts by providing three triggering flags: drift, warning, and control flags. We observe that an exponential increase of the error-rate addresses the drift's in the chronological defect datasets.
3. Conducted an empirical investigation to check whether class rebalancing techniques can help reduce the detection rate of concept drift from chronological defect datasets and impact the prediction performance (Gmean, pd , and pf) for cross-version defect prediction. Concept drift detection is model-aware for software defect datasets. We find that 50% of datasets are drift-prone when KNN classifier is applied, which is the maximum. DT achieves the minimum score of drift-proneness, i.e., 35%. Our experiment reveals that updating the model does not always improve the prediction performance and does not reduce the drift from the defect data. We observe that the class rebalancing techniques eliminates drift up to 31% from the chronological defect datasets and improves prediction performance.
4. We provide recommendation on which feature selection method to use that is robust for predicting defects for inter-release software projects. We observe that inter-release defect prediction models trained on the most recent releases do not always yield the best inter-release defect predictors.

7.4 Future Research Opportunities

The research investigated conducted in this thesis presented the impacts of concept drift and the advantages of adopting class rebalancing and feature selection techniques during the model training of SDP models and deletion of concept drift from the software defect datasets. Although, these empirical

investigations have stirred up several questions. Future works could address some of those. We present brief highlights of some of the likely future works below.

7.4.1 Investigating the impact on other class rebalancing techniques

Our empirical investigation in Chapter 5 focuses only on the Random Over Sampling, Random Under Sampling, Synthetic Minority Oversampling Technique, and Bootstrap Random Oversampling Examples class rebalancing techniques. Although prior SDP studies have also widely adopted these techniques [10, 12, 14, 15, 161], our findings have entirely bound these techniques. Future research should explore the impact of other class rebalancing techniques (e.g., Adaptive Synthetic Sampling technique).

7.4.2 Understanding the impact of configuration parameters

The thesis investigated the impacts of class rebalancing techniques using Caret parameter optimization and feature selection techniques for detecting concept drifts and SDP performance improvement. Parameters such as nearest-neighbors, the data sample rates for class rebalancing techniques, and parameters for convention learners could be examined on concept drift and SDP performance improvement.

7.4.3 Investigating the impact of other feature selection techniques

Our study in Chapter 6 focuses only on the correlation and consistency-based feature selection techniques and the AutoSpearman feature selection technique. Although prior SDP studies have also widely adopted these techniques [46, 76, 93, 146, 180]. Therefore, conclusions may not hold if another feature selection technique is applied. Future research should explore Wrapper-based Feature Selection Techniques on concept drift and impact for SDP (e.g., Recursive Feature Elimination technique).

7.4.4 Investigating the impact of deep learning models

Throughout our empirical studies in this thesis, we consider conventional classifiers. There are various deep learning solutions that could be used to the problem of defect predictions and concept drift detection. Since these classification models perform automatic feature engineering, it is possible that certain capture features are enough to minimize the effects of drift. In a further study, we would benefit from consideration of these types of models.

7.4.5 Revisiting the impact of concept drift in various setting of defect prediction models

Future research studies should revisit the previous studies in various settings to examine the impact of concept drift in SDP, such as just-in-time defect prediction [37, 69, 185], and heterogeneous defect prediction [23, 132]. Besides, search-based SDP techniques could recognize the best set of classification techniques [19] that could eliminate concept drifts from the software defect prediction datasets, which could be explored as future research to enhance the performance of software defect prediction models.

Appendix A

Publications

Under Review

1. Md Alamgir Kabir, Jacky Keung, Burak Turhan, Kwabena Ebo Bennin, "An Empirical Study on Class Rebalancing Techniques in Eliminating Concept Drift in Chronological Defect Datasets.", Automated Software Engineering, Submission Date: 25 Mar 2021. (Revise)
2. Md Alamgir Kabir, Jacky Keung, Burak Turhan, Kwabena Ebo Bennin, "Inter-release defect prediction with feature selection using temporal chunk-based learning.", Applied Soft Computing, 23 May 2021, (Revision Under Review)
3. Shuo Feng, Jacky Keung, Yan Xiao, Xiao Yu , Miao Zhang and Md Alamgir Kabir, "The impact of the distance metric and measure on SMOTE-based techniques in software

defect prediction.", Information and Software Technology,
Apr 26, 2021. (Revision Under Review)

Published Papers

1. Yang, Zhen, Jacky Keung, Md Alamgir Kabir, Xiao Yu, Yutian Tang, Miao Zhang, and Shuo Feng. "AComNN: Attention enhanced Compound Neural Network For financial time-series forecasting with cross-regional features." *Applied Soft Computing*, 2021, 107649.
2. Zhang, M., Keung, J., Xiao, Y., and Kabir, M. A., 2021. "Evaluating the effects of similar-class combination on class integration test order generation.", *Information and Software Technology*, 129, p.106438.
3. Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., and Zhang, M., 2021. "COSTE: Complexity-based Over-Sampling TEchnique to alleviate the class imbalance problem in software defect prediction.", *Information and Software Technology*, 129, p.106432.
4. Kabir, M. A., Keung, J., Bennin, K. E., and Zhang, M., 2020. "A drift propensity detection technique to improve the performance for cross-version software defect prediction.", In

2020 IEEE 44th Annual Computer Software and Applications Conference, pp. 882-891.

5. Kabir, M. A., Keung, J., Bennin, K. E., and Zhang, M., 2019. "Assessing the significant impact of concept drift in software defect prediction.", In 2019 IEEE 43rd Annual Computer Software and Applications Conference, pp. 53-58.
6. Zhang, M., Keung, J., Xiao, Y., Kabir, M. A., and Feng, S., 2019. "A heuristic approach to break cycles for the class integration test order generation.", In 2019 IEEE 43rd Annual Computer Software and Applications Conference, Vol. 1, pp. 47-52.

Bibliography

- [1] A. Agrawal and T. Menzies. “Is “Better Data” Better Than “Better Data Miners”?” In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 2018, pp. 1050–1061. DOI: 10.1145/3180155.3180197.
- [2] Tim Menzies et al. *jm1*. Dec. 2004. DOI: 10.5281/zenodo.268514. URL: <https://doi.org/10.5281/zenodo.268514>.
- [3] Sousuke Amasaki. “Cross-version defect prediction: use historical data, cross-project data, or both?” In: *Empirical Software Engineering* (2020), pp. 1–23.
- [4] Sousuke Amasaki. “Cross-Version Defect Prediction Using Cross-Project Defect Prediction Approaches: Does It Work?” In: *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE’18. Oulu, Finland: Association for Computing Machinery, 2018, 32–41. ISBN: 9781450365932. DOI: 10.1145/3273934.3273938. URL: <https://doi.org/10.1145/3273934.3273938>.
- [5] Sousuke Amasaki. “On Applicability of Cross-Project Defect Prediction Method for Multi-Versions Projects”. In: *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE. Toronto, Canada: Association for Computing Machinery, 2017, 93–96. ISBN: 9781450353052. DOI: 10.1145/3127005.3127015. URL: <https://doi.org/10.1145/3127005.3127015>.

- [6] Manuel Baena-Garcia et al. "Early drift detection method". In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. 2006, pp. 77–86.
- [7] Abdul Ali Bangash et al. "On the time-based conclusion stability of cross-project defect prediction models". eng. In: *Empirical software engineering : an international journal* (2020). ISSN: 1382-3256.
- [8] V. R. Basili, R. W. Selby, and D. H. Hutchens. "Experimentation in software engineering". In: *IEEE Transactions on Software Engineering* SE-12.7 (1986), pp. 733–743. DOI: 10.1109/TSE.1986.6312975.
- [9] V. R. Basili, F. Shull, and F. Lanubile. "Building knowledge through families of experiments". In: *IEEE Transactions on Software Engineering* 25.4 (1999), pp. 456–473. DOI: 10.1109/32.799939.
- [10] K. E. Bennin, J. Keung, and A. Monden. "Impact of the Distribution Parameter of Data Sampling Approaches on Software Defect Prediction Models". In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 2017, pp. 630–635. DOI: 10.1109/APSEC.2017.76.
- [11] K. E. Bennin et al. "Empirical Evaluation of Cross-Release Effort-Aware Defect Prediction Models". In: *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 2016, pp. 214–221. DOI: 10.1109/QRS.2016.33.
- [12] K. E. Bennin et al. "MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction". In: *IEEE Transactions on Software Engineering* 44.6 (2018), pp. 534–550. ISSN: 2326-3881. DOI: 10.1109/TSE.2017.2731766.
- [13] K. E. Bennin et al. "Revisiting the Impact of Concept Drift on Just-in-Time Quality Assurance". In: *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. 2020, pp. 53–59. DOI: 10.1109/QRS51102.2020.00020.

-
- [14] K. E. Bennin et al. "The Significant Effects of Data Sampling Approaches on Software Defect Prioritization and Classification". In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2017, pp. 364–373. DOI: 10.1109/ESEM.2017.50.
- [15] Kwabena Ebo Bennin, Jacky W. Keung, and Akito Monden. "On the relative value of data resampling approaches for software defect prediction". In: *Empirical Software Engineering* 24.2 (2019), pp. 602–636. ISSN: 1573-7616. DOI: 10.1007/s10664-018-9633-6. URL: <https://doi.org/10.1007/s10664-018-9633-6>.
- [16] Abraham Bernstein, Jayalath Ekanayake, and Martin Pinzger. "Improving Defect Prediction Using Temporal Features and Non Linear Models". In: *Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting. IWPSE '07*. Dubrovnik, Croatia: Association for Computing Machinery, 2007, 11–18. ISBN: 9781595937223. DOI: 10.1145/1294948.1294953. URL: <https://doi.org/10.1145/1294948.1294953>.
- [17] Albert Bifet and Ricard Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007, pp. 443–448.
- [18] Alexis Bondu, Benoît Grossin, and Marie-Luce Picard. "Density estimation on data stream: an application to change detection." In: *EGC*. Vol. 10. 2010, pp. 229–240.
- [19] Gerardo Canfora et al. "Defect prediction as a multiobjective optimization problem". In: *Software Testing, Verification and Reliability* 25.4 (2015), pp. 426–459.
- [20] Emily Carl et al. "Psychological and pharmacological treatments for generalized anxiety disorder (GAD): a meta-analysis of randomized controlled trials". In: *Cognitive behaviour therapy* 49.1 (2020), pp. 1–21.

- [21] Cagatay Catal and Banu Diri. "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem". In: *Information Sciences* 179.8 (2009), pp. 1040–1058. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2008.12.001>.
- [22] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.
- [23] Xiang Chen et al. "Revisiting heterogeneous defect prediction methods: How far are we?" In: *Information and Software Technology* 130 (2021), p. 106441. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2020.106441>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584920301944>.
- [24] S. R. Chidamber and C. F. Kemerer. "A metrics suite for object oriented design". In: *IEEE Transactions on Software Engineering* 20.6 (1994), pp. 476–493. DOI: 10.1109/32.295895.
- [25] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013.
- [26] Manoranjan Dash and Huan Liu. "Consistency-based search in feature selection". In: *Artificial Intelligence* 151.1 (2003), pp. 155–176. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/S0004-3702\(03\)00079-1](https://doi.org/10.1016/S0004-3702(03)00079-1).
- [27] Manoranjan Dash, Huan Liu, and Hiroshi Motoda. "Consistency Based Feature Selection". In: *Knowledge Discovery and Data Mining. Current Issues and New Applications*. Ed. by Takao Terano, Huan Liu, and Arbee L. P. Chen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 98–109. ISBN: 978-3-540-45571-4.
- [28] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine learning research* 7.Jan (2006), pp. 1–30.

-
- [29] Evgenia Dimitriadou et al. "Misc functions of the Department of Statistics (e1071), TU Wien". In: *R package* 1 (2008), pp. 5–24.
- [30] G. Ditzler et al. "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational Intelligence Magazine* 10.4 (2015), pp. 12–25. ISSN: 1556-6048. DOI: 10.1109/MCI.2015.2471196.
- [31] F. Dong et al. "Concept drift region identification via competence-based discrepancy distribution estimation". In: *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. 2017, pp. 1–7. DOI: 10.1109/ISKE.2017.8258734.
- [32] Richard O Duda. *Pattern classification*. eng. 2nd ed. New York: Wiley, 2001. ISBN: 0471056693.
- [33] Tore Dybå, Vigdis By Kampenes, and Dag I.K. Sjøberg. "A systematic review of statistical power in software engineering experiments". In: *Information and Software Technology* 48.8 (2006), pp. 745–755. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2005.08.009>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584905001333>.
- [34] Marco D'Ambros, Michele Lanza, and Romain Robbes. "Evaluating defect prediction approaches: a benchmark and an extensive comparison". In: *Empirical Software Engineering* 17.4-5 (2012), pp. 531–577.
- [35] J. Ekanayake et al. "Tracking concept drift of software projects using defect prediction quality". In: *2009 6th IEEE International Working Conference on Mining Software Repositories*. 2009, pp. 51–60. DOI: 10.1109/MSR.2009.5069480.
- [36] Jayalath Ekanayake et al. "Time variance and defect prediction in software projects". In: *Empirical Software Engineering* 17.4 (2012), pp. 348–389. ISSN: 1573-7616. DOI: 10.1007/s10664-011-9180-x. URL: <https://doi.org/10.1007/s10664-011-9180-x>.

- [37] Y. Fan et al. "The Impact of Changes Misclassified by SZZ on Just-in-Time Defect Prediction". In: *IEEE Transactions on Software Engineering* (2019), pp. 1–1. ISSN: 1939-3520. DOI: 10.1109/TSE.2019.2929761.
- [38] Shuo Feng et al. "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction". In: *Information and Software Technology* 129 (2021), p. 106432. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2020.106432>. URL: <http://www.sciencedirect.com/science/article/pii/S0950584920301889>.
- [39] RA Fisher. *Statistical Methods for Research Workers Biological Monographs and Manuals, Ed. 12*, New York, Hafner Pub. Co. 1954.
- [40] Milton Friedman. "A comparison of alternative tests of significance for the problem of m rankings". In: *The Annals of Mathematical Statistics* 11.1 (1940), pp. 86–92.
- [41] Joao Gama. *Knowledge discovery from data streams*. CRC Press, 2010.
- [42] João Gama et al. "A survey on concept drift adaptation". In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.
- [43] Joao Gama et al. "Learning with drift detection". In: *Brazilian symposium on artificial intelligence*. Springer. 2004, pp. 286–295.
- [44] Federica Garrazone et al. "Personality profile and depression in migraine: a meta-analysis". In: *Neurological sciences* 41.3 (2020), pp. 543–554.
- [45] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. "Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey". In: *ACM Comput. Surv.* 50.4 (Aug. 2017). ISSN: 0360-0300. DOI: 10.1145/3092566. URL: <https://doi.org/10.1145/3092566>.

-
- [46] B. Ghotra, S. McIntosh, and A. E. Hassan. "A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017, pp. 146–157. DOI: 10.1109/MSR.2017.18.
- [47] B. Ghotra, S. McIntosh, and A. E. Hassan. "Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 789–800. DOI: 10.1109/ICSE.2015.91.
- [48] Simon B Goldberg et al. "The experimental effects of psilocybin on symptoms of anxiety and depression: A meta-analysis". In: *Psychiatry research* 284 (2020), p. 112749.
- [49] L. Gong et al. "A Novel Class-Imbalance Learning Approach for Both Within-Project and Cross-Project Defect Prediction". In: *IEEE Transactions on Reliability* (2019), pp. 1–15. ISSN: 1558-1721. DOI: 10.1109/TR.2019.2895462.
- [50] Lina Gong et al. "Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019, pp. 698–709. DOI: 10.1109/ASE.2019.00071.
- [51] Paulo M. Gonçalves et al. "A comparative study on concept drift detectors". In: *Expert Systems with Applications* 41.18 (2014), pp. 8144 – 8156. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2014.07.019>. URL: <http://www.sciencedirect.com/science/article/pii/S0957417414004175>.

- [52] D. Gray et al. "The misuse of the NASA metrics data program data sets for automated software defect prediction". In: *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*. 2011, pp. 96–103. DOI: 10.1049/ic.2011.0012.
- [53] Mark Andrew Hall. "Correlation-based feature selection for machine learning". In: (1999).
- [54] T. Hall et al. "A Systematic Literature Review on Fault Prediction Performance in Software Engineering". In: *IEEE Transactions on Software Engineering* 38.6 (2012), pp. 1276–1304. ISSN: 1939-3520. DOI: 10.1109/TSE.2011.103.
- [55] James A Hanley and Barbara J McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." In: *Radiology* 143.1 (1982), pp. 29–36.
- [56] Ahmed Taha Haouari et al. "Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction". In: *Applied Soft Computing* 96 (2020), p. 106686. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2020.106686>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494620306244>.
- [57] M. Harman et al. "Dynamic adaptive Search Based Software Engineering". In: *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 2012, pp. 1–8. DOI: 10.1145/2372251.2372253.
- [58] Mark Harman et al. "Less is More: Temporal Fault Predictive Performance over Multiple Hadoop Releases". In: *Search-Based Software Engineering*. Ed. by Claire Le Goues and Shin Yoo. Cham: Springer International Publishing, 2014, pp. 240–246. ISBN: 978-3-319-09940-8.

-
- [59] Johannes Haug and Gjergji Kasneci. "Learning Parameter Distributions to Detect Concept Drift in Data Streams". In: *arXiv preprint arXiv:2010.09388* (2020).
- [60] Peng He et al. "An empirical study on software defect prediction with a simplified metric set". In: *Information and Software Technology* 59 (2015), pp. 170–190. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2014.11.006>.
- [61] Z. He et al. "Learning from Open-Source Projects: An Empirical Study on Defect Prediction". In: *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 2013, pp. 45–54. DOI: 10.1109/ESEM.2013.20.
- [62] Zhimin He et al. "An investigation on the feasibility of cross-project defect prediction". In: *Automated Software Engineering* 19.2 (2012), pp. 167–199.
- [63] Larry V Hedges and Ingram Olkin. *Statistical methods for meta-analysis*. Academic press, 2014.
- [64] S. Herbold, A. Trautsch, and J. Grabowski. "A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches". In: *IEEE Transactions on Software Engineering* 44.9 (2018), pp. 811–833. ISSN: 1939-3520. DOI: 10.1109/TSE.2017.2724538.
- [65] Christina Hermling and Elena Hermling. "Investigating the Boeing 737 MAX 8 Crisis and its Economic Effects". In: (2020).
- [66] S. Hosseini, B. Turhan, and D. Gunarathna. "A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction". In: *IEEE Transactions on Software Engineering* 45.2 (2019), pp. 111–147. ISSN: 2326-3881. DOI: 10.1109/TSE.2017.2770124.

- [67] Seyedrebbavar Hosseini, Burak Turhan, and Mika Mäntylä. “A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction”. In: *Information and Software Technology* 95 (2018), pp. 296–312. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2017.06.004>.
- [68] Y Hou. “Détection de concept drift”. In: *Master Degree*, University of Technology of Compiègne Hulten, G., Spencer, L., & Domingos, P.(2001). *Mining time-changing data streams. Paper presented at the Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, California.* 2012.
- [69] Qiao Huang, Xin Xia, and David Lo. “Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction”. In: *Empirical Software Engineering* 24.5 (2019), pp. 2823–2862.
- [70] S. Huang, R. Jin, and Z. Zhou. “Active Learning by Querying Informative and Representative Examples”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.10 (2014), pp. 1936–1949. DOI: 10.1109/TPAMI.2014.2307881.
- [71] S. Huda et al. “An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction”. In: *IEEE Access* 6 (2018), pp. 24184–24195. DOI: 10.1109/ACCESS.2018.2817572.
- [72] T. Jiang, L. Tan, and S. Kim. “Personalized defect prediction”. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2013, pp. 279–289. DOI: 10.1109/ASE.2013.6693087.
- [73] Yue Jiang, Bojan Cukic, and Yan Ma. “Techniques for evaluating fault prediction models”. In: *Empirical Software Engineering* 13.5 (2008), pp. 561–595.

-
- [74] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude. "Artefact: An R Implementation of the AutoSpearman Function". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018, pp. 711–711. DOI: 10.1109/ICSME.2018.00083.
- [75] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude. "AutoSpearman: Automatically Mitigating Correlated Software Metrics for Interpreting Defect Models". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018, pp. 92–103. DOI: 10.1109/ICSME.2018.00018.
- [76] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. "The impact of automated feature selection techniques on the interpretation of defect models". In: *Empirical Software Engineering* 25.5 (2020), pp. 3590–3638.
- [77] Cong Jin. "Cross-project software defect prediction based on domain adaptation learning and optimization". In: *Expert Systems with Applications* 171 (2021), p. 114637.
- [78] X. Jing et al. "An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems". In: *IEEE Transactions on Software Engineering* 43.4 (2017), pp. 321–339. DOI: 10.1109/TSE.2016.2597849.
- [79] Phillip Johnston and Rozi Harris. "The Boeing 737 MAX saga: lessons for software organizations". In: *Software Quality Professional* 21.3 (2019), pp. 4–12.
- [80] Kapil Juneja. "A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation". In: *Applied Soft Computing* 77 (2019), pp. 696–713. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2019.02.008>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494619300687>.

- [81] Prem Junsawang, Suphakant Phimoltares, and Chidchanok Lursinsap. "Streaming chunk incremental learning for class-wise data stream classification with fast learning speed and low structural complexity". In: *PLOS ONE* 14.9 (Sept. 2019), pp. 1–20. DOI: 10 . 1371 / journal . pone . 0220624. URL: <https://doi.org/10.1371/journal.pone.0220624>.
- [82] Marian Jureczko. *prop.* July 2010. DOI: 10 . 5281/zenodo . 322444. URL: <https://doi.org/10.5281/zenodo.322444>.
- [83] Marian Jureczko and Lech Madeyski. "Towards Identifying Software Project Clusters with Regard to Defect Prediction". In: *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. PROMISE '10. Timișoara, Romania: Association for Computing Machinery, 2010. ISBN: 9781450304047. DOI: 10 . 1145 / 1868328 . 1868342. URL: <https://doi.org/10.1145/1868328.1868342>.
- [84] Marian Jureczko and Diomidis Spinellis. "Using object-oriented design metrics to predict software defects". In: *Models and Methods of System Dependability*. Oficyna Wydawnicza Politechniki Wrocławskiej (2010), pp. 69–81.
- [85] M. A. Kabir et al. "A Drift Propensity Detection Technique to Improve the Performance for Cross-Version Software Defect Prediction". In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2020, pp. 882–891. DOI: 10 . 1109/COMPSAC48688 . 2020 . 0-154.
- [86] M. A. Kabir et al. "Assessing the Significant Impact of Concept Drift in Software Defect Prediction". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. 2019, pp. 53–58. DOI: 10 . 1109/COMPSAC . 2019 . 00017.

-
- [87] Md Alamgir Kabir et al. *supplemental materials*. <https://cutt.ly/AmnG8bx>. 2021.
- [88] Vigdis By Kampenes et al. "A systematic review of effect size in software engineering experiments". In: *Information and Software Technology* 49.11 (2007), pp. 1073–1086. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2007.02.015>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584907000195>.
- [89] B. A. Kitchenham et al. "Preliminary guidelines for empirical research in software engineering". In: *IEEE Transactions on Software Engineering* 28.8 (2002), pp. 721–734. DOI: 10.1109/TSE.2002.1027796.
- [90] Barbara Kitchenham et al. "Robust Statistical Methods for Empirical Software Engineering". In: *Empirical Software Engineering* 22.2 (2017), pp. 579–630. ISSN: 1573-7616. DOI: 10.1007/s10664-016-9437-5. URL: <https://doi.org/10.1007/s10664-016-9437-5>.
- [91] Ralf Klinkenberg and Thorsten Joachims. "Detecting Concept Drift with Support Vector Machines". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 487–494. ISBN: 1-55860-707-2. URL: <http://dl.acm.org/citation.cfm?id=645529.657791>.
- [92] E. Kocaguneli et al. "Active learning and effort estimation: Finding the essential content of software effort estimation data". In: *IEEE Transactions on Software Engineering* 39.8 (2013), pp. 1040–1053. ISSN: 1939-3520. DOI: 10.1109/TSE.2012.88.
- [93] Masanari Kondo et al. "The impact of feature reduction techniques on defect prediction models". In: *Empirical Software Engineering* 24.4 (2019), pp. 1925–1963.

- [94] Bartosz Krawczyk et al. "Ensemble learning for data stream analysis: A survey". In: *Information Fusion* 37 (2017), pp. 132–156. ISSN: 1566-2535. DOI: <https://doi.org/10.1016/j.inffus.2017.02.004>.
- [95] R. Krishna and T. Menzies. "Bellwethers: A Baseline Method for Transfer Learning". In: *IEEE Transactions on Software Engineering* 45.11 (2019), pp. 1081–1105. DOI: 10.1109/TSE.2018.2821670.
- [96] Max Kuhn. "Caret: classification and regression training". In: *Astrophysics Source Code Library* (2015), ascl-1505.
- [97] Max Kuhn et al. "Package 'caret'". In: *The R Journal* (2020).
- [98] S. Lessmann et al. "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings". In: *IEEE Transactions on Software Engineering* 34.4 (2008), pp. 485–496. DOI: 10.1109/TSE.2008.35.
- [99] Zhiqiang Li et al. "Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction". In: *Automated Software Engineering* 25.2 (2018), pp. 201–245.
- [100] Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. "Concept drift detection based on Fisher's Exact test". In: *Information Sciences* 442-443 (2018), pp. 220–234. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2018.02.054>. URL: <https://www.sciencedirect.com/science/article/pii/S0020025518301403>.
- [101] Qingwei Lin et al. "Predicting node failure in cloud service systems". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 480–490.
- [102] Qingwei Lin et al. "Predicting Node Failure in Cloud Service Systems". In: *Proceedings of the 2018 26th ACM Joint Meeting on European*

- Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, 480–490. ISBN: 9781450355735. DOI: 10.1145/3236024.3236060. URL: <https://doi.org/10.1145/3236024.3236060>.
- [103] Anjin Liu et al. “Regional Concept Drift Detection and Density Synchronized Drift Adaptation”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, 2017, 2280–2286. ISBN: 9780999241103.
- [104] C. Lokan and E. Mendes. “Investigating the use of chronological split for software effort estimation”. In: *IET Software* 3.5 (2009), pp. 422–434. ISSN: 1751-8814. DOI: 10.1049/iet-sen.2008.0107.
- [105] Chris Lokan and Emilia Mendes. “Applying Moving Windows to Software Effort Estimation”. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. ESEM ’09. USA: IEEE Computer Society, 2009, 111–122. ISBN: 9781424448425. DOI: 10.1109/ESEM.2009.5316019. URL: <https://doi.org/10.1109/ESEM.2009.5316019>.
- [106] Chris Lokan and Emilia Mendes. “Investigating the Use of Chronological Splitting to Compare Software Cross-Company and Single-Company Effort Predictions”. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. EASE’08. Italy: BCS Learning & Development Ltd., 2008, 136–145.
- [107] Chris Lokan and Emilia Mendes. “Investigating the Use of Duration-Based Moving Windows to Improve Software Effort Prediction”. In: *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01*. APSEC ’12. USA: IEEE Computer Society, 2012, 818–827.

- ISBN: 9780769549224. DOI: 10 . 1109 / APSEC . 2012 . 74. URL: <https://doi.org/10.1109/APSEC.2012.74>.
- [108] Chris Lokan and Emilia Mendes. "Investigating the use of duration-based moving windows to improve software effort prediction: A replicated study". In: *Information and Software Technology* 56.9 (2014). Special Sections from "Asia-Pacific Software Engineering Conference (APSEC), 2012" and "Software Product Line conference (SPLC), 2012", pp. 1063–1075. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2014.02.008>. URL: <http://www.sciencedirect.com/science/article/pii/S0950584914000500>.
- [109] Chris Lokan and Emilia Mendes. "Investigating the use of moving windows to improve software effort prediction: a replicated study". *eng*. In: *Empirical Software Engineering* 22.2 (2017), pp. 716–767. ISSN: 1382-3256.
- [110] H. Lu, E. Kocaguneli, and B. Cukic. "Defect Prediction between Software Versions with Active Learning and Dimensionality Reduction". In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*. 2014, pp. 312–322. DOI: 10 . 1109 / ISSRE . 2014 . 35.
- [111] J. Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2346–2363. ISSN: 2326-3865. DOI: 10 . 1109 / TKDE . 2018 . 2876857.
- [112] Jie Lu et al. "Data-driven decision support under concept drift in streamed big data". In: *Complex & Intelligent Systems* (2019). ISSN: 2198-6053. DOI: 10 . 1007 / s40747 - 019 - 00124 - 4. URL: <https://doi.org/10.1007/s40747-019-00124-4>.
- [113] Jie Lu et al. "Learning under concept drift: A review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2018), pp. 2346–2363.

-
- [114] Ning Lu, Guangquan Zhang, and Jie Lu. "Concept drift detection via competence models". In: *Artificial Intelligence* 209 (2014), pp. 11–28. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2014.01.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0004370214000034>.
- [115] Nicola Lunardon, Giovanna Menardi, and Nicola Torelli. "ROSE: A Package for Binary Imbalanced Learning." In: *R journal* 6.1 (2014).
- [116] YINGZHE LYU et al. "An Empirical Study of the Impact of Data Splitting Decisions on the Performance of AIOps Solutions". In: *ACM Transactions on Software Engineering and Methodology*, pp. to appear (2021).
- [117] Lech Madeyski and Marian Jureczko. "Which process metrics can significantly improve defect prediction models? An empirical study". In: *Software Quality Journal* 23.3 (2015), pp. 393–422. ISSN: 1573-1367. DOI: 10.1007/s11219-014-9241-7. URL: <https://doi.org/10.1007/s11219-014-9241-7>.
- [118] Osama A. Mahdi et al. "Fast Reaction to Sudden Concept Drift in the Absence of Class Labels". In: *Applied Sciences* 10.2 (2020). ISSN: 2076-3417. DOI: 10.3390/app10020606. URL: <https://www.mdpi.com/2076-3417/10/2/606>.
- [119] Ruchika Malhotra. "A systematic review of machine learning techniques for software fault prediction". In: *Applied Soft Computing* 27 (2015), pp. 504–518. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2014.11.023>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494614005857>.
- [120] Ruchika Malhotra. "An empirical framework for defect prediction using machine learning techniques with Android software". In: *Applied Soft Computing* 49 (2016), pp. 1034–1050. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2016.05.023>.

- [//doi.org/10.1016/j.asoc.2016.04.032](https://doi.org/10.1016/j.asoc.2016.04.032). URL: <https://www.sciencedirect.com/science/article/pii/S1568494616301806>.
- [121] T. Menzies, J. Greenwald, and A. Frank. "Data Mining Static Code Attributes to Learn Defect Predictors". In: *IEEE Transactions on Software Engineering* 33.1 (2007), pp. 2–13. DOI: 10.1109/TSE.2007.256941.
- [122] T. Menzies et al. "Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'"". In: *IEEE Transactions on Software Engineering* 33.9 (2007), pp. 637–640. DOI: 10.1109/TSE.2007.70721.
- [123] Tim Menzies et al. "Defect prediction from static code features: current results, limitations, new approaches". In: *Automated Software Engineering* 17.4 (2010), pp. 375–407.
- [124] Tim Menzies et al. "Implications of Ceiling Effects in Defect Predictors". In: *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*. PROMISE '08. Leipzig, Germany: Association for Computing Machinery, 2008, 47–54. ISBN: 9781605580364. DOI: 10.1145/1370788.1370801. URL: <https://doi.org/10.1145/1370788.1370801>.
- [125] James Miller. "Applying meta-analytical procedures to software engineering experiments". In: *Journal of Systems and Software* 54.1 (2000), pp. 29–39. ISSN: 0164-1212. DOI: [https://doi.org/10.1016/S0164-1212\(00\)00024-8](https://doi.org/10.1016/S0164-1212(00)00024-8). URL: <https://www.sciencedirect.com/science/article/pii/S0164121200000248>.
- [126] L. L. Minku and X. Yao. "DDD: A New Ensemble Approach for Dealing with Concept Drift". In: *IEEE Transactions on Knowledge and Data Engineering* 24.4 (2012), pp. 619–633. DOI: 10.1109/TKDE.2011.58.

- [127] Leandro Minku and Xin Yao. "Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models". eng. In: *Automated Software Engineering* 24.3 (2017), pp. 499–542. ISSN: 0928-8910.
- [128] Leandro L. Minku. "Transfer Learning in Non-stationary Environments". In: *Learning from Data Streams in Evolving Environments: Methods and Applications*. Ed. by Moamar Sayed-Mouchaweh. Cham: Springer International Publishing, 2019, pp. 13–37. ISBN: 978-3-319-89803-2.
- [129] Leandro L. Minku and Xin Yao. "Can Cross-Company Data Improve Performance in Software Effort Estimation?" In: *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*. PROMISE '12. Lund, Sweden: Association for Computing Machinery, 2012, 69–78. ISBN: 9781450312417. DOI: 10 . 1145 / 2365324 . 2365334. URL: <https://doi.org/10.1145/2365324.2365334>.
- [130] Leandro L. Minku and Xin Yao. "How to Make Best Use of Cross-Company Data in Software Effort Estimation?" In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, 446–456. ISBN: 9781450327565. DOI: 10 . 1145 / 2568225 . 2568228. URL: <https://doi.org/10.1145/2568225.2568228>.
- [131] K. Muthukumaran et al. "On the Effectiveness of Cost Sensitive Neural Networks for Software Defect Prediction". In: *Proceedings of the Eighth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2016)*. Ed. by Ajith Abraham et al. Cham: Springer International Publishing, 2018, pp. 557–570. ISBN: 978-3-319-60618-7.
- [132] J. Nam et al. "Heterogeneous Defect Prediction". In: *IEEE Transactions on Software Engineering* 44.9 (2018), pp. 874–896. DOI: 10 . 1109 / TSE . 2017 . 2720603.

- [133] Jaechang Nam and Sunghun Kim. "CLAMI: Defect Prediction on Unlabeled Datasets (T)". In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015, pp. 452–463. DOI: 10.1109/ASE.2015.56.
- [134] P Nemenyi. "Distribution-free multiple comparisons". In: *(Doctoral dissertation, Princeton University)* 64-6278 (1963).
- [135] Kyosuke Nishida and Koichiro Yamauchi. "Detecting concept drift using statistical testing". In: *International conference on discovery science*. Springer. 2007, pp. 264–269.
- [136] Kyosuke Nishida, Koichiro Yamauchi, and Takashi Omori. "ACE: Adaptive classifiers-ensemble system for concept-drifting environments". In: *International Workshop on Multiple Classifier Systems*. Springer. 2005, pp. 176–185.
- [137] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. "Machine learning based methods for software fault prediction: A survey". In: *Expert Systems with Applications* 172 (2021), p. 114595. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2021.114595>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417421000361>.
- [138] Jeff Papows. *Glitch: the hidden impact of faulty software*. Pearson Education, 2010.
- [139] Ali Pesaranghader and Herna L. Viktor. "Fast Hoeffding Drift Detection Method for Evolving Data Streams". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Paolo Frasconi et al. Cham: Springer International Publishing, 2016, pp. 96–111. ISBN: 978-3-319-46227-1.

- [140] F. Peters, T. Menzies, and L. Layman. "LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 801–811. DOI: 10.1109/ICSE.2015.92.
- [141] Fayola Peters, Tim Menzies, and Andrian Marcus. "Better cross company defect prediction". In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, pp. 409–418. DOI: 10.1109/MSR.2013.6624057.
- [142] Strategic Planning. "The economic impacts of inadequate infrastructure for software testing". In: *National Institute of Standards and Technology* (2002).
- [143] W. N. Poon et al. "Cross-Project Defect Prediction Using a Credibility Theory Based Naive Bayes Classifier". In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 2017, pp. 434–441. DOI: 10.1109/QRS.2017.53.
- [144] Faimison Porto et al. "A systematic study of cross-project defect prediction with meta-learning". In: *arXiv preprint arXiv:1802.06025* (2018).
- [145] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. "Recalling the "Imprecision" of Cross-Project Defect Prediction". In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. FSE '12. Cary, North Carolina: Association for Computing Machinery, 2012. ISBN: 9781450316149. DOI: 10.1145/2393596.2393669. URL: <https://doi.org/10.1145/2393596.2393669>.
- [146] Santosh Singh Rathore and Atul Gupta. "A Comparative Study of Feature-Ranking and Feature-Subset Selection Techniques for Improved Fault Prediction". In: *Proceedings of the 7th India Software Engineering*

- Conference. ISEC '14. Chennai, India: Association for Computing Machinery, 2014. ISBN: 9781450327763. DOI: 10.1145/2590748.2590755.
- [147] D. Rodriguez et al. "Detecting Fault Modules Applying Feature Selection to Classifiers". In: *2007 IEEE International Conference on Information Reuse and Integration*. 2007, pp. 667–672. DOI: 10.1109/IRI.2007.4296696.
- [148] Jeanine Romano et al. "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys". In: *annual meeting of the Florida Association of Institutional Research*. 2006, pp. 1–33.
- [149] Piotr Romanski, Lars Kotthoff, and Maintainer Lars Kotthoff. "Package 'FSelector'". In: URL <http://cran.r-project.org/web/packages/FSelector/index.html> (2013).
- [150] Felix Salfner, Maren Lenk, and Mirosław Malek. "A Survey of Online Failure Prediction Methods". In: *ACM Comput. Surv.* 42.3 (Mar. 2010). ISSN: 0360-0300. DOI: 10.1145/1670679.1670680. URL: <https://doi.org/10.1145/1670679.1670680>.
- [151] Jeffrey C. Schlimmer and Richard H. Granger. "Incremental learning from noisy data". In: *Machine Learning* 1.3 (1986), pp. 317–354. ISSN: 1573-0565. DOI: 10.1007/BF00116895. URL: <https://doi.org/10.1007/BF00116895>.
- [152] HJA Schouten et al. "Comparing two independent binomial proportions by a modified chi square test". In: *Biometrical Journal* 22.3 (1980), pp. 241–248.
- [153] T. Sethi and Gagandeep. "Improved approach for software defect prediction using artificial neural networks". In: *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and*

- Future Directions*) (ICRITO). 2016, pp. 480–485. DOI: 10.1109/ICRITO.2016.7785003.
- [154] E. Shihab. “Pragmatic prioritization of software quality assurance efforts”. In: *2011 33rd International Conference on Software Engineering (ICSE)*. 2011, pp. 1106–1109. DOI: 10.1145/1985793.1986007.
- [155] Emad Shihab. *An exploration of challenges limiting pragmatic software defect prediction*. Citeseer, 2012.
- [156] Swapnil Shukla et al. “Multi-objective cross-version defect prediction”. In: *Soft Computing* 22.6 (2018), pp. 1959–1980. ISSN: 1433-7479. DOI: 10.1007/s00500-016-2456-8. URL: <https://doi.org/10.1007/s00500-016-2456-8>.
- [157] Y. Song et al. “A fuzzy kernel c-means clustering model for handling concept drift in regression”. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2017, pp. 1–6. DOI: 10.1109/FUZZ-IEEE.2017.8015515.
- [158] R. Subramanyam and M. S. Krishnan. “Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects”. In: *IEEE Transactions on Software Engineering* 29.4 (2003), pp. 297–310. DOI: 10.1109/TSE.2003.1191795.
- [159] Zhongbin Sun et al. “CFPS: Collaborative filtering based source projects selection for cross-project defect prediction”. In: *Applied Soft Computing* 99 (2021), p. 106940. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2020.106940>. URL: <https://www.sciencedirect.com/science/article/pii/S1568494620308784>.
- [160] Zhongbin Sun et al. “Collaborative filtering based recommendation of sampling methods for software defect prediction”. In: *Applied Soft Computing* 90 (2020), p. 106163. ISSN: 1568-4946. DOI: <https://doi.org/10.1016/j.asoc.2020.106163>.

- org/10.1016/j.asoc.2020.106163. URL: <https://www.sciencedirect.com/science/article/pii/S1568494620301034>.
- [161] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto. “The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models”. In: *IEEE Transactions on Software Engineering* 46.11 (2020), pp. 1200–1219. ISSN: 1939-3520. DOI: 10.1109/TSE.2018.2876537.
- [162] R Core Team et al. “R: A language and environment for statistical computing”. In: (2013).
- [163] *The Promise Repository of Empirical Software Engineering Data*. 2015.
- [164] *The SEACRAFT Repository of Empirical Software Engineering Data*. 2017.
- [165] Luis Torgo and Maintainer Luis Torgo. “Package ‘DMwR’”. In: *Comprehensive R Archive Network* (2013).
- [166] H. Tu, Z. Yu, and T. Menzies. “Better Data Labelling with EMBLEM (and how that Impacts Defect Prediction)”. In: *IEEE Transactions on Software Engineering* (2020), pp. 1–1. ISSN: 1939-3520. DOI: 10.1109/TSE.2020.2986415.
- [167] Burak Turhan. “On the dataset shift problem in software engineering prediction models”. In: *Empirical Software Engineering* 17.1 (2012), pp. 62–74.
- [168] Burak Turhan, Ayşe Tosun Mısırlı, and Ayşe Bener. “Empirical evaluation of the effects of mixed project data on learning defect predictors”. In: *Information and Software Technology* 55.6 (2013), pp. 1101 – 1118. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2012.10.003>. URL: <http://www.sciencedirect.com/science/article/pii/S0950584912002108>.

- [169] Burak Turhan et al. "On the relative value of cross-company and within-company data for defect prediction". In: *Empirical Software Engineering* 14.5 (2009), pp. 540–578.
- [170] András Vargha and Harold D Delaney. "A critique and improvement of the CL common language effect size statistics of McGraw and Wong". In: *Journal of Educational and Behavioral Statistics* 25.2 (2000), pp. 101–132.
- [171] J. Vinagre et al. "Statistically robust evaluation of stream-based recommender systems". In: *IEEE Transactions on Knowledge and Data Engineering* (2019), pp. 1–1. DOI: 10.1109/TKDE.2019.2960216.
- [172] S. Wang and X. Yao. "Using Class Imbalance Learning for Software Defect Prediction". In: *IEEE Transactions on Reliability* 62.2 (2013), pp. 434–443. DOI: 10.1109/TR.2013.2259203.
- [173] S. Wang et al. "Concept drift detection for online class imbalance learning". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 2013, pp. 1–10. DOI: 10.1109/IJCNN.2013.6706768.
- [174] S. Wang et al. "Deep Semantic Feature Learning for Software Defect Prediction". In: *IEEE Transactions on Software Engineering* 46.12 (2020), pp. 1267–1293. DOI: 10.1109/TSE.2018.2877612.
- [175] Geoffrey I. Webb et al. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994. ISSN: 1573-756X. DOI: 10.1007/s10618-015-0448-4. URL: <https://doi.org/10.1007/s10618-015-0448-4>.
- [176] Geoffrey I Webb et al. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.
- [177] Leland Willkinson. "Statistical Methods in Psychology Journals." In: *American Psychologist* 54.8 (1999), pp. 594–604.

- [178] X. Xia et al. "HYDRA: Massively Compositional Model for Cross-Project Defect Prediction". In: *IEEE Transactions on Software Engineering* 42.10 (2016), pp. 977–998. ISSN: 1939-3520. DOI: 10.1109/TSE.2016.2543218.
- [179] Z. Xu et al. "Cross-version defect prediction via hybrid active learning with kernel principal component analysis". In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018, pp. 209–220. DOI: 10.1109/SANER.2018.8330210.
- [180] Z. Xu et al. "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison". In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, pp. 309–320. DOI: 10.1109/ISSRE.2016.13.
- [181] Zhou Xu et al. "Cross Version Defect Prediction with Representative Data via Sparse Subset Selection". In: *Proceedings of the 26th Conference on Program Comprehension*. ICPC '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, 132–143. ISBN: 9781450357142. DOI: 10.1145/3196321.3196331. URL: <https://doi.org/10.1145/3196321.3196331>.
- [182] Zhou Xu et al. "Software defect prediction based on kernel PCA and weighted extreme learning machine". In: *Information and Software Technology* 106 (2019), pp. 182–200. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2018.10.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584918302088>.
- [183] Zhou Xu et al. "TSTSS: A two-stage training subset selection framework for cross version defect prediction". In: *Journal of Systems and Software* 154 (2019), pp. 59–78. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2019.03.027>.

- [184] X. Yang and W. Wen. "Ridge and Lasso Regression Models for Cross-Version Defect Prediction". In: *IEEE Transactions on Reliability* 67.3 (2018), pp. 885–896. ISSN: 1558-1721. DOI: 10.1109/TR.2018.2847353.
- [185] Yibiao Yang et al. "Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better than Supervised Models". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, 157–168. ISBN: 9781450342186. DOI: 10.1145/2950290.2950353. URL: <https://doi.org/10.1145/2950290.2950353>.
- [186] Frank Yates. "Contingency tables involving small numbers and the χ^2 test". In: *Supplement to the Journal of the Royal Statistical Society* 1.2 (1934), pp. 217–235.
- [187] F. Zhang et al. "Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016, pp. 309–320. DOI: 10.1145/2884781.2884839.
- [188] L. Zhang, J. Lin, and R. Karim. "Sliding Window-Based Fault Detection From High-Dimensional Data Streams". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.2 (2017), pp. 289–303. ISSN: 2168-2232. DOI: 10.1109/TSMC.2016.2585566.
- [189] Nana Zhang et al. "WGNCS: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation". In: *Information Sciences* 570 (2021), pp. 545–576. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2021.05.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0020025521004540>.

- [190] Thomas Zimmermann et al. "Cross-Project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process". In: *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ESEC/FSE '09. Amsterdam, The Netherlands: Association for Computing Machinery, 2009, 91–100. ISBN: 9781605580012. DOI: 10 . 1145 / 1595696 . 1595713. URL: [https : / / doi . org / 10 . 1145 / 1595696 . 1595713](https://doi.org/10.1145/1595696.1595713).