

# Inter-release defect prediction with feature selection using temporal chunk-based learning: An empirical study

Md Alamgir Kabir<sup>a,\*</sup>, Jacky Keung<sup>a,\*</sup>, Burak Turhan<sup>b,c</sup> and Kwabena Ebo Bennin<sup>d</sup>

<sup>a</sup>Department of Computer Science, City University of Hong Kong, Hong Kong

<sup>b</sup>University of Oulu, Finland

<sup>c</sup>Monash University, Australia

<sup>d</sup>Information Technology Group, Wageningen University and Research, Wageningen, The Netherlands

---

## ARTICLE INFO

### Keywords:

Software defect prediction

Inter-release defect prediction

Feature selection

## ABSTRACT

Inter-release defect prediction (IRDP) is a practical scenario that employs the datasets of the previous release to build a prediction model and predicts defects for the current release within the same software project. A practical software project experiences several releases where data of each release appears in the form of chunks that arrive in temporal order. The evolving data of each release introduces new concept to the model known as concept drift, which negatively impacts the performance of IRDP models. In this study, we aim to examine and assess the impact of feature selection (FS) on the performance of IRDP models and the robustness of the model to concept drift. We conduct empirical experiments using 36 releases of 10 open-source projects. The Friedman and Nemenyi Post-hoc test results indicate that there were statistical differences between the prediction results with and without FS techniques. IRDP models trained on the data of most recent releases were not always the best models. Furthermore, the prediction models trained with carefully selected features could help reduce concept drifts.

---

## 1. Introduction

In recent years, machine learning (ML) applications have been integrated into numerous research fields related to our lives [1]. For example, ML-based prediction helps detect complex cardiac disease (e.g., congenital heart disease) that reduces the time and effort required to diagnose congenital heart diseases [2]. The implications of ML-based predictions not only enhance the medical sector but also influence the dining industry. The prediction techniques have gained much attention for the wine industry to predict the quality of wine that improves the winemaking technique [3]. Indeed, software systems are spreading through all aspects of our daily lives with the automatic vehicles and AI personal voice assistants such as Alexa, Cortana and Siri that have significant involvement in security, privacy and safety [4]. To build a reliable and high-quality software system, software quality assurance (SQA) guarantees that the developed system meets specific quality standards [5]. Furthermore, the test phase of the software development is mainly used to ensure the purpose of the software systems meets the client's requirements [6]. To assist in SQA (software testing) activities, software defect prediction (SDP) helps the software tester identify defects that help prioritize the scarce SQA resources [7]. Consequently, SDP is considered one of the predominant research topics by the SE community [8–12]. In SDP, the prediction models are built using machine learning (ML) techniques to identify defective modules or classes that are likely to be defective in a newly developed software [10, 13–16]. These prediction models assist software developers to allocate scarce testing resources such as code inspection in focusing on those modules that need special attention [13, 17]. Over the years, many ML techniques have been proposed to build the classification models for SDP [9, 18–21]. 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) [22]. 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 [23, 24]. This scenario can be categorized into two schemes [18], which are inter-project SDP and intra-project SDP. Intra-project SDP utilizes the current release to build the prediction model and testing data

---

\*Corresponding author

✉ makabir4-c@my.cityu.edu.hk (M.A. Kabir); Jacky.Keung@cityu.edu.hk (J. Keung); burak.turhan@oulu.fi (B. Turhan); kwabena.bennin@wur.nl (K.E. Bennin)

ORCID(s): 0000-0002-7136-6339 (M.A. Kabir); 0000-0002-3803-9600 (J. Keung); 0000-0003-1511-2163 (B. Turhan); 0000-0001-9140-9271 (K.E. Bennin)

from the same release of the software project, whereas inter-release defect prediction (IRDP) is a process where the historical data is retrieved from the previous release of the software project and tested on the current release of the same project, which is considered as more practical and realistic [25, 26]. In the SDP literature, IRDP is also referred to as cross-version defect prediction (CVDP) [25–27].

Recently, IRDP has attracted more interest from SE researchers due to some unique aspects [25–28]. For example, Shukla et al. [25] argued that prior release of a software project is more appropriate for building prediction models for better results. The authors alluded that the project characteristics regarding architectural design will be more or less the same across the releases. Therefore, prior release becomes the more suitable training set for predicting the defects. We were able to identify the cited research works (i.e., [18, 20, 25–33]) that studied long-running software projects for predicting defects in the IRDP context through our SDP literature review. Xu et al. [26–28] tried to mitigate the distribution difference among two consecutive releases. The studies of Amasaki [29–31] focused on the effects of cross-projects SDP approaches under IRDP. The studies [18, 20, 25, 32, 33] 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 it a critical system released over time [34]. Data generated in these environments are non-stationary [35]. The effects of non-stationary environments (NSE) are detrimental, and it poses challenges to learn from such an environment [36]. For example, data probabilistic properties change over time [37] which consequently can make a previously well-performing prediction model trained in a stationary environment become obsolete [38]. Furthermore, software projects produce several releases that appear in temporal order and become complex due to the frequent changes made in each release [27]. For instance, the current release of a software project becomes more complicated due to the frequent changes made in some modules in the prior release, which leads to the changes in data distribution among the releases. If the relationship between variables of underlying data distribution changes (i.e., concept drift [37, 39]) due to unforeseeable reasons after a stable period, the prediction model would not be reliable [40]. In such a scenario, the learning environment is unlikely to be stationary. The researchers who studied IRDP seem reluctant to consider inter-release defect datasets as non-stationary data distribution and do not investigate further by considering the temporal order of the distributions. Since each release’s data appears as a chunk in temporal order, the learning environment becomes temporal chunk-based learning. We formulate IRDP as a temporal chunk-based learning problem in which the data of different releases 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 [17, 41, 42] show that carefully selected features could achieve acceptable prediction performance in SDP. Therefore, it is essential to examine the usability of FS techniques in the IRDP context while considering temporal chunk-based learning over the releases of a software project. Furthermore, FS techniques may alleviate concept drift due to the variation in selecting features in each release of inter-release defect datasets. Therefore, our study investigates whether applying FS techniques separately on each release of the project when building IRDP classification models could improve prediction performance and examine the robustness of these trained IRDP models to concept drifts. 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 IRDP?**

### Abbreviations and acronyms

Abbreviations	Description
ML	Machine Learning
SE	Software Engineering
SQA	Software Quality Assurance
SDP	Software Defect Prediction
SE	Software Engineering
ML	Machine Learning
IRDP	Inter-Release Defect Prediction
CVDP	Cross-Version Defect Prediction
FS	Feature Selection
NSE	Non-Stationary Environments
DT	Decision Tree
KNN	K-Nearest Neighbour
LR	Logistic Regression
NB	Naïve Bayes
RF	Random Forest
CB	Chunk-Based
CFS	Correlation-based Feature Selection
CONFS	Consistency-based Feature Selection
AUTOS	Automated Feature Selection
ROC	Receiver Operating Characteristic
AUC	Area under the ROC curve
<i>pf</i>	false alarm rate

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

Based on the above RQs, we conduct experiments on 36 releases of 10 software projects provided by Madeyski and Jureczko [43, 44]. These datasets are selected because they have several releases and are widely used in previous studies of SDP [25–28, 30–33]. The datasets contain static code metrics and are used for predicting defects for inter-release at the class levels. We consider three best-performing FS techniques (correlation-based FS [45], consistency-based FS [46], and AutoSpearman [47]), five prediction models (DT, KNN, LR, NB, and RF), and three key 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 IRDP using temporal chunk-based learning. Thus, the contributions of our empirical analysis can be summarized as follows.

- Provide findings and insights which are useful to the software engineering researchers interested in improving the prediction performance of IRDP.
- Identify FS techniques that help to minimize the impact of concept drifts and improve the prediction performance of IRDP.

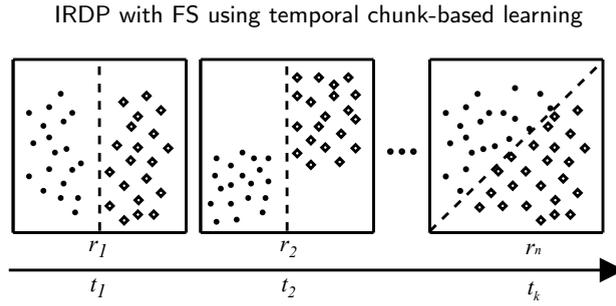
The empirical study reveals that there are statistically significant differences in performance values between the results with and without FS techniques when evaluated with AUC, Recall, and *pf*. We find that 50% of the inter-release defect datasets are drift-prone when the considered classifiers were trained on them. The FS techniques improve the prediction performance of IRDP models and minimizes the drifts from the defect datasets.

The rest of the paper is organized as follows: Section 2 formulates the problem for IRDP. Section 3 describes the methodological procedure applied to conduct the empirical study. We report the results obtained from the experiments in Section 4. Section 5 discusses the performance of our experimental results. Section 6 highlights the potential threats to validity of our experimental results. Section 7 summarizes the closely related work describing the studies about IRDP and finally, we conclude and state the future work in Section 8.

## 2. Problem Formulation

In this work, we formulate IRDP as the problem of temporal learning in which each release of the project is considered as a training dataset at each time step. The training dataset is used to build IRDP models. At each time step, the model is updated when the subsequent release arrives. Since each release’s data appears as a chunk in temporal order, the learning environment becomes temporal chunk-based learning. **An example of release-wise data chunk is shown in Figure 1. As the first data chunk at time  $t_1$  for release  $r_1$ , 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 release  $r_n$ . The term “chunk-based learning” refers to learning the data as chunk based on the “discard-after-learn” concept [48]. 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) [48]. Junsawang et al. [48] suggested learning data as a chunk at a time to overcome the impact of sensitivity of the learning data sequence. Overall, we formulate IRDP as a temporal chunk-based learning problem in which the data of different releases of a project appear at each time step. Here, the data of each release 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 IRDP, two nearest releases are considered for the defect prediction. At each time step, the defects are predicted in the next release in IRDP, bearing in mind that consecutive releases share similar characteristics, which helps to train the model accurately, as suggested by Amasaki [29–31]. Based on this setting, we conduct 26 pairs of inter-releases of the considered defect datasets. To manage the data in temporal learning environment, windowing techniques [49] have been developed to manage the data [38]. In our case, the data of each release arrives as a chunk. Therefore, a chunk-based learning strategy is more appropriate. Moreover, the release-based moving window is validated and recommended by Harman et al. [34], where window size refers to one release’s data. In our study, we consider the same strategy to build the prediction models for IRDP. In particular, the defect data is generated in such a manner so that the sequence of data  $(a_t, b_t)$  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



**Figure 1:** An example of release-wise chunk data with different classes.

examples can be produced by chunk-based (CB) settings [37, 38]. 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. [37]. Minku [50] defines concept drift and chunk-based (CB) learning as follows.

**Concept drift:** Given 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 [50];  $(a_t^{(i)}, b_t^{(i)}) \in A \times B$ ; 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.

### 3. Methodology

To answer the RQs and empirically validate the results, we conduct a systematic investigation by selecting appropriate inter-release benchmark datasets (Section 3.1), choosing best-performing FS techniques from SDP literature (Section 3.2), selecting widely-used learning algorithms (Section 3.3), evaluating the results using several performance metrics (Section 3.4), applying the robust statistical tests to obtain reliable experimental results (Section 3.5), and following a robust approach in conducting the experimental setup (Section 3.6). **To gain a comprehensive understanding of the impact of FS techniques for IRDP 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 IRDP model on the previous release and test the subsequent release within the same software project as a practical scenario. Before preparing the model, we first pre-process the data based on chronology. We split the releases from the main datasets into train and test sets based on the release's order. Thus, the datasets behave as a windowed stream containing  $v$  releases (see Figure 2). The train and test sets are processed to build the model by following the window-based operation, where each release is considered as a window. We build the prediction models by utilizing the default 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 2, we consider chunk-based learning where the data of each release 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 [51]. To answer RQ2, we conduct a hypothesis test to detect concept drift. We calculate the test statistics of two consecutive IRDP 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 3.6 discusses the motivation behind these RQs and illustrates the experimental setup. Furthermore, Figure 2 provides an overview of our empirical study.**

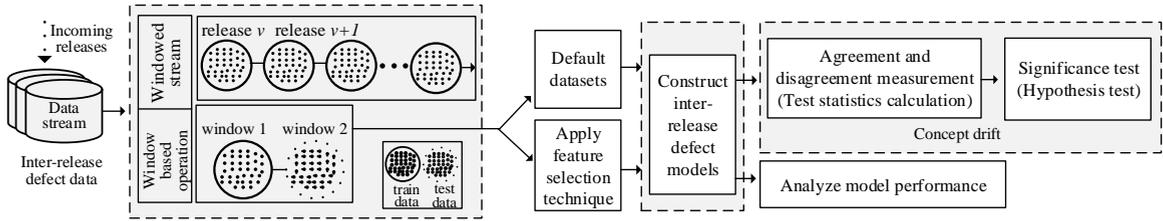


Figure 2: An overview of the design of our empirical study.

### 3.1. Benchmark datasets

We conduct our empirical investigation on a corpus of publicly available datasets containing 36 releases of 10 benchmark open-source software projects provided by Jureczko and Madeyski [43, 44], Jureczko and Spinellis [52]. These inter-release defect datasets have been widely utilized in the studies of IRDP [25–28, 30–33, 53]. We collect the corpus from the SEACRAFT repository<sup>1</sup> [54] (which was formerly known as the PROMISE repository [55]). A summary of the datasets, including releases and number of modules, is shown in Table 1. Each instance or module of these datasets includes 20 static code metrics with a labeled feature (BUG) listed in Table 2. 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. [28] observed that 40% of the modules of the subsequent releases were defective, which indicates the distribution differences. Such differences can make the releases drift-prone.

Our selected open-source software projects contain specified release that appears in chronological order. The release date of the versions is collected from version-control repositories, which are used in the study of Bangash et al. [56]. As mentioned in the Section 2, each release’s data appears as a chunk in temporal order, and the learning environment becomes chunk-based learning. Suppose that the ant project has five releases from 1.3 to 1.7 (Table 1). It appears in temporal order and creates temporal chunk-based learning environments. Note that, the project considered in our empirical study has at least three releases for constructing the scenario of IRDP.

### 3.2. 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 [17, 41, 42, 58]. 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 [17, 41, 42, 58, 59]. Ghotra et al. [41] 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 best 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. [42] on software defect datasets. Based on their experiment, they observed that among the filter-based FS techniques, correlation and consistency-based FS techniques performed best compared to other techniques. The study of Kondo et al. [17] on FS techniques for feature reduction on SDP models demonstrated that correlation and consistency-based FS techniques outperformed other techniques. Here, 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 [60]. 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 [45]. 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 [61].

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

**Table 1**  
A statistical summary of selected chronological defect datasets.

Project	release	Release 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%

Recently, Jiarpakdee et al. [47] 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 [62]. From their experimental study, they observe that AutoSpearman alleviates the correlated metrics better than other FS techniques [59]. 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 [62]. 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 IRDP using temporal chunk-based learning, we consider **all metric sets** as a baseline method that is widely used in the existing studies [26, 27].

### 3.3. Construct inter-release 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. [63] 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. [57] conduct an empirical study on 34 releases 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. [14] affirm the effectiveness of NB on NASA defect data sets. For our experimental study, we choose commonly-used five machine learning algorithms to con-

**Table 2**

List of the static code metrics used in this empirical study [57].

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

**Table 3**

Confusion matrix.

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

struct the inter-release 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.

### 3.4. Calculate model performance

For a comprehensive evaluation of the prediction models, three key 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 3) 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 3, 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. [14, 64]. 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 [65]. 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. The measures ( $pf$ , Recall, and AUC) are used to report the RQs results in our study. To make a fair-comparison with the state-of-the-art methods, we adopt F-measure ( $F\text{-measure} = \frac{2 * Precision * Recall}{Precision + Recall}$ ) where  $Precision = \frac{TP}{TP+FP}$  and G-mean ( $\sqrt{(\frac{TP}{TP+FN}) * (\frac{TN}{TN+FP})}$ ) in our study.

### 3.5. Statistical tests

The authenticity of empirical results can only be verified using statistical tests. Demsar [66] 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 [21] and Lessmann et al. [67]. Haouari et al. [18] advocated for the use of 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 [68].

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 using the non-parametric statistical test, Friedman test [69], 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 [18, 19, 21, 28, 67, 70–75]. We present the mean ranks of the methods for each performance measure in Table 5, post-hoc analysis using the Nemenyi test presented in Table 6.

### 3.6. Experimental setup

The main goal of this empirical study is not only focused on applying the FS techniques separately on each release for IRDP, 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 IRDP?

**Motivation:** The findings of prior research have revealed the benefits of adapting FS techniques to predict software defects [41, 42, 59, 76]. For example, Ghotra et al. [41] studied the impact of FS techniques on SDP. Still, the conclusion derived from their study is not compatible with the scenario of IRDP 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 release 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 IRDP models through the RQ1.

**Approach:** To assess whether FS techniques impact IRDP, we compute and analyze the prediction performance of IRDP 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 inter-release defect dataset). In the IRDP scenario, we utilize a temporal chunk-based learning strategy using the releases' order. Then we train a model using a release and test it with subsequent release defect datasets. We use each metric set generated by a FS technique as input to the studied five defect prediction models. We adopt three key performance indicators. We consider 10 benchmark open-source software projects containing 36 releases. As mentioned, the experiment is conducted in the inter-release 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 performance values  $\times$  5 classifiers  $\times$  3 performance measures). For 3 FS techniques, our experiment yields (390 $\times$ 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 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 [38, 77]. 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. [78]. Based on our previous works [35, 79], 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 releases, 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 [80, 81] 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 release's data exhibits a statistically significant prediction performance difference for identifying defective or non-defective modules of the models of consecutive release data, then a concept drift exists.

In this experimental study, we split the releases based on project chronology (also referred release-based moving window), as prior work applied a similar strategy (i.e., split the releases). For example, Xu et al. [26] consider splitting the releases of the defect datasets and tried to mitigate the distribution differences between two consecutive releases. The experimental setup is as follows: (1) We collect all the available releases from the sources and restructure them into a time frame. The windowing technique manages the releases whereas two time period contains two consecutive releases. (2) We train the prediction models using the release's data from the period and test the model using the release's data for the next period. (3) Similar to prior work [80, 81], we calculate the test statistics of two consecutive IRDP 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 release's defect data shows a significant difference on the new release'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. [80]. Nishida et al. [82] 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 release is small (see Table 1). In particular, we conduct the statistical analysis on the test statistics acquired from the two windows. We portray the drift detection results in Figure 7 A for default datasets. After that, we conduct the test again on metric sets obtained by performing the FS techniques separately for every release and show the results in Figure 7 B.

## 4. 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 IRDP?**

**Results:** To assess the abilities of FS techniques on IRDP in the temporal learning environment, we compare the performance results across each performance measure and exhibit the experimental results to answer RQ1. Table 4 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.

From Table 4, AUTOS FS technique 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 release 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 IRDP.

**Table 4**

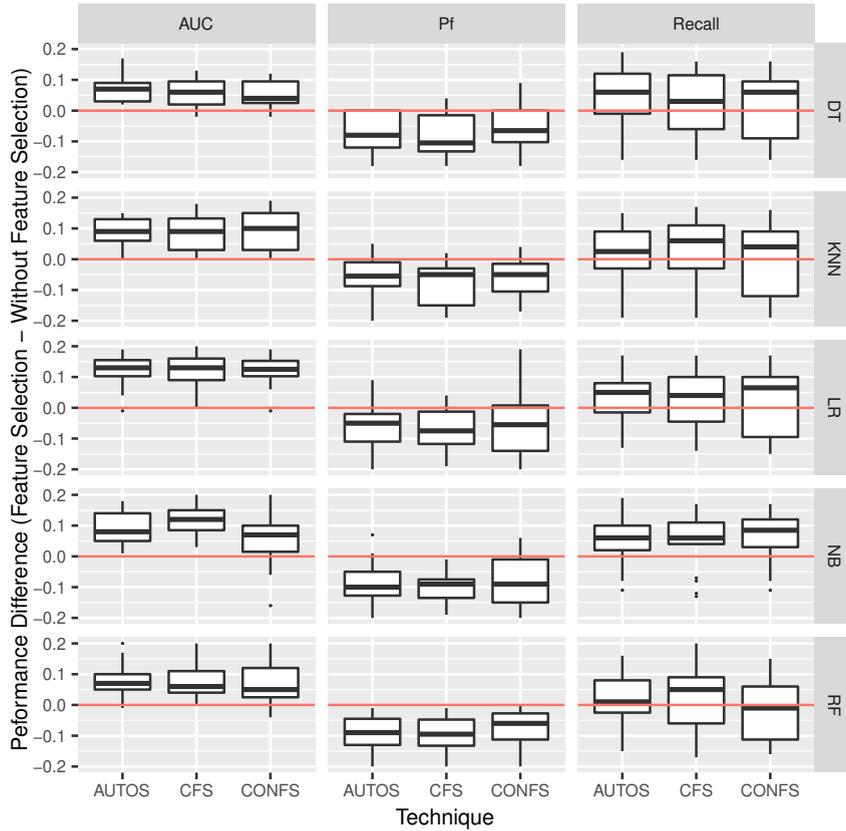
Summary of prediction results of inter-release defect prediction across the algorithms.

	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	<b>0.00</b>	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	<b>0.72</b>	<b>0.90</b>	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	<b>0.00</b>	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	<b>0.00</b>	0.70	0.87	0.55
CONFS-NB	0.75	1.00	0.00	0.50	0.94	<b>0.00</b>	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	<b>0.00</b>	0.61	0.78	0.50
CONFS-KNN	0.68	1.00	0.00	0.48	0.98	<b>0.00</b>	0.65	0.85	0.50
CONFS-LR	0.68	1.00	0.00	0.51	1.00	<b>0.00</b>	0.69	0.87	0.55
AUTOS-NB	<b>0.81</b>	1.00	0.38	0.53	0.91	0.06	<b>0.72</b>	0.89	0.49
AUTOS-RF	0.71	1.00	0.15	0.45	0.99	0.02	0.71	<b>0.90</b>	0.52
AUTOS-DT	0.73	1.00	0.00	0.38	0.92	<b>0.00</b>	0.70	<b>0.90</b>	0.50
AUTOS-KNN	0.68	1.00	0.01	0.44	1.00	<b>0.00</b>	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

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, AUTOS-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 [18, 72, 83]. Especially for a critical software system, where each release 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 IRDP 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, AUTOS-RF, and AUTOS-DT, respectively. However, in Table 4, 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 IRDP models in such a realistic environment.

Figure 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 this figure, the distributions centered at zero indicate that the performance measures are not impacted negatively or positively by FS techniques for IRDP. Figure 3 shows that when applying FS techniques, 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% IRDP models varies from 8% to 37% (i.e., the values of 1<sup>st</sup>-3<sup>rd</sup> quantiles). This result indicates that FS techniques tend to have a positive impact on AUC when they are employed to IRDP 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 IRDP models. As mentioned above, FS-based IRDP 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% IRDP models varies from -5% to 97% (i.e., min-max). We also observe that the performance distribution of the Recall measure for 75% IRDP models varies from -3% to 22% (i.e., the values of 1<sup>st</sup>-3<sup>rd</sup> quantiles). Overall, from the Figure 3, it can be seen that the FS techniques have a positive impact on the performance for correctly classifying



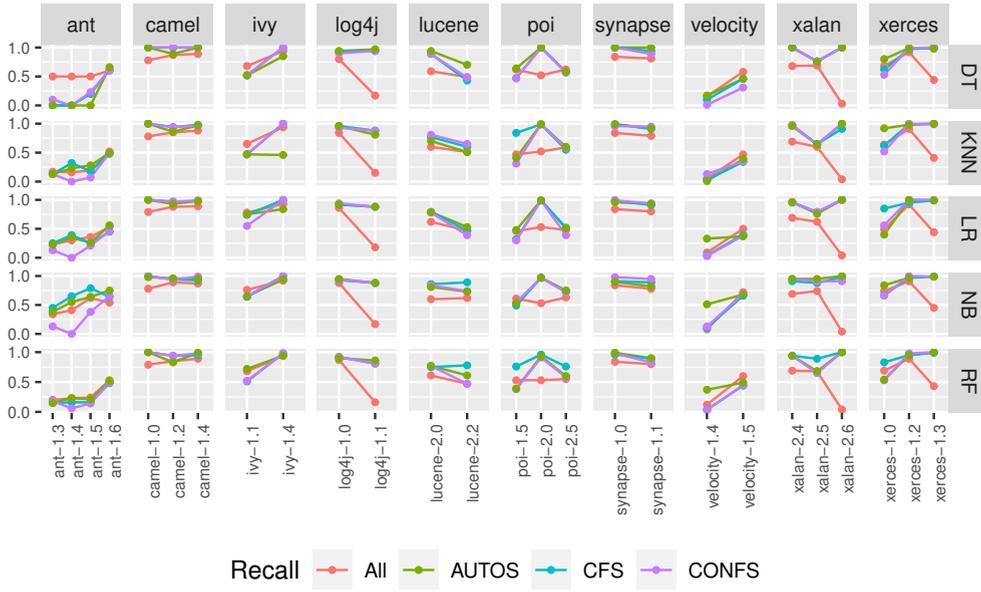
**Figure 3:** The performance difference when employing FS techniques to inter-release defect prediction models. A red line indicates no performance improvement.

the inter-release defective modules. Note that,  $pf$  is measured by the total number of non-defective modules predicted as defective. Interestingly, AUTOS, CFS, and CONFS FS techniques decrease  $pf$  and increase Recall, which implies the positive impact of FS techniques in classifying the inter-release defective modules.

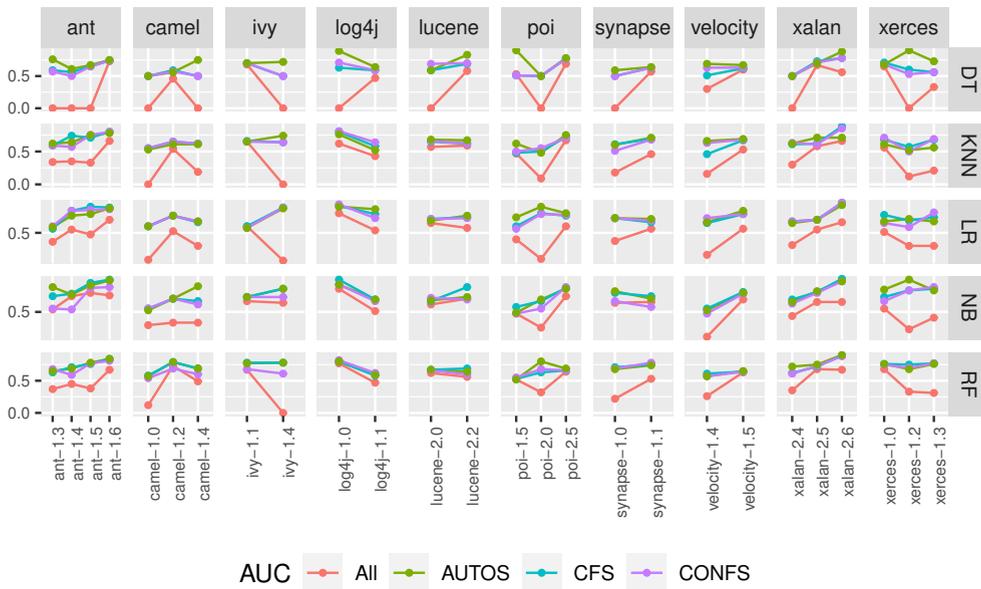
The prediction performance values across the releases of the considered datasets are depicted in the Figures 4 to 6. We observe that the Recall and AUC scores do not always increase over the releases with time in the temporal learning environment (Figure 4 and 5), which means that the latest release does not always give the highest values of Recall and AUC. In some cases, the initial releases are more suitable for correctly classifying the defective modules. For example, as a training set for an IRDP model, synapse-1.0 is able to classify the defective modules correctly (i.e., high Recall) than synapse-1.1. For  $pf$  values, initial releases across the considered datasets are more effective than have low rate of  $pf$  (Figure 6). We also see that the IRDP models remain competitive throughout the releases in the temporal chunk-based learning environment. Another observation can be drawn from the Figures 4 to 6: the IRDP models trained on the most recent releases do not always yield the best inter-release defect predictors. This observation agrees with the results of Harman et al. [34]. 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. [34], dynamic adaptive prediction systems [84] could be employed to recognize the best model. Point to be noted that the ability to underline the best models among the IRDP models is out of the scope of our study.

To achieve reliability and statistical validation, Demsar [66] 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 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 (5 classifiers  $\times$  4 types of

### IRDP with FS using temporal chunk-based learning



**Figure 4:** IRDP models performance across the releases of the considered datasets for Recall.



**Figure 5:** IRDP models performance across the releases of the considered datasets for AUC.

feature sets (i.e., obtained from ALL, CFS, CONFS, and AUTOS)) over the defect datasets in our selection. Therefore, the degree of freedom is 19. The computed Friedman test 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.

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 5, regarding the Post-hoc test results displayed in Table 6. In Table 6, 0

IRDP with FS using temporal chunk-based learning

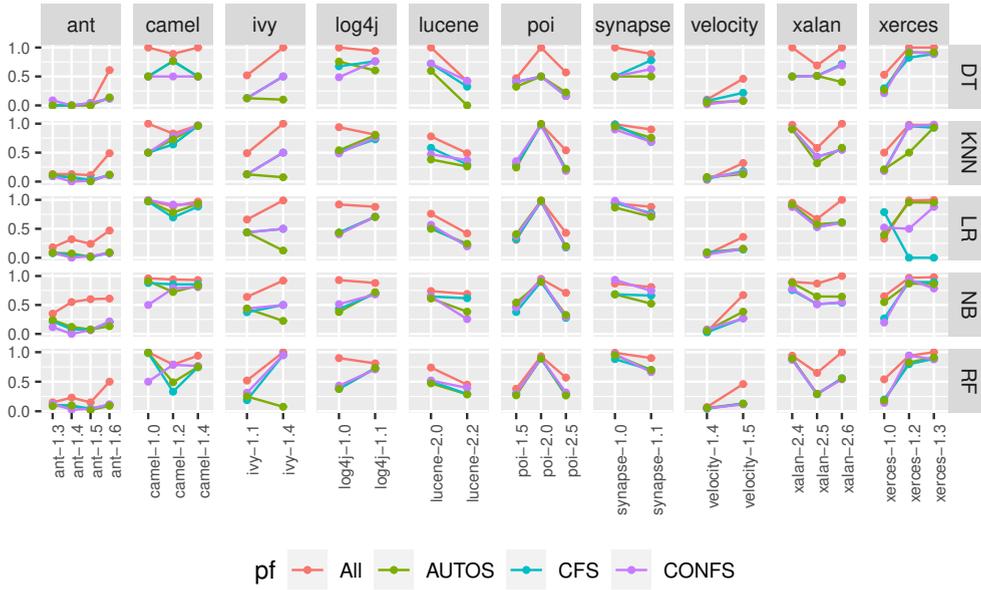


Figure 6: IRDP models performance across the releases of the considered datasets for *pf*.

Table 5

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 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	<b>4.8</b>	11.8	10.2	10.6	14.0	10.5	13.2	10.0	11.6
<i>pf</i>	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	<b>5.4</b>	8.8	10.1
AUC	5.5	5.2	3.6	3.3	<b>2.3</b>	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

refers to there is no statistical difference between the pair of classifiers otherwise, there is a statistical difference in terms of P for *pf*, R for Recall, and A for AUC. From the pairwise comparison, 71 is found to be statistically significant for *pf* and Recall, respectively, and 67 is found to be statistically significant for AUC.

Results drawn from Tables 5 and 6, 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 (Table 6). 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 IRDP when the learning environment is temporal chunk-based learning.

**Table 6**

The post-hoc analysis results. 0 refers to there is no statistical difference between the pair of classifiers otherwise, there is a statistical difference in terms of P for  $pf$ , R for Recall, and A for AUC. 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	A	0	0	-	-	-	-	-
AUTOS-RF	PRA	PRA	PRA	PRA	PRA	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	-	-	-
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

### Summary

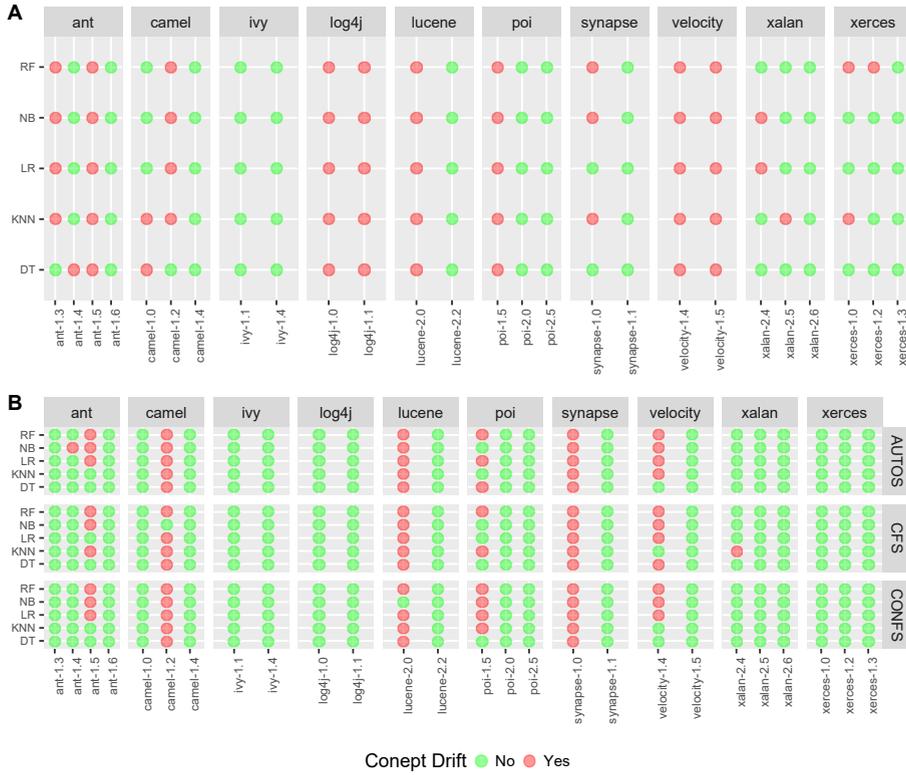
When there is a need to identify (predict) the most defect-prone modules for a critical software system, we require to choose the prediction models that achieve high Recall [18, 72, 83]. Especially for IRDP, where the software system becomes more critical in each release due to refactoring and organizational needs, 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 IRDP 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 IRDP models trained on the most recent releases do not always yield the best inter-release defect predictors (Figures 4 to 6).

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

**Results:** Figure 7 represents the status of concept drift for each release of the defect datasets. Figure 7 A depicts the detection of drift while considered all feature sets, and Figure 7 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 inter-release defect data. For example, the NB and RF models are probably more sensitive to the specific pattern in each release of the inter-release 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 release, the considered techniques help eliminate drift from the datasets.

Figure 8 shows the drift detection in percentage when FS techniques are applied to the datasets. In this figure, ALL represents to all features sets used to detect the percentage of concept drift in the defect datasets. Using all the feature sets, the identification rate of concept drift is much higher than the drift detection rate with FS techniques. We observe that CONFS FS technique 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 FS technique with DT, it achieves noteworthy improvement on average for Recall 71% and for AUC 61%. By considering the IRDP as temporal chunk-based learning, we observe that while the FS techniques are applied separately on each release of the project, it eliminated the drift

## IRDP with FS using temporal chunk-based learning



**Figure 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 releases of the project ( $p$ -value < 0.05 in Fisher's Exact Test); while a green symbol indicates otherwise ( $p$ -value  $\geq$  0.05 in Fisher's Exact Test).

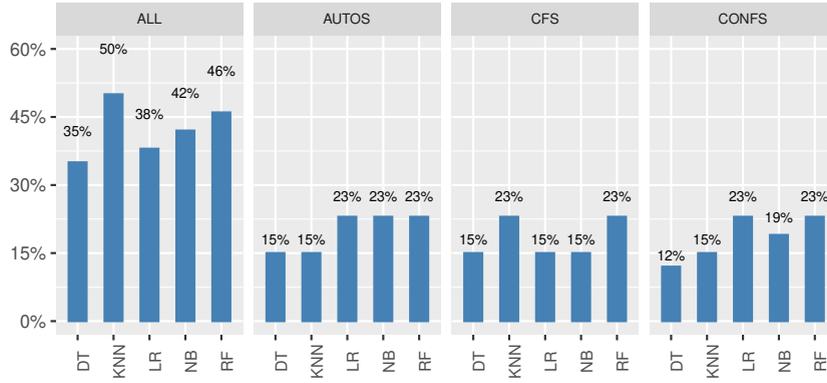
from all of the releases of the ivy, log4j, and xerces projects, whereas most of the releases contain drift in the default datasets. This observation guides us to recommend practitioners to consider the FS technique as an *attention technique* for concept drift adaptation [85] in the defect datasets 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 IRDP 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 inter-release 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 inter-release defect datasets are drift-prone while employed the classifiers only. The FS techniques help mitigate (i.e., drift elimination difference) the drift up to 23% (35%-12%) by the CONFS-DT technique and obtain 70% Recall on average in the IRDP scenario. Here, the drift elimination rate refers to the difference between the original drift rate and drift rate obtained after applying FS techniques with the specific classifier (identified 12% drift by CONFS FS technique with DT). Researchers and practitioners should consider concept drift for IRDP while learning environment as a temporal chunk-based temporal learning. For the concept drift adaptation process [85], the FS technique could be used as an attention technique.

## 5. Discussion

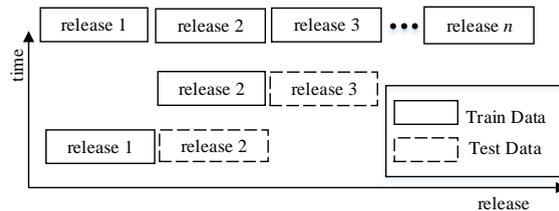
In this section, we first discuss results obtained from the RQ2 related to concept drift and then present a comparative discussion to verify the discriminating capacity of the FS techniques with the state-of-the-art methods for the IRDP performance concerned with RQ1.



**Figure 8:** The percentage of concept drift identification when applying FS techniques. Here, ALL refers to all feature sets considered for detecting drift in percentage and the drift detection in percentage exhibited while considered AUTOS, CFS, and CONFS. Here, a low value represents the best score.

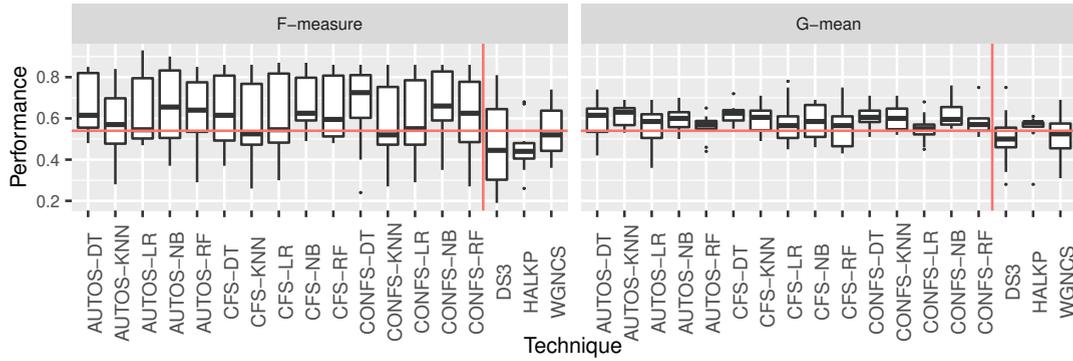
### 5.1. Is continuously retraining a model actionable enough?

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, 86–88]. To eliminate concept drift, the most common remedy in such a situation is to update the existing model [85]. 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 [85]. We elaborate on how continuously a model is updated and how an FS technique helps eliminate



**Figure 9:** Release-based moving window, where window size refer to one release's defect data.

the drift while updating the model and calibrate the model to be more actionable using the release-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 [89–97]. In SDP, Harman et al. [34] conducted a study on 8 Hadoop releases where chronology played an important role in predicting defect over time. Like Harman et al. work, in our empirical study, a model is retrained using the next release's defect data. As mentioned above, a release-based moving window technique is adopted to update the model (see Figure 9). We do not use all the historical data of previous releases to retrain the model. Furthermore, updating the prediction models using all the data of previous releases may not lead to better prediction performance [98]. Please note that we split the releases from the main datasets into train and test sets based on the release's order (i.e., release-based splitting). In Figure 7 A, we exhibit the drifts detection for each release of the datasets. Figure 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 inter-release defect datasets is *model-aware*, i.e., the percentage of drift detection varies from model to model. Figure 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 tech-



**Figure 10:** Comparison of FS-based IRDP 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.

niques 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. Furthermore, they could also consider adding FS techniques as an attention technique in concept drift adaptation.

## 5.2. General discussion

To verify the discriminating capacity of the considered FS techniques in the temporal chunk-based learning environment, we compare the IRDP 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 [26] and HALKP [27], and one hybrid method, WGNCS [99], considered. Briefly, Xu et al. [26] 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) [100] and Peter Filter (PF) [101]. In [27], 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. [99] 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 IRDP as depicted in Figure 10. To cope with chunk-based learning, we consider the average value of each inter-release pair for each project as the indicator value of the project. For instance, camel project contains three inter-release pairs (camel-1.0 → camel-1.2, camel-1.2 → camel-1.4, camel-1.4 → camel-1.6) (see Table 1). 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 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 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 IRDP models performed better than the considered state-of-the-art methods, as shown by the red line in Figure 10. Furthermore, RF, LR, and NB were used as baseline methods in the studies of SDP (e.g., [74, 102]). In RQ1, these baseline methods are compared with the FS-based methods that performed best in the IRDP chunk-based learning environment. Results computed for F-measure and G-mean are made available in additional material [103]. From the results of RQs, we conclude that FS techniques do improve the performance (Recall,  $pf$ , and AUC) of IRDP. 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 classification models in the temporal learning environment. Compared with the state-of-the-art methods, FS-based methods demonstrate better performance based on the experimental results. The recently developed FS technique, AutoSpearman that validated with the software defect datasets [47], performs best to identify (predict) the most defect-prone modules for the considered critical software systems. From the results, we also observe that the IRDP models trained on the most recent releases do not always yield the best predictors. Even the IRDP models are more competitive throughout the releases. In some cases, the prediction results obtained by using first-release data are more suitable. These observations agree with the findings of Harman et al. [34], 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 associated effects from the temporal learning environment.

## 6. 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 releases 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 IRDP. However, the considered benchmark projects were adopted by prior IRDP studies where static code metrics performed well [26–28]. Besides, Basili et al. [104] 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 [17, 41, 42, 58, 59]. The impacts of other FS techniques deserve further investigation. Five commonly used classifiers are used to build inter-release prediction models. Different classifiers 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.

## 7. Related work

In recent years, IRDP has emerged and drawn significant attention because of its applicability in practice. In this defect prediction scenario, the distribution of the dataset shares similar characteristics due to software project context, architectures, and development settings [105]. To predict defects within the same projects, most researchers attempted to build ML models merging the datasets of all prior releases and test the model using the latest release. In this section, we provide the related work of IRDP.

Xu et al. [26] tried to mitigate the distribution differences between the two releases by utilizing dissimilarity-based sparse subset selection (DS3). The study was conducted on 56 releases of 15 projects for IRDP. They discovered that distribution differences degraded the prediction performance of all the models and attempted to mitigate the distribution difference between two adjacent releases by sparse subset selection. However, it needs further effort to level the representative modules, as claimed by Xu et al. [28]. To overcome this issue, Xu et al. [28] proposed a two-stage training subset selection method by conducting a large-scale experiment on 50 releases 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. Bennin et al. [32] 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. [28] affirmed that the results obtained are not statistically significant among all prediction models. Xu et al. [27] proposed a two-phase framework by combining hybrid learning strategy and kernel principal component analysis (KPCA) where

features are selected from the current release and merged with the previous release 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 [29–31] focused on the effects of CPDP approaches under IRDP. These studies were conducted by utilizing multiple older releases. The empirical results indicated that some of the CPDP models could improve the performance of IRDP. Shukla et al. [25] considered IRDP as a multi-objective problem (i.e., maximizing recall by minimizing cost and misclassification) and conducted experiments on 30 releases from 11 projects. They observed that multi-objective logistic regression was more effective than single-objective methods. Yang and Wen [33] investigated ridge and lasso regression to conduct IRDP by addressing it as a problem of multicollinearity. The experiments were conducted on 41 releases from 11 projects. They noticed that ridge regression performed better compared with the linear and negative binomial regression. Lu et al. [106] compacted the techniques of dimensional reduction and active learning to address the issue of IRDP. The experiments were conducted on the three successive releases of the Eclipse project. From the experimental results, they observed that dimensional reduction performs better than filter and wrapper-based FS techniques. Zhang et al. [105] formulated IRDP as a problem of data selection. They addressed differences in data distribution and class overlapping to solve IRDP. Gao et al. [107] conducted an empirical study based on a logistic regression model with complex network features. They observed that complex network features had the ability to predict defects for inter releases than the merged feature sets. Zhang et al. [99] formulated a robust IRFP model by considering class imbalance, feature subset selection, feature matching, and convolutional Neural Network with SVM. They conducted experiments on 32 releases of 45 software projects and observed the satisfied defect prediction performance. Yao et al. [108] proposed transition class ratio and static metric category number evaluation metrics for defect prediction among the releases. The experimental study was conducted on 36 releases of 10 open-source software projects. They observed better performance in the evolution metrics than traditional static metrics. Harman et al. [34] conducted an investigation on 8 Hadoop releases 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 release are not always the best predictors. **In summary, previous research studies developed the IRDP models by considering the data distribution stationary (i.e., no concept drift). At the same time, they did not consider that the software companies grow with time, and the software projects experience several releases after a stable period, appearing in temporal order and making non-stationary data distribution. Thereby, the relationship between data variables changes (i.e., concept drift) over time due to a dynamic software development environment (e.g., refactoring and organizational changes). Therefore, the developed well-trained IRDP models may be obsolete after a certain time point and could be deceptive if the distribution changes over time, as affirmed by Dong et al.[78]. Different from previous research studies, we employ chunk-based learning based on the discard-after-learn concept formulated in Section 2 for IRDP. We examine the impact of correlation and consistency-based, and AutoSpearman FS techniques on the IRDP models trained on the temporal order of the inter-release defect datasets for predicting defects that face concept drift. We apply FS techniques separately on each release of the project when creating IRDP models to check whether it improved prediction performance and examine the robustness of the model to concept drift, which can well compensate for the deficiency that occurred due to concept drifts.**

## 8. Conclusions

This study focuses on the empirical assessment of the effectiveness of feature selection (FS) methods for inter-release defect prediction (IRDP). The impact of FS techniques on IRDP 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-release defect datasets. Additionally, the effect of concept drift in IRDP 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 IRDP 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 IRDP models.

We conducted a comprehensive empirical study on 36 releases, 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 IRDP 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 IRDP performance (AUC, Recall, and  $pf$ ) of the IRDP models across all the considered datasets. The IRDP 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 releases due to refactoring and organization demands, a prediction model with high Recall and low  $pf$  is recommended [18, 72, 83]. 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 IRDP models.
- In the temporal chunk-based learning environment, the IRDP models are build based on the release-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 IRDP models trained on the most recent releases do not always yield the best inter-release defect predictors. This result agrees with the observation of Harman et al. [34]. However, finding the best models in the IRDP scenario is beyond this empirical study.
- Compared with the state-of-the-art methods, CFS with NB and CONFES 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 CONFES 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 CONFES with LR. The most prediction performance of the FS-based IRDP 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 [85]. We observe that continuously re-training 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 IRDP models to eliminate concept drifts and improve prediction performance in a chunk-based temporal learning environment for software engineering researchers and practitioners.

Further extensions of our work involve other open-source software projects with more releases, utilizing the results obtained from this empirical study. We also intend to extend our current study by exploring other feature selection techniques. By the experimental evaluation of Artificial Immune Systems conducted by Haouari et al. [18], we plan to investigate such systems for concept drift detection and elimination.

## Acknowledgement

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No.11208017) and the research funds of City University of Hong Kong (7005028, 7005217), and the Research Support Fund by Intel (9220097), and funding supports from other industry partners (9678149, 9440227, 9440180 and 9220103).

## References

- [1] Henriikka Vartiainen, Matti Tedre, and Teemu Valtonen. Learning machine learning with very young children: Who is teaching whom? *International Journal of Child-Computer Interaction*, 25:100182, 2020.
- [2] Shih-Hsin Chen, Chun-Wei Wang, I-Hsin Tai, Ken-Pen Weng, Yi-Hui Chen, and Kai-Sheng Hsieh. Modified yolov4-densenet algorithm for detection of ventricular septal defects in ultrasound images. *International journal of interactive multimedia and artificial intelligence*, In Press(In Press):1–8, 2021.

- [3] Terry Hui-Ye Chiu, Chienwen Wu, and Chun-Hao Chen. A generalized wine quality prediction framework by evolutionary algorithms. *International journal of interactive multimedia and artificial intelligence*, In Press(In Press):1–11, 2021.
- [4] Matthew B. Hoy. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88, 2018.
- [5] Fabiano Pecorelli, Fabio Palomba, and Andrea De Lucia. The relation of test-related factors to software quality: A case study on apache systems. *Empirical Software Engineering*, 26(2):1–42, 2021.
- [6] Hui Xiao, Minhao Cao, and Rui Peng. Artificial neural network based software fault detection and correction prediction models considering testing effort. *Applied Soft Computing*, 94:106491, 2020.
- [7] Satya Pradhan, Venky Nanniyur, and Pavan K. Vissapragada. On the defect prediction for large scale software systems – from defect density to machine learning. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 374–381, 2020.
- [8] H. Tu, Z. Yu, and T. Menzies. Better data labelling with emblem (and how that impacts defect prediction). *IEEE Transactions on Software Engineering*, pages 1–1, 2020.
- [9] S. Hosseini, B. Turhan, and D. Gunarathna. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, 45(2):111–147, Feb 2019.
- [10] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.
- [11] Zhongbin Sun, Jingqi Zhang, Heli Sun, and Xiaoyan Zhu. Collaborative filtering based recommendation of sampling methods for software defect prediction. *Applied Soft Computing*, 90:106163, 2020.
- [12] Liu Xi, Li Haifeng, and Xie Xuyang. Intelligent radar software defect prediction approach and its application. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 32–37, 2020.
- [13] Kwabena Ebo Bennin, Jacky W Keung, and Akito Monden. On the relative value of data resampling approaches for software defect prediction. *Empirical Software Engineering*, 24(2):602–636, 2019.
- [14] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, Jan 2007.
- [15] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. Machine learning based methods for software fault prediction: A survey. *Expert Systems with Applications*, 172:114595, 2021.
- [16] Zhongbin Sun, Junqi Li, Heli Sun, and Liang He. Cfps: Collaborative filtering based source projects selection for cross-project defect prediction. *Applied Soft Computing*, 99:106940, 2021.
- [17] Masanari Kondo, Cor-Paul Bezemer, Yasutaka Kamei, Ahmed E Hassan, and Osamu Mizuno. The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, 24(4):1925–1963, 2019.
- [18] Ahmed Taha Haouari, Labiba Souici-Meslati, Fadila Atil, and Djamel Meslati. Empirical comparison and evaluation of artificial immune systems in inter-release software fault prediction. *Applied Soft Computing*, 96:106686, 2020.
- [19] S. Herbold, A. Trautsch, and J. Grabowski. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Transactions on Software Engineering*, 44(9):811–833, Sep. 2018.
- [20] Kapil Juneja. A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation. *Applied Soft Computing*, 77:696–713, 2019.
- [21] Ruchika Malhotra. An empirical framework for defect prediction using machine learning techniques with android software. *Applied Soft Computing*, 49:1034–1050, 2016.
- [22] Santosh S Rathore and Sandeep Kumar. An empirical study of some software fault prediction techniques for the number of faults prediction. *Soft Computing*, 21(24):7417–7434, 2017.
- [23] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, pages 1–1, 2021.
- [24] Ruchika Malhotra and Hitendra Singh Yadav. An improved cnn-based architecture for within-project software defect prediction. In V. Sivakumar Reddy, V. Kamakshi Prasad, Jiacun Wang, and K. T. V. Reddy, editors, *Soft Computing and Signal Processing*, pages 335–349, Singapore, 2021. Springer Singapore.
- [25] Swapnil Shukla, T Radhakrishnan, K Muthukumar, and Lalita Bhanu Murthy Neti. Multi-objective cross-version defect prediction. *Soft Computing*, 22(6):1959–1980, 2018.
- [26] Zhou Xu, Shuai Li, Yutian Tang, Xiapu Luo, Tao Zhang, Jin Liu, and Jun Xu. Cross version defect prediction with representative data via sparse subset selection. In *Proceedings of the 26th Conference on Program Comprehension*, pages 132–143, 2018.
- [27] Z. Xu, J. Liu, X. Luo, and T. Zhang. 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)*, pages 209–220, March 2018.
- [28] Zhou Xu, Shuai Li, Xiapu Luo, Jin Liu, Tao Zhang, Yutian Tang, Jun Xu, Peipei Yuan, and Jacky Keung. Tstss: A two-stage training subset selection framework for cross version defect prediction. *Journal of Systems and Software*, 154:59 – 78, 2019.
- [29] 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, page 93–96, New York, NY, USA, 2017. Association for Computing Machinery.
- [30] 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, page 32–41, New York, NY, USA, 2018. Association for Computing Machinery.
- [31] Sousuke Amasaki. Cross-version defect prediction: use historical data, cross-project data, or both? *Empirical Software Engineering*, pages 1–23, 2020.
- [32] K. E. Bennin, K. Toda, Y. Kamei, J. Keung, A. Monden, and N. Ubayashi. Empirical evaluation of cross-release effort-aware defect prediction models. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 214–221, Aug 2016.

- [33] X. Yang and W. Wen. Ridge and lasso regression models for cross-version defect prediction. *IEEE Transactions on Reliability*, 67(3):885–896, Sep. 2018.
- [34] Mark Harman, Syed Islam, Yue Jia, Leandro L. Minku, Federica Sarro, and Komsan Srivisut. Less is more: Temporal fault predictive performance over multiple hadoop releases. In Claire Le Goues and Shin Yoo, editors, *Search-Based Software Engineering*, pages 240–246, Cham, 2014. Springer International Publishing.
- [35] M. A. Kabir, J. W. Keung, K. E. Bennin, and M. Zhang. Assessing the significant impact of concept drift in software defect prediction. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 53–58, Jul 2019.
- [36] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.
- [37] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, Nov 2015.
- [38] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132 – 156, 2017.
- [39] Burak Turhan. On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering*, 17(1-2):62–74, 2012.
- [40] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, Dec 2019.
- [41] 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)*, pages 146–157, May 2017.
- [42] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia. The impact of feature selection on defect prediction performance: An empirical comparison. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–320, Oct 2016.
- [43] 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, New York, NY, USA, 2010. Association for Computing Machinery.
- [44] Lech Madeyski and Marian Jureczko. Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal*, 23(3):393–422, 2015.
- [45] Manoranjan Dash and Huan Liu. Consistency-based search in feature selection. *Artificial Intelligence*, 151(1):155 – 176, 2003.
- [46] Manoranjan Dash, Huan Liu, and Hiroshi Motoda. Consistency based feature selection. In Takao Terano, Huan Liu, and Arbee L. P. Chen, editors, *Knowledge Discovery and Data Mining. Current Issues and New Applications*, pages 98–109, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [47] 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)*, pages 92–103, Sep. 2018.
- [48] 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. *PLOS ONE*, 14(9):1–20, 09 2019.
- [49] L. Zhang, J. Lin, and R. Karim. Sliding window-based fault detection from high-dimensional data streams. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(2):289–303, Feb 2017.
- [50] Leandro L Minku. Transfer learning in non-stationary environments. In *Learning from Data Streams in Evolving Environments*, pages 13–37. Springer, 2019.
- [51] R Core Team et al. R: A language and environment for statistical computing, 2013.
- [52] Marian Jureczko and Diomidis Spinellis. Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pages 69–81, 2010.
- [53] Lucija Šikić, Petar Afrić, Adrian Satja Kurdija, and Marin Šilić. Improving software defect prediction by aggregated change metrics. *IEEE Access*, 9:19391–19411, 2021.
- [54] The seacraft repository of empirical software engineering data, 2017.
- [55] The promise repository of empirical software engineering data, 2015.
- [56] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of cross-project defect prediction models. *Empirical software engineering : an international journal*, 2020.
- [57] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, 59:170 – 190, 2015.
- [58] 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*, New York, NY, USA, 2014. Association for Computing Machinery.
- [59] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering*, pages 1–49, 2020.
- [60] Mark Andrew Hall. Correlation-based feature selection for machine learning. *phd dissertation*, 1999.
- [61] Piotr Romanski, Lars Kotthoff, and Maintainer Lars Kotthoff. Package 'fselector'. URL <http://cran.r-project.org/web/packages/FSelector/index.html>, 2013.
- [62] 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)*, pages 711–711, Sep. 2018.
- [63] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov 2012.
- [64] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald. Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'". *IEEE Transactions on Software Engineering*, 33(9):637–640, 2007.

- [65] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.
- [66] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.
- [67] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, July 2008.
- [68] Peter Bjorn Nemenyi. *Distribution-free multiple comparisons*. Princeton University, 1963.
- [69] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.
- [70] Lina Gong, Shujuan Jiang, Rongcun Wang, and Li Jiang. Empirical evaluation of the impact of class overlap on software defect prediction. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 698–709, 2019.
- [71] Zhou Xu, Jin Liu, Xiapu Luo, Zijiang Yang, Yifeng Zhang, Peipei Yuan, Yutian Tang, and Tao Zhang. Software defect prediction based on kernel pca and weighted extreme learning machine. *Information and Software Technology*, 106:182–200, 2019.
- [72] Yue Jiang, Bojan Cukic, and Yan Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595, 2008.
- [73] Zhiqiang Li, Xiao-Yuan Jing, Fei Wu, Xiaoke Zhu, Baowen Xu, and Shi Ying. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Automated Software Engineering*, 25(2):201–245, 2018.
- [74] Jaechang Nam and Sunghun Kim. Clami: Defect prediction on unlabeled datasets (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 452–463, 2015.
- [75] Marco D’Ambros, Michele Lanza, and Romain Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.
- [76] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz. Detecting fault modules applying feature selection to classifiers. In *2007 IEEE International Conference on Information Reuse and Integration*, pages 667–672, 2007.
- [77] Joao Gama. *Knowledge discovery from data streams*. CRC Press, 2010.
- [78] F. Dong, J. Lu, K. Li, and G. Zhang. Concept drift region identification via competence-based discrepancy distribution estimation. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–7, Nov 2017.
- [79] M. A. Kabir, J. W. Keung, K. E. Bennin, and M. Zhang. 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)*, pages 882–891, 2020.
- [80] Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. Concept drift detection based on fisher’s exact test. *Information Sciences*, 442-443:220 – 234, 2018.
- [81] Osama A. Mahdi, Eric Pardede, Nawfal Ali, and Jinli Cao. Fast reaction to sudden concept drift in the absence of class labels. *Applied Sciences*, 10(2), 2020.
- [82] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *International conference on discovery science*, pages 264–269. Springer, 2007.
- [83] Gagatay Catal and Banu Diri. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040 – 1058, 2009.
- [84] M. Harman, E. Burke, J. A. Clark, and X. Yao. Dynamic adaptive search based software engineering. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, 2012.
- [85] João Gama, Indrundefined Žilobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), March 2014.
- [86] J. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein. Tracking concept drift of software projects using defect prediction quality. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 51–60, May 2009.
- [87] Jayalath Ekanayake, Jonas Tappolet, Harald C Gall, and Abraham Bernstein. Time variance and defect prediction in software projects. *Empirical Software Engineering*, 17(4-5):348–389, 2012.
- [88] K. E. Bennin, N. b. Ali, J. Börstler, and X. Yu. 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)*, pages 53–59, 2020.
- [89] 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, page 69–78, New York, NY, USA, 2012. Association for Computing Machinery.
- [90] 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, page 446–456, New York, NY, USA, 2014. Association for Computing Machinery.
- [91] Leandro L Minku and Xin Yao. Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models. *Automated Software Engineering*, 24(3):499–542, 2017.
- [92] Chris Lokan and Emilia Mendes. Investigating the use of moving windows to improve software effort prediction: a replicated study. *Empirical Software Engineering*, 22(2):716–767, 2017.
- [93] Chris Lokan and Emilia Mendes. Investigating the use of duration-based moving windows to improve software effort prediction: A replicated study. *Information and Software Technology*, 56(9):1063 – 1075, 2014. Special Sections from “Asia-Pacific Software Engineering Conference (APSEC), 2012” and “Software Product Line conference (SPLC), 2012”.
- [94] 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, page 136–145, Swindon, GBR, 2008. BCS Learning & Development Ltd.
- [95] Chris Lokan and Emilia Mendes. Investigating the use of duration-based moving windows to improve software effort prediction. In *Proceed-*

- ings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01, APSEC '12, page 818–827, USA, 2012. IEEE Computer Society.
- [96] C. Lokan and E. Mendes. Investigating the use of chronological split for software effort estimation. *IET Software*, 3(5):422–434, October 2009.
- [97] 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, page 111–122, USA, 2009. IEEE Computer Society.
- [98] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. 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, page 480–490, New York, NY, USA, 2018. Association for Computing Machinery.
- [99] Nana Zhang, Shi Ying, Weiping Ding, Kun Zhu, and Dandan Zhu. Wgnccs: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation. *Information Sciences*, 570:545–576, 2021.
- [100] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5):540–578, 2009.
- [101] Fayola Peters, Tim Menzies, and Andrian Marcus. Better cross company defect prediction. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 409–418, 2013.
- [102] Feng Zhang, Quan Zheng, Ying Zou, and Ahmed E. Hassan. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 309–320, 2016.
- [103] Md Alamgir Kabir, Jacky W. Keung, Burak Turhan, and Kwabena E. Bennin. supplemental materials. <https://cutt.ly/AmnG8bx>, 2021.
- [104] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, July 1999.
- [105] Jie Zhang, Jiajing Wu, Chuan Chen, Zibin Zheng, and Michael R. Lyu. Cds: A cross-version software defect prediction model with data selection. *IEEE Access*, 8:110059–110072, 2020.
- [106] Huihua Lu, Ekrem Kocaguneli, and Bojan Cukic. Defect prediction between software versions with active learning and dimensionality reduction. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 312–322, 2014.
- [107] Houleng Gao, Minyan Lu, Cong Pan, and Biao Xu. Empirical study: Are complex network features suitable for cross-version software defect prediction? In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pages 1–5, 2019.
- [108] Zhexi Yao, Jiawei Song, Yushuai Liu, Tao Zhang, and Jinbo Wang. Research on cross-version software defect prediction based on evolutionary information. *IOP Conference Series: Materials Science and Engineering*, 563:052092, aug 2019.