

# The impact of training data selection on the software defect prediction performance and data complexity

Benyamin Langgu Sinaga<sup>1,2</sup>, Sabrina Ahmad<sup>1</sup>, Zuraida Abal Abas<sup>1</sup>, Antasena Wahyu Anggarajati<sup>3</sup>

<sup>1</sup>Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

<sup>2</sup>Department of Informatics, Universitas Atma Jaya Yogyakarta, Yogyakarta, Indonesia

<sup>3</sup>PT Setiap Hari Dipakai, Bandung, Indonesia

## Article Info

### Article history:

Received Feb 15, 2022

Revised May 16, 2022

Accepted Jul 19, 2022

### Keywords:

Software defect prediction  
Cross-project defect prediction  
Comparative study  
Data complexity measure  
Training data selection

## ABSTRACT

Directly learning a defect prediction model from cross-project datasets results in a model with poor performance. Hence, training data selection becomes a feasible solution to this problem. Limited comparative studies investigating the effect of training data selection on the prediction performance have presented contradictory results. Those studies also did not analyze why a training data selection method underperforms. This study aims to investigate the impact of training data selection on the defect prediction model and data complexity measures. The method is based on an empirical comparison between prediction performance and data complexity measure before and after selection. This study compared 13 training data selection methods on 61 projects using six classification algorithms and measured the data complexity using six complexity measures focusing on overlap class, noise level, and class imbalanced ratio. Experimental results indicate that the best method for each dataset varies depending on the dataset and classifiers. The training data selection most affects noise rate and class imbalance. We concluded that critically selecting the training data method could improve the performance of the prediction model. We recommend dealing with noise and unbalanced classes when designing training data methods.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



## Corresponding Author:

Sabrina Ahmad

Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka

St. Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

Email: sabrinaahmad@utem.edu.my

## 1. INTRODUCTION

Software review and testing are increasingly important in software organizations due to the strong demand for high-quality software [1]. Such demand requires the software developer to thoroughly review and test each software module to detect and correct possible defects. However, this approach is impractical since the software quality assurance (SQA) team has limited resources and time. Several studies found that only a tiny portion of the software modules caused most software errors [2], [3]. Therefore, the SQA teams must allocate limited resources effectively to the part of a software product that is most likely to contain defects [4], [5]. Using a software defect prediction (SDP) model lets the SQA team focus on prudently reviewing or testing the high defect-prone modules.

Although it looks promising, the lack of historical data makes it challenging to create the SDP model. Recent studies examined the feasibility of developing an SDP model using historical data from the other projects cross-project defect prediction (CPDP). However, CPDP has a disadvantage in that merely

using cross-project datasets to build the SDP model produces a model with unsatisfactory predictive performance. It is because the training and testing datasets have different data distributions. [6], [7].

Numerous solutions to this problem have been proposed, including data transformation [6], [8]–[12], data normalization [6], and training data selection [13]–[18]. Data transformation and normalization use all training instances to train an SDP model, which may contain irrelevant and noisy data. Zhang *et al.* [19] found it difficult to choose a transformation for a given pair of training and testing instances. Prior studies [6], [20] show that the effect of the transformation on the model performance varies on the same dataset. Studies [16], [18], [21] reported that the prediction model developed using selected cross-project data has a satisfied predictive performance. Therefore, selecting the relevant data for training an SDP model becomes an important and challenging task [16]. This study focused on the training data selection method and its corresponding performance on cross-project data.

Many studies have proposed various training data selection methods. Several of them [4], [16], [18], [22]–[24] compared the performance of the training data selection method to a baseline model, an SDP model built using all available training data, to examine the effectiveness of the method. The results are contradictory. Some researchers [22], [25] found that the SDP model built using selected training data performs better than the baseline model. However, the others discovered the opposite result: the baseline model outperforms the SDP model built using selected training data [23], [24]. One possible explanation for this conflicting result is that each study used a different experimental setting, such as different datasets, classifiers, performance metrics, or levels of analysis. This result raises an intriguing question: *does the training data selection positively impact defect prediction performance?*

Li *et al.* [22] also found a minor improvement in defect prediction model performance. It indicates that a defect prediction model built with selected training data is not necessarily better than one built with all training data. The findings motivate us to investigate the other factors that may affect the efficacy of the training data selection method. Ho and Baso [26] proposed data complexity measures to characterize the underlying complexity of a classification problem. The metrics reflect various data characteristics, such as class overlap, class separability, and decision boundary complexity. Cano [27] found that the complexity measures strongly relate to the performance of the classification algorithm. An SDP can be a classification problem that predicts whether an unseen instance is defective or not. We used complexity measures proposed in [26], [28] to investigate the selected cross-project datasets to understand how the findings in [22]–[24] could happen. No dedicated SDP research has investigated the impact of training data selection on training data complexity, to the best of our knowledge. *We raise a question: Does the selection of training data affect the training data complexity?*

Given the question mentioned above, we conducted a large-scale comparison of 13 training data selection methods, trained on 61 datasets from three dataset repositories (i.e., PROMISE, Markov decision process (MDP), AEEEM) using six classification algorithms. Unlike [25], our study focused on the approaches for selecting training data. We further build on [22], [23] by comparing 13 training data selection methods combined with six classifiers over 61 datasets from three repositories, i.e., AEEEM, PROMISE, and MDP. This research aims to determine how training data selection and classifier selection affect defect prediction performance and data complexity. To accomplish the intended goal, we proposed three research questions:

RQ1: *How does the impact of selecting training data on the SDP models?*

Selecting training data could impact the SDP model on most dataset releases. The effect of training data selection varies across dataset releases and the best training data selection method tends to be different for each dataset release. In addition, no single method consistently appears as the best method for selecting training data in all dataset releases. It confirms the no-free lunch theorem. When looking at cumulative performance on each dataset repository, Herbold [16] and Peters *et al.* [29] are the best-performing training data selection methods on the AEEEM and PROMISE datasets, despite their statistical insignificance compared to the baseline model.

RQ2: *Does the choice of classification algorithms impact the performance of an SDP model?*

The classification algorithm affects the prediction performance on most training data selection methods, with Naive Bayes (NB) being the top-performing classifier. No single method consistently performs best when paired with any classifier.

RQ3: *How does the impact of selecting training data on the complexity of training data?*

Selecting training data may affect data complexity measures, especially in noise rate and class imbalance, but the effect is not necessarily positive. The defect datasets are deemed to be complex data, which may cause some selection methods to have unsatisfied performance.

This study contributes in three ways. This study provides: i) a large-scale and comprehensive comparative research on training data selection, encompassing 13 approaches and 61 dataset releases from three dataset repositories (i.e., PROMISE, MDP, and AEEEM) using six classifiers, ii) a detailed analysis of

the impact of selecting training data on prediction performance at two levels of analysis: dataset release and dataset repository level, iii) a detailed analysis of how selecting training data affects training data complexity. To our best knowledge, this is the first study that examines the effect of training data selection on training data complexity in the SDP area. This paper is structured as follows: section 2 reviews the related works, section 3 outlines the methodology, section 5 presents the results and discussion, and section 5 draws on the conclusion and highlights future research.

## 2. RELATED WORKS

### 2.1. Training data selection

Training data selection is a method for selecting the appropriate training instances based on the target instances. Correct identification of suitable training data is critical since irrelevant training instances degrade the performance of an SDP model. Many methods for selecting training data have emerged, classified into several groups (i.e., instance, project, and multi-granularity) based on training data granularity [30].

Several studies have proposed training data selection methods at the instance level using the k-nearest neighbors (kNN) algorithm [18], [31], [32]. In [18], [32] employ testing instances to select the training data, while Peters *et al.* [31] use source instances to guide the selection. Turhan *et al.* [18] found that the defect prediction model has predictive performance close to those built using within-project data. Peters *et al.* [31] concluded that their proposed filter outperforms the within-project and Turhan filters. Later, He *et al.* [32] considered the similarity of both datasets and the defect count of each training data instance to select the relevant data. They concluded that their method is better than the Peters filter. Kawata *et al.* [33] introduced a cluster-based selection using density-based spatial clustering of applications with noise (DBSCAN), which merges the source dataset with the target dataset. The method then partitions the combined dataset into clusters and it selects as training instances the source instances found in a cluster containing at least one target instance. They concluded that the proposed method outperforms Turhan *et al.* and Peter's filter regarding G-measure and area under the curve (AUC). Menzies *et al.* [34] proposed a local filter that developed a model for separately predicting defects for each cluster.

Other studies investigated the selection of training data at the project level. Most methods use data characteristics to assess the closeness of the source and target datasets. Herbold [16] employs the kNN algorithm to choose the appropriate source data. The source and target data similarity were measured using the Euclidean distance based on mean and standard deviation. Herbold found that predictive performance improves significantly based on success rate and recall. Unlike Herbold, He *et al.* [17] do not use distributional features to calculate the similarity of training and testing datasets. Instead, they use performance accuracy to calculate the distance between the training and testing datasets. Selected training data are instances from datasets that exceed a pre-defined cutoff in terms of accuracy.

Studies [4], [22] proposed multi-granularity approaches that combine project and instance-level methods. They used Herbold's method at the project level to filter out the source data. At the instance level, He *et al.* [4] proposed two strategies that adopt the Burak filter and Peter filter, while Li *et al.* [22] used K-means to create clusters of similar instances. Clusters with at least one target instance are combined to form training data instances. Li *et al.* found that their filter defeated Herbold, Burak filter, and Peter filter.

Many methods for selecting data have been proposed; therefore, it is necessary to compare their performances. Several comparative studies compared the effectiveness of training data selection on the SDP performance Table 1. Li *et al.* [22] studied five methods for training data selection on 44 datasets and observed that selecting training data can improve the SDP models. Their analysis focused on the dataset release level. Bin *et al.* [23] compared nine methods on 33 datasets from PROMISE. The baseline method outperformed the defect prediction model built using selected training data. Herbold *et al.* [25] compared the cross-project defect prediction model. They included several approaches to dealing with distribution divergences, such as training data selection, transformation, and normalization. Herbold *et al.* compared nine training data selection methods. Luo *et al.* [24] conducted an experimental study on eleven datasets using ranking-oriented CPDP for nine training data selection methods. Their findings are similar to [23].

Table 1 displays inconsistent findings from several comparative studies. It could be because of the different experimental setups, such as different datasets, classifiers, or levels of analysis. Given the above contradictory findings, we compared the performance of 13 training data selection methods, trained on 61 datasets from three dataset repositories (i.e., PROMISE, MDP, and AEEEM) using six classification algorithms. The impact of training data selection is evaluated at two granularity levels: dataset release and dataset repository level. We also investigate the data complexity measure of the selected cross-project datasets to understand better how the findings in [22]-[24] could have occurred.

Table 1. Comparative studies on training data selection method for cross-project defect prediction

Author(s)	TDS methods	Number of datasets (repositories)	Classifiers	Analysis	Findings
Li <i>et al.</i> [22]	5	44 (PROMISE)	NB, SVM	Performance at dataset release level	Data filtering could improve the performance of SDP model.
Bin <i>et al.</i> [23]	9	33 (PROMISE)	RF	Performance at data repository level	Using all source data as training data provided better prediction accuracy.
Herbold <i>et al.</i> [25]	9	67 (PROMISE, MDP, AEEEM, RELINK, and NETGENE)	DT, LR, RBF Net, NB, RF, SVM	Performance at data repository level	LACE2 [29] using NB was the best training data selection methods.
Luo <i>et al.</i> [24]	9	11 (PROMISE)	LTR	Performance at data repository level	Selecting training data did not affect the performance of ranking oriented CPDP.
Proposed study	13	61 (PROMISE, MDP, and AEEEM)	DT, LR, MLP, NB, RF, SVM	Performance at dataset release and data repository, impact on data complexity	---

## 2.2. Data complexity measures

Data complexity measures are widely utilized to assess the dataset's underlying characteristics, i.e., class overlap, linear separability, and neighborhood among instances. These metrics are applied to investigate the inherent difficulty of a classification problem within a given dataset. Ho and Baso proposed the initial version of the data complexity metric [26]. Lorena *et al.* [28] then standardized the measures so that each measure is in the range [0, 1], with a high value indicating highly complex data.

Many studies have investigated the use of data complexity measures in various areas of research, such as class imbalance [35], feature selection [36], [37], classifier recommendation [38], and prediction of classifier performance [39]. We used data complexity measures to study why the SDP model built with selected training data performs similarly to or even worse than the one built with all training data. This study used the standardized version of the complexity measures categorized into three groups: feature overlap, linear separability, and neighborhood measure, as presented in Table 2.

Table 2. Data complexity measures

Symbol	Name	Description
Feature overlap		
F1	Fisher's discriminant ratio	This measure computes feature overlap across classes. F1's original version holds the highest discriminant ratio of all features. The standardized version adopts the inverse formulation of the original.
F2	Volume of overlapping region	This measure calculates feature value distributions within classes.
F3	Feature efficiency	This measure computes the efficiency of each feature. It checks for value overlap between instances of different classes for each feature. If there is an overlap region, it counts the instances in that region. It divides the number of instances in the overlap region by the number of all instances. F3 takes the smallest.
Linear separability		
L1	Sum of the error distance by linear programming.	This measure uses SVM to create a linear boundary and calculates the sum of distances between the incorrectly classified instances and the boundary.
L2	Error rate of linear classifier	This measure calculates the error rate of the linear SVM.
L3	Non-linearity of a linear classifier.	This measure interpolates pairs of training examples of the same class to create a new test dataset. A linear SVM is built using the original datasets and then applied to the test dataset. L3 has the error rate.
Neighborhood		
N1	Fraction of borderline point	This measure first builds a minimum spanning tree (MST). The proportion of vertices incident to edges that connect instances of opposite classes in the generated MST is used to calculate the N1 values.
N2	Ratio of intra/extra class nearest neighbor distance	This measure calculates the ratio between intra-class and inter-class distances. Intra-class distance is the sum of the distances between an instance and its nearest neighbor. Inter-class distance is the sum of the distances between an instance and its nearest enemy.
N3	Error rate of the nearest neighbor classifier	This measure estimates the error rate of a 1NN classifier using leave-one-out.
N4	Non-linearity of the nearest neighbor classifier	This measure is similar to L3, but it utilizes an NN classifier rather than a linear SVM.
Class Imbalanced		
C2	Imbalance Ratio	This measure assesses the imbalance ratio.

The improvement in the defect prediction performance is small [22]. Studies [23], [24] also found that the SDP model constructed from selected training data underperforms the baseline model. It encourages us to investigate the factors that may affect the efficacy of the training data selection method. We used data complexity measures to investigate the selected training dataset characteristics to understand how the findings in [22]–[24] could have occurred. No dedicated study has examined the impact of selecting training data on the data complexity in software defect prediction.

### 3. METHOD

#### 3.1. Experimental datasets

We experimented on 61 dataset releases from three different repositories, namely: PROMISE, MDP, and AEEEM, since they have been widely used in defect prediction studies [25], [40]–[42]. Table 3 and Table 4 present the description of each release. The PROMISE dataset [43] contains several projects with various releases. Each release (version) has 20 independent attributes and a dependent attribute representing the number of defects found in that release. Herbold [16] did not use proprietary datasets to avoid the influence of mixing open-source and proprietary datasets on the experimental results. He also eliminated small projects by only choosing versions having at least 100 instances. Following his arguments, we select 44 releases from 14 projects available from this repository.

The MDP has 12 releases from six different projects, each of which has a different number of metrics, and they share 17 static metrics in common. We used the cleaned version of the MDP dataset [44] since prior studies identified inconsistency issues and mislabeled data in the original version of this dataset [44]–[46]. We used all releases from this dataset repository. The AEEEM dataset [47] has five releases. This dataset contains 61 metrics. We used this dataset since D'Ambros *et al.* [47] pointed out that prediction methods using various metrics result in the best performance. We used all releases from this repository.

Table 3. Description of characteristics of benchmark datasets

Dataset	Projects	Releases	Level	Language	Metrics
AEEEM	5	5	class	Java	61
PROMISE	14	44	class	Java	20
MDP	6	12	class	Java, C/C++	21, 39, 40

Table 4. Statistical description of benchmark datasets

Dataset	No	release	#instances	#defective	%defective	No	release	#instances	#defective	%defective
PROMISE	1	ant-1.3	125	20	16	23	lucene-2.2	247	144	58
	2	ant-1.4	178	40	22	24	lucene-2.4	340	203	60
	3	ant-1.5	293	32	11	25	poi-1.5	237	141	59
	4	ant-1.6	351	92	26	26	poi-2.0	314	37	12
	5	ant-1.7	745	166	22	27	poi-2.5	385	248	64
	6	arc	234	27	12	28	poi-3.0	442	281	64
	7	camel-1.0	339	11	3	29	redaktor	176	27	15
	8	camel-1.2	608	216	36	30	synapse-1.0	157	16	10
	9	camel-1.4	872	145	17	31	synapse-1.1	222	60	27
	10	camel-1.6	965	188	19	32	synapse-1.2	256	86	34
	11	ivy-1.1	111	63	57	33	tomcat	858	77	9
	12	ivy-1.4	241	16	7	34	velocity-1.4	196	147	75
	13	ivy-2.0	352	40	11	35	velocity-1.5	214	142	66
	14	jedit-3.2	272	90	33	36	velocity-1.6	220	78	35
	15	jedit-4.0	306	75	25	37	xalan-2.4	723	110	15
	16	jedit-4.1	312	79	25	38	xalan-2.5	803	387	48
	17	jedit-4.2	367	48	13	39	xalan-2.6	885	411	46
	18	jedit-4.3	492	11	2	40	xalan-2.7	909	898	99
	19	log4j-1.0	135	34	25	41	xerces-init	162	77	48
	20	log4j-1.1	109	37	34	42	xerces-1.2	440	71	16
	21	log4j-1.2	205	189	92	43	xerces-1.3	453	69	15
	22	lucene-2.0	195	91	47	44	xerces-1.4	588	437	74
MDP	1	cm1	344	42	12	7	mw1	264	27	10
	2	jml	9593	1759	18	8	pc1	759	61	8
	3	kc1	2096	325	16	9	pc2	1585	16	1
	4	kc3	200	36	18	10	pc3	1125	140	12
	5	mc1	9277	68	1	11	pc4	1399	178	13
	6	mc2	127	44	35	12	pc5	17001	503	3
AEEEM	1	lucene	691	64	9	4	eclipse	997	206	21
	2	pde	1497	209	14	5	equinox	324	129	40
	3	mylyn	1862	245	13					

### 3.2. Training data selection methods

We used strict CPDP [48] since it is prevalent in defect prediction studies [25]. This approach does not require a labeled target dataset, which an organization may not have when building an SDP model. The representative training data selection methods were chosen based on their frequent use in SDP comparative studies [22]–[24], the granularity of the training data, and the selection strategy, Table 5 lists the methods.

Table 5. Overview of the included training data selection methods

Ref	Granularity	Characteristics	Selection methods	Label Name
[18]	Instance-level	K-nearest neighbors	Distance to target instances, using Euclidean.	Turhan09
[31]	Instance-level	K-nearest neighbors	Distance to target instances, one source instance per target instance.	Peters13a
[49]	Instance-level	K-nearest neighbors	Distance to target instances, using Hamming.	Ryu15
[29]	Instance-level	K-nearest neighbors	Distance to nearest unlike neighbors.	Peters15
[34]	Instance-level	Clustering	local cluster-based selection.	Menzies11
[33]	Instance-level	Clustering	instances in the same cluster as target instance.	Kawata15
[50]	Instance-level	Ranking-based	10% source instances with highest power.	Peters13b
[16]	Project-level	K-nearest neighbors	Distance to target dataset, based on data characteristics.	Herbold13
[17]	Project-level	Ranking-based	Top 10 datasets rank based on predictive accuracy.	ZHe13
[22]	Multi-granularity	K-nearest neighbors + clustering	Distance to target datasets (Herbold13) + instances in the same cluster as target instance.	YLi17
[4]	Multi-granularity	K-nearest neighbors + clustering	Distance to target datasets (Herbold13) + instances in the same cluster as target instance (Peters13a).	PHe0214
[4]	Multi-granularity	K-nearest neighbors at both level	Distance to target datasets (Herbold13) + distance to target instances (Turhan09).	PHe0114

### 3.3. Classification algorithms

We used six classifiers to evaluate the effectiveness of training data selection on the prediction performance [25], [40]. We considered different families of successful classifiers in finding defects [51]. The used classifiers are C4.5 (decision tree), Naive Bayes (probabilistic), LR (regression function), random forest (ensemble method), multi-layer perceptron (neural networks), and support vector machine. Tantithamthavorn *et al.* [52] found that optimizing classifier parameters has little effect on AUC. Thus, we used the WEKA classifiers with default parameters.

### 3.4. Performance measure

We used the AUC for the prediction measure since the AUC is unaffected by the imbalanced class problem, threshold-independent [52], and is widely used in SDP research [53]. The AUC has a maximum value of one. A value close to 0.5 indicates that the model behaves similarly to a random model.

### 3.5. Experimental settings

This study utilized a multi-source strict CPDP approach [6], [25], [48]. It creates a prediction model from multiple source training datasets and tests it on a single target dataset. We used this approach since several methods [4], [16], [17], [22] select the training instances based on the similarity between training and the testing dataset at the project level, which required multiple source training data. Such an experimental setup enables us to execute the experiment only once for each pair of training data selection and classifier [54], except for methods with a random component (i.e., Menzies11, Peters13b, Peters15, and ZHe13). For such methods, we experimented ten times and calculated the mean performance value.

Figure 1 shows the procedure of this comparative study. First, we set up training and target datasets using leave-one-out multi-source CPDP. For each repository, i.e., AEEEM, PROMISE, and MDP, each dataset release is selected as the target dataset. For example, when using the MDP dataset, if PC1 is the target dataset, the remaining releases are used as candidate training datasets, excluding those from the same project (i.e., PC2–PC5). Second, we apply each training data selection method to the candidate training datasets. Third, we train an SDP model using six classifiers on the selected training dataset. Fourth, we evaluate the model on the target dataset. Lastly, we calculate the complexity measure of the training datasets. Procedure 1 presents the steps of the prediction process with a sample of MDP datasets.

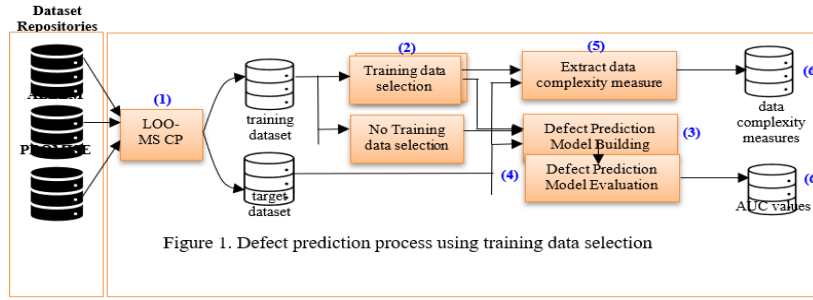


Figure 1. Defect prediction process using training data selection

Procedure 1	Cross-project defect prediction with training data selection
<b>Input:</b>	<ul style="list-style-type: none"> <li>datasets = {CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5}</li> <li>methods = {ALL, Herbold13, Kawata15, Menzies11, Peters13a, Peters13b, Peters15, PHe0114, PHe0214, Ryu15, Turhan09, YLi17, ZHe13}</li> <li>dcm = {F1, F2, F3, L1, N1, C2}</li> <li>classifiers = {DT, LR, MLP, NB, Rf, SVM}</li> </ul>
<b>Output:</b>	AUC_values, data_complexity_measures
<pre> 0 performance_values = ∅, data_complexity_values = ∅   for dataset in datasets do 1   # Leave one out multi-source cross-project (LOO-MS CP)     test.data = dataset     train.data = instances from other projects ## if test.ds = PC1, train.ds are all     instances from CM1, JM1, KC3, MC1, MC2, MW1     for method in methods do 2     train.datamethod = apply_selection (method, train.data) 3     SDP.model = train (train.dsmethod, classifiers) 4     AUC<sub>train.ds, method, classifiers</sub> = evaluate (SDP.model, test.data)     AUC_values = AUC_values ∪ AUC<sub>train.ds, method, classifiers</sub> 5     dcv<sub>train.ds, method, dcm</sub> = calculate_complexity (train.datamethod, dcm)     data_complexity_measures = data_complexity_measures ∪ dcv<sub>train.ds, method, dcm</sub>     end for   end for 6 return performance_values, data_complexity_measures </pre>	

#### 4. RESULTS AND DISCUSSION

This section presents the results and discussions on the impact of selecting training data on the prediction performance and data complexity measure and the impact of the classifier on the SDP model.

##### 4.1. RQ1: how does the impact of selecting training data on the SDP models?

We perform the evaluation in two-level analysis, dataset release level, and dataset repository level. We present two evaluation points at each level of analysis, i.e., the impact of applying training data selection and the best-performing method. At the dataset release level, we aim to identify the effect of applying training data selection and determine the best method for each dataset release. To reach this objective, we calculate the performance average, in terms of the AUC, of the training data selection methods across different classifiers. We then run the Wilcoxon and effect size tests to quantify the difference between training data selection and baseline methods. We choose the method with the highest performance average as the best training data selection method. At the dataset repository level, we aim to determine the overall effect of selecting training data for each dataset group. The Friedman test is used to compare the performance of various training data selection methods. If the test rejects the null hypothesis, we then conducted the post-hoc Nemenyi test. We also use the Wilcoxon test to compare the performance of each training data selection method to that of the baseline.

##### 4.1.1. Dataset release level

Table 6 (see in appendix) shows the performance average of training data selection methods across six machine learning classifiers for each dataset release. All represents a defect prediction model built using all training datasets, i.e., the baseline. The bold value indicates the best training data selection method for a dataset release. A "+" sign in the parentheses indicates that a training data selection method improves the baseline with a certain magnitude, while "--" suggests the opposite. S, M, L, N represents the magnitude of the delta, denoting small, medium, large, and negligible, respectively. For example, for the eclipse dataset release, Herbold13 has 0.7247 (+, M), meaning that Herbold13 has the highest average performance, i.e., 0.7247, and improves the performance of the baseline with a medium effect size. Column "%impr" indicates

the improvement achieved using the optimal method. The optimal method was Herbold13, with an improvement of 7.35%. Figure 2 graphically summarizes the impact of training data selection methods on each dataset release based on the number of methods that successfully improve the baseline method.

Table 6 (in appendix) and Figure 2 show that training data selection could enhance the defect prediction performance since at least one method performs better than the baseline in most dataset releases. The impact of selecting training data differs across dataset releases. The improvement obtained by the best method for each dataset ranges between 0 (no improvement, i.e., ant-1.3, ivy-1.4, redactor, kc1, and mw1) and 21.87% (PC2). None of the methods perform optimally on ant-1.3, ivy-1.4, redactor, kc1, and mw1. It might be because a particular dataset is more difficult to learn from than others [54]. Hosseini *et al.* [55] also identified several datasets that are hard to predict using their proposed method. From a different perspective, we suspect that datasets such as ant-1.3, kc1, and mw1 are easier to learn (i.e., AUC value larger than 0.7100). It is demonstrated by the fact that the prediction results obtained using the baseline method (without training data selection) are better than that using the data selection method.

Figure 2 also implies that training data selection does not necessarily improve the baseline model on each dataset release. Figure 3(a) confirms this result. On the one hand, Herbold13 and PHe0114 improve the baseline in at least 40% of dataset releases in the AEEEM dataset. Herbold13, Kawata15, Peters13b, PHe0114, and Turhan09 perform optimally in the PROMISE dataset, as they upgrade the baseline in over 40% of dataset releases. The same holds for Peters13b and Turhan09 in the MDP dataset. On the other hand, several training data selection methods are consistently inferior to the baseline method. For example, Menzies11, Peters13b, Ryu15, Turhan09, and YLi17 do not perform optimally in all AEEEM datasets. Similarly, the same is true for Menzies11, Kawata15, and Ryu15 in the MDP dataset. Menzies11 and Ryu15 perform better than the baseline in only four and three out of 44 dataset releases in the PROMISE dataset. Figure 3(b) shows that Herbold13 reaches the best performance on four out of five dataset releases on the AEEEM dataset. For the PROMISE dataset, the best-performing method on a dataset release is varied. Eleven methods, including ALL, have ever been the best method. Herbold13, Peters13b, and Turhan09 are the most often cited techniques. Peters13b is the method that frequently outperforms all others on the MDP dataset. The remaining five methods, including ALL, ever perform optimally on MDP datasets as well. There is no training data selection method having the best performance across all dataset releases.

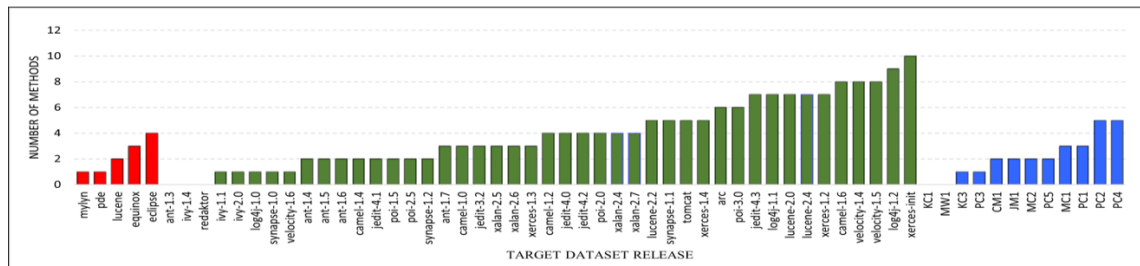


Figure 2. The number of training data selection methods performs better than the baseline on each dataset release. Red, green, and blue colors represent dataset release from AEEEM, PROMISE, and MDP datasets

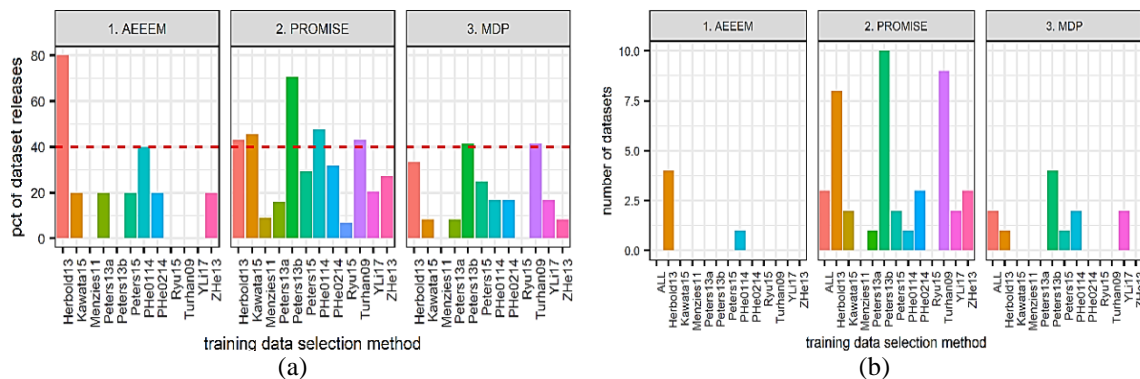


Figure 3. Improvement of training data selection over the baseline (a) the percentage of times a training data selection method improves the baseline model across all studied datasets and (b) the distribution of the best training data selection method across all studied datasets



#### 4.1.2. Dataset repository level

Tables 7-9 compare each training data selection method to the baseline (i.e., ALL) using the Wilcoxon test for six classifiers. Each value ( $\overline{\text{AUC}}$ ) represents the mean AUC of the training data selection method paired with a classifier across all dataset releases. For each classifier, bold values denote that the mean AUC of the corresponding method is higher than the baseline. The underlined value represents the highest mean AUC (i.e., the best performing method). A training data selection method statistically performs better or worse than the baseline if the p-values ( $p_{\text{val}}$ ) are less than the significance level (i.e., 0.05). For instance, on the AEEEM dataset, Herbold13 combined with DT substantially outperforms the baseline since the p-value (0.0317) is less than 0.05. Herbold13 is the best method when paired with DT.

Tables 7-9 show that training data selection could improve the defect prediction performance since, for each method, there exists at least one classifier that performs better than the baseline. Herbold13 and PHe014 are better than the baseline in three classifiers on the AEEEM dataset. For the PROMISE dataset, Kawata15 and Peters13b dominantly outperform the other methods since they are better than the baseline in five and four classifiers, respectively. However, no method could outperform the baseline in more than two classifiers for the MDP dataset.

Table 7. Performance of training data selection method on different classifiers on the AEEEM dataset

Method	DT		LR		MLP		NB		RF		SVM	
	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$
ALL	0.5684		0.7252		0.6986		0.7180		0.7298		0.5005	
Herbold13	<b><u>0.6200</u></b>	0.0317	0.6985	0.6905	0.6546	0.3095	0.7087	0.5476	<b><u>0.7376</u></b>	1.0000	<b><u>0.6633</u></b>	0.0097
Kawata15	0.5608	0.5284	<b>0.7253</b>	0.9166	0.6721	0.5476	<b>0.7184</b>	0.9166	0.7252	1.0000	0.5000	0.4237
Menzies11	<b>0.5807</b>	0.4206	0.5835	0.0079	0.5765	0.0159	0.6195	0.0317	0.5881	0.0079	0.4999	0.2330
Peters13a	0.5495	0.6905	0.6827	0.0556	0.6341	0.2222	0.6931	0.2222	0.7237	0.8413	0.5000	0.4237
Peters13b	0.5334	0.3095	0.7191	0.6905	0.6754	0.3095	0.7041	0.6905	0.5990	0.0079	0.5000	0.4237
Peters15	0.5566	0.3095	0.6573	0.0556	0.5767	0.0159	<b>0.7398</b>	0.5476	0.6994	0.3095	0.5000	0.4237
PHe0114	<b>0.5721</b>	0.5476	<b><u>0.7353</u></b>	0.6905	0.6349	0.3095	<b><u>0.7512</u></b>	0.4206	0.6959	0.2222	0.5000	0.4237
PHe0214	0.5425	0.4206	0.6917	0.5476	0.5980	0.0556	<b>0.7436</b>	0.6905	0.6900	0.4206	0.5000	0.4237
Ryu15	0.5374	0.0952	0.5729	0.0079	0.5451	0.0079	0.5948	0.0159	0.5383	0.0079	0.4996	0.1812
Turhan09	<b>0.5807</b>	0.4206	0.5835	0.0079	0.5765	0.0159	0.6195	0.0317	0.5881	0.0079	0.4999	0.2330
YLi17	<b>0.5894</b>	0.4206	0.6146	0.0159	0.5743	0.0317	0.6955	0.5476	0.6794	0.3095	<b>0.5129</b>	0.0449
ZHe13	<b>0.5959</b>	0.8413	0.5816	0.0079	0.5881	0.0317	0.6291	0.0556	0.6558	0.0317	<b>0.6107</b>	0.0097

Table 8. Performance of training data selection method on different classifiers on the PROMISE dataset

Method	DT		LR		MLP		NB		RF		SVM	
	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$
ALL	0.5805		0.6945		0.6673		0.6943		0.6707		0.5072	
Herbold13	0.5339	0.0106	0.6317	0.0043	0.6091	0.0035	0.6844	0.5846	0.6438	0.1180	<b><u>0.5856</u></b>	0.0000
Kawata15	0.5763	0.7543	<b>0.6967</b>	1.0000	<b>0.6688</b>	0.8258	<b>0.6954</b>	0.9468	<b>0.6745</b>	0.8453	<b>0.5075</b>	0.8248
Menzies11	0.5775	0.8347	0.5875	0.0000	0.5742	0.0000	0.5832	0.0000	0.5890	0.0000	<b>0.5095</b>	0.0454
Peters13a	0.5503	0.1476	0.6318	0.0017	0.5727	0.0000	0.6770	0.4016	0.6521	0.3764	0.5000	0.0041
Peters13b	<b>0.6328</b>	0.0013	<b>0.6946</b>	0.9567	<b>0.6749</b>	0.6037	0.6901	0.7480	0.6554	0.7734	0.5049	0.7763
Peters15	0.5694	0.5178	0.6647	0.0733	0.6011	0.0003	<b>0.6944</b>	0.9570	0.6467	0.1698	0.5000	0.0026
PHe0114	0.5719	0.5259	<b>0.7035</b>	0.9503	0.6188	0.0078	<b>0.7115</b>	0.4470	0.6532	0.3227	<b>0.5166</b>	0.9215
PHe0214	0.5694	0.3457	0.6651	0.0720	0.5901	0.0001	<b>0.7018</b>	0.7429	0.6474	0.1759	0.5000	0.0041
Ryu15	0.5600	0.1672	0.5869	0.0000	0.5624	0.0000	0.6161	0.0000	0.5582	0.0000	0.4994	0.0261
Turhan09	0.5766	0.8773	<b>0.7010</b>	0.8806	0.6206	0.0148	<b>0.7101</b>	0.4723	0.6547	0.4087	<b>0.5184</b>	0.9353
YLi17	0.5509	0.1160	0.6382	0.0210	0.5981	0.0006	0.6872	0.6980	0.6289	0.0902	<b>0.5176</b>	0.0167
ZHe13	<b>0.6027</b>	0.1506	0.6089	0.0000	0.5939	0.0004	0.6160	0.0000	0.6197	0.0068	<b>0.5887</b>	0.0000

Table 9. Performance of training data selection method on different classifiers on the MDP dataset

Method	DT		LR		MLP		NB		RF		SVM	
	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$	$\overline{\text{AUC}}$	$p_{\text{val}}$
ALL	0.7065		0.7301		0.7355		0.7462		0.7055		0.5000	
Herbold13	0.5427	0.0000	0.7182	0.7125	0.6467	0.4095	0.7229	0.2913	0.7029	0.8428	<b>0.6091</b>	0.0000
Kawata15	0.6578	0.1409	<b>0.7362</b>	0.8428	0.6977	0.7553	0.7237	0.2189	<b>0.7106</b>	0.9323	0.5000	1.0000
Menzies11	0.5315	0.0000	0.5255	0.0000	0.5228	0.0000	0.5513	0.0000	0.5345	0.0003	<b>0.5002</b>	0.3593
Peters13a	0.6133	0.0086	0.6592	0.2415	0.6046	0.0887	0.6729	0.0684	0.6731	0.3777	0.5000	1.0000
Peters13b	0.7039	0.8398	0.7225	0.9774	0.7296	0.9323	0.7327	0.3474	0.6744	0.6297	0.5000	1.0000
Peters15	0.5452	0.0006	0.7249	0.5899	0.6128	0.0173	<b>0.7706</b>	0.7125	0.6795	0.1978	0.5000	1.0000
PHe0114	0.5593	0.0002	0.6985	0.4776	0.6671	0.1277	<b>0.7721</b>	0.6297	<b>0.7263</b>	0.6297	0.5000	1.0000
PHe0214	0.5361	0.0002	0.6962	0.4095	0.6212	0.0205	<b>0.7732</b>	0.5137	<b>0.7286</b>	0.7553	0.5000	1.0000
Ryu15	0.5245	0.0000	0.5345	0.0000	0.5022	0.0000	0.6357	0.0007	0.5132	0.0000	0.4994	0.1471
Turhan09	0.6216	0.0072	<b>0.7666</b>	0.4883	0.7011	0.3474	<b>0.7692</b>	0.7125	0.6991	0.6297	0.5000	1.0000
YLi17	0.6255	0.1409	<b>0.7328</b>	1.0000	0.7228	0.7125	0.7355	0.4776	<b>0.7264</b>	0.6707	0.5000	1.0000
ZHe13	0.6512	0.1409	0.5981	0.0056	0.5991	0.0068	0.6066	0.0001	0.6294	0.0597	<b>0.5688</b>	0.0000

It is also difficult for Peters13 and Ryu15 to outperform the baseline as no classifier paired with these methods successfully enhances the prediction performance in all dataset repositories. All methods are inferior to the baseline when paired with MLP on the AEEEM and MDP datasets. The same results also happen on the MDP dataset when the training data selection methods are combined with the DT. Nevertheless, the results show that the prediction model trained on selected training data is viable for predicting the defect in a cross-project setting correctly.

To find the overall performance of each method on each dataset group, we calculate the average performance of the training data selection method across different classifiers and dataset releases. We ran the Friedman test and found a significant difference in performance average between pairwise training data selection methods. We then conducted the Nemenyi test, which is presented in Figure 4.

Figure 4(a) compares different training data selection methods using the Nemenyi test on the AEEEM dataset. It displays the mean performance rank of each method with a critical distance (CD) equal to 8.16. Herbold13 is the most optimal method for selecting training data since it achieves the best mean rank. It is the only method better than the baseline, despite their statistically insignificant margin. It enhances the baseline model in 80% of the AEEEM dataset releases with a significant difference in 60% of dataset releases, i.e., eclipse, equinox, and lucene see Table 6 (in appendix). All remaining methods fail to increase the prediction performance of the baseline.

Figure 4(b) compares different training data selection methods using the Nemenyi test on the PROMISE dataset. The critical distance is CD 2.75. The Nemenyi test identifies four groups. We find that the baseline method belongs to the first and second groups (from right to left), significantly outperforming Peters13a, Ryu15, and Menzies11 and comparable to the remaining methods. Peters13b has the highest impact on the prediction model on the PROMISE dataset. It is comparable to Kawata15, Turhan09, PHe0114, and the baseline and significantly outperforms the others. Peters13b achieves a mean AUC of 0.6421, whereas the baseline is 0.6358. Peters13b successfully outperforms the baseline in 31 out of 44 dataset releases; however, the performance differences between both approaches are significant only in 15 dataset releases, with the small (S) and medium (L) effect size.

Figure 4(c) displays the Nemenyi test results on the MDP dataset. The critical distance is 5.27. The Nemenyi test identifies four groups. We find that the baseline model belongs to the first group, significantly outperforming ZHe13, Ryu15, and Menzies11 and comparable to the remaining methods. No method positively impacts the prediction performance of the baseline. Table 6 (in appendix) shows that Peters13b performs better than the baseline in five dataset releases (CM1, JM1, KC3, MC2, PC1); the differences, however, are statistically insignificant (the effect size magnitude is negligible). On the contrary, the baseline is superior to Peters13b in seven dataset releases, on three of which (PC2, PC3, PC4), the prediction performances of the two approaches differ statistically.

As for the best method, statistical testing found no significant difference between the best training data selection method and the baseline. However, the mean rank value shows that Herbold13 and Peters13b outperform the baseline on the AEEEM and PROMISE datasets, respectively. Thus, Herbold13 is the best method for selecting training data for the AEEEM, while the best method for the PROMISE dataset is Peters13b, with an improvement of 1% and 3.6%, respectively the last three rows in Table 6 (in appendix).

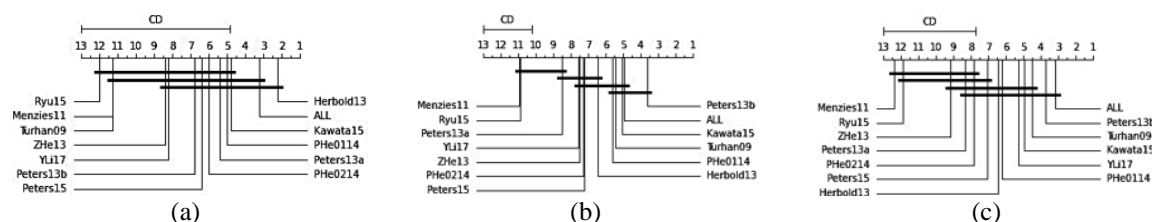


Figure 4. Nemenyi test for the comparison of training data selection methods for different dataset repositories (a) AEEEM datasets, (b) PROMISE datasets, and (c) MDP datasets

Answer to RQ1: the above analysis found that selecting training data could impact the SDP model on most dataset releases. The effect of training data selection varies across dataset releases and the best training data selection method tends to be different for each dataset release. In addition, no single method consistently appears as the best method for selecting training data in all dataset releases. It confirms the no-free lunch theorem. When looking at cumulative performance on each dataset repository, Herbold13 and Peters13b are the best-performing training data selection methods on the AEEEM and PROMISE datasets, despite their statistical insignificance compared to the baseline model.

#### 4.2. RQ2: Does the choice of classification algorithms impact the performance of an SDP model?

We present two evaluations to answer this research question: the impact of classifiers on training data selection performance and the best classifier for each dataset repository. For the former, we calculate the performance average of each classifier across training data selection methods on every dataset repository. While for the latter, we run the Friedman test. Suppose there is a significant difference in average performance between pairwise methods we conduct the post-hoc Nemenyi test.

Results for RQ2: For each classifier, we calculate the performance average of each training data selection method across all dataset releases. Figures 5(a)-(c) compare the performance different classifiers paired with different training data selection methods for each repository. We observed that the classifier affects the performance of the training data selection methods. The eleven methods for selecting training data exhibit a nearly similar pattern of performance average on the AEEEM, PROMISE, and MDP datasets. Since the imbalanced dataset, support vector machine (SVM) seems to fail to predict the defect. Except for Herbold13 and ZHe13, Their AUC values are close to 0.500, indicating that SVM behaves like a random model.

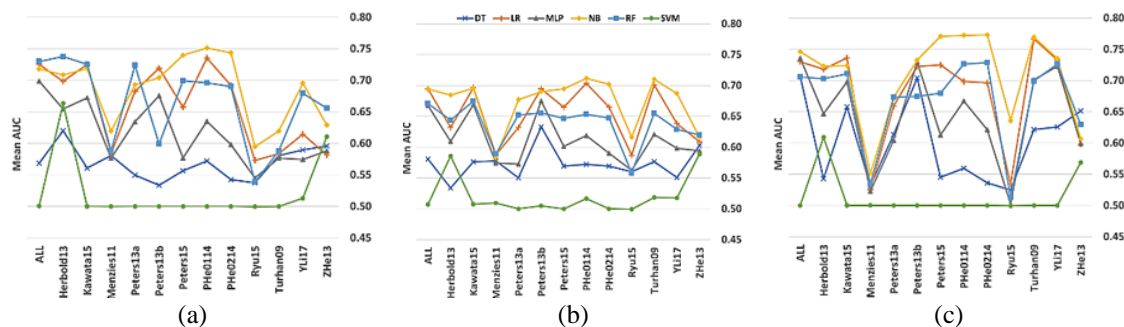


Figure 5. Performance comparisons of different classifiers paired with different training data selection methods, based on mean AUC (a) AEEEM, (b) PROMISE, and (c) MDP

Figure 5 shows that the NB is the best-performing classifier in most training data selection methods. For the AEEEM dataset, NB outperforms the other classifiers when paired with seven training data selection methods, i.e., Menzies11, Peters15, PHe0114, PHe0214, Ryu15, Turhan09, and YLi17. For the PROMISE dataset, the NB provides the best performance for eight training data selection methods, i.e., Herbold13, Peters13a, Peters15, PHe0114, PHe0214, Ryu15, Turhan09, and YLi17. While on the MDP dataset, NB outperforms all other classifiers, except for Kawata15, Peters13a, and ZHe13.

We ran the Friedman test to validate these results and found a significant difference in the average performance between pairwise classifiers. Figure 6(a)-(c) compare the performances of six classifiers using the Nemenyi for AEEEM, PROMISE, and MDP dataset. The NB always has the lowest mean rank in all datasets. It is consistently better or significantly better than the other classifiers across all evaluated datasets.

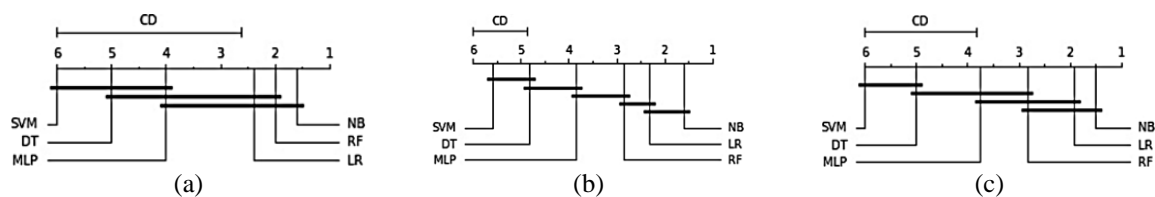


Figure 6. Nemenyi test for the comparison of classifiers for different dataset repositories (a) AEEEM datasets, (b) PROMISE datasets, and (c) MDP datasets

Menzies *et al.* [56] discovered that reducing training data does not affect NB performance. It is confirmed in our experiment. Our experiment reveals that the retention rate of the training data selection method varies. When applied to the PROMISE datasets, Peters15, Turhan09, and Kawata15 achieve retention rates of 1.95 percent, 14.8 percent, and 93.1 percent, respectively, indicating that the number of training data is significantly different. Despite the disparate retention rates, NB performs relatively consistently across the three methods. Yu *et al.* [57] studied the prediction performance in the imbalanced class and concluded that NB is more stable in imbalanced datasets. Catal [58] also pointed out that NB is the robust classifier for

supervised learning. Based on the findings of these studies, we believe this is why NB performs well in this study. This finding also substantiates the statement that simple classifiers, such as NB perform well when building models to predict defects [53].

SVM seems problematic in predicting the defect. It consistently performs poorly in all training data selection methods except Herbold13 and ZHe13. Their performance values are close to 0.500, indicating that SVM behaves like a random classifier. The possible explanation is that the used datasets have imbalanced nature, with more non-defect than defect modules. Bowes *et al.* [54] pointed out that the SVM classifier is sensitive to highly imbalanced datasets. As a result, most SVM-based training data selection methods underperform when trained on such unbalanced datasets. Herbold13 and ZHe13 are the exceptions. Both approaches deal with the imbalanced dataset using under-sampling and weighting. Therefore, in the case of imbalanced datasets, both methods could improve the SVM-based SDP models.

Answer to RQ2: the above analysis concludes that the classification algorithm affects the defect prediction performance on most training data selection methods, with the NB being the top-performing classifier. Furthermore, no single method consistently performs best when paired with any classifier.

#### 4.3. RQ3: How does the impact of selecting training data on the complexity of training data?

Table 6 (in appendix) shows that training data selection can improve prediction performance. However, most performance improvements have negligible or minor effects. It confirms the findings in [22]. It motivates us to investigate further the causes of such findings. This experiment deal with three CPDP challenges: class imbalance, noisy dataset, and class complexity. We studied whether selecting training data addresses this problem by studying the impact of selecting training data on the complexity of the defect datasets and relating the complexity with the performance of the prediction model.

We compared the data complexity measures of the training dataset before and after selection. We only focus on data complexity measures representing CPDP problems, such as N1, which is sensitive to noise in the dataset [36], [59], [60], and C2, which is related to imbalanced data. We also identify the complexity of the defect dataset used. We did not include all related measures because some of them, i.e., L2, L3, and N3, is based on a specific classifier's error rate that was not used in this experiment. Thus, we compute only F1, F2, F3, L1, N1, and C2 to detect class overlap (F1-F3), class separability (L1), presence of noise (N1), and class imbalance (C2). We used the standardized complexity measures (available in R package `ECOL`) proposed by Lorena *et al.* [28]. We quantify the impact of data selection on data complexity as the difference between the pre-selection and post-selection data complexity measures. Figure 7 shows the quartile box plot representing the difference between the pre-selection and post-selection data complexity measures. The positive boxplot denotes that the complexity of all datasets is reduced after being selected by a particular method. The negative boxplot means the opposite. The box plot spanning from positive to negative y-axis shows the effect of data selection on data complexity measures can be positive or negative. We present several observations:

F1 and F2 are the least affected complexity measures. No training data selection method could reduce this complexity measure, except for F2 in PROMISE datasets. It is reasonable since class overlap, and class separability have a stronger relationship with feature relevancy [36], not instance relevancy, likewise for L1, where most median values have the value 0.

C2 is the most impacted data complexity measure. Two training data selection methods, Herbold13 and ZHe13, consistently produce positive values for all corpus data, indicating that they successfully reduce class imbalance levels. It is understandable since both methods employ under-sampling and weighting to compensate for the unbalanced dataset. The improvement of C2 in the other methods is dependent on the imbalanced ratio of the training data. The more unbalanced the training data, the more likely C2 will have a better value after selection. The MDP datasets have a high imbalanced ratio, whereas the PROMISE dataset has a low ratio. The improvement of C2 is evident in the MDP dataset. For PROMISE datasets, the impact on this measure for methods other than Herbold13 and ZHe13 is unclear, as C2 can be either positive or negative. If training data selection is not appropriately handled, better result on the imbalanced class level after selecting training data is not guaranteed.

In general, training data selection has no positive effect on N1. Except for Ryu15, the median value of N1 is negative in all dataset corpora. It has two causes. First, the N1 values of Ryu are always positive since Ryu15 is the only training data selection method that performs noise filtering. Ryu15 measures the distance between a data instance and the entire data distribution using the Mahalanobis distance. Second, the training data selection method selects appropriate training instances based on the target instances. The methods select training data based on target and training dataset similarity. The target dataset in CPDP is unlabeled, so the similarity is calculated solely on metric values, excluding class information. This situation increases the likelihood of selecting duplicates or inconsistent instances, especially for methods that ignore redundant data instances during data selection. In addition, methods such as Herbold13, PHe0114, and

PHe0214 that select training data at the project level granularity are more susceptible to including redundant instances. It is because the similarity between the training and target datasets is measured using the distance between their data distributions, which means that datasets are selected in aggregate rather than instance by instance. This condition prevents the detection of redundant instances. We see that Herbold13, ZHe13, PHe0114, and PHe0214 have a higher difference in N1 than other methods.

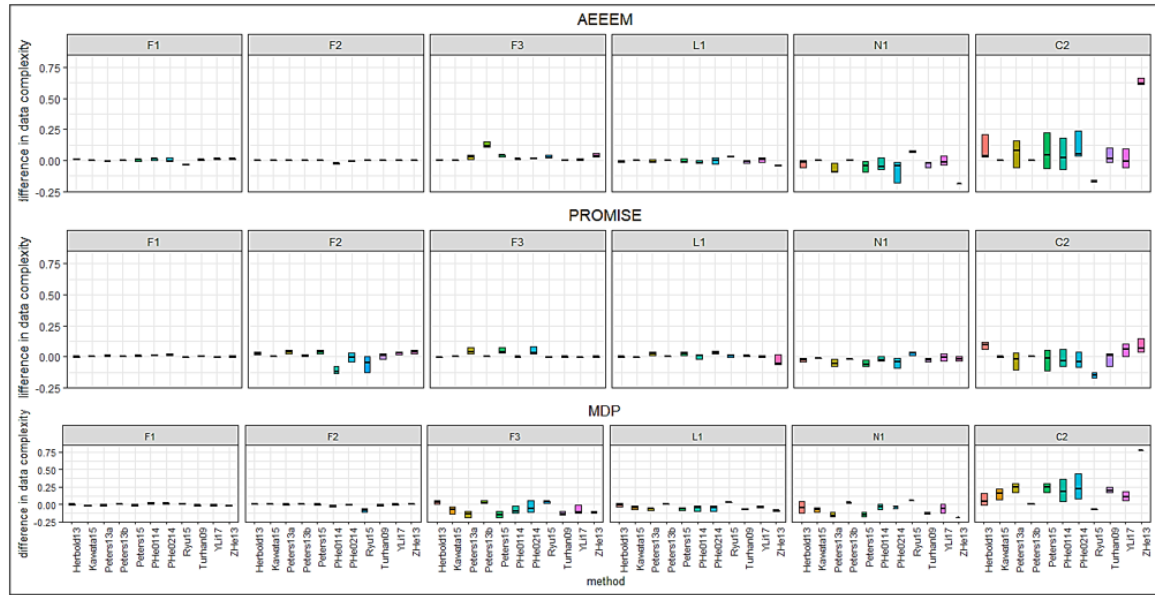


Figure 7. The impact of training data selection on the data complexity measures

Figure 8 (a)-(c) display the trend of the average value of F3, L1, N1, and C2 for each training data selection method for AEEEM, PROMISE, and MDP dataset, respectively. TRN represents the baseline method that uses all training data to build the prediction model. We observed that L1 and F3 tend to have constant values, especially in AEEEM and PROMISE. For N1 and C2, higher values indicate a more complex classification problem, resulting in lower performance. Except for Ryu15, the AUC tends to decrease as the values of N1 and C2 increase. It shows that N1 and C2 are suitable measures of the complexity of the defect data produced by data selection.

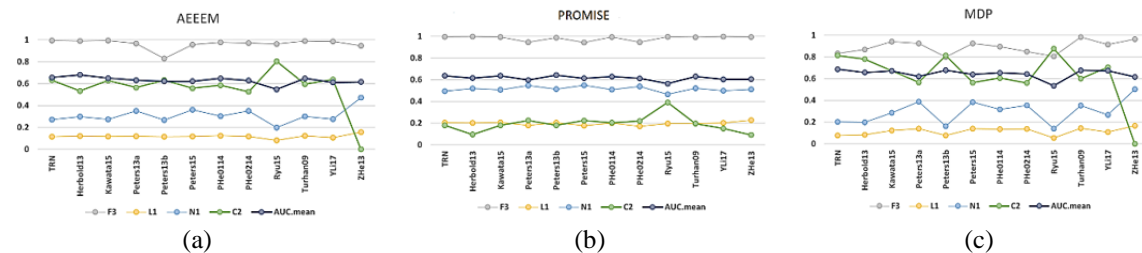


Figure 8. Average value of selected data complexity measures along with the AUC (a) AEEEM, (b) PROMISE, and (c) MDP

Based on the previous two-level analysis, training data selection could improve the baseline model. However, in Tables 6 (in Appendix), most improvements in prediction performance are statistically insignificant in that the improvement is relatively small. The training data selection methods also performed better on the PROMISE dataset than on the other datasets, see Figure 3(a). We suspect these statistically insignificant impacts relate to the complex characteristics of the defect datasets. The analysis reveals that the training datasets are essentially complex problems. They have high overlapping (high F3) and noisy instances and are highly imbalanced (high C2 for AEEEM and MDP). It makes such datasets more challenging to learn [54]. Hosseini *et al.*

[55] confirm this since they identified several datasets that are hard to predict using their proposed method. In our case, the analysis found that selecting training data did not necessarily reduce the complexity of the training data, and even the selection could increase the data complexity, Figure 7. It may result in insignificant performance gains. In some cases, after data selection, the prediction performance of the model even degrades. The imbalanced class ratio has a more significant effect on model performance than the noise ratio. As illustrated in Figure 3, the percentage of times a training data selection method improves the baseline methods in PROMISE is greater than the percentages for AEEEM and MDP. Figure 8 shows that the PROMISE datasets have lower C2 than the AEEEM and MDP datasets.

Focusing on training (TRN) dataset before selection, we can highlight several points in Figure 8: i) F3 is always a high value. A high F3 value indicates high overlap, ii) AEEEM and PROMISE have a high imbalanced rate, iii) the noise ratio is relatively high, at or above 20%. It implies that defect datasets are a complex problem because they present three challenges to CPDP: class imbalance, noisy dataset, and class overlapping. We suspect that these issues cause some selection methods to perform suboptimally. Answer to RQ3: training data selection may affect the data complexity measure, especially in N1 and C2, but the effect is not necessarily positive. The defect datasets are deemed to be complex data, which may cause some selection methods to have unsatisfied performance.

## 5. CONCLUSION

This study examined the impact of training data selection on the performance of an SDP model. We compared 13 training data selection methods using 61 releases of software from three dataset repositories (i.e., AEEEM, PROMISE, and MDP). We analyze in three dimensions: performance at a dataset release level, dataset repository level, and impact on data complexity. The results of the study are as follows: selecting training data could positively impact the defect prediction performance on most dataset releases. The best training data selection methods are different for each dataset release. When looking at cumulative performance on each dataset repository, most training data selection methods do not improve the baseline model, particularly the MDP dataset. We discovered that Herbold13 and Peters13b are the best methods on the AEEEM and PROMISE datasets. The classification algorithm affects the prediction performance on most training data selection methods, with the NB being the best-performing classifier. In addition, no single training data selection method consistently outperforms the others when paired with any classifier. Training data selection affects the data complexity measure, especially in N1 and C2, but the effect is not necessarily positive. The defect datasets are deemed to be complex data, which may cause some selection methods to have unsatisfied performance. Experimental results indicate that the best method for each dataset varies depending on the dataset and classifiers. For future works, an investigation on a recommendation system to select a suitable training data selection method for a particular dataset. We recommend dealing with noise and unbalanced classes when designing training data methods.

## ACKNOWLEDGEMENTS

This study has been supported by the Faculty of Information and Communication Technology Universiti Teknikal Malaysia Melaka, Malaysia, and the Technical Implementing Service unit of Information System, Faculty of Industrial Technology, Universitas Atma Jaya Yogyakarta, Indonesia.

## APPENDIX

Table 6. The performance of training data selection method for each dataset release. 'release' denotes the target or testing dataset instead of the training dataset

No	Release*	ALL	Herbold13	Kawata15	Menzies11	Peters13a	Peters13b	Peters15
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
1	eclipse	0.6751	<b>0.7247</b> (+, M)	0.6757 (+, N)	0.6319 (--, S)	0.6713 (--, N)	0.6486 (--, S)	0.6794 (+, N)
2	equinox	0.6669	<b>0.7019</b> (+, S)	0.6600 (--, N)	0.5736 (--, L)	0.5625 (--, L)	0.6145 (--, M)	0.6038 (--, M)
3	lucene	0.6362	<b>0.7075</b> (+, L)	0.6268 (--, N)	0.5478 (--, L)	0.6390 (+, N)	0.6327 (--, N)	0.5659 (--, M)
4	mylyn	0.6487	0.6110 (--, S)	0.6375 (--, N)	0.5509 (--, L)	0.6255 (--, S)	0.5768 (--, M)	0.6144 (--, S)
5	pde	0.6569	<b>0.6572</b> (+, N)	0.6515 (--, N)	0.5692 (--, L)	0.6545 (--, N)	0.6365 (--, N)	0.6445 (--, N)
6	ant-1.3	<b>0.7261</b>	0.6799 (--, S)	0.7182 (--, N)	0.6212 (--, L)	0.6501 (--, M)	0.7165 (--, N)	0.6053 (--, L)
7	ant-1.4	0.5457	0.5334 (--, S)	0.5397 (--, N)	0.5203 (--, L)	0.5211 (--, L)	0.5475 (+, N)	<b>0.5724</b> (+, L)
8	ant-1.5	0.6792	<b>0.7134</b> (+, S)	0.6666 (--, N)	0.6057 (--, M)	0.6144 (--, S)	0.7115 (+, S)	0.6430 (--, S)
9	ant-1.6	0.7188	<b>0.7297</b> (+, N)	0.7140 (--, N)	0.6196 (--, L)	0.6838 (--, S)	0.7296 (+, N)	0.6721 (--, S)
10	ant-1.7	0.6962	0.7029 (+, N)	0.6932 (--, N)	0.6087 (--, L)	0.6893 (--, N)	<b>0.7126</b> (+, N)	0.6874 (--, N)

Table 6. The performance of training data selection method for each dataset release. 'release' denotes the target or testing dataset instead of the training dataset (continue)

No	Release*	ALL	Herbold13	Kawata15	Menzies11	Peters13a	Peters13b	Peters15
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
11	arc	0.6526	0.6550 (+, N)	0.6625 (+, N)	0.5984 (--, M)	0.6363 (--, N)	0.6437 (--, N)	0.6744 (+, S)
12	camel-1.0	0.6350	0.5837 (--, S)	<b>0.6439</b> (+, N)	0.5986 (--, S)	0.5121 (--, L)	0.6417 (+, N)	0.5754 (--, M)
13	camel-1.2	0.5525	0.5426 (--, S)	0.5450 (--, S)	0.5216 (--, L)	0.5365 (--, M)	0.5528 (+, N)	0.5433 (--, S)
14	camel-1.4	0.6347	0.5751 (--, M)	0.6313 (--, N)	0.5498 (--, L)	0.5866 (--, M)	<b>0.6396</b> (+, N)	0.6234 (--, N)
15	camel-1.6	0.5528	<b>0.5734</b> (+, M)	0.5596 (+, N)	0.5302 (--, M)	0.5494 (--, N)	0.5724 (+, S)	0.5610 (+, N)
16	ivy-1.1	0.6528	0.6255 (--, S)	0.6345 (--, N)	0.5528 (--, L)	0.6072 (--, M)	<b>0.7090</b> (+, M)	0.5941 (--, M)
17	ivy-1.4	<b>0.6854</b>	0.6308 (--, M)	0.6831 (--, N)	0.6059 (--, L)	0.5858 (--, L)	0.6842 (--, N)	0.5611 (--, L)
18	ivy-2.0	0.7085	<b>0.7105</b> (+, N)	0.7003 (--, N)	0.6307 (--, L)	0.6618 (--, S)	0.7080 (--, N)	0.6991 (--, N)
19	jedit-3.2	0.7033	0.6091 (--, M)	0.7077 (+, N)	0.6125 (--, L)	0.6311 (--, M)	0.7108 (+, N)	0.6823 (--, N)
20	jedit-4.0	0.6769	0.5885 (--, L)	0.6776 (+, N)	0.6033 (--, L)	0.6313 (--, M)	0.6674 (--, N)	0.6382 (--, S)
21	jedit-4.1	0.7096	0.5908 (--, L)	0.7199 (+, N)	0.6540 (--, M)	0.6786 (--, S)	<b>0.7200</b> (+, N)	0.6575 (--, M)
22	jedit-4.2	0.7460	0.6328 (--, L)	0.7638 (+, N)	0.6824 (--, M)	0.6998 (--, S)	0.7607 (+, N)	0.6980 (--, S)
23	jedit-4.3	0.5812	0.5044 (--, L)	0.6005 (+, S)	0.5418 (--, L)	0.5361 (--, L)	0.5836 (+, N)	0.6023 (+, S)
24	log4j-1.0	0.7114	0.6983 (--, N)	0.7004 (--, N)	0.5895 (--, L)	0.6584 (--, S)	0.7056 (--, N)	0.6406 (--, M)
25	log4j-1.1	0.6984	0.7099 (+, N)	0.7045 (+, N)	0.5844 (--, L)	0.5685 (--, L)	0.6889 (--, N)	0.7098 (+, N)
26	log4j-1.2	0.5353	0.5574 (+, M)	0.5569 (+, S)	0.5437 (+, N)	0.4861 (--, L)	0.5580 (+, S)	0.5694 (+, S)
27	lucene-2.0	0.6417	<b>0.7058</b> (+, L)	0.6492 (+, N)	0.5583 (--, L)	0.6629 (+, S)	0.6719 (+, S)	0.6288 (--, N)
28	lucene-2.2	0.5842	<b>0.6244</b> (+, L)	0.5838 (--, N)	0.5305 (--, L)	0.5948 (+, N)	0.6058 (+, S)	0.5658 (--, S)
29	lucene-2.4	0.5972	0.6371 (+, M)	0.6057 (+, N)	0.5372 (--, L)	0.5610 (--, S)	0.6091 (+, N)	0.5971 (--, N)
30	poi-1.5	0.6365	0.6384 (+, N)	0.6335 (--, N)	0.5537 (--, L)	0.6235 (--, N)	<b>0.6402</b> (+, N)	0.5703 (--, M)
31	poi-2.0	0.5718	0.4833 (--, L)	0.5522 (--, S)	0.5550 (--, S)	0.5703 (--, N)	0.5549 (--, S)	0.6350 (+, M)
32	poi-2.5	0.6586	0.6261 (--, S)	<b>0.6987</b> (+, S)	0.5714 (--, L)	0.6413 (--, N)	0.6907 (+, S)	0.6510 (--, N)
33	poi-3.0	0.6752	0.6602 (--, N)	0.6911 (+, N)	0.5546 (--, L)	0.6783 (+, N)	<b>0.7286</b> (+, S)	0.6980 (+, N)
34	redaktor	<b>0.6569</b>	0.5897 (--, M)	0.6480 (--, N)	0.5923 (--, M)	0.5962 (--, M)	0.6280 (--, S)	0.5150 (--, L)
35	synapse-1.0	0.6939	0.6623 (--, S)	0.6823 (--, N)	0.5704 (--, L)	0.5739 (--, L)	<b>0.6964</b> (+, N)	0.6073 (--, L)
36	synapse-1.1	0.6127	<b>0.6474</b> (+, S)	0.6113 (--, N)	0.5218 (--, L)	0.5431 (--, M)	0.6382 (+, S)	0.6218 (+, N)
37	synapse-1.2	0.6642	<b>0.6886</b> (+, S)	0.6642 (+, N)	0.5454 (--, L)	0.6280 (--, S)	0.6885 (+, S)	0.6431 (--, S)
38	tomcat	0.6827	0.6986 (+, N)	0.6895 (+, N)	0.6457 (--, S)	0.6716 (--, N)	<b>0.7338</b> (+, S)	0.5839 (--, M)
39	velocity-1.4	0.4313	0.4682 (+, S)	0.4307 (--, N)	0.4609 (+, M)	0.4663 (+, S)	0.4248 (--, N)	0.4881 (+, L)
40	velocity-1.5	0.5698	0.5119 (--, M)	0.5712 (+, N)	0.5105 (--, L)	0.5223 (--, M)	0.6058 (+, M)	0.5912 (+, S)
41	velocity-1.6	0.6352	0.5759 (--, M)	0.6306 (--, N)	0.5360 (--, L)	0.5718 (--, M)	<b>0.6491</b> (+, N)	0.5937 (--, M)
42	xalan-2.4	0.6657	0.6599 (--, N)	0.6695 (+, N)	0.5959 (--, L)	0.6560 (--, N)	0.6696 (+, N)	0.6458 (--, S)
43	xalan-2.5	0.5657	0.5464 (--, M)	0.5620 (--, N)	0.5372 (--, L)	0.5388 (--, M)	0.5683 (+, N)	0.5589 (--, N)
44	xalan-2.6	0.5931	0.5952 (+, N)	0.5701 (--, S)	0.5613 (--, M)	<b>0.6120</b> (+, S)	0.5853 (--, N)	0.5894 (--, N)
45	xalan-2.7	0.6998	<b>0.7434</b> (+, S)	0.6771 (--, S)	0.5869 (--, L)	0.5734 (--, L)	0.7260 (+, S)	0.6142 (--, M)
46	xerces-1.2	0.4836	0.4618 (--, M)	0.4862 (+, N)	0.5211 (+, L)	0.5051 (+, M)	0.5236 (+, M)	0.4792 (--, N)
47	xerces-1.3	0.6977	0.6056 (--, L)	0.7028 (+, N)	0.5835 (--, L)	0.6561 (--, S)	0.5802 (--, M)	0.6789 (--, N)
48	xerces-1.4	0.6839	0.5824 (--, L)	0.6826 (--, N)	0.5604 (--, L)	0.6873 (+, N)	0.4997 (--, L)	<b>0.7121</b> (+, S)
49	xerces-init	0.4695	0.4904 (+, M)	0.4920 (+, L)	0.5213 (+, L)	0.4230 (--, L)	<b>0.5711</b> (+, L)	0.4805 (+, S)
50	CM1	0.6893	0.6287 (--, M)	0.6895 (+, N)	0.5234 (--, L)	0.6493 (--, S)	0.7022 (+, N)	0.6130 (--, L)
51	JM1	0.6484	0.5541 (--, L)	0.6477 (--, N)	0.5389 (--, L)	0.6395 (--, N)	<b>0.6509</b> (+, N)	0.6486 (+, N)
52	KC1	<b>0.7116</b>	0.6200 (--, M)	0.7053 (--, N)	0.5306 (--, L)	0.6566 (--, M)	0.7048 (--, N)	0.6391 (--, M)
53	KC3	0.6451	0.5701 (--, L)	0.6311 (--, N)	0.5023 (--, L)	0.5858 (--, L)	<b>0.6473</b> (+, N)	0.5685 (--, L)
54	MC1	0.7902	0.7916 (+, N)	0.7862 (--, N)	0.5184 (--, L)	0.7768 (--, N)	0.7860 (--, N)	<b>0.8030</b> (+, N)
55	MC2	0.6339	0.6256 (--, N)	0.6106 (--, S)	0.5326 (--, L)	0.5573 (--, L)	0.6363 (+, N)	0.6240 (--, N)
56	MW1	<b>0.7114</b>	0.6006 (--, L)	0.6959 (--, N)	0.4980 (--, L)	0.5830 (--, L)	0.6930 (--, N)	0.5940 (--, L)
57	PC1	0.6864	0.6913 (+, N)	0.6849 (--, N)	0.5608 (--, L)	0.5152 (--, L)	<b>0.7004</b> (+, N)	0.6198 (--, M)
58	PC2	0.6459	0.7574 (+, L)	0.5637 (--, S)	0.5237 (--, L)	0.5121 (--, M)	0.6075 (--, S)	0.5492 (--, M)
59	PC3	0.6626	<b>0.7082</b> (+, M)	0.6446 (--, N)	0.5235 (--, L)	0.6246 (--, S)	0.6400 (--, S)	0.5996 (--, M)
60	PC4	0.6064	0.6048 (--, N)	0.6005 (--, N)	0.5076 (--, L)	0.6237 (+, S)	0.5595 (--, M)	0.6187 (+, N)
61	PC5	0.8166	0.7325 (--, S)	0.7918 (--, N)	0.5715 (--, L)	0.7225 (--, M)	0.7982 (--, N)	0.7885 (--, N)
mean AEEEM		0,6568	0,6805	0,6503	0,5747	0,6305	0,6219	0,6216
mean PROMISE		0,6358	0,6148	0,6365	0,5701	0,5973	0,6421	0,6127
mean MDP		0,6873	0,6571	0,6710	0,5276	0,6205	0,6772	0,6388

Table 6. The performance of training data selection method for each dataset release. 'release' denotes the target or testing dataset instead of the training dataset (continue)

No	Release*	PHe0114	PHe0214	Ryu15	Turhan09	YLi17	ZHe13	%impr
(1)	(2)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
1	eclipse	0.6448 (--, S)	0.6832 (+, N)	0.5323 (--, L)	0.6319 (--, S)	0.6636 (--, N)	0.6408 (--, S)	7,35
2	equinox	0.6757 (+, N)	0.5903 (--, M)	0.5508 (--, L)	0.5736 (--, L)	0.5419 (--, L)	0.6718 (+, N)	5,24
3	lucene	0.6322 (--, N)	0.6008 (--, S)	0.5727 (--, L)	0.5478 (--, L)	0.6305 (--, N)	0.5580 (--, L)	11,21
4	mylyn	<b>0.6688</b> (+, N)	0.6464 (--, N)	0.5356 (--, L)	0.5509 (--, L)	0.5734 (--, L)	0.5832 (--, L)	3,10



Table 6. The performance of training data selection method for each dataset release. 'release' denotes the target or testing dataset instead of the training dataset (continue)

No	Release*	PHe0114	PHe0214	Ryu15	Turhan09	YLi17	ZHe13	%impr
(1)	(2)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
1	eclipse	0.6448 (--, S)	0.6832 (+, N)	0.5323 (--, L)	0.6319 (--, S)	0.6636 (--, N)	0.6408 (--, S)	7,35
2	equinox	0.6757 (+, N)	0.5903 (--, M)	0.5508 (--, L)	0.5736 (--, L)	0.5419 (--, L)	0.6718 (+, N)	5,24
3	lucene	0.6322 (--, N)	0.6008 (--, S)	0.5727 (--, L)	0.5478 (--, L)	0.6305 (--, N)	0.5580 (--, L)	11,21
4	mylyn	<b>0.6688</b> (+, N)	0.6464 (--, N)	0.5356 (--, L)	0.5509 (--, L)	0.5734 (--, L)	0.5832 (--, L)	3,10
5	pde	0.6195 (--, S)	0.6176 (--, S)	0.5487 (--, L)	0.5692 (--, L)	0.6457 (--, N)	0.5972 (--, L)	0,05
6	ant-1.3	0.6542 (--, M)	0.6087 (--, L)	0.5418 (--, L)	0.6500 (--, M)	0.6973 (--, S)	0.5954 (--, L)	0,00
7	ant-1.4	0.5214 (--, M)	0.5371 (--, N)	0.5334 (--, S)	0.5139 (--, M)	0.5284 (--, M)	0.5125 (--, L)	4,90
8	ant-1.5	0.6514 (--, S)	0.5973 (--, M)	0.6023 (--, M)	0.6332 (--, S)	0.6622 (--, N)	0.6362 (--, S)	5,04
9	ant-1.6	0.6616 (--, S)	0.6358 (--, M)	0.6152 (--, L)	0.6797 (--, S)	0.6786 (--, S)	0.6782 (--, S)	1,52
10	ant-1.7	0.6774 (--, N)	0.6795 (--, N)	0.6104 (--, L)	0.6864 (--, N)	0.6724 (--, S)	0.7061 (+, N)	2,36
11	arc	0.6990 (+, S)	<b>0.7034</b> (+, M)	0.5287 (--, L)	0.6848 (+, S)	0.6256 (--, S)	0.5859 (--, L)	7,78
12	camel-1.0	0.6236 (--, N)	0.5464 (--, L)	0.5428 (--, L)	0.6405 (+, N)	0.6280 (--, N)	0.6211 (--, N)	1,39
13	camel-1.2	0.5573 (+, N)	0.5328 (--, L)	0.5280 (--, L)	<b>0.5786</b> (+, M)	0.5545 (+, N)	0.5345 (--, M)	4,72
14	camel-1.4	0.6367 (+, N)	0.6148 (--, S)	0.5498 (--, L)	0.6344 (--, N)	0.6317 (--, N)	0.5963 (--, M)	0,77
15	camel-1.6	0.5524 (--, N)	0.5725 (+, S)	0.5283 (--, M)	<b>0.5766</b> (+, M)	0.5670 (+, S)	0.5668 (+, S)	4,31
16	ivy-1.1	0.6336 (--, N)	0.6407 (--, N)	0.5606 (--, L)	0.6494 (--, N)	0.6132 (--, S)	0.6361 (--, S)	8,60
17	ivy-1.4	0.6101 (--, L)	0.5942 (--, L)	0.5629 (--, L)	0.6200 (--, M)	0.6590 (--, S)	0.6108 (--, L)	0,00
18	ivy-2.0	0.6655 (--, S)	0.6891 (--, N)	0.5778 (--, L)	0.6645 (--, S)	0.6631 (--, S)	0.6828 (--, S)	0,29
19	jedit-3.2	<b>0.7182</b> (+, N)	0.6623 (--, S)	0.6386 (--, M)	0.7004 (--, N)	0.4814 (--, L)	0.6789 (--, S)	2,11
20	jedit-4.0	0.6946 (+, S)	0.6620 (--, N)	0.6172 (--, M)	<b>0.7114</b> (+, S)	0.5726 (--, L)	0.6877 (+, N)	5,10
21	jedit-4.1	0.6858 (--, S)	0.6608 (--, S)	0.6013 (--, L)	0.6948 (--, N)	0.5658 (--, L)	0.6906 (--, S)	1,47
22	jedit-4.2	0.7576 (+, N)	0.6953 (--, S)	0.6500 (--, L)	<b>0.7699</b> (+, S)	0.6002 (--, L)	0.7098 (--, S)	3,20
23	jedit-4.3	0.5987 (+, S)	0.5813 (+, N)	0.4887 (--, L)	0.6156 (+, M)	0.5626 (--, S)	<b>0.6165</b> (+, L)	6,07
24	log4j-1.0	0.6991 (--, N)	0.6318 (--, M)	0.5727 (--, L)	0.7081 (--, N)	<b>0.7114</b> (+, N)	0.6588 (--, M)	0,00
25	log4j-1.1	0.7139 (+, N)	<b>0.7260</b> (+, S)	0.5529 (--, L)	0.7008 (+, N)	0.7202 (+, N)	0.6369 (--, M)	3,95
26	log4j-1.2	0.5351 (--, N)	<b>0.5952</b> (+, L)	0.5009 (--, L)	0.5504 (+, S)	0.5660 (+, S)	0.5715 (+, M)	11,18
27	lucene-2.0	0.6498 (+, N)	0.6134 (--, S)	0.5454 (--, L)	0.6446 (+, N)	0.6358 (--, N)	0.6488 (+, N)	9,99
28	lucene-2.2	0.5689 (--, S)	0.5886 (+, N)	0.5439 (--, M)	0.5798 (--, N)	0.5580 (--, S)	0.5968 (+, S)	6,88
29	lucene-2.4	0.6070 (+, N)	0.6078 (+, N)	0.5378 (--, L)	0.5961 (--, N)	<b>0.6600</b> (+, M)	0.6295 (+, M)	10,52
30	poi-1.5	0.6288 (--, N)	0.6082 (--, S)	0.5674 (--, L)	0.6216 (--, N)	0.6336 (--, N)	0.5940 (--, M)	0,57
31	poi-2.0	0.6038 (+, S)	0.6124 (+, M)	0.5654 (--, N)	<b>0.6362</b> (+, M)	0.5127 (--, M)	0.5660 (--, N)	11,26
32	poi-2.5	0.6457 (--, N)	0.6380 (--, N)	0.5796 (--, L)	0.6190 (--, S)	0.6408 (--, N)	0.5296 (--, L)	6,09
33	poi-3.0	0.6756 (+, N)	0.6891 (+, N)	0.5825 (--, L)	0.6628 (--, N)	0.6619 (--, N)	0.6106 (--, M)	7,90
34	redaktor	0.5948 (--, M)	0.4816 (--, L)	0.5265 (--, L)	0.5364 (--, L)	0.4933 (--, L)	0.5290 (--, L)	0,00
35	synapse-1.0	0.6594 (--, S)	0.6381 (--, S)	0.6275 (--, M)	0.6630 (--, S)	0.6837 (--, N)	0.5340 (--, L)	0,36
36	synapse-1.1	0.5829 (--, S)	0.6317 (+, S)	0.5396 (--, L)	0.5873 (--, S)	0.6379 (+, S)	0.5664 (--, M)	5,67
37	synapse-1.2	0.6618 (--, N)	0.6489 (--, N)	0.5772 (--, L)	0.6509 (--, N)	0.6248 (--, S)	0.5885 (--, L)	3,67
38	tomcat	0.6358 (--, S)	0.6759 (--, N)	0.6101 (--, M)	0.6248 (--, S)	0.7063 (+, N)	0.6885 (+, N)	7,49
39	velocity-1.4	0.4906 (+, L)	0.4834 (+, L)	0.4745 (+, L)	<b>0.5130</b> (+, L)	0.3988 (--, S)	0.3923 (--, M)	18,94
40	velocity-1.5	0.5862 (+, S)	0.5898 (+, S)	0.5230 (--, M)	0.5800 (+, N)	0.6022 (+, S)	<b>0.6265</b> (+, L)	9,95
41	velocity-1.6	0.6068 (--, S)	0.5582 (--, L)	0.5553 (--, L)	0.6107 (--, S)	0.5830 (--, M)	0.5953 (--, M)	2,19
42	xalan-2.4	0.6732 (+, N)	0.6296 (--, S)	0.5891 (--, L)	<b>0.6769</b> (+, N)	0.6494 (--, N)	0.6583 (--, N)	1,69
43	xalan-2.5	0.5806 (+, S)	0.5582 (--, S)	0.5384 (--, L)	<b>0.5891</b> (+, M)	0.5544 (--, S)	0.5544 (--, S)	4,14
44	xalan-2.6	0.5802 (--, S)	0.5730 (--, S)	0.5831 (--, N)	0.5936 (+, N)	0.5884 (--, N)	0.5782 (--, S)	3,19
45	xalan-2.7	0.7165 (+, N)	0.5920 (--, L)	0.6481 (--, M)	0.7274 (+, S)	0.6865 (--, N)	0.6682 (--, S)	6,23
46	xerces-1.2	0.4887 (+, N)	0.4818 (--, N)	0.5030 (+, M)	0.4762 (--, S)	0.4792 (--, N)	<b>0.5252</b> (+, L)	8,60
47	xerces-1.3	0.7056 (+, N)	0.6824 (--, N)	0.6063 (--, L)	<b>0.7162</b> (+, N)	0.5517 (--, L)	0.6104 (--, L)	2,65
48	xerces-1.4	0.6900 (+, N)	0.6991 (+, N)	0.5656 (--, L)	0.7091 (+, N)	0.5697 (--, M)	0.6400 (--, M)	4,12
49	xerces-init	0.5078 (+, L)	0.4933 (+, L)	0.5148 (+, L)	0.4524 (--, M)	0.4805 (+, S)	0.4754 (+, N)	21,64
50	CM1	0.6492 (--, S)	0.5760 (--, L)	0.5298 (--, L)	0.6435 (--, S)	0.6542 (--, S)	0.6431 (--, M)	1,87
51	JM1	0.6306 (--, S)	0.6358 (--, N)	0.5548 (--, L)	0.6452 (--, N)	0.5942 (--, M)	0.5122 (--, L)	0,38
52	KC1	0.6817 (--, S)	0.6797 (--, S)	0.5357 (--, L)	0.6930 (--, N)	0.6790 (--, S)	0.5796 (--, L)	0,00
53	KC3	0.5940 (--, M)	0.5850 (--, M)	0.5286 (--, L)	0.6102 (--, S)	0.6314 (--, N)	0.5790 (--, L)	0,35
54	MC1	0.7196 (--, S)	0.7385 (--, S)	0.5584 (--, L)	0.7920 (+, N)	0.7666 (--, N)	0.7518 (--, S)	1,62
55	MC2	<b>0.6457</b> (+, N)	0.5986 (--, S)	0.5640 (--, L)	0.6132 (--, S)	0.6192 (--, S)	0.5838 (--, M)	1,86
56	MW1	0.6689 (--, S)	0.6408 (--, M)	0.5104 (--, L)	0.6578 (--, S)	0.6779 (--, S)	0.5873 (--, L)	0,00
57	PC1	0.6325 (--, M)	0.6318 (--, M)	0.5133 (--, L)	0.6901 (+, N)	0.6489 (--, S)	0.6470 (--, S)	2,04
58	PC2	0.6437 (--, N)	0.6765 (+, N)	0.5271 (--, L)	0.6977 (+, S)	<b>0.7872</b> (+, L)	0.6990 (+, S)	21,87
59	PC3	0.5651 (--, L)	0.5837 (--, L)	0.5212 (--, L)	0.6287 (--, S)	0.6474 (--, N)	0.5511 (--, L)	6,88
60	PC4	<b>0.6454</b> (+, S)	0.6180 (+, N)	0.5153 (--, L)	0.6205 (+, N)	0.5488 (--, M)	0.6059 (--, N)	6,42
61	PC5	0.7706 (--, S)	0.7460 (--, S)	0.5603 (--, L)	0.8234 (+, N)	<b>0.8311</b> (+, N)	0.5664 (--, L)	1,78
mean ABEEM		0,6482	0,6276	0,5480	0,5747	0,6110	0,6102	3,61
mean PROMISE		0,6293	0,6123	0,5638	0,6302	0,6035	0,6050	1,00
mean MDP		0,6539	0,6425	0,5349	0,6763	0,6738	0,6089	0,00






## REFERENCES

- [1] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, Oct. 2016, doi: 10.1007/s10664-015-9396-2.
- [2] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, Aug. 2000, doi: 10.1109/32.879815.
- [3] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, May 2007, doi: 10.1109/TSE.2007.1005.
- [4] P. He, B. Li, D. Zhang, and Y. Ma, "Simplification of training data for cross-project defect prediction," *arXiv preprint arXiv:1405.0773*, May 2014, doi: 10.11772/j.issn.1001-9081.2016.0000.
- [5] Y. Zhou et al., "How far we have progressed in the journey? an examination of cross-project defect prediction," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 1, pp. 1–51, Jun. 2018, doi: 10.1145/3183339.
- [6] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings - International Conference on Software Engineering*, May 2013, pp. 382–391, doi: 10.1109/ICSE.2013.6606584.
- [7] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100, doi: 10.1145/1595696.1595713.
- [8] A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," *2009 3rd Int. Symp. Empir. Softw. Eng. Meas. ESEM 2009*, pp. 460–463, 2009, doi: 10.1109/ESEM.2009.5316002.
- [9] Q. Zou, L. Lu, Z. Yang, X. Gu, and S. Qiu, "Joint feature representation learning and progressive distribution matching for cross-project defect prediction," *Information and Software Technology*, vol. 137, p. 106588, Sep. 2021, doi: 10.1016/j.infsof.2021.106588.
- [10] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, pp. 125–136, Mar. 2019, doi: 10.1016/j.infsof.2018.11.005.
- [11] F. Wu, X. Zheng, Y. Sun, Y. Gao, and X.-Y. Jing, "Joint domain adaption and pseudo-labeling for cross-project defect prediction," *IEICE Transactions on Information and Systems*, vol. 105, no. 2, pp. 432–435, 2022, doi: 10.1587/transinf.2021EDL8061.
- [12] Z. Xu et al., "Cross project defect prediction via balanced distribution adaptation based transfer learning," *Journal of Computer Science and Technology*, vol. 34, no. 5, pp. 1039–1062, 2019, doi: 10.1007/s11390-019-1959-z.
- [13] M. F. Sohan, M. A. Kabir, M. Rahman, S. M. H. Mahmud, and T. Bhuiyan, "Training data selection using ensemble dataset approach for software defect prediction," *International Conference on Cyber Security and Computer Science*, 2020, pp. 243–256.
- [14] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Training data selection for imbalanced cross-project defect prediction," *Computers & Electrical Engineering*, vol. 94, p. 107370, 2021, doi: 10.1016/j.compeleceng.2021.107370.
- [15] Z. Sun, J. Li, H. Sun, and L. He, "CFPS: collaborative filtering based source projects selection for cross-project defect prediction," *Applied Soft Computing*, vol. 99, p. 106940, Feb. 2021, doi: 10.1016/j.asoc.2020.106940.
- [16] S. Herbold, "Training data selection for cross-project defect prediction," in *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, Oct. 2013, pp. 1–10, doi: 10.1145/2499393.2499395.
- [17] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: an empirical study on defect prediction," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2013, pp. 45–54, doi: 10.1109/ESEM.2013.20.
- [18] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, Oct. 2009, doi: 10.1007/s10664-008-9103-7.
- [19] F. Zhang, I. Keivanloo, and Y. Zou, "Data transformation in cross-project defect prediction," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3186–3218, Dec. 2017, doi: 10.1007/s10664-017-9516-2.
- [20] Y. Jiang, B. Cukic, and T. Menzies, "Can data transformation help in the detection of fault-prone modules?," in *DEFECTS'08: 2008 International Symposium on Software Testing and Analysis-Proceedings of the 2008 Workshop on Defects in Large Software Systems 2008, DEFECTS'08*, Jul. 2008, pp. 16–20, doi: 10.1145/1390817.1390822.
- [21] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, Jun. 2012, doi: 10.1007/s10515-011-0090-3.
- [22] Y. Li, Z. Huang, Y. Wang, and B. Fang, "Evaluating data filter on cross-project defect prediction: comparison and improvements," *IEEE Access*, vol. 5, pp. 25646–25656, 2017, doi: 10.1109/ACCESS.2017.2771460.
- [23] Y. Bin, K. Zhou, H. Lu, Y. Zhou, and B. Xu, "Training data selection for cross-project defection prediction: which approach is better?," in *IACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2017, pp. 354–363, doi: 10.1109/ESEM.2017.49..
- [24] H. Luo, H. Dai, W. Peng, W. Hu, and F. Li, "An empirical study of training data selection methods for ranking-oriented cross-project defect prediction," *Sensors*, vol. 21, no. 22, p. 7535, 2021, doi: 10.3390/s21227535.
- [25] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, Sep. 2018, doi: 10.1109/TSE.2017.2724538.
- [26] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 289–300, 2002, doi: 10.1109/34.990132.
- [27] J. Cano, "Analysis of data complexity measures for classification," *Expert systems with applications*, vol. 40, no. 12, pp. 4820–4831, Sep. 2013, doi: 10.1016/j.eswa.2013.02.025.
- [28] A. C. Lorena, L. P. F. Garcia, J. Lehmann, M. C. P. Souto, and T. K. A. M. Ho, "How complex is your classification problem?: A survey on measuring classification complexity," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–34, 2019, doi: 10.1145/3347711.
- [29] F. Peters, T. Menzies, and L. Layman, "LACE2: better privacy-preserving data sharing for cross project defect prediction," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 801–811, May 2015, doi: 10.1109/ICSE.2015.92.
- [30] B. L. Sinaga, S. Ahmad, and Z. A. Abas, "A review of training data selection in software defect prediction," *Journal of Theoretical and Applied Information Technology*, vol. 98, no. 12, pp. 2092–2108, 2020.
- [31] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *IEEE International Working Conference on Mining Software Repositories*, 2013, pp. 409–418, doi: 10.1109/MSR.2013.6624057.
- [32] P. He, Y. He, L. Yu, and B. Li, "An improved method for cross-project defect prediction by simplifying training data," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–18, Jun. 2018, doi: 10.1155/2018/2650415.
- [33] K. Kawata, S. Amasaki, and T. Yokogawa, "Improving relevancy filter methods for cross-project defect prediction," in *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, Jul. 2015, pp. 2–7, doi: 10.1109/ACIT-CSI.2015.104.
- [34] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*,





- 2011, pp. 343–351, doi: 10.1109/ASE.2011.6100072.
- [35] V. H. Barella, L. P. F. Garcia, M. C. P. de Souto, A. C. Lorena, and A. C. P. L. F. de Carvalho, “Assessing the data complexity of imbalanced datasets,” *Information Sciences*, vol. 553, pp. 83–109, Apr. 2021, doi: 10.1016/j.ins.2020.12.006.
  - [36] L. C. Okimoto, R. M. Savii, and A. C. Lorena, “Complexity measures effectiveness in feature selection,” in *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, Oct. 2017, pp. 91–96, doi: 10.1109/BRACIS.2017.66.
  - [37] N. T. Dong and M. Khosla, “Revisiting feature selection with data complexity,” in *Proceedings-IEEE 20th International Conference on Bioinformatics and Bioengineering, BIBE 2020*, 2020, pp. 211–216, doi: 10.1109/BIBE50027.2020.00042.
  - [38] L. P. F. Garcia, A. C. Lorena, M. C. P. De Souto, and T. K. Ho, “Classifier recommendation using data complexity measures,” in *Proceedings - International Conference on Pattern Recognition*, 2018, pp. 874–879, doi: 10.1109/ICPR.2018.8545110.
  - [39] L. Morán-Fernández, V. Bolón-Canedo, and A. Alonso-Betanzos, “Can classification performance be predicted by complexity measures? A study using microarray data,” *Knowledge and Information Systems*, vol. 51, no. 3, pp. 1067–1090, 2017, doi: 10.1007/s10115-016-1003-3.
  - [40] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing*, vol. 27, pp. 504–518, Feb. 2015, doi: 10.1016/j.asoc.2014.11.023.
  - [41] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “Machine learning based methods for software fault prediction: a survey,” *Expert Systems with Applications*, vol. 172, p. 114595, Jun. 2021, doi: 10.1016/j.eswa.2021.114595.
  - [42] S. Hosseini, B. Turhan, and D. Gunarathna, “A Systematic literature review and meta-analysis on cross project defect prediction,” *IEEE Transactions on Software Engineering*, vol. X, no. 2, pp. 111–147, 2019, doi: 10.1109/TSE.2017.2770124.
  - [43] M. Promise and L. 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*, 2010, p. 1, doi: 10.1145/1868328.1868342.
  - [44] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.
  - [45] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, “The misuse of the NASA metrics data program data sets for automated software defect prediction,” *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 2011, pp. 96–103, doi: 10.1049/ic.2011.0012.
  - [46] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, “The jinx on the NASA software defect data sets,” in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Jun. 2016, pp. 1–5, doi: 10.1145/2915970.2916007.
  - [47] M. D'Ambros, M. Lanza, R. Robbes, M. D. Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” *2010 7th IEEE working conference on mining software repositories (MSR 2010)*, pp. 31–41, 2010, doi: 10.1109/MSR.2010.5463279.
  - [48] S. Herbold, A. Trautsch, and J. Grabowski, “Global vs. local models for cross-project defect prediction: A replication study,” *Empirical software engineering*, vol. 22, no. 4, pp. 1866–1902, 2017, doi: 10.1007/s10664-016-9468-y.
  - [49] D. Ryu, J.-I. Jang, and J. Baik, “A hybrid instance selection using nearest-neighbor for cross-project defect prediction,” *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 969–980, Sep. 2015, doi: 10.1007/s11390-015-1575-5.
  - [50] F. Peters, T. Menzies, L. Gong, and H. Zhang, “Balancing privacy and utility in cross-company defect prediction,” *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1054–1068, 2013, doi: 10.1109/TSE.2013.6.
  - [51] M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: the use of machine learning in software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, Jun. 2014, doi: 10.1109/TSE.2014.2322358.
  - [52] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “The impact of automated parameter optimization on defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, Jul. 2019, doi: 10.1109/TSE.2018.2794977.
  - [53] 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*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012, doi: 10.1109/TSE.2011.103.
  - [54] D. Bowes, T. Hall, and J. Petrić, “Software defect prediction: do different classifiers find the same defects?,” *Software Quality Journal*, vol. 26, no. 2, pp. 525–552, 2018, doi: 10.1007/s11219-016-9353-3.
  - [55] S. Hosseini, B. Turhan, and M. Mäntylä, “A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction,” *Information and Software Technology*, vol. 95, pp. 296–312, Mar. 2018, doi: 10.1016/j.infsof.2017.06.004.
  - [56] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, “Implications of ceiling effects in defect predictors,” in *Proceedings of the 4th international workshop on Predictor models in software engineering - PROMISE '08*, 2008, p. 47, doi: 10.1145/1370788.1370801.
  - [57] Q. Yu, S. Jiang, and Y. Zhang, “The performance stability of defect prediction models with class imbalance: An empirical study,” *IEICE Transactions on Information and Systems*, vol. 100, no. 2, pp. 265–272, 2017, doi: 10.1587/transinf.2016EDP7204.
  - [58] C. Catal, “Software fault prediction: A literature review and current trends,” *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011, doi: 10.1016/j.eswa.2010.10.024.
  - [59] L. P. F. Garcia, A. C. P. L. F. de Carvalho, and A. C. Lorena, “Effect of label noise in the complexity of classification problems,” *Neurocomputing*, vol. 160, pp. 108–119, 2015, doi: 10.1016/j.neucom.2014.10.085.
  - [60] L. P. F. Garcia, A. C. P. L. F. De Carvalho, and A. C. Lorena, “Noisy data set identification,” *International Conference on Hybrid Artificial Intelligence Systems*, vol. 8073, pp. 629–638, 2013, doi: 10.1007/978-3-642-40846-5\_63.

## BIOGRAPHIES OF AUTHORS







**Benyamin Langgu Sinaga**    received bachelor degree in electrical engineering from Department of Electrical Engineering, Gadjah Mada University, Yogyakarta, Indonesia in 1994. He obtained master degree in computer science from School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia in 2000. He currently pursuing Ph.D degree at Faculty of Information and Communication Technology, Universiti Teknikal Malaysia, Melaka. His research interests include software engineering, information system adoption, and machine learning. He can be contacted at email: benyamin.sinaga@uajy.ac.id.







**Sabrina Ahmad**     is an Associate Professor at the Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka. She has qualifications and undergone formal trainings in the area of software engineering and development. She is specialized in requirements engineering in both research and practice. She obtained her Ph.D in Computer Science from The University of Western Australia in 2012. She endeavors to maintain the bridge between academia and practitioners and therefore continue strengthening software engineering curriculum and apply the knowledge in the industries through consultation projects. Therefore, she continues to upgrade her skills and knowledge through professional training and certification. She obtained professional certification in requirements engineering, a certified tester and a certified professional IT architect. She can be contacted at email: [sabrinaahmad@utem.edu.my](mailto:sabrinaahmad@utem.edu.my).



**Zuraida Abal Abas**     is currently an Associate Professor at the Department of Intelligent Computing & Analytics, Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka. She obtained her Ph.D in Mathematics at Universiti Teknologi Malaysia; her M.Sc in Operational Research at London School of Economics, United Kingdom; and her B.Sc in Industrial Mathematics at Universiti Teknologi Malaysia. Her research interests are operational research, analytics, modeling and simulation. She can be contacted at email: [zuridaa@utem.edu.my](mailto:zuridaa@utem.edu.my).



**Antasena Wahyu Anggarajati**     received bachelor degree in informatics engineering from Department of Informatics Engineering, Atma Jaya University, Yogyakarta, Indonesia in 2008. Then pursued his passion in software development as software engineering until now. He currently leading a small team of engineers at Evermos. His interests include software engineering, project management, and computer science. He can be contacted at email: [antasenawahyu@gmail.com](mailto:antasenawahyu@gmail.com).