KIPS Tr. Software and Data Eng.
Vol.7, No.6 pp.205~220 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.6.205

A Comparative Study on Similarity Measure Techniques for Cross-Project Defect Prediction

Duksan Ryu† · Jongmoon Baik††

ABSTRACT

Software defect prediction is helpful for allocating valuable project resources effectively for software quality assurance activities thanks to focusing on the identified fault-prone modules. If historical data collected within a company is sufficient, a Within-Project Defect Prediction (WPDP) can be utilized for accurate fault-prone module prediction. In case a company does not maintain historical data, it may be helpful to build a classifier towards predicting comprehensible fault prediction based on Cross-Project Defect Prediction (CPDP). Since CPDP employs different project data collected from other organization to build a classifier, the main obstacle to build an accurate classifier is that distributions between source and target projects are not similar. To address the problem, because it is crucial to identify effective similarity measure techniques to obtain high performance for CPDP, In this paper, we aim to identify them. We compare various similarity measure techniques. The effectiveness of similarity weights calculated by those similarity measure techniques are evaluated. The results are verified using the statistical significance test and the effect size test. The results show k-Nearest Neighbor (k-NN), LOcal Correlation Integral (LOCI), and Range methods are the top three performers. The experimental results show that predictive performances using the three methods are comparable to those of WPDP.

Keywords : Cross-Project Defect Prediction, Similarity Measure, Outlier Detection

교차 프로젝트 결함 예측을 위한 유사도 측정 기법 비교 연구

류 덕 산† · 백 종 문††

요 약

소프트웨어 결함 예측은 결함이 자주 발생하는 모듈에 집중함으로써 소프트웨어 품질 보증 활동에 귀중한 프로젝트 리소스를 효과적으로 할당하는 데 도움이 될 수 있다. 회사 내에서 수집 된 충분한 기록 데이터를 사용하여 정확한 결함 발생 가능성이 높은 모듈 예측에 대해 WPDP (프로젝트 내 결함 예측)를 사용할 수 있다. 회사가 과거 데이터를 유지하지 못한 경우 CPDP (Cross-Project Defect Prediction) 메커니즘을 기반으로 오류를 예측하는 분류기를 만드는 것이 도움이 될 수 있다. CPDP는 다른 조직에서 수집 한 다른 프로젝트 데이터를 사용하여 분류기를 작성하기 때문에 정확한 분류기를 만드는데 가장 큰 장애물은 소스와 대상 프로젝트 간의 서로 다른 분포이다. 이 문제의 해결을 위해 효과적인 유사도 측정 기술을 식별하는 것이 중요하므로, 본 논문에서는 다양한 유사도 측정 기술을 CPDP 모델에 적용하여 성능을 비교한다. 유사도 가중치의 유효성을 평가하고, 통계적 유의성 검정 및 효과 크기 검정을 통해 결과를 검증한다. 실험 결과, k-Nearest Neighbor (k-NN), LOcal Correlation Integral (LOCI) 및 Range 방법이 유사도 측정 기술 중 상위 3 개에 속했고, 이들을 사용하는 CPDP 예측 성능이 WPDP의 성능과 유사하였다.

키워드 : 교차 프로젝트 결함 예측, 유사도 측정, 이상점 발견

1. Introduction

The failures of software-intensive systems have led to human and financial losses. It is crucial to identify and eliminate software defects in that they may cause system failures. The needs of software quality assurance activities, i.e., testing and inspection, are enormously increasing but resources for them are mostly limited. Thus, they should be

※ This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1A09917660, Artificial Intelligence-based Quantitative Quality Prediction and Evaluation Technique for Software Intensive System).

† 정 회 원 : KAIST, School of Computing, Research Professor
†† 비 회 원 : KAIST, School of Computing, Professor

Manuscript Received : October 17, 2017
First Revision : December 28, 2017
Accepted : February 5, 2018

* Corresponding Author : Duksan Ryu(dsryu@kaist.ac.kr)

allocated effectively onto fault-prone modules. Software Defect Prediction (SDP) is a typical way to identify fault-prone modules. SDP has been implemented through classification models built by various machine learning mechanisms [1-6]. When sufficient local data are available, classification models can predict defects with a high degree of accuracy. In this Within-Project Defect Prediction (WPDP) case, the same distribution between training and test data is the main reason of high accuracy. In case that a company does not maintain local bug repositories, Cross-Project Defect Prediction (CPDP) is the most useful. It builds a predictor by using different project data. Different distributions between a source project and a target project are the main factors lowering performance under cross-project (CP) environments. To deal with this problem, previous CPDP studies [7-10] aimed at identifying effective distributional dataset properties (e.g., mean, median, minimum, maximum, and range) for calculating similarity between a source project and a target project. In other words, it is crucial to identify effective similarity measure techniques to obtain high prediction performance under CP environments.

Software defect datasets typically have much fewer defective examples compared with non-defective examples. Due to this class imbalance problem, specific learners could not produce high predictive performance. It is also necessary to deal with the class imbalance under CP environments.

Although distributional characteristics were employed to measure the similarity in existing CPDP studies, the comprehensive comparison of similarity measure techniques has not been studied. This study investigates two research questions:

- RQ1: Which similarity measure technique is more effective for cross-project defect prediction?
- RQ2: Can our CPDP approach make the prediction performance similar to within-project defect prediction?

In this study, we intend to identify similarity measure techniques which are effective for enhancing the performance of CPDP. We compare a variety of similarity measure techniques based on angle, distance, density, cluster, projection and statistical methods. The effectiveness of similarity weights calculated by those similarity measure techniques are evaluated with the classification model named HISNN (Hybrid Instance Selection using Nearest Neighbor) [11]. The experimental results are verified with the statistical significance test and the effect size test.

The remaining sections of this study are organized as follows. In section 2, we cover related work. In section 3, similarity measure techniques we used in experiments are explained. The setup for experiments is described in section 4. In section 5, the experimental results are described. In section 6, threats to validity are explained. We summarize the study and discuss future work in the final section.

2. Related Work

2.1 Software Defect Prediction

Software defect prediction (SDP) studies how to identify fault-prone modules correctly over software bug database. Its main objective is to support decision-making for the effective allocation of limited resources, leading to software quality improvement and the cost reduction. Many researchers presented various Within-Project Defect Prediction (WPDP) methods [3, 4, 6] requiring sufficient historical data within an organization. A company has no local data when developing the first versions of the software system, i.e., pilot projects. In such cases, the company can apply Cross-Project Defect Prediction (CPDP) mechanism, utilizing defect data from other companies to construct a classifier.

Zimmermann et al. [7] presented that only 21 among 622 CPDP cases were successful. They asserted that the identification of process and data characteristics is vital for dealing with different distributions between source and target projects. They also suggested CPDP issues be investigated by more researchers.

Ma et al. [9] proposed a Transfer Naive Bayes (TNB) for CPDP. As a way of measuring the similarity between source and target projects, range [12] was used. The similarity weights calculated are used for building the proposed model.

Ryu et al.[13] suggested a boosting method for CPDP considering both different distributions and class imbalance. As similarity measure, the range was also used for computing the similarity weights.

Turhan et al. [14] presented the relevancy filtering method based on the nearest neighbor. As a similarity measure, k-Nearest Neighbor (k-NN) was used to choose training examples similar to the test project.

According to previous CPDP studies, the identification of distributional properties between source and target projects played an important role in the success of CPDP. Consequently, it is important to identify effective similarity measure techniques for CPDP.

2.2 Class Imbalance Learning

Mostly on software defect dataset, defective examples are much fewer than non-defective examples. This issue is called the class imbalance problem [15]. The class imbalance

issue results in lowering the predictive performance of the classifiers [4]. Since misclassification of the defective class (minority class) is directly associated with software quality, it is important for the learner to produce high overall performance and high performance for the defective class without worsening the performance of the non-defective class (majority class) in the context of class imbalance.

Grbac and Goran [16] investigated how the different levels of imbalance over software defect datasets affect the predictive stability of classification models. They presented that the high imbalance ratio could lead to the unstable predictive performance.

Chawla et al. [17] proposed Synthetic Minority Over-sampling TEchnique (SMOTE) as a way to deal with the class imbalance problem. Instead of duplicating the existing examples, SMOTE creates synthetic minority class examples. Such synthetic examples cause the classification model to generate less specific decision regions. The prediction of the minority class can be better generalized because the classifier learns more general regions for minority class examples. In our study, we apply SMOTE to achieve high classification performance considering the context of class imbalance.

3. Similarity Measure Techniques

In order to compute the similarity between a source project and a target project, previous CPDP studies [7-10] utilized distributional dataset properties, e.g., the mean, median, minimum, maximum, and range. Since those summary statistics only represent the overall dataset characteristics, we try to identify more sophisticated approaches reflecting the precise characteristics of a target project.

In this section, we describe various similarity measure techniques based on angle, distance, density, cluster, projection, and statistical methods that can be applied into CPDP. Such categories are based on the work of Aggarwal [18]. The similarity measure technique is closely related to the outlier detection in that an outlier can be described as a data point distant from other data. Since outliers may indicate data points belonging to a distribution different from the rest of the dataset, the outlier detection can be connected to the identification of dissimilar instances between the source and the target projects under CP settings. Based on the target project, the source project instances identified as outliers can be considered dissimilar instances to the target project. The source project instances identified as non-outliers can be considered similar instances with the target project.

3.1 Angle-Based Method

3.1.1 Angle-Based Outlier Factor (ABOF) [19] evaluates the variance over the angles between the difference vectors of a data point to all pairs of data points. Interior data points are probable to have data points around them at different angles, whereas data points at the boundaries are probable to enclose the whole data within a smaller angle. If many other data points are placed in similar directions, a data point is an outlier. If many other data points are placed in varying directions, a data point is not outlier.

3.2 Distance-Based Methods

3.2.1 Range [12], the difference between the smallest and the largest values, is one way to compute the similarity between source and target projects [9, 13].

Given a sequence $x_i = \{a_{i1}, a_{i2}, \cdots, a_{ik}\}$, a_{ij} is the jth attribute of x_i. The minimum and maximum values of jth attribute in test data are computed:

$$min_j = \min\{a_{1j}, a_{2j}, \cdots, a_{mj}\},\ max_j = \max\{a_{1j}, a_{2j}, \cdots, a_{mj}\}$$

where j = 1, 2, ⋯, k, k is the number of attributes and m is the number of test data. The following two vectors have the minimum and maximum values of the attribute on test data. Min = $\{min_1, min_2, \cdots, min_k\}$ and Max = $\{max_1, max_2, \cdots, max_k\}$. Next, the similarity weight of each training example can be computed as follows:

$$S_i = \sum_{j=1}^{k} h(a_{ij}) / k \quad (1)$$

where

$$h(a_{ij}) = \begin{cases} 1 \ if\ min_j \le a_{ij} \le max_j \\ 0 \ otherwise \end{cases},$$

a_{ij} is the jth attribute of instance x_i.

If $S_i = 1$, the ith instance is a similar instance. This indicates that all the attributes of the ith instance are positioned within the range.

3.2.2 k-Nearest Neighbor (k-NN) [20] selects the nearest neighbors based on the distance function as similar instances. Mostly, instance based learners use Euclidean distance defined as:

$$\text{Distance}(X,Y) = \sqrt{\sum_{i=1}^{k}(X_i - Y_i)^2} \quad (2)$$

where X and Y are the two instances, and X_i and Y_i (i = 1..k) are their attributes.

Besides Euclidean distance, Hamming distance [21] is a useful way to calculate the distance. Particularly, [11, 22]

employed the minimum hamming distance to identify the similar instances. The minimum hamming distance (Min-Ham) indicates the minimum distance in Hamming space where there is a neighbor.

Suppose Min-Ham$(\vec{x_i}) = \varepsilon$,

$$\text{HammingDistance}(\vec{x_i}, \vec{x_j}) \geq \varepsilon, \qquad \forall \vec{x_j} \in X \quad (3)$$

For each data instance of the target project, the nearest neighbors in the source project are selected as the similar instances.

3.3 Density-Based Methods

3.3.1 Parzen-Window Density Estimation [23] is a method commonly placing a Gaussian distribution around each of the training instances. Given a data point x, Parzen-window estimates the Probability Density Function (PDF) P(x) from which the data point was derived.

When the difference between a source data instance and the mean of the target project data is larger than three times the standard deviation in the target data distribution, the source data instance is a dissimilar instance.

3.3.2 Local Outlier Factor (LOF) [24] computes the local deviation of a data point according to its neighbors for the detection of outliers. The local density of a data point is compared to the local densities of its neighbors. As such regions of similar density can be identified. If points having a considerably lower density than their neighbors, they are outliers.

The LOF values of the source and the target project data are computed. If source project instances have a significantly lower density than their neighbors composed of the target project data, they are dissimilar instances.

3.3.3 LOcal Correlation Integral (LOCI) [25] is an algorithm to find outliers based on Multi-granularity DEviation Factor (MDEF) handling local density variations in the feature space.

The densities of each source project instance and its neighborhood composed of the target project data are used to compute the MDEF. If source project instances have a density similar to their neighbors composed of the target project data, they are similar instances.

3.4 Cluster-Based Method

3.4.1 k-means clustering [26] targets at partitioning n instances into k clusters where each instance falls into the cluster with the nearest mean. Based on the target project data, clusters are formed. If the source project data are positioned inside of the clusters, they are identified as similar instances.

3.5 Projection-Based Methods

3.5.1 Principal Component Analysis (PCA) [27] is an algorithm using an orthogonal linear transformation converting the data to principal components in a new coordinate system. The k-dimensional hyper-plane ($k<d$) minimizing the squared projection error is determined in PCA. A PCA on a target project is estimated and a source project is mapped onto the PCA subspace. The distance between the target project and the mapped source project is used to identify similar instances.

3.5.2 Self-Organizing Map (SOM) [28], a.k.a. Kohonen map is a method reducing the dimensions of data with self-organizing neural networks. Unit, a special type of data point is dynamic unlike the regular data points. Best Matching Unit (BMU) is defined as the unit closest to the input vector. SOM is trained with the target project data. Dissimilar instances in source project data lie relatively far away from the bulk of the target project data.

3.6 Statistical Models

3.6.1 Mixture Of Gaussians (MOG) model [29] is a kind of a mixture model that assumes subpopulations are present within an overall population. MOG parameters are estimated from the target project data using the Expectation-Maximization (EM) [29] algorithm. Data can be clustered with MOG. Dissimilar instances in source project data lie relatively far away from the clusters of the target project data.

4. Methodology

4.1 Hypotheses for Research Questions

1. RQ1: Which similarity measure technique is more effective for cross-project defect prediction?

We intend to check how differently similarity measure techniques affect the classification performance of CPDP. To this end, we compare various types of similarity measure techniques based on angle, distance, density, cluster, projection and statistical methods. As previous researches [7-10] pointed out, the major challenge of CPDP is poor classification performance owing to different distributions between source and target projects. To tackle this issue, previous studies used distributional characteristics as

similarity measure techniques. It is important to identify effective similarity measure techniques in the context of CPDP. To answer RQ1, the following hypotheses are formalized:

- $H1_0$: The performance of different similarity measure techniques is the same.
- $H1_A$: The performance of different similarity measure techniques is not the same.

2. RQ2: Can our CPDP approach make the prediction performance similar to within-project defect prediction?

It is commonly known that CPDP is inferior to WPDP using the same distributional data in terms of the classification performance [14]. In RQ2, we compare the HISNN approach using the three most effective similarity measure techniques with WPDP. Whereas the previous research [30] suggest a method using a mixed project data, combining CP and WP data, our proposed method uses only CP data. Since the collection of WP data requires additional effort and time, the benefits of our model is enormous if it is comparable to WPDP. Not only can it hasten the time of introducing SDP but also reduce the cost of introducing SDP. To answer RQ2, the following hypotheses are formalized:

- $H2_0$: The proposed CPDP approach performs similarly to within-project defect prediction.
- $H2_A$: The proposed CPDP approach performs differently from within-project defect prediction.

4.2 Benchmark Dataset

We employ the datasets from Jureczko and Spinellis [31] for experiments. They can be accessed in PROMISE repository [32]. They were extracted from open-source software (OSS) projects, proprietary projects and academic projects. In our experiments, 17 academic projects are merged into the project named student since their size is small. This merged version is also used in a previous CPDP study [30]. Totally 16 datasets are used in the experiments. Table 1 shows that the datasets encompass various software projects in terms of size and buggy ratio. The number of instances ranges from 26 to 904. The percentage of buggy modules ranges from 3.8 to 76.9. In Table 2, 17 academic projects that are merged into student project are described. In this study, the first versions of projects are only used since CPDP would be the most useful when there is no prior version available.

All the projects contain 20 features, i.e., static code and object-oriented metrics as shown in Table 3. The bug count information in the dataset is used as a class label. If an instance has any bug, it is marked as buggy module.

Table 1. Datasets Used in the Experiments

No	Project	# Instances	# Buggy	% Buggy	Description
1	ant	125	20	16	OSS
2	camel	339	13	3.8	OSS
3	ivy	111	63	56.7	OSS
4	jedit	272	90	33.1	OSS
5	log4j	135	34	25.2	OSS
6	lucene	195	91	46.7	OSS
7	pbeans	26	20	76.9	OSS
8	poi	237	141	59.5	OSS
9	prop-6	660	66	10	Proprietary
10	student	904	217	24	Academic
11	synapse	157	16	10.2	OSS
12	systemdata	65	9	13.8	OSS
13	tomcat	858	77	9	OSS
14	velocity	196	147	75	OSS
15	xalan	723	110	15.2	OSS
16	xerces	162	77	47.5	OSS

Table 2. Student Dataset

Projects
arc, berek, ckjm, e-learning, intercafe, kalkulator, nieruchomosci, pdftranslator, redaktor, serapion, skarbonka, sklebagd, szybkafucha, termoproject, workflow, wspomaganiepi, zuzel

Table 3. Features for the Experiments

Features
weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object classes (CBO), response for a class (RFC), lack of cohesion in methods (LCOM), lack of cohesion in methods (LCOM3), number of public methods (NPM), data access metric (DAM), measure of aggregation (MOA), measure of functional abstraction (MFA), cohesion among methods of class (CAM), inheritance coupling (IC), coupling between methods (CBM), average method complexity (AMC), afferent couplings (Ca), efferent couplings (Ce), maximum McCabe's cyclomatic complexity (Max(CC)), average McCabe's cyclomatic complexity (Avg(CC)), lines of code (LOC)

4.3 Comparative Framework

In this subsection, we describe the framework for comparing various similarity measure techniques. The comparative framework has two phases, i.e., preprocessing and Hybrid Instance Selection using Nearest-Neighbor (HISNN) [11]. Different similarity measure techniques are applied within the HISNN model.

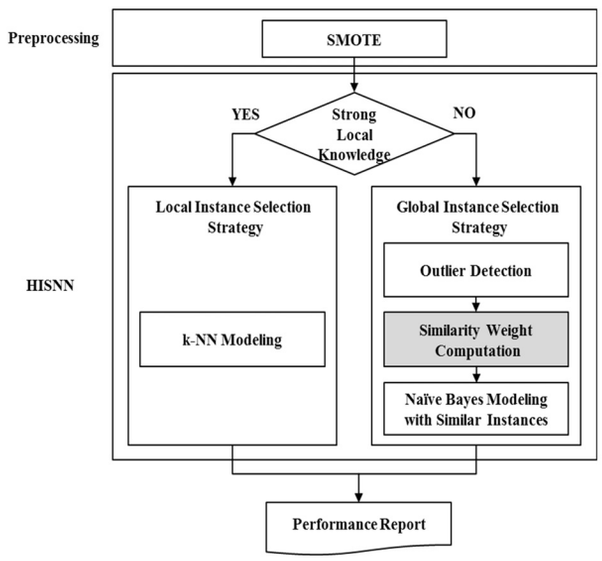

Fig. 1. The Comparative Framework

4.3.1 Preprocessing

In this phase, over-sampling with SMOTE [17] is performed. Software defect datasets mostly have much fewer defective instances compared with non-defective instances. Specific learners could not produce high predictive performance due to this class imbalance problem. SMOTE can be effective for adding up the number of the minority class instances in the training set. Thus, classification rules can be defined well and the predictive performance can be enhanced.

4.3.2 HISNN

Hybrid Instance Selection using Nearest-Neighbor (HISNN) [11] is a selective learning method based on the concept of LASER [22]. HISNN is a framework adapted for CPDP. It checks if each target project instance has the strong local knowledge on the basis of the source project. In

```
1  DATA = {ant, camel, ivy, jedit, log4j, lucene, pbeans, poi, prop-6, student, synapse, systemdata, tomcat, velocity,
   xalan, xerces}
2  Sim_Measure = {abof, kmeans, knn(E), knn(H), loci, lof, mog, parzen, pca, range, som}
3  Local_Learner = {k-NN}
4  Global_Learner = {Naive Bayes}
5  for sim_measure ∈ Sim_Measure
6    for data ∈ DATA do
7      CPTrain = DATA - data
8      CPTrain′ = Over-sample CPTrain′ using SMOTE
9      CPTest = data
10     for i = 1 → 100 do
11       {Shuffle CPTest in each iteration}
12       CPTrain_wo_Outlier_1 = Remove outliers in CPTrain′ based on CPTrain′
13       CPTrain_wo_Outlier_2 = Remove outliers in CPTrain′ based on CPTest
14       CPTrain_Similar = Select similar instances in CPTrain′ using sim_measure
15       CPTrain_Final = CPTrain_wo_Outlier_1 ∩ CPTrain_wo_Outlier_2 ∪ CPTrain_Similar
16       CPTrain_Final′ = Select unique instances of CPTrain_Final
17       CPTrain_Final′′ = Apply log filter to CPTrain_Final′
18       CPTest′ = Apply log filter to CPTest
19       for each instance of CPTest
20         if the strong local knowledge of CPTrain′ exists
21           Apply Local_Learner to the instance of CPTest using CPTrain′
22         else
23           Apply Global_Learner trained with CPTrain_Final′′ to the instance of CPTest′
24         end if
25       end for
26       Report CP_PD(i), CP_PF(i), CP_Bal(i)
27     end for
28   end for
29 end for
```

Fig. 2. The Process of the Experiments

this case, the nearest neighbors and the nearest cluster [33] are considered together.

If the nearest neighbors or the nearest cluster of an instance have the same class label, it is considered as having the strong local knowledge. If the strong local knowledge is identified, the local instance selection strategy using a local learner (i.e., k-NN) is performed. In case of the weak local knowledge, the global instance selection strategy is used. HISNN aims to build a global learner (i.e., Naive Bayes) using source data without irrelevant instances. Naive Bayes is selected since it has shown high performance in predicting defects compared with other classifiers [4, 34]. Naive Bayes is built using the WEKA machine learning toolkit [35].

At this time, HISNN employs two methods, i.e., outlier detection and similarity weight computation to select the source project instances similar to the target project in accordance with CPDP. In the step of the outlier detection, Mahalanobis distance [36] is used to identify abnormal instances among the source project data. The Mahalanobis distance between a data point and a distribution, measures how away the point is from the mean. A data instance is considered as an outlier if it is more than 3 standard deviations away from the mean of the entire data distribution.

Particularly, HISNN detects outliers among source data instances from both the source project distribution and the target project distribution. The outliers based on the source project distribution can be counted as the extraneous instance in the source project. The outliers based on the target project distribution can be considered as extraneous instances dropping down the predictive performance of the target project. In the step of the similarity weight computation, source project instances similar to the target project are detected. To form final training data, the source instances with no outliers and source instances similar to test data are combined. These combined training data are utilized to train a Naive Bayes model to predict target instances that require the global knowledge.

In this study, in the step of the similarity weight computation, different similarity measure techniques are applied. Similarity measure techniques explained in section 3 are used for experiments. knn(E) indicates k-Nearest Neighbor using Euclidean distance. As Turhan et al. [14] did, k is set as 10. knn(H) indicates k-Nearest Neighbor using the minimum Hamming distance. All the experimental settings except for the similarity weight computation are the same. For all the similarity measure techniques, instances selected in the local instance selection strategy are the same.

Fig. 2 shows the process of the comparison experiments. To set up the CP settings, we chose each dataset to be a test set and used the remaining datasets as a training set, The number of the minority class instances are increased by the SMOTE method. (lines 1-9)

Each experiment was iterated 100 times to reduce the random bias. By using Mahalanobis distance, outliers among source data instances are filtered out based on both the source project distribution and the target project distribution. The similar source instances are chosen with the similarity weight computed by the similarity measure technique. The final training data are formed by combining the source instances with no outliers and source project instances similar to the test project. Since such instances may not be unique, only unique instances are chosen for the next step. As previous studies [14, 37] recommended, log-filter (i.e., replacing N with ln(N)) is applied to training and test sets. It is employed to fulfill the normality assumption of Naive Bayes model. (lines 10-18)

Next, HISNN conducts the hybrid instance selection based on the local knowledge. Each target project instance is examined if there exists the strong local knowledge based on the source project data. The class label of the target project instance is found out by using the k-NN model. If there exists no strong local knowledge, the Naive Bayes model trained with the source project instances similar to the target project is applied to test the target project instance. The source data instances similar to the target data are only used to build Naive Bayes model. Log filter is only used when Naive Bayes model is applied. Details of HISNN can be found in [11]. (lines 19-25)

The performance results of CPDP model are reported by using Probability of Detection (PD), Probability of False alarm (PF), and Balance (Bal). (lines 26)

In RQ1, we aim to find more effective similarity measure techniques for CPDP. To this end, we compare the predictive results of HISNN using 11 similarity measure methods over 16 datasets under cross-project settings.

In RQ2, we check if HISNN using the three most effective similarity measure techniques found in RQ1 can provide comparable performance in comparison with WPDP. We compare our approach with Naive Bayes under WP settings. As a baseline, we also include the relevancy filter proposed by Turhan et al. [14] and Naive Bayes under CP environments. The relevancy filter is chosen since it is widely used [14, 30, 38]. For WPDP, we split training and test data randomly by

a 50:50 ratio. This method is generally used in the literature [10, 39]. We applied the log filter to all the cases.

4.4 Performance Evaluation

Software defect datasets typically have the class imbalance issue. To measure the classifier built on the imbalanced data set, multiple performance measures are recommended [40]. The performance on the buggy class is generally evaluated with Probability of Detection (PD) and Probability of False alarm (PF). Balance can evaluate the overall performance indicating how well a classifier can balance predictive performance between buggy and clean classes.

Table 4. Confusion Matrix

		Predicted class	
		Buggy	Clean
Actual class	Buggy	TP (True Positive)	FN (False Negative)
	Clean	FP (False Positive)	TN (True Negative)

As shown in Table 4, we can count the number of correct/incorrect classification with confusion matrix. True Positive (TP) is the number of buggy instances predicted correctly as buggy. True Negative (TN) is the number of clean instances predicted correctly as clean. False Positive (FP) is the number of clean instances predicted incorrectly as buggy. False Negative (FN) is the number of buggy instances predicted incorrectly as clean. Using such counts, we can derive the performance measures as follows:

- $\text{PD} = \frac{TP}{(TP + FN)}$
- $\text{PF} = \frac{FP}{(FP + TN)}$
- $\text{Balance} = 1 - \frac{\sqrt{(0-PF)^2+(1-PD)^2}}{\sqrt{2}}$

PD, a.k.a. recall, means the ratio of correct instances retrieved. PF, a.k.a. the false positive rate, means the ratio of clean instances misclassified within the clean class. Unlike PD, PF is better when its value is lower. Balance is defined as a Euclidean distance between the ideal (1, 0) point and the real (PD, PF) point. A classifier producing high Balance values is desired in that it can take a good balance of the performance between the buggy and the clean classes.

On our experiments, we have multiple similarity measure techniques over multiple data sets. For this case, the Friedman test [41, 42] with the post-hoc tests are recommended by Demsar [43]. If the null hypothesis meaning that the performance of the comparisons is similar is rejected by the Friedman test, this indicates that at least the performance between two learners are significantly different from each other. Then, we can proceed to find the groups of data that

Table 5. The Median PD Performance of HISNN using 11 Similarity Measure Techniques

No	Target	*abof*	*kmeans*	*knn(E)*	*knn(H)*	*loci*	*lof*	*mog*	*parzen*	*pca*	*range*	*som*
1	*ant*	**0.85**	**0.85**	0.8	0.8	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**	**0.85**
2	*camel*	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**	**0.615**
3	*ivy*	**0.571**	0.555	0.492	0.412	0.507	0.507	0.555	**0.571**	0.539	0.428	**0.571**
4	*jedit*	**0.788**	**0.788**	0.777	0.766	0.777	**0.788**	**0.788**	**0.788**	**0.788**	0.777	**0.788**
5	*log4j*	**0.764**	**0.764**	0.735	0.647	0.647	0.735	**0.764**	**0.764**	**0.764**	**0.764**	**0.764**
6	*lucene*	**0.593**	**0.593**	**0.593**	0.582	0.56	**0.593**	**0.593**	**0.593**	**0.593**	0.582	**0.593**
7	*pbeans*	0.5	0.5	**0.55**	0.1	0.3	0.5	0.5	0.5	0.4	0.45	0.45
8	*poi*	0.666	0.673	0.687	**0.702**	0.645	0.659	0.68	0.68	0.666	0.673	0.673
9	*prop-6*	**0.742**	**0.742**	0.712	**0.742**	0.636	0.651	**0.742**	**0.742**	**0.742**	0.712	**0.742**
10	*student*	0.626	0.658	**0.672**	0.654	0.635	0.663	0.654	0.663	0.645	0.654	0.658
11	*synapse*	**0.75**	**0.75**	0.687	**0.75**	0.687	0.687	**0.75**	**0.75**	**0.75**	0.687	**0.75**
12	*systemdata*	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**	**0.666**
13	*tomcat*	0.662	0.701	0.662	0.48	0.493	0.48	0.701	**0.714**	0.688	0.142	0.701
14	*velocity*	0.38	0.387	0.367	**0.401**	0.34	0.353	0.38	0.38	0.353	0.36	0.387
15	*xalan*	0.818	0.818	0.809	**0.827**	0.809	0.809	0.818	**0.827**	0.809	0.809	0.818
16	*xerces*	0.337	0.337	**0.35**	0.324	0.324	0.324	0.337	**0.35**	0.337	0.324	0.337
	Median	0.664	0.6695	0.669	0.6505	0.6355	0.655	**0.673**	**0.673**	0.666	0.66	0.6695

differ via a post-hoc test. As the post-hoc test, various test methods including Tukey's Honestly Significant Difference test [44], Nemenyi test [45], and Bonferroni-Dunn test [46] can be employed. We carried out Tukey's HSD test for our experiments.

We carry out Wilcoxon rank-sum test [47] at a confidence level of 95% to evaluate the performance between the two distributions. This non-parametric test is recommended because it does not make any assumption on the data distribution unlike a Student's t-test [48].

We perform the effect size test called A-statistics [49] to evaluate the magnitude of the improvement. Arcuri and Briand [48] indicated this test is appropriate for evaluating randomized algorithms in software engineering. A-statistics means the probability of algorithm X providing higher M values in comparison with another algorithm Y, where M is a performance measure. For instance, A = 0.64 indicates that X provides higher results 64% of the time. Based on the guidelines [49], X is better(or worse) than Y if A > 0.64, meaning a medium size difference. If A <= 0.64, X is not better (or worse) than Y.

4.5 Implementation of Similarity Measure Technique

As the similarity measure techniques in the experiments, we use outlier detection algorithms implemented in Data Description Toolbox (DDtools) [50]. DDtools is the one-class classification method dealing with a two-class classification problem, having the target and the outlier class. The target class is assumed to be sampled well. The outlier class is sampled very sparsely. For executing outlier detection algorithms, we use the default option FRACREJ = 0.1. FRACREJ indicates the fraction of targets rejected, i.e. the fraction of errors on the target class.

5. Result

We discuss the experimental results based on RQ1 and RQ2. For RQ1, we aim at identifying more effective similarity measure techniques under CP settings. For RQ2, we check if our CPDP model employing the selected similarity measure techniques is useful by comparing its performance with that of the WPDP model.

5.1 Experiment 1: RQ1

We show the performance results of HISNN using 11 similarity measure techniques over 16 datasets. Table 5 shows the median PD performance. knn(E) indicates k-nearest neighbor method using the Euclidean distance. knn(H) indicates k-nearest neighbor method using the minimum Hamming distance. In case of the median PD values, mog and parzen show the best performance (0.673) compared to other methods. loci is the worst performer showing 0.6355. The other methods mostly show about 0.67.

In Table 6, the median PF values are shown. In this case,

Table 6. The Median PF Performance of HISNN using 11 Similarity Measure Techniques

No	Target	*abof*	*kmeans*	*knn(E)*	*knn(H)*	*loci*	*lof*	*mog*	*parzen*	*pca*	*range*	*som*
1	*ant*	0.571	0.542	0.428	**0.419**	0.447	0.542	0.552	0.552	0.542	0.542	0.552
2	*camel*	0.45	0.46	0.487	0.478	**0.42**	0.429	0.466	0.466	0.457	0.447	0.469
3	*ivy*	0.333	0.333	0.208	0.187	0.27	0.312	0.354	0.333	0.333	**0.166**	0.333
4	*jedit*	0.505	0.505	0.483	0.489	**0.478**	0.505	0.505	0.505	0.505	**0.478**	0.505
5	*log4j*	0.346	0.366	0.396	0.415	**0.247**	0.336	0.376	0.376	0.346	0.316	0.356
6	*lucene*	0.461	0.461	0.442	0.509	**0.375**	0.451	0.48	0.48	0.461	0.403	0.461
7	*pbeans*	**0**	**0**	0.166	0.166	**0**	**0**	0.166	**0**	**0**	**0**	**0**
8	*poi*	0.416	0.427	0.427	0.427	**0.406**	0.416	0.427	0.427	0.416	0.416	0.427
9	*prop-6*	0.622	0.622	0.574	0.607	**0.469**	0.486	0.622	0.651	0.614	0.515	0.622
10	*student*	0.43	0.458	0.468	0.508	**0.425**	0.448	0.446	0.461	0.435	0.432	0.458
11	*synapse*	0.624	0.631	0.574	**0.56**	0.581	0.609	0.638	0.638	0.631	0.617	0.631
12	*systemdata*	0.446	0.446	0.41	0.446	**0.392**	0.446	0.517	0.517	0.446	**0.392**	0.446
13	*tomcat*	0.349	0.37	0.357	0.302	**0.18**	0.249	0.366	0.375	0.368	0.24	0.37
14	*velocity*	**0.428**	0.448	**0.428**	0.51	**0.428**	**0.428**	0.448	0.448	**0.428**	**0.428**	**0.428**
15	*xalan*	0.512	0.535	0.5	0.53	**0.481**	0.495	0.512	0.535	0.508	0.5	0.515
16	*xerces*	0.482	0.482	0.482	**0.47**	0.494	0.494	0.482	0.494	0.482	0.494	0.482
	Median	0.448	0.459	0.435	0.474	**0.4225**	0.447	0.473	0.473	0.4515	0.43	0.4595

Table 7. The Median Balance Performance of HISNN using 11 Similarity Measure Techniques

No	Target	*abof*	*kmeans*	*knn(E)*	*knn(H)*	*loci*	*lof*	*mog*	*parzen*	*pca*	*range*	*som*
1	*ant*	0.582	0.601	0.665	**0.671**	0.666	0.601	0.595	0.595	0.601	0.601	0.595
2	*camel*	0.58	0.575	0.56	0.565	**0.597**	0.592	0.572	0.572	0.577	0.582	0.57
3	*ivy*	**0.616**	0.607	0.611	0.564	0.602	0.587	0.598	**0.616**	0.598	0.579	**0.616**
4	*jedit*	0.612	0.612	0.623	0.616	**0.627**	0.612	0.612	0.612	0.612	**0.627**	0.612
5	*log4j*	0.703	0.692	0.663	0.614	0.695	0.697	0.686	0.686	0.703	**0.72**	0.697
6	*lucene*	0.565	0.565	0.575	0.534	**0.591**	0.57	0.554	0.554	0.565	0.589	0.565
7	*pbeans*	0.646	0.646	**0.66**	0.352	0.505	0.646	0.627	0.646	0.575	0.611	0.611
8	*poi*	0.622	0.619	0.625	**0.631**	0.618	0.619	0.623	0.623	0.622	0.625	0.619
9	*prop-6*	0.523	0.523	0.545	0.533	0.579	0.576	0.523	0.504	0.528	**0.582**	0.523
10	*student*	0.596	0.595	0.595	0.565	0.604	0.603	0.6	0.596	0.602	**0.608**	0.595
11	*synapse*	0.524	0.519	0.537	**0.566**	0.533	0.515	0.515	0.515	0.519	0.51	0.519
12	*systemdata*	0.606	0.606	0.625	0.606	**0.635**	0.606	0.564	0.564	0.606	**0.635**	0.606
13	*tomcat*	0.656	0.663	0.652	0.575	0.619	0.592	0.665	**0.666**	0.658	0.37	0.663
14	*velocity*	0.467	0.463	0.459	0.443	0.443	0.451	0.459	0.459	0.451	0.455	**0.471**
15	*xalan*	0.615	0.6	0.621	0.605	**0.633**	0.624	0.615	0.602	0.615	0.621	0.613
16	*xerces*	0.42	0.42	**0.428**	0.417	0.408	0.408	0.42	0.423	0.42	0.408	0.42
	Median	0.601	0.6005	**0.616**	0.5655	0.603	0.5965	0.5965	0.5955	0.5995	0.595	0.6005

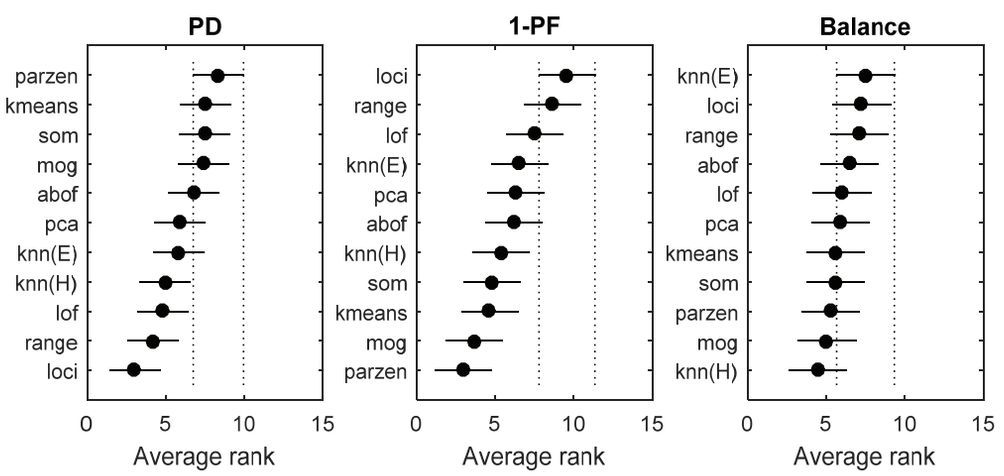

Fig. 3. Tukey's Critical-Difference Diagram for PD, 1-PF and Balance

loci shows the best PF performance (0.4225) by comparison with other methods. Most of the other methods show about 0.45.

In Table 7, the median Balance values are shown. With respect to Balance, knn(E) produces the best result (0.616) in comparison with other methods. The other methods mostly produce about 0.6.

For PD and 1-PF, Friedman test indicated that the performance difference among 11 similarity measure techniques are statistically significant, showing p-value < 0.05. Thus, $H1_0$ is rejected. However, in terms of Balance, the difference in performance scores among 11 similarity measure methods were not statistically significant, showing p-value > 0.05.

Fig. 3 shows the Tukey's HSD test results for HISNN using 11 similarity measure techniques when PD, 1-PF, and Balance are used as the performance measure. In Fig. 3, each group mean is represented by a symbol and 95% confidence interval is shown as a line around the symbol. If the intervals of two groups are disjoint, they are different significantly. If their intervals overlap, they are not different significantly.

Fig. 3 shows that the performance of PD, 1-PF and

Table 8. The Median PD, PF and Balance Performance of Classifiers

No	Target Data	CP Naive Bayes			CP Naive Bayes + NN			WP Naive Bayes			HISNN(knn(E))			HISNN(loci)			HISNN(range)		
		PD	*PF*	*B*	*PD*	*PF*	*B*	*PD*	*PF*	*B*	*PD*	*PF*	*B*	*PD*	*PF*	*B*	*PD*	*PF*	*B*
1	*ant*	**1**	0.666	0.528	0.9	0.495	0.642	0.85	0.471	0.639	0.8	0.428	0.665	0.85	0.447	**0.666**	0.85	0.542	0.601
2	*camel*	**0.692**	0.585	0.532	**0.692**	0.653	0.489	0.428	**0.352**	0.563	0.615	0.487	0.56	0.615	0.42	**0.597**	0.615	0.447	0.582
3	*ivy*	**0.746**	0.5	0.603	0.571	0.25	**0.649**	0.718	0.416	0.618	0.492	0.208	0.611	0.507	0.27	0.602	0.428	0.166	0.579
4	*jedit*	**0.933**	0.648	0.539	**0.933**	0.532	0.62	0.888	**0.428**	**0.68**	0.777	0.483	0.623	0.777	0.478	0.627	0.777	0.478	0.627
5	*log4j*	0.882	0.405	0.701	**0.941**	0.722	0.487	0.882	0.539	0.593	0.735	0.396	0.663	0.647	0.247	0.695	0.764	0.316	**0.72**
6	*lucene*	0.714	0.528	0.574	0.67	0.394	**0.636**	**0.76**	0.442	0.633	0.593	0.442	0.575	0.56	0.375	0.591	0.582	0.403	0.589
7	*pbeans*	0.55	0.166	**0.66**	0.5	0.333	0.575	**0.8**	0.666	0.507	0.55	0.166	**0.66**	0.3	0	0.505	0.45	0	0.611
8	*poi*	0.843	0.593	0.565	0.865	0.656	0.526	**0.873**	0.687	0.511	0.687	0.427	**0.625**	0.645	0.406	0.618	0.673	0.416	**0.625**
9	*prop-6*	**0.893**	0.67	0.52	0.712	0.518	0.58	0.59	**0.316**	**0.617**	0.712	0.574	0.545	0.636	0.469	0.579	0.712	0.515	0.582
10	*student*	0.617	**0.359**	**0.628**	0.732	0.48	0.611	**0.88**	0.655	0.529	0.672	0.468	0.595	0.635	0.425	0.604	0.654	0.432	0.608
11	*synapse*	**0.937**	0.73	0.481	**0.937**	0.702	0.501	0.75	**0.394**	**0.643**	0.687	0.574	0.537	0.687	0.581	0.533	0.687	0.617	0.51
12	*systemdata*	**0.777**	0.553	0.578	**0.777**	0.428	**0.658**	0.4	**0.25**	0.54	0.666	0.41	0.625	0.666	0.392	0.635	0.666	0.392	0.635
13	*tomcat*	0.792	0.389	0.687	0.246	0.186	0.451	**0.846**	0.37	**0.712**	0.662	0.357	0.652	0.493	0.18	0.619	0.142	0.24	0.37
14	*velocity*	0.578	0.612	0.474	**0.7**	0.653	0.492	0.689	0.44	**0.596**	0.367	0.428	0.459	0.34	0.428	0.443	0.36	0.428	0.455
15	*xalan*	0.9	0.61	0.562	**0.909**	0.654	0.532	0.854	**0.462**	**0.657**	0.809	0.5	0.621	0.809	0.481	0.633	0.809	0.5	0.621
16	*xerces*	0.415	0.588	0.413	0.441	0.635	0.401	**0.615**	**0.232**	**0.669**	0.35	0.482	0.428	0.324	0.494	0.408	0.324	0.494	0.408
	Median	**0.784**	0.586	0.5635	0.722	0.525	0.5535	0.780	0.434	**0.6175**	0.669	0.435	0.616	0.6355	**0.4225**	0.603	0.66	0.43	0.595

Balance is better when its average rank is higher. The performance results are sorted by the descending order of the average rank.

In terms of PD, parzen shows the best performance. Tukey's HSD tests showed that parzen scored statistically significantly higher than knn(H), lof, range and loci. With regard to 1-PF, loci shows the best performance. Tukey's HSD tests showed that loci scored statistically significantly better than knn(H), som, kmeans, mog and parzen. In regard to Balance, knn(E) performs the highest. Tukey's HSD tests showed that there is no similarity measure technique statistically better than the others regarding Balance.

Based on Balance, the three highest methods are knn(E), loci and range, indicating that they achieve the more balanced prediction performance between PD and PF. Thus, they are selected for further investigation. In the next subsection, knn(E), loci and range are compared with WPDP for RQ2.

5.2 Experiment 2: RQ2

In this subsection, we compared HISNN using knn(E), loci and range with Naive Bayes using a Nearest Neighbor filter (CP Naive Bayes + NN) and Naive Bayes under CP settings (CP Naive Bayes) as well as Naive Bayes under WP settings (WP Naive Bayes). Although the main purpose of RQ2 is to compare HISNN with WPDP, we additionally added CP Naive Bayes and CP Naive Bayes + NN as baselines for comparison.

In Table 8, the median PD, PF and Balance scores of classification models are presented. The best result of each case is marked in boldface. In Fig. 4, median PD and PF results of six classifiers over 16 datasets is illustrated in a scatter plot. Since the ideal point is at PD=1 and PF=0, a better classifier produces more data points at the bottom right of the figure.

CP Naive Bayes showed the worst PF value (0.586) whereas it showed the best PD value (0.784).Thus, more points are located at the top right of the areas. In case of high PF rates, it takes much time and effort for practitioners to explore false alarms. In cases of ant, jedit, prop-6, synapse, velocity, and xalan, PF values show more than 0.6. Such high PF rates are not acceptable to most software practitioners.

Turhan et al. [14] proposed the relevancy filter to mitigate high PF rates. It lowers from 0.586 to 0.525 based on the median PF values. Nevertheless, there are still cases showing high PF rates (camel, log4j, poi, synapse, velocity, xalan, and xerces). In Fig. 4, it also produces more data points at the top right of the figure.

In WPDP, since both training and test sets have the same data distribution, its prediction performance is expected to be the best. WP Naive Bayes produces high PD (0.780) and low PF (0.434). Regarding Balance, it also provides higher result (0.6175) compared to CP Naive Bayes (0.5635) and the

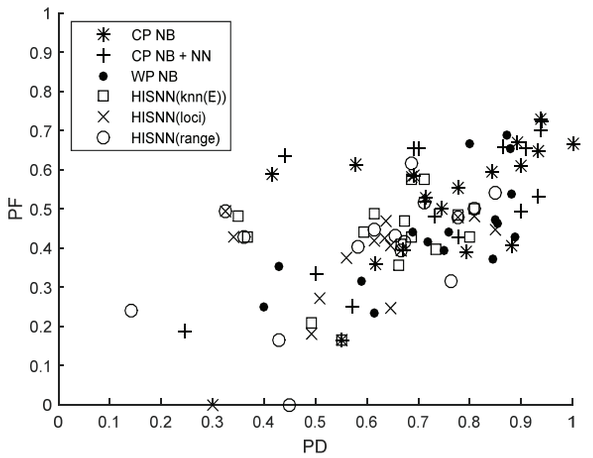

Fig. 4. Scatter Plot of Median PD and PF Scores of four Classifiers over 16 Datasets

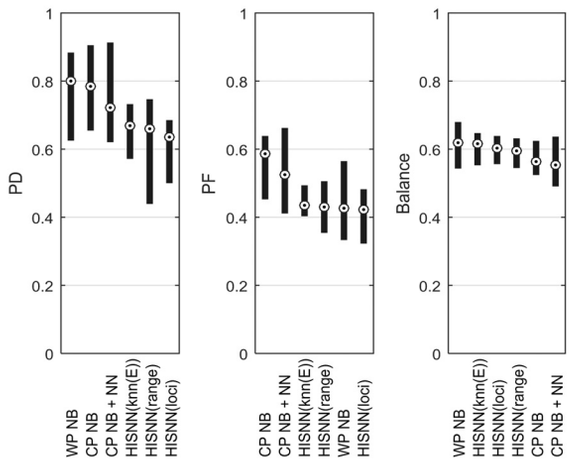

Fig. 5. Mini Boxplots of Median PD, PF and Balance Scores of Six Classifiers over 16 Datasets

relevancy filter (0.5535). However, there are cases producing high PF values in cases of pbeans (0.666), poi (0.687), and student (0.655).

HISNN using knn(E), loci and range produces the low PF value (0.435, 0.4225, 0.43) in comparison with the other classifiers. Compared to CP Naive Bayes and the relevance filter, they show better Balance and PF performance.

In Experiment 1, Tukey's HSD test was performed. However, it does not consider the actual performance, but only the relative ranking [6]. In this subsection, a method to find a statistically significant ranking of the approaches is described.

Based on [5, 6], the variability of the models across multiple runs can be evaluated. The mini box plot shows the first, second and third quartile of each experimental case. The models are sorted by their median values. A circle means the median and a bar represents the first-third quartile range. The smallest and the largest values are not shown. The 100 points for each target project are merged as each performance measure because we iterate each experiment 100 times. For 16 projects, 1600 points are utilized to compute the first, second and third quartile.

In Fig. 5, median PD, PF and Balance values of six classifiers sorted by median are illustrated by mini boxplots. HISNN using knn(E), loci and range reduces PF rates effectively whereas it shows the lowest PD rates. They show higher Balance values than CP NB and the relevancy filter but lower than WP NB.

In Table 9, HISNN using knn(E) is compared to other models according to the Wilcoxon rank-sum test at a 5% significance level and the A-statistics effect size test. We mark the significantly better performance of HISNN in boldface. The significant difference indicates p-value < 0.05 and A-statistics > 0.64 for PD and Balance (A-statistics < 0.36 for PF). The boldface indicates the significantly better performance of HISNN with p-value < 0.05 and A-statistics > 0.64 for PD and Balance (A-statistics < 0.36 for PF).

A practically useful classifier should produce high PD and low PF rates individually as well as high overall performance. In Table 9, the Wilcoxon rank-sum test showed that the difference in Balance scores between HISNN (knn(E)) and other classifiers under CP settings are statistically significant with p-value $<< 0.001$. Regarding Balance, the effect size values against CP NB and the relevancy filter are 0.61 (i.e., the small effect) and 0.65 (i.e., the medium effect) respectively. We check if HISNN is similar to WPDP to answer RQ2. In Table 9, the Wilcoxon rank-sum test showed that the difference in Balance rates between HISNN

Table 9. The Comparison of HISNN(knn(E)) with other Classifiers

HISNN (knn(E)) vs.		CP NB	CP NB + NN	WP NB
PD	p-value	<< 0.001	<< 0.001	<< 0.001
	A-statistic	0.25	0.32	0.28
PF	p-value	**<< 0.001**	**<< 0.001**	0.67
	A-statistic	**0.23**	**0.30**	0.50
Balance	p-value	**<< 0.001**	**<< 0.001**	<< 0.001
	A-statistic	0.61	**0.65**	0.43

Table 10. The Comparison of HISNN(loci) with other Classifiers

HISNN (loci) vs.		CP NB	CP NB + NN	WP NB
PD	p-value	<< 0.001	<< 0.001	<< 0.001
	A-statistic	0.22	0.27	0.25
PF	p-value	<< **0.001**	<< **0.001**	<< **0.001**
	A-statistic	**0.19**	**0.24**	0.42
Balance	p-value	<< **0.001**	<< **0.001**	<< 0.001
	A-statistic	0.61	0.60	0.41

Table 11. The Comparison of HISNN(range) with other Classifiers

HISNN (range) vs.		CP NB	CP NB + NN	WP NB
PD	p-value	<< 0.001	<< 0.001	<< 0.001
	A-statistic	0.24	0.30	0.27
PF	p-value	<< **0.001**	<< **0.001**	<< 0.001
	A-statistic	**0.23**	**0.27**	0.46
Balance	p-value	<< **0.001**	<< **0.001**	<< 0.001
	A-statistic	0.57	0.55	0.38

and Naive Bayes under WP settings are statistically significant with p-value << 0.001. Although HISNN is worse than WPDP in terms of the Wilcoxon rank-sum test, with respect to the effect size test for Balance, the effect size was 0.43 meaning a small effect. Thus, we reject $H2_0$. The comparison results using HISNN(loci) and HISNN(range) are shown in Table 10 and 11. The results of the statistical test and the effect size test are similar to that of HISNN (knn(E)). The experimental results represent that HISNN can be practically used under CP settings.

6. Threats to Validity

6.1 Threats to Construct Validity

In HISNN, target instances to use local knowledge are separated from target instances to use global knowledge. Then, the similarity measure techniques only apply to instances to use global knowledge. Without such separation, the similarity measure techniques using all the instances may perform better / worse for CPDP.

We used DD tools with the default option, i.e., FRACREJ = 0.1. Because we obtained the results using the single choice of the value, different conclusions may be reached when using other values.

6.2 Threats to External Validity

For our experiments, public datasets from the PROMISE repository are employed. Because other software projects have different distributional properties, our findings may not be generalizable to them. Nevertheless, totally 16 datasets encompass various software projects in terms of the type, size and defect ratio. By doing so, we mitigate threats to external validity as possible.

6.3 Threats to Conclusion Validity

For RQ1, Friedman test and Tukey's HSD test were performed. As Demsar [43] indicated, the Friedman statistic is considered as meaningful when the number of learners is bigger than 5 and the number of data sets is bigger than 10. This guideline is satisfied in that we have 11 learners built with different similarity measure techniques over 16 datasets.

For RQ2, we performed Wilcoxon rank-sum test and the A-statistics effect size test. The statistical significance difference between two classifiers was checked by Wilcoxon rank-sum test. We evaluated the magnitude of the improvement using A-statistics effect size test. Two methods have been accepted generally in software engineering [48].

7. Conclusion

Software Defect Prediction (SDP) has been widely studied as a mechanism to identify defective modules. When historical defect data are sufficiently available within a company, classifiers can predict defects with a high degree of accuracy. Within-Project Defect Prediction (WPDP) can produce high performance owing to the same distributional properties between training and test data. Cross-Project

Defect Prediction (CPDP) is the most helpful in case that historical bug data are unavailable. Since a classifier is built with different project data in CPDP, the main reason lowering the predictive performance under cross-project (CP) environments is different distributions between a source project and a target project. To tackle the problem, it is crucial to employ effective similarity measure techniques identifying source project instances similar to the target project.

In this study, we aim to identify similarity measure techniques which are effective for enhancing the prediction performance of CPDP. We compare 11 similarity measure techniques based on angle, distance, density, cluster, projection and statistical methods over 16 datasets. The effectiveness of similarity weights calculated by those similarity measure techniques are evaluated with the classifier named HISNN (Hybrid Instance Selection using Nearest Neighbor). Then we investigated if HISNN is similar to WPDP when it is used with the appropriate similarity measure technique. The experimental results are verified by using the statistical significance test and the effect size test. We identified the effective similarity measure techniques under CP settings. In addition, without historical data, we showed that CPDP can be comparable to WPDP if appropriate similarity measure is employed.

We showed various similarity measure techniques associated with the HISNN model that can be employed for CPDP. The PF rates were effectively decreased but the PD rates became lower as well. In future studies, we will investigate how to remedy this shortcoming. In particular, the relationships between each similarity measure technique and three performance measures (i.e., PD, PF and Balance) need to be investigated in depth.

References

[1] S. Kim, E. Whitehead, and Y. Zhang, "Classifying software changes: Clean or buggy?," *Softw. Eng. IEEE Trans.*, Vol. 34, No.2, pp.181-196, 2008.

[2] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, Vol.81, No.5, pp.649-660, May 2008.

[3] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, Vol.83, No.1, pp.2-17, Jan. 2010.

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng.*, Vol.38, No.6, pp.1276-1304, Nov. 2012.

[5] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Autom. Softw. Eng.*, Vol.17, No.4, pp.375-407, May 2010.

[6] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empir. Softw. Eng.*, Vol.17, No.4-5, pp.531-577, Aug. 2012.

[7] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pp.91-100, 2009.

[8] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Autom. Softw. Eng.*, Vol.19, No.2, pp.167-199, Jul. 2011.

[9] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technol.*, Vol.54, No.3, pp.248-256, Mar. 2012.

[10] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 35th International Conference on Software Engineering*, pp.382-391, 2013.

[11] D. Ryu, J. Jang, and J. Baik, "A Hybrid Instance Selection using Nearest-Neighbor for Cross-Project Defect Prediction," *J. Comput. Sci. Technol.*, Vol.30, No.5, pp.969-980, 2015.

[12] G. Woodbury, "An Introduction to Statistics." Cengage Learning, 2001.

[13] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empir. Softw. Eng.*, Vol.21, No.1, pp.43-71, Feb. 2016.

[14] 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," *Empir. Softw. Eng.*, Vol.14, No.5, pp.540-578, Jan. 2009.

[15] T. Pang-Ning, M. Steinbach, and V. Kumar, "Introduction to Data Mining." 2006.

[16] T. Grbac, G. Mausa, and B. Basic, "Stability of Software Defect Prediction in Relation to Levels of Data Imbalance.," in *Proceedings of the 2nd Workshop of SQAMIA*, 2013.

[17] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE : Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, Vol.16, pp.321-357, 2002.

[18] C. C. Aggarwal, "Outlier Analysis." New York, NY: Springer New York, 2013.

[19] H.-P. Kriegel, M. Schubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data," *Proceeding 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* (KDD '08), pp.444-452, 2008.

[20] N. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *Am. Stat.*, Vol.46, No.3, pp.175-185, 1992.
[21] R. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.*, Vol.XXIX, No.2, 1950.
[22] B. Raman and T. R. Ioerger, "Enhancing Learning using Feature and Example selection," Texas A&M Univ. Coll. Station. TX, USA, 2003.
[23] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Stat.*, Vol.33, No.3, pp.1065-1076, 1962.
[24] M. Breunig, H. Kriegel, R. Ng, and J. Sander, "LOF: identifying density-based local outliers," *ACM Sigmod Rec.*, pp.1-12, 2000.
[25] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "LOCI: Fast outlier detection using the local correlation integral," *Proc. - Int. Conf. Data Eng.*, pp.315-326, 2003.
[26] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, Vol.28, No.2, pp.129-137, 1982.
[27] I. T. Jolliffe, "Principal Component Analysis." Springer, 2002.
[28] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, Vol.43, No.1, pp.59-69, 1982.
[29] C. M. Bishop, "Pattern recognition and machine learning." New York, New York, USA: Springer, 2006.
[30] B. Turhan, A. Tosun Mısırlı, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Inf. Softw. Technol.*, Vol.55, No.6, pp.1101-1118, Jun. 2013.
[31] M. Jureczko and D. Spinellis, "Using Object-Oriented Design Metrics to Predict Software Defects," in Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010, pp.69-81.
[32] T. Menzies et al., "The PROMISE Repository of empirical software engineering data," 2012. [Online]. Available: http://openscience.us/repo/.
[33] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, When is "nearest neighbor" meaningful? Springer-Verlag, 1999.
[34] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Trans. Softw. Eng.*, Vol.34, No.4, pp.485-496, 2008.
[35] M. Hall, E. Frank, and G. Holmes, "The WEKA data mining software: an update," *ACM SIGKDD Explor. Newsl.*, Vol.11, No.1, pp.10-18, 2009.
[36] P. C. Mahalanobis, "On the generalised distance in statistics," in *Proceedings of the National Institute of Sciences of India*, Vol.2, No.1, pp.49-55, 1936.
[37] F. Menzies T, Greenwald J, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, Vol.33, No.1, pp.2-13, 2007.
[38] B. Turhan, A. Tosun, and A. Bener, "Empirical Evaluation of Mixed-Project Defect Prediction Models," in *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp.396-403, 2011.
[39] Y. Kamei, S. Matsumoto, A. Monden, K. I. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," *IEEE Int. Conf. Softw. Maintenance*, ICSM, 2010.
[40] S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," *IEEE Trans. Reliab.*, Vol.62, No.2, pp.434-443, Jun. 2013.
[41] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Am. Stat. Assoc.*, No.32, pp.675-701, 1937.
[42] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings.," *Ann. Math. Stat.*, No.11, pp.86-92, 1940.
[43] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, Vol.7, pp.1-30, 2006.
[44] J. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, No.5, pp.99-114, 1949.
[45] P. Nemenyi, "Distribution-free multiple comparisons.," Princeton University, 1963.
[46] O. J. Dunn, "Multiple comparisons among means," *J. Am. Stat. Assoc.*, No.56, pp.52-64, 1961.
[47] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bull., pp.80-83, 1945.
[48] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *2011 33rd International Conference on Software Engineering* (ICSE), pp.1-10, 2011.
[49] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *J. Educ. Behav. Stat.*, Vol.25, No.2, pp.101-132, 2000.
[50] D. M. J. Tax, "DDtools, the Data Description Toolbox for Matlab." 2014.

Duksan Ryu

http://orcid.org/0000-0002-9556-0873
e-mail : dsryu@kaist.ac.kr

He received his Ph.D. degree in School of Computing from KAIST in 2016. He earned a bachelor's degree in computer science from Hanyang University and a Master's dual degree in software engineering from KAIST and Carnegie Mellon University. His research areas are software defect prediction and software reliability engineering.

Jongmoon Baik

http://orcid.org/0000-0002-2546-7665

e-mail : jbaik@kaist.ac.kr

He received his M.S. degree and Ph.D. degree in computer science from University of Southern California in 1996 and 2000 respectively. He received his B.S. degree in computer science and statistics from Chosun University in 1993. He worked as a principal research scientist at Software and Systems Engineering Research Laboratory, Motorola Labs, where he was responsible for leading many software quality improvement initiatives. Currently, he is an associate professor in School of Computing at Korea Advanced Institute of Science and Technology (KAIST). His research activity and interest are focused on software six sigma, software reliability & safety, and software process improvement.