Muhammed Maruf Öztürk

COMPLEXFUZZY: NOVEL CLUSTERING METHOD FOR SELECTING TRAINING INSTANCES OF CROSS-PROJECT DEFECT PREDICTION

Abstract *Over the last decade, researchers have investigated to what extent cross-project defect prediction (CPDP) shows advantages over traditional defect prediction settings. These works do not take the training and testing data of defect prediction from the same project; instead, dissimilar projects are employed. Selecting the proper training data plays an important role in terms of the success of CPDP. In this study, a novel clustering method called complexFuzzy is presented for selecting the training data of CPDP. The method reveals the most defective instances that the experimental predictors exploit in order to complete the training. To that end, a fuzzy-based membership is constructed on the data sets. Hence, overfitting (which is a crucial problem in CPDP training) is alleviated. The performance of complexFuzzy is compared to its 5 counterparts on 29 data sets by utilizing 4 classifiers. According to the obtained results, complexFuzzy is superior to other clustering methods in CPDP performance.*

Keywords cross-project defect prediction, complexFuzzy, training instance selection, fuzzy clustering

Citation Computer Science 22(1) 2021: 3–37

1. Introduction

It is widely known that software maintenance accounts for up to 50% of total development costs [7, 21, 56]. To reduce these costs, good planning is a possible solution, thereby predicting defects. Defect prediction, which is an interesting research area of software engineering, aims to estimate future defects by using the historical data of software projects [28, 52, 71]. However, one critical point during defect prediction is the structure of the data sets. Previous works had performed prediction through different versions of the same projects because there were not enough defect prediction data sets [45]. In such experiments (namely, within-project defect prediction [WPDP]), data sets having the same metrics and a different number of instances are employed. As the number of data sets increases with a specific diversity level, the bias extracted from the defect prediction is desired to be transferred from local to global; however, this is difficult in WPDP due to the lack of historical data.

Researchers prefer to take training and testing data from different projects rather than working on the same domain to reach a certain bias. This method is called CPDP; its main objective is to observe the success of a prediction model in which the training and testing data is taken from different projects [38, 65, 66, 74]. The key issues related to CPDP are comprised of 1) a great number of features are difficult to select, and 2) dividing CPDP data sets into testing and training groups is often effort-intensive. A trivial mistake in the training process may dramatically affect the success of CPDP. Various studies have been developed to cope with these problems.

Despite the fact that WPDP tends to produce promising results in terms of F-measure and precision [19], performing CPDP is a must in some cases in which enough within-project data is not available. On the other hand, CPDP requires data filtering or feature selection methods to improve the prediction models. Data quality is a critical issue that should be removed in order to obtain reliable models [20, 30].

In [16], CPDP was investigated in terms of training data selection without considering data-filtering methods. On the other hand, the data used in CPDP become reliable through regional data analyses. Ma et al. [35] simply focused on improving the learning methods to be utilized in CPDP. Their method prefers a weighted model to transfer data from the target to the source; it does not regard data distribution while performing CPDP. Rahman et al.'s work [45] argued that CPDP has equal or better prediction results when compared to WPDP. However, their experimental design was established by disregarding the defectiveness metrics of the training data. The method presented in this paper contributes to CPDP by proposing a new clustering method for selecting training data. By this way, a classifier is able to increase the success of the prediction of defective instances.

The method presented in this paper aims to develop a new training instance selection method for CPDP. In this context, a clustering method (namely, complexFuzzy) is proposed to reveal the most defective cluster of the instances.

The paper's detailed objectives are as follows:

- to improve detection of true positive rate in CPDP;
- to bring new clustering method that can be used in preprocessing;
- to discuss the effects of training instance selection in CPDP;
- to reveal which type of clustering is viable in training instance selection of CPDP.

The contribution of the paper can be summarized as follows: 1) a novel fuzzy-clustering method is developed for clustering defect prediction data sets that it is superior to similar methods in terms of cluster centers; 2) complexFuzzy is a clustering method that has the potential to reap the benefits of some areas such as metric selection in CPDP and heterogeneity; 3) a metric formula has been developed so that it may be adapted to software cost estimation, test case prioritization, and fault localization; and 4) the effectiveness of the method has been confirmed by employing $10 \cdot 10$ cross-validation. In particular, complexFuzzy demonstrated high performance with regard to performance parameters such as the area under the curve (AUC) and F-measure.

The algorithm complexFuzzy shows clear advantages over traditional methods, which are as follows: 1) it has a distinctive property regarding the selection of the training data in which the software metrics are utilized; 2) unlike preceding works, this study presents a new method that may help practitioners solve the heterogeneity problem in CPDP; and 3) complexFuzzy includes a sophisticated mathematical model that helps detect the most defective instances. Defective instances are easily learned in the training thanks to the model. This functionality is the main advantage that the model provides.

The remainder of the paper is organized as follows. The background of CPDP and related works are described in Section 2. The method and experimental design are detailed in Section 3. The obtained results are presented in Section 4. The threats of the validity are discussed in Section 5. Lastly, Section 6 summarizes the results and mentions future work.

1.1. Motivation

Clustering defect prediction data sets requires an indicator that shows the accuracy level of a cluster label. This need emerges when noisy data sets are employed in a prediction experiment. To date, a clustering method using defectiveness metrics to select training data has yet to be developed. With respect to CPDP, the studies focusing on metric matching represent the majority of the related literature. However, as the selection of training data dramatically affects the success of the prediction, there is a need for developing a method that explores suitable ways of selecting the training data. Such a study could pave the way for improved CPDP methods.

Detecting defective instances via clustering could further improve the software development process to some extent. For instance, the most defective software module can be recoded to prevent possible defects. Furthermore, an organizational deficiency can be removed with the help of clustering.

In traditional clustering methods, distance measurement techniques are generally preferred when assigning an instance to a cluster. They do not consider an instance by examining defectiveness metrics to generate clusters. To address this problem, complexFuzzy generates defective clusters from an instance pool by considering the defectiveness of related instances. This determines the membership level of an instance. In this respect, complexFuzzy has a distinctive property in creating clusters. This is the first direction of the contribution. On the other hand, matching or filtering the software metrics to perform CPDP creates a critical reliability thread. In doing so, valuable information may unintentionally be ignored due to the metric selection. Instead, an experimental evaluation encompassing all of the metrics of the data sets can yield reliable results. The second direction of the contribution is that complexFuzzy is an adaptable clustering method; thus, it may be used in other software engineering problems such as test case prioritization and fault localization through modifying the membership determination step of complexFuzzy.

1.2. Research questions

In this section, research questions (RQ) depicting the contributions of the paper are ordered as follows:
RQ1. To what extent is complexFuzzy able to detect cluster centers?
RQ2. Is complexFuzzy superior to similar methods in CPDP?
RQ3. Is instance-based CPDP better than feature-based alternatives?
RQ4. What is the performance of clustering methods in training data selection depending on the scale of the data sets?

RQ1 is especially prominent for complexFuzzy to be an alternative in the subfields of software engineering. The answer to RQ1 reveals the competitiveness of the method. If a clustering method is used for selecting training data instances, RQ2 helps us find the originality of the method. In addition to this, RQ2 determines whether complexFuzzy is viable for CPDP. RQ3 could pave the way for future works that examine execution time, memory consumption, and the working method with various data sets of complexFuzzy. RQ4 investigates which sizes of data sets are viable to use with complexFuzzy in CPDP.

2. Background and related works

2.1. Preliminaries

If $x_1, ..., x_n$ denotes the instances of a software project, n is the number of instances, and $y_1, ..., y_n$ denotes the defectiveness labels. These labels include $1/0$ values of $true/false$ strings. Moreover, they could be $0/1...t$ that is within a range of $t = 0, ..., \infty$. In such cases, binary classification is conducted by converting t to $1/0$.

Let p denote the number of modules in a software system, with each module having different instance sizes such as $m_1, ..., m_p$. Thus, if c_m represents the number of instances of the related software module, $c_{m1}, ..., c_{mp}$ shows the list

of the number of instances. Inherently, $m_1, ..., m_p$ may be taken from the same software project. In doing so, m is generally the same for each software metric. The prediction is completed by using similar metrics. The defect prediction data set is divided into parts according to the experimental design. For instance, if z index is used in the division, the training and testing data sets are denoted by $m_1, ..., m_z$ and $m_z, ..., m_p$, respectively. Thus, the training and testing are performed on the same software project. Prediction that is performed on the same software project is called within-project defect prediction (WPDP).

Conversely, the case in CPDP is as follows: let $m_1, ..., m_n$ be the software in which the training data set is taken. $s_1, ..., s_t$ represents the software modules that include the testing data set. In conclusion, the training and testing data sets are either taken from a different project or different versions of a project. This is the reason why such predictions are called CPDP.

The issues of CPDP vary depending on the number of instances and metric types. If two projects are selected that include different numbers of software metrics, the training and testing phases of the prediction cannot be performed in the usual way. This problem is handled mostly by making a feature selection on the project with further metrics. Another problem (called heterogeneity) originates from the metric type [37]. If there are two metric groups, the proper metrics are selected via metric matching. Metric matching is a deep research topic that requires a great number of statistical analyses.

If the within- and cross-project defect prediction parameters are denoted by g_w, f_w, auc_w and g_c, f_c, auc_c, $g_c \leq g_w, f_c \leq f_w, auc_c \leq auc_w$ is desired to make CPDP worth performing. Here, g, f, and auc refer to the G-mean, F-measure, and auc area under the curve, respectively. The processes (which include determining the training and testing data) of the within- and cross-project defect prediction are illustrated in Figure 1.

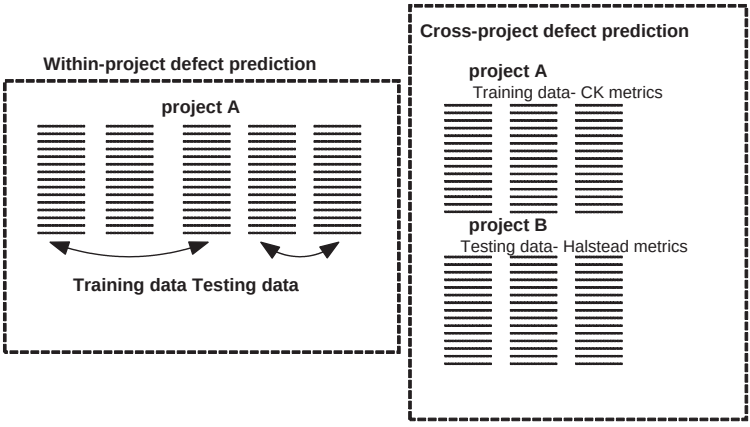

Figure 1. Data-selection process difference between within- and cross-project defect prediction

The training and testing data is taken from Project A in the within-project defect prediction. Conversely, the training and testing data is taken from different projects in CPDP, and the metrics might not match (as can be found in the CK-Halstead metrics).

2.2. Cross-project defect prediction

Zimmermann et al.'s work [74] was the first to express that working on the same domain or process does not improve prediction performance. Furthermore, their study paved the way for determining how the combinations of training and testing data should be devised. In [70], a connectivity-based algorithm was developed for classifying defect prediction data sets. The obtained results showed that unsupervised classifiers outperformed supervised ones in terms of AUC. In addition to this, a spectral clustering (SC) algorithm outperformed the alternatives. However, SC is not for generating suitable training data sets; instead, it is used for labeling an entire data set. Ryu et al. [49] proposed a cost-sensitive boosting approach for CPDP. In their method, the training data has weights and a level of class imbalance in which these parameters are used for enhancing prediction accuracy. The method yielded promising results in general performance parameters. A credibility-based classifier was proposed by Poon et al. [43]. It weights training and testing data by utilizing the standard deviation of the instances. The method is an improved version of Naive Bayes that showed better performance than Naive Bayes in terms of G-mean. However, the method did not consider data distribution during the experiment.

In [31], a new oversampling technique (namely, CDE-SMOTE) was proposed. It alleviates the class imbalance problem by estimating the class distribution of defect data sets. However, CDE-SMOTE was only tested on process metrics with a limited number of performance measures. In addition to this, during the validation of CDE-SMOTE, the heterogeneity of the software metrics was not considered. Herbold et al. [18] compared 24 different CPDP approaches. They concluded that, if a large-scale project is used, the performance difference is not remarkable in various CPDP methods. The learning type used in the training phase substantially affects CPDP. For instance, in [59], a semi-supervised technique was proposed for cross-project prediction settings. This outperformed four competing methods in four performance parameters. Instead, CPDP needs to be investigated in terms of heterogeneous metrics. In this respect, metric matching techniques are recently focused on this domain [24]. Zhou et al. [73] investigated CPDP in an unusual way. They proposed a module size model rather than benefiting from training data to perform CPDP. In doing so, practitioners could save much time completing the prediction process. In [38], a new cluster-based feature selection method was proposed for CPDP. This was compared with four alternatives in terms of precision, recall, F1-score, and AUC. Despite the fact that the method yielded promising results, it does not bring any novelty with respect to the selection of training data instances. Porto et al. proposed a meta-learning method to increase CPDP performance [44]. They pointed out that a CPDP method should be selected regarding the properties of the project being predicted.

2.3. Instance selection-based studies

In [17], the importance of training data selection in cross-project defect prediction was investigated. According to the obtained results, utilizing a data-selection method to determine the training data is crucial to improving the success rate of cross-project defect prediction. Cross-project defect prediction was considered to be a multi-objective optimization problem [53], but the experiment was not based on selecting training instances that have higher complexities. A two-phase CPDP approach was presented by Xia et al. [60]. The method (called Hydra) consists of a genetic algorithm and ensemble-learning phases. This significantly improved the prediction performance of 29 data sets. Selecting the proper training data is prominent for cross-project defect prediction. In [69], a novel training data-selection approach (namely, MT) was proposed. Normality, parameters are used in MT that improve three prediction parameters. He et al. [15] conducted an experimental study on 15 data sets. They stated that distance-based training data selection is better than baseline methods. However, their study does not discuss the effects of clustering-based prediction when fuzzy-like methods are used. Kamei et al. [25] investigated just-in-time (JIT) defect prediction on within- and cross-project experiments. One of their findings is that JIT does not give any tips for cross-project prediction performance if it is established on a within-project configuration. This study focused on selecting the training data by using the similarity between two projects rather than dividing a project into small parts to determine the suitable training data. In [20], it is asserted that instance selection for training data can be strengthened via feature selection. This study concluded that cross-project defect prediction approaches should be enriched by developing new techniques such as clustering that can be used for determining training or testing data sets.

There are various clustering methods that can be used while working with big data groups. Some of these are Fuzzy c-mean [47], k-means [14], self-organizing map (SOM) [57], model-based clustering (MBC) [9], hierarchical clustering (HC) [54], WaveCluster [51], and OptiGrid [22]. Fuzzy c-mean and k-means are based on Euclidean distance. New methods have been revealed as a way of measuring the distance between instances. Basic approaches have led to the development of improved versions of measurement methods [58, 61]. The underlying mechanism of complexFuzzy is similar to that of Fuzzy c-mean. This is the reason why it was utilized for devising complexFuzzy. Fuzzy c-mean gives membership values to the instances while dividing them into the clusters. The algorithm complexFuzzy has been revealed by making modifications and improvements on Fuzzy c-mean to determine a new defectiveness level on defect prediction data sets (except for defect labeling). The third step of Fuzzy c-mean computes the membership matrix via only Euclidean distance. On the other hand, complexFuzzy regards a complexity coefficient while computing the membership matrix values via Euclidean distance (as detailed in subsequent sections).

In SOM, the instances are generally represented with a two-dimensional map. Such a map consists of clustered hexagons. Naive SOM is the most common version of SOM; therefore, the experiment includes the first version of SOM rather than an

improved version of it. The main difference between k-means and SOM is that the number of clusters is determined by distance matrix techniques instead of a random technique in SOM [13].

The clustering results of HC are represented with tree-based structures called dendrograms [50]. In the experiment, an agglomerative type of HC is employed with Ward's minimum variance method, which aims to merge pairs of clusters that have minimum distances.

In MBC, the data is assumed to come from two or more clusters rather than one cluster [10]. This case creates a distribution model in the sense that a data point has a probability of belonging to its cluster.

There are some reasons why Fuzzy c-mean and k-means are involved in the experiment. First, k-means has proven its validity in clustering. It has a great number of improved versions that are used in different research fields [26, 27, 34, 67]. For instance, k-means++ is a sophisticated type of k-means. Over the last decade, an unprecedented effort has been directed towards validating the efficiency of k-means++ [6, 11, 62]. On the other hand, fuzzy c-mean is quite popular among researchers who work in the area of engineering. It is easily applied to numeric data sets thanks to its scalability and feasibility [33]. Second, new methods that are developed for clustering are needed to be compared with the pioneer ones; so, initiatives have been selected for the comparison. Last, fuzzy c-mean and k-means are frequently employed in software engineering problems [2, 64, 68].

Defect prediction data sets are quite rich in terms of metric diversity [8]. This case helped to increase the number of feature-focused works in CPDP [66]. However, there are some problems that originate from noise rate in software data sets and incorrect defect labeling. These problems could lead to wrongly interpreted data or calculations. Therefore, the imperfections and deficiencies in data instances should be eliminated. Furthermore, some software metrics determine or affect defect labeling; thus, selecting training instances using merely defect labels is inadequate. From the point of view of data instances, selecting training instances in CPDP is investigated in this work. Table 1 summarizes some studies that bring new approaches to CPDP. In this table, the number of all data sets for a study and the number of common data sets to our study are denoted by DS and CommonDS, respectively. It is worth noting that recent studies have a higher number of common data sets than relatively older studies.

Table 1

Summary of some works that handle CPDP (C→conference; J→journal)

Name	Publication type	Ref	Description	DS	CommonDS
A two-phase transfer learning model for cross-project defect prediction	J	(Liu et al., 2018 [32])	Proposes a transfer learning model for CPDP	42	22
Data Transformation in Cross-project Defect Prediction	J	(Zhang et al., 2017 [69])	Investigates transformation effects on CPDP	18	9
Global vs. local models for cross-project defect prediction	J	(Herbold et al., 2017 [18])	Compare global and local models on CPDP	79	24
A transfer cost-sensitive boosting approach for cross-project defect prediction	J	(Ryu et al., 2017 [49])	Investigates CPDP in terms of transfer learning	15	8

Table 1 (cont.)

Name	Publication type	Ref	Description	DS	CommonDS
An investigation on the feasibility of cross-project defect prediction	J	(He et al., 2012 [16])	Focuses on selecting training data	17	12
Multi-objective cross-project defect prediction	C	(Canfora et al., 2013 [5])	Proposes a multi-objective method for CPDP	10	7
Cross-project Defect Prediction Using a Connectivity-based Unsupervised Classifier	C	(Zhang et al., 2016 [70])	Compares classifiers on CPDP	26	7
HYDRA: Massively compositional model for cross-project defect prediction	J	(Xia et al., 2016 [60])	Proposed a hybrid model for CPDP	31	8
LACE2: Better privacy-preserving data sharing for cross project defect prediction	C	(Peters et al., 2015 [41])	Investigates data sharing in CPDP	17	6
Which is More Important for Cross-Project Defect Prediction: Instance or Feature?	C	(Yu et al., 2016 [65])	Investigates instance filtering on CPDP	6	2
FeSCH: A Feature Selection Method using Clusters of Hybrid-data for Cross-Project Defect Prediction	C	(Ni et al., 2017 [38])	Proposes a feature selection method for CPDP	5	2
A feature matching and transfer approach for cross-company defect prediction	J	(Yu et al., 2017 [66])	Presents a feature matching technique for CPDP	16	5
Better Cross Company Defect Prediction	C	(Peters et al., 2013 [42])	Presents a data filtering for CPDP	56	14
On the relative value of cross-company and within-company data for defect prediction	J	(Turhan et al., 2009 [55])	Investigates sample number for CPDP	8	0
Transfer learning for cross-company software defect prediction	J	(Ma et al., 2012 [35])	Focuses on data features of CPDP	10	0
Recalling the imprecision of cross-project defect prediction	C	(Rahman et al., 2012 [45])	Investigates quality tradeoffs of CPDP	9	3
Cross-project defect prediction using a credibility theory based naive bayes classifier	C	(Poon et al., 2017 [43])	Proposes a classifier for CPDP	11	3
Evaluating Data Filter on Cross-Project Defect Prediction: Comparison and Improvements	J	(Li et al., 2017 [29])	Presents a comparison of CPDP models	44	23
Combined classifier for cross-project defect prediction: an extended empirical study	J	(Zhang et al., 2018 [71])	Investigates composite algorithms for CPDP	10	8
A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction	J	(Ni et al., 2017 [39])	Proposes a cluster-based method for CPDP	8	2
HDA: Cross-Project Defect Prediction via Heterogeneous Domain Adaptation With Dictionary Learning	J	(Zhang et al., 2018 [63])	Proposes a heterogeneous method for CPDP	12	4
Dissimilarity Space Based Multi-Source Cross-Project Defect Prediction	J	(Ren et al., 2019 [46])	Develops a density-based method for CPDP	17	0

2.4. Feature selection-based studies

Nam et al. [37] developed a new heterogeneous defect prediction (HDP) approach that involves metric selection and metric matching. Their method increased the AUC scores of the data sets dramatically. A similar study proposing a feature matching algorithm was also done by Yu et al. [66]. They used feature distribution curves to get feature distances. Their method achieved great success regarding cross-company defect prediction with 16 data sets. Ryu and Baik [48] developed a multi-objective technique for cross-project defect prediction. Their technique was established based on Harmony Search, which is a heuristic optimization method that is widely-known among practitioners. Although the results of the study are in favor of diversity metrics, they do not include any tips for how the training data should be selected in a cross-project prediction experiment. Fukushima et al. [12] focused on just-in-time prediction for cross-project data sets. They used a similarity metric that matches suit-

able training and testing data. Ensemble methods were also proposed in their work to yield more-accurate cross-project models. Defect prediction data sets are generally collected by a great number of researchers; thus, they employ different metrics during this process. In cross-project defect prediction, this case creates a heterogeneity issue; to overcome this problem, metric matching is one of the preferred ways.

3. Method

The main steps of the algorithm are seen in Figure 2. In the first step, the data set group is taken. The proposed method is then compared to Fuzzy c-mean, K-means, SOM, MBC, and HC. The remaining steps are repeated for each algorithm. Thus, while one data group denotes defective instances that are used for training, other clusters represent testing instances. A performance comparison is the last step, in which the results are recorded by performing the training and testing process in different projects. Code smell metrics (including WMC, cohesion, and coupling) are involved in determining the membership level of complexFuzzy. The reason is that these metrics give tips to figure out defectiveness. More specifically, they constitute the underlying formula of complexFuzzy.

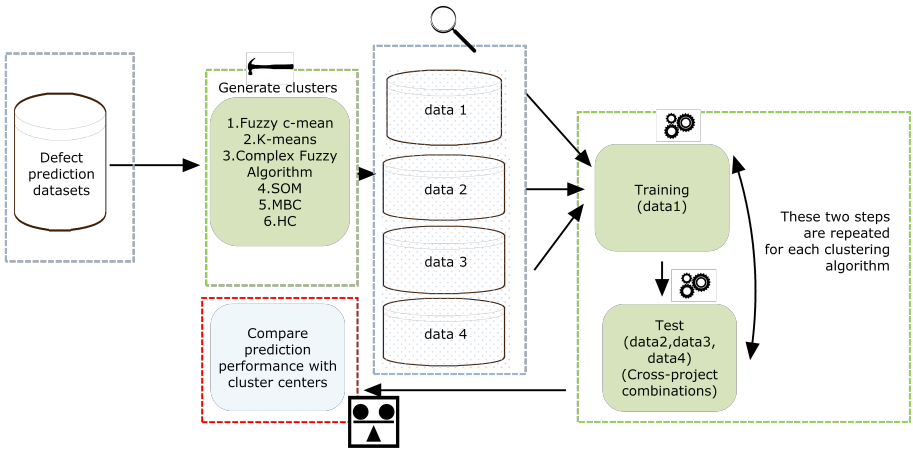

Figure 2. Main steps of proposed method

Let $S = (s_1, s_2, s_3, s_4, s_5, ..., s_n)$ denote defect prediction instances in which the number of instances is n. If the metric values of the instances are denoted with $MS = (ms_1, ms_2, ms_3, ms_4, ms_5, ..., ms_t)$, then t is the number of metrics. The sum of metric values $\sum_{i=1}^{t} ms_{1i}$ is used for normalization analysis to decide whether the data set is suitable for parametric tests. $D = (d_1, d_2, d_3, d_4, d_5, ..., d_n)$ represent the distances of a set of instances. A distance d_w selected from D is the distance for an s_w from its cluster center. Let k denote the number of clusters; then, the instances are divided into clusters $C = (c_1, c_2, c_3, ..., c_k)$ via fuzzy clustering.

The objective function of complexFuzzy is J where the membership interval is [0,1]. To calculate J, Equation 1 is employed. Here, m denotes the fuzziness index of the value if it is greater than 1. $||s_i - c_l||$ calculates the Euclidean distance between point s_i and related cluster center c_l. Membership is denoted with U. The clusters are generated via Equation 2, which aims to predict the instance classes. Equation 2 aims to minimize the sum of the prediction errors of the instances. $Error()$ denotes the error function that takes the class label of an instance $class(s_i)$.

$$J = \sum_{i=1}^{n} \sum_{l=1}^{k} (U_{ij})^m ||s_i - c_l||^2 \tag{1}$$

$$y = \sum_{i=1}^{n} Error(class(s_i)) \tag{2}$$

The membership matrix is initialized at the beginning of the algorithm through Equation 3 ($U_{ij} = \mu_{ij}^r$).

$$U[i,j] = \frac{1}{\sum_{p=1}^{k} \frac{s_i - c_j}{s_i - c_r}^{\frac{2}{m-1}}} \tag{3}$$

Thereafter, fuzzy clusters C are computed via Equation 4, which is meant to minimize the value of J. Equations 2 and 3 are computed iteratively until this requirement is satisfied.

$$C = \frac{\sum_{p=1}^{n} ((\mu_{ij})^r)^m . s_i}{\sum_{i=1}^{n} (\mu_{ij})^r)^m} \tag{4}$$

Initially, the experimental data sets have values of 20 software metrics and their defectiveness label. In complexFuzzy, these values are converted to $p(x, y)$ points; thus, $p(x, y)$ is generated for each instance. The cluster points of the instances are calculated by using Equation 5. In this equation, n denotes the number of metrics. $n/2$ refers to the half of all metrics; thus, x is the mean of the first half of the metrics of the related instance. The same calculation is performed on the second half of the metrics to obtain y. Consequently, the metric values have a decisive role in determining $p(x, y)$. The cluster centers are determined depending on the scale of the data sets afterwards. Usually, the number of clusters ranges from 2 to 5. It is feasible to work with a lower number of clusters if the data sets are small-scale. Although complexFuzzy has some similarities to Fuzzy c-mean in terms of instance values (except for the defectiveness label), it differs in specifying the membership levels of the clusters.

$$x = \frac{\sum_{i=0}^{n/2} x_i}{n/2} y = \frac{\sum_{i=n/2}^{n} y_i}{n/2} \tag{5}$$

A coefficient has been devised for affecting the generation of the membership matrix of complexFuzzy. This coefficient is extracted from specific software metrics, including WMC, RFC, and LCOM, which have shown great promise [4] in creating software

quality indicators. In doing so, each membership value can be changed by ±50, which is a boundary value that was obtained via various trials. This means that the minimum magnitude affecting the level of membership changes according to the properties of the data set. The formula of the coefficient is presented in Equation 6.

$$coEfficient = \frac{\sum_{i=0}^{n}(wmc_i * lcom_i)/rfc_i}{n} \tag{6}$$

n refers to the number of data sets, and rfc is in the denominator because it is an inversely proportional metric to LCOM. WMC is written to the nominator of Equation 6 by controlling whether its value is higher than the threshold (that is, 24) in the experiment. When this value is exceeded, it is reduced as the membership level becomes lower. complexFuzzy can be described in six steps as in Algorithm 1.

Algorithm 1 complexFuzzy Algorithm

Step1: Input all instances as p(x,y) with clusters, fuzziness, and coEfficient.
Step2: Define membership matrix U depending on number of instances and number of clusters.
Step3: Iterate through all instances to create initial U matrix. (Compute $diff = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2}$), IF $coefficient > 10$ diff+=50 else diff-=50; $U[i,j]$=(diff==0) ? 10^{-5} : diff , S+=$U[i,j]$
Step4: Update $U[i,j]$ for each cluster by calculating $U_{ij} = \frac{1}{\sum_{p=1}^{k} \frac{s_i - c_j}{s_i - c_r}^{\frac{2}{m-1}}}$, nsum2+=$U[i,j]$
Step5: $U[i,j] = U[i,j]/sum2$;
Step6: Recalculate cluster indexes by comparing max and $U[i,j]$ values.

Despite the fact that there are some similarities between complexFuzzy and Fuzzy c-mean, they have substantial differences. The differences start with $coEfficient$ defined in Step 1. After membership matrix U is defined in Step 2, the values of membership $U[i,j]$ are specified depending on $coEfficient$. In $U[i,j]$, i denotes the index of the instance, and j denotes the index of the cluster. The sum of these values ($sum2$) is calculated in the next step. Each $U[i,j]$ is updated according to its fuzziness parameter, and the sum of $U[i,j]$ is assigned to $sum2$ in Step 4. $U[i,j]$ values are redefined with the formula described in Step 5. In the last step, max (which is -1 in beginning) is compared with all $U[i,j]$ values. If $max < U[i,j]$, max is assigned as $max = U[i,j]$, and the index of point is determined with $p.ClusterIndex = (max == 0.5)?0.5 : c_j$. c_j represents the related cluster. The algorithm should be re-executed until the accuracy is at the desired level. The accuracy is provided with the procedure presented in Algorithm 2. When the accuracy is obtained as desired, the clustering calculations are suspended. Initially, the cluster parameters and thresholds are given to the algorithm as inputs. The values of $U[i,j]$ are multiplied by the cluster-center Euclidean distance and summed. If the result is less than the threshold, this means that the desired accuracy is obtained; otherwise, the steps are repeated from Step 2.

Algorithm 2 Iteration procedure of complexFuzzy

Step1: Input cluster parameters, threshold with iteration count.
Step2: Calculate objective function (result+=$U[i,j]^{fuzziness} * euclidean(p(x,y),c[j])^2$)
Step3: Calculate cluster centers.
Step4: If $result < threshold$, break; else return step2;

4. Experimental setup

4.1. Data sets

In the study, 29 data sets were used to devise a prediction experiment. The experimental data sets have CK and LOC metrics. The data sets have been selected by reviewing similar works relevant to cross-project defect prediction. The decision formula of the complexFuzzy membership function was completed by utilizing CK metrics (which is widely acknowledged, as it includes beneficial tips for the defectiveness level) [1, 23, 36]. The experimental data sets were retrieved from different open-source projects that have four or fewer versions. The details of the data sets are given in Table 2.

Table 2
Details of projects used in experiment

Project	Version	Number of instances	Total defects	% Defects
ant	1.7	745	338	22
arc	1	234	234	14
berek	1	43	70	37
camel	1	339	14	3
camel	1.2	608	522	35
camel	1.4	872	335	16
camel	1.6	965	500	19
e-learning	1	64	9	7
ivy	1.1	111	233	56
ivy	1.4	241	18	16
ivy	2	352	56	40
jedit	3.2	272	382	33
jedit	4	306	226	24
jedit	4.1	312	217	25
jedit	4.2	367	106	13
jedit	4.3	492	12	2
kalkulator	1	27	7	22
log4j	1	135	61	25
log4j	1.1	109	86	33
log4j	1.2	205	498	92
lucene	2	198	268	46
lucene	2.2	247	414	58

Table 2 (cont.)

Project	Version	Number of instances	Total defects	% Defects
lucene	2.4	340	632	59
nieruchomosci	1	27	13	37
tomcat	6	858	114	8
xalan	2.4	723	156	15
xalan	2.5	803	531	48
xalan	2.6	885	625	46

As shown from the table, these data sets include the different number of instances (ranging from 27 to 965). This case helped generalize the experimental results. While "Total defects" denotes the total number of defects in a data set, "% Defects" shows the percentage of defects encountered in all instances. The "0/1" values in the instances are of great importance for producing this column. Those instances that have "1" or a greater number of defects create the same effect in the clustering.

Some metrics of software defect data sets can also be used for the indicators of defectiveness (except for defect labels). Software that has high cohesion and low coupling helps to improve the quality [4]; otherwise, maintaining the software becomes much more difficult. In addition to this, metrics such as WMC (weighted method per class) are used for prioritizing the test cases; software modules with a high WMC are given a high priority for test execution [40].

Table 3

Metrics of experimental data sets

Name	Description
wmc	Weighted Methods per Class
dit	Depth of inheritance
noc	Number of children
cbo	Coupling between objects
rfc	Response for a class
lcom	Lack of cohesion
ca	Afferent coupling
ce	Efferent couplings
npm	Number of Public Methods
lcom3	A variant of LCOM
loc	Line of codes
dam	Data Access Metric
moa	Measure of. Aggregation
mfa	Measure of functionality abstraction
cam	Cohesion Among Methods of class
ic	Inheritance Coupling
cbm	Coupling between Methods
amc	Average Method Complexity
max_cc	Maximum Class Coupling
avg_cc	Average Class Coupling

The details of the metrics of the data sets are presented in Table 3. Data sets including 20 software metrics are from the projects coded with object-oriented languages. WMC, LCOM, and RFC were utilized to conjecture the defectiveness level of an instance in the complexFuzzy algorithm.

4.2. Machine configurations and setup conditions

The experiment was completed via the R package and C# programming language. The membership values of the points and clustering information were generated by executing C# codes. The figures illustrating the centers of the clusters via normalizing data points were drawn with R package functions (including "rnorm", "plot", and "points"). The mean AUC and F-measure results were obtained by using four predictors harnessed on the R package. The experiment was performed on a CentOS Linux machine with a 64-bit/Intel(R) Xenon(R)/2.9 GHz/32 CPU Core server with 263 GB RAM and a Tesla C1060 graphics processor. The iteration count of complexFuzzy is restricted to 20. While the large-scale data sets were able to reach this count, the small-scale ones were not.

One of the most crucial points during the experiment is to devise cross-project combinations. If $d_1, ..., dn$ denotes the data sets, then n is 29. If one of the data sets is employed as the training data, the others are used as the testing data. In other words, if d_1 is selected as the training data, the testing process is executed for $d_2, ..., d_n$. Consequently, the total number of iterations of the testing is $combinationCount = n * (n - 1)$. For all of the data sets, 812 combinations are produced. Excluding combinations of different versions of the same projects, this resulted in 758 combinations. Thus, these are the ultimate testing operations.

Five data sets have 200 or fewer instances. During the experiment, it was detected that overlapping centers occur when four centers are determined for these data sets. Thus, distinguishing the centers is much more difficult and complex than expected. To solve this problem, two centers are determined in the data sets having 200 or fewer data sets. Such a problem does not occur in other data sets in which four cluster centers are determined. To illustrate the cluster overlapping problem, Figure 3 has been drawn with the ivy-1.1 data set that has 111 instances.

Table 4 presents detailed information about the iteration, time, and threshold. Note that 0.75 is the critical value to proceed with the iteration. The time required to complete a related iteration doubles after 0.75. Prior to this value, the increase rate of the time is constant depending on the threshold. However, the values given in Table 4 could change in accordance with the types of the used data sets.

The centers have been determined as follows:

1) compute mean of instance values;
2) divide data sets into four parts according to this mean;
3) choose maximum values of these parts to assign cluster centers.

Table 4
Iteration and required time for obtaining results according to *threshold* specified by Algorithm 2. *threshold* changes between 0 and 1 for execution of complexFuzzy

Iteration	Time (minutes)	Threshold
10	5	0.2
20	15	0.3
30	27	0.4
40	43	0.6
50	59	0.7
60	70	**0.75**
70	90	0.83
80	110	0.89
90	150	0.92
100	180	0.94

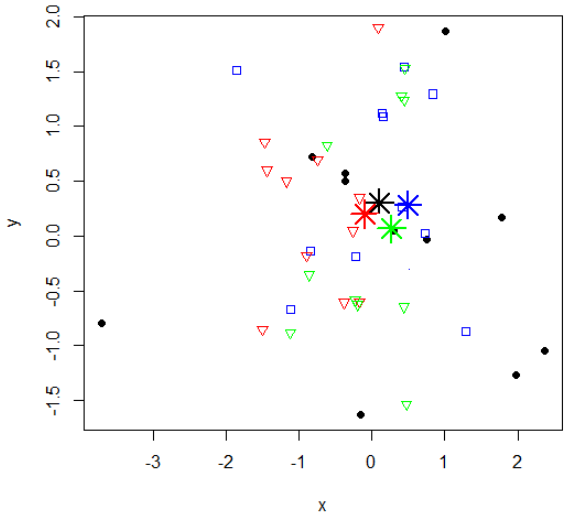

Figure 3. Example of overlapping instance. This figure is of ivy-1.1 cluster results (in which four cluster centers were used)

An example of the raw values of the metrics is presented in Figure 4. complexFuzzy generates a new sheet through a naive data sheet as seen in Figure 5. In the figure, "InstanceIndex" represents the initial instance index before executing complexFuzzy. $p(x, y)$ is in the column named "Point." The cluster index has four different values ranging from "0" to "3." "Value" denotes $U[i, j]$. ***Generally, "0" indicates the instances of defective cluster and training instances that are constituted from this cluster for each data set.***

wmc	dit	noc	cbo	rfc	lcom		bug
3	1	0	10	18	3	CONTINUED	0
9	3	0	5	26	16	METRICS	0
9	1	0	5	19	8	.....	1
1	1	0	10	1	0		0
3	2	0	2	5	0		0
5	2	0	7	14	6		0
11	3	0	3	24	17		0
CONTINUED	VALUES	...	...	...	...		...

METRIC VALUES

CLASS VALUES

Figure 4. Examples of raw metric values. In experiment, column count shows number of features. Last column indicates defectiveness of related software module. Instances having "0" in "bug" do not include any defects

InstanceIndex	Point	ClusterIndex	Value1
0	4.60-14.12	0	0.9176960325 2899
4	3.40-9.42	0	0.8713389281 60539
8	1.10-1.70	0	0.7904647752 48996
12	8.70-11.54	0	0.8277612089 6491
16	7.50-20.89	0	0.8920434788 44525
20	3.60-2.38	0	0.7856221180 86435
28	9.50-20.66	0	0.8291834834 43268
32	6.40-13.58	0	0.8863876643 04627
36	7.10-30.77	0	0.6074039281 06438
40	8.40-8.77	0	0.8069487208 67525
44	7.00-5.42	0	0.7904687958 66468
CONTINUED VALUES	....	...	....

MEMBERSHIP VALUES

Figure 5. Clustered instance points with membership values

In CPDP, the AUC and F-measure performance parameters were recorded. These parameters were obtained by employing 10*10 cross-validation. The mean results on all of the data sets of the predictors (including naiveBayes, Bayes, random forest, and J48) were discussed with tables and figures. The confusion matrix that is constituted with the classification results includes the counts of TP, FP, FN, and TN. The graph drawn by utilizing the true positive rate and false positive rate is called the receiver operating characteristic (ROC). The area under this curve is the AUC; it is desired

that this be close to 1. The formulas of the performance parameters used in the evaluation of the method are presented in Table 5.

Table 5
Performance parameters used in experiment

Name	Formula
True positive rate	TP/(TP+FN)
False positive rate	FP/(FP+TN)
F-measure	$\frac{(2*Recall*Precision)}{(Recall+Precision)}$

5. Results

The answers for the research questions are ordered in terms of both the cluster centers in the working way of complexFuzzy and some performance parameters.
RQ1. To evaluate RQ1, the cluster centers are observed. Furthermore, a clustering evaluation metric called NMI [3] is employed. Having an NMI that is close to 1 means a successful clustering. NMI takes cluster assignment set S for the instances in a data set. Clustering result C must be known to construct the NMI formula as in Equation 7. While $I(S, C)$ denotes mutual information, the entropy is represented with $H(.)$. Table 6 presents the NMI results of all of the comparison algorithms. According to these results, complexFuzzy outperformed the others for all data sets except for camel 0.9 and camel 1.2.

$$NMI(S, C) = \frac{I(S, C)}{\sqrt{H(S)H(C)}} \tag{7}$$

First, complexFuzzy was compared to k-means and Fuzzy c-mean, SOM, MBC, and HC with regard to the cluster centers in Appendix A. This comparison is divided into three groups: small-scale, medium-scale, and large-scale data set comparisons. Figure 9a presents the cluster centers and the related points of lucene 2.2 generated by Fuzzy c-mean. Cluster points $p(x, y)$ were normalized within a range of (-2,2). While a black-colored center denotes a defective cluster, the others are of non-defective instances. When these centers are compared with Figures 9b, 9c, 9d, 9e, and 9f, it is clear that the distance between the centers is far greater in complexFuzzy than in the other algorithms. Furthermore, a black star indicating the center of a defective cluster is further away than the other points. This result showed that the centers drawn by complexFuzzy were more accurately obtained. The cluster structures of camel-1.4 in Fuzzy c-mean, K-means, SOM, HC, complexFuzzy, and MBC can be seen in Figures 10b, 10c, 10d, 10e, and 10f, respectively. This data set yielded better results than lucene 2.2 in terms of both the distances between the clusters and the difference of the defective cluster center. Despite the fact that the data distribution is similar in both Fuzzy c-mean and complexFuzzy, conflicting results were observed in the cluster centers. One of the small-scale data sets is ivy-1.1; complexFuzzy

outperformed the other algorithms in this data set as well. However, Fuzzy c-mean produced more adjacent cluster points in such data sets. The details can be examined in Figures 11b, 11c, 11d, 11e, and 11f, respectively.

Table 6
NMI comparison of clustering algorithms on all data sets

Data Set	Fuzzy c-mean	K-means	complexFuzzy	SOM	MBC	HC
ant	0.331	0.641	**0.818**	0.511	0.520	0.500
arc	0.514	0.568	**0.726**	0.323	0.506	0.511
berek	0.567	0.555	**0.856**	0.501	0.622	0.677
camel 0.9	**0.750**	0.302	0.661	0.640	0.558	0.601
camel 1.2	**0.804**	0.38	0.301	0.573	0.671	0.515
camel 1.4	0.710	0.75	**0.801**	0.759	0.661	0.678
camel 1.6	0.514	0.55	**0.890**	0.623	0.610	0.420
e-learning	0.521	0.24	**0.795**	0.813	0.570	0.518
ivy 1.1	0.403	0.37	**0.772**	0.545	0.681	0.567
ivy 1.4	0.800	0.84	**0.854**	0.813	0.780	0.793
ivy 2	0.501	0.430	**0.679**	0.479	0.347	0.458
jedit 3.2	0.702	0.65	**0.762**	0.442	0.468	0.471
jedit 4	0.755	0.619	**0.781**	0.500	0.469	0.488
jedit 4.1	0.521	0.58	**0.692**	0.569	0.517	0.524
jedit 4.2	0.522	0.600	**0.718**	0.652	0.677	0.648
jedit 4.3	0.677	0.72	**0.802**	0.561	0.549	0.576
kalkulator	0.910	0.91	**0.882**	0.611	0.654	0.678
log4j	0.666	0.701	**0.792**	0.779	0.747	0.738
log4j 1.1	0.820	0.678	**0.849**	0.588	0.544	0.551
log4j 1.2	0.504	0.67	**0.806**	0.421	0.459	0.408
lucene 2	0.711	0.75	**0.779**	0.319	0.307	0.355
lucene 2.2	0.772	0.63	**0.779**	0.579	0.546	0.618
lucene 2.4	0.771	0.72	**0.835**	0.810	0.797	0.768
nieruchomosci 1	0.775	0.825	**850**	0.619	0.767	0.718
tomcat 6	0.733	0.651	**0.764**	0.588	0.597	0.653
xalan 2.4	0.788	0.79	**0.847**	0.819	0.877	0.768
xalan 2.5	0.566	0.723	**0.801**	0.787	0.761	0.908
xalan 2.6	0.766	0.764	**0.880**	0.879	0.847	0.828

The clustering structures of SOM, HC, and MBC are similar to that of k-means. However, they do not have many separated clusters as compared to complexFuzzy (especially defective clusters). The small but distinct differences among the comparison algorithms may have originated from the determination of the cluster numbers and the way of assigning an instance to its cluster.

SOM, HC, and MBC produce different plots to examine their cluster types. In order to present the comparable results, the plot of each clustering algorithm has been converted to the same view.

RQ2. The mean F-measure results of all of the data sets of the predictors are given in Table 7. Note that, in some data sets (such as camel, jedit, and xalan), the prediction

success is higher than with the others. This group consists of medium-large scale data sets. Furthermore, the formula presented in Equation 6 is rather decisive on the average success.

Table 7

Mean F-measure values of four predictors involving Bayes, naiveBayes, random forest, and J48 on 29 data sets. Values generated employing 758 cross combinations. Clustering methods were utilized to generate testing data. For each data set, associated result is mean of some combinations of target data sets. Boldfaced values are best in their respective rows

Source data set	Version	Fuzzy c-mean	K-means	complexFuzzy	SOM	MBC	HC
ant	1.7	0.51	0.5	0.50	0.45	0.48	**0.55**
arc	1	0.54	0.55	**0.57**	0.52	0.49	0.51
berek	1	0.52	0.51	**0.57**	0.49	0.52	0.53
camel	1	**0.68**	0.64	0.64	0.56	0.51	0.50
camel	1.2	0.68	0.68	**0.67**	0.43	0.54	0.47
camel	1.4	0.74	0.73	**0.79**	0.65	0.62	0.63
camel	1.6	0.72	0.71	**0.78**	0.70	0.69	0.65
e-learning	1	0.59	0.57	**0.65**	0.56	0.58	0.62
ivy	1.1	0.6	0.61	**0.65**	0.51	0.58	0.63
ivy	1.4	0.57	**0.64**	0.6	0.58	0.59	0.60
ivy	2	0.56	**0.63**	0.61	0.55	0.58	0.62
jedit	3.2	0.75	0.73	**0.79**	0.71	0.53	0.66
jedit	4	**0.78**	0.72	0.7	0.72	0.75	0.71
jedit	4.1	0.76	0.69	0.68	0.65	**0.78**	0.69
jedit	4.2	**0.73**	0.68	0.7	0.71	0.67	0.70
jedit	4.3	0.79	0.73	0.74	0.76	0.65	**0.81**
kalkulator	1	0.62	0.61	**0.75**	0.70	0.63	0.64
log4j	1	0.52	0.49	**0.56**	0.54	0.55	0.51
log4j	1.1	0.53	0.51	**0.58**	0.54	0.53	0.55
log4j	1.2	0.51	0.55	**0.59**	0.43	0.56	0.57
lucene	2	0.54	**0.59**	0.55	0.48	0.51	0.47
lucene	2.2	0.51	0.5	**0.6**	0.53	0.52	0.49
lucene	2.4	0.54	0.58	**0.64**	0.61	0.60	0.57
nieruchomosci	1	0.61	0.62	**0.63**	0.56	0.52	0.59
tomcat	6	0.62	**0.64**	0.61	0.58	0.51	0.61
xalan	2.4	0.76	0.74	**0.84**	0.61	0.65	0.69
xalan	2.5	**0.78**	0.75	0.74	0.61	0.65	0.66
xalan	2.6	0.77	**0.79**	0.78	0.75	0.72	0.73

The more the magnitude of $coEfficient$ increases, the better F-measure results that are yielded due to its significant effect on the membership level of an instance through $diff$. For instance, the $coEfficient$ values of jedit-3.2 and log4j-1 projects are 2.5 and 6, respectively. Although complexFuzzy produced high performance in the log4j projects, it does not show such success in the jedit projects. In this case, the wmc, rfc, and rfc metrics are prominent. To achieve high prediction scores, the rfc of the experimental data should be at a minimum. On the other hand, those projects with high levels of wmc and rfc are feasible when creating clusters with

complexFuzzy. The data sets that have low F-measure values are the small-scale ones. While Fuzzy c-mean and K-means yielded similar results in terms of their cluster centers, the testing data obtained with complexFuzzy produced higher or equal prediction success than the other two clustering methods in CPDP.

Compared with Zhang et al.'s work [69], complexFuzzy is preferable in terms of its F-measure and AUC scores. For camel 1.6, complexFuzzy has an overwhelming F-measure value (0.78), while their experiment produced an F-measure value of 0.33. When it comes to an AUC comparison, the best AUC score of Zhang et al.'s work is 0.82, which is worse than that of complexFuzzy (0.96). Herbold [17] stressed the importance of the quality issues of CPDP; the findings of complexFuzzy verified his assertion. Hosseini et al. [19] performed a CPDP experiment through genetic instance selection. Their method produced a worse F-measure value than complexFuzzy; the F-measure difference between their method and complexFuzzy is 0.2.

In Figure 6, the ROC curves of the comparison algorithms are presented (along with the AUC values in CPDP). These results are the mean of the records obtained in 29 data sets. The highest AUC was achieved by complexFuzzy (0.96), and the lowest was from HC (0.53). On the other hand, the AUC values of Fuzzy c-mean and HC are similar; they produced relatively worse values than the others. Unexpectedly, k-means is a preferable method in that it yielded a better AUC than all of the other comparison algorithms (except for complexFuzzy).

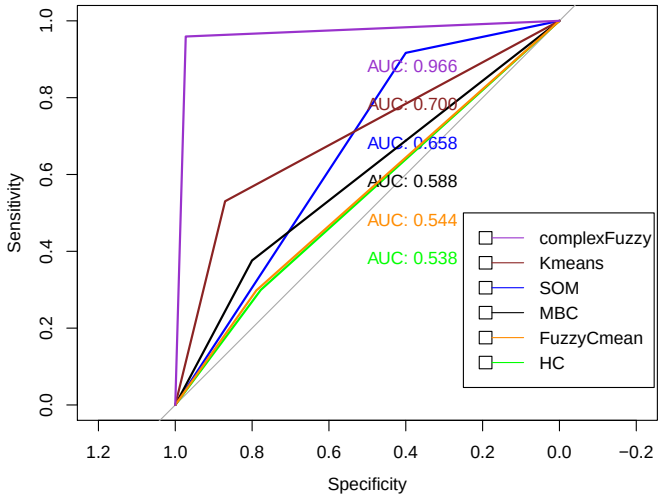

Figure 6. ROC curves of mean cross-project defect prediction in which training data is obtained through six different clustering algorithms. Performance values are means of four classifiers. complexFuzzy outperforms others, producing highest AUC (0.96)

RQ3. Specifically, the metric selection is much more important for data sets that have large-scale software metrics. However, data sets with 30 or fewer features may

produce promising results (for instance, the focused methods from the experiment). Furthermore, unsupervised methods such as clustering could be used after metric selection. In [66], a feature matching-based CPDP was performed by Yu et al.; however, they were only able to reach a 0.79 F-measure in some data sets. On the other hand, complexFuzzy achieved 0.84 F-measure for the xalan data set. Peters et al. [41] proposed a training data-filtering method for CPDP. They achieved a significant improvement in accuracy and G-mean. Likewise, our study has taken step towards understanding the importance of training data selection in CPDP. In summary, complexFuzzy has the potential to be an alternative for metric selection methods with regard to the AUC and F-measure performance parameters.

RQ4: The distance between a defective cluster center and the centers of other clusters is evident, especially in medium- and large-scale data sets. In addition to this, a similar case was detected in small-scale data sets such as ivy-1.1. The F-measure values of medium- and large-scale data sets are better than small-scale data sets in terms of CPDP (as can be seen in Table 7). For instance, some projects such as camel, jedit, and xalan consist of large-scale data sets. In these data sets, the F-measure is relatively high when compared to the other data sets. SOM, MBC, and HC are not better than the others in the sense that the F-measure is the only performance parameter. In summary, complexFuzzy should be executed with large-scale data sets.

In order to provide new insight into the effectiveness of complexFuzzy, a training time comparison is presented in Table 8.

Table 8

Comparison of average training times of data obtained after performing clustering (milliseconds). Boldfaced values are lowest values

Data Set	Version	Fuzzy c-mean	K-means	complexFuzzy	SOM	MBC	HC
ant	1.7	**0.511**	0.609	0.708	0.513	0.520	0.518
arc	1	0.578	0.98	0.704	0.623	0.610	**0.518**
berek	1	**0.404**	0.501	0.407	0.523	0.641	0.517
camel	1	0.9	0.65	**0.302**	0.661	0.640	0.558
camel	1.2	0.304	0.38	**0.301**	0.573	0.671	0.515
camel	1.4	0.73	0.75	**0.601**	0.759	0.661	0.678
camel	1.6	0.51	0.55	0.490	0.623	0.610	0.420
e-learning	1	0.21	0.24	**0.205**	0.813	0.570	0.518
ivy	1.1	0.39	0.37	**0.322**	0.545	0.681	0.567
ivy	1.4	0.8	0.84	**0.774**	0.813	0.780	0.793
ivy	2	0.403	0.43	**0.309**	0.479	0.347	0.458
jedit	3.2	0.7	0.65	**0.402**	0.442	0.468	0.471
jedit	4	0.87	0.619	**0.501**	0.500	0.469	0.488
jedit	4.1	0.503	0.58	**0.492**	0.569	0.517	0.524
jedit	4.2	0.79	**0.602**	0.618	0.652	0.677	0.648
jedit	4.3	0.84	0.72	**0.502**	0.561	0.549	0.576
kalkulator	1	0.96	0.91	**0.582**	0.611	0.654	0.678
log4j	1	0.713	0.701	**0.692**	0.779	0.747	0.738
log4j	1.1	0.801	0.678	**0.519**	0.588	0.544	0.551

Table 8 (cont.)

Data Set	Version	Fuzzy c-mean	K-means	complexFuzzy	SOM	MBC	HC
log4j	1.2	0.543	0.67	**0.406**	0.421	0.459	0.408
lucene	2	0.81	0.75	**0.209**	0.319	0.307	0.355
lucene	2.2	0.773	0.63	**0.542**	0.579	0.546	0.618
lucene	2.4	0.81	0.72	**0.665**	0.810	0.797	0.768
nieruchomosci	1	0.84	0.825	0.705	**0.619**	0.767	0.718
tomcat	6	0.737	0.651	**0.574**	0.588	0.597	0.653
xalan	2.4	0.823	0.79	**0.647**	0.819	0.877	0.768
xalan	2.5	0.849	0.723	**0.711**	0.787	0.941	0.908
xalan	2.6	0.866	0.764	**0.758**	0.879	0.947	0.858

According to this table, complexFuzzy is a formidable rival to other clustering algorithms. Except for the four data sets that included ant-1.7, arc, berek, camel 1.6, nieruchomosci, and jedit 4.2, complexFuzzy produced minimum values. These data sets are not similar in terms of their instance numbers; therefore, the difference in training times of the clustering algorithms may have originated from the software metrics rather than the project's scale. It can also clearly be seen in Table 8 that Fuzzy c-mean yields better results than K-means in the overall evaluation. On the other hand, SOM, MBC, and HC do not have reasonable training times.

6. Threats to validity

The 29 data sets used in the experiment have the same metrics. However, performing CPDP while considering heterogeneity has recently gained a great deal of interest from researchers. Employing data sets that do not require any metric matching, the experiment creates a threat in terms of the generality of the results. As the main objective of the experiment is to observe the effects of the training data sets that are selected with clustering for CPDP, this threat is not so important.

One of the factors that affected the results of the methods is the fuzziness parameter. Determining such a parameter in an unusual way may lead to wrongly interpreting the results. In the experiment, the fuzziness parameter was assigned as two. This value was determined by examining the best ranges from similar studies [72].

The algorithm complexFuzzy has been compared with K-means and Fuzzy c-mean. The reason is that these two clustering methods have construction similarities. Furthermore, the two comparison methods can be considered to be pioneers in their field.

Four cluster centers were used in the experimental data sets (except for the small--scale ones). While one of the centers represents a defective cluster, the instances that include no defects are represented with three clusters. The cluster points are assigned by calculating the means of the instances that have two labels. However, it is ambiguous whether randomly defined cluster centers do indeed yield favorable results. The centers of the data sets were not changed during the $10 \cdot 10$ cross-validation by considering this case.

The experimental data sets that consisted of similar data sets used in the preceding works are important for the validity of the study. In this respect, 29 data sets were selected that were comprised of both feature and instance selection works. Process metrics do indeed produce successful results as well as static code metrics. However, the experiment does not have any data sets that have process metrics. Having only static code metrics may create a threat for the validity, but the metrics that specify the object-oriented parameters were used for both the compatibility of the objective and the performance comparison. Moreover, the formula developed for determining the membership matrix values of complexFuzzy was on the basis of static code metrics.

The calculation presented in Equation 6 was done in complexFuzzy. This calculation did not adversely affect the complexity due to the lack of some expressions (such as loops). This is $O(n^2)$ (as in Fuzzy c-mean).

As the metrics given in Equation 6 affect the value of the membership matrix, it should be investigated whether the distributions of three metric populations are identical. An average result was obtained when applying the Kruskal Wallis test on the three selected metrics, as the metrics do not have a normal distribution. $P < 0.05$ represents a remarkable difference between the mean of the metric groups. The statistic that indicates the mean values of the metrics is presented in Figure 7.

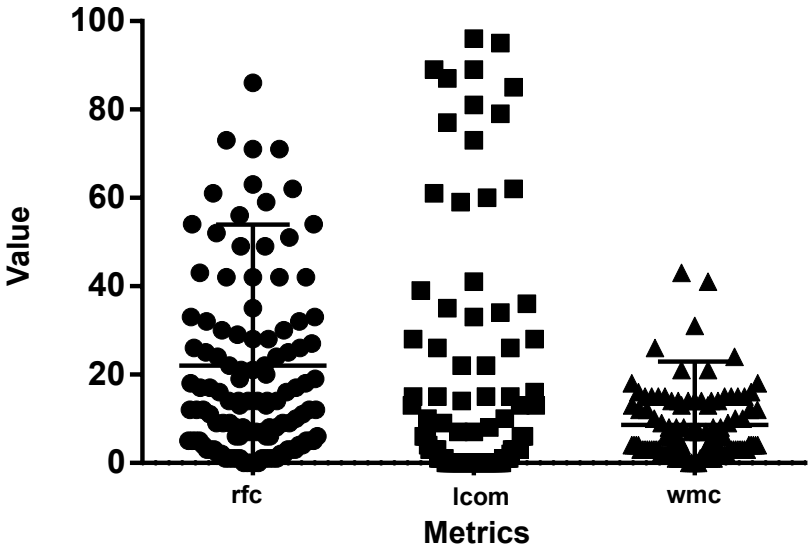

Figure 7. Kruskal Wallis test samples, including RFC, LCOM, and WMC metrics ($p < 0.05$)

Besides the success rates, it is also important to evaluate the error rates when classifying instances. One of the predictors is the random forest algorithm, which was involved in the experiment. In this algorithm, the error rates of the prediction were illustrated in Figures 8a and 8b for all of the experimental data sets that depended on

the generated trees. In these figures, the numbers of trees change between 0 and 100. The black line (namely, out-of-bag [OOB]) shows the average error of each sample of the training observations. The red line represents the instances that have no defects. The green line is labeled with "true1"; it represents those instances with one defect. Those instances with two or more defects are labeled "true2," and their color is blue. Since each execution changes the error rates, Figures 8a and 8b are the averages of 50 executions.

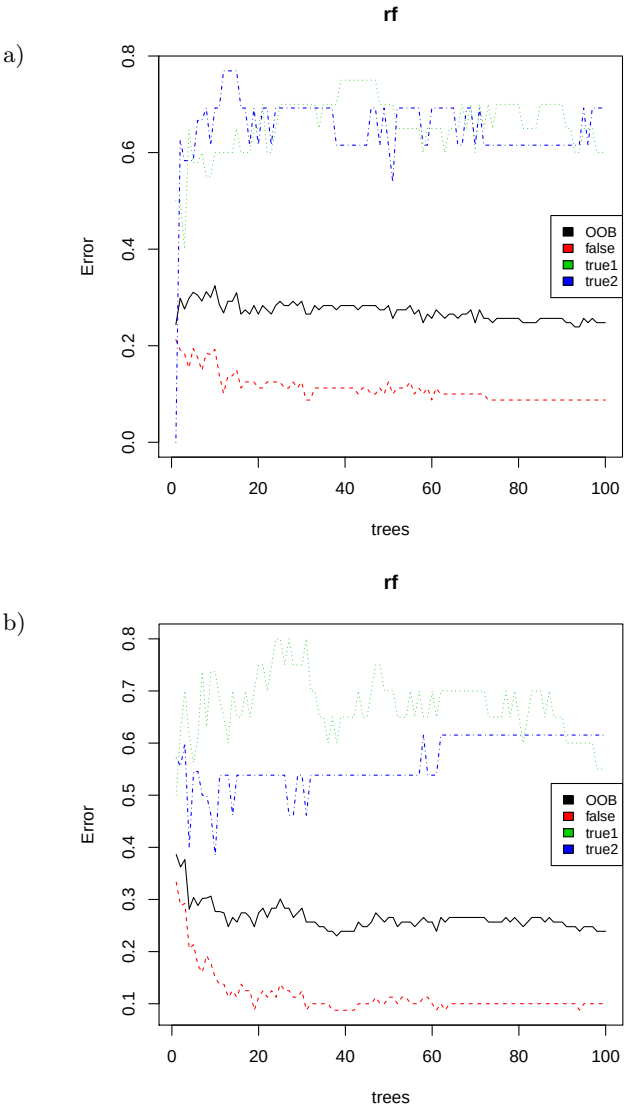

Figure 8. Mean error rates of random forest predictor: a) WPDP; b) CPDP

While "true1" and "true2" were greater than 0.6 in WPDP, only "true1" was greater than 0.6 in CPDP. CPDP produced "true2" values that were lower than 0.6. The results were similar in two figures in terms of OOB. Consequently, complexFuzzy is much more suitable for software systems that have a high level of defectiveness.

In the training and testing processes, four classifiers (Bayes, naïve Bayes, random forest, and J48) were used. Competitive classifiers such as k-nearest neighbor, artificial neural network, and support vector machine were not involved in the experiment, as these classifiers are not common among CPDP practitioners. However, they are planned to be employed in an improved version of the baseline study.

7. Conclusion and future works

In this study, a new way of selecting training instances of CPDP (namely, complexFuzzy) is proposed. The success rates for k-means and Fuzzy c-mean are also calculated in CPDP for comparison with complexFuzzy. Performance values were recorded via 758 cross combinations of CPDP in the experimental design. Four different classifiers were harnessed in CPDP with $10 \cdot 10$ cross-validation. The AUC and F-measure performance parameters were selected. In these parameters, complexFuzzy outperformed Fuzz c-mean and k-means. In particular, complexFuzzy produced promising results in medium- and large-scale data sets. Furthermore, the cluster centers of the method are far more discrete than others. This case may have facilitated the selection of defective instances to expose CPDP.

In summary, complexFuzzy produced the results that were as competitive as those recorded in the featured selection methods. Moreover, it was able to reveal some tips that can be used for performing instance-focused future works.

In future works, the impacts of hybrid methods merging feature and instance selection will be investigated in CPDP. In addition, the formula designed in the study for the indicator of the defectiveness will be further analyzed to adapt it to the process metrics. The algorithm complexFuzzy utilizes Euclidean distance in clustering; however, another direction is to compare the success of complexFuzzy with other popular clustering methods that employ Minkowski or Manhattan distances in calculating the distances between clusters.

Acknowledgements

We thank TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA Resources) for the numerical calculations reported in this work.

APPENDIX A

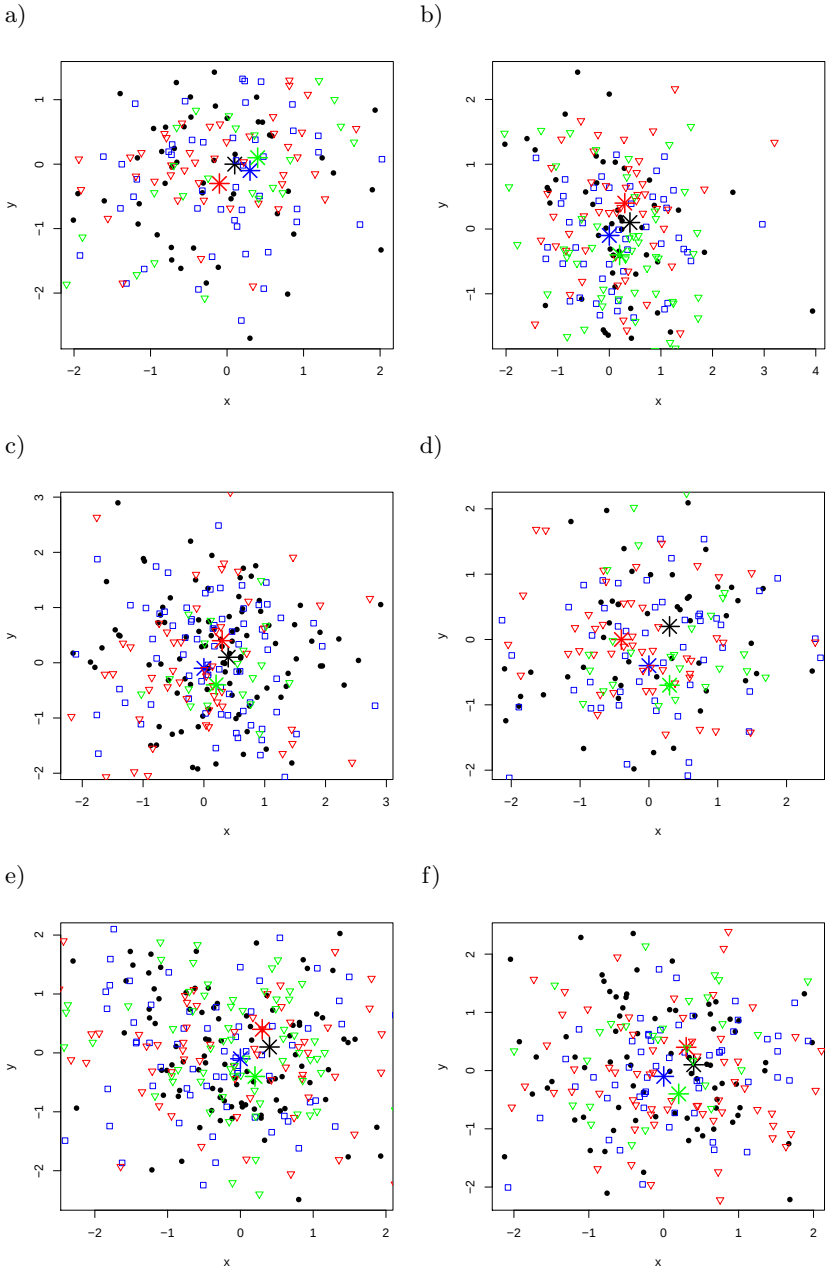

Figure 9. Cluster centers of lucene-2.2 data set: a) Fuzzy c-mean; b) k-means; c) Hierarchical; d) complexFuzzy; e) SOM; f) MBC

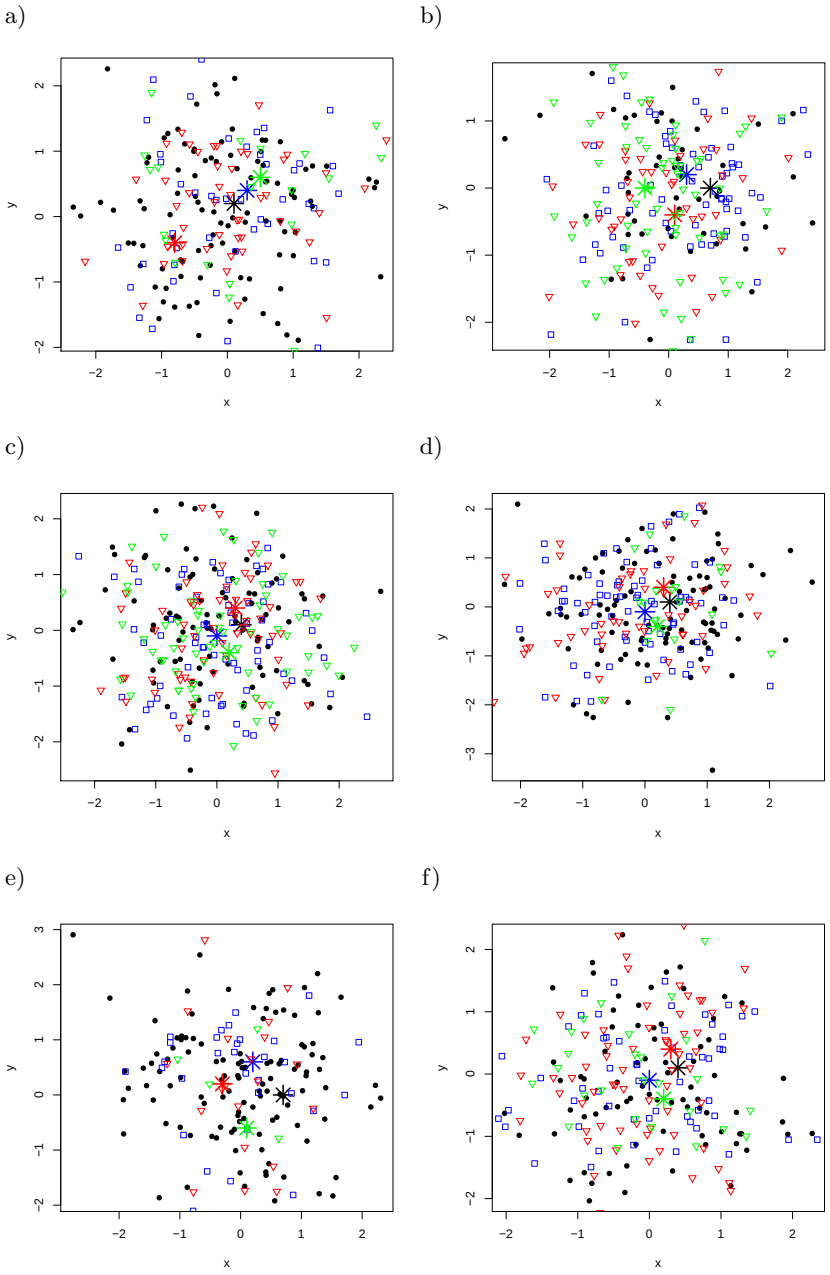

Figure 10. Cluster centers of camel-1.4 data set: a) Fuzzy c-mean; b) k-means; c) Hierarchical; d) complexFuzzy; e) SOM; f) MBC

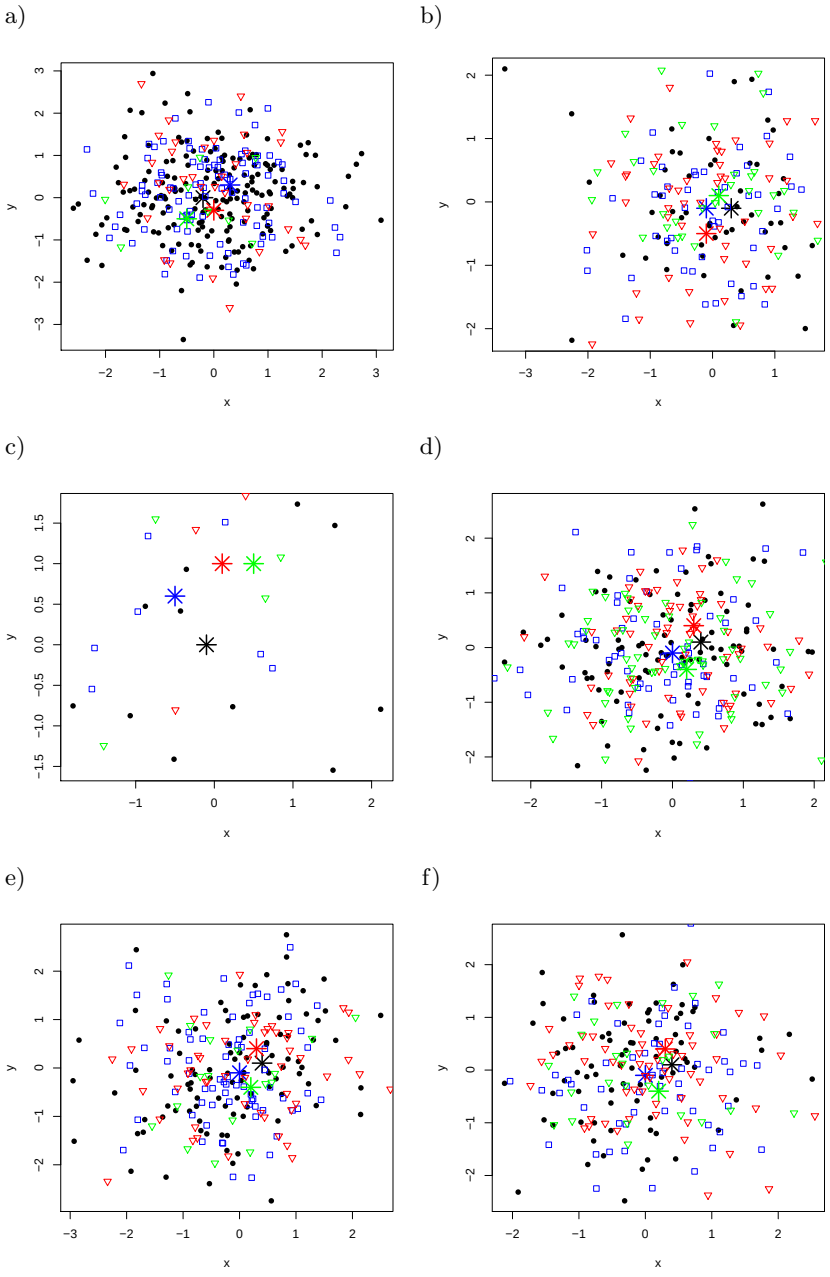

Figure 11. Cluster centers of ivy-1.1 data set: a) Fuzzy c-mean; b) k-means; c) Hierarchical; d) complexFuzzy; e) SOM; f) MBC

References

[1] Bansal M., Agrawal C.: Critical Analysis of Object Oriented Metrics in Software Development. In: *2014 Fourth International Conference on Advanced Computing and Communication Technologies*, pp. 197–201, IEEE, 2014. doi: 10.1109/ACCT.2014.106.

[2] Bishnu P.S., Bhattacherjee V.: Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm, *IEEE Transactions on Knowledge and Data Engineering*, vol. 24(6), pp. 1146–1150, 2012. doi: 10.1109/TKDE.2011.163.

[3] Cai D., Zhang C., He X.: Unsupervised feature selection for multi-cluster data. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining – KDD'10*, p. 333, ACM Press, New York, USA, 2010. doi: 10.1145/1835804.1835848.

[4] Candela I., Bavota G., Russo B., Oliveto R.: Using Cohesion and Coupling for Software Remodularization. Is It Enough? *ACM Transactions on Software Engineering and Methodology*, vol. 25(3), pp. 1–28, 2016. doi: 10.1145/2928268.

[5] Canfora G., De Lucia A., Di Penta M., Oliveto R., Panichella A., Panichella S.: Multi-objective Cross-Project Defect Prediction. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 252–261, IEEE, 2013. doi: 10.1109/ICST.2013.38.

[6] Chan J.Y.K., Leung A.P.: Efficient k-means++ with random projection. In: *International Joint Conference on Neural Networks 2017*, pp. 94–100, 2017.

[7] Chikofsky E.J., Cross J.H.: Reverse engineering and design recovery: A taxonomy, *IEEE Software*, vol. 7(1), pp. 13–17, 1990.

[8] Fenton N., Bieman J.: *Software Metrics: A Rigorous and Practical Approach*, CRC Press, 2014.

[9] Fraley C., Raftery A.E.: Model-Based Clustering, Discriminant Analysis, and Density Estimation, *Journal of the American Statistical Association*, vol. 97(458), pp. 611–631, 2002.

[10] Fraley C., Raftery A.E., Murphy T.B., Scrucca L.: MCLUST version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation, *Technical report*, vol. 597, 2012.

[11] Fritzke B.: The k-means-u* algorithm: non-local jumps and greedy retries improve k-means++ clustering, *arXiv preprint arXiv:170609059*, 2017.

[12] Fukushima T., Kamei Y., McIntosh S., Yamashita K., Ubayashi N.: An empirical study of just-in-time defect prediction using cross-project models. In: *Proceedings of the 11th Working Conference on Mining Software Repositories – MSR 2014*, pp. 172–181, ACM Press, New York, USA, 2014. doi: 10.1145/2597073.2597075.

[13] García H.L., González I.M.: Self-organizing map and clustering for wastewater treatment monitoring, *Engineering Applications of Artificial Intelligence*, vol. 17(3), pp. 215–225, 2004.

[14] Hartigan J.A., Wong M.A.: Algorithm AS 136: A K-Means Clustering Algorithm, *Journal of the Royal Statistical Society Series C (Applied Statistics)*, vol. 28(1), pp. 100–108, 1979.

[15] He P., Ma Y., Li B.: TDSelector: A Training Data Selection Method for Cross--Project Defect Prediction, *arXiv preprint arXiv:161209065*, 2016.

[16] He Z., Shu F., Yang Y., Li M., Wang Q.: An investigation on the feasibility of cross-project defect prediction, *Automated Software Engineering*, vol. 19(2), pp. 167–199, 2012.

[17] Herbold S.: Training data selection for cross-project defect prediction. In: *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, pp. 1–10, 2013.

[18] Herbold S., Trautsch A., Grabowski J.: A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches, *IEEE Transactions on Software Engineering*, vol. 44(9), pp. 811–833, 2018.

[19] Hosseini S., Turhan B., Gunarathna D.: A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction, *IEEE Transactions on Software Engineering*, vol. 45(2), pp. 111–147, 2019.

[20] Hosseini S., Turhan B., Mäntylä M.: 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, 2018.

[21] Huang J., Li Y.F., Xie M.: An empirical analysis of data preprocessing for machine learning-based software cost estimation, *Information and Software Technology*, vol. 67, pp. 108–127, 2015.

[22] Ishida M., Takakura H., Okabe Y.: High-Performance Intrusion Detection Using Optigrid Clustering and Grid-Based Labelling. In: *2011 IEEE/IPSJ International Symposium on Applications and the Internet*, pp. 11–19, IEEE, 2011.

[23] Jabangwe R., Börstler J., Šmite D., Wohlin C.: Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review, *Empirical Software Engineering*, vol. 20(3), pp. 640–693, 2015. doi: 10.1007/s10664-013-9291-7.

[24] Jing X., Wu F., Dong X., Qi F., Xu B.: Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 496–507, 2015.

[25] Kamei Y., Fukushima T., McIntosh S., Yamashita K., Ubayashi N., Hassan A.E.: Studying just-in-time defect prediction using cross-project models, *Empirical Software Engineering*, vol. 21(5), pp. 2072–2106, 2016.

[26] Kant S., Ansari I.A.: An improved K means clustering with Atkinson index to classify liver patient dataset, *International Journal of System Assurance Engineering and Management*, vol. 7(1), pp. 222–228, 2016.

[27] Kumar G.R., Mangathayaru N., Narasimha G.: An improved k-Means Clustering algorithm for Intrusion Detection using Gaussian function. In: *Proceedings of The International Conference on Engineering & MIS 2015*, pp. 1–7, 2015.

[28] Laradji I.H., Alshayeb M., Ghouti L.: Software defect prediction using ensemble learning on selected features, *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[29] Li Y., Huang Z., Wang Y., Fang B.: 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.

[30] Liebchen G.A., Shepperd M.: Data sets and data quality in software engineering. In: *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, pp. 39–44, 2008.

[31] Limsettho N., Bennin K.E., Keung J.W., Hata H., Matsumoto K.: Cross project defect prediction using class distribution estimation and oversampling, *Information and Software Technology*, vol. 100, pp. 87–102, 2018.

[32] Liu C., Yang D., Xia X., Yan M., Zhang X.: A two-phase transfer learning model for cross-project defect prediction, *Information and Software Technology*, 107, pp. 125–136, 2019. doi: 10.1016/j.infsof.2018.11.005.

[33] Ludwig S.: MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability, *International Journal of Machine Learning and Cybernetics*, vol. 6(6), pp. 923–934, 2015.

[34] Ma L., Gu L., Li B., Ma Y., Wang J.: AN Improved K-Means Algorithm Based on Mapreduce and Grid, *International Journal of Grid & Distributed Computing*, vol. 8(1), pp. 189–200, 2015.

[35] Ma Y., Luo G., Zeng X., Chen A.: Transfer learning for cross-company software defect prediction, *Information and Software Technology*, vol. 54(3), pp. 248–256, 2012.

[36] Malhotra R., Chug A.: Application of Group Method of Data Handling model for software maintainability prediction using object oriented systems, *International Journal of System Assurance Engineering and Management*, vol. 5(2), pp. 165–173, 2014. doi: 10.1007/s13198-014-0227-4.

[37] Nam J., Fu W., Kim S., Menzies T., Tan L.: Heterogeneous Defect Prediction, *IEEE Transactions on Software Engineering*, vol. 44(9), pp. 874–896, 2018. doi: 10.1109/TSE.2017.2720603.

[38] Ni C., Liu W., Gu Q., Chen X., Chen D.: FeSCH: A Feature Selection Method Using Clusters of Hybrid-Data for Cross-Project Defect Prediction. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 51–56, IEEE, 2017.

[39] Ni C., Liu W.S., Chen X., Gu Q., Chen D.X., Huang Q.G.: A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction, *Journal of Computer Science and Technology*, vol. 32(6), pp. 1090–1107, 2017. doi: 10.1007/s11390-017-1785-0.

[40] Öztürk M.: Adapting code maintainability to bat-inspired test case prioritization. In: *Proceedings – 2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications, INISTA 2017*, 2017. doi: 10.1109/INISTA.2017.8001134.

[41] Peters F., Menzies T., Layman L.: LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pp. 801–811, IEEE, 2015. doi: 10.1109/ICSE.2015.92.

[42] Peters F., Menzies T., Marcus A.: Better cross company defect prediction. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 409–418, 2013.

[43] Poon W.N., Bennin K.E., Huang J., Phannachitta P., Keung J.W.: Cross-Project Defect Prediction Using a Credibility Theory Based Naive Bayes Classifier. In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 434–441, IEEE, 2017.

[44] Porto F., Minku L., Mendes E., Simao A.: A Systematic Study of Cross-Project Defect Prediction With Meta-Learning, *arXiv preprint arXiv:180206025*, 2018.

[45] Rahman F., Posnett D., Devanbu P.: Recalling the "imprecision" of cross-project defect prediction. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pp. 1–11, 2012.

[46] Ren S., Zhang W., Munir H., Xia L.: Dissimilarity Space Based Multi-Source Cross-Project Defect Prediction, *Algorithms*, vol. 12(1), p. 13, 2019. doi: 10.3390/a12010013.

[47] Rezaee M.R., Lelieveldt B.P., Reiber J.H.: A new cluster validity index for the fuzzy *c*-mean, *Pattern Recognition Letters*, vol. 19(3-4), pp. 237–246, 1998.

[48] Ryu D., Baik J.: Effective multi-objective naïve Bayes learning for cross-project defect prediction, *Applied Soft Computing*, vol. 49, pp. 1062–1077, 2016. doi: 10.1016/j.asoc.2016.04.009.

[49] Ryu D., Jang J.I., Baik J.: A transfer cost-sensitive boosting approach for cross-project defect prediction, *Software Quality Journal*, vol. 25(1), pp. 235–272, 2017.

[50] Salvador S., Chan P.: Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In: *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 576–584, IEEE, 2004.

[51] Sheikholeslami G., Chatterjee S., Zhang A.: WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In: *VLDB*, vol. 98, pp. 428–439, 1998.

[52] Shepperd M., Bowes D., Hall T.: Researcher Bias: The Use of Machine Learning in Software Defect Prediction, *IEEE Transactions on Software Engineering*, vol. 40(6), pp. 603–616, 2014.

[53] Shukla S., Radhakrishnan T., Muthukumaran K., Neti L.B.M.: Multi-objective cross-version defect prediction, *Soft Computing*, vol. 22(6), pp. 1959–1980, 2018.

[54] Suzuki R., Shimodaira H.: Pvclust: an R package for assessing the uncertainty in hierarchical clustering, *Bioinformatics*, vol. 22(12), pp. 1540–1542, 2006.

[55] Turhan B., Menzies T., Bener A.B., Di Stefano J.: On the relative value of cross-company and within-company data for defect prediction, *Empirical Software Engineering*, vol. 14(5), pp. 540–578, 2009. doi: 10.1007/s10664-008-9103-7.

[56] Turver R.J., Munro M.: An early impact analysis technique for software maintenance, *Journal of Software Maintenance: Research and Practice*, vol. 6(1), pp. 35–52, 1994.

[57] Vesanto J., Alhoniemi E.: Clustering of the self-organizing map, *IEEE Transactions on Neural Networks*, vol. 11(3), pp. 586–600, 2000.

[58] Wang X., Wang Y., Wang L.: Improving fuzzy *c*-means clustering based on feature-weight learning, *Pattern Recognition Letters*, vol. 25(10), pp. 1123–1132, 2004.

[59] Wu F., Jing X.Y., Dong X., Cao J., Xu M., Zhang H., Ying S., Xu B.: Cross--project and within-project semi-supervised software defect prediction problems study using a unified solution. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 195–197, IEEE, 2017.

[60] Xia X., Lo D., Pan S.J., Nagappan N., Wang X.: HYDRA: Massively Compositional Model for Cross-Project Defect Prediction, *IEEE Transactions on software Engineering*, vol. 42(10), pp. 977–998, 2016.

[61] Xu Y., Qu W., Li Z., Ji C., Li Y., Wu Y.: Fast Scalable k-means++ Algorithm with MapReduce. In: *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 15–28, Springer, 2014.

[62] Xu Y., Qu W., Li Z., Min G., Li K., Liu Z.: Efficient *k*-Means++ Approximation with MapReduce, *IEEE Transactions on parallel and distributed systems*, vol. 25(12), pp. 3135–3144, 2014.

[63] Xu Z., Yuan P., Zhang T., Tang Y., Li S., Xia Z.: HDA: Cross-Project Defect Prediction via Heterogeneous Domain Adaptation with Dictionary Learning, *IEEE Access*, vol. 6, pp. 57597–57613, 2018. doi: 10.1109/ACCESS.2018.2873755.

[64] Yoon K.A., Kwon O.S., Bae D.H.: An Approach to Outlier Detection of Software Measurement Data using the K-means Clustering Method. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pp. 443–445, IEEE, 2007. doi: 10.1109/ESEM.2007.49.

[65] Yu Q., Jiang S., Qian J.: Which Is More Important for Cross-Project Defect Prediction: Instance or Feature?. In: *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, pp. 90–95, IEEE, 2016.

[66] Yu Q., Jiang S., Zhang Y.: A feature matching and transfer approach for cross-company defect prediction, *Journal of Systems and Software*, vol. 132, pp. 366–378, 2017.

[67] Yu S.S., Chu S.W., Wang C.M., Chan Y.K., Chang T.C.: Two improved k-means algorithms, *Applied Soft Computing*, vol. 68, pp. 747–755, 2018.

[68] Yuan X., Khoshgoftaar T.M., Allen E.B., Ganesan K.: An application of fuzzy clustering to software quality prediction. In: *Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pp. 85–90, IEEE, 2000. doi: 10.1109/ASSET.2000.888052.

[69] Zhang F., Keivanloo I., Zou Y.: Data Transformation in Cross-Project Defect Prediction, *Empirical Software Engineering*, vol. 22(6), pp. 3186–3218, 2017.

[70] Zhang F., Zheng Q., Zou Y., Hassan A.E.: Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 309–320, IEEE, 2016.

[71] Zhang Y., Lo D., Xia X., Sun J.: Combined classifier for cross-project defect prediction: an extended empirical study, *Frontiers of Computer Science*, vol. 12(2), pp. 280–296, 2018.

[72] Zhou K., Fu C., Yang S.: Fuzziness parameter selection in fuzzy c-means: The perspective of cluster validation, *Science China Information Sciences*, vol. 57, pp. 1–8, 2014. doi: 10.1007/s11432-014-5146-0.

[73] Zhou Y., Yang Y., Lu H., Chen L., Li Y., Zhao Y., Qian J., Xu B.: 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(1), pp. 1–51, 2018.

[74] Zimmermann T., Nagappan N., Gall H., Giger E., Murphy B.: 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*, pp. 91–100, 2009.

Affiliations

Muhammed Maruf Öztürk
Suleyman Demirel University, Department of Computer Engineering, Isparta, Turkey,
muhammedozturk@sdu.edu.tr, ORCID ID: https://orcid.org/0000-0001-6446-9754

Received: 03.04.2020
Revised: 12.07.2020
Accepted: 12.07.2020