Soft Computing (2019) 23:13679–13690
https://doi.org/10.1007/s00500-019-03907-6

METHODOLOGIES AND APPLICATION

Unsupervised software defect prediction using signed Laplacian-based spectral classifier

Aris Marjuni[1,2] · Teguh Bharata Adji[1] · Ridi Ferdiana[1]

Published online: 20 March 2019

Abstract
The lack of training dataset availability is the most popular issue in the software defect prediction, especially when dealing with new project development. Adopting training dataset from other software projects probably will not be the best solution because of the software metrics heterogeneity issues across projects. Unsupervised approaches have been proposed to address this issue, where the software prediction model is built without training dataset. Spectral classifier is one of these unsupervised approaches that has been applied successfully to address the lack of training dataset. However, this method leaves an issue when the dataset does not meet the requirement of nonnegative Laplacian assumption. This case would be occurred if there were nonnegative values of the adjacency matrix. It is well known that spectral classifier works with the Laplacian matrix, where the Laplacian matrix is constructed by adjacency matrix. In this paper, the signed Laplacian-based spectral classifier is proposed to solve the negative values problem in the adjacency matrix by converting the negative values into absolute values. The experimental results show that the proposed method could improve the performance of unsupervised classifiers compared to the unsigned Laplacian-based spectral classifier method. Hence, the proposed method is strongly suggested as unsupervised software defects prediction for the software projects that have no historical software dataset.

Keywords Unsupervised software defect prediction · Spectral clustering · Absolute adjacency matrix · Signed Laplacian · Unsigned Laplacian

1 Introduction

Software testing is one of the most important phases in the software development project. In the testing phase, engineers need in-depth test and review on each module to find a potential defects and fix them before released to end users. However, finding defects by tracking a coding path in each module is generally less efficient, as the amount of coding paths in large module or software usually has distributed exponentially (Zhang and Zhang 2007). Developers might be spending more time and resources, and it causes the software testing cost become expensive (Wahono 2015). Another alternative for finding the software defect could be performed by applying software defect prediction (SDP) model (He et al. 2012; Arar and Ayan 2015; Lee et al. 2016).

Implementation of the SDP model provides effectiveness to detect software defects rather than using manual software review, where SDP model can detect 71% of software defects while manual software review only detects about 60% of software defects (Menzies et al. 2010). The contribution of the SDP model will help the developer's team to optimize resources during the software testing phase regarding to achieve software quality. Thus, research in SDP has become an active topic in software engineering to get a better performance of SDP models.

Software defect prediction works with past software metrics dataset for model training (Punitha and Chitra 2013; Petersen 2011). Most of the SDP models are built by supervised approach where the model is trained by labeled datasets

Communicated by V. Loia.

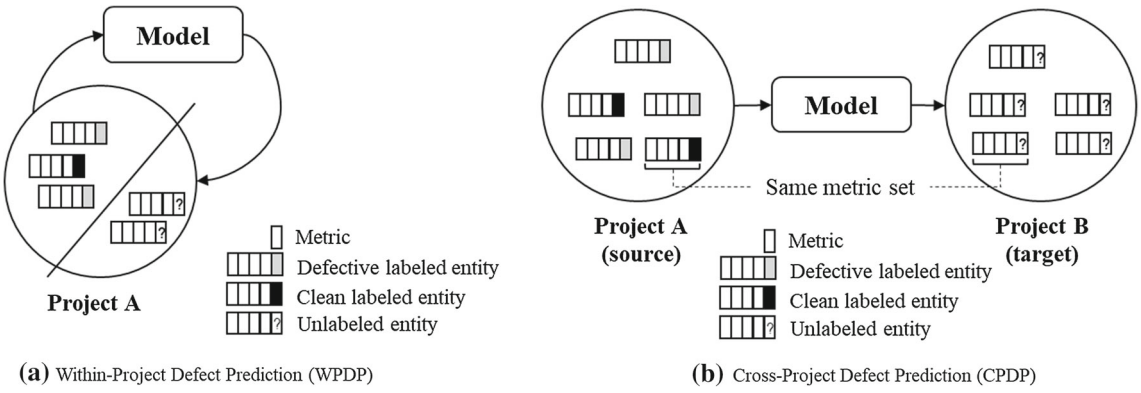
✉ Teguh Bharata Adji
adji@ugm.ac.id

Aris Marjuni
aris.marjuni@mail.ugm.ac.id; aris.marjuni@dsn.dinus.ac.id

Ridi Ferdiana
ridi@ugm.ac.id

[1] Department of Electrical and Information Engineering, Faculty of Engineering, Universitas Gadjah Mada, Yogyakarta 55281, Indonesia

[2] Faculty of Computer Science, Dian Nuswantoro University, Semarang 50131, Indonesia



(a) Within-Project Defect Prediction (WPDP) **(b)** Cross-Project Defect Prediction (CPDP)

Fig. 1 Within-project and cross-project defect prediction scenario (Nam et al. 2017)

to predict defect prone for targeted project (Nam and Kim 2015). Based on source of training datasets, the supervised approach can be divided into two models that are within-project defect prediction and cross-project defect prediction. Within-project defect prediction is SDP model where training dataset and testing dataset are from the same projects, while cross-project defect prediction is SDP model where training dataset comes from different projects (Zhang et al. 2014).

In within project, the defect prediction model is trained by labeled dataset to predict the unlabeled target dataset, as illustrated in Fig. 1a. For example, suppose a new version of software **A** is currently being developed. Also suppose that there is a past software metrics dataset with defective and clean class values of previous versions. The SDP model for the new version of software **A** could be built using the past software metrics of old version of **A**. After the model has been trained, then it will be used to predict new class for unlabeled software metrics of **A**.

In recent years, within-project defect prediction is the most widely proposed model in software defect prediction research because it produces an excellent performance as compared to other models. This is not surprising as this model is built using the same distribution in training and testing datasets. However, it is impossible to build within-project defect prediction models for new software projects since there is a difficulty to provide prior dataset for training (Nam et al. 2017; Zhang et al. 2016). To address this issue, the defect prediction model could be built using cross-project defect prediction approach, as illustrated in Fig. 1b.

In cross-project defect prediction model, training datasets are adopted from other existing projects. For example, suppose a new software project, namely project **B**, is currently being developed. Since there is no historical software metrics dataset of project **B**, the existing similar software from another project, namely project **A**, can be used in the SDP model to predict defect prone for project **B**. The historical data of project A will be used as training dataset with in-depth attributes selection to get a similar dataset distribution between project **A** and project **B**.

This cross-project defect prediction approach is very promising to address unavailability dataset issues in the within-project defect prediction model. However, there are many limitations to build cross-project defect prediction model in practice. First, it is very difficult to find similar software from another project for training dataset which has the same metric distribution with target project. Second, although there might a similar project, the intersection metrics between source and target datasets must be adjusted to get similar dataset distribution. It means the different metrics might be removed from the datasets.

Removing some metrics without prior evaluation is not a good decision as it may decrease the performance of cross-project defect prediction model when the omitted metrics are actually influencing the model. Hence, heterogeneous between source and target datasets has become a challenge issue in cross-project defect prediction model development (Ni et al. 2017; Ryu et al. 2015). To address heterogeneous dataset issue in cross-project defect prediction, some researchers proposed unsupervised approach where the defect prediction model was built using intrinsic target dataset directly without training dataset. In this paper, the unsupervised classifier based on the spectral clustering model proposed by Zhang et al. (2016) is referred as a baseline study.

In spectral clustering, software entities are viewed as a collection of interconnected entities where connectivity between software entities is determined by a similarity function calculated from the software metric values. Software entities that have high degree of similarity are grouped on the same cluster using spectral clustering to get non-overlapped defective and clean clusters. Finally, the average row sums of the normalized metrics of each cluster are applied for labeling on each software entity. The cluster with larger average row sum is considered as the cluster containing defective entities, and all entities within this cluster are labeled as defective. Their proposed method achieved a median AUC value of 0.71, and

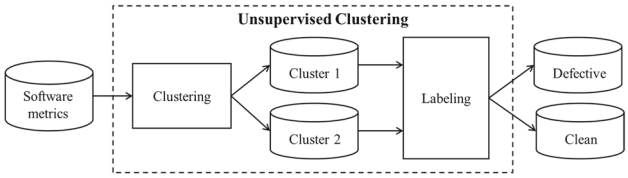


Fig. 2 Unsupervised defect prediction scenario (Zhang et al. 2016)

it outperforms common classifiers such as Random Forest, Naïve Bayes, and Logistic Model Tree with median AUC values of 0.70, 0.68, and 0.68, respectively. Their model has a good achievement to address heterogeneity between source and target datasets.

However, there is an opportunity to improve the spectral clustering application performance. The use of a similarity function in the baseline method has not considered the negative values of the adjacency matrix, since the negative values are converted into zero to meet the nonnegative Laplacian matrix assumption. It means that connectivity between entities with negative similarity is considered as no connection, while it actually has a connection with opposite sign.

In software defect dataset, the original dataset itself actually has no negative values since almost software metric has rarely negative values. Negative values usually come from data preprocessing, such as z-score transformation to make the data as close as normal distribution and also to handle outlier data by scaling transformation. In order to use spectral clustering, there is a nonnegative assumption for Laplacian matrix (Zaki and Wagner 2014; Aggarwal and Reddy 2014) which is built by an adjacency matrix. Hence, if adjacency matrix contains negative values, then it does not meet nonnegative assumption.

In this paper, an absolute transformation for handling negative values in adjacency matrix is proposed for the unsupervised spectral classifier. This paper is organized as follows. Section 2 presents the related works of unsupervised software defect prediction based on spectral clustering. Section 3 explains the detail of our proposed methods for handling negative values in adjacency matrix in order to perform unsupervised software defect prediction based on spectral clustering. Section 4 describes the experimental setup, including the datasets, experiment design, and performance evaluation. Section 5 presents the experimental results and discussion. Finally, the conclusion and future works are summarized in Sect. 6.

2 Related works

Unsupervised software defect prediction generally works with unlabeled dataset. In the software defect prediction scheme, there are two main processes for obtaining the class labels of software entities in the dataset. First, clustering process is applied to group the software entities into clusters which have similar metrics using distance or similarity measures. Second, labeling process is performed on each cluster to obtain defective or clean cluster labels for all software entities. This scheme of unsupervised software defect prediction is illustrated in Fig. 2.

Unsupervised software defect prediction was initially proposed by Zhong et al. (2004) where k-means and natural gas clustering algorithms are applied to cluster software modules. The natural gas algorithm outperforms k-means algorithm. However, it needs an expert to manually label each cluster whether defective or non-defective. Hence, the defect prediction will be dependent on the capability of the human expert.

Catal et al. (2009) proposed the x-means to cluster the software entities. The software entity is predicted as defective if one of its metric's value is higher than the corresponding threshold metric value. The threshold metrics that have been used in this approach included the lines of code, cyclomatic complexity, unique operator, unique operand, total operand, and total operator.

Bishnu and Bhattacherjee (2012) proposed the quad trees algorithm for clustering and classification that outperforms Naïve Bayes and Linear Discriminant Analysis. Abaei et al. (2013) proposed the utilized self-organizing map (SOM) for clustering. The SOM method is applied to cluster software entities. The cluster label is obtained by comparing all metric values with the threshold metric values, which have been used by Catal et al. (2009) previously.

Nam and Kim (2015) proposed the median of metric values for clustering. The median of all entities are computed, and then the number of metrics that have higher values than the median is identified. All entities with the same number of identified higher values are placed in the same cluster. For labeling, all clusters are then partitioned into two groups, the top half of clusters are then labeled as defective, and the bottom half are labeled as clean.

One of the most recent unsupervised software defect prediction is proposed by Zhang et al. (2016) using spectral-based classifier (e.g., Algorithm 1), which is adopted from the spectral graph clustering. The software entities are viewed as the set of nodes and the connectivity between entities are viewed as an edges. Spectral clustering is applied to group the software entities into defective and clean clusters using the eigenvectors characteristic. The cluster labels are then performed by computing the average row sum of each clusters. A cluster with larger average row sum than another cluster is labeled as a defective cluster, and all entities in that cluster are predicted as defective. The complete steps for this unsupervised spectral classifier are explained as follows.

1. Normalize software metrics using z-score in Eq. 1.

$$\hat{y}_j = \frac{y_j - \bar{y}_j}{s_j} \tag{1}$$

where is normalized value of the jth metric, $y_j = \{a_{1_j}, a_{2_j}, \ldots, a_{n_j}\}^{\mathrm{T}}$ is a vector value of the jth metric, a_{ij} is the value of the jth metric on the ith software entity, n is the number of entities in the project, $\bar{y}_j$ is the average value of y_j, and s_j is the standard deviation of y_j.

2. Construct an adjacency matrix W from all the similarity between each pair of software entities, which is computed by normalized values of software metrics. Suppose x_i and x_j are matrices of the metric values of software entities i and j, respectively. The similarity between two entities x_i and x_j is defined as dot product between two matrices x_i and x_j (Aggarwal and Reddy 2014), as shown in Eq. 2.

$$W = (w_{ij}) = x_i \cdot x_j = \sum_{k=1}^{m} a_{ik}.a_{kj} \tag{2}$$

where a_{ik} is the kth metric value on the ith software entity, and m is the total number of metrics. The similarity w_{ij} between the ith and jth entities could be positive, negative, or zero (Zaki and Wagner 2014; Aggarwal and Reddy 2014).

3. Calculate the symmetric normalized Laplacian matrix L_{sym} (Aggarwal and Reddy 2014), as follows.

$$L_{\mathrm{sym}} = I - D^{-1/2} W D^{-1/2} \tag{3}$$

where I is the unity matrix with size of n, W is the adjacency matrix obtained from Eq. 2, and $D^{-1/2}$ is the degree of normalized diagonal matrix computed by Eq. 4.

$$D^{-1/2} = \mathrm{diag}\{d_1^{-1/2}, d_2^{-1/2}, \ldots, d_n^{-1/2}\} \tag{4}$$

where $d^{-1/2}$ is the degree of ith entity obtained by Eq. 5.

$$d^{-1/2} = \left\{ \sum_{j=1}^{n} w_{ij} \right\}^{-1/2} \tag{5}$$

where w_{ij} is the similarity between the ith and jth entities. In spectral clustering, the symmetric normalized Laplacian matrix is assumed as nonnegative, where all of similarity values must be positive semidefinite or $w_{ij} \geq 0$. In baseline method, the negative similarities are converted into zero to meet the nonnegative Laplacian assumption (Zhang et al. 2016).

4. Perform the Eigen decomposition on L_{sym} to get eigenvalues and eigenvectors of L_{sym}, and select the second smallest eigenvalues for clustering. The selected eigenvalue is denoted as v_1.
5. Perform the bipartition on v_1 using zero threshold. Suppose v_{1i} is the eigenvector values related to the eigenvalue of the ith software entity. All software entities with $v_{1i} > 0$ are suspected as defective and separated to the defective cluster (denoted as C_{pos}). Conversely, all of the remaining entities are separated to the clean cluster (C_{neg}) (Zhang et al. 2016).
6. Label each cluster as defective or clean by computing the average row sum of all entities on each cluster. A cluster with larger average row sum than another cluster is predicted as defective, and all entities within this cluster are labeled as defective, and conversely (Zhang et al. 2016).

Algorithm 1 Spectral-based unsupervised software defect clustering (Zhang et al. 2016).

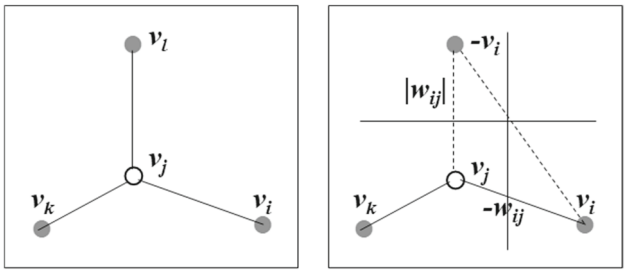
Require: A matrix with rows as software entities and columns as metrics.

Ensure: A vector of defect proneness of all software entities.

1. Normalize software metric using z-score transformation.
2. Construct an adjacency matrix W.
3. Calculate the normalized signed Laplacian matrix L_{sym}.
4. Perform the Eigen decomposition on L_{sym} matrix.
5. Select the second smallest eigenvalue v_1 and select the eigenvector v_{1i} corresponds to the v_1.
6. Perform the bipartition on v_{1i} using zero threshold.
7. Label each cluster as defective or clean.

3 Proposed method

Based on the related works, the unsigned Laplacian-based spectral classifier would be used as the baseline for the proposed method to improve its performance. The unsigned Laplacian is constructed by the adjacency matrix as shown in Eq. 3. The adjacency matrix consists of the similarity between each pair of software entities, as shown in Eq. 2. The elements of adjacency matrix could be either positive, zero, or negative values. If the adjacency matrix W contains negative values, then the $D^{-1/2}$ matrix in Eq. 4 does not exist and the matrix Laplacian L would no longer be positive semidefinite (Knyazev 2017). To address that issue, the absolute transformation can be applied for all adjacency matrix elements to get nonnegative values. In term of graph theory, the graph with no negative values in adjacency matrix is called unsigned graph, as shown in Fig. 3a, while the graph with both negative and positive values in adjacency matrix is called signed graph, as shown in Fig. 3b (Knyazev 2017; Kunegis et al. 2010; Gallier 2016).



(a) Unsigned Graph **(b)** Signed Graph

Fig. 3 Unsigned graph and signed graph diagram (Kunegis et al. 2010)

The negative values of adjacency matrix can be represented as the edge of two opposing nodes (Kunegis et al. 2010). If $w_{ij} \in W$ is a negative value of the edge between node v_i and v_j (i.e., $w_{ij} < 0$), then the node v_i takes value of $-v_i$ adjacent to v_j with $|w_{ij}|$ as the new edge value. Suppose $G = (V, W)$ is a signed graph, where V is a set of vertices $v_{i(i=1,2,3,\ldots,n)}$, W is a symmetric matrix with zero diagonal of nxn in size, and $w_{ij} \in W$ is an edge value between v_i and v_j with $v_i, v_j \in V$. The signed degree of ith entity (Gallier 2016) in Eq. 5 becomes:

$$\bar{d_i}^{-1/2} = \bar{d}(v_i)^{-1/2} = \sum_{j=1}^{n} |w_{ij}|^{-1/2} \tag{6}$$

and the signed degree of normalized matrix D in Eq. 4 becomes:

$$\bar{D}^{-1/2} = \text{diag}\{\bar{d}_1^{-1/2}, \bar{d}_2^{-1/2}, \ldots, \bar{d}_n^{-1/2}\} \tag{7}$$

Hence, the normalized of signed Laplacian matrix could be defined as:

$$\bar{L}_{\text{sym}} = \bar{D}^{-1/2} L \bar{D}^{-1/2} = I - \bar{D}^{-1/2} W \bar{D}^{-1/2} \tag{8}$$

To prove that $\bar{L}^{\text{sym}}$ is positive semidefinite can be evaluated by the quadratic form $x^{\text{T}}\bar{L}x$ using Definition 1 and Proposition 1, as explained in Gallier (2016).

Definition 1 For any real number $\lambda \in \Re$ where $\Re$ is a real number set, the sign of λ is defined as:

$$sgn(\lambda) = \begin{cases} +1 & \text{if } \lambda > 0 \\ 0 & \text{if } \lambda = 0 \\ -1 & \text{if } \lambda < 0 \end{cases} \tag{9}$$

Proposition 1 *For any nxn symmetric matrix $W = (w_{ij})$, if $\bar{L} = \bar{D} - W$ where $\bar{D}$ is the signed degree matrix associated with W, then:*

$$x^{\text{T}}\bar{L}x = \frac{1}{2}\sum_{i,j=1}^{m} |w_{ij}|(x_i - \text{sgn}(w_{ij})x_{ij})^2 \tag{10}$$

Algorithm 2 Modified spectral classifier based software defect clustering.

Require: A matrix with rows as software entities and columns as metrics

Ensure: A vector of defect proneness of all software entities.

1. Normalize software metric using z-score transformation.
2. Construct an adjacency matrix W.
3. Perform absolute transformation of W (e.g., $|W|$).
4. Calculate the normalized signed Laplacian matrix $\bar{L}_{sym}$.
5. Perform the Eigen decomposition on $\bar{L}_{sym}$ matrix.
6. Select the second smallest eigenvalue v_1 and select the eigenvector v_{1i} corresponds to the v_1.
7. Perform the bipartition on v_{1i} using zero threshold.
8. Label each cluster as defective or clean.

for all $x \in \Re$ where $\Re$ is a real number set. Consequently, $\bar{L}$ is positive semidefinite.

Proof

$$\begin{aligned} x^{\text{T}}\bar{L}x &= x^{\text{T}}(\bar{D} - W)x \\ &= x^{\text{T}}\bar{D} - x^{\text{T}}Wx \\ &= \sum_{i=1}^{m} \bar{d}_i x_i^2 - \sum_{i,j=1}^{m} w_{ij}x_i x_j \\ &= \sum_{i,j=1}^{m} (|w_{ij}|x_i^2 - w_{ij}x_i x_j) \\ &= \sum_{i,j=1}^{m} (|w_{ij}|(x_i^2 - \text{sgn}(w_{ij})x_i x_j)) \\ &= \frac{1}{2}\left(\sum_{i,j=1}^{m} (|w_{ij}|(x_i^2 - \text{sgn}(w_{ij})x_i x_j + x_i^2))\right) \\ &= \frac{1}{2}\left(\sum_{i,j=1}^{m} (|w_{ij}|(x_i - \text{sgn}(w_{ij})x_j)^2)\right) \end{aligned} \tag{11}$$

The right-hand side of Eq. 11 is positive semidefinite; hence, $\bar{L}$ is also positive semidefinite. Thus, substituting x to $\bar{D}^{-1/2}$ on the left-hand side of Eq. 11 will result in:

$$\begin{aligned} x^{\text{T}}\bar{L}x &= (\bar{D}^{-1/2}x)^{\text{T}}\bar{L}(\bar{D}^{-1/2}x) \\ &= x^{\text{T}}\bar{D}^{-1/2}\bar{L}\bar{D}^{-1/2}x \\ &= x^{\text{T}}\bar{L}_{\text{sym}}x \end{aligned} \tag{12}$$

Thus, the $x^{\text{T}}\bar{L}x$ from Eqs. 11 and 12 can be written as:

$$x^{\text{T}}\bar{L}x = \frac{1}{2}\sum_{i,j=1}^{m} |w_{ij}|(x_i - \text{sgn}(w_{ij})x_{ij})^2 \tag{13}$$

Hence, $\bar{L}_{\text{sym}}$ is positive semidefinite. □

Table 1 Overview of NASA MDP datasets

Project	Description	Number of entities	Defective	
			Number	Percentage
CM1	Spacecraft instrument	327	42	12.8
KC3	Storage management for ground data	194	36	18.6
MC1	Zero gravity experiment related to combustion	1988	46	2.3
MC2	Video guidance system	125	44	35.2
MW1	Zero gravity experiment related to combustion	253	27	10.7
PC1	Flight software from an earth orbiting satellite	705	61	8.7
PC2	Dynamic simulator for attitude control systems	745	16	2.1
PC3	Flight software for earth orbiting satellite	1077	134	12.4
PC4	Flight software for earth orbiting satellite	1287	177	13.8
PC5	Flight software for earth orbiting satellite	1711	471	27.5

Finally, the implementation of an absolute adjacency matrix on unsupervised based spectral classifiers is summarized in Algorithm 2, as the modified of spectral classifier in Algorithm 1.

4 Experimental setup

In order to ensure that the performance evaluation process of the proposed method works in a structured manner, the experimental setup in this experiment is prepared as follows.

4.1 Dataset

In this experiment, the public dataset of NASA MDP (Menzies et al. 2016) is used to evaluate the proposed method performance. The 10 datasets are used in this study, that are CM1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 software project datasets. The NASA MDP dataset was written in C/C++ language for spacecraft instrumentation, satellite flight control, scientific data processing, and storage management of ground data (Tomar and Agarwal 2016). The overview of these datasets is described in Table 1, and it is available in the PROMISE data defect repository (Zhang et al. 2016; Menzies et al. 2016). The public dataset is chosen due to extensively used in software defect prediction studies.

The purpose of this study is to address the challenge of unavailability of training dataset in cross-project software prediction through unsupervised approach. Hence, all of datasets in this experiment are used once as testing datasets to validate the proposed model (Zhang et al. 2016).

4.2 Experiment design

To evaluate the performance of the proposed method, the experimental design in this experiment is conducted into three steps as follows.

The first step is data preprocessing, including the data cleaning and the data normalization. The data cleaning is performed to remove the duplication records and the missing values of metrics. The z-score transformation in Eq. 1 is then applied to improve the data normalization. This data normalization corresponds to the step 1 in Algorithm 2.

The second step is data clustering using the signed Laplacian-based spectral clustering. This step corresponds to the step 2 until step 7 in Algorithm 2. The motivation of this proposed method is to address the negative values issue of Laplacian matrix in the spectral clustering. For evaluation, the clustering performance is measured by the Davies Bouldin Index as shown in Eq. 15. The performance of the signed Laplacian-based spectral clustering will be compared to the unsigned Laplacian-based spectral clustering, with the null hypothesis $H0_1$: there is no difference in the performance between the signed Laplacian and unsigned Laplacian-based spectral clustering.

The two-tailed Wilcoxon signed-rank test is used to evaluate the $H0_1$ hypothesis with 95% confidence (i.e., $p < 0.05$). The null hypotheses $H0_1$ would be rejected if there is a statistical value of p less than 0.05, and the both of clustering performance are significantly different. The two-tailed Wilcoxon signed-rank test is chosen to test the clustering performance because it is nonparametric and does not require any assumptions on the data distribution.

The third step is labeling or classifying process to obtain defective or clean label for all entities, corresponds to the step 8 in Algorithm 2. This experiment applied unsupervised approach based on spectral classifier, so it does not need training dataset. All of the datasets are used once to test the proposed model as testing datasets. This approach is addressed to solve unavailability training datasets issues in cross-project software defect prediction. In case for labeling, the new label is generated directly from clustering results in the second step. The output of clustering process only has

two clusters that are defective and clean clusters, and it will be used as reference for labeling.

The performance of classifier is measured by $recall$, $precision$, $accuracy$, and AUC, which are explained in Sect. 4.3. All of the proposed method performance values will be compared to the baseline method, with the null hypothesis $H0_2$: there is no difference in the performance between the signed and unsigned Laplacian-based spectral classifier. The null hypothesis $H0_2$ would be tested using the two-tailed Wilcoxon signed-rank test with 95% confidence (i.e., $p < 0.05$). For decision, the null hypotheses $H0_2$ would be rejected if there is a statistical value of p less than 0.05 and the performance of both classifiers is significantly different.

4.3 Performance evaluation

In this experiment, the clustering performance evaluation is measured using the Davies Bouldin Index. It measures the cluster compactness by comparing the distance between the cluster means. The smaller values of the Davies Bouldin Index indicate better performance of clustering, where the clusters are well partitioned (Zaki and Wagner 2014).

Suppose μ_i and μ_j are the mean of cluster C_i and C_j, respectively, and are the total variance of cluster C_i and C_j, respectively, and is the distance between cluster mean of μ_i and μ_j, then the Davies Bouldin (DB) measure for a pair of cluster C_i and C_j is defined as follows.

$$\mathrm{DB}_{ij} = \frac{\sigma_{\mu_i} + \sigma_{\mu_j}}{\delta(\mu_i, \mu_j)} \tag{14}$$

and the Davies Bouldin Index (DBI) is defined as:

$$\mathrm{DBI} = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} \mathrm{DB}_{ij} \tag{15}$$

where k is number of clusters.

For prediction evaluation, the predictive performance is measured using the analysis data of confusion matrix (Hall et al. 2012), as described in Table 2. A confusion matrix (Bishnu and Bhattacherjee 2012) consists of actual cluster labels properties in rows and predicted cluster labels properties in columns, as shown in Table 2 as follows.

- The true negative (TN) is the number of clean entities that are predicted as clean.
- The false positive (FP) is the number of clean entities that are predicted as defective and is usually called Type I Error.
- The false negative (FN) is the number of defective entities that are predicted as clean and is usually called Type II Error.

Table 2 Confusion matrix

Actual	Predicted	
	False (clean)	True (defective)
False (clean)	True negative (TN)	False positive (FP)
True (defective)	False negative (FN)	True positive (TP)

- The true positive (TP) is the number of defective entities that are predicted as defective.

The predictive performance measures (Hall et al. 2012) are then computed as follows.

- $recall$: it represents the proportion of defective entities to the all entities that are actually defective, which is formulated by Eq. 16, and is usually called as sensitivity or true positive rate.

$$recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{16}$$

- $precision$: it represents the proportion of defective entities to the all entities that are correctly predicted as defective, which is formulated by Eq. 17.

$$precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{17}$$

- $accuracy$: it represents the proportion of all entities that are correctly predicted to the total of entities, which is formulated by Eq. 18.

$$accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \tag{18}$$

- AUC: it stands for Area under Curve of ROC (the receiver operating characteristics), which is constructed by the true positive rate and false positive rate as a single curve. The true positive rate is calculated by Eq. 16, while the false positive rate (Zhang et al. 2017) is calculated by Eq. 19.

$$\mathrm{fpr} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{19}$$

5 Experimental results and discussion

This experiment consists three stages that are data preprocessing, clustering, and labeling. The first stage is performing the data preprocessing by applying the z-score transformation on each dataset. The second stage is obtaining the dataset clusters by applying the signed Laplacian-based spectral clustering. The last stage is labeling each both clusters and

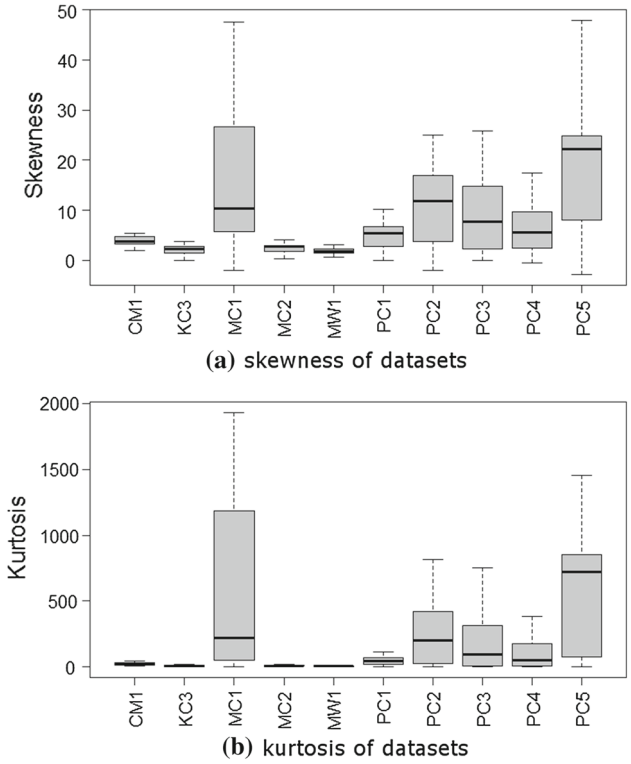


Fig. 4 The box-plot of **a** skewness and **b** kurtosis of datasets

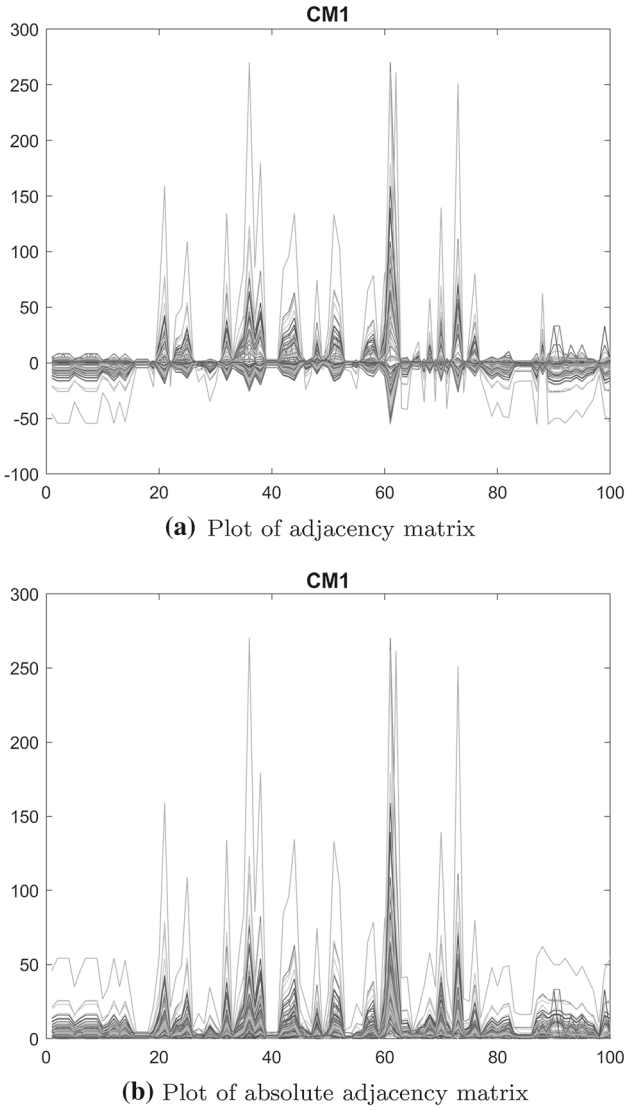


Fig. 5 Adjacency and absolute adjacency matrices distribution of CM1 subset dataset

entities, whether defective or clean. The details of the experimental results are presented in this section.

5.1 Data preprocessing

Software metrics have varying scales in the range of values. In data mining, the software metrics data are commonly normalized to make each metric contribute to the defect prediction model in the same of scale (Nam et al. 2013).

In this experiment, the z-score transformation in Eq. 1 is applied to rescale the datasets within the values of 0 to 1. However, this z-score transformation only rescales the spread of raw data distribution and does not change the shape of distribution based on the skewness and kurtosis values. The z-score transformation itself is a data transformation that could make the software metrics closes to the standard normal distribution (Zhang et al. 2016), and it also improves the prediction performance of classification models (Nam et al. 2013). Hence, it needs to select another normalization method not only to rescale the dataset, but also leads the normalized distribution characteristics at once.

The skewness and kurtosis values among all datasets after z-score transformation are illustrated in Fig. 4. Almost all datasets have highly skewed compared to the ideal skewness, where the ideal skewness values are in the range of -0.8 to 0.8. Also, almost kurtosis values still have highly values compared to the zero value as the perfect kurtosis (Osborne and Carolina 2010).

5.2 Clustering

The clustering process is started by constructing the adjacency matrix of W using Eq. 2. The adjacency matrix could contain a negative, zero, and positive values. To meet the nonnegative Laplacian matrix assumption, the negative values of adjacency matrix are converted into its positive values by absolute transformation. For illustration, Fig. 5a and b shows the example of adjacency matrix values distributions using the CM1 dataset, before and after absolute transformation, respectively. All of the negative values in adjacency matrix in Fig. 5a are transformed into their absolute values, as shown in Fig. 5b.

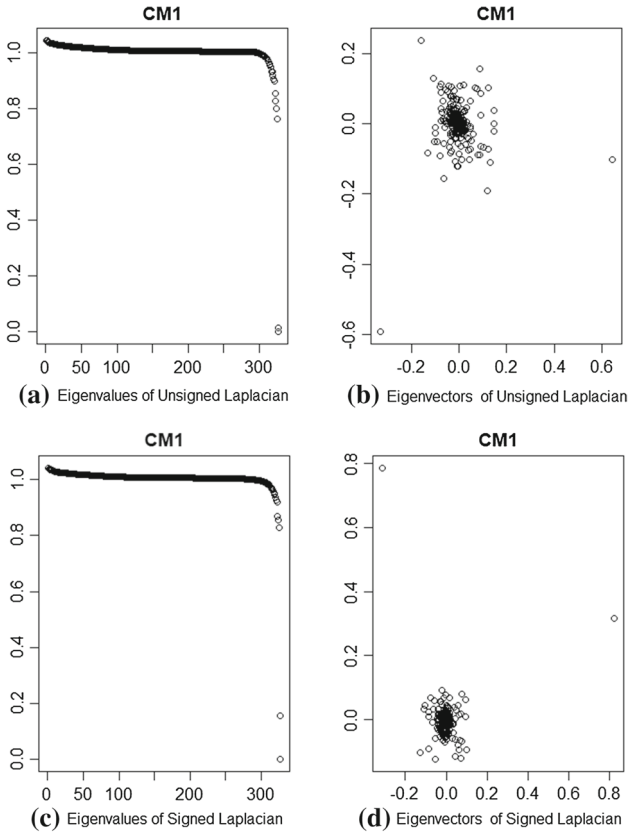


Fig. 6 Eigen values and eigenvectors of signed and unsigned Laplacian of CM1 subset dataset

The next step is calculating the Laplacian matrix. Figure 6a and b shows the eigenvalues and eigenvectors of unsigned Laplacian, respectively. The second smallest eigenvalue of unsigned Laplacian is closer to zero and makes the wide gap with the other eigenvalues. For the signed Laplacian, the second smallest eigenvalue become higher and makes the distribution of eigenvectors more compact, as shown in Fig. 6c and d, respectively. These changes are due to the increase in the number of positive signs, as shown in Table 3, and represent the increase in similarities between entities.

The second smallest eigenvalue itself will be used to select the appropriate eigenvectors for the clustering. Suppose v_1 is the second smallest of signed Laplacian matrix, and v_{1i} is the eigenvectors correspond to the v_1, where $i = 1, 2, \ldots, n$ and n is the number of entities. For clustering process, all software entities with $v_{1i} \geq 0$ are separated to the defective cluster, otherwise, the remaining software entities are separated to the clean cluster.

The performances of these clustering methods are evaluated using the DBI values, as shown in Table 4. The proposed method outperforms the baseline method on five datasets (KC3, PC1, PC2, PC3, PC4, and PC5), fairly comparable

Table 4 Performance comparison of spectral clustering (in DBI)

Dataset	Unsigned Laplacian	Signed Laplacian
CM1	**1.3**	**1.3**
KC3	1.6	**1.4**
MC1	**1.9**	2.0
MC2	**1.7**	1.9
MW1	**1.4**	**1.4**
PC1	2.2	**1.9**
PC2	2.7	**2.2**
PC3	2.8	**2.4**
PC4	2.4	**1.8**
PC5	2.6	**2.1**

The bold font indicates the better performance

Table 3 Overview of adjacency matrix values

Dataset	Number of signed values in W^{a}			Number of signed values in W_0^{b}		Number of signed values in W_a^{c}	
	+	0	−	+	0	+	0
CM1	2128	0	4752	47,606	11,734	59,168	172
KC3	2909	0	4891	10,586	9514	20,000	100
MC1	95,909	0	2,56,617	1,431,208	1,162,295	2,592,364	1139
MC2	1705	0	3248	4717	3414	8064	64
MW1	3395	0	6373	17,886	17,094	34,848	132
PC1	16,526	0	35,237	505,496	473,804	978,600	700
PC2	17,234	0	39,520	640,688	616,217	1,256,112	793
PC3	13,114	0	28,511	318,164	315,211	632,812	563
PC4	9096	0	18,987	146,454	141,966	288,040	380
PC5	15,486	0	51,812	734,118	834,988	1,568,220	886

[a] W: raw dataset matrix after z-transformation
[b] W_0: adjacency matrix of W with zero transformation
[c] W_a: adjacency matrix of W with absolute transformation

Table 5 Performance comparison of unsigned Laplacian (SL) and signed Laplacian (UL) based spectral classifiers

Dataset	*precision*		*recall*		*accuracy*		*AUC*	
	UL	SL	UL	SL	UL	SL	UL	SL
CM1	0.68	**0.72**	**0.92**	**0.92**	0.66	**0.71**	**0.68**	**0.68**
KC3	0.71	**0.78**	0.87	**0.88**	0.67	**0.75**	0.64	**0.75**
MC1	**0.79**	0.76	**0.96**	0.94	**0.78**	**0.78**	**0.69**	0.67
MC2	**0.77**	0.74	**0.76**	0.71	**0.69**	0.68	0.68	**0.75**
MW1	**0.69**	**0.69**	**0.95**	**0.95**	**0.69**	**0.69**	**0.70**	**0.70**
PC1	0.70	**0.73**	**0.95**	0.94	0.67	**0.75**	0.75	**0.78**
PC2	0.63	**0.72**	**0.90**	0.89	0.64	**0.77**	**0.76**	**0.76**
PC3	0.67	**0.72**	**0.93**	0.92	0.65	**0.79**	0.72	**0.77**
PC4	0.68	**0.79**	**0.91**	0.90	0.67	**0.72**	0.65	**0.67**
PC5	0.75	**0.78**	**0.87**	0.81	**0.79**	0.78	0.71	**0.74**
Average	0.71	**0.74**	**0.90**	0.89	0.69	**0.74**	0.70	**0.73**

The bold font indicates the better performance

on two datasets (CM1 and MW1), but underperforms on the other two datasets (MC1 and MC2). Using the two-tailed Wilcoxon signed-rank test at the 95% of confidence level results, the p-value of this test is 0.0469 with the z-value of -2.0304. Hence, the null hypothesis $H0_1$ is rejected and the performance of both clustering methods is significantly different with the DBI's median values of 2.05 and 1.90, respectively. It means that the proposed method could produce a better performance in term of compactness.

5.3 Labeling

Labeling of all software entities is performed by computing the row sum of all entities in all clusters using the signed Laplacian-based spectral classifier (SL). To compare the performance, the unsigned Laplacian-based spectral classifiers (UL) is chosen as the baseline method. Table 5 shows the performance of the baseline and the proposed methods.

In term of *precision*, as shown in Table 5, the proposed method outperforms the baseline method in CM1, KC3, PC1, PC2, PC3, PC4, and PC5 datasets. However, the proposed method underperforms in MC1 and MC2 datasets. The *precision* average values of both classifiers are 0.71 and 0.74, respectively. It means that the baseline and the proposed method correctly predict about 71% and 74% of all predicted defective entities. Using the two-tailed Wilcoxon signed-rank test at 95% level of confidence results, the p-value is 0.049 and the z-value is -2.0732. Since the p-value is less than 0.05, then the H_{02} is rejected and the *precision* performances of these classifiers are significantly different.

In term of *recall*, the baseline method outperforms the proposed method in almost of datasets. In this performance, the proposed method outperforms only on KC3 dataset, as shown in Table 5. The *recall* average values of both classifiers are 0.90 and 0.89, respectively. It means that the baseline and the proposed methods correctly predict about 90% and 89% of all actual defective entities, respectively. These *recall* performances are significantly different by the two-tailed Wilcoxon signed-rank test with 95% level of confidence, where the p-value is 0.0469 and the z-value is -2.1004. Since the p-value is less than 0.05, then the H_{02} is rejected and the recall performance of both classifiers are significantly different. Based on the *recall* values, both of classifiers have a high *recall*. It means that these classifiers predict the labels of entities correctly when compared to the actual labels.

In terms of *accuracy*, the proposed method is more accurate to predict the defective entities than the baseline method. As shown in Table 5, the *accuracy* average values of the baseline and the proposed methods are 0.69 and 0.74, respectively. The proposed method outperforms the baseline method on CM1, KC3, PC1, PC2, PC3, and PC4 datasets, but underperforms on MC2 and PC5. The two-tailed Wilcoxon signed-rank test at 95% level of confidence in performance shows that the accuracy of both classifiers is significantly different, with the p-value is 0.0313 and the z-value is -2.1004.

The proposed method is also evaluated by the *AUC* values on each dataset. The *AUC* value represents the probability of classifier to rank the randomly chosen defect module higher than the randomly chosen non-defect modules (Wahono et al. 2014). The *AUC* average values of the proposed and baseline methods are 0.70 and 0.73, respectively. This performance is significantly different using the two-tailed Wilcoxon signed-rank test at 95% level of confidence, where the p-value is 0.0469 and the z-value is -2.1129. In terms of *AUC*, the proposed method outperforms the baseline method on KC3, MC2, PC1, PC3, PC4, and PC5, as shown in Table 5.

5.4 Discussion

In summary, the proposed method could improve the performance in *precision*, *accuracy*, and *AUC*, although it underperforms in the *recall* term. This performance improvement was suspected due to the increase in the number of similarities between entities through the implementation of absolute signed Laplacian matrix. In this case, the negative values of adjacency matrix were considered as the positive similarity based on their absolute values rather than converted to zero. As the result, the number of similarities between entities in the absolute adjacency matrix increases, as shown in Table 3.

The increase in the numbers of similarities will affect to the signed Laplacian clustering and classifier, because the Laplacian matrix itself is built by the absolute adjacency matrix. For example, if two entities have negative similarity, then the adjacency values in the unsigned Laplacian matrix of this similarity is converted into the zero values to meet the characteristic of the nonnegative Laplacian matrix and it is considered as no similarity between those entities. Conversely, the negative value in the adjacency matrix is converted to their absolute values in the signed Laplacian matrix and considered as positive similarity. Hence, the number of similarities between entities become increases and it affects to the signed Laplacian-based clustering performances.

Another finding of this experiment is found in the use of z-score transformation for the preprocessing phase in the baseline method. The z-score transformation is used to standardize the dataset to close to the standard normal distribution. However, the z-score transformation did not alter the value of skewness and kurtosis in order to lead to the normal distribution. Besides, if the attribute value is larger than its average, then it will lead the negative values that well known as an issue in the Laplacian matrix assumption. The signed Laplacian itself is usually proposed to meet the Laplacian matrix assumption. Hence, it is necessary to try another transformation in future experiment in order to avoid the issues of the spectral-based clustering and classification.

6 Conclusions

Spectral classifier has been applied successfully in unsupervised software defect prediction application area to solve unavailability and heterogeneity issues in training dataset. However, the use of spectral classifier becomes a main issue when the adjacency matrix has a negative value since the spectral classifier works with nonnegative Laplacian matrix which is built by the adjacency matrix.

In this experiment, the signed Laplacian-based spectral classifier has been proposed on the spectral-based clustering and classification to address the negative values of adjacency matrix. In this proposed method, the negative values are converted into the absolute values to increase the number of similarities. The experimental results show that the signed Laplacian classifier achieves better performance than the unsigned Laplacian-based spectral classifier in term of *precision*, *accuracy*, and *AUC*. Besides, the proposed method is also suitable to be applied for spectral clustering instead using the unsigned Laplacian matrix.

For future works, the next two experiments could be considered to improve the signed Laplacian-based spectral classifier. First, selecting the best data transformations for data preprocessing to avoid the negative values in the adjacency matrix, and also to increase the normality of dataset distribution. Second, applying a feature selection method on dataset to construct the signed Laplacian only with the relevant attributes to increase the performance of spectral classifier.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal participants This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Abaei G, Rezaei Z, Selamat A (2013) Fault prediction by utilizing self-organizing map and threshold. In: Proceedings of the 2013 IEEE international conference on control system, computing and engineering (ICCSCE), pp 465–470

Aggarwal CK, Reddy C (2014) Data clustering: algorithms and applications. CRC Press, Boca Raton, pp 177–194

Arar ÖF, Ayan K (2015) Software defect prediction using cost-sensitive neural network. Appl Soft Comput 33:263–277

Bishnu PS, Bhattacherjee V (2012) Software fault prediction using quad tree-based K-means clustering algorithm. IEEE Trans Knowl Data Eng 24(6):1146–1150

Catal C, Sevim U, Diri B (2009) Software fault prediction of unlabeled program modules. In: Proceedings of the world congress on engineering, pp 1–6

Gallier J (2016) Spectral theory of unsigned and signed graphs. applications to graph clustering: a survey, pp 1–122. arXiv:1601.04692

Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304

He Z, Shu F, Yang Y, Li M, Wang Q (2012) An investigation on the feasibility of cross-project defect prediction. Autom Softw Eng 19(2):167–199

Knyazev AV (2017) Signed Laplacian for spectral clustering revisited, pp 1–24. arXiv:1701.01394v1

Kunegis J, Schmidt S, Lommatzsch A, Lerner J, De Luca EW, Albayrak S (2010) Spectral analysis of signed graphs for clustering, predic-

tion and visualization. In: Proceedings of the SIAM international conference on data mining, pp 559–570
Lee T, Nam J, Han D, Kim S, In H (2016) Developer micro interaction metrics for software defect prediction. IEEE Trans Softw Eng 42(11):1015–1035
Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng 17(4):375–407
Menzies T, Krishna R, Pryor D (2016) The promise repository of empirical software engineering data. North Carolina State University, Department of Computer Science, Raleigh
Nam J, Kim S (2015) CLAMI: defect prediction on unlabeled datasets. In: Proceedings of the 30th IEEE/ACM international conference on automated software engineering (ASE), pp 452–463
Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: Proceedings of the 35th international conference on software engineering (ICSE), vol 34(2), pp 382–391
Nam J, Fu W, Kim S, Menzies T, Tan L (2017) Heterogeneous defect prediction. IEEE Trans Softw Eng 99:1–23
Ni C, Liu WS, Chen X (2017) A cluster based feature selection method for cross-project software defect prediction. J Comput Sci Technol 32(6):1090–1107
Osborne JW, Carolina N (2010) Improving your data transformations: applying the Box-Cox transformation. Pract Assess Res Eval 15(12):1–9
Petersen K (2011) Measuring and predicting software productivity: a systematic map and review. Inf Softw Technol 53(4):317–343
Punitha K, Chitra S (2013) Software defect prediction using software metrics: a survey. In: Proceedings of the the 2013 international conference on information communication and embedded systems (ICICES), pp 555–558
Ryu D, Jang JI, Baik J (2015) A hybrid instance selection using nearest-neighbor for cross-project defect prediction. J Comput Sci Technol 30(5):969–980
Tomar D, Agarwal S (2016) Prediction of defective software modules using class imbalance learning. Appl Comput Intell Soft Comput 2016:1–12
Wahono RS (2015) A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks. J Softw Eng 1(1):1–16
Wahono RS, Suryana N, Ahmad S (2014) Metaheuristic optimization based feature selection for software defect prediction. J Softw 9(5):1324–1333
Zaki MJ, Wagner MJ (2014) Data mining and analysis. Cambridge Univerity Press, Cambridge, pp 472–514
Zhang H, Zhang X (2007) Comments on 'data mining static code attributes to learn defect predictors'. IEEE Trans Softw Eng 33(9):635–636
Zhang F, Mockus A, Keivanloo I, Zou Y (2014) Towards building a universal defect prediction model. In: Proceedings of the 11th working conference on mining software repositories (MSR), pp 182–191
Zhang F, Zheng Q, Zou Y, Hassan AE (2016) Cross-project defect prediction using a connectivity based unsupervised classifier. In Proceedings of the 38th international conference on software engineering (ICSE), pp 309–320
Zhang F, Keivanloo I, Zou Y (2017) Data transformation in cross-project defect prediction. Empir Softw Eng 22:3186–3218
Zhong S, Khoshgoftaar TM, Seliya N (2004) Unsupervised learning for expert-based software quality estimation. In: Proceedings of the eighth IEEE international conference on high assurance systems engineering, pp 149–155