ISC-FS: An Improved Spectral Clustering with Feature Selection for Defect Prediction

Xuan Zhou[1], Lu Lu[1,2*], and Yexia Qin[2,3]

[1]School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[2]Modern Industrial Technology Research Institute, South China University of Technology, Meizhou, China
[3]School of Environment and Energy, South China University of Technology, Guangzhou, China
[*]Corresponding author email: lul@scut.edu.cn

***Abstract*—We notice the lack of historical data with labels always exists in software defect prediction (SDP). Unsupervised learning and cross-project defect prediction (CPDP) have tried to address this problem. However, traditional unsupervised learning always requires manual intervention while CPDP faces the challenge of heterogeneity between different projects. Therefore, this paper proposed a framework called Improved Spectral Clustering with Feature Selection (ISC-FS) to conduct unsupervised learning for defect prediction without human effort in this paper. First, ISC-FS clusters the software entities and gets pseudo-labels. Second, we do a feature selection, of which the key idea is different clusters hold the different magnitude of features. Last, the selected features are fed to a spectral clustering method based on connectivity-distance. To validate the proposed method, experiments were carried out on 28 projects from PROMISE and NASA datasets, and comparisons were made with the other five unsupervised methods. The results show the promising performance that ISC-FS can outperform referential methods.**

***Keywords*—Software defect prediction, Unsupervised learning, Spectral clustering**

I. Introduction

With the expansion scale of contemporary software, software defect prediction (SDP) has attracted more and more attention, which can help reduce the test burden and optimize the allocation of testers and developers. It is usually believed that the cost of fixing bugs after deployment is higher than that during development. Given the huge cost and limited budget, it is important that SDP is involved in the early stage of the software life cycle.

Researchers have investigated the different algorithms and features to improve the performance of SDP. SDP tasks usually consist of two steps: capturing features from software entities and training a classifier through machine learning methods. Supervised learning methods predominated in the previous research related to SDP. But unfortunately, SDP is not widely used in industry [1]. That is mainly because that the historical data with labels required in supervised learning is often lack in pre-release software; more importantly, it is also expensive and difficult to collect in post-release software [2].

One way to solve this problem is cross-project defect prediction (CPDP), which attempts to use prediction models built by other projects with sufficient historical data [3]. The main challenge CPDP faces are heterogeneity. On the one hand, different projects may have dissimilar features [4]. On the other hand, even if a source project with the same features as the target project is selected, CPDP still faces the differences in data distribution between source projects and target projects [5]. As shown in Figure 1, the classifier trained by source projects may not be suitable for target projects. Moreover, existing CPDP researches mainly focused on the establishment of a usable target prediction model. However, a large number of irrelevant data usually makes the prediction model perform worse. That explains why CPDP is challenging to achieve promising performance in practice [6].

DOI reference number: 10.18293/SEKE2020-017

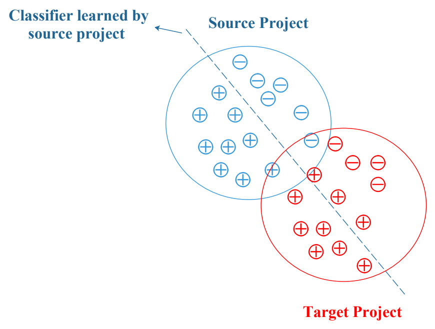


Fig. 1. Divergent data distribution between different projects.

Meanwhile, unsupervised learning has also been used in defect prediction to solve the shortage of labeled data. Nevertheless, early studies on unsupervised learning for SDP often need human effort. To solve this problem, [7] proposed novel approaches showing the defective possibility for software entities by using the magnitude of features. More recently, [8] presented a connectivity-based unsupervised classifier, different from traditional distance-based methods.

As far as this paper is concerned, with the heuristic of [8], we proposed our unsupervised framework for SDP called the Improved Spectral Clustering with Feature Selection (ISC-FS). First, given the importance of data quality [9], the feature selection considering data distribution is added. Second, compared with traditional clustering methods, spectral clustering has the adaptability to different data distributions.

To make the adjacency matrix used in spectral clustering closer to the physical meaning, we present a new adjacency matrix definition, which will be explicated in Section III. Furthermore, following the previous point of view [10], a particular threshold is adopted when labeling entities to avoid human intervention in the classification process. At last, we get the final prediction.

In summary, the contributions of this paper are listed as follows:

- We proposed a novel framework called ISC-FS for SDP on unlabeled datasets.
- To access our proposed method, experiments were conducted on 23 projects in the PROMISE dataset and 5 projects in the NASA dataset. Results showed that ISC-FS performs better over traditional unsupervised methods and state-of-the-art unsupervised approaches for SDP.

The remaining of this paper is structured as followed: In section II, we introduce the related work about SDP and unsupervised learning. Section III presents our method in details. Section IV describes the experimental setup and Section V illustrates the experimental result. In Section VI, we point out the potential threats to validity. The last section concludes and discusses future work.

II. Related Work

We introduce the related work on SDP and unsupervised learning in this section.

A. Software Defect Prediction

SDP is a process predicting whether the software entities have defects or not. There exists many studies on defect prediction. [11] combined semantic and structural scattering to capture project and human characteristics as resource features to build a prediction model. [12] proposed cross-entropy, which carries information representing the difference between two probability distributions. However, all of the above work are using supervised methods.

That is because that people believe unsupervised classifiers usually shown disappointing performance. However, according to the meta-analysis in [2], generally speaking, unsupervised models do not seem to perform worse than supervised models. Under this circumstance, and taking into account that it is easier to collect unlabeled data in the big data era, an effective unsupervised model for SDP when lacking the labeled data is necessary.

B. Unsupervised Learning

Unsupervised learning is a machine learning technique classifying instances to different classes without labeled data. Due to acquiring labeled data is a difficult task, unsupervised learning has been applied in many fields. [13] employed it to the end-to-end training of visual features on large-scale datasets. In SDP, [14] first attempted expert-based unsupervised learning including k-means and neural-gas. Besides, [15] labeled the cluster with a certain threshold. The typical process of unsupervised learning for SDP mainly consisted of two steps, 1) clustering software instances into k clusters; and 2) identifying each class is defect-prone or not.

Recently, spectral clustering has been used in SDP [8]. Spectral clustering, based on Laplacian mapping, uses minimal cuts to characterize the original data with special orientations that carry the cutting information, and then clusters. The main idea is treating data as vertices (V) in space, and these vertices can be connected by edges (E). In SDP, edges represent the connection between software instances, and its weight is determined by the connection between two instances. For clustering, the graph $G = (V, E)$ will be cut into two disjoint sets. The key idea to this cut is that the edge weights between the subgraphs are as low as possible, and the edge weights within the subgraphs are as high as possible.

III. Method

In this section, we explicate our method in detail. Figure 2 illustrates the framework of ISC-FS. To be more specific, ISC-FS includes five steps: (1) preprocessing, (2) spectral clustering, (3) labeling cluster and get pseudo-labels (4) feature selection using pseudo-labels; and (5) re-clustering and labeling by selected features to obtain the final predictions.

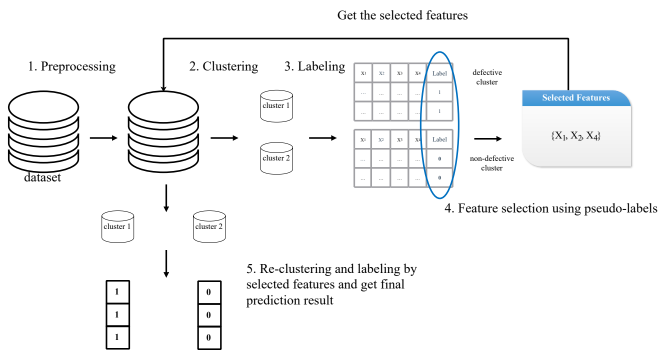

Get the selected features
1. Preprocessing
2. Clustering
3. Labeling
dataset
cluster 1
cluster 2
X1
X2
X3
X4
Label
defective cluster
non-defective cluster
Selected Features
{X1, X2, X4}
4. Feature selection using pseudo-labels
5. Re-clustering and labeling by selected features and get final prediction result

Fig. 2. Overview of ICS-FS framework.

A. Preprocessing software features

Software features usually have various sizes. Take the features in the PROMISE dataset as an example, *loc* means the line of code, which is usually greater than one hundred,or even thousands. And *dam*, the data access metrics, is the ratio of all private or protected attributes in the class to all attributes, which means its range is [0,1]. Consequently, ISC-FS used the z-score normalization, which makes data to a normal distribution with a mean of 0 and a standard deviation of 1. For every specific feature x, the normalized features $x^{'}$ can be shown as:

$$x^{'} = \frac{x - mean(x)}{std(x)} \tag{1}$$

where $mean(x)$ is the average value of x, and $std(x)$ means the standard deviation of x.

Moreover, in some datasets, we observe there are some missing values, which we assign to the average of all existing values of the corresponding features.

B. Spectral Clustering

Guided by the idea of spectral clustering, an adjacency matrix $W \in \mathbb{R}^{n \times n}$ to represent the weight of edges is required at first and we note that n is the number of entities. The previous work [8] adopted the dot product to generate adjacency matrix, as shown in Equation 2.

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

where m represents the number of features, and a_{ij} represents the value of the j_{th} feature of the i_{th} entity.

However, since spectral clustering assumes that values in the adjacency matrix are non-negative, it simply sets the negative value to zero, which will lead to the loss of some original information. Furthermore, according to Equation 2, two more distant points may have larger weights, which is inconsistent with physical meaning. Thus, we proposed a new adjacency matrix definition as shown in the Equation 3.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \sum_{k=1}^{m} exp(-(a_{ik} - a_{jk})^2) & \text{if } i \neq j \end{cases} \tag{3}$$

From Equation 3, each feature has a value ranged in [0,1] represents its similarity, and the sum represents the similarity of two entities. Besides, since it is meaningless to focus on the self-circle, the self-circle value is set to zero.

Second, the Laplacian matrix is calculated by the following formulas:

$$L_{ij}^{sym} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{\sqrt{d_i * d_j}} & \text{if } i \neq j \end{cases} \tag{4}$$

where L^{sym} represents the symmetric normalized Laplacian matrix, and $d_i = \sum_{j=1}^{n} w_{ij}$.

Last, we conduct the eigendecomposition on the Laplacian matrix L^{sym}. Follow the normalized cut algorithm proposed by [10], we use the second smallest eigenvector, denoted as v, for clustering.

C. Labeling Cluster

After spectral clustering, entities has been divided into two groups. C_1 and C_2 are used to denote two groups respectively. To determine whether a cluster is a defective one, we use the following heuristic: *entities with more defect-proneness tend to have higher complexity*. The features in our datasets measure complexity, which means that larger values represent higher complexity. In this case, we adopt the average row sums of the normalized features to determine which cluster is defective. Calculated by Equation 5, if AVS_{C_1} is greater than AVS_{C_2}, we mark C_1 as defective cluster. Otherwise, C_2 is considered as cluster containing defective entities. Specially, the labels obtained by the first clustering are called pseudo-labels, which are used to assist feature selection, and the labels obtained by the second clustering are the final results.

$$AVS_{C_i} = \frac{\sum_{entity \in C_i} RowSum(entity)}{size(C_i)} \tag{5}$$

where AVS_{C_i} represents the average row sums of cluster C_i, $RowSum(entity)$ is the sum of all feature values of the specified entity, $size(C_i)$ means the size of cluster C_i.

D. Feature-selection

Feature selection has a greater influence than classifier selection [17]. Since that the irrelevant or redundant features not only require more computational cost but also reduce the performance of prediction models [18], feature selection based on feature violation scores (FVS) is added in ISC-FS by removing features with less relevant information. FVS can be calculated by the following equation:

$$FVS_i = \frac{V_i}{Num} \tag{6}$$

where V_i means the number of violations in the i_{th} features and Num is the number of entities.

A violation is a value that does not follow the defect proneness heuristic. To reduce manual intervention, the median or average value is commonly used as a special threshold. Nevertheless, in the inherently imbalanced SDP field, the median or average tends to weaken the prediction performance. Thus, we selected the corresponding defect percentile from pseudo-labels as the threshold.

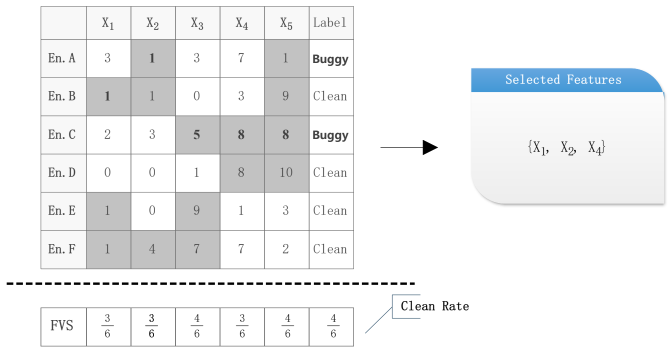
Algorithm 1 FeatureSelection(x, pseudoY)

```
1: $percentileX = percentile(x)$
2: for $i = 1$ to $row$ do
3:     for $j = 1$ to $col$ do
4:         if $pseudoY(i)$ xor $x(i, j) >= percentileX(j)$ then
5:             $violate(j) + +$
6:         end if
7:     end for
8: end for
9: for $j = 1$ to $col$ do
10:    if $violate(j) < percentileX(j)$ then
11:        select the corresponding feature
12:    end if
13: end for
```

The algorithm is shown in Algorithm 1. We give a specific example in Figure 3, the clean rate is $\frac{4}{6}$, and the corresponding percentile is calculated for each feature, shown in bold. For instance, the Entity D is labeled as clean and its X_5 is 10, which is not less than the corresponding percentile, so it should be considered as a violation. After calculating all the FVS, select the features with FVS less than the clean rate. Therefore, X_1, X_2, X_4 as features are finally selected.

E. Re-clustering and labeling by selected features

After feature selection, the results might differ from the pseudo-labels used in feature-selection. In this case, it is necessary to repeat the step of spectral clustering and labeling clustering.



	X_1	X_2	X_3	X_4	X_5	Label
En. A	3	1	3	7	1	**Buggy**
En. B	1	1	0	3	9	Clean
En. C	2	3	5	8	8	**Buggy**
En. D	0	0	1	8	10	Clean
En. E	1	0	9	1	3	Clean
En. F	1	4	7	7	2	Clean
FVS	$\frac{3}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{4}{6}$

Fig. 3. An instance of computing feature violation scores (FVS), violations are shown in a dark gray shade.

IV. Experiment Setup

A. Experiment Datasets

Table I lists the statistical characteristics of the 28 datasets from two groups, the PROMISE and the NASA, used in our experiments, which is commonly used in recent SDP researches [19] [20].

- The PROMISE dataset collected by [21], including data from different versions. As it is likely to be a significant overlap between different versions of a project, we considered a version as a separate project. Each project in PROMISE dataset contains 20 features.
- The NASA dataset is collected from the NASA Metrics Data Program. Each NASA project includes various static code metrics (CMs) of a NASA software system or subsystem, as well as the corresponding defect label data.

B. Comparative Methods

The following five unsupervised learning methods are selected to compare with ISC-FS:

- K-means: A traditional classic clustering algorithm, first used in SDP in [14].
- Partition around medoids (PAM): A method of K-medoids. The method uses medoids as a reference point, which solves the problem that K-means is extremely susceptible to extreme values.
- CLA: An unsupervised method proposed by [7] for SDP. The name is taken from the first letter of the steps are **C**lustering and **LA**beling instances.
- CLAMI: An improved version of CLA by adding **M**etric selection and **I**nstance selection.
- Spectral Clustering (SC): A connectivity-based unsupervised method proposed by [8].

It should be pointed out that we adopted the same heuristic rules as ISC-FS are adopted when labeling the cluster in the first two baselines, while the last three algorithms have their own clear rules in the corresponding researches [7] [8].

C. Experiment Datasets

The classifier has many widely used measures for classifiers, such as accuracy, which is the most traditional measure in classification tasks, and it represents the ratio of correct prediction.

However, accuracy usually does not deal well with the imbalanced datasets. Furthermore, a critical value for the probability of defect-proneness is required when computing accuracy and many other measures (e.g. F-score). The critical value can affect the performance and the default value (i.e. 0.5) may not be the best critical value [22]. Thus, we adopted the Area Under Curve (AUC), which is independent of critical value and has good tolerance for imbalanced datasets, to evaluate the effectiveness of the approaches.

AUC is defined as the area under the ROC (receiver operating characteristic curve). ROC refers to a curve of the false positive rate (FPR) against the true positive rate (TPR). The FPR and TPR can be expressed in Equation 7 and 8. From the above definition, it can be seen the AUC value ranges in [0, 1], and an AUC value of 0.5 indicates that the effect of the classifier is almost the same as the random guessing. The higher AUC is, a better result implies.

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

where TP, FN, FP and TN can be calculated from Table II.

V. Result

We evaluated our proposed method by the Scott-Knott test [23], using hierarchical clustering to divide different methods into groups with significant statistical differences. In this study, we adopted the normality and effect size aware variant of the standard Scott-Knott test, Scott-Knott test effect size difference (ESD) [24]. Based on the traditional Scott-Knott test, the Scott-Knott ESD test has the following improvements: (1) correct non-normal distribution inputs, which are considered to be normally distributed in traditional Scott-Knott testes; and (2) merge any two statistically different groups with a negligible effect size into one group. We used the function *sk_esd* in the *ScottKnottESD* R package to make the implementation.

The result is shown in Figure 5. After comprehensive observation, ISC-FS outperformed 5 reference methods in our experiments. The following points can be drawn from a detailed observation:

- In the PROMISE dataset, the average AUC of ISC-FS was 0.694, which outperformed K-means, PAM, CLA, CLAMI, SC by 44.58%, 43.68%, 5.15%, 14.33%, 3.74% respectively.
- In the NASA dataset, the average AUC of ISC-FS was 0.685, which outperformed K-means, PAM, CLA, CLAMI, SC by 42.12%, 35.91%, 2.24%, 2.39%, 1.48% respectively.
- The results are divided into groups with different statistical differences in Figure 5. Our results are statistically different from the other comparison methods in both of the two datasets.

TABLE I
DATASETS

Group	Granularity	Project	Version	Avg. instance	Avg. Buggy Rate(%)	# of metrics
PROMISE	class	ant	1.3 1.4 1.5 1.6 1.7	338	19.58	20
		ivy	1.1 2.0	232	34.06	
		jedit	4.0 4.1 4.2 4.3	369	16.29	
		log4j	1.0	135	25.19	
		lucence	2.2 2.4	294	59.00	
		poi	3.0	442	63.57	
		synapse	1.0 1.2	257	21.70	
		velocity	1.5 1.6.1	222	50.21	
		xalan	2.4 2.5 2.7	812	52.23	
		camel	1.2	608	35.53	
NASA	function	CM1	–	505	9.50	37
		KC1		2107	15.42	21
		KC3		458	9.39	39
		KC4		125	50.40	13
		MC2		161	32.30	39

TABLE II
THE CONFUSION MATRIX

	Predicted defective	Predicted non-defective
Actual defective	True Positive (TP)	False Negative (FN)
Actual non-defective	False Positive (FP)	True Negative (TN)

- K-means and PAM showed the lowest performance because they adopt a different mechanism with spectral clustering, which only clustering based on Euclidean distance between entities.
- We noticed that the CLAMI performed worse than CLA. However, in [7], CLAMI is proposed as an enhanced version of CLA by adding metric selection and instance selection to improve the ability of prediction models. We studied on it and observed that CLAMI only selected features with the minimum metric violation score (MVS), however, in a real dataset with a large number of instances, the MVS of different features often varies. In this case, dozens of features will be selected as few features in some projects, which will weaken the performance.

VI. Threats to validity

We discuss a few threats to the validity in this section.

A. Implementation of CLA and CLAMI

In this study, we compared our approaches with 5 referential methods. Due to the unpublished implementations of CLA and CLAMI, we have reimplemented our version according to the corresponding paper. Although we strictly followed the procedures reported, our implementation may not reflect all details in the comparative method.

B. AUC might not be the only suitable measures

We used AUC as measures to evaluate the performance of SDP models. There are many other measures (e.g., G-measure, MCC) can be used for performance evaluation. In fact, AUC is a widely used evaluation measure in SDP tasks [4] [7] [8].

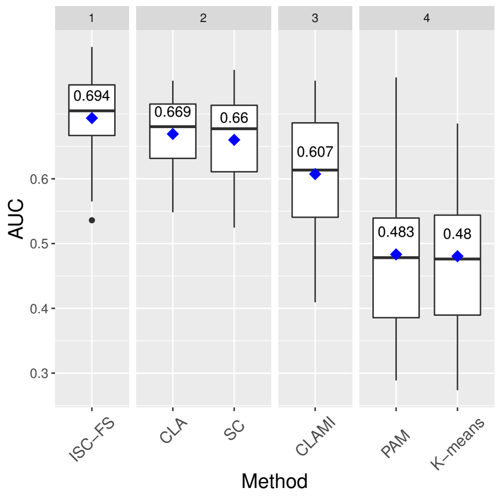
C. Experimental results might not be generalizable:

In our experiment, we selected 28 projects that have been widely used in SDP. However, diverse software projects have different characteristic, so there is no guarantee that our findings will be applicable to other projects. More validation should be conducted in the future.

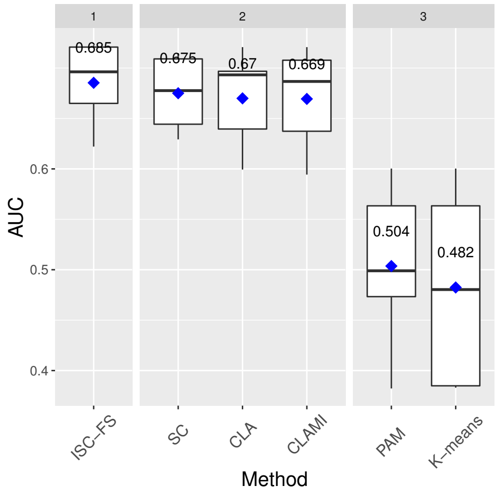
VII. Conclusion and future works

In this paper, we proposed an unsupervised learning method called ISC-FS to solve the problem of lacking historical data in SDP. The main advantage of ISC-FS is that it introduces feature selection considering data distribution and improves spectral clustering. A large number of experiments on 28 projects from 2 groups have been conducted to assess that the proposed method can perform better in terms of AUC than the referential approaches.

In the future, we plan to evaluate our approach with more datasets from different sources. Furthermore, we will try to extend our approach to the semi-supervised version by introducing a few labeled data.



(a) Promise



(b) NASA

Fig. 4. The Scott-Knott ESD ranking of 6 methods

Acknowledgment

This work was supported in part by National Nature Science Foundation of China under grant no. 61370103, the Guangdong Province Application Major Fund under grant no. 2019A0101019 and Guangzhou Produce & Research Fund under grant no. 201902020004.

References

[1] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "The adoption of machine learning techniques for software defect prediction: An initial industrial validation," in *Joint Conference on Knowledge-Based Software Engineering*. Springer, 2014, pp. 270–285.

[2] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *arXiv preprint arXiv:1907.12027*, 2019.

[3] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2017.

[4] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, 2017.

[5] S. Qiu, H. Xu, J. Deng, S. Jiang, and L. Lu, "Transfer convolutional neural network for cross-project defect prediction," *Applied Sciences*, vol. 9, no. 13, p. 2660, 2019.

[6] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100.

[7] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 452–463.

[8] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 309–320.

[9] T. M. Khoshgoftaar and N. Seliya, "The necessity of assuring quality in software measurement data," in *10th International Symposium on Software Metrics, 2004. Proceedings.* IEEE, 2004, pp. 119–130.

[10] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Departmental Papers (CIS)*, p. 107, 2000.

[11] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5–24, 2017.

[12] X. Zhang, K. Ben, and J. Zeng, "Cross-entropy: A new metric for software defect prediction," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 111–122.

[13] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[14] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation." in *HASE*. Citeseer, 2004, pp. 149–155.

[15] C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 199–204.

[16] J. Yang and H. Qian, "Defect prediction on unlabeled datasets by using unsupervised clustering," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2016, pp. 465–472.

[17] A. Agrawal and T. Menzies, "Is better data better than better data miners?: on the benefits of tuning smote for defect prediction," in *Proceedings of the 40th International Conference on Software engineering*. ACM, 2018, pp. 1050–1061.

[18] W. Liu, S. Liu, Q. Gu, X. Chen, and D. Chen, "Fecs: A cluster based feature selection method for software fault prediction with noises," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 276–281.

[19] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[20] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2016.

[21] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, p. 9.

[22] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, 2016.

[23] E. Jelihovschi, J. C. Faria, and I. B. Allaman, "The scottknott clustering algorithm," *Universidade Estadual de Santa Cruz-UESC, Ilheus, Bahia, Brasil*, 2014.

[24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.