2016 IEEE/ACM 38th IEEE International Conference on Software Engineering

Cross-project Defect Prediction Using a Connectivity-based Unsupervised Classifier

Feng Zhang[1], Quan Zheng[1], Ying Zou[2], and Ahmed E. Hassan[1]
[1]School of Computing, Queen's University, Canada
[2]Department of Electrical and Computer Engineering, Queen's University, Canada
[1]{feng, quan, ahmed}@cs.queensu.ca, [2]ying.zou@queensu.ca

ABSTRACT

Defect prediction on projects with limited historical data has attracted great interest from both researchers and practitioners. Cross-project defect prediction has been the main area of progress by reusing classifiers from other projects. However, existing approaches require some degree of homogeneity (*e.g.*, a similar distribution of metric values) between the training projects and the target project. Satisfying the homogeneity requirement often requires significant effort (currently a very active area of research).

An unsupervised classifier does not require any training data, therefore the heterogeneity challenge is no longer an issue. In this paper, we examine two types of unsupervised classifiers: a) distance-based classifiers (*e.g.*, *k*-means); and b) connectivity-based classifiers. While distance-based unsupervised classifiers have been previously used in the defect prediction literature with disappointing performance, connectivity-based classifiers have never been explored before in our community.

We compare the performance of unsupervised classifiers versus supervised classifiers using data from 26 projects from three publicly available datasets (*i.e.*, AEEEM, NASA, and PROMISE). In the cross-project setting, our proposed connectivity-based classifier (via spectral clustering) ranks as one of the top classifiers among five widely-used supervised classifiers (*i.e.*, random forest, naive Bayes, logistic regression, decision tree, and logistic model tree) and five unsupervised classifiers (*i.e.*, *k*-means, partition around medoids, fuzzy C-means, neural-gas, and spectral clustering). In the within-project setting (*i.e.*, models are built and applied on the same project), our spectral classifier ranks in the second tier, while only random forest ranks in the first tier. Hence, connectivity-based unsupervised classifiers offer a viable solution for cross and within project defect predictions.

CCS Concepts

•**Software and its engineering** → *Software verification and validation; Software defect analysis;*

ICSE '16, May 14 - 22, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3900-1/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884839

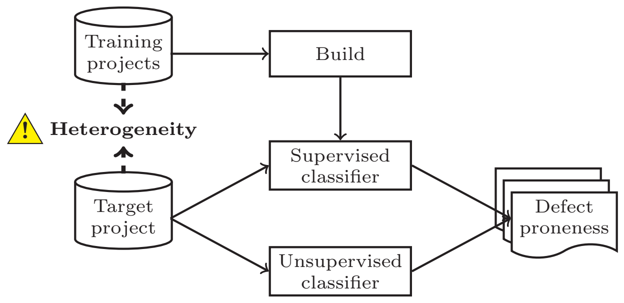

Figure 1: Illustration of the heterogeneity challenge.

Keywords

defect prediction, heterogeneity, cross-project, unsupervised, spectral clustering, graph mining

1. INTRODUCTION

A defect is an error in a software system that causes a system to behave improperly or produce unexpected results. Fixing defects typically consumes about 80% of the total budget of a software project [55]. Such cost can be significantly reduced if defects are fixed in an early stage [3, 11, 18, 37, 38, 46, 54]. Hence, defect prediction models are often used to prioritize quality improvement and defect avoidance efforts.

However, defect prediction is not widely adopted in industry [45, 49, 56]. The barriers include the cost of collecting up-to-date training data (*e.g.*, defect data) [45, 49, 56, 57], the low generalizability of prediction models [49], and the lack of automated tooling for the prediction process [10, 49, 56]. Moreover, many companies lack the needed resources and technical expertise to prepare data for building defect prediction models [45]. A typical solution (*i.e.*, cross-project prediction) is to apply defect prediction models that are built using data from other training projects using supervised classifiers [56, 62].

As illustrated in Figure 1, the major challenge in cross-project prediction comes from the heterogeneity between the training projects and the target project [13, 41]. The heterogeneity may be caused by diverse development settings (*e.g.*, varying user requirements and developer experiences) [8, 32]. It is common that the distribution of metric values of software entities (*e.g.*, files or classes) exhibits significant differences across projects with varied contexts (*e.g.*, size and programming language) [64]. Another heterogeneity challenge in cross-project prediction, as pointed out recently by Nam and Kim [40], is that different projects may have different sets of metrics all together.

To mitigate such challenges, an unsupervised classifier could be used. As shown in Figure 1, such classifiers do not require any training data, and are therefore by nature free of the challenges that are due to heterogeneity of the training and target projects. However, distance-based unsupervised classifiers (*e.g.*, k-means) have shown disappointing performance for within-project defect prediction (*e.g.*, [65]).

In this study, we propose to apply a connectivity-based unsupervised classifier that is based on spectral clustering [43, 59]. Unlike distance-based unsupervised classifiers that partition the data based on Euclidean distance, spectral clustering considers the connectivity among all entities and therefore has many advantages [33]. In defect prediction, the connectivity among software entities can be determined by their similarity in metric values. Our key intuition for exploring spectral clustering is that defective entities tend to cluster around the same neighbourhoods (*i.e.*, clusters), as observed as well by by Menzies *et al.* [35] and Bettenburg *et al.* [4] in their work on local prediction models.

To evaluate the feasibility of using unsupervised classifiers for cross-project prediction, we perform an experiment using three publicly available datasets (*i.e.*, AEEEM [14], NASA [42], and PROMISE [29]) that include 26 projects in total. Our major findings are presented as follows:

- Unsupervised classifiers underperform supervised classifiers in general. However, a connectivity-based unsupervised classifier (*i.e.*, via spectral clustering) can compete with supervised classifiers.

- In the cross-project setting, our proposed spectral clustering based classifier achieves a median AUC value of 0.71, and ranks as one of the top classifiers.

- In the within-project setting, our spectral clustering based classifier ranks in the second tier, the same as three commonly used supervised classifiers (*i.e.*, logistic regression, logistic model tree, and naive Bayes). The random forest classifier appears in the first rank.

- A deeper investigation confirms our intuition that defective entities have significantly stronger connections with other defective entities than with clean entities.

As a summary, we propose to tackle cross-project predictions from a different perspective, *i.e.*, using a connectivity-based unsupervised classifier. Our spectral classifier is relatively simple (the implementation with 17 lines of R code is provided in Appendix A). Moreover, our spectral classifier is unsupervised, therefore it can be applied on a project without training data.

Paper organization. Section 2 presents the background and related work. In Section 3, we describe details of our spectral classifier. Experimental setup and case study results are presented in Sections 4 and 5, respectively. Section 6 closely examines the defect data in order to better understand the strong performance of our spectral classifier. The threats to validity of our work are discussed in Section 7. We conclude the paper and provide insights for future work in Section 8.

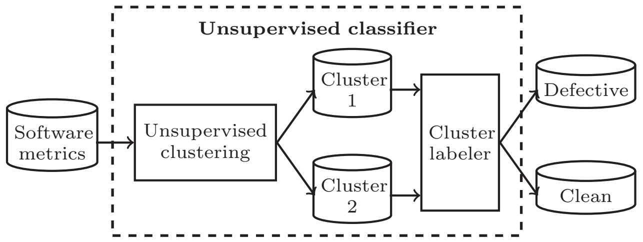

Figure 2: A typical process to do defect prediction using an unsupervised classifier.

2. BACKGROUND AND RELATED WORK

In this section, we first present the related work on cross-project defect prediction and unsupervised defect prediction. We then describe essential backgrounds on unsupervised classifiers.

2.1 Cross-project Defect Prediction

Prior attempts for cross-project defect prediction often resulted in disappointing performance (*e.g.*, [23, 48, 57, 66]). The major challenge is the heterogeneous distribution of metric values between the training projects and the target project [13, 41].

To reduce the heterogeneity in cross-project defect prediction, there are two major approaches:

(1) Using a model derived from training entities that are most similar to the entities in the target project (*e.g.*, [26, 27, 32, 35, 57]); For instance, He *et al.* [26, 27] propose to filter the training set based on distributional characteristics (*e.g.*, mean and standard deviation) of both the training and the target sets. Turhan *et al.* [57] propose to perform nearest neighbour filtering (NN-filtering).

(2) Transforming the metrics of both the training projects and the target project to increase their similarity (*e.g.*, [12, 34, 41, 58]). For instance, Ma *et al.* [34] apply Transfer Naive Bayes (TNB), Nam *et al.* [41] use Transfer component analysis (TCA), Chen *et al.* [12] use double transfer boosting (DTB) model. Our previous work [62] proposes a context-aware rank transformation approach that transforms software metrics based on project contexts (*e.g.*, programming language and project size), and builds a universal defect prediction model that achieves comparable performance as within-project models.

Another challenge in cross-project defect prediction is that the set of metrics is often different among projects. Nam and Kim [40] propose an approach to deal with heterogeneous sets of metrics between the training projects and the target project.

2.2 Unsupervised Defect Prediction

Unsupervised defect prediction predicts defect proneness without requiring access to training data. As illustrated in Figure 2, a typical process for predicting defects using an unsupervised classifier has two steps: 1) clustering software entities into k clusters (usually two clusters); and 2) labelling each cluster as a defective or clean cluster. However, there exists a limited number of studies in the literature on unsupervised defect prediction. One reason is that unsupervised classifiers usually underperform supervised ones (*e.g.*, random forest and logistic regression) in terms of their predictive power.

An initial attempt to use unsupervised defect classifiers is by Zhong *et al.* [65] who apply k-means and neural-gas clustering in defect prediction. Zhong *et al.* [65] observe that a neural-gas classifier outperforms k-means in terms of predictive power, but runs slower. However, their approach requires one to specify the expected number of clusters, and involves experts to determine which cluster contains defective entities (*i.e.*, label the cluster). Catal *et al.* [9] propose to use metric values to label the clusters. Bishnu and Bhattacherjee [5] propose to apply quad trees to initialize the cluster centres of k-means clustering. In addition to k-means clustering based classifiers, Abaei *et al.* [1] propose to use self-organizing maps (SOM) and Yang *et al.* [60] propose to apply the affinity propagation clustering algorithm. Recently, Nam and Kim [39] proposed to label the clusters using thresholds on selected metrics.

2.3 Background on Unsupervised Classifiers

Unsupervised classifiers make use of clustering methods. Clustering is a common way to explore groups of similar entities. Frequently applied clustering methods include hierarchical clustering and k-means. Hierarchical clustering produces clusters based on the structure of a similarity or dissimilarity matrix. K-means clustering is used to cluster high-dimensional data that are linearly separable [17].

In recent years, spectral clustering has become one of the most effective techniques for clustering [43, 59]. Unlike distance-based classifiers (*e.g.*, k-means clustering) that divide a data set based on Euclidean distance, spectral clustering partitions a data set based on the connectivity between its entities. Spectral clustering is performed on a graph consisting of nodes and edges. In the context of defect prediction, each node represents a software entity (*e.g.*, file or class). Each edge represents the connection between software entities, and its weight is measured by the similarity of metric values between its two ends.

Similarity definition. A widely used similarity is the dot product between vectors of two nodes i and j [2, 6, 16], as shown in Equation (1).

$$w_{ij} = \mathbf{x_i} \cdot \mathbf{x_j} = \sum_{k=1}^{m} a_{ik} a_{kj} \tag{1}$$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ denote the metric values of software entities i and j, respectively; a_{ik} is the value of the kth metric on the ith software entity, and m is the total number of metrics.

From the geometric perspective, the similarity w_{ij} can be interpreted as $\mathbf{x_i} \cdot \mathbf{x_j} = |\mathbf{x_i}||\mathbf{x_j}| cos\theta_{ij}$, where $|x_i|$ and $|x_j|$ are the norms, and θ_{ij} is the angle between two vectors. It is the length of the projection of one vector onto the other unit vector.

From a correlation perspective, the similarity w_{ij} is basically the unnormalized Pearson correlation coefficient [7] between nodes i and j. Each element in vector x_i represents a metric value. It is unnormalized, since it makes little sense to normalize the values across metrics belonging to the same software entity. The similarity w_{ij} can be positive, negative or zero. A positive value indicates a positive correlation between two software entities, and a negative value indicates a negative correlation. A value of zero indicates that there is no linear correlation. It is meaningless to study the self-circle of a software entity, therefore we set the self-similarity (*i.e.*, all w_{ii}) to zero.

Algorithm 1: Spectral clustering based classifier for defect prediction

```
Input: A matrix with rows as software entities and
    columns as metrics.
Output: A vector of defect proneness of all software
    entities.
1: Normalize software metrics using $z$-score.
2: Construct a weighted adjacency matrix $W$.
3: Calculate the Laplacian matrix $L_{sym}$.
4: Perform the eigendecomposition on $L_{sym}$.
5: Select the second smallest eigenvector $\mathbf{v_1}$.
6: Perform the bipartition on $v_1$ using zero.
7: Label each cluster as defective or clean.
```

Spectral clustering steps. A popular algorithm for spectral clustering is to minimize the normalized cut [53]. The normalized cut is a disassociation measure to describe the cost of cutting two partitions in a graph [53]. This algorithm partitions a graph into two subgraphs to gain high similarity within each subgraph while achieving low similarity across the two subgraphs.

The input for spectral clustering is a weighted adjacency matrix that stores the similarity between each pair of nodes in the graph. There are three major steps in the algorithm:

(1) Computing the Laplacian matrix from the weighted adjacency matrix, where the Laplacian matrix is a widely used matrix representation of a graph in graph theory;

(2) Performing an eigendecomposition on the Laplacian matrix;

(3) Selecting a threshold on the second smallest eigenvector to obtain the bipartitions of the graph.

3. OUR SPECTRAL CLASSIFIER

In this section, we describe details on our spectral clustering based classifier (see Algorithm 1). The *R* implementation of our spectral classifier consists of 17 lines of code (see Appendix A).

3.1 Preprocessing Software Metrics

Software metrics have varied scales. Hence, software metrics are often normalized before further processing [24, 41, 44]. For instance, Nam *et al.* [41] find that applying z-score to normalize software metrics can significantly improve the predictive power of defect prediction models. The advantage of z-score is that a normalized software metric has a mean value of zero and a variance of one.

Our spectral classifier uses the z-score for the normalization of each metric. We use $\mathbf{y_j}$ to denote a vector of values of the jth metric in a project. Then $\mathbf{y_j} = \{a_{1j}, \ldots, a_{nj}\}^T$, where n is the number of entities in the project, and a_{ij} is the value of the jth metric on the ith software entity. The vector $\mathbf{y_j}$ is normalized as $\hat{\mathbf{y}}_\mathbf{j} = \frac{\mathbf{y_j} - \bar{\mathbf{y}}_\mathbf{j}}{s_j}$, where $\bar{\mathbf{y}}_\mathbf{j}$ is the average value of $\mathbf{y_j}$ and s_j is the standard deviation of $\mathbf{y_j}$. This step corresponds to Line 1 in Algorithm 1.

3.2 Spectral Clustering

We now describe the three steps for spectral clustering.

Step 1. The first step is to calculate the Laplacian matrix L_{sym}. The symmetric Laplacian matrix L_{sym} is derived from the adjacency matrix W that stores the similarity between each pair of software entities. The adjacency matrix W is computed directly from the normalized software metrics (*i.e.*, Line 2 in Algorithm 1). In spectral clustering, there is usually an assumption that all values of the similarity are non-negative [36]. Hence, we set all negative w_{ij} to zero.

The symmetric Laplacian matrix L_{sym} is calculated using $L_{sym} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ (*i.e.*, Line 3 in Algorithm 1), where the matrix I is the unit matrix with size n, the matrix D is a diagonal matrix of row sums of W, and $D^{-\frac{1}{2}} = Diag(d_1^{-\frac{1}{2}}, \ldots, d_n^{-\frac{1}{2}})$, where $d_i^{-\frac{1}{2}} = (\sum_{j=1}^{n} w_{ij})^{-\frac{1}{2}}$.

Step 2. The second step is to perform the eigendecomposition on the symmetric Laplacian matrix L_{sym} (*i.e.*, Line 4 in Algorithm 1). Eigenvalues will always be ordered increasingly [33, 53]. We follow the normalized cut algorithm by Shi and Malik [53] and use the second smallest eigenvector for clustering (*i.e.*, Line 5 in Algorithm 1). We use v_1 to denote the second smallest eigenvector of L_{sym}.

Step 3. The third step is to separate all entities into two clusters. Shi and Malik [53] propose to apply a particular threshold, such as zero or median, on the second smallest eigenvector $\mathbf{v_1}$. If the median is used, then 50% of entities are predicted as defective. Inspecting 50% of entities requires significant effort. Hence, we adopt zero as the threshold value of $\mathbf{v_1}$ (*i.e.*, Line 6 in Algorithm 1) to create two non-overlapped clusters. We use v_{1i} to denote the ith value of $\mathbf{v_1}$, where $i \in \{1, \ldots, n\}$, and n is the total number of software entities in the given project. The value v_{1i} corresponds to the eigenvalue of the ith software entity. All entities with $v_{1i} > 0$ create a cluster called C_{pos}, and all entities with $v_{1i} < 0$ create the other cluster called C_{neg}. In the following subsection, we describe how to determine whether cluster C_{pos} contains defective entities, or cluster C_{neg} does.

3.3 Labelling Defective Cluster

The last step (*i.e.*, Line 7 in Algorithm 1) of applying the spectral clustering based classifier in defect prediction is to label the defective cluster.

We use $C_{defective}$ to denote the cluster that contains defective entities only, and use C_{clean} to represent the cluster that contains clean entities only.

To determine whether C_{pos} or C_{neg} is the defective cluster $C_{defective}$, we use the following heuristic: *For most metrics, software entities containing defects generally have larger values than software entities without defects.* This heuristic is based on our field's extensive empirical observations on the relationship between software metrics and defect proneness. For instance, Gaffney [19] find that larger files have a higher likelihood to experience defects than smaller files. Kitchenham *et al.* [30] report that more complex files are more likely to experience defects than files with lower complexity. Similar findings are also observed in many other studies (*e.g.*, [15, 25, 39]).

With this heuristic in mind, we use the average row sums of the normalized metrics of each cluster to determine which cluster is defective. The row sum is the sum of all metric values of the same entity. We compute the average row sum of all entities within each cluster (*i.e.*, either C_{pos} or C_{neg}). The cluster with larger average row sum is considered as the cluster containing defective entities. We label all entities within this cluster as defective (*i.e.*, $C_{defective}$), and all the remaining entities as clean (*i.e.*, C_{clean}).

However, the aforementioned heuristic does not necessarily work for all kinds of metrics. For instance, in the case where smaller values indicate less chance of defects, the aforementioned heuristic should be reversed. We suggest practitioners to derive the appropriate heuristic based on their set of metrics.

4. EXPERIMENT SETUP

In this section, we present the experimental setup to evaluate the performance of our spectral classifier.

4.1 Corpora

We examined data from three commonly studied datasets: AEEEM [14], NASA [42], and PROMISE [29]. The three datasets are publicly available and have been used extensively in defect prediction studies (*e.g.*, [20, 22, 35, 41]). A brief description on each dataset and our selected metrics are presented as follows.

D1. The AEEEM dataset was prepared by D'Ambros *et al.* [14] to compare the performance of different sets of metrics. Accordingly, the AEEEM dataset contains the most number of metrics. In particular, it has 61 metrics, including product, process, previous-defect metrics, and entropy-based metrics.

All projects in the AEEEM dataset have 61 identical software metrics. We use all 61 metrics in our study.

D2. The NASA dataset was collected by the NASA Metrics Data Program. Shepperd *et al.* [51] observe that the original NASA dataset contains many repeated and inconsistent data points, and they clean up the NASA dataset. In this study, we use the cleaned NASA dataset that is available in the PROMISE repository.

In the NASA dataset, projects do not share the same set of metrics. For instance, project KC3 has 39 metrics while project JM1 has 21 metrics. Since supervised classifiers require exact the same sets of metrics, we only select the 20 metrics that are common across all of the 11 studied NASA projects.

D3. The PROMISE dataset was prepared by Jureczko and Madeyski [29]. It contains open source Java projects and has object-oriented metrics.

In the PROMISE dataset, projects do not have the same set of metrics. Hence, we select the 20 metrics that are common across all of the 10 studied PROMISE projects.

In general, the selected projects have diverse size (*i.e.*, having 125 to 7,782 instances) and varied percentage of defective entities (*i.e.*, ranging from 2.1% to 63.6%). The summary of all selected projects is presented in Table 1. More details about these metrics can be found on the corresponding website of each dataset.

Table 1: An overview of the studied projects.

Dataset	Project	# of Entities	Defective (#)	Defective (%)
AEEEM	Eclipse JDT Core	997	206	20.7%
	Equinox	324	129	39.8%
	Apache Lucene	691	64	9.3%
	Mylyn	1,862	245	13.2%
	Eclipse PDE UI	1,497	209	14.0%
NASA	CM1	327	42	12.8%
	JM1	7,782	1,672	21.5%
	KC3	194	36	18.6%
	MC1	1,988	46	2.3%
	MC2	125	44	35.2%
	MW1	253	27	10.7%
	PC1	705	61	8.7%
	PC2	745	16	2.1%
	PC3	1,077	134	12.4%
	PC4	1,287	177	13.8%
	PC5	1,711	471	27.5%
PROMISE	Ant v1.7	745	166	22.3%
	Camel v1.6	965	188	19.5%
	Ivy v1.4	241	16	6.6%
	Jedit v4.0	306	75	24.5%
	Log4j v1.0	135	34	25.2%
	Lucene v2.4	340	203	59.7%
	POI v3.0	442	281	63.6%
	Tomcat v6.0	858	77	9.0%
	Xalan v2.6	885	411	46.4%
	Xerces v1.3	453	69	15.2%
	Average	1,036	196	18.9%

4.2 Performance Measure

There are many performance measures, such as precision, recall, accuracy, F-measure and the Area Under the receiver operating characteristic Curve (AUC). However, a cut-off value on the predicted probability of defect proneness is required when computing precision, recall, accuracy, and F-measure. The default cut-off is 0.5 which may not be the best cut-off value in practice [63]. On the other hand, the AUC value is independent of a cut-off value and is not impacted by the skewness of defect data. Lessmann *et al.* [31] and Ghotra *et al.* [20] suggest to use the AUC value for better cross-dataset comparability. Hence, we select the AUC measure as our performance measure.

When computing the AUC measure, a curve of the false positive rate is plotted against the true positive rate. Accordingly, the AUC value measures the probability that a randomly chosen defective entity ranks higher than a randomly chosen clean entity. An AUC value of 0.5 implies that a classifier is no better than random guessing. A larger AUC value indicates a better performance. In particular, Gorunescu [21] advises the following guideline to interpret the AUC value: 0.90 to 1.00 as excellent prediction, 0.80 to 0.90 as a good prediction, 0.70 to 0.80 as a fair prediction, 0.60 to 0.70 as a poor prediction, and 0.50 to 0.60 as a failed prediction.

4.3 Classifiers for Comparison

To find if our spectral classifier is applicable for defect prediction in a cross-project setting, we compare its performance with nine off-the-shelf classifiers. We not only select supervised classifiers, but also choose distance-based unsupervised classifiers.

For supervised classifiers, we select five classifiers that have been commonly applied to build defect prediction models. The five classifiers are random forest (RF), naive Bayes (NB), logistic regression (LR), decision tree (J48), and logistic model tree (LMT).

For distance-based unsupervised classifiers, we choose four classifiers that have been previously used in the defect prediction literature [9, 65]. The four classifiers include k-means clustering (KM), partition around medoids (PAM), fuzzy C-means (FCM), and neural-gas (NG). These classifiers are based on Euclidean distance, therefore employ a different clustering mechanism than spectral clustering (SC).

4.4 Scott-Knott Test

To compare the performance across the large number of datasets, we apply the Scott-Knott test [28] using the 95% confidence level (*i.e.*, $\alpha = 0.05$). The Scott-Knott test can overcome the issue of overlapping multiple comparisons that are obtained from other tests, such as the Mann-Whitney U test [52]. The Scott-Knott test has been used in defect prediction studies to compare the performance across different classifiers [20].

The Scott-Knott test recursively ranks the evaluated classifiers through hierarchical clustering analysis. In each iteration, the Scott-Knott test separates the evaluated classifiers into two groups based on the performance measure (*i.e.*, the AUC value). If the two groups have statistically significant difference in the AUC value, the Scott-Knott test executes again within each group. If no statistically distinct groups can be created, the Scott-Knott test terminates [20].

5. CASE STUDY RESULTS

In this section, we present our research questions, along with our motivation, approach, and findings.

RQ1. How does our spectral classifier perform in cross-project defect prediction?

Motivation. Unlike supervised classifiers, unsupervised classifiers do not have to deal with the challenge of heterogeneity between the training projects and the target project. While distance-based classifiers (*e.g.*, k-means clustering) underperform supervised classifiers, connectivity-based unsupervised classifiers have not been explored in our community. Hence, it is of significant interest to investigate if connectivity-based classifiers (particularly via spectral clustering) can provide comparable performance as supervised classifiers in the context of cross-project defect prediction.

Approach. To address this question, we need to get the performance of all studied classifiers for each project. For each classifier, all entities of the target project are used to obtain its performance.

Supervised classifiers require a training project. All supervised classifiers under study require the exact same set of metrics between the training and the target projects. As the three studied datasets (*i.e.*, AEEEM, NASA, and PROMISE) have different sets of metrics, we make cross-project defect prediction within the same dataset. For each target project, we select all other projects from the same dataset for training. For instance, if the target project is "Eclipse JDT Core", then each supervised classifier is used to build four models using each of the remaining projects within the same dataset (*i.e.*, "Equinox", "Apache Lucene", "Mylyn", and "Eclipse PDE UI"), respectively. We compute the average AUC values of these four models to measure the performance of the corresponding classifier on the target project, since it is unknown which model performs the best on the target project prior to the prediction.

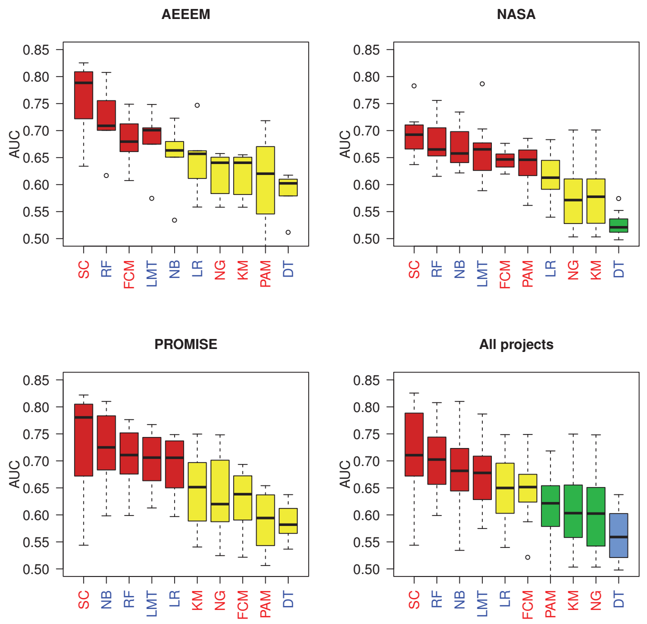

Figure 3: The boxplots of AUC values of all studied supervised (blue labels) and unsupervised (red labels) classifiers (for the abbreviations, see Section 4.3). Different colors represents different ranks (red > yellow > green > blue).

Unsupervised classifiers do not require training projects. We directly apply the studied unsupervised classifiers on the target project. When do clustering, we create k clusters. We set $k = 2$ for clustering, since this setting yields the best performance in defect prediction (*e.g.*, [20]). In the resulting two clusters, one cluster is labelled as defective, and the other cluster is labelled as clean, using the heuristic that is described in Section 3.3.

To compare the predictive power among all classifiers, we apply the Scott-Knott test with the 95% confidence level to rank all classifiers across projects within the same dataset. We examine the Scott-Knott ranks per dataset. Furthermore, we perform one large Scott-Knott run where we input all the AUC values for all the classifiers across all datasets.

Findings. Our spectral classifier achieves good results for defect prediction in the cross-project setting. In general, our spectral classifier significantly outperforms all other unsupervised classifiers, and it has slightly better performance than the best supervised classifier under study (*i.e.*, random forest).

Our spectral classifier ranks the first in all the three studied datasets. The colors in Figure 3 illustrate the ranks of all classifiers. The boxplots show the distribution of the AUC values of each classifier under study. Classifiers with boxplots of the same color are ranked at the same tier. The performances of classifiers in the same tier are not statistically distinct. Among all supervised and unsupervised classifiers, only two supervised classifiers (*i.e.*, random forest and logistic model tree) are in the same ranking tier across all the three datasets as our spectral classifier.

The exact AUC values of the top four classifiers (*i.e.*, our spectral classifier, random forest, naive Bayes, and logistic model tree) on each project are presented in Table 2. In particular, the median AUC values of the top four classifiers

Table 2: The AUC values of the top four classifiers in cross-project defect prediction (Bold font highlights the best performance).

Dataset	Project	SC	RF	NB	LMT
AEEEM	Eclipse JDT Core	**0.83**	0.81	0.68	0.75
	Equinox	**0.81**	0.70	0.66	0.71
	Apache Lucene	**0.79**	0.76	0.72	0.70
	Mylyn	**0.63**	0.62	0.53	0.57
	Eclipse PDE UI	**0.72**	0.71	0.65	0.67
NASA	CM1	**0.67**	0.66	0.66	0.62
	JM1	**0.66**	0.62	0.64	0.60
	KC3	0.64	**0.65**	0.62	0.63
	MC1	0.69	**0.71**	0.66	0.67
	MC2	**0.68**	0.62	0.64	0.59
	MW1	**0.70**	0.67	**0.70**	0.67
	PC1	0.71	**0.73**	0.70	0.70
	PC2	0.78	0.76	0.73	**0.79**
	PC3	**0.72**	0.70	0.70	0.68
	PC4	0.65	**0.67**	0.63	**0.67**
	PC5	**0.71**	0.66	0.66	0.63
PROMISE	Ant v1.7	**0.79**	0.75	0.77	0.75
	Camel v1.6	**0.62**	0.60	0.60	0.61
	Ivy v1.4	0.70	**0.71**	0.68	0.70
	Jedit v4.0	**0.79**	0.74	0.75	0.73
	Log4j v1.0	**0.82**	0.76	0.81	0.74
	Lucene v2.4	0.67	0.68	**0.69**	0.66
	POI v3.0	**0.82**	0.71	0.78	0.69
	Tomcat v6.0	**0.80**	0.78	**0.80**	0.77
	Xalan v2.6	0.54	**0.66**	0.60	0.62
	Xerces v1.3	**0.77**	0.69	0.70	0.71
Median		**0.71**	0.70	0.68	0.68

across all projects under study are: 0.71, 0.70, 0.68 and 0.68, respectively.

We observe that distance-based unsupervised classifiers (*e.g.*, *k*-means) do not perform as well as supervised classifiers. The poor performance of these distance-based classifiers may explain why unsupervised classifiers are not widely applied in defect prediction.

In summary, our results clearly show that applying connectivity-based unsupervised classification is a promising direction to tackle the heterogeneity challenge in cross-project defect prediction. Our connectivity-based unsupervised classifier is based on spectral clustering. We suspect that the success of spectral clustering is because defective entities are more similar to other defective entities than other clean entities in terms of the values of their various software metrics. Such intuition is supported through recent work by Menzies *et al.* [35] and Bettenburg *et al.* [4] on local defect prediction models.

Our spectral classifier performs the best among all studied classifiers that include five supervised and five unsupervised classifiers. Therefore, applying the connectivity-based unsupervised classification is a promising direction to tackle the challenge of heterogeneous data in cross-project defect prediction.

RQ2. Does our spectral classifier perform well in within-project defect prediction?

Motivation. In comparison to a cross-project setting, the chance of experiencing heterogeneous training and target data is much lower in a within-project setting. As unsupervised classifiers can save significant effort in defect data collection, we are interested to find if our connectivity-based

Table 4: The average AUC values of the top five classifiers in both cross-project (CP) and within-project settings (WP). The column "*diff*" shows the difference between cross-project models and within-project models.

Dataset	Project	RF			LR			SC			LMT			NB		
		CP	WP	*diff*	CP	WP	*diff*	CP	WP	*diff*	CP	WP	*diff*	CP	WP	*diff*
AEEEM	Eclipse JDT Core	0.81	0.87	*0.06*	0.75	0.79	*0.04*	0.83	0.83	*0*	0.75	0.82	*0.07*	0.68	0.74	*0.06*
	Equinox	0.70	0.84	*0.14*	0.61	0.64	*0.03*	0.81	0.80	*-0.01*	0.71	0.79	0.08	0.66	0.72	*0.06*
	Apache Lucene	0.76	0.81	*0.05*	0.66	0.63	*-0.03*	0.79	0.79	*0*	0.70	0.78	0.08	0.72	0.74	*0.02*
	Mylyn	0.62	0.82	*0.20*	0.56	0.79	*0.23*	0.63	0.63	*0*	0.57	0.78	0.21	0.53	0.65	*0.12*
	Eclipse PDE UI	0.71	0.78	*0.07*	0.66	0.73	*0.07*	0.72	0.72	*0*	0.67	0.75	0.08	0.65	0.67	*0.02*
NASA	CM1	0.66	0.68	*0.02*	0.61	0.74	*0.13*	0.67	0.67	*0*	0.62	0.65	0.03	0.66	0.67	*0.01*
	JM1	0.62	0.67	*0.05*	0.55	0.69	*0.14*	0.66	0.66	*0*	0.60	0.68	*0.08*	0.64	0.65	*0.01*
	KC3	0.65	0.71	*0.06*	0.59	0.64	*0.05*	0.64	0.64	*0*	0.63	0.63	*0*	0.62	0.65	*0.03*
	MC1	0.71	0.81	*0.10*	0.64	0.74	*0.10*	0.69	0.69	*0*	0.67	0.58	*-0.09*	0.66	0.68	*0.02*
	MC2	0.62	0.65	*0.03*	0.54	0.66	*0.12*	0.68	0.67	*-0.01*	0.59	0.67	*0.08*	0.64	0.66	*0.02*
	MW1	0.67	0.72	*0.05*	0.59	0.64	*0.05*	0.70	0.70	*0*	0.67	0.63	*-0.04*	0.70	0.71	*0.01*
	PC1	0.73	0.83	*0.10*	0.68	0.82	*0.14*	0.71	0.71	*0*	0.70	0.75	*0.05*	0.70	0.68	*-0.02*
	PC2	0.76	0.74	*-0.02*	0.65	0.66	*0.01*	0.78	0.78	*0*	0.79	0.53	*-0.26*	0.73	0.71	*-0.02*
	PC3	0.70	0.78	*0.08*	0.65	0.81	*0.16*	0.72	0.72	*0*	0.68	0.71	*0.03*	0.70	0.73	*0.03*
	PC4	0.67	0.91	*0.24*	0.63	0.88	*0.25*	0.65	0.65	*0*	0.67	0.88	*0.21*	0.63	0.74	*0.11*
	PC5	0.66	0.76	*0.10*	0.60	0.73	*0.13*	0.71	0.71	*0*	0.63	0.72	*0.09*	0.66	0.68	*0.02*
PROMISE	Ant v1.7	0.75	0.82	*0.07*	0.74	0.80	*0.06*	0.79	0.79	*0*	0.75	0.81	0.06	0.77	0.78	*0.01*
	Camel v1.6	0.60	0.71	*0.11*	0.61	0.73	*0.12*	0.62	0.62	*0*	0.61	0.69	*0.08*	0.60	0.67	*0.07*
	Ivy v1.4	0.71	0.67	*-0.04*	0.69	0.55	*-0.14*	0.70	0.70	*0*	0.70	0.57	*-0.13*	0.68	0.64	*-0.04*
	Jedit v4.0	0.74	0.80	*0.06*	0.72	0.77	*0.05*	0.79	0.78	*-0.01*	0.73	0.78	*0.05*	0.75	0.75	*0*
	Log4j v1.0	0.76	0.80	*0.04*	0.74	0.69	*-0.05*	0.82	0.78	*-0.04*	0.74	0.81	*0.07*	0.81	0.81	*0*
	Lucene v2.4	0.68	0.77	*0.09*	0.65	0.75	*0.10*	0.67	0.66	*-0.01*	0.66	0.75	*0.09*	0.69	0.73	*0.04*
	POI v3.0	0.71	0.88	*0.17*	0.70	0.83	*0.13*	0.82	0.81	*-0.01*	0.69	0.83	*0.14*	0.78	0.82	*0.04*
	Tomcat v6.0	0.78	0.81	*0.03*	0.75	0.82	*0.07*	0.80	0.80	*0*	0.77	0.81	*0.04*	0.80	0.80	*0*
	Xalan v2.6	0.66	0.85	*0.19*	0.60	0.81	*0.21*	0.54	0.54	*0*	0.62	0.81	*0.19*	0.60	0.76	*0.16*
	Xerces v1.3	0.69	0.83	*0.14*	0.72	0.77	*0.05*	0.77	0.77	*0*	0.71	0.74	*0.03*	0.70	0.79	*0.09*
Median		**0.70**	**0.80**	*0.07*	**0.65**	**0.74**	*0.09*	**0.71**	**0.71**	*0*	**0.68**	**0.75**	*0.07*	**0.68**	**0.72**	*0.02*

Table 3: Ranks of all studied classifiers for within-project defect prediction based on 1,000 evaluations.

Overall ranks	Classifier	Median rank	Average rank	Standard deviation
1	RF	1	1.42	0.64
2	LR	2	3.19	2.15
	SC	3	3.35	1.67
	LMT	3	3.42	1.94
	NB	3.5	3.54	1.27
3	FCM	6	5.96	1.08
4	PAM	6.5	6.73	1.69
	NG	7	6.85	1.67
	DT	7	6.89	1.56
	KM	7.5	7.35	1.55

unsupervised classifier (*i.e.*, the proposed spectral classifier) can still compete with supervised classifiers in a within-project setting.

Approach. To evaluate the performance of supervised classifiers in a within-project setting, the essential step is to separate all entities of a project into two sets. One set is for training a model and the other one is the target set to apply the model. Both supervised and unsupervised classifiers are applied on the same target set of entities. The only difference is that supervised classifiers require an additional step to build a model from the training set of entities.

To create the training and target sets, we apply a two-fold cross validation (*i.e.*, a 50:50 random split) that has been previously applied in the defect prediction literature [39, 47]. For a 50:50 random split, each classifier is evaluated twice: 1) the first half is used as the training data while the other half is used as the target data; and 2) the second half is used as the training data while the first half is used as the target data. To deal with the randomness of sampling, we repeat the random splits for 500 times (*i.e.*, 500 times of two-fold cross validation). In total, 1,000 evaluations are performed for each classifier on each project. To get the performance of each classifier on each project, we compute the average AUC value of the total 1,000 evaluations.

To find statistically distinct ranks of all classifiers, we follow the approach of Ghotra *et al.* [20] and perform a double Scott-Knott test. The double Scott-Knott test ensures a robust ranking of all classifiers across projects, regardless of their exact AUC values. The first Scott-Knott test is performed on each individual project to rank all classifiers based on their AUC values for that particular project. The obtained ranks are used in the second run of the Scott-Knott test to yield a global ranking of all classifiers across all studied projects.

Findings. Generally speaking, in a within-project setting, supervised classifiers outperform unsupervised classifiers. There is only one unsupervised classifier (*i.e.*, our spectral classifier) among the top five classifiers.

The detailed rankings are presented in Table 3, including the global ranks of all classifiers across all projects, and the statistics (*i.e.*, median, average, and standard deviation) of the ranks of each classifier as obtained in the first Scott-Knott test on the results of 1,000 evaluations. In particular, our spectral classifier has a median rank of 3, and is ranked in the same tier as three widely used classifiers, *i.e.*, logistic regression, logistic model tree, and naive Bayes.

The actual AUC values of the top five classifiers (*i.e.*, random forest, logistic regression, our spectral classifier, logistic model tree, and naive Bayes) on each project are presented in Table 4. The AUC values in both cross-project and within-project settings are shown, as well as their difference (*i.e.*, the AUC value in a within-project setting minus the AUC value in a cross-project setting).

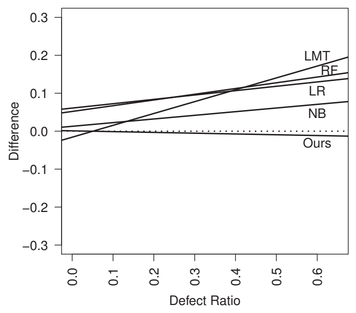

Figure 4: The regression lines of the performance difference of the top five classifiers between a within-project setting and a cross-project setting over the ratio of defects of each project. (The dotted line is the horizontal base line.)

Our spectral classifier achieves almost the same predictive power between cross-project and within-project settings across all projects under study, as shown in Table 4. The size of the target project in a within-project setting is only half of that in a cross-project setting, highlighting that our spectral classifier tends to be robust when the size of the target project changes.

A within-project model can sometimes significantly underperform a cross-project model, although a within-project model generally outperforms a cross-project model. For example, looking at Table 4, and for project "Ivy v1.4", the top four supervised classifiers experience a downgraded performance when changing from a cross-project setting to a within-project setting. In particular, the random forest classifier achieves an AUC value of 0.71 in a cross-project setting, but yields a lower AUC value of 0.67 in a within-project setting. We conjecture that the decrease in performance when changing to a within-project setting is caused by the low ratio of defects (*i.e.*, the low percentage of defective entities) in the target project. For instance, project "Ivy v1.4" has a ratio of defects of 6.6% with only 16 defective entities. Similar observations are noted in other projects, such as "Apache Lucene" and "PC2".

Supervised classifiers tend to experience a performance decrease, if the ratio of defects becomes lower. To illustrate the relationships between the performance of each classifier and the ratio of defects, we plot regression lines of the performance difference of the top five classifiers over the ratio of defects in Figure 4. In comparison to supervised classifiers, our spectral clustering based classifier is more robust across a varying ratio of defects. One possible reason is that supervised classifiers experience a significant class-imbalance problem on these projects, while our spectral classifier is an unsupervised approach, therefore has no issue of class-imbalance. We conjecture that, for projects with a low ratio of defects, our spectral clustering based classifier may be more suitable than the supervised classifiers.

*In a within-project setting, our spectral classifier ranks in the second tier with only random forest ranking in the first tier. However, our spectral classifier may be more suitable for projects with heavily imbalanced (*i.e.*, very low percentage of defective entities) defect data.*

6. WHY DOES IT WORK?

In this section, we present an in-depth analysis to understand why our spectral classifier, which is a connectivity-based classifier, achieves good results in defect prediction. As aforementioned, spectral clustering separates all entities in a project based on the connections among entities. We conjecture that software entities may reside within two "social network"-like communities: 1) one community is formulated by defective entities; and 2) the other one is established by clean entities.

6.1 Essential Definitions

Community definition. We define a community as a set of members (*i.e.*, software entities) that have much stronger connections with each other than with members from other communities. A connection is basically an edge in a graph, as mentioned in Section 2.3. We define an edge between entities i and j using Equation (2).

$$e_{ij} = \mathbb{1}(w_{ij}) \tag{2}$$

where $\mathbb{1}(w_{ij}) = 1$ *if* $w_{ij} > 0$, *and* $\mathbb{1}(w_{ij}) = 0$ *otherwise.*

As described in Section 2.3, w_{ij} represents the similarity or the correlation between entities i and j. Hence, e_{ij} equals to 1, if there is a positive correlation between entities i and j. We denote the set of all edges as E, then $E = \{e_{ij}\}$.

We construct the community as follows. For each project, we partition the entities into two sets based on their defect proneness. We use V_d to denote the set of actual defective entities, and V_c to denote the set of actual clean entities. A software entity can be either defective or clean. Hence, there is no overlap between V_d and V_c, and the union of V_d and V_c contains all entities within the same project.

Connectivity measurement. We define deg^{dd}, the total degree of all defective entities, using Equation (3). We define deg^{dd}, the total degree of all clean entities, using Equation (4). Similarly, we define deg^{cd}, the total number of edges between each pair of defective and clean entities, using Equation (5).

$$deg^{dd} = \sum_{i \in V_d} \sum_{j \in V_d} e_{ij}, \quad j \neq i \tag{3}$$

$$deg^{cc} = \sum_{i \in V_c} \sum_{j \in V_c} e_{ij}, \quad j \neq i \tag{4}$$

$$deg^{cd} = \sum_{i \in V_c} \sum_{j \in V_d} e_{ij} \tag{5}$$

To measure the connectivity among entities within V_d or V_c, or between V_d and V_c, we further define the ratio of edges (*i.e.*, connections) as follows.

$$\phi^{dd} = \frac{deg^{dd}}{|V_d|(|V_d| - 1)} \tag{6}$$

$$\phi^{cc} = \frac{deg^{cc}}{|V_c|(|V_c| - 1)} \tag{7}$$

$$\phi^{cd} = \frac{deg^{cd}}{|V_c||V_d|} \tag{8}$$

To illustrate the computation, we present an example in Figure 5. There are three defective and four clean entities. Each defective entity has connections to all other two defective entities. Hence, $deg^{dd} = 2+2+2 = 6$ and $\phi^{dd} = \frac{6}{3\times 2} = 1.000$. Similarly, we can get $deg^{cc} = 2+2+2+2 = 8$ and

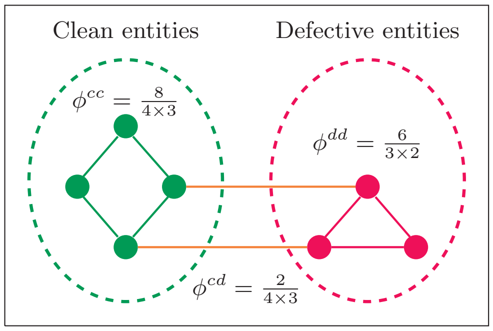
Clean entities
Defective entities
$\phi^{cc} = \frac{8}{4\times3}$
$\phi^{dd} = \frac{6}{3\times2}$
$\phi^{cd} = \frac{2}{4\times3}$

Figure 5: Illustrating example of computing the ratio of edges (*i.e.*, ϕ^{dd}, ϕ^{cc}, ϕ^{cd}).

$\phi^{cc} = \frac{8}{4\times3} = 0.667$, and $deg^{cd} = 2$ and $\phi^{cd} = \frac{2}{4\times3} = 0.167$.

6.2 Hypotheses

For each project, we compute the ratios ϕ^{dd}, ϕ^{cc}, and ϕ^{cd} based on the actual defect proneness. To compare the connectivity among entities across all projects under study, we test the following hypotheses:

$H0_1$: there is no difference in the ratios of connections from defective entities to other defective entities (ϕ^{dd}) and clean entities ϕ^{cd}.

$H0_2$: there is no difference in the ratios of connections from clean entities to other clean entities (ϕ^{cc}) and defective entities ϕ^{cd}.

Hypotheses $H0_1$ and $H0_2$ are two sided and paired, since each project has three unique values: ϕ^{dd}, ϕ^{cc}, and ϕ^{cd}. To test the hypotheses, we apply paired Mann-Whitney U test using the 95% confidence level (*i.e.*, $\alpha < 0.05$). We further compute the Cliff's δ [50] as the effect size to quantify the difference. Both the Mann-Whitney U test and the Cliff's δ are non-parametric statistical methods, and do not require a particular distribution of assessed variables. An effect size is large, if Cliff's $|\delta| \geq 0.474$ [50].

6.3 Empirical Findings

We observe that in general **the connections between defective and clean entities are weaker than the connection among defective entities and the connections among clean entities**. Table 5 presents the detailed values of our three measures (*i.e.*, ϕ^{cc}, ϕ^{cd}, and ϕ^{dd}) for each project. For instance, in project "Eclipse JDT Core", the ratio of connections among defective entities $\phi^{dd} = 0.564$. The ratio of connections among clean entities $\phi^{cc} = 0.614$. These two ratios are significantly greater than the ratio of connections between clean and defective entities which is $\phi^{cd} = 0.365$.

Defective entities have significantly stronger connections with other defective entities than with clean entities. The p-value of the Mann-Whitney U test is is 4.20e-05, when comparing the ratios ϕ^{dd} and ϕ^{cd} across all projects. The difference is large, as the corresponding Cliff's $|\delta|$ is 0.654>0.474.

Similarly, clean entities have significantly stronger connections with other clean entities than with defective entities (*i.e.*, the p-value of the Mann-Whitney U test is 8.55e-06). The difference is also large, as Cliff's $|\delta|$ is 0.769>0.474.

As a summary, our observation indicates that either defective or clean entities are similar in terms of metric values, but defective and clean entities are less likely to experience similar metric values. In other words, there roughly exist two communities based on defect proneness. Entities within the same community have stronger connections than cross communities. This may be the reason as to why the proposed connectivity-based unsupervised classifier (*i.e.*, our spectral classifier) achieves empirically good results in defect prediction.

Table 5: The values of ϕ^{cc}, ϕ^{cd}, and ϕ^{dd} for each project. (Bold font highlights the minimum value per row).

Dataset	Project	ϕ^{cc}	ϕ^{cd}	ϕ^{dd}
AEEEM	Eclipse JDT Core	0.614	**0.365**	0.564
	Equinox	0.694	**0.443**	0.470
	Apache Lucene	0.554	**0.374**	0.556
	Mylyn	0.575	**0.442**	0.489
	Eclipse PDE UI	0.576	**0.426**	0.512
NASA	CM1	0.616	**0.497**	0.502
	JM1	0.628	**0.515**	0.519
	KC3	0.585	0.498	**0.477**
	MC1	0.572	**0.437**	0.540
	MC2	0.646	**0.495**	0.496
	MW1	0.551	**0.439**	0.546
	PC1	0.594	**0.470**	0.556
	PC2	0.594	**0.442**	0.602
	PC3	0.586	**0.450**	0.593
	PC4	0.583	**0.489**	0.577
	PC5	0.714	**0.574**	0.588
PROMISE	Ant v1.7	0.522	**0.398**	0.606
	Camel v1.6	0.487	**0.455**	0.481
	Ivy v1.4	0.482	**0.417**	0.508
	Jedit v4.0	0.504	**0.402**	0.536
	Log4j v1.0	0.538	**0.368**	0.535
	Lucene v2.4	0.542	**0.438**	0.459
	POI v3.0	0.605	**0.390**	0.537
	Tomcat v6.0	0.485	**0.380**	0.630
	Xalan v2.6	0.540	0.439	**0.438**
	Xerces v1.3	0.488	**0.394**	0.504
	Median	0.576	**0.439**	0.536

There roughly exist two communities of entities: a defective community and a clean community of entities. Within-community connections are significantly stronger than cross-community connections.

7. THREATS TO VALIDITY

In this section, we describe the threats to validity of our study under common guidelines by Yin [61].

Threats to conclusion validity concern the relation between the treatment and the outcome. The major threat is that we only compare our approach with off-the-shelf classifiers. Future work should explore state-of-the-art cross project defect classifiers. Unfortunately the implementation of such specialized classifiers are rarely available and often require a considerable amount of setup – making them hard for practitioners to easily adopt. Hence we chose to compare against commonly used and readily available classifiers.

Threats to internal validity concern our selection of subject systems and analysis methods. We select 26 projects that have been commonly used in the defect prediction literature. These projects are from different domains, include both open source and industrial projects, and have different sets of metrics. However, evaluating our approach on a large scale of projects is always desirable. Nevertheless our findings raise a very poignant point about the importance of exploring connectivity-based unsupervised classifiers in future defection prediction research. Moreover, the simplicity of our approach makes exploring it in future studies as a very lightweight and simple step to perform.

Threats to external validity concern the possibility to generalize our results. Our approach only requires software metrics that can be computed in a standard way by publicly available tools. However, only metrics that are collected in the three data sets are applied in our experiments. Replication studies using different sets of metrics may prove fruitful.

Threats to reliability validity concern the possibility of replicating this study. All the three studied data sets are publicly available. Moreover, the *R* implementation of our approach is provided in Appendix A.

8. CONCLUSION

As new or small projects do not have sufficient training data, cross-project defect prediction has attracted great interest from both researchers and practitioners (*e.g.*, [26, 27, 32, 34, 35, 41, 57, 58]). The major challenge in cross-project defect prediction is the heterogeneity between the training projects and the target project (*e.g.*, different distributions of metric values [13, 41] and different sets of metrics [40]).

This study brings a new insight to tackle this challenge using connectivity-based unsupervised classifiers. Unsupervised classifiers do not require any training data, and therefore have no issue of heterogeneity. Apart from distance-based unsupervised classifiers (*e.g.*, *k*-means clustering), the connectivity-based unsupervised classifiers assume that defective entities tend to cluster around the same area, a similar intuition as the recent work on local prediction models by Menzies *et al.* [35] and Bettenburg *et al.* [4].

To evaluate the performance of our proposed spectral classifier, we perform experiments using 26 projects from three publicly available datasets (*i.e.*, AEEEM [14], NASA [42], and PROMISE [29]). The results show that the proposed connectivity-based unsupervised classifier (*i.e.*, our spectral classifier) achieves impressive performance in a cross-project setting. Specifically, our spectral classifier ranks as one of the top classifiers among five supervised classifiers (*e.g.*, random forest) and five unsupervised classifiers (*e.g.*, *k*-means). In a within-project setting, our spectral classifier ranks in the second tier, the same as three widely used supervised classifiers (*e.g.*, logistic regression, logistic model tree, and naive Bayes) with random forest as the only classifier in the first tier.

As a summary, our contributions are as follows:

- **Demonstrating that connectivity-based unsupervised classification (particularly via spectral clustering) performs well in a cross-project setting.** Our experiments show that our connectivity-based unsupervised classifier (via spectral clustering) can achieve similar or better performance than several commonly used supervised and unsupervised classifiers. We believe that unsupervised classification holds great promise in defect prediction, especially in a cross-project setting and for highly skewed within-project settings.
- **Demonstrating the existence of two (defective and clean) separated communities of software entities based on the connectivity between the entities in each community.** We believe that this observation highlights the importance for the software engineering research community to explore more advanced techniques for unsupervised defect prediction instead of current strong reliance on supervised classifiers.

APPENDIX

A. R IMPLEMENTATION OF OUR SPECTRAL CLASSIFIER

In Listing 1, we present the *R* implementation of our spectral classifier.

Listing 1: *R* implementation of our approach.

```
spectral_clustering_based_classifier <- function(A) {
  # Normalize software metrics.
  normA = apply(A, 2, function(x){(x-mean(x))/sd(x)})
  # Construct the weighted adjacency matrix W.
  W = normA %*% t(normA)
  # Set all negative values to zero.
  W[W<0] = 0
  # Set the self-similarity to zero.
  W = W - diag(diag(W))
  # Construct the symmetric Laplacian matrix Lsym.
  Dnsqrt = diag(1/sqrt(rowSums(W)))
  I = diag(rep(1, nrow(W)))
  Lsym = I - Dnsqrt %*% W %*% Dnsqrt
  # Perform the eigendecomposition.
  ret_egn = eigen(Lsym, symmetric=TRUE)
  # Pick up the second smallest eigenvector.
  v1 = Dnsqrt %*% ret_egn$vectors[, nrow(W)-1]
  v1 = v1 / sqrt(sum(v1^2))
  # Divide the data set into two clusters.
  defect_proneness = (v1>0)
  # Label the defective and clean clusters.
  rs = rowSums(normA)
  if(mean(rs[v1>0])<mean(rs[v1<0]))
      defect_proneness = (v1<0)
  # Return the defect proneness.
  defect_proneness
}
```

References

[1] G. Abaei, Z. Rezaei, and A. Selamat. Fault prediction by utilizing self-organizing Map and Threshold. In *2013 IEEE International Conference on Control System, Computing and Engineering*, pages 465–470. IEEE, Nov. 2013.

[2] C. C. Aggarwal, editor. *Data Classification: Algorithms and Applications*. CRC Press, 2014.

[3] O. F. Arar and K. Ayan. Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33:263–277, Aug. 2015.

[4] N. Bettenburg, M. Nagappan, and A. E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 60–69, June 2012.

[5] P. Bishnu and V. Bhattacherjee. Software fault prediction using quad tree-based k-means clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1146–1150, June 2012.

[6] P. Blanchard and D. Volchenkov. *Mathematical Analysis of Urban Spatial Networks*. Springer Berlin Heidelberg, Heidelberg, Germany, 2009.

[7] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social Networks*, 21(4):375 – 395, 2000.

[8] L. C. Briand, W. L. Melo, and J. Wüst. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28(7):706–720, July 2002.

[9] C. Catal, U. Sevim, and B. Diri. Metrics-driven software quality prediction without prior fault data. In S.-I. Ao and L. Gelman, editors, *Electronic Engineering and Computing Technology*, volume 60 of *Lecture Notes in Electrical Engineering*, pages 189–199. Springer Netherlands, 2010.

[10] C. Catal, U. Sevim, and B. Diri. Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. *Expert Systems with Applications*, 38(3):2347–2353, Mar. 2011.

[11] E. Ceylan, F. Kutlubay, and A. Bener. Software Defect Identification Using Machine Learning Techniques. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 240–247. IEEE, 2006.

[12] L. Chen, B. Fang, Z. Shang, and Y. Tang. Negative samples reduction in cross-company software defects prediction. *Information and Software Technology*, 62:67–77, June 2015.

[13] A. Cruz and K. Ochimizu. Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 460–463, Oct. 2009.

[14] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, pages 31–41. IEEE CS Press, May 2010.

[15] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, Aug. 2012.

[16] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[17] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556. ACM, 2004.

[18] M. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 38(2.3):258–287, 1999.

[19] J. E. Gaffney. Estimating the number of faults in code. *IEEE Transactions on Software Engineering*, SE-10(4):459–464, July 1984.

[20] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th IEEE International Conference on Software Engineering*, volume 1, pages 789–800, May 2015.

[21] F. Gorunescu. *Data mining concepts, models and techniques.* Springer, Berlin, 2011.

[22] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the nasa metrics data program data sets for automated software defect prediction. In *Proceedings of the 15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*, pages 96–103, April 2011.

[23] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov. 2012.

[24] J. Han, M. Kamber, and J. Pei. *Data Mining: concepts and techniques.* Morgan Kaufmann, Boston, 3 edition, 2012.

[25] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st IEEE International Conference on Software Engineering*, pages 78 –88, 2009.

[26] Z. He, F. Peters, T. Menzies, and Y. Yang. Learning from open-source projects: An empirical study on defect prediction. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 45–54, Oct. 2013.

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

[28] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman. Scottknott: A package for performing the scott-knott clustering algorithm in r. *Trends in Applied and Computational Mathematics*, 15(1):3–17, 2014.

[29] 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*, pages 9:1–9:10, 2010.

[30] B. Kitchenham, L. Pickard, and S. Linkman. An evaluation of some design metrics. *Software Engineering Journal*, 5(1):50–58, Jan 1990.

[31] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering (TSE)*, 34(4):485–496, 2008.

[32] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, 19(2):201–230, June 2012.

[33] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007.

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

[35] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 343–351. IEEE Computer Society, 2011.

[36] B. Mohar. The laplacian spectrum of graphs. In *Graph Theory, Combinatorics, and Applications*, pages 871–898. Wiley, 1991.

[37] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering*, pages 181–190. ACM, May 2008.

[38] R. Mullen and S. Gokhale. Software Defect Rediscoveries: A Discrete Lognormal Model. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pages 203–212. IEEE, 2005.

[39] J. Nam and S. Kim. Clami: Defect prediction on unlabeled datasets. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, 2015.

[40] J. Nam and S. Kim. Heterogeneous defect prediction. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ESEC/FSE '15, 2015.

[41] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391. IEEE Press, 2013.

[42] NASA. Metrics Data Program. http://openscience.us/repo/defect/mccabehalsted, 2015. [Online; accessed 25-August-2015].

[43] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[44] M. Ohlsson and P. Runeson. Experience from replicating empirical studies on prediction models. *Proceedings of the 8th IEEE Symposium on Software Metrics*, pages 217–226, 2002.

[45] T. Ostrand and E. Weyuker. On the automation of software fault prediction. In *Testing: Academic and Industrial Conference - Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings*, pages 41–48, Aug. 2006.

[46] L. Pelayo and S. Dick. Applying novel resampling strategies to software defect prediction. In *Annual Conference of the North American Fuzzy Information processing Society*, NAFIPS '07, pages 69–72, June 2007.

[47] M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM.

[48] R. Premraj and K. Herzig. Network versus code metrics to predict defects: A replication study. In *2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 215–224, 2011.

[49] 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. *Knowledge-based Software Engineering, JCKBSE 2014*, 466:270–285, 2014.

[50] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In *Annual Meeting of the Florida Association of Institutional Research*, pages 1–33, Feb. 2006.

[51] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, Sept 2013.

[52] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures, Fourth Edition*. Chapman & Hall/CRC, Jan. 2007.

[53] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug. 2000.

[54] F. Shull, V. Basili, B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. In *Proceedings of the 8th IEEE Symposium on Software Metrics*, pages 249–258, 2002.

[55] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report Planning Report 02-3, National Institute of Standards and Technology, May 2002.

[56] A. Tosun, A. Bener, B. Turhan, and T. Menzies. Practical considerations in deploying statistical methods for defect prediction: A case study within the Turkish telecommunications industry. *Information and Software Technology*, 52(11):1242–1257, Nov. 2010.

[57] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5):540–578, Oct. 2009.

[58] S. Watanabe, H. Kaiya, and K. Kaijiri. Adapting a fault prediction model to allow inter languagereuse. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, PROMISE '08, pages 19–24. ACM, 2008.

[59] A. R. Webb and K. D. Copsey. *Statistical Pattern Recognition, Third Edition*. John Wiley & Sons, Inc., 2011.

[60] B. Yang, Q. Yin, S. Xu, and P. Guo. Software quality prediction using affinity propagation algorithm. In *Proceedings of the 2008 IEEE International Joint Conference on Neural Networks. IJCNN 2008*, pages 1891–1896, June 2008.

[61] R. K. Yin. *Case Study Research: Design and Methods - Third Edition*. SAGE Publications, 3 edition, 2002.

[62] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14, pages 41–50, Piscataway, NJ, USA, 2014. IEEE Press.

[63] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, pages 1–39, 2015.

[64] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan. How does context affect the distribution of software maintainability metrics? In *Proceedings of the 29th IEEE International Conference on Software Maintainability*, ICSM '13, pages 350 – 359, 2013.

[65] S. Zhong, T. Khoshgoftaar, and N. Seliya. Unsupervised learning for expert-based software quality estimation. In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering*, pages 149–155, Mar. 2004.

[66] 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 the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM.