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

Feng Zhang\textsuperscript{1}, Quan Zheng\textsuperscript{1}, Ying Zou\textsuperscript{2}, and Ahmed E. Hassan\textsuperscript{1}

\textsuperscript{1}School of Computing, Queen’s University, Canada
\textsuperscript{2}Department of Electrical and Computer Engineering, Queen’s University, Canada
\textsuperscript{1}\{feng, quan, ahmed\}@cs.queensu.ca, \textsuperscript{2}ying.zou@queensu.ca

\section*{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.

\section*{CCS Concepts}
\begin{itemize}
\item Software and its engineering \rightarrow Software verification and validation; Software defect analysis;
\end{itemize}

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

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

\section*{Keywords}
defect prediction, heterogeneity, cross-project, unsupervised, spectral clustering, graph mining

\section*{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 chal-
challenges 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 neighborhoods (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.
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 Bhattacharjee \[5\] 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).

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

where \( x_i \) and \( x_j \) denote the metric values of software entities \( i \) and \( j \), respectively; \( a_{ik} \) is the value of the \( k \)th metric on the \( i \)th software entity, and \( m \) is the total number of metrics.

From the geometric perspective, the similarity \( w_{ij} \) can be interpreted as \( x_i \cdot x_j = ||x_i|| ||x_j|| \cos \theta_{ij} \), where \( ||x_i|| \) and \( ||x_j|| \) are the norms, and \( \theta_{ij} \) 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 \( x \) 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.

**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 \( y_j \) to denote a vector of values of the \( j \)th metric in a project. Then \( y_j = \{a_{1j}, \ldots, a_{nj}\}^T \), where \( n \) is the number of entities in the project, and \( a_{ij} \) is the value of the \( j \)th metric on the \( i \)th software entity. The vector \( y_j \) is normalized as \( y_j^{\prime} = \frac{y_j - \bar{y}_j}{s_j} \), where \( \bar{y}_j \) is the average value of \( y_j \) and \( s_j \) is the standard deviation of \( y_j \). 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 $D^{-\frac{1}{2}} = \text{Diag}(d_1^{-\frac{1}{2}}, \ldots, d_n^{-\frac{1}{2}})$, where $d_i^{-\frac{1}{2}} = \left(\sum_{j=1}^{n} w_{ij}\right)^{-\frac{1}{2}}$.

**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 $v_1$. 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 $v_1$ (i.e., Line 6 in Algorithm 1) to create two non-overlapped clusters. We use $v_{i+1}$ to denote the $i$th value of $v_1$, where $i \in \{1, \ldots, n\}$, and $n$ is the total number of software entities in the given project. The value $v_{i+1}$ corresponds to the eigenvalue of the $i$th software entity. All entities with $v_{i+1} > 0$ create a cluster called $C_{pos}$, and all entities with $v_{i+1} < 0$ 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 $C_{\text{defective}}$ to denote the cluster that contains defective entities only, and use $C_{\text{clean}}$ to represent the cluster that contains clean entities only.

To determine whether $C_{pos}$ or $C_{neg}$ is the defective cluster $C_{\text{defective}}$, 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., $C_{\text{defective}}$), and all the remaining entities as clean (i.e., $C_{\text{clean}}$).

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.

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

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 Ghota 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 do not 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 they 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.
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 (\( e.g., \) 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 (for the abbreviations, see Section 4.3). Different colors represents different ranks (red > yellow > green > blue).

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.

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

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

Median: 0.71, 0.70, 0.68, 0.68

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
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]. The actual AUC values of the top five classifiers, ran-

**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).

---

### 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Project</th>
<th>CP</th>
<th>WP</th>
<th>diff</th>
<th>WP</th>
<th>diff</th>
<th>CP</th>
<th>WP</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEEEM</td>
<td>Eclipse JDT Core</td>
<td>0.81</td>
<td>0.87</td>
<td>0.06</td>
<td>0.75</td>
<td>0.79</td>
<td>0.04</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Equinox</td>
<td>0.70</td>
<td>0.84</td>
<td>0.14</td>
<td>0.61</td>
<td>0.64</td>
<td>0.03</td>
<td>0.81</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Apache Lucene</td>
<td>0.76</td>
<td>0.81</td>
<td>0.05</td>
<td>0.66</td>
<td>0.63</td>
<td>0.00</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>Mylyn</td>
<td>0.62</td>
<td>0.82</td>
<td>0.20</td>
<td>0.56</td>
<td>0.79</td>
<td>0.25</td>
<td>0.63</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>Eclipse PDE U1</td>
<td>0.71</td>
<td>0.78</td>
<td>0.07</td>
<td>0.66</td>
<td>0.73</td>
<td>0.00</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>NASA</td>
<td>CM1</td>
<td>0.66</td>
<td>0.68</td>
<td>0.02</td>
<td>0.61</td>
<td>0.74</td>
<td>0.13</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>JM1</td>
<td>0.62</td>
<td>0.67</td>
<td>0.05</td>
<td>0.55</td>
<td>0.69</td>
<td>0.14</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>KC3</td>
<td>0.65</td>
<td>0.71</td>
<td>0.06</td>
<td>0.59</td>
<td>0.64</td>
<td>0.00</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>MC1</td>
<td>0.71</td>
<td>0.80</td>
<td>0.10</td>
<td>0.64</td>
<td>0.74</td>
<td>0.10</td>
<td>0.69</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>MC2</td>
<td>0.62</td>
<td>0.65</td>
<td>0.03</td>
<td>0.54</td>
<td>0.66</td>
<td>0.12</td>
<td>0.68</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>MW1</td>
<td>0.67</td>
<td>0.72</td>
<td>0.05</td>
<td>0.59</td>
<td>0.64</td>
<td>0.05</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>PC1</td>
<td>0.73</td>
<td>0.83</td>
<td>0.10</td>
<td>0.68</td>
<td>0.82</td>
<td>0.14</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>PC2</td>
<td>0.76</td>
<td>0.74</td>
<td>0.02</td>
<td>0.65</td>
<td>0.66</td>
<td>0.04</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>PC3</td>
<td>0.70</td>
<td>0.78</td>
<td>0.08</td>
<td>0.65</td>
<td>0.81</td>
<td>0.16</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>PC4</td>
<td>0.67</td>
<td>0.91</td>
<td>0.34</td>
<td>0.63</td>
<td>0.88</td>
<td>0.25</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>PC5</td>
<td>0.66</td>
<td>0.76</td>
<td>0.10</td>
<td>0.69</td>
<td>0.73</td>
<td>0.15</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>PROMISE</td>
<td>Ant v1.7</td>
<td>0.75</td>
<td>0.82</td>
<td>0.07</td>
<td>0.74</td>
<td>0.80</td>
<td>0.06</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>Camel v1.6</td>
<td>0.60</td>
<td>0.71</td>
<td>0.11</td>
<td>0.61</td>
<td>0.73</td>
<td>0.12</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>Ivy v1.4</td>
<td>0.71</td>
<td>0.67</td>
<td>0.04</td>
<td>0.69</td>
<td>0.55</td>
<td>0.14</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>Jedit v4.0</td>
<td>0.74</td>
<td>0.80</td>
<td>0.06</td>
<td>0.72</td>
<td>0.77</td>
<td>0.05</td>
<td>0.79</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>Log4j v1.0</td>
<td>0.76</td>
<td>0.80</td>
<td>0.04</td>
<td>0.74</td>
<td>0.69</td>
<td>0.05</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>Lucene v2.4</td>
<td>0.68</td>
<td>0.77</td>
<td>0.05</td>
<td>0.65</td>
<td>0.75</td>
<td>0.10</td>
<td>0.67</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>POI v3.0</td>
<td>0.71</td>
<td>0.88</td>
<td>0.17</td>
<td>0.70</td>
<td>0.83</td>
<td>0.13</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>Tomcat v6.0</td>
<td>0.78</td>
<td>0.81</td>
<td>0.03</td>
<td>0.75</td>
<td>0.82</td>
<td>0.07</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Xalan v2.6</td>
<td>0.66</td>
<td>0.85</td>
<td>0.19</td>
<td>0.60</td>
<td>0.81</td>
<td>0.21</td>
<td>0.54</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>Xerces v1.3</td>
<td>0.69</td>
<td>0.83</td>
<td>0.14</td>
<td>0.72</td>
<td>0.77</td>
<td>0.05</td>
<td>0.77</td>
<td>0.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overall</th>
<th>Median rank</th>
<th>Average rank</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>1</td>
<td>1.42</td>
<td>0.64</td>
</tr>
<tr>
<td>LR</td>
<td>2</td>
<td>3.19</td>
<td>2.15</td>
</tr>
<tr>
<td>SC</td>
<td>3</td>
<td>3.35</td>
<td>1.67</td>
</tr>
<tr>
<td>LMT</td>
<td>3</td>
<td>3.42</td>
<td>1.94</td>
</tr>
<tr>
<td>NB</td>
<td>3.5</td>
<td>3.54</td>
<td>1.27</td>
</tr>
<tr>
<td>FCMM</td>
<td>6</td>
<td>5.96</td>
<td>1.08</td>
</tr>
<tr>
<td>4.5</td>
<td>6.5</td>
<td>6.73</td>
<td>1.69</td>
</tr>
<tr>
<td>NG</td>
<td>7</td>
<td>8.65</td>
<td>1.67</td>
</tr>
<tr>
<td>DT</td>
<td>7</td>
<td>8.89</td>
<td>1.56</td>
</tr>
<tr>
<td>KM</td>
<td>7.5</td>
<td>7.35</td>
<td>1.55</td>
</tr>
</tbody>
</table>
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).

$$e_{ij} = 1(w_{ij})$$  (2)

where 1$(w_{ij}) = 1$ if $w_{ij} > 0$, and 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 $deg^{cd}$, 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).

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

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

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

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

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

$$\phi^{cd} = \frac{deg^{cd}}{|V_c||V_d|}$$  (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 $\phi^{dd} = \frac{6}{15} = 0.4$. Similarly, we can get $deg^{cc} = 2 + 2 + 2 + 2 = 8$ and $\phi^{cc} = \frac{8}{6} = 1.333$. Finally, $deg^{cd} = 2 + 2 + 2 + 2 + 2 = 10$ and $\phi^{cd} = \frac{10}{8} = 1.25$.
6.2 Hypotheses

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

$H_0_0$: there is no difference in the ratios of connections from defective entities to other defective entities ($\phi^{dd}$) and clean entities $\phi^{cd}$.

$H_0_2$: there is no difference in the ratios of connections from clean entities to other clean entities ($\phi^{cc}$) and defective entities $\phi^{cd}$.

Hypotheses $H_0_1$ and $H_0_2$ are two sided and paired, since each project has three unique values: $\phi^{cc}$, $\phi^{cd}$, and $\phi^{dd}$. 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 $\delta$ [50] as the effect size to quantify the difference. Both the Mann-Whitney U test and the Cliff’s $\delta$ 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., $\phi^{cc}$, $\phi^{cd}$, and $\phi^{dd}$) 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 4.20e-05, when comparing the ratios $\phi^{dd}$ and $\phi^{cd}$ 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.

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 defect 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 second 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.

```r
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.
  Lsym = Dnsqrt %*% (W - diag(rep(1, nrow(W))))
  # Perform the eigendecomposition.
  ret_egl = eigen(Lsym, symmetric=TRUE)
  # Pick up the second smallest eigenvector.
  v1 = Dnsqrt %*% ret_egl$eigenvectors[, 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.
}
```

References


