Transfer Defect Learning

Jaechang Nam*, Sinno Jialin Pan†, and Sunghun Kim*
*Department of Computer Science and Engineering
The Hong Kong University of Science and Technology, China
{jcnam, hunkim}@cse.ust.hk
†Institute for Infocomm Research, Singapore
jspan@i2r.a-star.edu.sg

***Abstract*—Many software defect prediction approaches have been proposed and most are effective in within-project prediction settings. However, for new projects or projects with limited training data, it is desirable to learn a prediction model by using sufficient training data from existing source projects and then apply the model to some target projects (cross-project defect prediction). Unfortunately, the performance of cross-project defect prediction is generally poor, largely because of feature distribution differences between the source and target projects.**

In this paper, we apply a state-of-the-art transfer learning approach, TCA, to make feature distributions in source and target projects similar. In addition, we propose a novel transfer defect learning approach, TCA+, by extending TCA. Our experimental results for eight open-source projects show that TCA+ significantly improves cross-project prediction performance.

***Index Terms*—cross-project defect prediction, transfer learning, empirical software engineering**

I. Introduction

Recently, numerous effective software defect prediction approaches have been proposed and received a tremendous amount of attention [1], [2], [3], [4], [5]. Most approaches employ machine learning classifiers to build a prediction model from data sets mined from software repositories, and the model is used to identify software defects. However, most approaches are evaluated in within-project settings, i.e., a prediction model is built from a part of a project and the model is evaluated with the remainder of the project by 10-fold cross validation [2], [6], [7], [8] and/or random instance splits [5], [9], [10].

In practice, cross-project defect prediction is necessary. New projects often do not have enough defect data to build a prediction model. This cold-start is a well-known problem for recommender systems [11] and can be addressed by using cross-project defect prediction to build a prediction model using data from other projects. The model is then applied to new projects.

However, cross-project defect prediction often yields poor performance. Zimmermann et al. [12] evaluated cross-project defect prediction performance based on data from 12 projects (622 combinations). They found that only 21 pairs yielded reasonable prediction performance.

One of the main reasons for the poor cross-project prediction performance is the difference between the data distributions of source and target projects. Most machine learning classifiers are designed under the assumption that training and test data are represented in the same feature space and are drawn from the same data distribution [13]. This assumption usually holds for within-project prediction but might not hold for cross-project prediction.

To overcome the data distribution difference between source and target projects, transfer learning techniques have been proposed [13], [14], [15], [16], [17]. Most of them aim to extract common knowledge from one task domain and transfer it to another, and the transferred knowledge is used to train a prediction model [13].

On the basis of transfer learning, we propose *transfer defect learning* by adapting an existing method, Transfer Component Analysis (TCA) [18] developed by the second author of this paper. TCA aims to find a latent feature space for the data of both the source and target projects by minimizing the distance between the data distributions while preserving the original data properties. Once the latent space is learned, we can map the data of the source and target projects onto it. In this way, TCA tries to discover a new feature representation for the data of both the source and target projects, and to transform the data based on the new feature representation. As a result, the data distribution difference can be reduced. Then, we train a classifier on the source project data transformed by TCA, and finally apply the classifier to the transformed target project data for prediction. Most standard machine learning classifiers can be reused to achieve good performance using the transformed data in cross-project defect prediction. Furthermore, TCA does not use any defect information, as only data distribution information is used in transforming both the source and target project data.

However, we observed that the performance of cross-project defect prediction with TCA is sensitive to normalization, a data preprocessing technique widely used in machine learning [19].

Based on this observation, we propose TCA+, which automatically selects a suitable normalization before applying TCA for cross-project defect prediction. Specifically, TCA+ provides a set of rules for selecting an appropriate normalization option based on a given source–target pair to yield better cross-project prediction performance.

We evaluate our transfer defect learning approaches with two data sets found in the literature: ReLink [8] and AEEEM [20]. ReLink includes three projects: Apache HTTP server, Safe, and ZXing. AEEEM involves five projects: <u>A</u>pache Lucene, <u>E</u>clipse JDT, <u>E</u>clipse PDE UI, <u>E</u>quinox, and Mylyn.[1] For these two data sets, we conduct 26 cross-project defect prediction tasks in total. Our transfer learning approaches significantly improve the performance of cross-project defect prediction in terms of F-measure [21] for both data sets. Logistic regression with TCA+ yields average F-measures of 0.61 and 0.41 for ReLink and AEEEM respectively, whereas the average F-measures without TCA+ are 0.49 and 0.32 respectively.

Although transfer learning has been shown to be effective in other domains, to the best of our knowledge, we are the first to observe improved prediction performance by applying TCA for defect prediction. In addition, we propose TCA+, which adapts TCA and selects suitable normalization options based on a given cross-project prediction pair.

The remainder of this paper is organized as follows: Section II introduces our approaches. Section III describes our experimental setup. Section IV presents the results. Section V discusses the threats to validity. Section VI surveys the defect prediction and transfer learning techniques. Finally, Section VII concludes this study and discusses plans for future work.

II. Approach

In this section, we first provide an overview of transfer defect learning, and then present our proposed solutions in detail.

A. Transfer Defect Learning Overview

In our cross-project defect prediction, which learns a model from a source project and applies the learned model to a target project, we assume that the source and target projects have the same set of features (Section III). However, their feature distributions may differ.

The objective of our transfer defect learning is to make the feature distributions of the source and target projects similar. Numerous transfer learning techniques have been proposed by the machine learning and data mining communities to address such distribution difference problems [13], [14], [15], [16], [17]. In this paper, we use a state-of-the-art transfer learning approach, *transfer component analysis* (TCA) [18], for cross-project defect prediction. In addition, we propose an extended TCA, called TCA+, for the defect prediction domain. In the following sections, we explain the basic notation and problem definitions for transfer learning, and then we describe TCA and TCA+ in detail.

B. Notation and Problem Definition

In the sequel, both matrices and vectors are written in boldface (e.g., $\mathbf{X}$, $\mathbf{x}$). The transpose of a vector or matrix is denoted by a superscript $^\top$, while tr$(\mathbf{X})$ and $\mathbf{X}^{-1}$ denote the trace and inverse, respectively, of $\mathbf{X}$. For a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, whose rows and columns correspond to instances and feature vectors respectively, $\mathbf{x}_i$ denotes the i-th instance of $\mathbf{X}$ and $\mathbf{x}^{(j)}$ denotes the j-th feature vector, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

Based on this notation, our problem is formally defined as follows:

- Let the given source project data set be $\mathcal{D}_S = \{(\mathbf{x}_{S_i}, y_{S_i})\}_{i=1}^{n_1}$, where $\mathbf{x}_{S_i} \subseteq \mathbb{R}^{1 \times m}$ is the input (e.g., a file is represented by a vector of metrics), and y_{S_i} is the corresponding defect information (e.g., clean or buggy). Let the given target project data set be $\mathcal{D}_T = \{\mathbf{x}_{T_i}\}_{i=1}^{n_2}$. Assume that $\mathbf{x}_{S_i}$ and $\mathbf{x}_{T_i}$ are represented by the same set of metrics. Let n_1 and n_2 be the numbers of files in the source and target projects respectively. Let $\mathcal{P}(\mathbf{X}_S)$ and $\mathcal{P}(\mathbf{X}_T)$ be the marginal distributions of $\mathbf{X}_S = \{\mathbf{x}_{S_i}\}_{i=1}^{n_1}$ and $\mathbf{X}_T = \{\mathbf{x}_{T_i}\}_{i=1}^{n_2}$ from the source and target projects respectively. In general, $\mathcal{P}(\mathbf{X}_S)$ and $\mathcal{P}(\mathbf{X}_T)$ can be different in the cross-project setting. The objective of TCA is to explicitly learn a transformation mapping φ to map the data of both the source and target projects onto a latent feature space such that the difference between the data distributions of $\varphi(\mathbf{X}_S)$ and $\varphi(\mathbf{X}_T)$ becomes small, and thus a standard model f trained on $\varphi(\mathbf{X}_S)$ and the corresponding defect information $\mathbf{Y}_S$ can achieve precise predictions on $\varphi(\mathbf{X}_T)$.

The details of TCA are given in the following section.

C. Transfer Component Analysis

Transfer Component Analysis (TCA) [18], proposed by the second author of this paper, is a state-of-the-art feature extraction technique for transfer learning. The motivation behind TCA is that some common latent factors may exist between source and target domains, even though the observed features of the domains are different. To reveal the latent factors, we project the domains onto a new space, which is called the latent space. In this way, the domain difference can be reduced, while the original data structures (e.g., data variance and local geometric structure) can be preserved. As a result, the latent space spanned by the latent factors can be used as a bridge for cross-domain classification tasks. For example, Internet Explorer 8 and Firefox are two projects represented by the same set of standard metrics, but they might have very different metric values since their development processes are significantly different [12]. However, both of them are Web browsers. This implies that they have some commonality in coding, even though the commonality may be hidden. If the hidden commonality can be discovered and used to represent the data of the two projects, then the cross-project difference may be dramatically reduced. Thus, a defect prediction model trained on the data of one project can be successfully applied to the other project.

Mathematically, TCA tries to learn a transformation φ to map the original data of source and target domains to a latent space where the difference between domains $\text{Dist}(\varphi(\mathbf{X}_S), \varphi(\mathbf{X}_T))$ is small and the data variance after transformation $\text{Var}(\{\varphi(\mathbf{X}_S), \varphi(\mathbf{X}_T)\})$ is large. Assume that the transformation $\varphi : \mathbb{R}^m \to \mathbb{R}^d$ maps the original feature

[1]D'Ambros et al. [20] collected a data set of metrics for a benchmark of existing defect prediction approaches. We call this benchmark data AEEEM, after the first letters of five projects.

TABLE I: Preliminary cross-project prediction results (F-measures) for ReLink using TCA with different normalization options under logistic regression. Values greater than Baseline are in boldface.

Source⇒Target	Baseline	TCA				
		NoN	N1	N2	N3	N4
Safe⇒**Apache**	0.52	**0.68**	**0.75**	**0.60**	**0.72**	**0.64**
Apache⇒**Safe**	0.56	**0.71**	0.54	**0.64**	0.48	**0.72**
Apache⇒**ZXing**	0.46	0.32	**0.49**	0.42	0.09	0.45

vectors to a d-dimensional subspace. Then, the objective of TCA at a high level can be written as follows:

$$\begin{aligned} \arg\min_{\varphi} \quad & \mathrm{Dist}(\varphi(\mathbf{X}_S), \varphi(\mathbf{X}_T)) + \lambda\, R(\varphi), \\ \text{s.t.} \quad & \text{constraints on } \varphi(\mathbf{X}_S) \text{ and } \varphi(\mathbf{X}_T), \end{aligned} \tag{1}$$

where φ is to be learned by minimizing the difference between domains and satisfying some constraints to preserve data variance after transformation, $R(\varphi)$ is a regularization term in φ to avoid overfitting, and $\lambda \geq 0$ is a tradeoff parameter to control the influence of the regularization term in the objective. Regularization is a common technique used in machine learning and data mining [22], [23].

In many real-world applications, the transformation φ can also be taken as linear [24], [25], which can be expressed as $\varphi(\mathbf{x}) = \mathbf{x}\boldsymbol{\Theta}$, where $\boldsymbol{\Theta} \in \mathbb{R}^{m \times d}$ is a matrix that maps m-dimensional feature vectors to d-dimensional ones. After learning the transformation $\boldsymbol{\Theta}$ by using TCA, we can map the original data of the source and target domains, $\mathbf{X}_S$ and $\mathbf{X}_T$, to the latent space by using $\varphi(\mathbf{X}_S) = \mathbf{X}_S\boldsymbol{\Theta}$ and $\varphi(\mathbf{X}_T) = \mathbf{X}_T\boldsymbol{\Theta}$, respectively. Then, we train a classifier f on the transformed source data $\mathbf{X}_S\boldsymbol{\Theta}$ and the corresponding labels $\mathbf{Y}_S$. Finally, we use the model f to make predictions on the target domain test data $f(\mathbf{X}_T\boldsymbol{\Theta})$.

D. Normalization for Data Preprocessing

For TCA, we apply normalization, which is a data preprocessing technique for machine learning and data mining [19]. Normalization gives all features of a data set an equal weight and is thus known to be useful for classification algorithms [19]. Graf et al. also confirmed that normalization can improve prediction performance of classification models [26]. For this reason, we normalize the data of source and target projects before building classification models with TCA. There are many normalization methods. In this study, we use min-max and z-score normalization methods because these are commonly used in the machine learning literature [19], [27].

The equations for normalization of both the source and target project data used in our experiments are listed below.

- NoN: No normalization is applied.
- N1: For each value $x_i^{(j)}$ of a feature vector $\mathbf{x}^{(j)}$, $\widetilde{x}_i^{(j)} = \frac{(x_i^{(j)} - min(\mathbf{x}^{(j)}))}{max(\mathbf{x}^{(j)}) - min(\mathbf{x}^{(j)})}$, where $\widetilde{x}_i^{(j)}$ is the normalized value of the original value $x_i^{(j)}$. Furthermore, $min(\mathbf{x}^{(j)})$ and $max(\mathbf{x}^{(j)})$ are the minimum and maximum values of $\mathbf{x}^{(j)}$ respectively. Note that this equation is applicable to either source or target project data.
- N2: For each value $x_i^{(j)}$ of a feature vector $\mathbf{x}^{(j)}$, $\widetilde{x}_i^{(j)} = \frac{(x_i^{(j)} - mean(\mathbf{x}^{(j)}))}{std(\mathbf{x}^{(j)})}$, where $mean(\mathbf{x}^{(j)})$ and $std(\mathbf{x}^{(j)})$ are the mean value and standard deviation of $\mathbf{x}^{(j)}$ respectively. Note that this equation is applicable to either source or target project data.
- N3: For each value $x_i^{(j)}$ of a feature vector $\mathbf{x}^{(j)}$, $\widetilde{x}_i^{(j)} = \frac{(x_i^{(j)} - mean(\mathbf{x}_{src_i}^{(j)}))}{std(\mathbf{x}_{src_i}^{(j)})}$, where $\mathbf{x}_{src_i}^{(j)}$ is the j-th feature vector of the source project data.
- N4: For each value $x_i^{(j)}$ of a feature vector $\mathbf{x}^{(j)}$, $\widetilde{x}_i^{(j)} = \frac{(x_i^{(j)} - mean(\mathbf{x}_{tar_i}^{(j)}))}{std(\mathbf{x}_{tar_i}^{(j)})}$, where $\mathbf{x}_{tar_i}^{(j)}$ is the j-th feature vector of the target project data.

Option N1 represents the min-max normalization with a value range from zero to one [19]. In other words, the minimum and maximum values of an original data set are transformed into zero and one respectively.

Option N2 represents the z-score normalization [27], which makes the mean zero and the standard deviation one. From N2, we also derive the options N3 and N4. In the case of N3, the mean and standard deviation are computed only based on the source project data but applied to both the source and target project data for normalization. In contrast, in the case of N4, the mean and standard deviation are computed only based on the target project data but applied to both project data for normalization. The reason we apply N3 and N4 derived from N2 is to consider a case where the size of either the source or target data set is too small to estimate the data distribution.

Table I shows preliminary F-measure results from using TCA with different normalizations for some cross-prediction pairs (e.g., Safe⇒Apache) on ReLink.[2] We observed that normalization on source and/or target data sets improves defect prediction performance. However, prediction results of TCA vary according to different normalization options. This clearly shows that using a suitable normalization option can make TCA more effective in terms of prediction performance.

E. TCA+

We extend TCA and propose TCA+ to achieve better cross-project prediction performance. As shown in Table I, we observed that prediction performance varies according to different normalization selections. Based on this observation, we propose an algorithm to select appropriate normalization options for a given cross-prediction pair.

Normalization selections start with the basic idea of identifying similarity of data set characteristics between the source and target projects. To implement normalization selection, we first define a *Data set Characteristic Vector* (DCV) and extract a DCV from each project. To extract a DCV, we measure the distance between a pair of instances. Euclidean distance, which is the most popular distance measure [19], is used in this approach. For each project data set, we compute the overall Euclidean distance of all pairs of instances, $DIST$, as follows:

$$DIST = \{d_{ij} : \forall i, j, 1 \leq i < n, 1 < j \leq n, i < j\}, \tag{2}$$

[2]Hereafter a rightward double arrow (⇒) denotes prediction.

TABLE II: Six elements of a characteristic vector for a data set.

Name	Description
$dist\_mean$	Mean value of $DIST$
$dist\_median$	Median value of $DIST$
$dist\_min$	Minimum value of $DIST$
$dist\_max$	Maximum value of $DIST$
$dist\_std$	Standard deviation of $DIST$
$numInstances$	The number of instances

TABLE III: Conditions for assignment of nominal values to similarity vector $\mathbf{s}_{S \Rightarrow T}$.

Nominal values of $\mathbf{s}_{S \Rightarrow T}[e]$	Condition
MUCH MORE	$\mathbf{c}_S[e] \times 1.6 < \mathbf{c}_T[e]$
MORE	$\mathbf{c}_S[e] \times 1.3 < \mathbf{c}_T[e] \leq \mathbf{c}_S[e] \times 1.6$
SLIGHTLY MORE	$\mathbf{c}_S[e] \times 1.1 < \mathbf{c}_T[e] \leq \mathbf{c}_S[e] \times 1.3$
SAME	$\mathbf{c}_S[e] \times 0.9 \leq \mathbf{c}_T[e] \leq \mathbf{c}_S[e] \times 1.1$
SLIGHTLY LESS	$\mathbf{c}_S[e] \times 0.7 \leq \mathbf{c}_T[e] < \mathbf{c}_S[e] \times 0.9$
LESS	$\mathbf{c}_S[e] \times 0.4 \leq \mathbf{c}_T[e] < \mathbf{c}_S[e] \times 0.7$
MUCH LESS	$\mathbf{c}_T[e] < \mathbf{c}_S[e] \times 0.4$

where d_{ij} is the Euclidean distance between instances i and j, and n is the number of instances of a project data set. Then, we compute the minimum, maximum, mean, median, and standard deviation of $DIST$ as characteristics to represent the project data. In addition, we include the number of data set instances into the characteristic vector. The reason for this is that more instances can provide more information to build a better prediction model. Thus, a DCV includes six elements as shown in Table II. For a source project S and a target project T, we represent the DCVs by $\mathbf{c}_S$ and $\mathbf{c}_T$ respectively.

To measure the similarity between a source project S and a target project T using DCVs, we define a *similarity vector* (SV), which represents the difference between $\mathbf{c}_S$ and $\mathbf{c}_T$ with a degree of similarity such as "much more" (very different), "less" (different), or "same".

We compute the degree of similarity, $\mathbf{s}_{S \Rightarrow T}[e]$, for each element e of a DCV, where $\mathbf{c_S}[e]$ and $\mathbf{c_T}[e]$ are the values of an element e in $\mathbf{c_S}$ and $\mathbf{c_T}$ respectively, as shown in Table III. To compute the degree of similarity, we multiply $\mathbf{c_S}[e]$ by a certain factor and then compare the result with $\mathbf{c_T}[e]$. In the case of "SAME", we used 0.9 and 1.1 as factors to allow small deviations [12]. By increasing or decreasing those factors, we change the extent of deviation and define various degrees of similarity: MUCH LESS, LESS, SLIGHTLY LESS, SLIGHTLY MORE, MORE, and MUCH MORE. For example, if the mean distance ($dist\_mean$) of instances in Apache (source) is $\mathbf{c}_{Apache}[dist\_mean] = 905$, and the corresponding value for Safe (target) is $\mathbf{c}_{Safe}[dist\_mean] = 238$, the degree of similarity for those values, $\mathbf{s}_{Apache \Rightarrow Safe}[dist\_mean]$, is designated as "MUCH LESS" ($238 < 905 \times 0.4 = 362$).

Using an SV and properties of normalization options, we propose a set of decision rules to select suitable normalization options for TCA as follows (the rules are sequentially applied).

Rule1: If $\mathbf{s}_{S \Rightarrow T}[dist\_mean]$ and $\mathbf{s}_{S \Rightarrow T}[dist\_std]$ are individually "SAME", do not apply any normalization. If the means and standard deviations of $DIST$ for the source and target data are similar, we assume their data sets are similar enough, and do not normalize data.

Rule2: If $\mathbf{s}_{S \Rightarrow T}[dist\_min]$, $\mathbf{s}_{S \Rightarrow T}[dist\_max]$, and $\mathbf{s}_{S \Rightarrow T}[numInstaces]$ are individually either "MUCH LESS" or "MUCH MORE," choose N1 (min-max normalization). Both the minimum and maximum distances of instances may depend on the minimum and maximum values of each feature in the original data. A large gap in $dist\_max$ or $dist\_min$ between the source and target data may indicate different distributions. This distribution difference may generate more effects when there is a great difference in $numInstances$ between two projects. Thus, we also include $\mathbf{s}_{S \Rightarrow T}[numInstances]$ in this rule.

Rule3: N3 is a variation of z-score normalization, which uses the $dist\_mean$ and $dist\_std$ of only a source project for normalization. Thus, we apply N3 when a target project belongs to one of these: First, $\mathbf{s}_{S \Rightarrow T}[dist\_std]$ is "MUCH MORE" and the number of instances of a target project is less than that of a source project. In this case, the target data are sparse as $dist\_std$ is large. In the second case, $\mathbf{s}_{S \Rightarrow T}[dist\_std]$ is "MUCH LESS" and the number of instances of a target project is greater than that of a source project. This indicates that the target data are dense. We regard these two cases to imply that the target data may have very little statistical information, so we normalize both the source and target data by using the $dist\_mean$ and $dist\_std$ of the source project.

Rule4: N4 is another variation of z-score normalization, which uses the $dist\_mean$ and $dist\_std$ of only a target project for normalization. Since N4 and N3 are symmetric, we derive a rule for N4 symmetrically from that for N3. In other words, $\mathbf{s}_{S \Rightarrow T}[dist\_std]$ is "MUCH MORE" and the number of instances of a target project is greater than that of a source project. As another condition, $\mathbf{s}_{S \Rightarrow T}[dist\_std]$ is "MUCH LESS" and the number of instances of a target project is less than that of a source project.

Rule5: If none of the rules (Rules 1 to 4) are applicable, we choose N2 (z-score normalization) to make the mean and standard deviation similar between two projects.

Algorithm 1 formally shows the decision rules for TCA+. We evaluate the rules for TCA+ in Section IV-B.

III. Experimental Setup

In this section, we describe the experimental setup in detail, including the defect prediction process, subject systems, metrics (features), and evaluation measures.

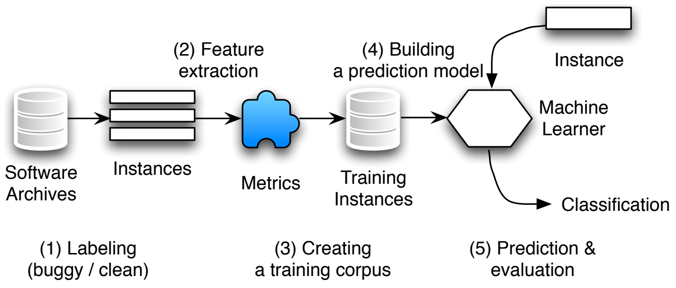
A. Defect Prediction Process

Figure 1 shows the file-level defect prediction process used in our experiment. This is a typical prediction process commonly used in the literature [6], [7], [28], [29].

The first step of the process is collecting files (instances) and labeling them. The labeling task is based on the number of post-release-defects for each file. If a file has at least one post-release-defect, it is labeled as *buggy*. Otherwise, it is labeled as *clean*. Then, defect prediction metrics such as complexity metrics [30], [31] are used as features. The instance features

Algorithm 1 Decision Rules for Normalization Selections

```
Input: Similarity vector $\mathbf{S}_{S\Rightarrow T}$ , where $S$ and $T$ are source and target
    projects.
Output: Normalization option (NoN|N1|N2|N3|N4)
    /*Rule1*/
 1: if $\mathbf{S}_{S\Rightarrow T}[dist\_mean]$ = SAME and
    $\mathbf{S}_{S\Rightarrow T}[dist\_std]$ = SAME then
 2:     return NoN
 3: end if
    /*Rule2*/
 4: if $\mathbf{S}_{S\Rightarrow T}[numInstances]$ and $\mathbf{S}_{S\Rightarrow T}[dist\_min]$ and
    $\mathbf{S}_{S\Rightarrow T}[dist\_max]$ are MUCH MORE or MUCH LESS then
 5:     return N1
 6: end if
    /*Rule3*/
 7: if ($\mathbf{S}_{S\Rightarrow T}[dist\_std]$ = MUCH MORE and
    $\mathbf{S}_{S\Rightarrow T}[numInstances]$ < SAME) or
    ($\mathbf{S}_{S\Rightarrow T}[dist\_std]$ = MUCH LESS and
    $\mathbf{S}_{S\Rightarrow T}[numInstances]$ > SAME) then
 8:     return N3
 9: end if
    /*Rule4*/
10: if ($\mathbf{S}_{S\Rightarrow T}[dist\_std]$ = MUCH MORE and
    $\mathbf{S}_{S\Rightarrow T}[numInstances]$ = MUCH MORE) or
    ($\mathbf{S}_{S\Rightarrow T}[dist\_std]$ = MUCH LESS and
    $\mathbf{S}_{S\Rightarrow T}[numInstances]$ = MUCH LESS) then
11:     return N4
12: end if
    /*Ruel5*/
13: return N2
```



(2) Feature extraction
(4) Building a prediction model
Instance
Machine Learner
Software Archives
Instances
Metrics
Training Instances
Classification
(1) Labeling (buggy / clean)
(3) Creating a training corpus
(5) Prediction & evaluation

Fig. 1: Defect Prediction Process [6]

and labels are used to train prediction models using machine learning classifiers. Finally, the trained models predict whether new instances are *buggy* or *clean*.

The instances used to build models constitute a training set, whereas those used to test the learned models constitute a test set. In within-project prediction, the training and test sets are from the same project. For cross-project prediction, the training set is from one project (the source) and the test set is from another project (the target).

B. Benchmark Sets

For our experiments, we used two existing defect benchmark data sets, ReLink and AEEEM [8], [20]. Table IV and V summarize these data sets.

The first data set, ReLink, was collected by Wu et al. [8], and the defect information in ReLink has been manually verified and corrected. Bird et al. found labeling bias in automatic defect information extraction [32]. They also found that noise has a non-trivial impact on defect prediction performance [32], [33]. To avoid noise and its influence, we use manually verified defect data such as ReLink. ReLink has 26 complexity metrics, which are widely used in defect prediction [8], [30], [31], [33]. Table VI lists only 7 metrics selected from among the 26 metrics in the interest of brevity (for a detailed description of each metric, refer to the Understand web site [34]).

TABLE VI: List of metrics selected for ReLink. (For a detailed description of all metrics used, refer to the Understand Web site [34].)

Metrics	Description
AvgCyclomatic	Average cyclomatic complexity for all nested functions of methods.
AvgLine	Average number of lines for all nested functions or methods.
CountLine	Number of all lines for all nested functions or methods.
CountStmt	Number of statements.
MaxCyclomatic	Maximum cyclomatic complexity of all nested functions or methods.
RatioCommentToCode	Ratio of comment lines to code lines.
SumCyclomatic	Sum of cyclomatic complexity of all nested functions or methods.

The second data set, AEEEM, was collected by D'Ambros et al. [20]. AEEEM consists of 61 metrics: 17 source code metrics, 5 previous-defect metrics, 5 entropy-of-change metrics, 17 entropy-of-source-code metrics, and 17 churn-of-source-code metrics [20]. In particular, AEEEM includes linearly decayed entropy (LDHH) and weighted churn (WCHU). Both LDHH and WCHU have been verified as informative defect predictors [20]. Table VII lists selected metrics among the 61 metrics.

C. Experimental Design

To evaluate our proposed approaches, we conduct experiments in different settings: within-project defect prediction, cross-project defect prediction without transfer learning, and cross-project defect prediction with TCA or TCA+.

1) Within-project Prediction: We present the within-project defect prediction results, since these results provide references for cross-project prediction results.

For within-project defect prediction, we used the 50:50 random split to obtain training and test sets [35]. This provides a setting similar to that of cross-project prediction, assuming that the data sizes of the training set (source) and test set (target) are similar. One alternative would be 10-fold cross validation, but it essentially uses 90% training data and 10% test data. Moreover, random split is widely used in the literature [5], [9], [10].

The 50:50 random split randomly selects 50% of the instances for a training set, and the remaining 50% for a test set. This random selection process for training and test sets may be biased and may affect the prediction performance. Hence, we repeat this process 100 times with different 50:50 random splits and report the average prediction results.

2) Cross-project Prediction without Transfer Learning: For cross-project prediction, we first identify all combinations of the project pairs in data sets. For example, ReLink has six cross-project combinations: Apache⇒Safe, Apache⇒ZXing, Safe⇒Apache, Safe⇒ZXing, ZXing⇒Apache, and

TABLE IV: Projects of ReLink used in the experiment [8].

Subject	Type	Version	# of files(instances)	# of buggy files (%)	# of metrics
Apache HTTP Server	Web server	2.0	194	98(50.52%)	26
OpenIntents Safe	Security	Revision 1088–2073	56	22(39.29%)	26
ZXing	Bar-code reader library	1.6	399	118(29.57%)	26

TABLE V: Projects of AEEEM used in the experiment [20].

Subject	Type	Time period	# of files(instances)	# of buggy files (%)	# of metrics
Equinox (EQ)	OSGi framework	1.1.2005–6.25.2008	325	129(39.69%)	61
Eclipse JDT Core (JDT)	Development	1.1.2005–6.17.2008	997	206(20.66%)	61
Apache Lucene (LC)	Text search engine library	1.1.2005–10.8.2008	399	691(9.26%)	61
Mylyn (ML)	Task management	1.17.2005–3.17.2009	1862	245(13.16%)	61
Eclipse PDE UI (PDE)	Development	1.1.2005–9.11.2008	1492	209(14.01%)	61

TABLE VII: List of metrics selected for AEEEM. (For a detailed description of all metrics we used, refer to D'Ambros et al. [20].

Category	Metrics	Description
Source code	*ck_oo_cbo*	Coupling Between Objects
	ck_oo_numberOfLinesOfCode	Number of lines of code.
Previous defects	*numberOfBugsFoundUntil*	Number of all bugs.
	numberOfCriticalBugsFoundUntil	Number of bugs whose severity is critical and blocker.
Entropy	*CvsEntropy*	Entropy of code changes.
	CvsLogEntropy	Logarithmically decayed *CvsEntropy*.
Entropy of source code	*LDHH_cbo*	Linearly decayed entropy of *ck_oo_cbo*.
	LDHH_numberOfLinesOfCode	Linearly decayed entropy of *ck_oo_numberOfLinesOfCode*.
Churn of source code	*WCHU_cbo*	Weighted churn of *ck_oo_cbo*.
	WCHU_numberOfLinesOfCode	Weighted churn of *ck_oo_numberOfLinesOfCode*.

ZXing⇒Safe. In the same manner, we also identify all 20 cross-project combinations from the five projects of AEEEM. We cannot mix the projects of ReLink with those of AEEEM, since they have different sets of metrics.

Then, we build a prediction model using all instances from one project (source) and predict defects in another project (target) by using the model. For example, we build a prediction model from all instances in Apache and use that model to predict defects for all instances in Safe (Apache⇒Safe).

Unlike within-project defect prediction, the evaluation of cross-project prediction does not involve any randomness, because all instances from one project constitute a training set and all instances from another project constitute a test set. We performed the evaluation once.

3) Cross-project Prediction with Transfer Learning (TCA): The third phase of our experiments is cross-project defect prediction with TCA. This is similar to the cross-project prediction setting without transfer learning. However, a prediction model trained on the source project is not applied directly to the target project. Instead, we first transform both the source and target data sets using TCA. Then, we train a prediction model on the transformed source project and use that prediction model to predict defects in the transformed target project. We compare the defect prediction performance for various normalization options in Section IV.

4) Cross-project Prediction with TCA+: We conduct cross-project defect prediction with TCA+, which consists of two steps: normalization selections and TCA application. As introduced in Section II-E, we apply Algorithm 1 to select a normalization option for each source-target project pair before applying TCA. We report the cross prediction results of TCA+ for both ReLink and AEEEM in Section IV.

D. Machine Learning Classifier

As the underlying machine learning classifier for both within- and cross-project prediction, we use logistic regression, which is widely used in the defect prediction literature [6], [7], [12], [36], [37]. Specifically, we use the logistic regression implemented in LIBLINEAR [38], an award-winning library for large linear classification, which is widely used in machine learning and data mining [39], [40]. For LIBLINEAR execution, we use the options "-S 0" (i.e., logistic regression) and "-B -1" (i.e., no bias term added).

E. Evaluation Measures

To evaluate prediction accuracy, we use the F-measure [21], which is the harmonic mean of precision and recall.

The following are used to define precision, recall, and F-measure: (i) predicting a buggy instance as buggy ($b \rightarrow b$); (ii) predicting a buggy instance as clean ($b \rightarrow c$); and (iii) predicting a clean instance as buggy ($c \rightarrow b$). We use these outcomes to evaluate the accuracy of defect prediction with the measures defined below [21], [23], [41]:

Precision: The ratio of the number of instances correctly classified as buggy ($N_{b \rightarrow b}$) to the number of instances classified as buggy.

$$\text{Buggy precision: } P(b) = \frac{N_{b\rightarrow b}}{N_{b\rightarrow b} + N_{c\rightarrow b}} \qquad (3)$$

Recall: The ratio of the number of instances correctly classified as buggy ($N_{b \rightarrow b}$) to the number of buggy instances.

$$\text{Buggy recall: } R(b) = \frac{N_{b\rightarrow b}}{N_{b\rightarrow b} + N_{b\rightarrow c}} \qquad (4)$$

F-measure: A composite measure of precision *P(b)* and recall *R(b)* for buggy instances.

$$\text{Buggy F-measure: } F(b) = \frac{2 \times P(b) \times R(b)}{P(b) + R(b)} \qquad (5)$$

Usually, there are trade-offs between precision and recall. For example, by sacrificing precision, it may be possible to improve recall. These trade-offs make it difficult to compare the performances of several prediction models by using only either the precision or recall value [2], [23], [41]. For this reason, we compare prediction results using F-measure values, which fall in the range $[0, 1]$. The higher is the F-measure, the better is the performance.

IV. Results

In this section, we present detailed experimental results for TCA with different normalization options (Section IV-A) and TCA+ (Section IV-B).

A. TCA with Different Normalization Options

Table VIII and IX list F-measure values of cross-project prediction applying TCA with different normalization options. In each table, the last row contains the average F-measures of all cross prediction combinations for each normalization option, while the other rows contain the F-measures of every cross-prediction combination. The Baseline column lists the results of cross-project prediction without TCA. We used logistic regression to build classifiers.

Table VIII and IX show that different normalization options affect the prediction results. Compared to Baseline, we can observe increments in average F-measure with several normalization options. For example, the average F-measure for ReLink with N4 (Table VIII) represents an increase from 0.49 to 0.59. Likewise, the average F-measure for AEEEM (Table IX) is significantly increased from 0.31 to 0.41 by using N2 option. However, TCA with NoN or N1 does not show any improvement in terms of average F-measure. For example, the average F-measure for ReLink with N1 (Table VIII) represents a decrease from 0.49 to 0.44.

Table VIII and IX also list detailed F-measures for each cross-prediction combination. F-measure values greater than the corresponding Baseline values are presented in boldface.

In the case of ReLink with N2, the F-measures of four out of six cross-prediction combinations increased by 0.06–0.33. However, two out of the six combinations decreased by 0.04. This indicates that, for some prediction combinations, TCA cannot yield better cross-project defect prediction performance than the Baseline. This indicates that, for a specific prediction combination, TCA might not find a correct latent space spanned by common latent features. To address this issue, we propose TCA+, which is an extension of TCA to determine the suitable normalization option for a given prediction combination as explained in Section II-E.

TABLE VIII: Cross-project prediction results for ReLink using TCA with various normalization options under logistic regression. Values greater than Baseline are in boldface.

Source⇒Target	Baseline	TCA				
		NoN	N1	N2	N3	N4
Safe⇒**Apache**	0.52	**0.68**	**0.75**	**0.60**	**0.72**	**0.64**
ZXing⇒**Apache**	0.69	0.30	0.26	0.65	**0.72**	0.64
Apache⇒**Safe**	0.56	**0.71**	0.54	**0.64**	0.48	**0.72**
ZXing⇒**Safe**	0.59	0.35	0.08	**0.65**	**0.64**	**0.70**
Apache⇒**ZXing**	0.46	0.32	**0.49**	0.42	0.09	0.45
Safe⇒**ZXing**	0.10	**0.35**	**0.52**	**0.43**	**0.29**	**0.42**
Average	0.49	0.45	0.44	**0.57**	**0.49**	**0.59**

TABLE IX: Cross-project prediction results for AEEEM using TCA with various normalization options under logistic regression. Values greater than Baseline are in boldface.

Source⇒Target	Baseline	TCA				
		NoN	N1	N2	N3	N4
JDT⇒**EQ**	0.31	0.17	**0.43**	**0.59**	**0.51**	**0.60**
LC⇒**EQ**	0.50	0.12	0.29	**0.62**	**0.68**	**0.60**
ML⇒**EQ**	0.24	0.19	0.19	**0.56**	**0.71**	**0.64**
PDE⇒**EQ**	0.43	0.19	0.13	**0.58**	**0.60**	**0.52**
EQ⇒**JDT**	0.39	0.39	0.36	**0.48**	**0.43**	**0.54**
LC⇒**JDT**	0.48	0.00	0.41	**0.56**	0.43	**0.49**
PDE⇒**JDT**	0.47	0.09	0.02	**0.54**	**0.48**	**0.48**
ML⇒**JDT**	0.42	0.14	0.25	**0.52**	**0.43**	**0.44**
EQ⇒**LC**	0.27	0.27	0.21	**0.27**	0.27	0.25
JDT⇒**LC**	0.24	0.03	**0.29**	**0.31**	**0.35**	**0.26**
ML⇒**LC**	0.10	0.05	**0.14**	**0.25**	**0.24**	**0.27**
PDE⇒**LC**	0.33	0.09	0.06	0.27	0.33	0.25
EQ⇒**ML**	0.19	**0.29**	**0.20**	**0.23**	0.04	**0.24**
JDT⇒**ML**	0.27	0.06	0.27	**0.32**	**0.36**	0.25
LC⇒**ML**	0.20	0.00	0.20	**0.29**	**0.22**	**0.27**
PDE⇒**ML**	0.27	0.04	0.11	**0.29**	**0.34**	0.21
EQ⇒**PDE**	0.31	**0.36**	0.25	**0.33**	0.28	**0.34**
JDT⇒**PDE**	0.27	0.09	**0.38**	**0.39**	**0.38**	**0.36**
LC⇒**PDE**	0.32	0.02	0.25	**0.37**	0.32	**0.35**
ML⇒**PDE**	0.27	0.13	**0.32**	**0.37**	0.27	**0.35**
Average	0.31	0.14	0.24	**0.41**	**0.38**	**0.38**

TCA with N2, N3, or N4 significantly improves cross-project defect prediction results of both ReLink and AEEEM data sets in terms of average F-measure. However, the results of some cross-prediction combinations are not improved by TCA.

B. TCA+

Table X and XI compare F-measure values between TCA and TCA+. In particular, we compare TCA+ to TCA with N4 and N2, which lead to the best results for ReLink and AEEEM, respectively. F-measure values greater than Baseline values (cross-prediction without TCA or TCA+) are presented in boldface.

As shown in Table X and XI, the F-measures with TCA+ increase for all cross-prediction combinations in comparison to Baseline. For example, the F-measure of ZXing⇒Apache with TCA+ (0.72) is above Baseline (0.69), while the F-measure (0.64) with TCA (N4) is less than the Baseline.

TABLE X: Comparison of cross- and within-project prediction results between TCA+ and the best TCA option N4, in ReLink. Logistic regression is used. Values greater than Baseline are in boldface. Underline means that all values of source and target combinations with a specific normalization option are increased compared to Baseline.

Source⇒Target	Baseline	TCA (N4)	TCA+	Within Target⇒Target
Safe⇒**Apache**	0.52	**0.64**	**0.64**	0.64
ZXing⇒**Apache**	0.69	0.64	**0.72**	
Apache⇒**Safe**	0.49	**0.72**	**0.72**	0.62
ZXing⇒**Safe**	0.59	**0.70**	**0.64**	
Apache⇒**ZXing**	0.46	0.45	**0.49**	0.33
Safe⇒**ZXing**	0.10	**0.42**	**0.43**	
Average	0.49	**0.59**	**<u>0.61</u>**	0.53

TABLE XI: Comparison of cross- and within-project prediction results between TCA+ and the best TCA option N2, for AEEEM. Logistic regression is used. Values greater than Baseline are in boldface. Underline means that all values of source and target combinations with a specific normalization option are increased compared to Baseline.

Source⇒Target	Baseline	TCA (N2)	TCA+	Within Target⇒Target
JDT⇒**EQ**	0.31	**0.59**	**0.60**	0.58
LC⇒**EQ**	0.50	**0.62**	**0.62**	
ML⇒**EQ**	0.24	**0.56**	**0.56**	
PDE⇒**EQ**	0.43	**0.58**	**0.60**	
EQ⇒**JDT**	0.39	**0.48**	**0.54**	0.56
LC⇒**JDT**	0.48	**0.56**	**0.56**	
ML⇒**JDT**	0.42	**0.52**	**0.43**	
PDE⇒**JDT**	0.47	**0.54**	**0.48**	
EQ⇒**LC**	0.27	**0.27**	**0.27**	0.37
JDT⇒**LC**	0.24	**0.31**	**0.31**	
ML⇒**LC**	0.10	**0.25**	**0.25**	
PDE⇒**LC**	0.33	0.27	**0.33**	
EQ⇒**ML**	0.19	**0.23**	**0.23**	0.27
JDT⇒**ML**	0.27	**0.32**	**0.36**	
LC⇒**ML**	0.20	**0.29**	**0.29**	
PDE⇒**ML**	0.27	**0.29**	**0.29**	
EQ⇒**PDE**	0.31	**0.33**	**0.33**	0.30
JDT⇒**PDE**	0.27	**0.39**	**0.38**	
LC⇒**PDE**	0.32	**0.37**	**0.37**	
ML⇒**PDE**	0.27	**0.37**	**0.37**	
Average	0.32	**0.41**	**<u>0.41</u>**	0.42

For some cross-prediction combinations, TCA+ does not outperform TCA. In the case of ReLink (Table X), the F-measure value of Apache⇒Safe with TCA (0.72) is the same as that with TCA+. Moreover, the F-measure of ZXing⇒Safe with TCA (0.70) is greater than that with TCA+ (0.64). However, TCA+ outperforms TCA for most combinations, and TCA+ outperforms Baseline for all combinations. In addition, we conducted the Wilcoxon matched-pairs test [42] to check if the differences are statistically significant. The p-value for ReLink is 0.03 and for AEEEM is $0.000 < 0.05$, which indicates that the differences between Baseline values and TCA+ values are statistically significant with 95% confidence (p-value < 0.05).

TCA+ outperforms Baseline for all combinations. The difference is statistically significant (p-value < 0.05*).*

Table X and XI also show within-project (Target⇒Target) performances as references. For example, the F-measure of the within-project prediction Apache⇒Apache for ReLink (Table X) is 0.64. The within-project performance is considerably better than that of cross-project prediction (Baseline) as confirmed in [12], [43]. For example, in Table XI, the average F-measure of EQ⇒EQ (0.58) is better than that of ML⇒EQ (0.24). However, the cross-project prediction performance with TCA+ is comparable to within-project prediction performance. For example, the F-measure of ML⇒EQ with TCA+ (0.56) is a bit lower (decreased by 0.02). In case of LC⇒EQ, the F-measure (0.62) is greater than the within-project result (0.58).

The performance of cross-project prediction with TCA+ is comparable to within-project prediction performance.

V. Threats to Validity

Systems are open-source projects. All systems examined in this paper were developed as open-source projects. Hence, they might not be representative of closed-source projects.

Experimental results might not be generalizable. We conducted our experiments using eight projects included in the ReLink and AEEEM data sets. It is possible that we accidentally selected systems that have better (or worse) than average cross-project defect prediction performance.

Decision rules in TCA+ might not be generalizable. The TCA+ rules were built by understanding normalization options and differences between source-target project data sets. We showed that these rules are effective for ReLink and AEEEM. However, these rules might not be generalizable to other data sets.

VI. Related Work

A. Defect Prediction

Software defect prediction is an active research area [2], [5], [28], [30], [31] in software engineering. Researchers have proposed new defect prediction algorithms and/or new metrics to predict defects effectively. Most of them predict defects through machine learning approaches [2], [5], [6], [44], [45]. However, most prediction models are evaluated in within-project prediction settings [2], [5], [6].

Recently, many researchers have studied cross-project defect prediction [12], [43], [46], [47], [48]. Zimmermann et al. [12] evaluated cross-project prediction performance for 12 projects and their 622 combinations. They found that current defect prediction models do not perform well for cross-project defect prediction. Turhan et al. [43] analyzed Cross-Company (CC) and Within-Company (WC) data for defect prediction, and they confirmed the difficulty of reusing CC data to predict defects in the software of other companies. Further, Turhan et al. applied nearest-neighbor (NN) filtering for CC data to select instances for better cross-project prediction [43]. Premraj and Herzig [46] compared network and code metrics for defect prediction and also built six cross-project defect prediction models using those metrics sets. Results in their study confirm that cross-project defect prediction is a challenging problem. However, Rahman et al.found that cross-project defect prediction works as well as within-project prediction in terms of cost effectiveness [47].

Ma et al. [48] proposed a novel cross-company defect prediction algorithm, Transfer Naive Bayes (TNB), which adapts Naive Bayes with weighting training data. However, TNB is designed only for Naive Bayes, since it tunes conditional probabilities for each instance in a source project.

Our approach transforms data of both source and target projects, and transformed data can be used for any machine learning algorithms to facilitate cross-project defect prediction. We showed that techniques for transforming data sets improve cross-project prediction performance without altering the existing prediction models.

B. Transfer Learning

Transfer learning has been attracting increasing attention in machine learning and data mining over the last six years [13]. Most machine learning approaches achieve good performance when training data and test data have the same feature space and distribution [13]. Learning models need to be rebuilt when the feature space and distribution change. In this case, it is necessary to recollect training data and label them again. Rebuilding a model is often expensive, and labeling new training data requires considerable effort. Transfer learning addresses these issues by transferring knowledge extracted from a related but different domain, which can be regarded as the source project in defect prediction, to build precise predictive models in the domain of interest, which can be regarded as the target project.

The problem setting of cross-project defect prediction is related to the domain adaptation setting in transfer learning [13]. Provided that sufficient labeled data are available in the source domain, the previous works in domain adaptation can be classified into two sub-settings: (1) a small amount of labeled data are available in the target domain [49], [50] or (2) only some unlabeled data are available in the target domain [15], [50], [51]. In this paper, we focus on the second sub-setting by assuming that in the target project, no defect information is available.

When only some unlabeled data are available in the target domain, the previous domain adaptation approaches can be classified as either feature-based or instance-based approaches [13]. Feature-based approaches assume that a so-called *good* feature representation exists across the source and target domains [52]. Based on this feature representation, the domain difference can be reduced and many existing models can be reused for cross-domain classification. Some methods have been proposed for uncovering such good feature representations. These include topic-bridged probability latent semantic analysis (TPLSA) [53], maximum mean discrepancy embedding (MMDE) [51], and TCA [18]. In this paper, we have applied TCA, which is a state-of-the-art feature-based transfer learning method, for cross-project defect prediction. Furthermore, we have proposed TCA+ by extending TCA to exhibit to better prediction performance for the defect prediction domain.

In terms of applications, transfer learning has been widely studied to address cross-domain problems in text classification [15], [53], [54], natural language processing [25], [50], WiFi-based indoor localization [55], and computer vision [56]. In this study, we adapted transfer learning for cross-project defect prediction.

VII. Conclusion

In this paper, we applied transfer learning approaches, TCA and TCA+, for cross-project defect prediction. TCA [18] is a state-of-the-art transfer learning technique proposed by the second author of this paper, and TCA+ is a novel approach to select suitable normalization options for TCA of a given source-target project pair. Our empirical study has shown that TCA+ significantly improves cross-project defect prediction performance.

Although transfer learning has been shown to be effective in other domains, we are the first to observe improved prediction performance by applying TCA for cross-project defect prediction. In addition, we proposed TCA+, which provides decision rules to select suitable normalization options based on a given cross-project prediction pair.

Transfer learning may benefit other prediction and recommendation systems [57], [58]. Furthermore, it is possible to transfer knowledge across domains. For example, we might gain knowledge from mailing lists [59] and then apply it to the bug report triage problem [60] after transferring the knowledge. We plan to explore other applications of transfer learning, including cross-domain knowledge transfer.

All data used in our experiments are publicly available at `https://sites.google.com/site/transferdefect/`.

Acknowledgments

Our thanks to Thomas Zimmermann, Qiang Yang, Bin Cao, and Dongyeop Kang for valuable feedback on our research. We are also grateful to Dongsun Kim, Jindae Kim, and Jaewoo Song for their detailed comments and helpful suggestions. This work is supported in part by the project COLAB, funded by Macau Science and Technology Development Fund.

References

[1] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of ICSE 2009*.

[2] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Softw. Eng.*, vol. 34, March 2008.

[3] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, January 2007.

[4] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of ICSE 2006*.

[5] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of ICSE 2008*.

[6] T. Lee, J. Nam, D. Han, S. Kim, and I. P. Hoh, "Micro interaction metrics for defect prediction," in *Proceedings of ESEC/FSE 2011*.

[7] 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 ICSE 2008*.

[8] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of ESEC/FSE 2011*.

[9] M. Kläs, F. Elberzhager, J. Münch, K. Hartjes, and O. von Graevemeyer, "Transparent combination of expert and measurement data for defect prediction: an industrial case study," in *Proceedings of ICSE 2010*.

[10] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proceedings of FSE 2008*.

[11] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *Proceedings of SIGIR 2002*.

[12] 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 ESEC/FSE 2009*.

[13] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, October 2010.

[14] W. Fan, I. Davidson, B. Zadrozny, and P. S. Yu, "An improved categorization of classifier's sensitivity on sample selection bias," in *Proceedings of ICDM 2005*.

[15] W. Dai, G. rong Xue, Q. Yang, and Y. Yu, "Transferring naive bayes classifiers for text classification," in *Proceedings of AAAI 2007*.

[16] J. Huang, A. Gretton, B. Schölkopf, A. J. Smola, and K. M. Borgwardt, "Correcting sample selection bias by unlabeled data," in *Proceedings of NIPS 2007*.

[17] M. Sugiyama, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanabe, "Direct importance estimation with model selection and its application to covariate shift adaptation," in *Proceedings of NIPS 2008*.

[18] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, Feb. 2011.

[19] J. Han, M. Kamber, and J. Pei, *Data mining : concepts and techniques*, 3rd ed. Waltham, Mass.: Elsevier/Morgan Kaufmann, 2012.

[20] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings of MSR 2010*.

[21] D. C. Montgomery, G. C. Runger, and N. F. Hubele, *Engineering statistics*, 2nd ed. New York: John Wiley, 2001.

[22] I. H. Witten and E. Frank, *Data mining : practical machine learning tools and techniques*, 2nd ed. Amsterdam: Morgan Kaufmann, 2005.

[23] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[24] B. Chen, W. Lam, I. W. Tsang, and T.-L. Wong, "Extracting discriminative concepts for domain adaptation in text mining," in *Proceedings of KDD 2009*.

[25] S. J. Pan, X. Ni, J.-T. Sun, Q. Yang, and Z. Chen, "Cross-domain sentiment classification via spectral feature alignment," in *Proceedings of WWW 2010*.

[26] A. B. A. Graf and S. Borer, "Normalization in support vector machines," in *Proceedings of DAGM 2001 Pattern Recognition*.

[27] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised learning," *Internaltional Journal of Computer Science*, vol. 1, 2006.

[28] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, April 2005.

[29] H. Zhang, "An investigation of the relationships between lines of code and defects," in *Proceedings of ICSM 2009*.

[30] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, October 1996.

[31] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Trans. Softw.*, vol. 22, no. 12, Dec. 1996.

[32] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and Balanced? Bias in Bug-Fix Datasets," in *Proceedings of ESEC/FSE 2009*.

[33] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of ICSE 2011*.

[34] Understand 2.0. [Online]. Available: {http://www.scitools.com/products/understand/}

[35] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *Proceedings of ICSM 2010*.

[36] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of FSE 2008*.

[37] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," in *Proceedings of ESEC/FSE 2011*.

[38] LIBLINEAR. [Online]. Available: {http://www.csie.ntu.edu.tw/∼cjlin/liblinear/}

[39] ICML 2008 large scale learning challenge. [Online]. Available: {http://largescale.ml.tu-berlin.de/summary/}

[40] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, June 2008.

[41] Z. Zheng, X. Wu, and R. Srihari, "Feature selection for text categorization on imbalanced data," *SIGKDD Explor. Newsl.*, vol. 6, June 2004.

[42] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, 1945.

[43] 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 Softw. Eng.*, vol. 14, October 2009.

[44] N. Fenton and M. Neil, "A critique of software defect prediction models," *Software Engineering, IEEE Transactions on*, vol. 25, no. 5, sep/oct 1999.

[45] S. Shivaji, E. J. W. Jr., R. Akella, and S. Kim, "Reducing features to improve bug prediction," *Automated Software Engineering, International Conference on*, vol. 0, 2009.

[46] R. Premraj and K. Herzig, "Network versus code metrics to predict defects: A replication study," in *Proceedings of ESEM 2011*.

[47] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of FSE 2012*.

[48] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technol.*, vol. 54, no. 3, Mar. 2012.

[49] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of ICML 2007*.

[50] J. Jiang and C. Zhai, "Instance weighting for domain adaptation in NLP," in *Proceedings of ACL 2007*.

[51] S. J. Pan, J. T. Kwok, and Q. Yang, "Transfer learning via dimensionality reduction," in *Proceedings of AAAI 2008*.

[52] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *Proceedings of NIPS 2007*.

[53] G.-R. Xue, W. Dai, Q. Yang, and Y. Yu, "Topic-bridged plsa for cross-domain text classification," in *Proceedings of SIGIR 2008*.

[54] R. Raina, A. Y. Ng, and D. Koller, "Constructing informative priors using transfer learning," in *Proceedings of ICML 2006*.

[55] Q. Yang, S. J. Pan, and V. W. Zheng, "Estimating location using wi-fi," *Intelligent Systems, IEEE*, vol. 23, no. 1, jan.-feb. 2008.

[56] L. Duan, I. Tsang, D. Xu, and S. Maybank, "Domain transfer svm for video concept detection," in *Proceedings of CVPR 2009*.

[57] Y. Shin, A. Meneely, L. Williams, and J. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *Software Engineering, IEEE Transactions on*, vol. 37, no. 6, nov.-dec. 2011.

[58] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of ESEC/FSE 2009*.

[59] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *Proceedings of MSR 2007*.

[60] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of ICSE 2006*.