Improving Cross-Project Defect Prediction with Weighted Software Modules via Transfer Learning

Yizhou Chen\(^1\) and Heng Dai\(^2\)

\(^1\)School of Computer Science, Wuhan University, Wuhan, Hubei, China
\(^2\)School of Mechanical and Electrical Engineering, Wuhan Qingchuan University, Wuhan, Hubei, China

Email: \(^1\)chenyizhou_whu_cs@163.com; \(^2\)daiheng726@163.com

Abstract. Cross-project defect prediction is investigated to resolve the trouble that software program defect prediction besides historic data. However, there are differences in distribution of software metrics of different software projects, which decreases the overall performance of cross-project defect prediction. This research presents a transfer learning-based cross-project fault prediction approach. The weights of the training software modules are determined by analogy with gravity and compared to those in the test set. The costs associated with different prediction errors are considered. Cost-sensitive C4.5 is employed on weighted training data for cross-project defect prediction. 10 projects from NASA are used in our experiments. Our suggested technique delivers promising cross-project defect prediction results, according to the results, 0.81, 0.41 and 0.8 in average PD value, F-measure value and AUC value, which are best compared to Naïve Bayes based cross-project defect prediction.

Keywords. Software defect prediction; transfer learning; cross-project; cost-sensitive.

1. Introduction

Nowadays more and more activities rely on software. The reliability of software has become a crucial issue. Software defects are errors leading to unexpected results, which reduces the software quality. The In order to make sure that software program applications work properly, software defect prediction is indispensable [1].

After examining a range of literature, we found that many people have proposed a variety of software defect prediction studies [2-6]. Software metrics contain important information about software, which can be employed in software defect prediction [7]. Most software program defect prediction algorithms are primarily based on supervised gaining knowledge of models. Data for software defect prediction models are trained on software metrics and error-prone labels from historical data of software projects. The trained models are then employed to predict software modules which are waiting for labelling.

However, for software projects with limited historical data, supervised learning-based software defect prediction is hardly effective. There is no historical data for newly developed software projects. Cross-project software defect prediction was proposed to solve that problem. Defect prediction fashions are educated based totally on labeled facts from different software program projects. There are differences in distribution of software metrics of different software projects, which hinders the defect prediction across projects [8].

We propose to use transfer learning to forecast cross-project defects. Software measurements from the training set are compared to metrics from the test set (target software modules). The weight of
training software modules is then determined using a gravity analogy. The costs associated with different prediction errors are considered. Finally, cost-sensitive C4.5 is employed on weighted training data for defect prediction across projects.

We conduct tests on 10 public projects from the NASA dataset to assess the efficacy of deficiency prediction across projects based on transfer learning. It also outperformed the Nave Bayes and C4.5 based cross-item defect prediction models on the testing data. Specifically, the average PD value was 0.81, the F-measure value was 0.41, and the AUC value was 0.8.

2. Related Work
Thomas et al. [9] performed a large-scale investigation on defect prediction across projects, and identified a number of parameters that did impact the success of defect prediction across projects. Turhan et al. [10] used a NN-filter to trim data in order to anticipate cross-project defects. Irrelevant data was eliminated by picking comparable data based on accessible software metrics and eliminating non-similar data. Cost-sensitive cross-project software prediction works as effective as intra-project prediction and outperforms the random model, according to Rahman et al. [11]. He et al. [12] supplied an education statistics choice method in the prediction of defects across project, which uses factual similarity measures and characteristic subset choice to put off needless and unstable factors of the education data. Their method produces high-quality defect predictors that are similar and scalable across projects, allowing them to solve the challenge of defect prediction. He et al. [13] have done similar work. They also presented a strategy for selecting training data. The method incorporates the distribution characteristics of the data to select the training data. Its prediction results are equivalent to those obtained from the same project's training data. For cross-project software flaws, Ma et al. [14] presented the Transfer Naïve Bayes (TNB) technique, which transfers records from the target statistics to the supply data. The metric information of the target software modules is compared with the collected information by computing the metric information. The similarity between the training software modules is calculated. Based on the similarity, weights are assigned to each training software module, and the TNB technique is then applied to the weighted training software modules. For defect prediction across projects, Jaechang et al. [15] used Transfer Component Analysis (TCA), which makes the software metrics distributions in training and target projects identical. TCA+ enhances defect prediction across projects performance, according to their research. Lin et al. [16] presented the Double Transfer Boosting (DTB) method. The distribution similarity of cross-project software modules is increased by data gravitation re-weights, and the negative software metrics are removed by transfer boosting.

3. Method
In this part, we propose a solution for transfer learning-based defect prediction across projects. The idea of our method is to weight training software modules based on how comparable the source and goal software metrics are. And then, C4.5 decision tree algorithm is employed on these weighted training software modules.

3.1. Cost-Sensitive C4.5 Defect Prediction Method
Let $S = \{(s_1, c_1), (s_2, c_2), ...,(s_m, c_m)\}$ be the training software modules in the defect prediction across projects issue, and $m$ be the number of software modules in the training set, class $s_i$ of the software module is denoted by $c_i$, $c_i \in (0, 1)$, mark defective software modules as 1, and software modules with no defects are marked as 0. Let $T = \{t_1, t_2, ..., t_n\}$ represent the target software modules, and in the target modules, the number of software modules is denoted as $n$. In C4.5, the software modules can be labeled according to maximizing $E(C_0) - (E(C_1) + E(C_2))$. $C_0$ is the software modules before splitting, $C_1$ and $C_2$ are software modules after splitting. The entropy of splitting software modules into $c$ classes is defined as follows:

$$E(C) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$
where \( p_i = \frac{n_i}{n} \) is the probability of software modules which belong to class \( C_i \).

To consider the costs associated with different forecasting errors, we define the cost matrix \( L \) as follows in table 1:

| Actual class | Predicted class |
|--------------|-----------------|
| 0            | 0               | 1               |
| 1            | \( \alpha \)    | 0               |

And we define the cost function as follows:

\[
\begin{align*}
COS(t) &= COS(t, 0) + COS(t, 1) \\
COS(t, 0) &= p_0L_{00} + p_1L_{10} = \alpha p_1 \\
COS(t, 1) &= p_0L_{01} + p_1L_{11} = p_0
\end{align*}
\]

where \( p_0 \) and \( p_1 \) are the probability of software module \( t \) predicted as non-defective and defective, respectively. \( L_{ij} \) is the number of cost matrix \( L \) in row \( i \) and column \( j \). Then the software modules are labeled according to minimizing \( COS(t) = COS(t, 0) + COS(t, 1) \) in C4.5. Actual faulty software modules that are projected to be non-faulty cost more than actual non-defective software modules that are projected to be defective. Thus, the value of \( \alpha \) is specified larger than 1 to pay more attention to false negative misclassification.

### 3.2. Transfer Learning Based Defect Prediction

In order to use transfer learning to anticipate software defects across projects, software metrics from the training set of the target software module are used to compare with the test set metrics. Then, we calculate weights for training software modules by analogy with gravity. Finally, cost-sensitive C4.5 is employed for defect prediction across projects on weighted training data.

In target software modules, \( t_i = \{x_{i1}, x_{i2}, ..., x_{ij}\} \), \( x_{ij} \) is the \( j \)th software metric of \( t_i \). The maximum and minimum values of software index \( J \) are calculated on the test set to obtain the information of the target software module. Details are as follows:

\[
\begin{align*}
max_j &= \max \{x_{ij}, x_{2j}, ..., x_{nj}\} \\
min_j &= \min \{x_{1j}, x_{2j}, ..., x_{nj}\}
\end{align*}
\]

where \( j = 1, 2, ..., k \), \( n \) is the number of software modules in the target collection, and \( k \) is the number of software metrics. Then, we have \( MAX = \{max_1, max_2, ..., max_k\} \) and \( MIN = \{min_1, min_2, ..., min_k\} \), which contain the information of test set.

The similarity between the training and test sets is calculated by computing the placement of the metrics between the maximal and minimal values for the training software module. In training software modules, \( s_i = \{y_{i1}, y_{i2}, ..., y_{ij}\} \), \( y_{ij} \) is the \( j \)th software metric of \( s_i \). We calculate the similarity of metrics as follows:

\[
sim_i = \sum_{j=1}^{k} f(y_{ij})
\]

where \( f(y_{ij}) = \begin{cases} 1 & \text{if } min_j \leq y_{ij} \leq max_j, \ y_{ij} \text{ is the } j \text{th software metric of } s_i. \\ 0 & \text{otherwise} \end{cases} \)

Li et al. [17] introduced data gravitation based classification. We follow their work for weighing training software modules. The gravitational pressure is proportional to the product of the two hundreds and inversely proportional to the rectangular of the distance between them, in accordance to Newton’s familiar gravitation law:
\[ F = G \frac{m_1 m_2}{r^2} \]

where \( G \) is the universal gravitational constant, \( m_1 \) is the mass of one of the items, \( m_2 \) is the mass of the second item, \( r \) is the distance between the center of masses of the two items, and \( F \) is the attraction force between the two items.

**Algorithm 1.** Transfer learning based defect prediction across projects

**Input:** training software modules \( S \), target software modules \( T \)

1. calculate the MAX, MIN of \( T \)
2. for \( s_i \) in \( S \)
3. calculate \( w_i \) according to Eq.(9)
4. end for
5. cost-sensitive C4.5
6. for \( t_i \) in \( T \)
7. predict \( t_i \) according to Eq.(2)
8. end for

We calculated the gravitational force between the test set and the training software module. Suppose the mass of one software metric is \( M \), then the mass of the test set is \( kM \) and the mass of each training software module is \( \text{sim}_i M \). Thus, the weight \( w_i \) of the training software module \( s_i \) is proportional to \( k\text{sim}_i M^2 \) and inversely proportional to \( r_i^2 = (k - \text{sim}_i + 1)^2 \). In the gravity formula, these values are the product of the two masses and the square of the distance, respectively. As a result, the following is how metrics are weighted:

\[ w_i = \frac{m_1 m_2}{r_i^2} = \frac{k\text{sim}_i M^2}{(k - \text{sim}_i + 1)^2} \propto \frac{\text{sim}_i}{(k - \text{sim}_i + 1)^2} \]

Based on this formula, we can calculate the weight \( w_i \). Its weight is positively correlated with the similarity between software module \( s_i \) and test set.

The transfer learning based defect prediction across projects is presented in Algorithm 1.

**4. Experiment Setup**

**4.1. Benchmark Dataset**

In our experiments, we employ 10 publicly available projects from NASA dataset. We transform the number of defect of software modules into the binary label, which means that the modules without defect are labeled as 0, others are labeled as 1. Table 2 shows the detailed description of each project, where \# of modules represents the quantity of software modules, \# of defective represents the quantity of defective modules, \% of defective represents the percentage of defective software modules among all modules (in percent).

**4.2. Evaluation Indicators**

In our study, software defect prediction is a dichotomous task, which labels the software modules as defective or not. There are four typical outputs for software defect prediction as follows: The number of fault-prone modules that have been designated as faulty is known as True Positive (TP); True Negative (TN) signifies the number of non-faulty modules; False Positive (FP) means the number of non-defective modules identified as defective; and False Negative (FN) means the number of fault-prone modules categorized as non-defective [18, 19].

The precision is defined as the percentage of fault-prone software modules marked as defective to all software modules marked as defective.

\[ \text{Precision} = \frac{TP}{TP + FP} \]
Table 2. Description of the benchmark dataset.

| Project | # of modules | # of defective | % of defective |
|---------|--------------|----------------|---------------|
| KC1     | 1162         | 294            | 25.3%         |
| KC2     | 325          | 92             | 28.31%        |
| KC3     | 324          | 42             | 12.96%        |
| MC1     | 1952         | 35             | 1.79%         |
| MC2     | 155          | 51             | 32.9%         |
| MW1     | 375          | 28             | 7.47%         |
| PC1     | 919          | 60             | 6.53%         |
| PC2     | 1409         | 148            | 10.5%         |
| PC3     | 1270         | 177            | 13.94%        |
| PC4     | 1694         | 457            | 26.98%        |

The recall represents the percentage of fault-prone software modules marked as defective to all fault-prone software modules. In addition, Probability of Detection (PD) is equivalent to recall.

Recall = PD = \( \frac{TP}{TP+FN} \)

A model that has both high-level of accuracy and recall is considered a high performance defect prediction model. However, the increase of the precision value usually results in the decrease of the recall value, and vice versa. Therefore, we employ F-measure to balance the precision and recall.

The advantage of the evaluation metric F-measure is that it takes into account both precision and recall, which is defined as

F-measure = \( \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \)

Area The location below a Receiver Operating Characteristic (ROC) curve depicting the relative trade-off between PD (the y-axis) and PF (the x-axis) is calculated the use of the receiver running attribute curve (AUC) (the x-axis). As a result, the AUC fee represents the possibility that an inaccurate software program module would rank greater than a non-defective software program module when picked at random. Baljinder et al. [20] suggests to use AUC cost for higher cross-project comparability.

5. Experiment Results

**Method:** Each project in NASA dataset is regarded as target software modules, and all other projects are regarded as training software modules. We select Naïve Bayes (NB) and C4.5 to compare with our proposed method (cost-sensitive C4.5). To consider the overall performance of these processes for defect prediction across projects, we compute their PD values, F-measure values, and AUC values.

**Results:** The exact PD values of three methods on each project are presented in table.

According to table 3, we observe that our method and NB achieve a similar performance for defect prediction across projects. In particular, the average PD values of two methods are same, which are 0.81. C4.5 has a poor track record for defect prediction across projects, with an average PD value of 0.29. In six projects that are considered target tests, our technique beats NB for defect prediction across projects. In all NASA dataset projects, our technique beats C4.5 for defect prediction across projects.

In table 3, we observe that our method achieves the best performance for defect prediction across projects. In particular, the average F-measure values of three methods are 0.38, 0.33, 0.41, respectively. Our method outperforms NB for defect prediction across projects in most projects except KC2 project. Our method outperforms C4.5 for defect prediction across projects in all projects of NASA dataset. In summary, our proposed approach performs the quality amongst all studied methods.
Table 3. PD, F-measure and AUC values of three methods for defect prediction across projects (bold font indicate the best performance).

| project | PD | F-measure | AUC |
|---------|----|-----------|-----|
| NB C4.5 Cost-sensitive C4.5 NB C4.5 Cost-sensitive C4.5 NB C4.5 Cost-sensitive C4.5 |
| KC1     | 0.86 0.34 | 0.72 0.46 0.41 | 0.46 0.68 0.62 | 0.71 |
| KC2     | 0.88 0.46 | 0.82 0.6 0.5 | 0.59 0.83 0.68 | 0.83 |
| KC3     | 0.88 0.19 | 0.76 0.34 0.24 | 0.37 0.75 0.62 | 0.75 |
| MC1     | 0.75 0.03 | 0.83 0.07 0.04 | 0.1 0.71 0.54 | 0.83 |
| MC2     | 0.73 0.37 | 0.82 0.54 0.4 | 0.55 0.68 0.56 | 0.71 |
| MW1     | 0.7 0.18 | 0.71 0.25 0.23 | 0.29 0.77 0.49 | 0.78 |
| PC1     | 0.73 0.17 | 0.82 0.24 0.24 | 0.27 0.77 0.59 | 0.84 |
| PC3     | 0.85 0.2  | 0.77 0.33 0.25 | 0.36 0.78 0.61 | 0.8 |
| PC4     | 0.85 0.48 | 0.96 0.48 0.51 | 0.53 0.85 0.79 | 0.91 |
| PC5     | 0.83 0.47 | 0.85 0.53 0.5 | 0.56 0.74 0.66 | 0.79 |
| Average | 0.81 0.29 | 0.81 0.38 0.33 | 0.41 0.76 0.62 | 0.8 |

In phrases of defect prediction across projects, we discover that our method performs the best. In particular, the common AUC values of three techniques are 0.76, 0.62, 0.8, respectively. Our approach outperforms NB and C4.5 for defect prediction across projects in all initiatives of NASA dataset. The AUC of 0.7 is regarded as promising prediction overall performance in [21]. In summary, our proposed technique performs the nice amongst all studied methods.

Real faulty software modules that are projected to be non-faulty cost more than actual non-defective software modules that are projected to be defective. Thus, the value of \( \alpha \) is specified larger than 1 to pay more attention to false negative misclassification. We use different values of \( \alpha \), and plot probability of false alarm versus recall for each \( \alpha \) value. Figure 1 depicts these numbers, with the PC4 project serving as the target project for defect prediction across projects.

We observe that if we do not consider costs, which means \( \alpha = 1 \), then the recall is about 50% with the probability of false alarm of 3%. By considering cost using cost factors larger than 1, we raise the recall to its top limits while concurrently lowering the chance of false alert. We found that a cost factor around \( \alpha = 50 \) is a reasonable choice for PC4 project regarded as target project in defect prediction across projects.

![Figure 1. Recall and probability of false alarm for different cost factor \( \alpha \).](image-url)

6. Conclusion
Defect prediction across projects is researched to tackle the hassle of a lack of historic information for newly constructed software program defect prediction.
To address the disparities in software metric distribution across training and target projects, we proposed to use transfer learning-based defect prediction across projects. We first compare the software metrics of the training set with those of the test set. By analogy with gravity, the weight of training software module is calculated. The costs associated with different prediction errors are considered. Cost-sensitive C4.5 is employed on weighted training data for defect prediction across projects.

We evaluated our proposed approach on 10 open projects in the NASA dataset. The method achieved better performance than Naïve Bayes for defect prediction across projects, 0.81, 0.41 and 0.8 in average PD value, F-measure value and AUC value.

References
[1] Nor Shahida Mohamad Y, John G, Jean-Guy S and Rajesh V 2020 A revised open source usability defect classification taxonomy Information and Software Technology (128) 106396.
[2] Miroslaw O, Miroslaw S and Wilhelm M 2019 Simsax: A measure of project similarity based on symbolic approximation method and software defect inflow Inf. Softw. Technol. (115) 131-147.
[3] Yu X, Liu J, Keung J W, Li Q, Bennin K E, Xu Z, Wang J and Cui X 2020 Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM IEEE Transactions on Reliability (69) 139-153.
[4] Li W, Zhang W, Jia X, Huang Z 2020 Effort-Aware semi-Supervised just-in-Time defect prediction Inf. Softw. Technol (126) 106364.
[5] Chen X, Zhang D, Zhao Y, Cui Z, Ni C 2019 Software defect number prediction: Unsupervised vs supervised methods Inf. Softw. Technol. (106) 161-181.
[6] Soumaya R, Marouane K, Hanzhang W, Bruce R 2020 Web service design defects detection: A bi-level multi-objective approach Inf. Softw. Technol. (121) 106255.
[7] Thomas S, David B, Tracy H 2019 Automatically identifying code features for software defect prediction: Using AST N-grams Inf. Softw. Technol. (106) 142-160.
[8] Li N, Martin S, Guo Y 2020 A systematic review of unsupervised learning techniques for software defect prediction Inf. Softw. Technol. (122) 106287.
[9] Thomas Z, Nachiappan N, Harald G, Emanuel G and Brendan M 2009 Cross-project defect prediction: A large scale experiment on data vs. domain vs. process Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering pp 91-100.
[10] Turhan, B, Tim M, Ayşe B, Justin D 2009 On the relative value of cross-company and within-company data for defect prediction Empirical Software Engineering (Springer) 14 (5) 540-578.
[11] Rahman F, Daryl P and Premkumar D 2012 Recalling the “imprecision” of cross-project defect prediction Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering pp 1-11.
[12] He Z, Fayola P, Tim M and Ye Y 2013 Learning from open-source projects: An empirical study on defect prediction ACM/IEEE International Symposium on Empirical Software Engineering and Measurement pp 45-54.
[13] He Z, Shu F, Ye Y, Li M and Wang Q 2012 An investigation on the feasibility of cross-project defect prediction Automated Software Engineering 19 (2) 167-199.
[14] Ma Y, Luo G, Xue Z and Chen A 2012 Transfer learning for cross-company software defect prediction Information and Software Technology 54 (3) 248-256.
[15] Nam J, Pan S J and Kim S 2013 Transfer defect learning 35th International Conference on Software Engineering (ICSE) pp 382-391.
[16] Lin C, Bin F, Shang Z and Tang Y 2015 Negative samples reduction in cross-company software defects prediction Information and Software Technology 62 67-77.
[17] Li Z, Bo Y, Chen Y and Ajith A 2009 Data gravitation based classification Information Sciences 179 (6) 809-819.
[18] Xu Z, Liu J, Luo X, Yang Z, Zhang Y, Yuan P, Tang Y and Zhang T 2019 Software defect prediction based on kernel PCA and weighted extreme learning machine Inf. Softw. Technol. (106) 182-200.

[19] Zhou T, Sun X, Xia X, Liu B and Chen X 2019 Improving defect prediction with deep forest Inf. Softw. Technol. (114) 204-216.

[20] Baljinder G, Shane M and Ahmed E 2015 Revisiting the impact of classification techniques on the performance of defect prediction models IEEE/ACM 37th IEEE International Conference on Software Engineering pp 789-800.

[21] Emanuel G, Marco D, Martin P and Harald C 2012 Method-level bug prediction Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement pp 171-180.