Hindawi
Scientific Programming
Volume 2022, Article ID 6339684, 28 pages
https://doi.org/10.1155/2022/6339684

Research Article

Tsbagging: A Novel Cross-Project Software Defect Prediction Algorithm Based on Semisupervised Clustering

Shiqi Tang,[1] Song Huang,[1] ErHu Liu,[1] YongMing Yao,[1] KaiShun Wu,[1] and Haijin Ji[2]

[1]*Command and Control Engineering College, Army Engineering University of PLA, Nanjing, JiangSu 210007, China*
[2]*School of Computer Science and Technology, Huaiyin Normal University, Huaian, JiangSu, China*

Correspondence should be addressed to Song Huang; hs0317@163.com

Received 16 November 2021; Revised 13 April 2022; Accepted 26 April 2022; Published 28 September 2022

Academic Editor: Jiwei Huang

Software defect prediction (SDP) is an important technology which is widely applied to improve software quality and reduce development costs. It is difficult to train the SDP model when software to be test only has limited historical data. Cross-project defect prediction (CPDP) has been proposed to solve this problem by using source project data to train the defect prediction model. Most of CPDP methods build defect prediction models based on the similarity of feature space or data distance between different projects. However, when the target project has a small amount of label data, these methods usually do not consider this part of data information. Therefore, when the distribution between source project and target project is quite different, these methods are difficult to achieve good prediction performance. To solve this problem, this paper proposes a CPDP method based on a semisupervised clustering (namely, Tsbagging). Tsbagging has two stages; in the first stage, we cluster to the source project data based on the limited labeled data in the target project and assign different weights to these source project data according to the clustering results. In the second stage, we use bagging method to train the prediction model based on the weight assigned in the first stage. The experimental results show that the performance achieved by Tsbagging is better than other existing SDP methods.

1. Introduction

SDP is a very important technology in the software testing stage. It can quickly predict defects and provide guidance for allocating test resources and manpower in the early stage of software development [1, 2]. At present, most SDP methods use machine learning technology to build defect prediction models [3–5]. For example, Lessmann et al. [6] and Shepperd et al. [7] used traditional machine learning algorithms, such as decision tree, naive Bayes (NB), neural network, and support vector machine (SVM) to SDP, and achieved good results. Elish et al. [8] compared 8 machine learning methods on the NASA dataset, and the results showed that SVM was better than other algorithms. In addition, researchers also improved the traditional machine learning methods based on the characteristics of SDP. For example, Wang et al. [9] proposed a Compressed C4.5 Models (CCM) based on C4.5 decision tree by using Spearman rank correlation coefficient. Ji et al. [10] applied the Kolmogorov–Smirnov test on the datasets from the promise database and found that feature space of those datasets was not normally distributed. Therefore, they improved the NB and proposed an SDP method based on kernel density estimation. Moreover, Ji et al. [11] proposed a weighted naive Bayes method (WNB). WNB used the information diffusion model instead of the default probability density function of normal distribution to calculate the probability density of each feature. This method also achieved good performance. Li et al. [12] proposed CoForest and ACoForest. CoForest is a sampling method based on semisupervised learning. On the basis of random forest classification, this method can find optimal sampling example by random sampling. ACoForest further enhanced the prediction performance of CoForest base on active learning. Studies demonstrated that these two methods showed better performance than other traditional machine learning methods. Bejjanki et al. [13] proposed a new class imbalance reduction algorithm (CIR) for the class imbalance problem in defect prediction dataset. This method

made the defect data and nondefect data in unbalanced dataset symmetrical by considering the distribution characteristics of the unbalanced dataset. Experiments showed that CIR has better defect prediction performance than traditional methods.

Most of the above research studies have focused on within project defect prediction (WPDP). However, sometimes, few local training data can be available in the target software in which defects must be detected, because past local defective modules are expensive to be labeled for the module development in an unfamiliar domain [4, 14]. To solve this problem, the CPDP method used the source project data training model to predict defects in the target project. Zimmermann et al. [15] constructed 622 cross-project task combinations from 12 projects to evaluate CPDP model performance. The experimental results showed that traditional defect prediction methods have difficulty to achieving good CPDP performance. It is because that these methods assumed that the training and test data have similar distributions, whereas data from different projects have different distributions. Ma et al. [16] assigned different weights to the data in the source project by calculating the similarity between those data and the target project and then trained a weighted Bayes classifier (namely, TNB) according to these weights. Nam et al. [17] proposed the TCA+, TCA + used the transfer component analysis technology to find the potential feature space of different project data, when the potential space is determined, and it mapped the source project and target project to the space to eliminate the difference of feature distribution between different projects. Turhan et al. [18] proposed a CPDP method based on nearest neighbors (namely NN). NN constructed a training dataset by selecting nearest neighbors with target project data in source project and use the training dataset to train the defect prediction model. Kawata et al. [19] proposed a CPDP method based on DBSCAN [20], they use DBSCAN to find subclusters and then select subclusters which have at least one record of the target project data to build and train the dataset. Ryu et al. [21] proposed a CPDP method (VCB + SVM). Firstly, VCB + SVM calculates the similarity weight of each instance in the source project based on the dataset of the target project and then constructs the defect prediction model based on the support vector machine and boosting method. Chen et al. [22] proposed a CPDP method collective transfer defect prediction (CTDP) based on multisource transfer learning. Firstly, CTDP extended the source project dataset by using different normalization and TCA. Then, they built several base classifiers on the extended source project dataset. According to the contribution of each classifier to the target project, CTDP uses the particle swarm optimization algorithm to adaptively weight these base classifiers to construct an ensemble classifier. Finally, CTDP uses the ensemble classifier to predict defect. He et al. [23] proposed a CPDP method on multisource transfer learning (FSS + bagging). FSS + bagging has three steps. Firstly, FSS + bagging calculated the similarity between the candidate source project and target project and selected the first k most similar source project. Next, for each source project which be selected, FSS + bagging removed some unstable features to reduce the data distribution difference between source project and target project. Finally, FSS + bagging used bagging to get the final defect prediction results. To solve the class imbalance in CPDP, Limsettho et al. [24] proposed a new oversampling method CDE-SMOTE, which used CDE to estimate the class distribution of the target project and uses SMOTE to modify the class distribution of training data until it is similar to the distribution of target project. Although the above methods have achieved good performance in CPDP, most of them build CPDP models based on the similarity of feature space or data distance between source project and target project. However, when the target project has limited labeled data, those methods often do not take into account this part of label information. Turhan et al. [25] found that mixing a limit of target project data and source project data which are selected by the NN method to train classifier can significantly improve the performance of CPDP. However, this method also does not consider the limited label information when selecting the source project data. Therefore, when the data distribution between the target project and the source project was very different (especially, when concept shift occurs), those method are difficult to achieve good performance. Chen et al. [26] propose a two-stage transfer learning method, double transfer boosting (DTB), to solve this problem. In the first stage, DTB uses the data gravitation method proposed by Ma et al. [16] to assign weights to the source project data; in the second stage, DTB uses Tradaboost [27] to build classifier based on the weight assigned in the first stage. However, DTB does not consider the impact of concept shift when assigning weights, which affects the final defect prediction performance. This paper proposes a CPDP method based on semisupervised clustering (namely, Tsbagging). In Tsbagging, we firstly proposed a semisupervised clustering algorithm (namely, TSCluster) to clustering the source project data based on limited target project data. Next, we trained several base classifiers based on the clustering results and integrated these classifiers by bagging to predict defect. We compare the performance of Tsbagging with other SDP methods on 42 defect prediction datasets.The experimental results show that Tsbagging has better performance than other methods.

2. Methodology

2.1. Nearest Neighbor Filter. Since the CPDP method can use source project data to help train the defect prediction model when target project only has limited software historical data, it has always been the research focus in the field of SDP. However, an important problem in CPDP is how to select data similar to target project from source project. The NN (nearest neighbor filter) is a commonly used technology for selecting data from source project [18, 25]. The specific steps are as follows: firstly, for each data in the target project, NN select K-nearest neighbors of it from the source project based on the Euclidean distance. If there are n data in the target project, the number of finally selected nearest neighbors is $k \times n$. Then, NN removes duplicate data in the selected nearest neighbor data to build the final train dataset. Finally, NN uses the final train dataset to train the SDP model. Moreover, Turhan found that the NN + WP (WP means within project) which is trained by the limited target data and source data selected by NN can achieve better

performance than NN. However, both NN and NN + WP only select data in the source project based on the similarity between the data distances of different projects. When the data distribution between target project and source project is very different, especially in the case of concept shift, it is difficult for NN to accurately select the data similar to the target project from the source project.

2.2. Concept Shift. Concept shift is also called concept drift [28]. According to Moreno-Torres's theory [28], the concept shift is defined as: $P^S(x) = P^T(x)$, $P^S(y \mid x)$ and $P^T(y \mid x)$ are different. Where $P^S(y \mid x)$ and $P^T(y \mid x)$ represent the posteriori probabilities in the source project and the target project, respectively. $P^S(x)$ and $P^T(x)$ represent the marginal distribution of source and target projects, respectively. Moreno-Torres considers that nonstationary environments can lead to concept shift, and the relationship between the input and class variables has changes. Kanayake at al. [29] finds the concept shift can occur in SDP and influence the defect prediction quality. In CPDP, different development environments or application fields can lead to the concept shift. When concept shift occurs, due to $P^S(y \mid x)$ and $P^T(y \mid x)$ are different, even if the distance between source project data and target project data is close, their label maybe very different. Therefore, when the concept shift occurs, the NN method is difficult to accurately select the appropriate data from source project. As shown in Figure 1, all data are distributed in four areas A, B, C, and D, in which circular points represent nondefect data, triangular points represent defect data, red points represent target project data, and blue points represent source project data. As a result of concept shift, it is found that the target project's label is different to its nearest neighbors in areas A and D. If NN is used to select data from the source project, the data with large difference from the distribution of the target project will be selected.

2.3. Semisupervised Clustering. Semisupervised clustering can use additional information to achieve better clustering performance than unsupervised clustering. At present, there are mainly two types of semisupervised clustering methods. One is based on connected and unconnected constraints. The connected constraints means the data belong to the same cluster, and the unconnected constraints means the data do not belong to the same cluster. For example, Wagstaff et al. [30] proposed the constrained k-means algorithm. This algorithm assigns appropriate clusters to each data by checking whether the data violate the given connected and unconnected constraint pairs in the clustering process of k-means. The other type of the semisupervised clustering method is based on the given cluster label information. This method helps clustering by taking the given cluster label as a constraint seed. For example, Bsau et al. [31] proposed the constrained seed k-means method, and this method initializes the cluster center of k-means according to the given cluster label information and did not change the cluster membership relationship of seed samples in the clustering. In this paper, we use a limit of target project data as constraint seeds to cluster the source project data selected by the NN method and exclude the data with large differences with the target project data for achieving better prediction performance.

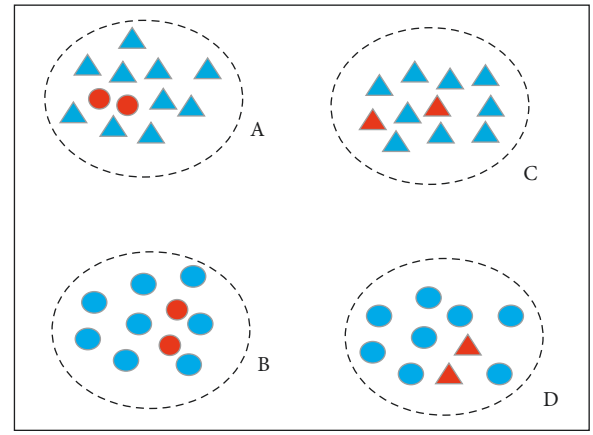

Figure 1: Schematic diagram of concept shift.

3. Tsbagging

In this section, we propose a CPDP method, namely Tsbagging. Tsbagging mainly consists of clustering stage and ensemble stage. In the clustering stage, we propose a semisupervised method (TSCluster) to clustering the data of the source project and assign different weights to these data according to the clustering results. In the ensemble stage, based on the weights given in the clustering stage, we use bagging [32] to train several classifiers and integrate them to predict the defect. In the following sections, we describe Tsbagging in details.

3.1. Problem Description. In this paper, we define three sets: $\mathbf{D}_S = \{(\mathbf{x}_i^S, \ldots, \mathbf{y}_i^S) | i = 1, 2, \ldots, Sn\}$, $\mathbf{D}_{\text{WP}} = \{(\mathbf{x}_j^{\text{WC}}, \mathbf{y}_j^{\text{WC}}) | j = 1, 2, \ldots, Wn\}$, and $\mathbf{D}_{\text{Test}} = \{(\mathbf{x}_t^{\text{Test}}) | t = 1, 2, \ldots, Tn\}$, where $\mathbf{D}_S$ represents the source project dataset,$\mathbf{D}_{\text{WP}}$ represents the labeled dataset in the target project (WP means within project), and $\mathbf{D}_{\text{Test}}$ represents the dataset to be tested in the target project. Note that the data from $\mathbf{D}_{\text{test}}$ and $\mathbf{D}_{\text{WP}}$ belong to the same distribution, while $\mathbf{D}_S$ belongs to different distribution from $\mathbf{D}_{\text{test}}$ and $\mathbf{D}_{\text{WC}}$. $\mathbf{x}_i^S$, $\mathbf{x}_j^{\text{WC}}$, and $\mathbf{x}_t^{\text{Test}}$ are m-dimensional vectors. Each dimension in the vector represents a measured value of a metric to a program module. $\mathbf{y}_i^S$ and $\mathbf{y}_j^{\text{WC}} \in \{-1, 1\}$, -1 represents that a module is a nondefect proneness module, 1 means that a module is a defect proneness module, Sn, Wn, and Tn represent the data amount of $\mathbf{D}_S$, $\mathbf{D}_{\text{WP}}$, and $\mathbf{D}_{\text{Test}}$, respectively, and $Wn \ll Tn$.

3.2. Overall Flow of Tsbagging. The overall process of Tsbagging is shown in Figure 2. Tsbagging consists of clustering stage and ensemble stage. In the clustering stage, firstly, we use the NN filter to select the candidate dataset (namely, FCCData) from $\mathbf{D}_S$, and then we propose a two-step clustering method. In the first clustering, we use the k-means to cluster the data from different class in $\mathbf{D}_{\text{WP}}$, respectively, so that in the final clustering result, the data in

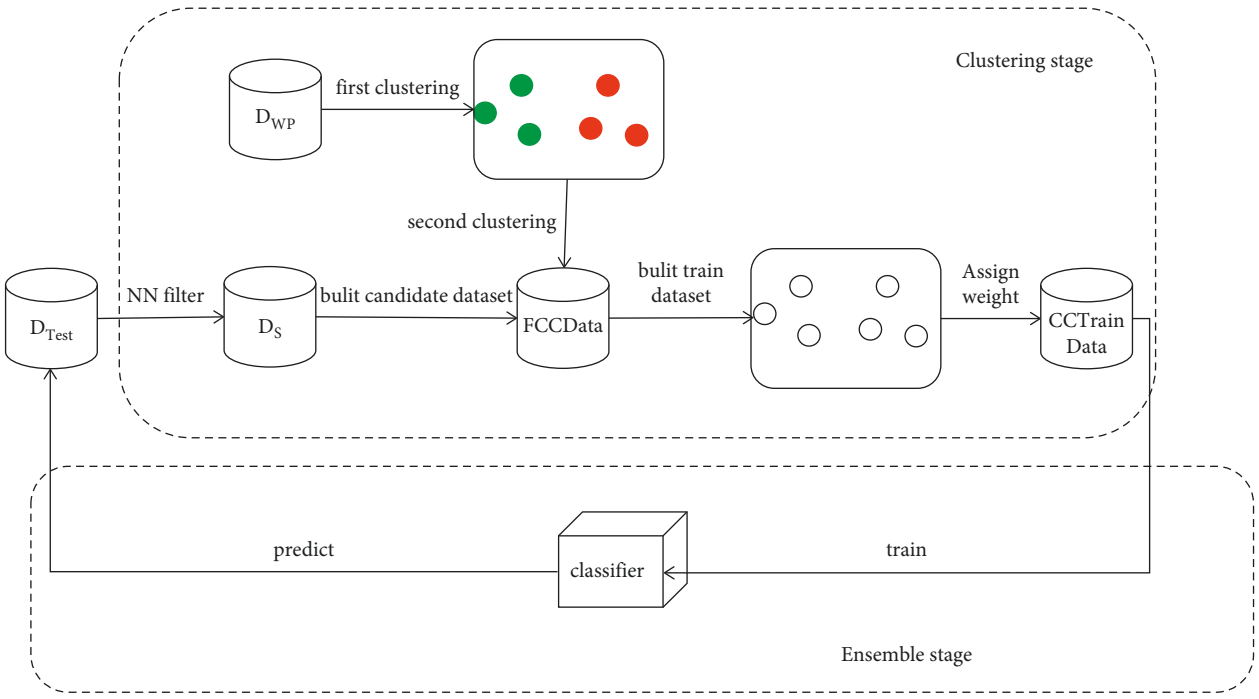

Figure 2: Overall flow of Tsbagging.

each cluster belong to the same class, and this class label is regarded as the cluster label. For example, in Figure 2, the result of the first cluster contains six clusters. A red circle indicates that all the data in the cluster are defective data, and a green circle indicates that all the data in the cluster are nondefective data. In the second clustering, we use the semisupervised method based on the given clustering label information to cluster FCCData. In this clustering process, we take the clusters gathered by the first clustering as constraint seed, and we do not change the cluster membership of data from $\mathbf{D}_{WP}$ in this cluster. For example, in Figure 2, there are still six clusters in the second result. After the second clustering, we assign weights to data in each cluster based on the clustering hypothesis to build the final training dataset (namely, CCTrainData). The clustering hypothesis is that if all data in the dataset obey the same distribution, data from same cluster are more likely to have the same label. Conversely, if the label of data from the same cluster is different, they do not obey the same distribution. So, the idea of assign weight as follows: in each cluster, if the label of a data from $\mathbf{D}_S$ is the same as the cluster label, we believe that the data obey the clustering hypothesis; that is, the data may obey the data distribution of the target project, so we give the data a higher weight. If the label of the data $\mathbf{D}_S$ is different from the cluster label, we believe that the data violate the clustering hypothesis. It means that the data may not obey the data distribution of the target project, so the data are given a lower weight. When all the data from $\mathbf{D}_S$ are given weights, we use the bagging method to train the final classifier from CCTrainData to predict $\mathbf{D}_{test}$. We will describe the algorithm in detail in the following section.

3.3. Clustering Stage. The clustering stage has two steps. In the first step, we use the NN method to select the candidate dataset (FCCData). In the second step, we use TSCluster proposed to clustering FCCData and assgin weights to the data in FCCData according to the clustering results. The flow of TSCluster is shown in algorithm 1.

TSCluster has two clustering processes. In the first clustering, we cluster to the defect data and nondefect data in $\mathbf{D}_{WP}$, respectively, and construct the initial cluster seed of the second clustering processes according to the clustering results. Specifically, we first select defect dataset and nondefect dataset from $\mathbf{D}_{WP}$, repectively. Next, we use k-means method to clustering these datasets to build a defect cluster set and nondefect cluster set (where defect cluster set has pk clusters and nondefect cluster set has nk clusters). Then, this two cluster sets are merged to built the initial cluster seed (namely, InitClusterSet) for semisupervised clustering. The specific process is shown in algorithm 2.

In the second clustering, based on the idea that is similar to constrained k-means [31], we use the InitClusterSet constructed by the first clustering as a constraint seed to clustering the FCCData dataset. Specifically, in each iteration of the clustering process, we use InitClusterSet to initialize the new cluster set and then calculate the distance between FCCData's data and each cluster center; then, we assign data to the nearest cluster. It is worth noting that in this process, we do not change the cluster membership of the data from $\mathbf{D}_{WP}$. When all FCCData's data are assigned, we update the centers of all clusters; if all cluster centers are not updated, the clustering is terminated and the final cluster set (finalClusterSet) is obtained.

After two clustering, we assign different weights to the data in each cluster. The specific steps are as follows: firstly,

Input: rep, pk,nk,FCCData $\mathbf{D}_{\mathrm{WP}}$, ϵ
Output: CCTrainData
//The first clustering to construct the initial cluster set
InitClusterSet = FristCluser(pk, nk, D_{wp});
//The second clustering
finalClusterSet = [];
for i in range(1,InitClusterSet.length) do
  finalClusterSet.add(InitClusterSet[i]);
for q in range(1,rep) do //rep is the number of cluster iterations
  nowClustersSet = [];
  for i in range(1,InitClusterSet.length) do //Initialize cluster seed according to InitClusterSet
    nowClustersSet.add(InitClusterSet[i]);
  for $(x_i, y_i) \in$ FCCData do
    Calculatethe distance between $(x_i y_i)$ and the mean u_j of finalClusterSet[j](1 $< j <$ InitClusterSet.length): $d_{ij} = \|x_i - u_j\|_2$;
    Find the cluster closest to $(x_i y_i)$according to d_{ij}:
$t_i =$ $\underset{t_i \in \{1,2,\ldots,\mathrm{InitclusterSet.length}\}}{\operatorname{argmin}} d_{ij}$;
    nowClustersSet $[t_i]$.add((x_i, y_i));
  if the mean value u_j of finalClusterSet[j] is equal to the mean value now u_j of nowClustersSet[j] then //Stopping condition
    break;
  finalClusterset = nowClustersSet;
//Set label
finalClusterSet[j].Clusterlabel = y_{init},where, (1 $< j <$ InitClusterSet.length), $(x_{\mathrm{init}} y_{\mathrm{init}}) \in$ InitClusterSet[j];
//Assign weight
CCTrainData = [];
for j in range(1,InitClusterSet.length) do
  for $(x_i y_i) \in$ finalClusterSet[j] do
    if $(x_i y_i) \in$ FCCData then
      $w_{ij} = \epsilon * \log((1/\mathrm{dis}(x_i, u_j))+e)$;
      if y_i == finalClusterSet[j].Clusterlabel then
        X_i.weight = w_{ij};
      else
        X_i.weight = 1/ w_{ij};
      CCTrainData.add($(x_i y_i)$);
return CCTrainData;

Algorithm 1: TSCluster.

for the the jth cluster in final cluster set (finalClusterSet[j]), we label a cluster label (namely, Clusterlabel), and set Clusterlabel is y_{init}, where $(x_{\mathrm{init}} y_{\mathrm{init}}) \in$ InitClusterSet[j]; then, according to the clustering assumption (i.e., data in the same cluster are more likely to have the same label), we assign higher weight to data which label is the same as Clusterlabel and assign lower weight to data which label is different from Clusterlabel. The specific calculation formula is as follows:

$$w_{ij} = \begin{cases} \epsilon * \log\left(\left(\frac{1}{\mathrm{dis}\left(x_i, u_j\right)}\right) + e\right) y_i = \text{clusterlabel}, \\ \frac{1}{\epsilon * \log\left(\left(1/\mathrm{dis}\left(x_i, u_j\right)\right) + e\right)} y_i \neq \text{clusterlabel}, \end{cases} \tag{1}$$

where $(\mathbf{x}_i, \mathbf{y}_i) \in$ finalClusterSet[j] $\cap$ FCCData, u_j is the cluster center of finalClusterSet[j], and dis(.) is the Euclidean distance between x_i and u_j, ϵ is an amplification factor, and in this paper, $\epsilon = 2$. In equation (1), the weight of data with the same label as Clusterlabel is higher than the weight of data with different label to Clusterlabel. For data $(\mathbf{x}_i \mathbf{y}_i)$, when $\mathbf{y}_i$ = Clusterlabel, it is closer to the cluster center, it is more likely obey the data distribution of target project; therefore, the weight of data is higher. On the other hand, when $y_i \neq$ Clusterlabel, it is closer to the cluster center, the data are less likely to obey the data distribution of target project; therefore, the weight of data is lower. After all data have been assigned weight, we add these data to a final training dataset (namely, CCTrainData), and then in the ensemble stage, we use CCTrainData to train the classifier.

3.4. Ensemble Stage. In the ensemble stage, we use the bagging method to train several classifiers (in this paper, we use NB as a classifier) and integrate these classifiers to predict the target data. Firstly, we sample data from CCTrainData to built several datasets according to the weight assigned in the clustering stage. It should be noted

```
Input: pk,nk, D_WP
Output: InitClusterSet
DefectDataset = SelectDefectData(D_WP)//Selectdefect data;
NonDefectDataset = SelectNonDefectData(D_WP)//Select non defect data;
if pk > DefectDataset.size then
   pk = DefectData.size;
if nk > NonDefectDataset.size then
   nk = NonDefectData.size;
//Use kmeans to clustering the DefectDataset and NonDefectDataset
DefectClusterSet = Kmeans(pk,DefectData);
NonDefectClusterSet = Kmeans(nk,NonDefectData);
InitClusterSet = DefectClusterSet + NonDefectClusterSet;
Remove the empty cluster in InitClusterSet;
Return InitClusterSet:
```

Algorithm 2: FristCluser.

```
Input: D_S, D_WP, D_Test, ϵ,rep,pk,nk,Classifiersize
Classifierpool = [];
//Using NN to select the candidate dataset
FCCData = NN (D_Test D_S);
//Clustering stage
CCTrainData = TSCluster(rep, pk,nk,FCCData, D_WP);
//Ensemble stage
for i in range(1,classifiersize) do
   TrainData_i = D_WP + resampleWithWeights (CCTrainData);
   Classifier_i = Train (TrainData_i)//train the classifier;
   Classifierpool.add (Classifier_i);
Use equation (2) to predict the defects of D_Test.
```

Algorithm 3: Tsbagging.

that a data with a higher weight mean a higher probability of it be sampled. Each dataset has CClen data. CClen is the amount of data in CCTrainData. Then, we combine these datasets with $\mathbf{D}_{\mathrm{WP}}$ to train the base classifiers, and add these base classifiers to a classifier set (Classifierpool). Finally, when all classifiers are trained, we use the plurality voting to ensemble all base classifiers in the Classifierpool to predict the data in the $\mathbf{D}_{\mathrm{Test}}$. The specific ensemble method is as follows:

$$H(x) = C_{\arg\max_j \sum_{i=1}^{\mathrm{Classifiersize}} I\left(h_i\left(x_t^{\mathrm{Test}}\right)=C_j\right)}, \tag{2}$$

where C_j is the predicted label, and its value is $C_0 = 1$ or $C_1 = 1$ Classifiersize is the amount of all base classifiers in the Classifierpool. When the classifier classifies x as c_j, $\mathrm{h}_i(x)$ is the prediction function of the ith classifier, $\mathrm{h}_i(x) \in \{-1, 1\}$, I(x)is an indicator when x is true, and $\mathrm{I}(x) = 1$, otherwise $I(x) = 0$. The overall process of Tsbagging is shown in algorithm 3.

4. Experiment

In this section, we describe three experiments we conducted to verify the performance of Tsbagging.

4.1. Datasets. In this experiment, we use seven projects in the public database promise [33] to build cross-project defect prediction datasets. The detailed description of these datasets is shown in Table 1. Each dataset in these datasets represents the test results of a program module, which contains 20 attributes and one label. The detailed description of these attributes is shown in Table 2 [34]. The label values are −1 and 1.1 represents defective module, and −1 represents nondefective module. Similar to reference [35–37], in each experiment, we select two different datasets from the seven datasets, and one of which is used as the target project and the other as the source project to build the CPDP experimental dataset (for example, we use poi-2.0 as the target project and synapse-1.2 as the source project); therefore, we have created 42 experimental datasets.

4.2. Performance Measures. In this paper, we use $F1$-measure, MCC, g-measure, and balance [36, 38–40] to evaluate the performance of defect prediction. These performance measures are calculated from the confusion matrix shown in Table 3, where TP means true defect, FP means false defect, FN means false nondefect, and TN means true nondefect.

Table 1: Details of the experiment dataset.

Project	Module	Defect-prone module	Defect-prone (%)
Poi-2.0	314	37	12
Synapse-1.2	256	86	34
Ant-1.6	351	92	26
Camel-1.2	608	216	36
log4j-1.1	109	37	34
Jedit-4.0	306	75	25
Xerces-1.2	440	71	16

Table 2: Attribute description.

Attribute	
wmc	Weighted methods per class
dit	Depth of inheritance tree
noc	Number of children
rfc	Response for a class
lcom	Lack of cohesion in methods
lcom3	Normalized version of LCOM
npm	Number of public methods
Loc	Lines of code
dam	Data access metric
moa	Measure of aggregation
mfa	Measure of function abstraction
cam	Cohesion among methods of class
ic	Inheritance coupling
amc	Average method complexity
Ca	Afferent couplings
ce	Efferent couplings
max_cc	Maximum values of methods in the same class
avg_cc	Mean values of methods in the same class of methods in a given class
cbm	Coupling between methods
cbo	Coupling between object classes

Table 3: Obfuscation matrix.

		Actual	
		Defect	Nondefect
Predicted	Defect	TP	FP
	Nondefect	FN	TN

Recall is a measure of completeness, describing probabilities of true defective modules in comparison with the total number of defective modules:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3}$$

Precision is a measure of exactness, which defines the probabilities of the presence of modules that are truly defective from the number of modules predicted to be defective:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4}$$

PF shows the proportion of all the modules without defects predicted to be defective:

$$\mathrm{PF} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \quad (5)$$

$F1$-measure is the harmonic average of recall and precision. An algorithm with a higher $F1$-measure value (namely, $F1$ value) implies a better performance as

$$F1 = \frac{2 * \mathrm{recall} * \mathrm{Precision}}{\mathrm{recall} + \mathrm{Precision}}. \quad (6)$$

Matthews correlation coefficient (MCC) can take into account the measurement of TP, FP, TN, and TN in a more comprehensive way. Its range of values is between [−1, 1], where 1 denotes a perfect prediction, and −1 indicates complete disagreement between actual and predicted values. The more the values are means the better the performance of the classifier will be:

$$\mathrm{MCC} = \frac{\mathrm{TP} * \mathrm{TN} - \mathrm{FN} * \mathrm{FP}}{\sqrt{(\mathrm{TP} + \mathrm{FN})(\mathrm{TP} + \mathrm{FP})(\mathrm{FN} + \mathrm{TN})(\mathrm{FP} + \mathrm{TN})}}. \quad (7)$$

G-measure [14] is the harmonic average of recall and (1–PF). An algorithm with a higher g-measure value implies a better performance as

$$\mathrm{g-measure} = \frac{2 * \mathrm{Recall} * (1 - \mathrm{PF})}{\mathrm{Recall} + (1 - \mathrm{PF})}. \quad (8)$$

The measure Balance is introduced by calculating the Euclidean distance from the real (recall, PF) point to (1, 0), since the point (recall = 1, PF = 0) is the optimal point where all defects are detected without missing:

$$\mathrm{Balance} = 1 - \frac{\sqrt{(0 - \mathrm{PF})^2 + (1 - \mathrm{Recall})^2}}{\sqrt{2}}. \quad (9)$$

4.3. Experimental Design. In this section, we design experiments to verify the performance of Tsbaging in the following four questions:

RQ1. Can Tsbagging effectively overcome the impact of different data distributions between different projects, so as to obtain better prediction performance compared with traditional defect prediction methods?

To research this question, we compare Tsbagging with other traditional SDP methods and calculate their $F1$-measure, MCC, g-measure, and balance values. It is worth noting that the traditional SDP methods do not take into account the impact of data distribution differences between source project and target project when training prediction models. If the performance of Tsbagging is better than other traditional SDP methods, Tsbagging can effectively overcome the impact of different data distributions between different projects. Specifically, we compare Tsbagging with Naive Bayesian (NB) [18], NB + WP, and adaboost [41], where adaboost and NB only use $\mathbf{D}_S$ to train prediction models, and Tsbagging, NB + WP use $\mathbf{D}_S$ and $\mathbf{D}_{\mathrm{WP}}$ data to train prediction models, where the amount of $\mathbf{D}_{\mathrm{WP}}$ is 10%, 15%, and 25% of the target project data, respectively.

RQ2. Can Tsbagging achieve better prediction performance than other CPDP methods?

In order to research this question, we compare Tsbagging with other CPDP methods and calculate their $F1$-measure, MCC, g-measure, and balance values. Specifically, we compare Tsbagging with NN, DBSCAN filter [19], VCB + SVM [21], NN + WP, and DTB, in which NN, DBSCAN filter, and VCB + SVM only use $\mathbf{D}_S$ data to train prediction models , Tsbagging, DTB [26], and NN + WP [25] use $\mathbf{D}_S$ and $\mathbf{D}_{\mathrm{WP}}$ data to train prediction models, where the amount of $\mathbf{D}_{\mathrm{WP}}$ is also 10%, 15%, and 25% of the target project data, respectively. It is worth noting that NN, DBSCAN filter, VCB + SVM, and NN + WP do not take into account the impact of concept shift, and DTB also do not take into account the impact of concept shift when it assigns weights. If the performance of Tsbagging is better than other CPDP methods, Tsbagging can effectively overcome the impact of concept shift.

RQ3. Are the experimental results of **RQ1** and **RQ2** statistically significant?

Similar to the literature [39, 40], this paper uses the hypothesis test to statistically analyze the experimental results of **RQ1** and **RQ2**, so as to verify whether the conclusions are statistically significant. Specifically, we use the Wilcoxon rank sum test to judge whether the experimental results of Tsbagging have significant differences compad with other methods. In this section, experiments are carried out on 42 datasets and 10 times are repeated on each dataset. Therefore, this paper will make statistical analysis on these 420 data points. The confidence level used by the Wilcoxon rank sum test is 0.05. Moreover, we use the box plot to display the distribution of those 420 points in more detail. The box plot has five numerical aspects: minimum, lower quartile, median, upper quartile, and maximum, and it is often used in SDP experiments [42–44].

RQ4. How do we determine the number of clusters (i.e., nk and pk) for Tsbagging?

In order to study this problem, we will measure the prediction performance of Tsbagging under different nk and pk , then we find out the reasonable value range of nk and pk. Specifically, we calculate $F1$-measure, MCC, g-measure, and balance of Tsbagging under nk = pk = 3, 5, 7, and 10, respectively. Tsbgging also uses 10%, 15%, and 25% of the target project data, respectively.

It should be noted that in each experiment, we randomly take 10%, 15%, and 25% of the data from target data as $\mathbf{D}_{\mathrm{WP}}$ and the remaining data as $\mathbf{D}_{\mathrm{Test}}$. We repeat 10 times experiments on each dataset and take the average value of these 10 times experiments as the final result. In addition, as suggested by Turhan et al. [18], we replace all numeric values with a "log-filter", i.e., N with $\ln(N)$. This spreads out skewed curves more evenly across the space from the minimum to maximum values. This "spreading" can significantly improve the effectiveness of data mining, as the distribution of log-

filtered feature values fits better to the normal distribution assumption [47]. In this experiment, all methods are implemented by the famous machine learning framework Weka3.9 [45] and the running environment of experiment is Java 11 and Windows 10. The parameter settings of each method are shown in Table 4.

5. Experimental Results

In this section, we answer the four research questions proposed.

*5.1. Results for **RQ1**.* In order to answer **RQ1**, we calculated the prediction performance of Tsbagging and other SDP methods on different $\mathbf{D}_{\mathrm{WP}}$. The experimental results are shown in Tables 5–7. From the experimental results, it can be seen that NB and Adaboost only use the source project data to train the prediction model, so they are difficult to achieve good prediction performance. NB + WP uses a limit of target project data during training. Therefore, the performance of NB + WP is better than NB and Adaboost. However, NB + WP still does not take into account the impact of distribution differences between different projects, so the performance of NB + WP is still lower than that of Tsbagging. Moreover, on the 10% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 18.5%, 52.7%, 24.5%, and 13.5%, respectively. On the 15% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 21.6%, 59.6%, 25.6%, and 14.1%, respectively. On the 25% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 24%, 71.4%, 26.7%, and 15.2%, respectively. In summary, the Tsbagging can use the TScluster method to find out the data which obey the distribution of the target project, so as to effectively overcome the impact caused by the difference of data distribution between projects and achieve better prediction performance.

*5.2. Results for **RQ2**.* In order to answer **RQ2**, we compare Tsbagging with other CPDP methods and calculate their $F1$, MCC, g-measure, and balance on different datasets. The experimental results are shown in Tables 8–10. The experimental results show that due to NN, DBSCAN filter, and VCB + SVM only use $\mathbf{D}_S$ to train the prediction model, when the distribution of source project and target project is quite different, especially it occurs concept shift, NN, DBSCAN filter, and VCB + SVM are difficult to achieve good prediction performance, even DBSCAN filter and VCB + SVM have negative transfer [46](that is, the CPDP method's prediction performance is lower than the traditional SDP methods). DTB and NN + WP use a limit of target project data to training the prediction model, so their performance are better than NN, DBSCAN filter, and VCB + SVM. However, NN + WP and DTB also do not consider the impact of concept shift when selecting $\mathbf{D}_S$ data and assigning weights, so their $F1$, MCC, g-measure, and balance are lower than Tsbagging. Moreover, on the 10% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 23.2%, 60.6%, 26.4%, and 18.6%, respectively. On the 15% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 26.6%, 74.1%, 36.1%, and 19%, respectively. On the 25% target project dataset, the largest increases of $F1$, MCC, g-measure, and balance are 28.6%, 84.4%, 38%, and 20.3%, respectively. In summary, Tsbagging can overcome the impact of concept shift using semi-supervised clustering to achieve better performance than other CPDP methods.

*5.3. Results for **RQ3**.* In order to verify whether the experimental results of **RQ1** and **RQ2** are statistically significant, this paper uses the Wilcoxon rank sum test and box plot to statistically analyze all experimental results. Tables 11–13 show the p values on different datasets. The experimental results show that the p values of the four performance measures on all datasets are lower than 0.05 compared with other defect prediction methods. Therefore, it shows that the prediction results of Tsbagging are statistically significantly different from those of other methods. On the other hand, the box plot of performance measures is shown on each dataset. From Figures 3–14, we can see that the median and lower quartile of MCC, g-measure, and balance are higher than all of other methods, and the median and lower quartile of $F1$ are higher than most of the other methods (the median values of DTB and Tsbagging are roughly the same level, but the lower quartile of Tsbagging is significantly higher than that of DTB). Therefore, it can be seen that the prediction performance of Tsbagging is better than other defect prediction methods. In conclusion, the experimental results of **RQ1** and **RQ2** are statistically significant.

*5.4. Results for **RQ4**.* In order to answer **RQ4**, we calculate $F1$, MCC, g-measure, and balance values of Tsbagging under different nk and pk, where the value range of nk and pk is 3, 5, 7, and 10. The experimental results are shown in Table 14 and Figure 15–17. The Y-axis and X-axis of Figures 15–17, represent a performance index and its corresponding value, respectively. From the experimental results, it can be seen that the prediction performance of Tsbagging does not change significantly in different nk and pk, and they are better than other methods. Therefore, it is reasonable to take 3, 5, 7, and 10 for nk and pk at the same time for Tsbagging.

6. Validity Threats

6.1. Internal Validity. We compared our method with other SDP methods.To avoid the potential faults as much as possible during the implementation process of the

Table 4: Parameter setting of each method.

Algorithm	Parameter
Adaboost	The number of iterations is 20, and the basic classifier is NB.
NN	The number of K-nearest neighbors is 10, and NB is used as the basic classifier.
DBSCAN filter	The number of minimum records was set to 10, and the distance is set to 10
VCB + SVM	The number of iterations is 5
Tsbagging	ϵ = 2. The number of clustering iterations is 100, pk = 5, nk = 5, classifiersize = 20, and the base classifier is NB
DTB	The number of iterations is 20, and the base classifier is NB

Table 5: Tsbagging compared with other SDP methods on 10% target data.

Performance	NB	Adaboost	NB + WP	Tsbagging
*F*1	0.37331	0.37367	0.38891	0.4427
MCC	0.13551	0.14112	0.14956	0.20694
g-measure	0.43927	0.44549	0.45031	0.54691
Balance	0.50141	0.50704	0.50897	0.56982

Table 6: Tsbagging compare with other SDP methods on 15% target data.

Performance	NB	Adaboost	NB + WP	Tsbagging
*F*1	0.37437	0.37436	0.39413	0.45525
MCC	0.1379	0.14313	0.15446	0.22013
g-measure	0.44025	0.44611	0.45297	0.55298
Balance	0.50216	0.50761	0.51133	0.57317

Table 7: Tsbagging compare with other SDP methods on 25% target data.

Performance	NB	Adaboost	NB + WP	Tsbagging
*F*1	0.37225	0.37249	0.40313	0.46171
MCC	0.13295	0.13845	0.16239	0.22791
g-measure	0.4378	0.44367	0.46055	0.55494
Balance	0.50056	0.50585	0.51659	0.57654

Table 8: Tsbagging compared with other CPDP methods on 10% target data.

Performance	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
*F*1	0.3871	0.36847	0.35915	0.40169	0.40849	0.4427
MCC	0.15157	0.13112	0.12887	0.16283	0.16352	0.20694
g-measure	0.468	0.43237	0.40261	0.47439	0.50011	0.54691
Balance	0.52107	0.49851	0.48052	0.52564	0.5351	0.56982

Table 9: Tsbagging compared with other CPDP methods on 15% target data.

Performance	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
*F*1	0.39069	0.36885	0.35966	0.41024	0.41026	0.45525
MCC	0.15524	0.13274	0.1264	0.17305	0.16393	0.22013
g-measure	0.47141	0.4326	0.40631	0.48199	0.49243	0.55298
Balance	0.52293	0.49886	0.48159	0.52979	0.52915	0.57317

experiment, we implement these models based on the Java machine learning library Weka.

6.2. External Validity. External validity refers to the degree to generalize the research results to other situations. The most commonly used promise dataset in the cross-project defect prediction research is selected as the experimental data to ensure that the experimental results have certain representative significance. At the same time, the prediction performance is evaluated in terms of four well-known performance measures (i.e., *F*1-measure, g-measure, balance, and MCC) to ensure the generality of the experiment.

6.3. Statistical Validity. In this paper, we use the Wilcoxon rank sum test to statistically analyze the experimental results. The Wilcoxon rank sum test is a nonparametric test. It has no

Table 10: Tsbagging compared with other CPDP methods on 25% target data.

Performance	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
*F*1	0.38972	0.36728	0.35878	0.4201	0.42511	0.46171
MCC	0.15334	0.12828	0.12355	0.18224	0.17822	0.22791
g-measure	0.47416	0.43104	0.40177	0.49139	0.50464	0.55494
Balance	0.52438	0.49775	0.47932	0.53669	0.53751	0.57654

Table 11: Wilcoxon rank sum test p value on 10% target project data.

p value	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB
*F*1-measure	2.74*e*−09	2.35*e*−09	1.19*e*−05	6.93*e*−06	5.50*e*−09	3.74*e*−17	0.0015	0.0078
MCC	3.30*e*−13	4.18*e*−12	1.18*e*−08	1.16*e*−07	1.91*e*−13	7.76*e*−15	1.51*e*−05	8.36*e*−05
G-measure	5.04*e*−17	7.48*e*−14	1.91*e*-14	8.65*e*−09	1.33*e*−17	2.24*e*−30	2.01*e*−08	3.96*e*−07
Balance	7.67*e*−18	2.69*e*−14	1.73*e*-14	2.69*e*−09	2.15*e*−18	3.15*e*−31	2.10*e*−08	1.72*e*−07

Table 12: Wilcoxon rank sum test p value on 15% target project data.

p value	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB
*F*1-measure	1.23*e*−11	1.06*e*−11	8.60*e*−07	1.21*e*−07	2.07*e*−11	9.09*e*−22	5.97*e*−04	8.08*e*−04
MCC	1.84*e*−16	3.70*e*−15	1.69*e*−10	9.13*e*−10	1.29*e*−16	4.35*e*−20	1.09*e* − 4	2.48*e*−06
G-measure	9.07*e*−18	1.60*e*−14	3.55*e*−15	3.78*e*−09	2.86*e*−18	3.01*e*−31	3.99*e*−08	6.56*e*−10
Balance	6.41*e*−19	2.46*e*−15	2.01*e*−15	6.11*e*−10	1.76*e*−19	1.86*e*−32	2.95*e*−08	1.30*e*−10

Table 13: Wilcoxon rank sum test p value on 25% target project data.

p value	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB
*F*1-measure	3.18*e*−15	2.30*e*−15	1.16*e*−06	1.83*e*−10	7.21*e*−15	1.25*e*−24	6.58*e*−04	0.0054
MCC	1.01*e*−21	1.94*e*−20	3.69*e*−11	5.8*e*−13	1.48*e*-21	2.02*e*−26	1.6*e*−05	3.46*e*−05
G-measure	1.98*e*−19	2.68*e*−16	2.62*e*−14	2.36*e*−09	8.94*e*−20	3.52*e*−33	8.41*e*−07	7.94*e*−09
Balance	1.60*e*−21	8.85*e*−18	1.55*e*−14	1.01*e*−10	7.53*e*−22	6.28*e*−36	6.31*e*−07	7.60*e*−10

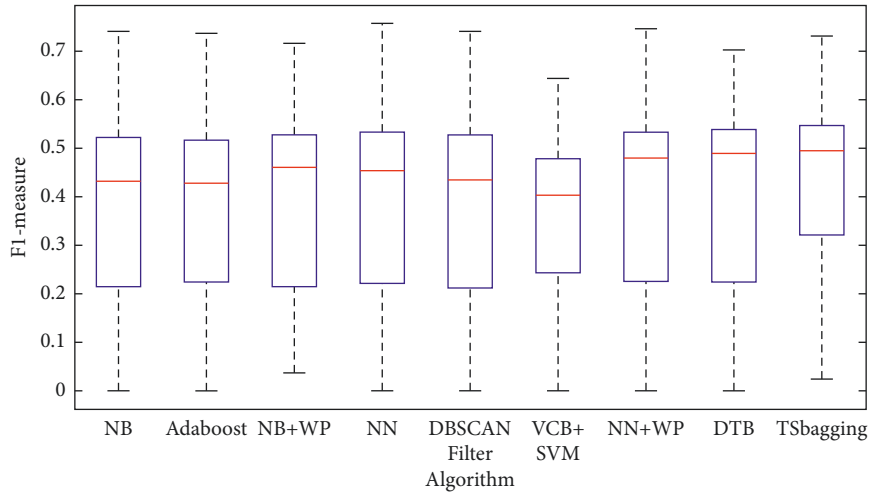

Figure 3: Box plot of *F*1-measure on 10% target project data.

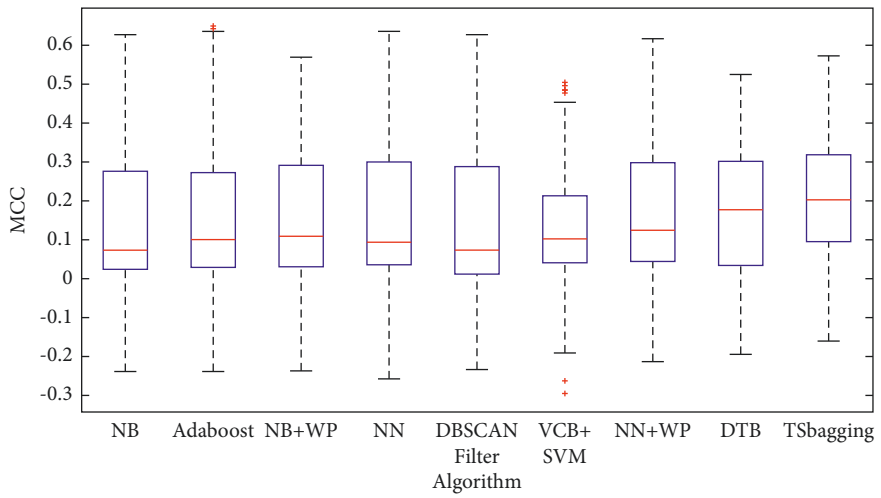

Figure 4: Box plot of MCC on 10% target project data.

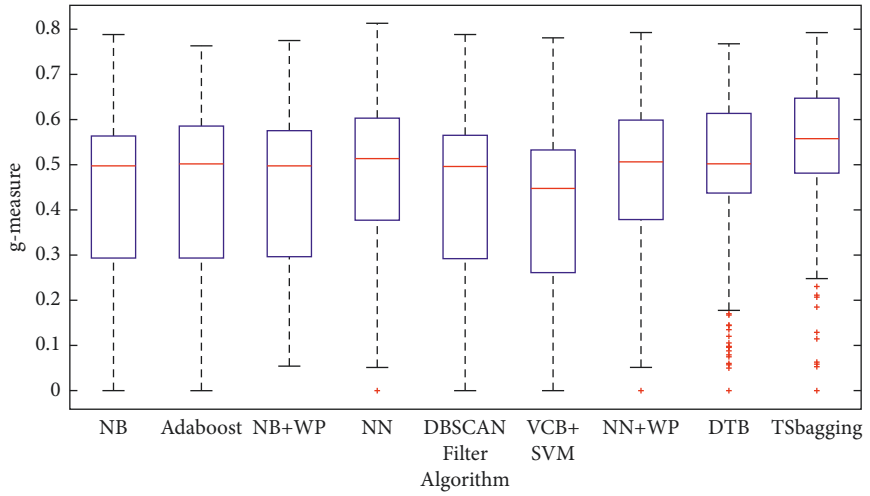

Figure 5: Box plot of g-measure on 10% target project data.

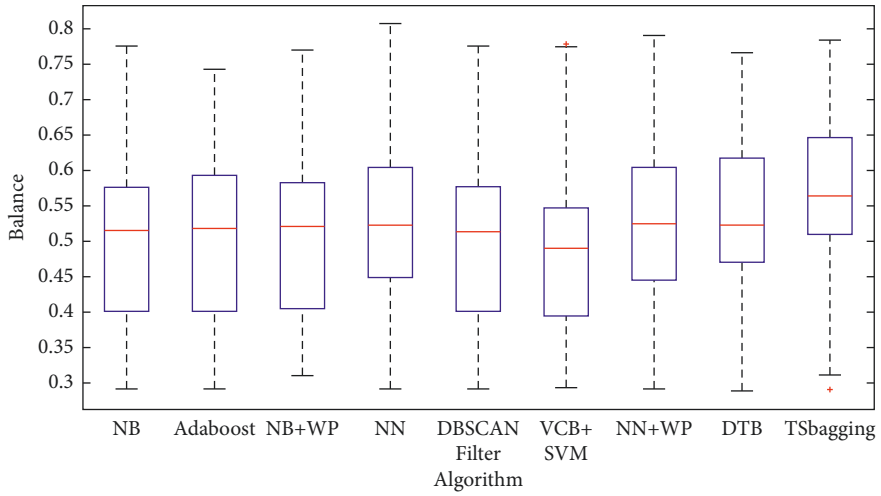

Figure 6: Box plot of balance on 10% target project data.

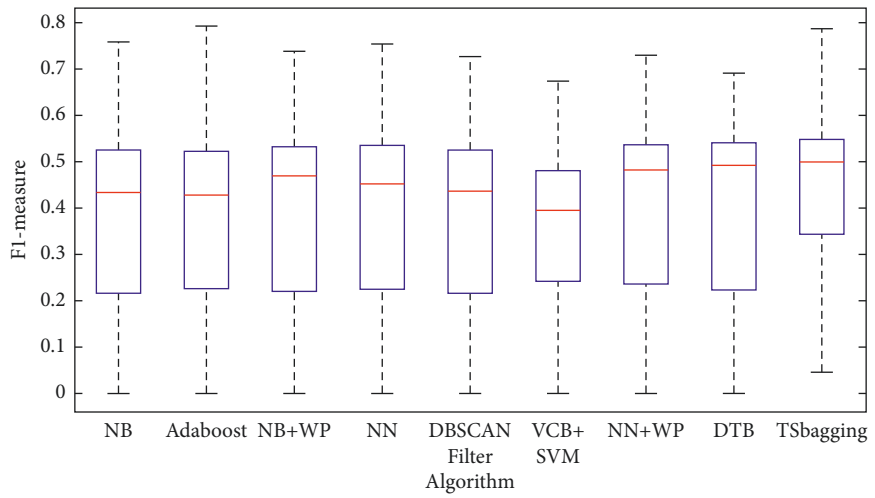

Figure 7: Box plot of $F1$-measure on 15% target project data.

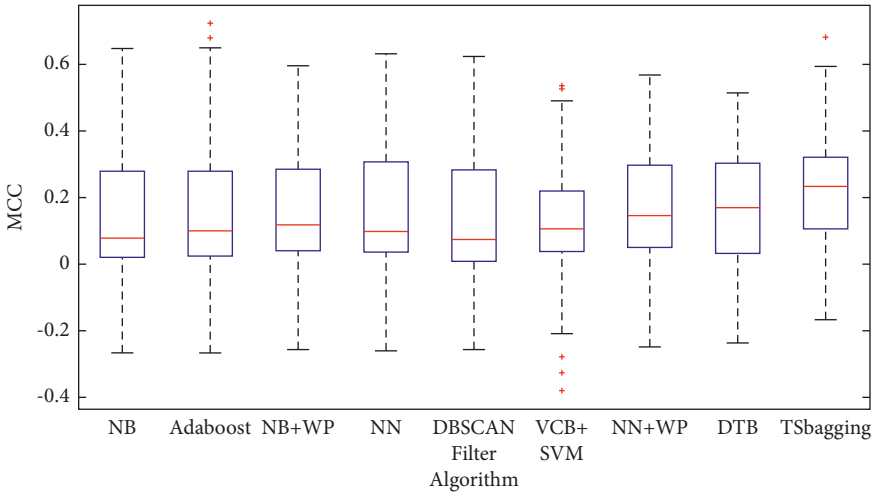

Figure 8: Box plot of MCC on 15% target project data.

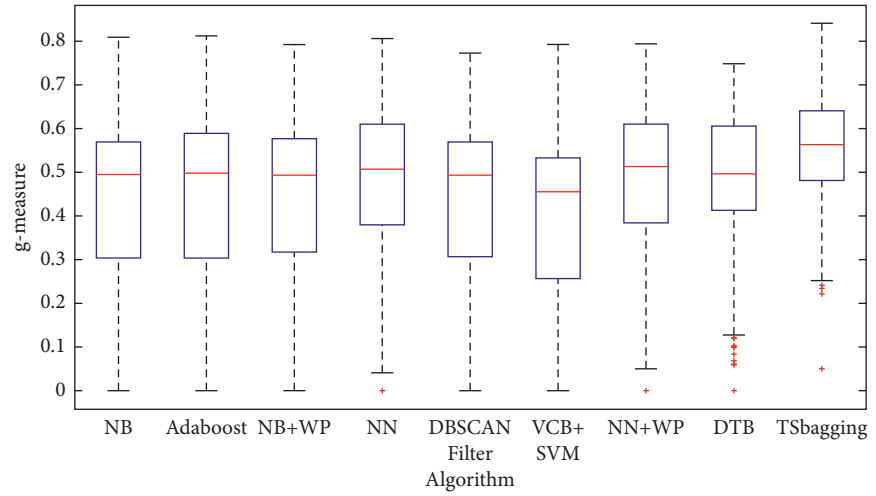

Figure 9: Box plot of g-measure on 15% target project data.

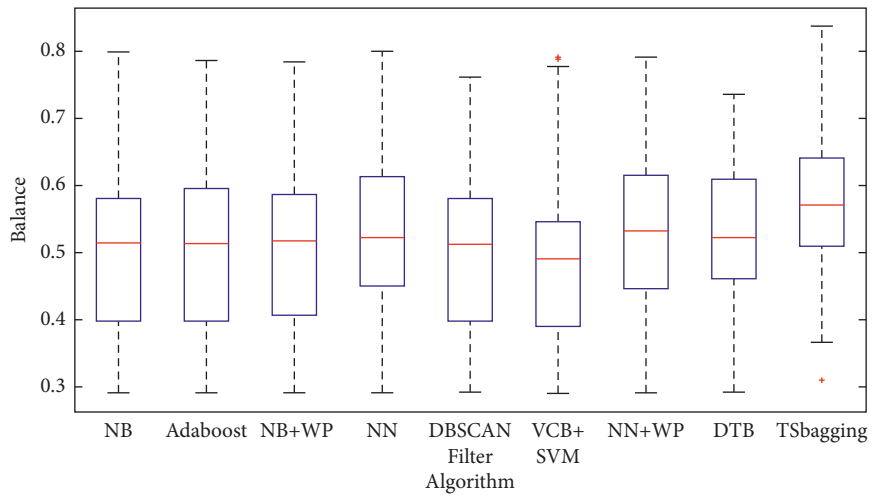

Figure 10: Box plot of balance on 15% target project data.

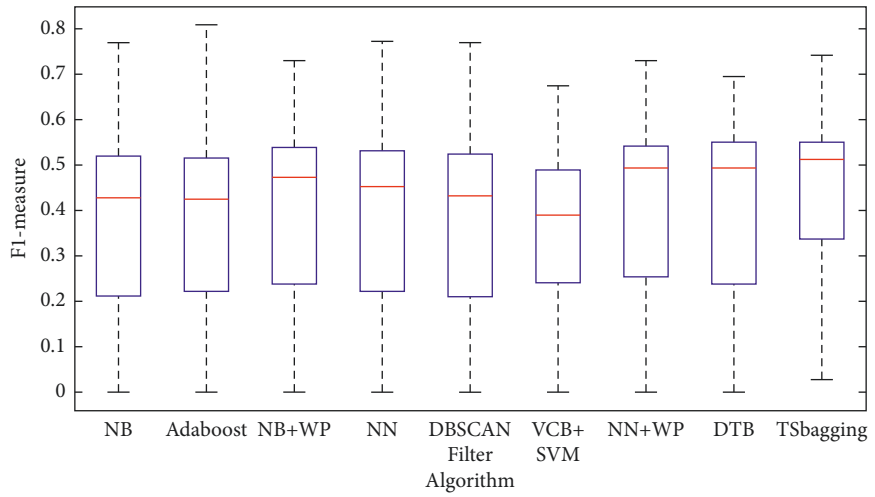

Figure 11: Box plot of $F1$-measure on 25% target project data.

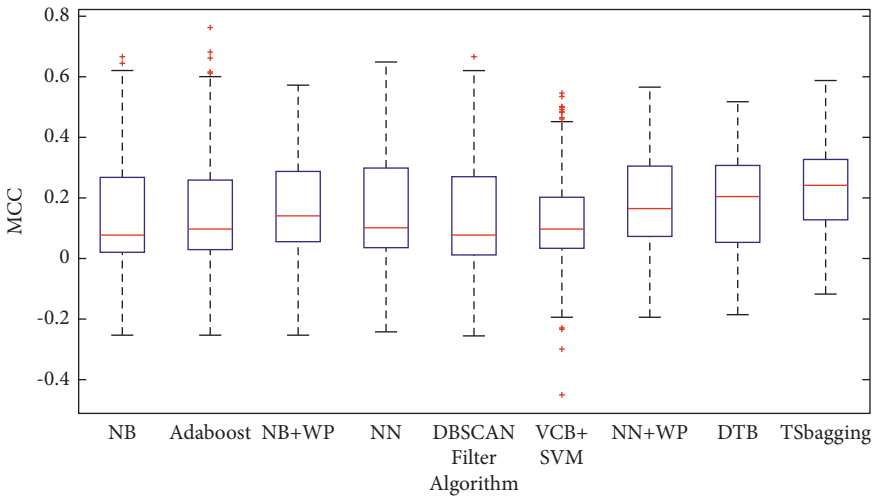

Figure 12: Box plot of MCC on 25% target project data.

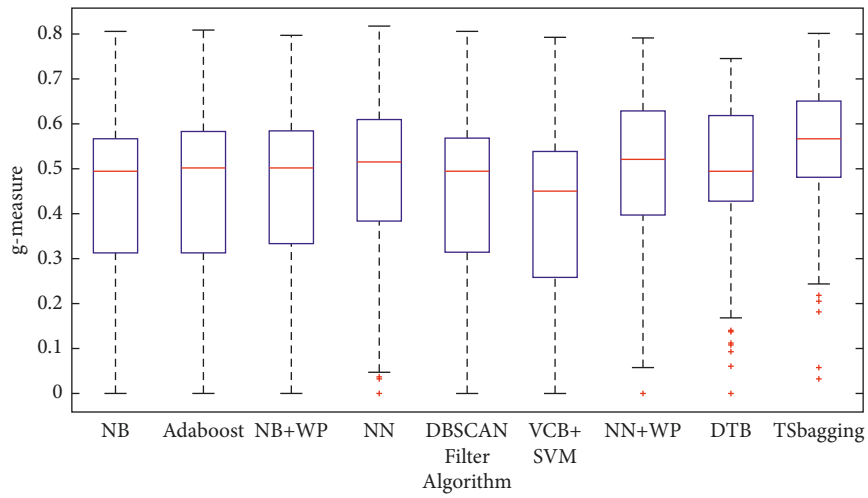

Figure 13: Box plot of g-measure on 25% target project data.

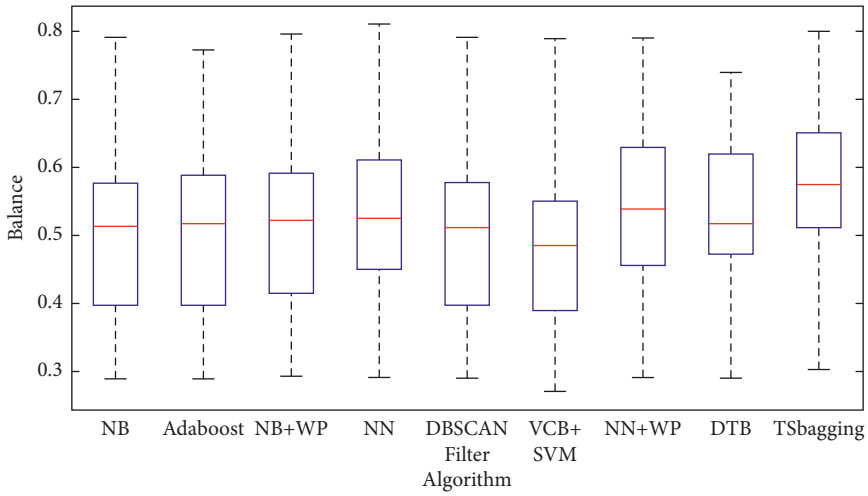

Figure 14: Box plot of balance on 25% target project data.

Table 14: The performance of Tsbagging on different nk and pk.

Performance		$nk = pk = 3$	$nk = pk = 5$	$nk = pk = 7$	$nk = pk = 10$
10% target data	$F1$	0.44065	0.44270	0.44066	0.43836
	MCC	0.20188	0.20694	0.20284	0.20313
	g-measure	0.54467	0.54691	0.54250	0.54014
	Balance	0.56711	0.56982	0.56604	0.56507
15% target data	$F1$	0.45418	0.45525	0.45454	0.45598
	MCC	0.21972	0.22013	0.22131	0.22120
	g-measure	0.55401	0.55298	0.55335	0.55630
	Balance	0.57371	0.57317	0.57455	0.57661
25% target data	$F1$	0.46521	0.46171	0.46441	0.46497
	MCC	0.23202	0.22791	0.23202	0.23190
	g-measure	0.55863	0.55494	0.55795	0.55874
	Balance	0.57858	0.57654	0.57842	0.57867

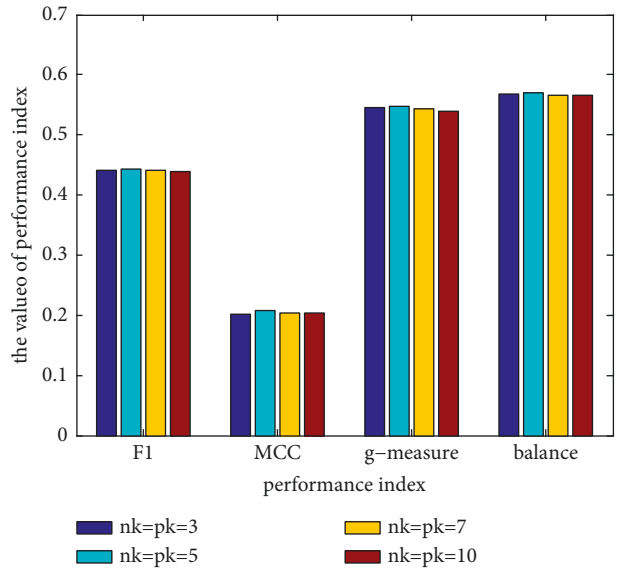

Figure 15: Tsbagging's performance under different nk and pk target project data on 10% target project data.

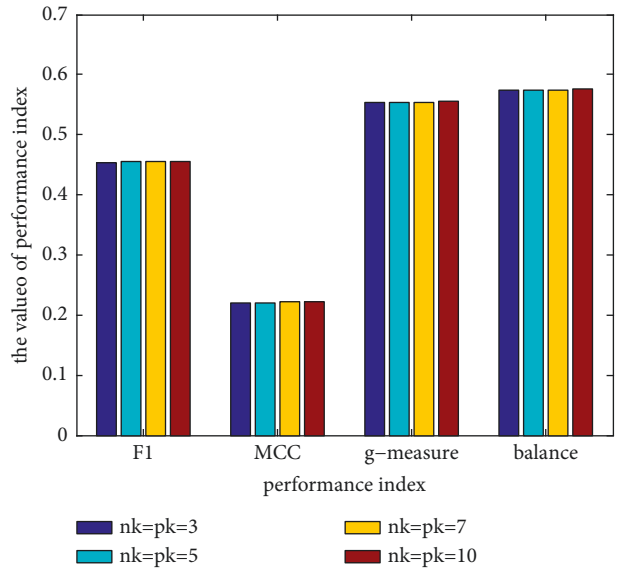

Figure 16: Tsbagging's performance under different nk and pk target project data on 15% target project data.

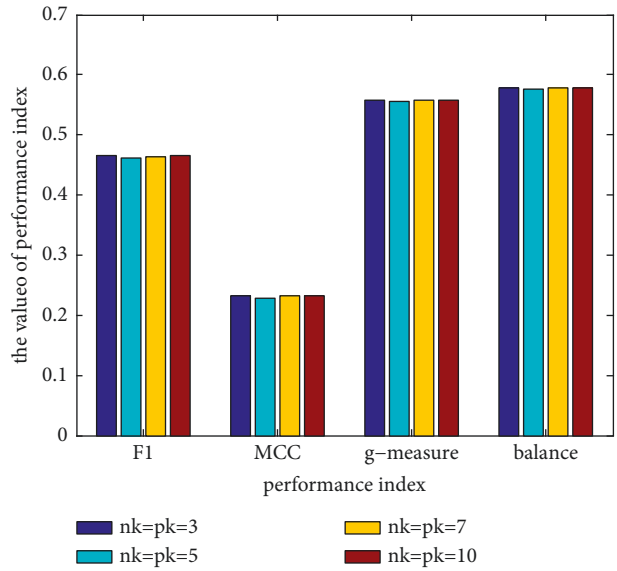

Figure 17: Tsbagging's performance under different nk and pk target project data on 25% target project data.

Table 15: The *F*1 in 10% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.21735	0.21904	0.21404	0.2155	0.21639	0.23235	0.21333	0.21665	0.22607
ant-1.6 ⇒ poi-2.0	0.22358	0.22889	0.22455	0.21878	0.22161	0.21416	0.21662	0.1993	0.21584
camel-1.2- ⇒ poi-2.0	0.20549	0.20549	0.20572	0.21303	0.20485	0.18016	0.20608	0.21133	0.22765
log4j-1.1 ⇒ poi-2.0	0.21467	0.21467	0.22093	0.21978	0.21521	0.22636	0.22176	0.20868	0.21572
jedit-4.0 ⇒ poi-2.0	0.2162	0.26957	0.2074	0.20484	0.2162	0.2147	0.20257	0.19213	0.21595
xerces-1.2 ⇒ poi-2.0	0.03518	0.03518	0.07843	0.00408	0	0.2608	0.01185	0.09972	0.24428
poi-2.0 ⇒ synapse-1.2	0.54364	0.54364	0.5553	0.58095	0.54364	0.50871	0.56321	0.59415	0.55864
ant-1.6 ⇒ synapse-1.2	0.5953	0.55626	0.58767	0.58044	0.58532	0.40549	0.58804	0.58745	0.59561
camel-1.2- ⇒ synapse-1.2	0.50718	0.50718	0.50702	0.51584	0.50195	0.34559	0.51367	0.54653	0.52893
log4j-1.1 ⇒ synapse-1.2	0.53011	0.53011	0.55312	0.54023	0.53737	0.49163	0.54942	0.54602	0.56122
jedit-4.0 ⇒ synapse-1.2	0.58037	0.5972	0.5785	0.56968	0.58037	0.2854	0.5738	0.57984	0.57098
xerces-1.2 ⇒ synapse-1.2	0.10306	0.10306	0.102	0.08632	0.06006	0.48965	0.10213	0.09281	0.44261
poi-2.0 ⇒ ant-1.6	0.52069	0.52069	0.51521	0.52199	0.52069	0.38772	0.51697	0.50217	0.52821
synapse-1.2 ⇒ ant-1.6	0.49947	0.50619	0.50766	0.49896	0.49899	0.49348	0.50633	0.51329	0.51819
camel-1.2- ⇒ ant-1.6	0.41971	0.41971	0.42608	0.42529	0.42324	0.31031	0.42949	0.46786	0.45333
log4j-1.1 ⇒ ant-1.6	0.50175	0.50175	0.5088	0.50175	0.51521	0.57478	0.5088	0.53242	0.52713
jedit-4.0 ⇒ ant-1.6	0.55276	0.59208	0.5604	0.55533	0.55276	0.40075	0.56422	0.5348	0.57201
xerces-1.2 ⇒ ant-1.6	0.08695	0.08695	0.13622	0.18041	0.06484	0.45114	0.21772	0.31335	0.44708
poi-2.0 ⇒ camel-1.2	0.46086	0.46086	0.51667	0.4695	0.46086	0.50283	0.51742	0.53499	0.51786
synapse-1.2 ⇒ camel-1.2	0.47467	0.45434	0.50601	0.47351	0.47618	0.40148	0.50761	0.50633	0.49581
ant-1.6 ⇒ camel-1.2	0.42734	0.36304	0.48021	0.42773	0.4276	0.30902	0.48243	0.49156	0.48163
log4j-1.1 ⇒ camel-1.2	0.47334	0.47334	0.51292	0.47772	0.48463	0.38806	0.51226	0.51046	0.51613
jedit-4.0 ⇒ camel-1.2	0.40347	0.40569	0.45689	0.40315	0.40347	0.21309	0.46236	0.49466	0.45534
xerces-1.2 ⇒ camel-1.2	0.12117	0.12117	0.18614	0.05679	0.12117	0.47673	0.17253	0.14243	0.47279
poi-2.0 ⇒ log4j-1.1	0.60153	0.60153	0.60798	0.59802	0.60214	0.49732	0.59661	0.53887	0.5446
synapse-1.2 ⇒ log4j-1.1	0.61422	0.59282	0.62118	0.62964	0.61664	0.40917	0.62354	0.59303	0.6248
ant-1.6 ⇒ log4j-1.1	0.70303	0.70573	0.66764	0.70235	0.68642	0.44252	0.68613	0.6387	0.6329
camel-1.2- ⇒ log4j-1.1	0.54205	0.54205	0.54205	0.56575	0.54205	0.30019	0.54003	0.56057	0.55413
jedit-4.0 ⇒ log4j-1.1	0.5081	0.50894	0.57968	0.56308	0.50887	0.2941	0.65088	0.61637	0.57729
xerces-1.2 ⇒ log4j-1.1	0.09821	0.09821	0.09787	0.025	0	0.42067	0.01818	0.22218	0.39049
poi-2.0 ⇒ jedit-4.0	0.47839	0.47839	0.46862	0.46148	0.47839	0.42892	0.47671	0.49122	0.47553
synapse-1.2 ⇒ jedit-4.0	0.44287	0.44009	0.45569	0.47216	0.44536	0.47536	0.48161	0.49349	0.493
ant-1.6 ⇒ jedit-4.0	0.53598	0.48923	0.52952	0.53656	0.54119	0.55492	0.53015	0.53153	0.53275
camel-1.2- ⇒ jedit-4.0	0.38182	0.38182	0.39133	0.43417	0.37797	0.24804	0.44492	0.45252	0.45857
log4j-1.1 ⇒ jedit-4.0	0.44633	0.44633	0.46468	0.45268	0.45209	0.54864	0.46678	0.47041	0.4871
xerces-1.2 ⇒ jedit-4.0	0.06342	0.06342	0.0726	0.5108	0.04334	0.33488	0.51579	0.51123	0.51133

Table 15: Continued.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
poi-2.0 ⇒ xerces-1.2	0.27102	0.27102	0.30633	0.26847	0.27102	0.25367	0.28799	0.24271	0.28724
synapse-1.2 ⇒ xerces-1.2	0.20992	0.238	0.22005	0.21827	0.20915	0.24258	0.22003	0.209	0.26393
ant-1.6 ⇒ xerces-1.2	0.25193	0.26689	0.27925	0.25161	0.2513	0.19761	0.27626	0.26298	0.30391
camel-1.2- ⇒ xerces-1.2	0.15626	0.15626	0.16707	0.15995	0.16056	0.18903	0.17926	0.19847	0.23449
log4j-1.1 ⇒ xerces-1.2	0.2204	0.2204	0.23348	0.2204	0.21719	0.27222	0.23372	0.24739	0.31214
jedit-4.0 ⇒ xerces-1.2	0.23927	0.27693	0.28141	0.24605	0.23927	0.20993	0.28179	0.25688	0.32005
Average	0.37331	0.37367	0.38891	0.3871	0.36847	0.35915	0.40169	0.40849	0.4427

Table 16: The MCC in 10% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.04146	0.0468	0.03122	0.03552	0.03836	0.09204	0.02901	0.04179	0.06015
ant-1.6 ⇒ poi-2.0	0.07266	0.08273	0.07609	0.06087	0.0679	0.05311	0.05565	0.01819	0.05374
camel-1.2- ⇒ poi-2.0	0.00197	0.00197	0.00316	0.03918	−0.00171	0.0598	0.00712	0.03155	0.06879
log4j-1.1 ⇒ poi-2.0	0.02798	0.02798	0.04775	0.04388	0.02959	0.05547	0.05055	0.01709	0.03109
jedit-4.0 ⇒ poi-2.0	0.04572	0.14578	0.02578	0.02065	0.04572	0.21011	0.01465	−0.00551	0.04474
xerces-1.2 ⇒ poi-2.0	−0.05519	−0.05519	−0.01737	−0.08453	−0.08197	0.14514	−0.07864	−0.0121	0.13389
poi-2.0 ⇒ synapse-1.2	0.2281	0.2281	0.24838	0.3167	0.2281	0.08048	0.27762	0.33512	0.27296
ant-1.6 ⇒ synapse-1.2	0.34214	0.29559	0.32688	0.31677	0.32458	0.2309	0.32858	0.32876	0.35227
camel-1.2- ⇒ synapse-1.2	0.07812	0.07812	0.07821	0.10231	0.05546	0.08566	0.10008	0.22315	0.16238
log4j-1.1 ⇒ synapse-1.2	0.18415	0.18415	0.24222	0.20913	0.20152	0.17312	0.23284	0.23679	0.2635
jedit-4.0 ⇒ synapse-1.2	0.33082	0.35833	0.32527	0.3162	0.33082	0.14731	0.31654	0.31941	0.31014
xerces-1.2 ⇒ synapse-1.2	0.02056	0.02056	−0.01086	−0.00372	−0.02659	0.16193	0.00313	−0.06274	0.23093
poi-2.0 ⇒ ant-1.6	0.31631	0.31631	0.3103	0.32147	0.31631	0.11177	0.31329	0.28237	0.3324
synapse-1.2 ⇒ ant-1.6	0.28572	0.29169	0.30237	0.2856	0.28473	0.25716	0.29918	0.31243	0.31474
camel-1.2- ⇒ ant-1.6	0.06981	0.06981	0.09539	0.09182	0.08431	0.07549	0.10807	0.22748	0.18147
log4j-1.1 ⇒ ant-1.6	0.27726	0.27726	0.29384	0.27726	0.31083	0.38494	0.29384	0.32948	0.32528
jedit-4.0 ⇒ ant-1.6	0.35812	0.42032	0.37204	0.36236	0.35812	0.16285	0.37795	0.33238	0.39119
xerces-1.2 ⇒ ant-1.6	0.00586	0.00586	0.0264	0.02115	−0.00586	0.1586	0.04636	0.12517	0.24598
poi-2.0 ⇒ camel-1.2	0.04845	0.04845	0.11824	0.0731	0.04845	0.06494	0.11859	0.14025	0.13895
synapse-1.2 ⇒ camel-1.2	0.05486	0.02581	0.10936	0.05313	0.0583	0.06712	0.11354	0.09542	0.09865
ant-1.6 ⇒ camel-1.2	0.04561	0.09218	0.0987	0.04673	0.04583	0.04783	0.10026	0.10816	0.10045
log4j-1.1 ⇒ camel-1.2	0.08919	0.08919	0.11222	0.09612	0.10825	0.04427	0.11057	0.10984	0.11353
jedit-4.0 ⇒ camel-1.2	0.04018	0.08971	0.07659	0.05415	0.04018	0.01809	0.08082	0.10837	0.08168
xerces-1.2 ⇒ camel-1.2	−0.01572	−0.01572	−0.00798	−0.08612	−0.01572	0.0351	−0.01124	−0.09313	0.09873
poi-2.0 ⇒ log4j-1.1	0.35992	0.35992	0.37309	0.35256	0.36121	0.11348	0.35028	0.23715	0.25212
synapse-1.2 ⇒ log4j-1.1	0.37414	0.34903	0.38719	0.41234	0.3777	0.23871	0.3947	0.3325	0.40296
ant-1.6 ⇒ log4j-1.1	0.57342	0.61782	0.49759	0.55578	0.55568	0.39557	0.50976	0.45798	0.46938
camel-1.2- ⇒ log4j-1.1	0.19723	0.19723	0.19723	0.25103	0.19723	0.0639	0.18827	0.24226	0.22693
jedit-4.0 ⇒ log4j-1.1	0.38971	0.31301	0.42013	0.4096	0.38715	0.0013	0.48522	0.40331	0.378
xerces-1.2 ⇒ log4j-1.1	0.02629	0.02629	0.0231	−0.10293	−0.09771	0.0604	−0.1124	−0.02266	0.22173
poi-2.0 ⇒ jedit-4.0	0.2523	0.2523	0.24064	0.22643	0.2523	0.13598	0.25664	0.27847	0.25459
synapse-1.2 ⇒ jedit-4.0	0.18899	0.1818	0.21759	0.25309	0.19549	0.25905	0.27109	0.28933	0.28531
ant-1.6 ⇒ jedit-4.0	0.36555	0.29402	0.35571	0.36171	0.3738	0.43791	0.35168	0.35186	0.35506
camel-1.2- ⇒ jedit-4.0	0.0011	0.0011	0.03118	0.16696	−0.01053	0.08204	0.18669	0.2064	0.22254
log4j-1.1 ⇒ jedit-4.0	0.19852	0.19852	0.2391	0.2135	0.21254	0.37538	0.24477	0.25452	0.27873
xerces-1.2 ⇒ jedit-4.0	−0.08398	−0.08398	−0.0932	0.3206	−0.09301	0.06644	0.32188	0.30997	0.31651
poi-2.0 ⇒ xerces-1.2	0.08079	0.08079	0.11773	0.07624	0.08079	−0.01301	0.09186	−0.01771	0.08854
synapse-1.2 ⇒ xerces-1.2	−0.03109	0.02916	−0.02039	−0.01354	−0.0329	0.05872	−0.02186	−0.04965	0.07668
ant-1.6 ⇒ xerces-1.2	0.07711	0.11999	0.10634	0.07585	0.07528	0.1022	0.1034	0.06771	0.14325
camel-1.2- ⇒ xerces-1.2	−0.21544	−0.21544	−0.19035	−0.21143	−0.20842	−0.08365	−0.16711	−0.09859	0.01437
log4j-1.1 ⇒ xerces-1.2	−0.02272	−0.02272	−0.01153	−0.02272	−0.03044	0.10208	−0.01087	0.01852	0.13313
jedit-4.0 ⇒ xerces-1.2	0.06524	0.10245	0.10635	0.0711	0.06524	0.09268	0.10639	0.05656	0.16396
Average	0.13551	0.14112	0.14956	0.15157	0.13112	0.12887	0.16283	0.16352	0.20694

Table 17: The g-measure in 10% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.37852	0.38743	0.38522	0.37208	0.3733	0.5043	0.3782	0.43156	0.4624
ant-1.6 ⇒ poi-2.0	0.49228	0.5637	0.48636	0.47518	0.48545	0.49406	0.4633	0.48664	0.48378
camel-1.2- ⇒ poi-2.0	0.19881	0.19881	0.20069	0.2152	0.19309	0.34885	0.2201	0.33574	0.3712
log4j-1.1 ⇒ poi-2.0	0.38143	0.38143	0.36505	0.38417	0.38082	0.50291	0.36586	0.45224	0.40139
jedit-4.0 ⇒ poi-2.0	0.51015	0.61152	0.48873	0.4971	0.51015	0.25283	0.48276	0.47325	0.50172
xerces-1.2 ⇒ poi-2.0	0.0531	0.0531	0.12413	0.00605	0	0.56005	0.01794	0.18421	0.46903
poi-2.0 ⇒ synapse-1.2	0.58467	0.58467	0.58286	0.66191	0.58467	0.21403	0.62466	0.62105	0.61717
ant-1.6 ⇒ synapse-1.2	0.66187	0.65538	0.65104	0.65489	0.65734	0.44687	0.65292	0.6483	0.67149
camel-1.2- ⇒ synapse-1.2	0.18959	0.18959	0.19477	0.2478	0.18932	0.32853	0.2513	0.39922	0.37584
log4j-1.1 ⇒ synapse-1.2	0.47605	0.47605	0.48289	0.47768	0.4688	0.57965	0.46992	0.56855	0.55189
jedit-4.0 ⇒ synapse-1.2	0.67344	0.68803	0.67078	0.66483	0.67344	0.26639	0.66614	0.66012	0.65957
xerces-1.2 ⇒ synapse-1.2	0.11254	0.11254	0.1148	0.09392	0.06524	0.53258	0.11355	0.10897	0.5117
poi-2.0 ⇒ ant-1.6	0.59134	0.59134	0.5885	0.58462	0.59134	0.29351	0.58408	0.5641	0.62884
synapse-1.2 ⇒ ant-1.6	0.532	0.57739	0.54951	0.52738	0.53069	0.59593	0.54842	0.56569	0.59318
camel-1.2- ⇒ ant-1.6	0.23632	0.23632	0.24248	0.24324	0.25566	0.25993	0.25879	0.38817	0.37878
log4j-1.1 ⇒ ant-1.6	0.55325	0.55325	0.55651	0.55325	0.55619	0.70158	0.55651	0.63933	0.61025
jedit-4.0 ⇒ ant-1.6	0.69973	0.73252	0.70124	0.70195	0.69973	0.41869	0.70513	0.67816	0.7108
xerces-1.2 ⇒ ant-1.6	0.09783	0.09783	0.16135	0.22824	0.07157	0.47352	0.2798	0.37064	0.53637
poi-2.0 ⇒ camel-1.2	0.51661	0.51661	0.49902	0.53249	0.51661	0.17727	0.49547	0.44585	0.51687
synapse-1.2 ⇒ camel-1.2	0.50282	0.49841	0.51023	0.50269	0.50475	0.4971	0.51272	0.47944	0.50491
ant-1.6 ⇒ camel-1.2	0.52141	0.44616	0.54433	0.52189	0.52157	0.37025	0.54393	0.53672	0.5419
log4j-1.1 ⇒ camel-1.2	0.54255	0.54255	0.48839	0.54554	0.55127	0.47929	0.48773	0.49322	0.46831
jedit-4.0 ⇒ camel-1.2	0.50492	0.50363	0.53477	0.50474	0.50492	0.18205	0.53619	0.5315	0.53147
xerces-1.2 ⇒ camel-1.2	0.13714	0.13714	0.22179	0.06415	0.13714	0.43101	0.20379	0.17879	0.53806
poi-2.0 ⇒ log4j-1.1	0.58836	0.58836	0.60165	0.57982	0.58977	0.28165	0.57299	0.48678	0.56466
synapse-1.2 ⇒ log4j-1.1	0.68076	0.68347	0.6755	0.71227	0.67235	0.46088	0.69747	0.62817	0.70121
ant-1.6 ⇒ log4j-1.1	0.75194	0.7296	0.73671	0.76178	0.73516	0.45769	0.75698	0.70004	0.69043
camel-1.2- ⇒ log4j-1.1	0.28088	0.28088	0.28088	0.39386	0.28088	0.18111	0.32507	0.4569	0.44371
jedit-4.0 ⇒ log4j-1.1	0.54739	0.58243	0.6394	0.61925	0.54959	0.20014	0.71771	0.67072	0.65009
xerces-1.2 ⇒ log4j-1.1	0.10637	0.10637	0.10636	0.02621	0	0.44137	0.02134	0.28553	0.44933
poi-2.0 ⇒ jedit-4.0	0.60899	0.60899	0.57241	0.62868	0.60899	0.34885	0.61435	0.61757	0.61819
synapse-1.2 ⇒ jedit-4.0	0.50139	0.52859	0.52993	0.55467	0.49543	0.63409	0.57699	0.6033	0.62777
ant-1.6 ⇒ jedit-4.0	0.68498	0.6551	0.6743	0.69606	0.69184	0.64182	0.68721	0.69636	0.69478
camel-1.2- ⇒ jedit-4.0	0.28746	0.28746	0.2902	0.55845	0.28685	0.26934	0.50835	0.50782	0.55998
log4j-1.1 ⇒ jedit-4.0	0.5373	0.5373	0.5491	0.53082	0.52683	0.71009	0.54828	0.63189	0.61715
xerces-1.2 ⇒ jedit-4.0	0.08036	0.08036	0.09621	0.66562	0.05347	0.37698	0.67419	0.64884	0.66284
poi-2.0 ⇒ xerces-1.2	0.51351	0.51351	0.53368	0.51202	0.51351	0.27437	0.51329	0.47294	0.52542
synapse-1.2 ⇒ xerces-1.2	0.44296	0.46166	0.45367	0.45035	0.44232	0.42122	0.45457	0.44428	0.47218
ant-1.6 ⇒ xerces-1.2	0.4435	0.41774	0.49036	0.44462	0.44454	0.26058	0.4848	0.4969	0.50633
camel-1.2- ⇒ xerces-1.2	0.35625	0.35625	0.37207	0.35817	0.36018	0.29755	0.38461	0.43281	0.45199
log4j-1.1 ⇒ xerces-1.2	0.47281	0.47281	0.46904	0.47281	0.46902	0.4397	0.46945	0.4922	0.56267
jedit-4.0 ⇒ xerces-1.2	0.41581	0.48428	0.4961	0.42957	0.41581	0.30107	0.49758	0.49002	0.53473
Average	0.43927	0.44549	0.45031	0.468	0.43237	0.40261	0.47439	0.50011	0.54691

Table 18: The balance in 10% target project data.

Source- ⇒ Target	NB	adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.45053	0.45577	0.45203	0.44637	0.4475	0.52674	0.44815	0.47638	0.49994
ant-1.6 ⇒ poi-2.0	0.51697	0.56402	0.5139	0.50463	0.51199	0.50894	0.49688	0.49851	0.50736
camel-1.2- ⇒ poi-2.0	0.36717	0.36717	0.36801	0.37586	0.36463	0.45121	0.37578	0.42868	0.45966
log4j-1.1 ⇒ poi-2.0	0.44956	0.44956	0.4448	0.45347	0.44956	0.52305	0.44562	0.48172	0.45991
jedit-4.0 ⇒ poi-2.0	0.52085	0.61158	0.5025	0.50591	0.52085	0.39465	0.49594	0.4839	0.51668
xerces-1.2 ⇒ poi-2.0	0.31051	0.31051	0.33844	0.29364	0.29182	0.58362	0.29797	0.3699	0.5346
poi-2.0 ⇒ synapse-1.2	0.59187	0.59187	0.59522	0.66182	0.59187	0.38585	0.62883	0.62563	0.6236
ant-1.6 ⇒ synapse-1.2	0.6612	0.6554	0.65129	0.65517	0.65728	0.4945	0.65283	0.64945	0.66957
camel-1.2- ⇒ synapse-1.2	0.36607	0.36607	0.36822	0.39936	0.36541	0.44214	0.39983	0.47044	0.46332
log4j-1.1 ⇒ synapse-1.2	0.51455	0.51455	0.52114	0.51675	0.51087	0.5828	0.5126	0.58042	0.57031
jedit-4.0 ⇒ synapse-1.2	0.67343	0.68801	0.67076	0.66484	0.67343	0.39969	0.66611	0.66041	0.65976
xerces-1.2 ⇒ synapse-1.2	0.33425	0.33425	0.33451	0.32859	0.31604	0.55502	0.34046	0.33061	0.56864
poi-2.0 ⇒ ant-1.6	0.59787	0.59787	0.59725	0.59214	0.59787	0.42581	0.59359	0.57948	0.62958
synapse-1.2 ⇒ ant-1.6	0.55383	0.58771	0.56611	0.5505	0.55293	0.60975	0.56535	0.57771	0.60105
camel-1.2- ⇒ ant-1.6	0.38592	0.38592	0.38944	0.38972	0.39498	0.40891	0.39707	0.46428	0.45738

Table 18: Continued.

Source- ⇒ Target	NB	adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
log4j-1.1 ⇒ ant-1.6	0.56959	0.56959	0.57238	0.56959	0.57061	0.70133	0.57238	0.63964	0.61399
jedit-4.0 ⇒ ant-1.6	0.6997	0.7325	0.69905	0.70193	0.6997	0.48396	0.70282	0.67758	0.70811
xerces-1.2 ⇒ ant-1.6	0.32854	0.32854	0.35401	0.38016	0.31868	0.51756	0.40429	0.45467	0.57409
poi-2.0 ⇒ camel-1.2	0.52042	0.52042	0.52292	0.5348	0.52042	0.36165	0.52081	0.49416	0.53679
synapse-1.2 ⇒ camel-1.2	0.51409	0.50548	0.52787	0.51368	0.51586	0.51284	0.53005	0.50816	0.5222
ant-1.6 ⇒ camel-1.2	0.52247	0.48849	0.54693	0.52299	0.52261	0.44348	0.54694	0.54386	0.54582
log4j-1.1 ⇒ camel-1.2	0.54413	0.54413	0.51504	0.54733	0.5532	0.50116	0.51447	0.51754	0.50346
jedit-4.0 ⇒ camel-1.2	0.51215	0.52065	0.53679	0.51496	0.51215	0.36152	0.53842	0.54029	0.53575
xerces-1.2 ⇒ camel-1.2	0.34269	0.34269	0.3757	0.31427	0.34269	0.47251	0.36845	0.35429	0.54342
poi-2.0 ⇒ log4j-1.1	0.59443	0.59443	0.60506	0.58811	0.59552	0.4146	0.58309	0.5229	0.57495
synapse-1.2 ⇒ log4j-1.1	0.67868	0.68339	0.67215	0.71123	0.66954	0.49545	0.69535	0.62904	0.69943
ant-1.6 ⇒ log4j-1.1	0.73978	0.70812	0.73306	0.75549	0.72405	0.50476	0.75393	0.69688	0.68864
camel-1.2- ⇒ log4j-1.1	0.40873	0.40873	0.40873	0.4817	0.40873	0.374	0.43159	0.50466	0.49999
jedit-4.0 ⇒ log4j-1.1	0.56476	0.59234	0.6405	0.62092	0.56652	0.36482	0.71686	0.66711	0.65771
xerces-1.2 ⇒ log4j-1.1	0.33212	0.33212	0.33206	0.3025	0.29259	0.4908	0.30018	0.4029	0.51512
poi-2.0 ⇒ jedit-4.0	0.6136	0.6136	0.583	0.62896	0.6136	0.45681	0.61714	0.62147	0.62444
synapse-1.2 ⇒ jedit-4.0	0.53199	0.54963	0.5525	0.57084	0.52844	0.63487	0.58784	0.61063	0.63072
ant-1.6 ⇒ jedit-4.0	0.68018	0.65546	0.67056	0.69317	0.68668	0.63825	0.6848	0.69581	0.69302
camel-1.2- ⇒ jedit-4.0	0.4033	0.4033	0.40725	0.56903	0.40174	0.41578	0.53742	0.53634	0.57836
log4j-1.1 ⇒ jedit-4.0	0.55668	0.55668	0.56709	0.55289	0.55004	0.70997	0.56647	0.63306	0.62162
xerces-1.2 ⇒ jedit-4.0	0.3193	0.3193	0.32388	0.6656	0.30996	0.46062	0.67365	0.64911	0.66237
poi-2.0 ⇒ xerces-1.2	0.52881	0.52881	0.54974	0.52704	0.52881	0.40088	0.53323	0.48112	0.53969
synapse-1.2 ⇒ xerces-1.2	0.46304	0.48874	0.46972	0.47183	0.46216	0.46981	0.46984	0.45604	0.5047
ant-1.6 ⇒ xerces-1.2	0.48776	0.47759	0.51877	0.48821	0.48809	0.39799	0.5151	0.51525	0.53279
camel-1.2- ⇒ xerces-1.2	0.35711	0.35711	0.37337	0.35993	0.36175	0.38984	0.38826	0.43321	0.47712
log4j-1.1 ⇒ xerces-1.2	0.4792	0.4792	0.48112	0.4792	0.47472	0.48878	0.48155	0.50005	0.56961
jedit-4.0 ⇒ xerces-1.2	0.47161	0.51447	0.52388	0.47972	0.47161	0.42522	0.5248	0.51111	0.55701
Average	0.50141	0.50704	0.50897	0.52107	0.49851	0.48052	0.52564	0.5351	0.56982

Table 19: The *F*1 in 15% target project data.

Source- ⇒ Target	NB	adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.21693	0.21873	0.20889	0.21459	0.216	0.23915	0.2074	0.21406	0.23694
ant-1.6 ⇒ poi-2.0	0.22963	0.23248	0.22775	0.22287	0.2276	0.22163	0.22391	0.20765	0.23422
camel-1.2- ⇒ poi-2.0	0.20672	0.20672	0.20742	0.21371	0.20603	0.21389	0.20862	0.21229	0.21602
log4j-1.1 ⇒ poi-2.0	0.21097	0.21097	0.21633	0.21715	0.21203	0.22227	0.21824	0.20555	0.23494
jedit-4.0 ⇒ poi-2.0	0.22215	0.27816	0.21793	0.21141	0.22215	0.20433	0.21456	0.20702	0.23645
xerces-1.2 ⇒ poi-2.0	0.0342	0.0342	0.0957	0.00392	0	0.25374	0.02181	0.10784	0.3069
poi-2.0 ⇒ synapse-1.2	0.5466	0.5466	0.53595	0.59014	0.5466	0.47983	0.54241	0.5825	0.54787
ant-1.6 ⇒ synapse-1.2	0.60116	0.55934	0.58981	0.5912	0.59402	0.40608	0.5931	0.59125	0.58973
camel-1.2- ⇒ synapse-1.2	0.51171	0.51171	0.51249	0.52083	0.50677	0.39913	0.51986	0.54917	0.53828
log4j-1.1 ⇒ synapse-1.2	0.53645	0.53645	0.55172	0.54527	0.54186	0.50398	0.55216	0.56258	0.5724
jedit-4.0 ⇒ synapse-1.2	0.59245	0.60616	0.58368	0.58148	0.59245	0.27294	0.57969	0.58891	0.56904
xerces-1.2 ⇒ synapse-1.2	0.09605	0.09605	0.10769	0.09133	0.06804	0.45593	0.09505	0.14233	0.46483
poi-2.0 ⇒ ant-1.6	0.51504	0.51504	0.50466	0.516	0.51504	0.43684	0.51474	0.50059	0.52528
synapse-1.2 ⇒ ant-1.6	0.50586	0.51445	0.52187	0.50565	0.50603	0.50773	0.52114	0.52141	0.53755
camel-1.2- ⇒ ant-1.6	0.41543	0.41543	0.42734	0.41897	0.41849	0.25827	0.4352	0.46126	0.47368
log4j-1.1 ⇒ ant-1.6	0.50188	0.50188	0.51869	0.50356	0.51299	0.55277	0.51861	0.54114	0.54228
jedit-4.0 ⇒ ant-1.6	0.55014	0.58331	0.56847	0.55087	0.55014	0.37734	0.568	0.54585	0.56526
xerces-1.2 ⇒ ant-1.6	0.09455	0.09455	0.14754	0.17726	0.05572	0.44861	0.24714	0.30967	0.50417
poi-2.0 ⇒ camel-1.2	0.45885	0.45885	0.51939	0.46633	0.45885	0.46679	0.52088	0.53341	0.51599
synapse-1.2 ⇒ camel-1.2	0.4672	0.44736	0.50115	0.46597	0.46761	0.38933	0.50207	0.50628	0.50358
ant-1.6 ⇒ camel-1.2	0.42921	0.36418	0.49158	0.43127	0.42929	0.31302	0.49448	0.50748	0.49195
log4j-1.1 ⇒ camel-1.2	0.46468	0.46468	0.51155	0.46995	0.47722	0.39404	0.51072	0.51333	0.51548
jedit-4.0 ⇒ camel-1.2	0.4075	0.41008	0.46659	0.40652	0.4075	0.22315	0.47109	0.50134	0.46225
xerces-1.2 ⇒ camel-1.2	0.11908	0.11908	0.20851	0.05385	0.10548	0.47024	0.19383	0.14993	0.47231
poi-2.0 ⇒ log4j-1.1	0.5895	0.5895	0.58852	0.59397	0.59083	0.44481	0.59297	0.53237	0.58286
synapse-1.2 ⇒ log4j-1.1	0.61953	0.60099	0.61149	0.59674	0.61926	0.41953	0.61528	0.58504	0.60243
ant-1.6 ⇒ log4j-1.1	0.71516	0.71791	0.69483	0.71323	0.69931	0.45513	0.67631	0.63445	0.64854

Table 19: Continued.

Source- ⇒ Target	NB	adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
camel-1.2- ⇒ log4j-1.1	0.54053	0.54053	0.53921	0.54665	0.54053	0.39224	0.54324	0.56348	0.54541
jedit-4.0 ⇒ log4j-1.1	0.52924	0.52523	0.63529	0.58965	0.52809	0.26844	0.66327	0.62366	0.60732
xerces-1.2 ⇒ log4j-1.1	0.10235	0.10235	0.10108	0.15162	0	0.45645	0.15547	0.20119	0.55011
poi-2.0 ⇒ jedit-4.0	0.48248	0.48248	0.48011	0.46536	0.48248	0.43714	0.49369	0.50174	0.5123
synapse-1.2 ⇒ jedit-4.0	0.44256	0.43796	0.45721	0.46249	0.44489	0.46818	0.47944	0.49506	0.47609
ant-1.6 ⇒ jedit-4.0	0.53958	0.49645	0.52806	0.54081	0.54208	0.55206	0.53757	0.53675	0.53698
camel-1.2- ⇒ jedit-4.0	0.37449	0.37449	0.3888	0.43318	0.36995	0.24369	0.44748	0.44796	0.46533
log4j-1.1 ⇒ jedit-4.0	0.45357	0.45357	0.46913	0.46469	0.45665	0.54593	0.47622	0.4859	0.48499
xerces-1.2 ⇒ jedit-4.0	0.06583	0.06583	0.07191	0.52178	0.04438	0.38537	0.51821	0.50973	0.50487
poi-2.0 ⇒ xerces-1.2	0.26486	0.26486	0.3081	0.2612	0.26486	0.26134	0.30412	0.23711	0.30942
synapse-1.2 ⇒ xerces-1.2	0.20511	0.23249	0.22143	0.21449	0.20431	0.24319	0.2207	0.19993	0.25847
ant-1.6 ⇒ xerces-1.2	0.25349	0.26784	0.28404	0.25193	0.25337	0.19038	0.28994	0.27164	0.32317
camel-1.2- ⇒ xerces-1.2	0.144	0.144	0.16389	0.15381	0.14931	0.17171	0.1754	0.18165	0.24017
log4j-1.1 ⇒ xerces-1.2	0.22393	0.22393	0.28556	0.22342	0.22056	0.27426	0.28501	0.2357	0.30555
jedit-4.0 ⇒ xerces-1.2	0.24285	0.27615	0.2823	0.25402	0.24285	0.18466	0.2809	0.22451	0.30917
Average	0.37437	0.37436	0.39413	0.39069	0.36885	0.35966	0.41024	0.41026	0.45525

Table 20: The MCC in 15% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.04659	0.05218	0.02094	0.03902	0.04367	0.10738	0.01674	0.04017	0.08655
ant-1.6 ⇒ poi-2.0	0.07001	0.07751	0.06574	0.05316	0.06508	0.04381	0.05514	0.02188	0.07801
camel-1.2- ⇒ poi-2.0	0.00379	0.00379	0.00746	0.03819	−0.00014	0.08989	0.01393	0.03194	0.0474
log4j-1.1 ⇒ poi-2.0	0.03082	0.03082	0.04816	0.05021	0.03406	0.05728	0.05456	0.0202	0.08751
jedit-4.0 ⇒ poi-2.0	0.04495	0.15032	0.03456	0.02146	0.04495	0.16069	0.02759	0.01266	0.07243
xerces-1.2 ⇒ poi-2.0	−0.05125	−0.05125	0.00733	−0.08175	−0.07834	0.12892	−0.06523	0.0059	0.20199
poi-2.0 ⇒ synapse-1.2	0.23656	0.23656	0.19632	0.33558	0.23656	0.03225	0.21802	0.30926	0.23708
ant-1.6 ⇒ synapse-1.2	0.34686	0.2969	0.32536	0.3292	0.3346	0.26436	0.3319	0.33161	0.33882
camel-1.2- ⇒ synapse-1.2	0.08038	0.08038	0.08465	0.11162	0.05843	0.10986	0.1109	0.22164	0.17905
log4j-1.1 ⇒ synapse-1.2	0.19658	0.19658	0.23636	0.21828	0.20929	0.19156	0.23694	0.26801	0.29093
jedit-4.0 ⇒ synapse-1.2	0.34939	0.3714	0.32719	0.33413	0.34939	0.14672	0.31549	0.32838	0.29801
xerces-1.2 ⇒ synapse-1.2	−0.00326	−0.00326	−0.03605	−0.00332	−0.01833	0.07862	−0.01171	−0.01622	0.2244
poi-2.0 ⇒ ant-1.6	0.30474	0.30474	0.28521	0.30935	0.30474	0.13202	0.30594	0.28546	0.32308
synapse-1.2 ⇒ ant-1.6	0.29061	0.30088	0.32064	0.29137	0.29126	0.27988	0.31906	0.31893	0.34124
camel-1.2- ⇒ ant-1.6	0.07018	0.07018	0.11958	0.08443	0.08269	0.09672	0.1466	0.22242	0.24577
log4j-1.1 ⇒ ant-1.6	0.28212	0.28212	0.31498	0.28648	0.30999	0.34738	0.31519	0.35305	0.34967
jedit-4.0 ⇒ ant-1.6	0.35871	0.41093	0.39158	0.36008	0.35871	0.12548	0.39187	0.35393	0.38612
xerces-1.2 ⇒ ant-1.6	0.02182	0.02182	0.02674	0.02375	−0.01596	0.1707	0.06008	0.10263	0.27853
poi-2.0 ⇒ camel-1.2	0.04496	0.04496	0.13357	0.06854	0.04496	0.07154	0.13421	0.13798	0.1357
synapse-1.2 ⇒ camel-1.2	0.04688	0.01933	0.10633	0.04495	0.04734	0.06988	0.10833	0.09883	0.10611
ant-1.6 ⇒ camel-1.2	0.04721	0.09551	0.10804	0.05144	0.04687	0.0349	0.11155	0.12334	0.11053
log4j-1.1 ⇒ camel-1.2	0.07892	0.07892	0.1063	0.08713	0.09978	0.04966	0.10365	0.10861	0.11339
jedit-4.0 ⇒ camel-1.2	0.04857	0.0964	0.07586	0.05972	0.04857	0.02268	0.0849	0.11409	0.07812
xerces-1.2 ⇒ camel-1.2	−0.01659	−0.01659	0.00016	−0.1039	−0.03027	0.03386	−0.00504	−0.08476	0.0654
poi-2.0 ⇒ log4j-1.1	0.34972	0.34972	0.35085	0.35503	0.35235	0.07906	0.35555	0.23699	0.3309
synapse-1.2 ⇒ log4j-1.1	0.38475	0.36456	0.37212	0.35842	0.38393	0.24436	0.3866	0.31815	0.37756
ant-1.6 ⇒ log4j-1.1	0.59342	0.63266	0.5252	0.56517	0.5766	0.40986	0.48576	0.4221	0.50346
camel-1.2- ⇒ log4j-1.1	0.20451	0.20451	0.19945	0.21893	0.20451	0.06997	0.21107	0.25947	0.20912
jedit-4.0 ⇒ log4j-1.1	0.4058	0.33282	0.45428	0.44343	0.39787	−0.0208	0.48217	0.3947	0.41144
xerces-1.2 ⇒ log4j-1.1	0.04404	0.04404	0.02878	0.02168	−0.08211	0.09775	0.01872	−0.02838	0.3301
poi-2.0 ⇒ jedit-4.0	0.25754	0.25754	0.25715	0.23182	0.25754	0.15573	0.27907	0.29927	0.31109
synapse-1.2 ⇒ jedit-4.0	0.18904	0.17776	0.22115	0.23579	0.19514	0.2564	0.2681	0.29585	0.25511
ant-1.6 ⇒ jedit-4.0	0.37043	0.30097	0.35268	0.36711	0.37428	0.43055	0.36362	0.35846	0.36005
camel-1.2- ⇒ jedit-4.0	−0.0053	−0.0053	0.0416	0.17347	−0.0189	0.07565	0.20644	0.20924	0.24378
log4j-1.1 ⇒ jedit-4.0	0.20939	0.20939	0.24377	0.23326	0.21754	0.36934	0.25877	0.27377	0.27023
xerces-1.2 ⇒ jedit-4.0	−0.07628	−0.07628	−0.09949	0.33699	−0.08512	0.05744	0.32308	0.31095	0.31327
poi-2.0 ⇒ xerces-1.2	0.06724	0.06724	0.12141	0.06054	0.06724	−0.0024	0.11419	−0.03749	0.12921
synapse-1.2 ⇒ xerces-1.2	−0.02888	0.0293	−0.0105	−0.0099	−0.03076	0.07072	−0.01594	−0.07297	0.05125
ant-1.6 ⇒ xerces-1.2	0.07842	0.12117	0.11754	0.07635	0.07749	0.09184	0.1244	0.08072	0.179
camel-1.2- ⇒ xerces-1.2	−0.22809	−0.22809	−0.1917	−0.212	−0.21915	−0.09392	−0.162	−0.12824	0.0048
log4j-1.1 ⇒ xerces-1.2	−0.02278	−0.02278	0.08287	−0.02399	−0.03085	0.103	0.08178	−0.01724	0.14794
jedit-4.0 ⇒ xerces-1.2	0.06937	0.10106	0.11302	0.07907	0.06937	0.06818	0.10609	−0.00024	0.14148
Average	0.1379	0.14313	0.15446	0.15524	0.13274	0.1264	0.17305	0.16393	0.22013

Table 21: The g-measure in 15% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN+WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.37978	0.38917	0.37365	0.37083	0.37476	0.49478	0.36911	0.4147	0.50306
ant-1.6 ⇒ poi-2.0	0.4906	0.55866	0.48384	0.47154	0.48375	0.4827	0.46853	0.49148	0.49303
camel-1.2- ⇒ poi-2.0	0.19596	0.19596	0.20197	0.21075	0.18988	0.36438	0.21828	0.3448	0.29397
log4j-1.1 ⇒ poi-2.0	0.38523	0.38523	0.3679	0.38822	0.38347	0.50387	0.37124	0.43346	0.45466
jedit-4.0 ⇒ poi-2.0	0.50928	0.61322	0.49297	0.49749	0.50928	0.26811	0.49031	0.47657	0.52239
xerces-1.2 ⇒ poi-2.0	0.05022	0.05022	0.14541	0.00587	0	0.55676	0.03385	0.17954	0.56967
poi-2.0 ⇒ synapse-1.2	0.58754	0.58754	0.49559	0.67172	0.58754	0.22679	0.54416	0.55605	0.52322
ant-1.6 ⇒ synapse-1.2	0.66425	0.65515	0.64992	0.65893	0.66199	0.45351	0.65119	0.65525	0.64187
camel-1.2- ⇒ synapse-1.2	0.1838	0.1838	0.19169	0.24688	0.18357	0.36508	0.25475	0.38402	0.38293
log4j-1.1 ⇒ synapse-1.2	0.48096	0.48096	0.47855	0.47874	0.46979	0.59054	0.47277	0.54938	0.56176
jedit-4.0 ⇒ synapse-1.2	0.68288	0.69459	0.66618	0.67401	0.68288	0.26391	0.65798	0.65419	0.64564
xerces-1.2 ⇒ synapse-1.2	0.10611	0.10611	0.12511	0.09961	0.07392	0.48023	0.10618	0.1685	0.54312
poi-2.0 ⇒ ant-1.6	0.58395	0.58395	0.56867	0.57585	0.58395	0.34358	0.57333	0.52711	0.60425
synapse-1.2 ⇒ ant-1.6	0.53381	0.58095	0.5742	0.52854	0.53305	0.61244	0.5719	0.57707	0.62657
camel-1.2- ⇒ ant-1.6	0.23678	0.23678	0.23912	0.24841	0.25361	0.24441	0.27818	0.379	0.43903
log4j-1.1 ⇒ ant-1.6	0.55704	0.55704	0.58494	0.55706	0.55883	0.68466	0.58394	0.64095	0.64653
jedit-4.0 ⇒ ant-1.6	0.70068	0.72829	0.70375	0.70125	0.70068	0.42845	0.70279	0.68722	0.69978
xerces-1.2 ⇒ ant-1.6	0.1056	0.1056	0.17709	0.22229	0.06142	0.47947	0.32327	0.33732	0.58502
poi-2.0 ⇒ camel-1.2	0.51533	0.51533	0.52156	0.53093	0.51533	0.24648	0.51775	0.44761	0.5323
synapse-1.2 ⇒ camel-1.2	0.50191	0.49745	0.51219	0.50133	0.50178	0.48948	0.51349	0.47782	0.49583
ant-1.6 ⇒ camel-1.2	0.52197	0.44549	0.54478	0.52409	0.52189	0.3767	0.54461	0.53928	0.54381
log4j-1.1 ⇒ camel-1.2	0.53764	0.53764	0.4586	0.54117	0.54709	0.48274	0.45602	0.45489	0.44583
jedit-4.0 ⇒ camel-1.2	0.51063	0.51001	0.52665	0.50995	0.51063	0.16507	0.53245	0.52515	0.52672
xerces-1.2 ⇒ camel-1.2	0.13457	0.13457	0.25168	0.06185	0.11915	0.4286	0.23281	0.18793	0.50365
poi-2.0 ⇒ log4j-1.1	0.58792	0.58792	0.57526	0.61206	0.59079	0.13138	0.59285	0.45773	0.58384
synapse-1.2 ⇒ log4j-1.1	0.68611	0.69138	0.67088	0.68737	0.67472	0.46661	0.68608	0.6285	0.6749
ant-1.6 ⇒ log4j-1.1	0.76187	0.74172	0.76672	0.77677	0.74557	0.46983	0.75109	0.67993	0.70376
camel-1.2- ⇒ log4j-1.1	0.29166	0.29166	0.2915	0.3392	0.29166	0.24881	0.32707	0.46646	0.39334
jedit-4.0 ⇒ log4j-1.1	0.57168	0.59938	0.70678	0.64452	0.57354	0.23806	0.73169	0.65459	0.6788
xerces-1.2 ⇒ log4j-1.1	0.1097	0.1097	0.10965	0.16635	0	0.47169	0.17543	0.25202	0.61351
poi-2.0 ⇒ jedit-4.0	0.61254	0.61254	0.57688	0.63127	0.61254	0.35812	0.62851	0.60781	0.65966
synapse-1.2 ⇒ jedit-4.0	0.49942	0.52421	0.53494	0.5214	0.49379	0.62433	0.57019	0.60792	0.59458
ant-1.6 ⇒ jedit-4.0	0.68427	0.66224	0.6645	0.69702	0.68758	0.64198	0.68916	0.69761	0.6976
camel-1.2- ⇒ jedit-4.0	0.28229	0.28229	0.28921	0.56286	0.28227	0.2773	0.49589	0.50507	0.54892
log4j-1.1 ⇒ jedit-4.0	0.5373	0.5373	0.54756	0.54234	0.52519	0.70644	0.56413	0.63679	0.60983
xerces-1.2 ⇒ jedit-4.0	0.08276	0.08276	0.0964	0.67742	0.05412	0.41143	0.67796	0.64947	0.65056
poi-2.0 ⇒ xerces-1.2	0.49976	0.49976	0.53439	0.49763	0.49976	0.27721	0.53025	0.46279	0.52605
synapse-1.2 ⇒ xerces-1.2	0.44563	0.46448	0.46654	0.45506	0.44495	0.41912	0.47	0.44472	0.50265
ant-1.6 ⇒ xerces-1.2	0.44604	0.41853	0.48442	0.44448	0.44747	0.25347	0.49369	0.50467	0.51165
camel-1.0- ⇒ xerces-1.2	0.34572	0.34572	0.36466	0.35516	0.35131	0.32337	0.37791	0.41089	0.46207
log4j-1.1 ⇒ xerces-1.2	0.47163	0.47163	0.51824	0.47105	0.46774	0.43581	0.51775	0.48011	0.49797
jedit-4.0 ⇒ xerces-1.2	0.41782	0.47984	0.4901	0.43992	0.41782	0.27729	0.495	0.44573	0.53063
Average	0.44025	0.44611	0.45297	0.47141	0.4326	0.40631	0.48199	0.49243	0.55298

Table 22: The balance in 15% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.45171	0.45726	0.44444	0.44621	0.44878	0.52382	0.44125	0.46773	0.52717
ant-1.6 ⇒ poi-2.0	0.51523	0.5589	0.51045	0.50092	0.51022	0.50229	0.50002	0.50282	0.51824
camel-1.2- ⇒ poi-2.0	0.36599	0.36599	0.36868	0.3741	0.36331	0.4651	0.37579	0.433	0.41039
log4j-1.1 ⇒ poi-2.0	0.45165	0.45165	0.44577	0.45623	0.45135	0.52323	0.4482	0.47303	0.50297
jedit-4.0 ⇒ poi-2.0	0.51998	0.61328	0.50773	0.50631	0.51998	0.39607	0.50397	0.49084	0.53586
xerces-1.2 ⇒ poi-2.0	0.30975	0.30975	0.34842	0.29371	0.29195	0.57319	0.30437	0.36358	0.59052
poi-2.0 ⇒ synapse-1.2	0.5948	0.5948	0.53102	0.67158	0.5948	0.38483	0.56438	0.57829	0.55065
ant-1.6 ⇒ synapse-1.2	0.66349	0.65517	0.65023	0.65881	0.66169	0.50239	0.65142	0.65491	0.64275
camel-1.2- ⇒ synapse-1.2	0.3637	0.3637	0.36709	0.39787	0.36312	0.45855	0.40102	0.46209	0.46522
log4j-1.1 ⇒ synapse-1.2	0.51823	0.51823	0.51805	0.51772	0.51179	0.59334	0.51456	0.56733	0.57729
jedit-4.0 ⇒ synapse-1.2	0.68285	0.69455	0.6661	0.67398	0.68285	0.40083	0.65844	0.65505	0.6466
xerces-1.2 ⇒ synapse-1.2	0.33147	0.33147	0.33757	0.33025	0.3193	0.51112	0.33265	0.35587	0.57807
poi-2.0 ⇒ ant-1.6	0.59263	0.59263	0.58237	0.58593	0.59263	0.4582	0.58571	0.55207	0.61001
synapse-1.2 ⇒ ant-1.6	0.55514	0.59028	0.58462	0.55135	0.55459	0.62301	0.58316	0.58651	0.62846
camel-1.2- ⇒ ant-1.6	0.38612	0.38612	0.38828	0.39181	0.39402	0.40797	0.40681	0.45913	0.49418

Table 22: Continued.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
log4j-1.1 ⇒ ant-1.6	0.57237	0.57237	0.59414	0.57221	0.57269	0.68619	0.5933	0.63938	0.64844
jedit-4.0 ⇒ ant-1.6	0.70059	0.72826	0.69944	0.70119	0.70059	0.48302	0.6979	0.68454	0.69685
xerces-1.2 ⇒ ant-1.6	0.33165	0.33165	0.35978	0.3779	0.31499	0.52444	0.42603	0.4339	0.59258
poi-2.0 ⇒ camel-1.2	0.51892	0.51892	0.53832	0.5329	0.51892	0.39064	0.53591	0.49496	0.5461
synapse-1.2 ⇒ camel-1.2	0.51198	0.50348	0.52867	0.51122	0.512	0.50936	0.52973	0.5079	0.51858
ant-1.6 ⇒ camel-1.2	0.52312	0.48849	0.54895	0.52526	0.52299	0.44403	0.5494	0.54794	0.54893
log4j-1.1 ⇒ camel-1.2	0.53914	0.53914	0.49757	0.54293	0.54904	0.50482	0.49582	0.49588	0.49091
jedit-4.0 ⇒ camel-1.2	0.51714	0.52553	0.53159	0.51905	0.51714	0.35442	0.53711	0.53776	0.53275
xerces-1.2 ⇒ camel-1.2	0.3417	0.3417	0.38873	0.31272	0.3356	0.47286	0.38006	0.3587	0.5157
poi-2.0 ⇒ log4j-1.1	0.59462	0.59462	0.58524	0.61738	0.5969	0.34796	0.59965	0.50693	0.59667
synapse-1.2 ⇒ log4j-1.1	0.68345	0.69114	0.66876	0.6872	0.67135	0.49812	0.6831	0.6301	0.6739
ant-1.6 ⇒ log4j-1.1	0.74871	0.71986	0.76383	0.77182	0.73284	0.51209	0.74877	0.67892	0.70659
camel-1.2- ⇒ log4j-1.1	0.41416	0.41416	0.41403	0.4412	0.41416	0.39765	0.4323	0.51103	0.46983
jedit-4.0 ⇒ log4j-1.1	0.58244	0.60593	0.70455	0.64179	0.58401	0.37278	0.7286	0.65226	0.67859
xerces-1.2 ⇒ log4j-1.1	0.33354	0.33354	0.33334	0.36626	0.29267	0.50976	0.36957	0.38643	0.61712
poi-2.0 ⇒ jedit-4.0	0.61678	0.61678	0.58792	0.63153	0.61678	0.46776	0.63149	0.61272	0.66038
synapse-1.2 ⇒ jedit-4.0	0.53071	0.54635	0.55616	0.54788	0.52732	0.6287	0.58309	0.61282	0.60297
ant-1.6 ⇒ jedit-4.0	0.67888	0.66233	0.66096	0.6936	0.68201	0.63893	0.68541	0.69558	0.69523
camel-1.2- ⇒ jedit-4.0	0.40037	0.40037	0.40754	0.57313	0.39884	0.42112	0.52933	0.53487	0.56801
log4j-1.1 ⇒ jedit-4.0	0.55718	0.55718	0.56671	0.56205	0.54911	0.7063	0.57805	0.63809	0.61612
xerces-1.2 ⇒ jedit-4.0	0.32049	0.32049	0.32341	0.67693	0.3105	0.47072	0.67725	0.65003	0.65135
poi-2.0 ⇒ xerces-1.2	0.51802	0.51802	0.54813	0.51544	0.51802	0.4056	0.54489	0.47003	0.54414
synapse-1.2 ⇒ xerces-1.2	0.46474	0.49036	0.47927	0.47512	0.46381	0.4691	0.4794	0.44789	0.51412
ant-1.6 ⇒ xerces-1.2	0.48928	0.47815	0.51669	0.48814	0.48997	0.39452	0.52309	0.5225	0.54251
camel-1.2- ⇒ xerces-1.2	0.34653	0.34653	0.37001	0.35746	0.35266	0.39534	0.3882	0.41194	0.48327
log4j-1.1 ⇒ xerces-1.2	0.47862	0.47862	0.52816	0.47793	0.47401	0.48649	0.52765	0.48356	0.52944
jedit-4.0 ⇒ xerces-1.2	0.473	0.51166	0.52325	0.48624	0.473	0.4101	0.52451	0.47554	0.55351
Average	0.50216	0.50761	0.51133	0.52293	0.49886	0.48159	0.52979	0.52915	0.57317

Table 23: The $F1$ in 25% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.22157	0.2231	0.21241	0.21842	0.22022	0.21143	0.21091	0.22626	0.22908
ant-1.6 ⇒ poi-2.0	0.2202	0.22	0.21963	0.21698	0.21792	0.22091	0.21746	0.20464	0.22265
camel-1.2- ⇒ poi-2.0	0.19981	0.19981	0.20016	0.20483	0.19894	0.18356	0.20066	0.21269	0.20927
log4j-1.1 ⇒ poi-2.0	0.20682	0.20682	0.21069	0.20938	0.2065	0.21871	0.21127	0.20021	0.21657
jedit-4.0 ⇒ poi-2.0	0.22222	0.27819	0.21936	0.21415	0.22222	0.21025	0.21264	0.21727	0.23975
xerces-1.2 ⇒ poi-2.0	0.03698	0.03698	0.12947	0.01925	0	0.24169	0.03499	0.17119	0.32523
poi-2.0 ⇒ synapse-1.2	0.5302	0.5302	0.54476	0.56072	0.5302	0.50375	0.55603	0.58603	0.56622
ant-1.6 ⇒ synapse-1.2	0.59441	0.55583	0.58674	0.58661	0.58409	0.39324	0.59043	0.5966	0.58627
camel-1.2- ⇒ synapse-1.2	0.51287	0.51287	0.51254	0.53354	0.50728	0.50987	0.52904	0.5495	0.54653
log4j-1.1 ⇒ synapse-1.2	0.53147	0.53147	0.55096	0.53997	0.53873	0.49363	0.55034	0.55832	0.57484
jedit-4.0 ⇒ synapse-1.2	0.57705	0.5975	0.58251	0.5657	0.57705	0.34001	0.57443	0.58706	0.56546
xerces-1.2 ⇒ synapse-1.2	0.09981	0.09981	0.12385	0.13578	0.05687	0.42338	0.18391	0.11896	0.46766
poi-2.0 ⇒ ant-1.6	0.51313	0.51313	0.52088	0.51825	0.51313	0.37951	0.52605	0.49653	0.524
synapse-1.2 ⇒ ant-1.6	0.49141	0.50022	0.50759	0.49581	0.49085	0.48764	0.50663	0.50331	0.52212
camel-1.2- ⇒ ant-1.6	0.42134	0.42134	0.43293	0.4335	0.42499	0.23641	0.44721	0.47112	0.47851
log4j-1.1 ⇒ ant-1.6	0.50684	0.50684	0.52035	0.50685	0.51952	0.53681	0.52091	0.54352	0.53935
jedit-4.0 ⇒ ant-1.6	0.5568	0.59558	0.57592	0.55947	0.5568	0.37195	0.57714	0.57341	0.57256
xerces-1.2 ⇒ ant-1.6	0.09629	0.09629	0.20584	0.21004	0.07397	0.43389	0.41154	0.46481	0.4998
poi-2.0 ⇒ camel-1.2	0.45753	0.45753	0.53492	0.46614	0.45753	0.51613	0.53744	0.54715	0.53557
synapse-1.2 ⇒ camel-1.2	0.47598	0.45369	0.51662	0.47366	0.47918	0.40277	0.52071	0.52348	0.52179
ant-1.6 ⇒ camel-1.2	0.43041	0.36565	0.50931	0.43087	0.43073	0.31917	0.51085	0.5106	0.5052
log4j-1.1 ⇒ camel-1.2	0.47458	0.47458	0.52942	0.47878	0.48791	0.43168	0.5286	0.53337	0.53452
jedit-4.0 ⇒ camel-1.2	0.4008	0.40287	0.4978	0.40315	0.4008	0.14196	0.49773	0.50859	0.48527
xerces-1.2 ⇒ camel-1.2	0.12046	0.12046	0.24536	0.0548	0.11296	0.42407	0.23657	0.21797	0.51423
poi-2.0 ⇒ log4j-1.1	0.59097	0.59097	0.5966	0.58852	0.59469	0.50438	0.58745	0.54995	0.59083
synapse-1.2 ⇒ log4j-1.1	0.61788	0.60483	0.62101	0.62145	0.61773	0.42062	0.62703	0.59681	0.61619
ant-1.6 ⇒ log4j-1.1	0.7147	0.70824	0.67219	0.71635	0.68947	0.44983	0.67108	0.66448	0.66505

Table 23: Continued.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
camel-1.2- ⇒ log4j-1.1	0.54464	0.54464	0.53442	0.56618	0.54317	0.50904	0.53789	0.56188	0.55143
jedit-4.0 ⇒ log4j-1.1	0.48857	0.49348	0.62464	0.54785	0.49929	0.33271	0.6475	0.58833	0.60487
xerces-1.2 ⇒ log4j-1.1	0.09814	0.09814	0.09659	0.04769	0	0.35789	0.06657	0.3248	0.48249
poi-2.0 ⇒ jedit-4.0	0.48378	0.48378	0.48464	0.46474	0.48378	0.34855	0.49805	0.50193	0.51403
synapse-1.2 ⇒ jedit-4.0	0.43249	0.42865	0.45812	0.45564	0.43523	0.48247	0.47541	0.49398	0.49633
ant-1.6 ⇒ jedit-4.0	0.54366	0.49603	0.53467	0.5503	0.54875	0.56255	0.53667	0.54448	0.5338
camel-1.2- ⇒ jedit-4.0	0.38557	0.38557	0.40757	0.42807	0.38191	0.25482	0.45313	0.45366	0.48725
log4j-1.1 ⇒ jedit-4.0	0.44737	0.44737	0.47181	0.46869	0.45222	0.55102	0.48111	0.49272	0.50239
xerces-1.2 ⇒ jedit-4.0	0.05482	0.05482	0.07989	0.5114	0.03833	0.3138	0.5123	0.50494	0.52048
poi-2.0 ⇒ xerces-1.2	0.27907	0.27907	0.34832	0.27729	0.27907	0.26822	0.32793	0.25816	0.34697
synapse-1.2 ⇒ xerces-1.2	0.21435	0.24253	0.2853	0.22239	0.21374	0.24325	0.28035	0.24668	0.30398
ant-1.6 ⇒ xerces-1.2	0.23889	0.24929	0.27539	0.24249	0.2404	0.20319	0.28133	0.24288	0.30749
camel-1.2- ⇒ xerces-1.2	0.15675	0.15675	0.18071	0.1642	0.15888	0.16822	0.1997	0.198	0.26158
log4j-1.1 ⇒ xerces-1.2	0.21469	0.21469	0.28428	0.21403	0.21163	0.26254	0.28507	0.28617	0.31669
jedit-4.0 ⇒ xerces-1.2	0.22928	0.26477	0.28537	0.24437	0.22928	0.20339	0.28231	0.22202	0.30742
Average	0.37225	0.37249	0.40313	0.38972	0.36728	0.35878	0.4201	0.42511	0.46171

Table 24: The MCC in 25% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.05035	0.05501	0.02175	0.04051	0.04595	0.04378	0.01782	0.06524	0.06665
ant-1.6 ⇒ poi-2.0	0.06888	0.07001	0.06832	0.06116	0.06349	0.0672	0.06312	0.03463	0.07734
camel-1.2- ⇒ poi-2.0	−0.01118	−0.01118	−0.00903	0.01442	−0.01636	0.05507	−0.00218	0.04958	0.03677
log4j-1.1 ⇒ poi-2.0	0.0233	0.0233	0.0356	0.03109	0.02235	0.05592	0.03761	0.00941	0.05017
jedit-4.0 ⇒ poi-2.0	0.05373	0.15577	0.04613	0.03593	0.05373	0.20895	0.02971	0.04271	0.09112
xerces-1.2 ⇒ poi-2.0	−0.05347	−0.05347	0.03448	−0.07016	−0.08038	0.10638	−0.0604	−0.00094	0.22189
poi-2.0 ⇒ synapse-1.2	0.20689	0.20689	0.22937	0.28795	0.20689	0.09906	0.25892	0.32027	0.28751
ant-1.6 ⇒ synapse-1.2	0.34184	0.29659	0.32608	0.33152	0.32393	0.20288	0.33501	0.34762	0.33912
camel-1.2- ⇒ synapse-1.2	0.07531	0.07531	0.07527	0.13499	0.05106	0.1413	0.12319	0.21456	0.19087
log4j-1.1 ⇒ synapse-1.2	0.18588	0.18588	0.23634	0.20678	0.20345	0.18195	0.23487	0.25745	0.29421
jedit-4.0 ⇒ synapse-1.2	0.32527	0.35805	0.32505	0.30844	0.32527	0.15163	0.30632	0.32037	0.29081
xerces-1.2 ⇒ synapse-1.2	0.02141	0.02141	0.00235	0.05379	−0.02177	0.14213	0.0723	−0.03981	0.23907
poi-2.0 ⇒ ant-1.6	0.31232	0.31232	0.32045	0.32593	0.31232	0.10796	0.33195	0.28966	0.3342
synapse-1.2 ⇒ ant-1.6	0.27652	0.28719	0.30632	0.28793	0.27534	0.26255	0.30476	0.30008	0.33066
camel-1.2- ⇒ ant-1.6	0.06295	0.06295	0.10948	0.09873	0.07858	0.04234	0.1525	0.2285	0.23991
log4j-1.1 ⇒ ant-1.6	0.28163	0.28163	0.3078	0.282	0.31322	0.31347	0.30946	0.34729	0.33855
jedit-4.0 ⇒ ant-1.6	0.35986	0.42155	0.39313	0.36445	0.35986	0.19718	0.39551	0.38933	0.38788
xerces-1.2 ⇒ ant-1.6	0.01532	0.01532	0.07133	0.05505	0.00409	0.16364	0.22275	0.26642	0.29212
poi-2.0 ⇒ camel-1.2	0.03682	0.03682	0.14069	0.06086	0.03682	0.06813	0.14461	0.15724	0.14024
synapse-1.2 ⇒ camel-1.2	0.06439	0.03074	0.12416	0.06066	0.07132	0.06436	0.13364	0.12908	0.1365
ant-1.6 ⇒ camel-1.2	0.04939	0.09523	0.12176	0.05073	0.04967	0.03839	0.12408	0.11233	0.11659
log4j-1.1 ⇒ camel-1.2	0.09166	0.09166	0.13639	0.09786	0.11396	0.07834	0.13434	0.14573	0.15212
jedit-4.0 ⇒ camel-1.2	0.03497	0.08675	0.10638	0.05209	0.03497	0.03755	0.10656	0.10983	0.09542
xerces-1.2 ⇒ camel-1.2	−0.01612	−0.01612	0.01274	−0.09768	−0.02406	0.00661	−0.00094	−0.0585	0.11901
poi-2.0 ⇒ log4j-1.1	0.34681	0.34681	0.35831	0.34874	0.35447	0.13663	0.33868	0.25597	0.34083
synapse-1.2 ⇒ log4j-1.1	0.37894	0.36861	0.38509	0.39871	0.37735	0.23366	0.39654	0.33746	0.3779
ant-1.6 ⇒ log4j-1.1	0.5889	0.62453	0.47258	0.56073	0.55899	0.37714	0.4655	0.45587	0.46184
camel-1.2- ⇒ log4j-1.1	0.19278	0.19278	0.15317	0.23627	0.18734	0.10121	0.16937	0.23757	0.20891
jedit-4.0 ⇒ log4j-1.1	0.36819	0.29175	0.44475	0.39534	0.36253	−0.08447	0.45811	0.33994	0.39918
xerces-1.2 ⇒ log4j-1.1	0.01931	0.01931	0.00468	−0.0952	−0.09753	−0.01279	−0.04716	0.086	0.26322
poi-2.0 ⇒ jedit-4.0	0.24747	0.24747	0.25445	0.22287	0.24747	0.1061	0.27782	0.28604	0.30378
synapse-1.2 ⇒ jedit-4.0	0.17792	0.16895	0.23304	0.23225	0.18486	0.28789	0.26698	0.29587	0.30022
ant-1.6 ⇒ jedit-4.0	0.37375	0.30008	0.36137	0.38006	0.382	0.44387	0.36069	0.36978	0.35436
camel-1.2- ⇒ jedit-4.0	0.00603	0.00603	0.07677	0.15164	−0.00448	0.08119	0.20063	0.20544	0.27256
log4j-1.1 ⇒ jedit-4.0	0.19507	0.19507	0.24925	0.2402	0.20694	0.37714	0.26786	0.28423	0.30455
xerces-1.2 ⇒ jedit-4.0	−0.09664	−0.09664	−0.10951	0.32385	−0.0994	0.04146	0.32388	0.30976	0.33705
poi-2.0 ⇒ xerces-1.2	0.08842	0.08842	0.18916	0.08528	0.08842	−0.00485	0.16233	−0.00184	0.18799
synapse-1.2 ⇒ xerces-1.2	−0.02024	0.03874	0.0813	−0.00514	−0.02162	0.05058	0.07233	0.00823	0.13593
ant-1.6 ⇒ xerces-1.2	0.06153	0.0998	0.0946	0.06553	0.06257	0.10261	0.10304	0.01729	0.13992
camel-1.2- ⇒ xerces-1.2	−0.22329	−0.22329	−0.16741	−0.20899	−0.21931	−0.06122	−0.12991	−0.11638	0.03224
log4j-1.1 ⇒ xerces-1.2	−0.03289	−0.03289	0.0768	−0.03442	−0.04051	0.08967	0.07766	0.0967	0.14149
jedit-4.0 ⇒ xerces-1.2	0.05384	0.08957	0.11957	0.06745	0.05384	0.08641	0.11435	−0.02059	0.14149
Average	0.13295	0.13845	0.16239	0.15334	0.12828	0.12355	0.18224	0.17822	0.22791

Table 25: The g-measure in 25% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.38074	0.38868	0.37042	0.37675	0.37333	0.45809	0.37071	0.42617	0.45927
ant-1.6 ⇒ poi-2.0	0.49524	0.55273	0.48179	0.4813	0.48763	0.50887	0.46142	0.49206	0.48525
camel-1.2- ⇒ poi-2.0	0.19317	0.19317	0.19625	0.21137	0.18549	0.31783	0.21277	0.34897	0.2855
log4j-1.1 ⇒ poi-2.0	0.38311	0.38311	0.36507	0.38385	0.38144	0.50493	0.36358	0.42943	0.39939
jedit-4.0 ⇒ poi-2.0	0.51533	0.61828	0.48588	0.50396	0.51533	0.24225	0.47565	0.4752	0.52197
xerces-1.2 ⇒ poi-2.0	0.0548	0.0548	0.1964	0.02943	0	0.52032	0.05736	0.32597	0.59033
poi-2.0 ⇒ synapse-1.2	0.57678	0.57678	0.54783	0.64922	0.57678	0.2203	0.5796	0.59318	0.59626
ant-1.6 ⇒ synapse-1.2	0.66258	0.65539	0.64127	0.66496	0.65805	0.42423	0.65185	0.65022	0.65322
camel-1.2- ⇒ synapse-1.2	0.18791	0.18791	0.19674	0.29682	0.18763	0.4172	0.30897	0.39927	0.37863
log4j-1.1 ⇒ synapse-1.2	0.47361	0.47361	0.47211	0.46827	0.4664	0.58372	0.4644	0.5256	0.54492
jedit-4.0 ⇒ synapse-1.2	0.67074	0.68789	0.66913	0.66144	0.67074	0.26807	0.65455	0.63063	0.64789
xerces-1.2 ⇒ synapse-1.2	0.10849	0.10849	0.14066	0.1456	0.06123	0.48958	0.20351	0.13909	0.52103
poi-2.0 ⇒ ant-1.6	0.5938	0.5938	0.61449	0.59029	0.5938	0.31106	0.61738	0.53558	0.61936
synapse-1.2 ⇒ ant-1.6	0.53161	0.58131	0.57408	0.53248	0.53005	0.61485	0.57092	0.56076	0.60824
camel-1.2- ⇒ ant-1.6	0.22847	0.22847	0.24928	0.27998	0.24885	0.17634	0.32138	0.3871	0.42795
log4j-1.1 ⇒ ant-1.6	0.55516	0.55516	0.58427	0.55398	0.55763	0.64721	0.58324	0.63916	0.63472
jedit-4.0 ⇒ ant-1.6	0.69904	0.73123	0.70383	0.70135	0.69904	0.39194	0.70514	0.69837	0.70117
xerces-1.2 ⇒ ant-1.6	0.10908	0.10908	0.25724	0.26623	0.08211	0.48855	0.54038	0.54407	0.57483
poi-2.0 ⇒ camel-1.2	0.50962	0.50962	0.46286	0.52496	0.50962	0.22703	0.45602	0.38443	0.45397
synapse-1.2 ⇒ camel-1.2	0.50765	0.5009	0.49203	0.50713	0.51126	0.49803	0.49517	0.45866	0.49529
ant-1.6 ⇒ camel-1.2	0.52348	0.44842	0.52789	0.52407	0.52371	0.3825	0.52786	0.50584	0.52294
log4j-1.1 ⇒ camel-1.2	0.54332	0.54332	0.45766	0.54582	0.55343	0.52062	0.45673	0.45985	0.47235
jedit-4.0 ⇒ camel-1.2	0.50297	0.5015	0.51744	0.50551	0.50297	0.08559	0.52027	0.4984	0.5335
xerces-1.2 ⇒ camel-1.2	0.13632	0.13632	0.30054	0.0626	0.1279	0.46627	0.29136	0.27931	0.48771
poi-2.0 ⇒ log4j-1.1	0.58618	0.58618	0.59424	0.59684	0.59446	0.32934	0.57959	0.46732	0.5989
synapse-1.2 ⇒ log4j-1.1	0.68212	0.69161	0.66838	0.69969	0.67039	0.47629	0.68288	0.62527	0.67829
ant-1.6 ⇒ log4j-1.1	0.7593	0.72805	0.74169	0.77737	0.73511	0.46941	0.7354	0.71583	0.72148
camel-1.2- ⇒ log4j-1.1	0.27774	0.27774	0.27654	0.40839	0.27755	0.31619	0.33671	0.45259	0.39525
jedit-4.0 ⇒ log4j-1.1	0.52964	0.57107	0.69869	0.60524	0.54806	0.25193	0.71677	0.59919	0.67803
xerces-1.2 ⇒ log4j-1.1	0.10699	0.10699	0.10691	0.05721	0	0.43487	0.07803	0.39786	0.55067
poi-2.0 ⇒ jedit-4.0	0.60148	0.60148	0.5608	0.62539	0.60148	0.29879	0.62134	0.60345	0.65043
synapse-1.2 ⇒ jedit-4.0	0.4971	0.52296	0.553	0.52188	0.49241	0.64929	0.58782	0.63753	0.6411
ant-1.6 ⇒ jedit-4.0	0.68967	0.65968	0.67333	0.70764	0.69636	0.6496	0.68749	0.70436	0.69073
camel-1.2- ⇒ jedit-4.0	0.29305	0.29305	0.30287	0.55169	0.29398	0.2775	0.49509	0.49543	0.57816
log4j-1.1 ⇒ jedit-4.0	0.53395	0.53395	0.55166	0.55531	0.52358	0.70935	0.56675	0.63955	0.62187
xerces-1.2 ⇒ jedit-4.0	0.07054	0.07054	0.11215	0.67344	0.04768	0.3508	0.66916	0.64845	0.67185
poi-2.0 ⇒ xerces-1.2	0.51905	0.51905	0.59801	0.51804	0.51905	0.19585	0.57087	0.45156	0.59704
synapse-1.2 ⇒ xerces-1.2	0.44777	0.46602	0.52097	0.45635	0.44726	0.39801	0.51412	0.48908	0.50783
ant-1.6 ⇒ xerces-1.2	0.42268	0.39351	0.49441	0.42822	0.42601	0.27351	0.5022	0.48711	0.52555
camel-1.2- ⇒ xerces-1.2	0.35254	0.35254	0.38542	0.36099	0.35484	0.29099	0.40415	0.42073	0.48448
log4j-1.1 ⇒ xerces-1.2	0.46503	0.46503	0.50713	0.46425	0.46132	0.42764	0.50804	0.50975	0.55772
jedit-4.0 ⇒ xerces-1.2	0.40958	0.47478	0.49183	0.43955	0.40958	0.30975	0.49161	0.46258	0.54245
Average	0.4378	0.44367	0.46055	0.47416	0.43104	0.40177	0.49139	0.50464	0.55494

Table 26: The balance in 25% target project data.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
synapse-1.2 ⇒ poi-2.0	0.45286	0.45752	0.44299	0.44935	0.44856	0.49239	0.44178	0.47822	0.49939
ant-1.6 ⇒ poi-2.0	0.51809	0.55355	0.50977	0.5084	0.51246	0.52195	0.49744	0.50653	0.51354
camel-1.2- ⇒ poi-2.0	0.36385	0.36385	0.36521	0.37253	0.36047	0.43848	0.37173	0.43685	0.40637
log4j-1.1 ⇒ poi-2.0	0.44931	0.44931	0.44305	0.45116	0.44832	0.52249	0.44257	0.46784	0.46283
jedit-4.0 ⇒ poi-2.0	0.52593	0.61834	0.50653	0.51409	0.52593	0.39103	0.49672	0.49868	0.53937
xerces-1.2 ⇒ poi-2.0	0.31134	0.31134	0.36846	0.30245	0.29198	0.54971	0.31367	0.42632	0.60543
poi-2.0 ⇒ synapse-1.2	0.58437	0.58437	0.56784	0.64926	0.58437	0.38635	0.59703	0.60794	0.60944
ant-1.6 ⇒ synapse-1.2	0.66196	0.65543	0.64166	0.66481	0.6581	0.48192	0.65153	0.64912	0.65365
camel-1.2- ⇒ synapse-1.2	0.36525	0.36525	0.36895	0.42918	0.36454	0.49336	0.43427	0.47079	0.46723
log4j-1.1 ⇒ synapse-1.2	0.51308	0.51308	0.51408	0.51076	0.50945	0.58665	0.50915	0.55095	0.56559
jedit-4.0 ⇒ synapse-1.2	0.67072	0.68782	0.66922	0.66145	0.67072	0.40558	0.65487	0.63341	0.64832
xerces-1.2 ⇒ synapse-1.2	0.33275	0.33275	0.34437	0.35097	0.31475	0.52977	0.38293	0.34312	0.57644
poi-2.0 ⇒ ant-1.6	0.5999	0.5999	0.62083	0.59627	0.5999	0.43512	0.62193	0.55777	0.62335
synapse-1.2 ⇒ ant-1.6	0.55368	0.59083	0.5848	0.55391	0.55262	0.62334	0.5823	0.5747	0.61267
camel-1.2- ⇒ ant-1.6	0.3822	0.3822	0.39278	0.41176	0.39164	0.36756	0.43063	0.46381	0.48857

Table 26: Continued.

Source- ⇒ Target	NB	Adaboost	NB + WP	NN	DBSCAN filter	VCB + SVM	NN + WP	DTB	Tsbagging
log4j-1.1 ⇒ ant-1.6	0.57104	0.57104	0.59376	0.5701	0.57172	0.65392	0.59295	0.63918	0.63667
jedit-4.0 ⇒ ant-1.6	0.69898	0.73113	0.69878	0.70129	0.69898	0.47066	0.69973	0.694	0.69674
xerces-1.2 ⇒ ant-1.6	0.33304	0.33304	0.39654	0.39944	0.32276	0.52264	0.57063	0.57548	0.58785
poi-2.0 ⇒ camel-1.2	0.51396	0.51396	0.5035	0.52781	0.51396	0.38564	0.49994	0.46032	0.49819
synapse-1.2 ⇒ camel-1.2	0.51858	0.5079	0.51921	0.51764	0.52198	0.51303	0.52212	0.4995	0.52198
ant-1.6 ⇒ camel-1.2	0.5245	0.4901	0.54044	0.52513	0.52469	0.44802	0.5408	0.52583	0.53662
log4j-1.1 ⇒ camel-1.2	0.54505	0.54505	0.49995	0.5478	0.55557	0.52915	0.49924	0.5018	0.51005
jedit-4.0 ⇒ camel-1.2	0.51003	0.51903	0.53133	0.51508	0.51003	0.32415	0.53294	0.52027	0.53973
xerces-1.2 ⇒ camel-1.2	0.34239	0.34239	0.40962	0.31333	0.33901	0.48501	0.40463	0.39336	0.51522
poi-2.0 ⇒ log4j-1.1	0.59347	0.59347	0.59999	0.60379	0.59994	0.43996	0.58937	0.51215	0.60637
synapse-1.2 ⇒ log4j-1.1	0.67974	0.69145	0.66444	0.6981	0.66762	0.50449	0.67879	0.62701	0.67604
ant-1.6 ⇒ log4j-1.1	0.74734	0.70581	0.74037	0.77371	0.7252	0.51072	0.73268	0.71018	0.71707
camel-1.2- ⇒ log4j-1.1	0.4079	0.4079	0.40682	0.49001	0.40775	0.43494	0.43615	0.50225	0.46972
jedit-4.0 ⇒ log4j-1.1	0.55449	0.58447	0.69996	0.6103	0.56667	0.37545	0.71172	0.6061	0.67913
xerces-1.2 ⇒ log4j-1.1	0.3326	0.3326	0.33229	0.31309	0.29256	0.46839	0.32197	0.47603	0.57913
poi-2.0 ⇒ jedit-4.0	0.60714	0.60714	0.57524	0.62555	0.60714	0.42804	0.62399	0.60867	0.65142
synapse-1.2 ⇒ jedit-4.0	0.52848	0.54466	0.56944	0.5476	0.52596	0.65162	0.5965	0.64051	0.6422
ant-1.6 ⇒ jedit-4.0	0.68443	0.65984	0.66896	0.70427	0.6908	0.64504	0.68379	0.70218	0.68779
camel-1.2- ⇒ jedit-4.0	0.40611	0.40611	0.41609	0.5621	0.40527	0.42345	0.52901	0.52824	0.58995
log4j-1.1 ⇒ jedit-4.0	0.55422	0.55422	0.5696	0.57221	0.54772	0.70964	0.5813	0.64033	0.62625
xerces-1.2 ⇒ jedit-4.0	0.31544	0.31544	0.328	0.67324	0.30776	0.4441	0.66808	0.64885	0.66922
poi-2.0 ⇒ xerces-1.2	0.53355	0.53355	0.60671	0.53238	0.53355	0.37095	0.58673	0.47761	0.60446
synapse-1.2 ⇒ xerces-1.2	0.46897	0.49377	0.53014	0.47741	0.46829	0.4576	0.52391	0.4954	0.53213
ant-1.6 ⇒ xerces-1.2	0.47441	0.46305	0.51873	0.47794	0.47635	0.40588	0.5252	0.49669	0.54488
camel-1.2- ⇒ xerces-1.2	0.35332	0.35332	0.38889	0.36261	0.35591	0.39275	0.41178	0.42222	0.49784
log4j-1.1 ⇒ xerces-1.2	0.47203	0.47203	0.52595	0.47112	0.46759	0.48071	0.52682	0.52959	0.56616
jedit-4.0 ⇒ xerces-1.2	0.46699	0.50779	0.52146	0.48472	0.46699	0.42984	0.52165	0.47568	0.55946
Average	0.50056	0.50585	0.51659	0.52438	0.49775	0.47932	0.53669	0.53751	0.57654

requirements for the distribution of dataset, and it is more convenient to test whether there are differences between the two groups of distribution. The results of the Wilcoxon rank sum test show that the p value on all datasets is less than 0.05. Therefore, it can be said that our experiment is statistically significant. In addition, we use the box plot to analyze the experimental results. The box plot can show the overall distribution of the prediction results, and the results also show that Tsbagging achieves better defect prediction performance than others.

7. Conclusion and Future Work

This paper proposes a CPDP method Tsbagging based on semisupervised clustering. Tsbagging combines the semi-supervised clustering method with the transfer learning method to overcome shortcomings of traditional SDP methods in CPDP. The experimental results (the detailed description of each experimental result is shown in Tables 15–26) show that it has some advantages in performance compared with other defect prediction methods of Tsbagging. In the future work, we will try to integrate the information in multiple source projects for knowledge transfer based on the multisource transfer learning framework, so as to help the classifier achieve better performance in CPDP.

Appendix

This document shows the details of each experiment. Each experimental dataset is represented in the form of (source project ⇒ target project) in this paper.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the National Key R&D Program of China (no. 2018YFB1403400) and Natural Science Research Project of Huai'an (Grant No. HABZ202023).

References

[1] X. Guoqiang, X. Shiyi, P. Xiaohong, and L. Zhao, "Prediction of number of software defects based on SMOTE," *International Journal of Performability Engineering*, vol. 17, no. 1, pp. 123–134, 2021.

[2] W. E. Wong, X. Li, and P. A. Laplante, "Be more familiar with our enemies and pave the way forward: a review of the roles bugs played in software failures," *Journal of Systems and Software*, vol. 133, pp. 68–94, 2017.

[3] Y. Li, W. E. Wong, S.-Y. Lee, and F. Wotawa, "Using trirelation networks for effective software fault-proneness prediction," *IEEE Access*, vol. 7, Article ID 63080, 2019.

[4] X. Chen, Q. Gu, W. S. Liu, S. L. Liu, and C. Ni, "Survey of static software defect prediction," *Journal of Software*, vol. 26, no. 1, pp. 1–25, 2016.

[5] X. Chen, L. P. Wang, Q. Gu, Z. Wong, and C. Ni W. S. Liu, "A survey on cross-project software defect prediction methods," *Chinese Journal of Computers*, vol. 41, no. 1, pp. 254–274, 2018.

[6] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[7] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: the use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, 2014.

[8] K. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[9] J. Wang, B. Shen, and Y. Chen, "Compressed C4.5 Models for Software Defect Prediction," in *Proceedings of the International Conference on Quality Software IEEE*, pp. 13–16, Xi'an, China, August 2012.

[10] H. Ji, S. Huang, X. Lv, Y. Wu, and Y. Feng, "Empirical studies of a kernel density estimation based naive bayes method for software defect prediction," *IEICE - Transactions on Info and Systems*, vol. E102.D, no. 1, pp. 75–84, 2019.

[11] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Software Quality Journal*, vol. 27, no. 3, pp. 923–968, 2019.

[12] M. Li, H. Zhang, and R. Wu, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.

[13] K. K. Bejjanki, J. Gyani, and N. Gugulothu, "Class imbalance reduction (CIR): a novel approach to software defect prediction in the presence of class imbalance," *Symmetry*, vol. 12, no. 3, pp. 407–430, 2020.

[14] H. Steffen, A. Trautsch, and J. Grabowsk, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, 2017.

[15] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "A large scale experiment on data vs. domain vs. process," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Software Engineering*, pp. 91–100, Amsterdam, The Netherlands, August 2009.

[16] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[17] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 35th International Conference on Software Engineering*, pp. 382–391, San Francisco, CA, USA, May 2013.

[18] B. Turhan, T. Menzies, and A. B. Bener, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[19] K. Kawata, S. Amasaki, and T. Yokogawa, "Improving Relevancy Filter Methods for Cross-Project Defect Prediction," in *Proceedings of the 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, pp. 2–7, Okayama, Japan, July 2015.

[20] M. A. Ester, "Density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the international conference knowledge Discovery and Data Mining*, pp. 226–231, Portland, Oregon, August 1996.

[21] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Software Engineering*, vol. 21, no. 21, pp. 43–71, 2016.

[22] J. Chen, K. Hu, Y. Yang, Y. Liu, and Q. Xuan, "Collective transfer learning for defect prediction," *Neurocomputing*, vol. 416, pp. 103–116, 2020.

[23] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from Open-Source Projects: An Empirical Study on Defect Prediction," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 45–54, Baltimore, MD, USA, October 2013.

[24] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," *Information and Software Technology*, vol. 100, no. AUG, pp. 87–102, 2018.

[25] B. Turhan and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information and Software Technology*, vol. 55, no. 6, pp. 1101–1118, 2013.

[26] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, no. 1, pp. 67–77, 2015.

[27] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of the 24th international conference on Machine learning - ICML '07*, pp. 193–200, Corvalis, Oregon, July 2007.

[28] J. G. Moreno-Torres, T. Raeder, N. V. Chawla, and F. A. Herrera, "Unifying view on dataset shift in classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[29] J. kanayake, J. Tappolet, H. C. Gall, and A. Bernstein, "Tracking concept drift of software projects using defect prediction quality," in *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pp. 16-17, Vancouver, BC, Canada, May 2009.

[30] K. Wagstaff and C. Cardie, "Constrained K-means clustering with background knowledge," in *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 577–584, MA, USA, July 2001.

[31] S. Basu, A. Banerjee, and R. J. Mooney, "Semi-supervised clustering by seeding," in *Proceedings of the Nineteenth International Conference*, pp. 19–26, New South Wales, Sydney, Australia, July 2002.

[32] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[33] T. Menzies, B. Caglayan, and Z. He, "The PROMISE Repository of empirical software engineering data," 2012, http://openscience.us/repo/.

[34] H. Ji and S. Huang, "Kernel entropy component analysis with nongreedy L1-norm maximization," *Computational Intelligence and Neuroscience*, vol. 2018, Article ID 6791683, 9 pages, 2018.

[35] H. Tong, B. Liu, and S. H. Wang, "Transfer-learning Oriented Class Imbalance Learning for Cross-Project Defect Prediction," 2019, https://arxiv.org/abs/1901.08429.

[36] H. Tong, B. Liu, and S. Wang, "Kernel spectral embedding transfer ensemble for heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, p. 1, 2019.

[37] Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and

cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2017.

[38] S. Xin Yao and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[39] R. Krishna and T. Menzies, "Bellwethers: a baseline method for transfer learning," *IEEE Transactions on Software Engineering*, vol. 45, no. 11, pp. 1081–1105, 2019.

[40] S. Ren, W. Zhang, H. S. Munir, and L. Xia, "Dissimilarity space based multi-source cross-project defect prediction," *Algorithms*, vol. 12, no. 1, pp. 13–23, 2019.

[41] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[42] Y. Wu, S. Huang, H. Ji, and C. C. Zheng, "A novel Bayes defect predictor based on information diffusion function," *Knowledge-Based Systems*, vol. 144, pp. 1–8, 2018.

[43] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 969–980, 2015.

[44] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Software Quality Journal*, vol. 25, no. 1, pp. 235–272, 2017.

[45] I. H. Witten and E. Frank, "Data mining," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76-77, 2002.

[46] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[47] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.