Learning from Open-Source Projects: An Empirical Study on Defect Prediction

Zhimin He[1,2]
[1]Lab for Internet Software Technology, Institute of Software Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
hezhimin@itechs.iscas.ac.cn

Fayola Peters, Tim Menzies
Lane Department of CS & EE, West Virginia University
Morgantown, USA
fayolapeters@gmail.com, tim@menzies.us

Ye Yang
Lab for Internet Software Technology, Institute of Software Chinese Academy of Sciences
Beijing, China
yangye@nfs.iscas.ac.cn

Abstract—**The fundamental issue in cross project defect prediction is selecting the most appropriate training data for creating quality defect predictors. Another concern is whether historical data of open-source projects can be used to create quality predictors for proprietary projects from a practical point-of-view. Current studies have proposed statistical approaches to finding these training data, however, thus far no apparent effort has been made to study their success on proprietary data. Also these methods apply brute force techniques which are computationally expensive. In this work we introduce a novel data selection procedure which takes into account the similarities between the distribution of the test and potential training data. Additionally we use feature subset selection to increase the similarity between the test and training sets. Our procedure provides a comparable and scalable means of solving the cross project defect prediction problem for creating quality defect predictors. To evaluate our procedure we conducted empirical studies with comparisons to the within company defect prediction and a relevancy filtering method. We found that our proposed method performs relatively better than the filtering method in terms of both computation cost and prediction performance.**

Keywords-software defect prediction; cross-project; instance selection; feature subset selection; data similarity

I. Introduction

Software defect prediction seeks to precisely detect defect-prone modules and thus helps to allocate limited resources for testing and maintenance. Most studies conducted in this area are based on the within project learning context [1]-[5]. The cross project defect prediction problem is motivated by the fact that local historical data for model training is unavailable sometimes in practice. This field of study has therefore drawn a lot of attentions in recent years [6]-[10].

For target projects without local historical data, it is attractive to learn from publicly accessible historical data, e.g., data from open-source projects. In this paper, we explore how to make use of open-source project data to predict defects for proprietary projects. Considering the differences between development environment of open-source projects and proprietary projects, it would be risky to directly apply knowledge of open-source projects to proprietary projects. For example, a widely cited comparison study conducted by Zimmermann et al. shows that cross project defect prediction rarely succeed. [6]. Hence the following question:

Can we transfer defect prediction knowledge from open-source projects to proprietary projects?

To the best of our knowledge, this is the first study in the literature that focuses on this problem. We argue that learning from open-source projects is a practical approach for proprietary projects lacking local historical data because:

- The measurement data of open-source projects is easy and cheap to obtain;
- Companies are usually not so willing to share data of their proprietary projects because of the risk of privacy and safety [11]. So it's not easy to get data of "similar" proprietary projects from other companies.

So far, there are some studies aims to find out solutions for cross project defect predictions [7]-[9]. However, none of them demonstrated their success on proprietary projects. Another drawback of some of these work is the computation cost. For example, Turhan et at. proposed to select cross training data by using the Nearest Neighbors filter, which has a quadratic runtime complexity with regard to data set size [7]. Our previous method for training data selection finds training data in a brute force way[8]. Considering learning from a large number of open-source projects, the scalability problem is crucial, which motivates us to find more scalable solutions for cross project defect prediction.

In this paper, we investigated the feasibility of transferring defect prediction knowledge from open-source projects to proprietary projects. We proposed a new training data selection method for cross project defect prediction based on data similarity and answer the following research questions:

- RQ1: To what extent can open-source project data guide defect prediction for proprietary projects?
- RQ2: Does our proposed method perform better than existing cross project defect prediction methods in the literature?
- RQ3: Is our proposed method more scalable than existing approaches?

We conducted empirical studies on a large-scale data set comprising of 34 releases of 10 open-source projects and 34 releases of seven proprietary projects. We first organized a simple simulation experiment to verify the usability of open-source projects data for building defect prediction models for proprietary projects; then by introducing a data change analysis approach, we proposed a new training data selection method for cross project defect prediction. Comparison results between our proposed method and a state-of-the-art data filtering method demonstrate the advantages on both prediction performance and computation cost.

This study makes the following contributions:

- An investigation on the usability of open-source project data to defect prediction for proprietary projects;
- A method that helps to find out appropriate training data for cross project defect prediction.

The rest of this paper is organized as follows: Section II describes research work related to this study. Section III describes methodology including objective data, learners, and performance assessment method. Section IV presents the empirical investigation on the usability of open-source project data and the data selection approach we proposed along with its comparison to other cross project defect prediction approaches. We report the results of our experiments in Section V. We provide some discussion in Section VI and list the potential threats to the validity of our findings in Section VII. Finally, we present conclusions and future work in Section VIII.

II. Related work

A. Cross Project Defect Prediction

Software defect prediction is an effective way to optimize testing resource allocation and assist quality assurance [1]-[4]. It can identify modules that are more likely to contain defects prior to testing. In the past decade, various defect prediction models have been proposed and their effectiveness has been demonstrated in the within project context [1]-[5]. An embarrassing fact in practice is that local historical data may be unavailable sometimes, thus a hot topic of defect prediction is learning from other projects, i.e., cross project defect prediction.

Zimmermann et al. pointed out that cross project defect prediction failed in most cases training data is not selected carefully [6]. Turhan et al. suggested using the nearest neighbor filtering (KNN filter) method to improve performance of cross project defect prediction [7]. The KNN filter method was further applied to effort estimation by Kocaguneli et al. [12]. Watanabe et al. tried to adapt fault prediction models from a Java project to a C++ project. They achieved the goal by using a method called "metrics compensation" which is essentially a sort of data transformation [11]. Another recent work of Ma et al. proposed to use an improved Transfer Naïve Bayes (TBN) learner to handle the cross project defect prediction problem [9]. One of our previous study focus on the feasibility of cross project defect prediction, and we proposed to use data distribution characteristics to guide the selection of appropriate training data [8].

Most of aforementioned studies draw their conclusions based on open-source project data. This study is different for its focus on transferring defect prediction knowledge from open-source projects to proprietary projects. Also, some proposed methods such as the KNN filter [7] and our previous data selection method in [8] are computationally expensive, in this study we seek to find out cross project prediction methods that are more scalable.

B. Data Similarity

A key problem of transferring defect prediction knowledge from one project to another is the differences between training data and test data. Intuitively, the target project should learn from source projects with similar properties, or from a mathematic point-of-view, learn from training data with similar distributions. The problem of measuring the similarity between distributions of two data sets attracted lots of attentions in the past decades, other concepts such as "two-sample test", "data change detection", and "data shift detection" are also related to this topic [13]. Gretton et al. proposed a kernel method to measure the distance between two data sets which is called Maximum Mean Discrepancy (MMD) [14]. The MMD measure has become very popular in recent years, however, it is troubled by the computation cost: MMD has at least quadratic runtime complexity with regard to training set size. Another empirical framework proposed by Hido et al. also aims at measuring data similarity [15]. They proposed a *virtual classifier* approach to examine the distance between two data sets. Hido's framework was further extended by Dries et al. in their recent work [16]. Other research work such as [17] and [18] are also related to data similarity measuring.

Besides data similarity measuring and data change detection, there are also research work focusing on reducing discrepancy between data sets, which can be mostly found in the field of transfer learning [19]. The proposed methods mainly focus on two directions: instance selection and feature selection. For example, the Kernel Mean Match (KMM) method proposed by Huang et al. tries to reduce the MMD distance between data sets by assigning different weights to each instance of training set [20]. And feature selection methods such as Structural Correspondence Learning [21], Maximum Mean Discrepancy Embedding [22], and Transfer Component Analysis [23] et al. try to find out a reduced feature space where the distance between training data and test data is minimized.

In this study, we adopt Hido's framework [15] to measure similarity between cross project training data and test data, and then learn defect prediction models from close projects (i.e., instance selection). We further reduce data discrepancy by removing some unstable features (i.e., feature subset selection). To the best of our knowledge, our study is the first one that adopts both instance selection and feature subset selection in the cross project defect prediction research filed.

III. Methodology

A. Data

In this study, a total number of 68 datasets (34 releases of 10 open-source projects and 34 releases of seven proprietary projects) were investigated. These data sets were collected and shared by Jureczko et al. [24], [25]. Each instance in a data set represents a java class of the release and consists of two parts: instance features including 20 static code metrics and a labeled feature "bug" indicating how many defects in that class. In this study, we consider a class as defect-free (DF) if the value of bug is equal to 0; otherwise, defect-prone (DP). The goal of defect predictions in this study is to identify defect-prone classes precisely. These data sets are publicly accessible and have been used in many previous studies, which make it easy to repeat and compare our experiments to other related studies. Table I and Table II provide more details of the objective data sets and the static code metrics used to construct defect prediction models.

TABLE I. OBJECTIVE DATA SETS

Open-source project data				Proprietary project data			
Dataset	**Size**	**#DP**	**%DP**	**Dataset**	**Size**	**#DP**	**%DP**
ant-1.3	187	20	10.7	p1-v128	3619	220	6.1
ant-1.4	265	40	15.1	p1-v164	3541	319	9.0
ant-1.5	401	32	8.0	p1-v192	3692	85	2.3
ant-1.6	523	92	17.6	p1-v44	4081	376	9.2
ant-1.7	1066	166	15.6	p1-v9	4455	149	3.3
camel-1	436	13	3.0	p1-v92	3670	1287	35.1
camel-1.2	765	216	28.2	p2-v225	1864	147	7.9
camel-1.4	1122	145	12.9	p2-v236	2403	76	3.2
camel-1.6	1252	188	15.0	p2-v245	2023	103	5.1
ivy-1.1	135	63	46.7	p2-v256	2025	625	30.9
ivy-1.4	321	16	5.0	p2-v265	2372	229	9.7
ivy-2	477	40	8.4	p2-v276	2472	334	13.5
jEdit-3.2.1	508	90	17.7	p3-v285	1709	177	10.4
jEdit-4	606	75	12.4	p3-v292	2330	209	9.0
jEdit-4.1	644	79	12.3	p3-v305	2388	89	3.7
lucene-2	288	91	31.6	p3-v318	2440	365	15.0
lucene-2.2	381	144	37.8	p4-v347	2906	162	5.6
lucene-2.4	536	203	37.9	p4-v355	2802	924	33.0
poi-1.5	300	141	47.0	p4-v362	2865	213	7.4
poi-2.0RC1	386	37	9.6	p42-v452	317	33	10.4
poi-2.5.1	466	248	53.2	p42-v453	259	20	7.7
poi-3	531	281	52.9	p42-v454	295	13	4.4
synapse-1	162	16	9.9	p43-v461	1730	32	1.9
synapse-1.1	230	60	26.1	p43-v472	1740	32	1.8
synapse-1.2	269	86	32.0	p43-v481	1884	33	1.8
velocity-1.4	224	147	65.6	p43-v492	1888	27	1.4
velocity-1.5	246	142	57.7	p43-v501	2172	83	3.8
velocity-1.6	261	78	29.9	p43-v512	2265	134	5.9
xalan-2.4	862	110	12.8	p5-v4	3514	264	7.5
xalan-2.5	945	387	41.0	p5-v40	3815	466	12.2
xalan-2.6	1170	411	35.1	p5-v85	3509	930	26.5
xerces-init	206	77	37.4	p5-v121	3445	425	12.3
xerces-1.2	515	71	13.8	p5-v157	2863	367	12.8
xerces-1.3	545	69	12.7	p5-v185	3260	268	8.2
mean	507	120	25.7	mean	2547	271	9.9
median	451	88	17.7	median	2421	193	7.8

B. Learning Algorithms

In this study we use three learners widely used in defect prediction research. First, Random Forest (RF) is selected for its good performance. A benchmarking study done by Lessmann et al. showed that RF performs significantly better than 21 other predictors [4]. Second, we select Naïve Bayes according to recommendation of Menzies et al. which reported that the simple NB learner performs as well as complex learners [2]. Third, Logistic Regression (LR) is favored by Zimmermann et al. [6].

1) Random Forest. RF is a collection of decision trees where each tree is learned from a bootstrap sample (randomly sampling the data with replacement) [26]. The features used to find the best split at each node is a randomly chosen subset of the total number of features. Each tree in the collection is used to classify a new instance. The forest then selects a classification by choosing the majority result.

2) Naive Bayes. NB is a statistical learning algorithm that assumes features are equally important and statistically independent. The NB learner builds its classification power based on the Baye's rule shown below [27]:

$$P(c_k|x) = P(c_k)P(x|c_k)/P(x) \quad (1)$$

where c_k is a member of the set of class label values (in this study c_k could be "defect-prone" or "defect-free"), x represents a test instance. NB computes the conditional probability of the test instance being labeled c_k. The c_k with the highest probability is chosen as the label for x.

3) Logistic Regression. Afzal recommend LR when the dependent variable is dichotomous [28]. LR avoids the Gaussian assumption used in standard Naive Bayes. The form of the logistic regression model is:

$$log(p/(1-p)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k \quad (2)$$

where p is the probability that defect would be found in the module and $x_1, x_2, \ldots, x_k$ are the independent variables. $\beta_0, \beta_1, \ldots, \beta_k$ are the regression coefficients estimated using maximum likelihood.

C. Data Preprocess

We adopted some data preprocess operations to address quality problems of the experimental data.

1) Data sampling. in each prediction we applied under-sampling to training data to reduce the effects of imbalance distribution (i.e., the ratio of defect-free classes is usually much higher than that of defect-prone classes). During our experiments we found that under-sampling helps to improve prediction performance significantly, which is consistent with what Menzies et al. found [29].

2) Normalization. We normalize each attribute to the 0-1 intervals to reduce the effect of different value scales.

3) Discretization. For predictions using the Naïve Bayes learner, we convert continuous attributes into nominal values using 5 equal-frequency bins. The reason why we adopted equal-frequency is that distributions of most attributes are seriously skewed in our experimental data sets. For leaners that can handle numerical attributes (i.e., Random Forest, Logistic Regression), we input the original data without discretization operation.

D. Performance Assessment

In the context of defect prediction, defect-prone classes are usually considered as positive instances. Hence the four categories of defect prediction results are defined as below:

- *TP (True Positive)*: defect-prone classes that are classified correctly;
- *FN (False Negative)*: defect-prone classes that are wrongly classified to be defect-free;
- *TN (True Negative)*: defect-free classes that are classified correctly;
- *FP (False Positive)*: defect-free classes that are wrongly classified to be defect- prone.

TABLE II. DESCRIPTION OF THE STATIC CODE METRICS USED FOR DEFECT PREDICTION

Attributes	Symbols	Description
Weighted Methods per Class	WMC	The number of methods in the class (assuming unity weights for all methods)
Depth of Inheritance Tree	DIT	Provides the position of the class in the inheritance tree
Number of Children	NOC	Measures the number of immediate descendants of the class
Coupling Between Object classes	CBO	Increased when the methods of one class access services of another
Response for a Class	RFC	Number of methods invoked in response to a message to the object
Lack of Cohesion in Methods	LCOM	Number of pairs of methods that do not share a reference to an instance variable
Lack of Cohesion in Methods, different from LCOM	LCOM3	If m, a are the number of methods, attributes in a class number and $\mu(a)$ is the number of methods accessing an attribute, then $\mathrm{lcom3} = \left(\left(\frac{1}{a}\sum_j^a \mu(a_j)\right) - \mathrm{m}\right)/(1-\mathrm{m})$
Number of Public Methods	NPM	Counts all the methods in a class that are declared as public
Data Access Metric	DAM	Ratio of the number of private (protected) attributes to the total number of attributes
Measure of Aggregation	MOA	Count of the number of data declarations (class fields) whose types are user defined classes
Measure of Function Abstraction	MFA	Number of methods inherited by a class plus number of methods accessible by member methods of the class
Cohesion among Methods of class	CAM	Summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods
Inheritance Coupling	IC	Number of parent classes to which a given class is coupled
Coupling Between Methods	CBM	Total number of new/redefined methods to which all the inherited methods are coupled
Average Method Complexity	AMC	E.g., number of JAVA byte codes
Afferent couplings	Ca	How many other classes use the specific class
Efferent couplings	Ce	How many other classes is used by the specific class
Maximum McCabe	Max (CC)	Maximum McCabe's Cyclomatic Complexity values of methods in the same class
Average McCabe	Avg (CC)	Average McCabe's Cyclomatic Complexity values of methods in the same class
Lines of Code	LOC	Measures the volume of code
Number of Bugs	Bug	Number of bugs detected in the class

In this study we employ probability of defects (*PD*), probability of false alarm (*PF*), and the *G-measure* which is the harmonic mean of *PD* and 1-*PF*, to assess performance of defect prediction models. There is a debate on right metrics to use when evaluating software defect prediction results [30], we do not explore this as it is out of the scope of this paper. Instead, we adopt these three indicators based on the widely cited studies done by Menzies et al. [2], [3]. According to definitions of these three indicators, we favor prediction results with high *PD*, low *PF*, and high *G-measure*. Eq. (3) and (4) shows how we calculate these three measures.

$$PD = TP\,/\,(TP{+}FN) \qquad FP = FP\,/\,(TN{+}FP) \tag{3}$$

$$G = 2{*}PD{*}(1{-}PF)\,/\,(PD{+}\,(1{-}PF)) \tag{4}$$

To eliminate the potential bias caused by data sampling, we repeated each prediction by 10 times and use the median performance as the overall prediction performance.

IV. RESEARCH METHOD

To examine research questions of section I, we conduct empirical studies on aforementioned objective data sets in Table I. Our empirical studies include:

- A simulation experiment to verify the feasibility of using open-source project data to build defect prediction models for proprietary projects.
- An empirical study to verify the effectiveness and computational cost of a new proposed training data selection method. The idea behind our proposed data selection method is to select potential training data based on data similarity. More specifically, we first find cross training sets that are close to the test set, and then we find features that cause the differences between training set and test set (i.e., unstable features), finally we remove these unstable features and learn cross project defect prediction models from remaining data of close training sets.

The following two sub-sections describe details of these two empirical studies.

A. Usability of Open-Source Project Data

Before employing open-source project data to build defect prediction models for proprietary projects, we need to assess their usability for model training, i.e. we need to find evidence showing that learning from open-source project is theoretically or empirically feasible. To achieve this goal we conducted a simple simulation experiment as follows:

- *Cross project defect prediction using the most appropriate training set (**CPDP-Best**)*. For each test set (i.e., a release of proprietary projects) we train defect prediction models on each individual cross project training set (i.e., a release of open-source projects) and observed its performance on the test set. We then select the best prediction performance for each test set. The results from this simple simulation provide empirical evidence indicating whether cross project defect prediction is feasible.

We have to point out that CPDP-Best is a post-facto approach and it does not work for practical prediction settings. We use CPDP-Best here as a baseline to explore the theoretical possibility of learning from open-source projects. We compare prediction performance of CPDP-Best with those of two other baselines described below:

- *Within project defect prediction (**WPDP**)*. This baseline represents prediction performance of a within project learning scenario, i.e., 5-folds cross validation on the test set. Theoretically, performance of the within project defect prediction is the best performance we can achieve since in this scenario the training data and the test data are sampled from the same distribution.

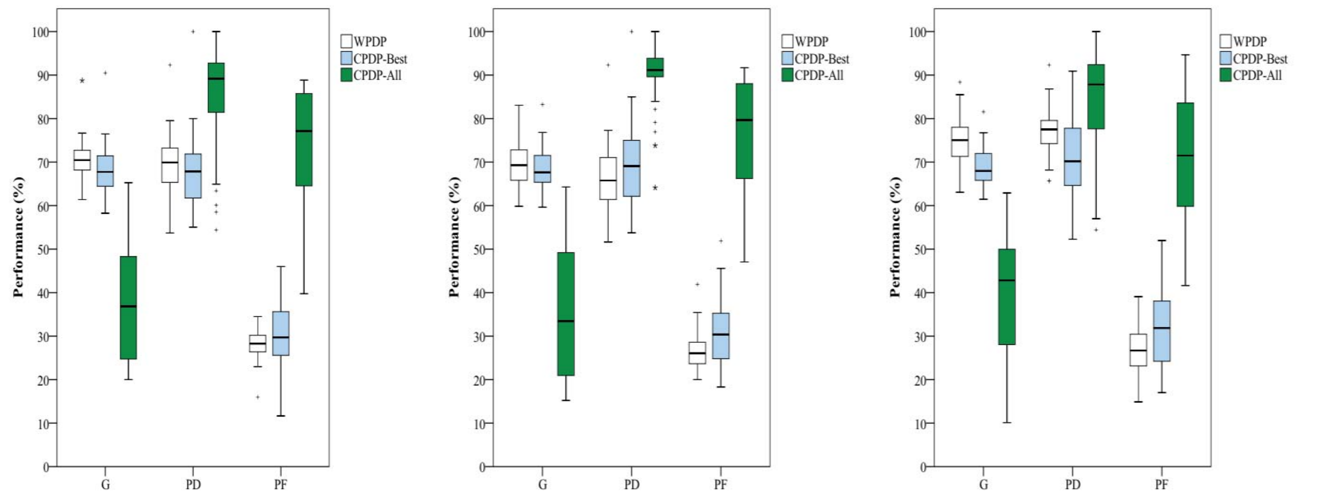
- *Cross project defect prediction using all available training data (**CPDP-All**).* This baseline represents a very intuitive cross project defect prediction setting: combine all open-source project data into a big training set without any data selection or filtering operation, and then learn prediction models from it. This cross project defect prediction would have success only if all open-source projects and proprietary projects hold the same global pattern for defect prediction. Turhan et al. found that CPDP-All would generate prediction results with very high *PF*, which was the main reason that motivated the data selection research for cross project defect prediction [7].

Fig. 1 illustrates the comparisons of prediction performance. Note that in this simulation experiment, prediction results of CPDP-Best refer to prediction results that have the greatest *G-measure* values. Intuitively we can see that performance of CPDP-Best is close to that of WPDP, and CPDP-ALL produce very bad defect prediction results in terms of *G-measure*. The finding that learning from all cross training data without data filtering will lead to higher *PF* is consist with what Turhan et al. found [7].

Results of this simulation experiment provide empirical evidence showing that learning defect prediction models from open-source project data is a feasible way for proprietary projects lacking local historical data. However, simply learning prediction models from all available open-source project training data without data selection is not a good practice. Thus we can answer our first research question:

RQ1: *To what extent can open-source project data guide defect prediction for proprietary projects?*

A1: *Empirical evidence of our simple simulation experiment confirms the potential usability of open-source project for building defect prediction models for proprietary projects. With carefully selecting training data (e.g., the cross training data leads to the highest performance), it is possible for cross project defect prediction to achieve performance close to that of within project defect prediction.*



(a) Naïve Bayes (b) Logistic Regression (c) Random Forest

Figure 1. Comparisons of prediction performance among CPDP-Best, WPDP, and CPDP-All

B. Data Selection for Cross Project Defect Prediction

In section IV-A we show that learning from open-source projects is theoretically possible when using the right training data. Consequently, the problem is how to select these appropriate cross training data. In this study we proposed a new training data selection method based on data similarity. Our goal is to learn from training data that have the similar distribution with that of the test set. Our proposed method achieves this goal by two operations: instance selection and feature subset selection.

Considering the various development environments and inherent characteristics of different projects, it possible that the historical data extracted from different project are different, i.e., the distribution of the data may be different from each other from a mathematic point-of-view. Thus an intuitive idea for cross project defect prediction is to learn from historical projects that are similar to the target project. The KNN filter introduced by Turhan et al. is a typical way to find out "similar" training data for model training [7]. However, our proposed method do instance selection in a different way, we select training instances on the granularity of data set, i.e., we select or filter out some cross training sets rather than some individual data instances. We argue that the properties of each training/test set are reflections of the properties of the project that generated that data set (or in other words, the "context" of the project). And it is possible to find out "similar" projects to learn from by measuring similarity between data sets. In the proposed data selection method, we adopt a simple and intuitive strategy for instance selection: for each test set, we only learn from the top N closest training sets.

After instance selection, we further employ feature subset selection to filter out features that cause differences between distributions of training data and test data (i.e., unstable features). The intuition is that we should learn on these "common" feature subset in case training data and test data are different. However, we have to point out that feature subset selection is a trade-off between data similarity and training information. The more unstable features we filter out, the more similar the training data and test data will be, while at the same time the more supervised information in the training data for model learning is lost. In our proposed training data selection method, we adopt a data change analysis framework to identify these unstable features.

It is natural that the common feature subset between different training set and test set pairs are different. And according to our experimental setting, we try to learn defect prediction models for proprietary project from multiple open-source projects. So to combine the knowledge of multiple training sets, we adopt the bagging ensemble method to get the final prediction results for the test set. More specifically, we first learn prediction models on each individual training set after feature subset selection, and then apply these prediction models to the test set and use majority voting to combine prediction results.

Fig. 2 gives an overview of our proposed training data selection method for cross project defect prediction. We then describe technical details about measuring similarity between data sets, identifying unstable features, the bagging ensemble strategy, and empirical evaluation of the proposed method, respectively.

```
Input:
    Training sets $D_{train}^{(1)}$, $D_{train}^{(2)}$, … , $D_{train}^{(m)}$;
    Test set $D_{test}$;
    top_N;        //learning from top_N closest training sets
    FSS_ratio;    //the ratio of unstable features filtered out
    score_thre;   //the threshold for bagging.
Output:
    Prediction results for $D_{test}$.
Procedure:
    1) For each training set $D_{train}^{(i)}$ (i = 1, 2, …, m), calculate the
       distance between $D_{train}^{(i)}$ and $D_{test}$;
    2) Select top_N training sets with lowest distance to the test set;
    3) For each selected training set of step 2, remove
       FSS_ratio*100 percent of features that are most unstable;
    4) Learn prediction models on each selected training set (step 2)
       after feature subset selection (step 3), and ensemble prediction
       results from each prediction model by using bagging (i.e,
       majority voting with a threshold score_thre).
```

Figure 2. Overview of the proposed training data selection method

Measuring Data Similarity: The data similarity measureing method we adopted is derived from the data change detection and analysis framework proposed by Hido et al. [15]. Given a training set D_{train} that comprises of M labeled instances $\{(x_{train}^{(1)}, y_{train}^{(1)}), (x_{train}^{(2)}, y_{train}^{(2)}), \ldots(x_{train}^{(M)}, y_{train}^{(M)})\}$, and a test set D_{test} that comprises of N unlabeled instances $\{(x_{test}^{(1)}), (x_{test}^{(2)}), \ldots(x_{test}^{(N)})\}$, the data similarity measuring problem is to calculate the similarity between the marginal distribution of the training set (noted as $P(X_{train})$) and that of the test set (noted as $P(X_{test})$). The key idea of the data similarity measuring method is as follows:

1) Randomly sample K instances from training set D_{train}, noted as SAM_{train}, and K instances from test set D_{test}, noted as SAM_{test}. Then Attach a hypothetical label +1 to each instance of SAM_{train}, and −1 to each instance of SAM_{test}.

2) Combine SAM_{train} and SAM_{test} into a new data set SAM whose size should be $2K$. Train a classifier C on SAM in a supervised fashion and evaluate its accuracy on SAM.

3) Use the accuracy of the classifier C in step 2 as a measurement of distance between D_{train} and D_{test}

Fig. 3 shows an overview of the data similarity measuring method, where △ and □ indicate instances of SAM_{train} and SAM_{test}, respectively. If marginal distributions of D_{train} and D_{test} are actually different, instances of SAM_{train} and SAM_{test} should be correctly classified by the classifier. Thus high classification accuracy indicates a difference between D_{train} and D_{test}. For example, if $P(X_{train}) = P(X_{test})$, the classification accuracy will be about 0.5. However, if the accuracy is significantly larger than 0.5, we can infer that $P(X_{train})$ and $P(X_{test})$ are different with each other.

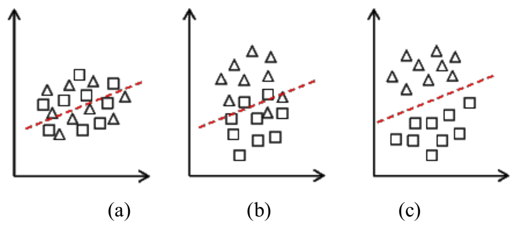


Figure 3. Overview of the data similarity measuring method, these pictures show examples that distribution of two data sets are (a) very close to each other, (b) partly close, and (c) totally different, respectively.

In this study, we employed the Logistic Regression algorithm to build the classifier mentioned in step 2 above, and the parameter K was set to be 500. We evaluate the classifier using 5-folds cross validation. To eliminate the potential bias caused by data sampling operation in step 1, we repeated step 1 and step 2 for 10 times and use the mean value of accuracies get from these 10 rounds as the final accuracy (noted as Acc). Our calculation to convert classification accuracy to data distance is as follow:

$$Distance\ (D_{train}, Dtest) = 2*(Acc - 0.5) \quad (5)$$

Identifying Unstable Features: To identify which part of features cause the differences between D_{train} and D_{test}, we revisit the hypothetical data set SAM generated for measuring data distance. The intuition is that features that play a dominant role in the classification are the ones that characterize the difference. Hido et al suggested identifying unstable features by exploring the structure of classifier C learned from SAM [15]. For example, if C was constructed using the Logistic Regression algorithm, features with high regression coefficients would be more unstable; or if *C* was constructed using the Decision Tree algorithm, features close to the root of the tree can be seen as unstable ones [15].

In this study, we identify unstable features by comparing the Information Gain associated with each feature. More specifically, we calculate information gain of each feature against data set *SAM*, and see features with the most high information gains as unstable features.

Ensemble Prediction Results: The intuition of adopting bagging to merge prediction results produced by each individual defect prediction model is that the stable feature subset selected may be different on different training set, thus we need an ensemble mechanism to combine prediction results derived from heterogeneous training sets. Bagging is essentially a method to eliminate the potential prediction bias caused by each individual predictor by integrating their predictive power [31]. We employed bagging to combine defect prediction results generated by different training sets after feature subset selection. The process of the bagging ensemble method can be described as follows:

Given a set of classifiers $\{C_1, C_2 \dots C_p\}$ and the test set D_{test}, for each instance in D_{test}:

1) Get prediction result of each classifier for that instance, noted as r_i $(i = 1, 2, \dots p)$.

2) Compute the voting score of that instance as:

$$score = \sum_{i=1}^{p} score_i \qquad (6)$$

where $score_i = 1$ if r_i says the prediction result is "defect-prone", otherwise $score_i = 0$.

3) If the voting score is greater than a pre-defined threshold *score_thre*, the final prediction result of that instance will be set as "defect-prone".

V. Evaluation

To evaluate the effectiveness of our proposed training data selection approach, we compare its performance with that of the KNN filter [7]. During the comparisons, we set the parameter *K=10* for KNN filter. Also for our data selection approach, we set *top_N=10* and *score_thre=0.5* in all experiments. As for the parameter *FSS_ratio* for feature subset selection, we try *FSS_ratio* from 0.1 to 0.9 since it is a trade-off between data similarity and training information, and observe the change of prediction performance.

Fig. 4 illustrates the change of prediction performance when using different parameter settings for feature subset selection, where the X-axis indicates how many unstable features are removed. We can see that the Random Forest, Naive Bayes, and Logistic Regression learner can achieve highest performance after removing 60%, 70%, and 80% of features, respectively. An intuitive conclusion is that cross project defect prediction can achieve best performance by only learning from 20% to 40% features. What's more, we can see that the Random Forest learner performs worse than Naïve Bayes and Logistic Regression, while in the within project learning scenario Random Forest performs best. A potential explanation for this observation is that there may be over-fitting happening for Random Forest learner. Our suggestion is to use simple learners such as Naïve Bayes rather than complicated learners such as Random Forest when doing cross project defect prediction.

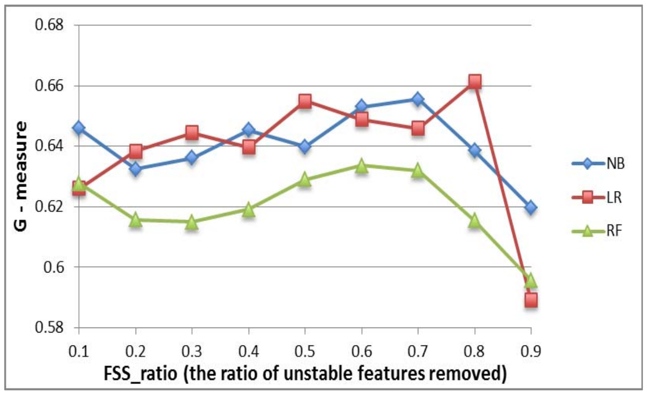


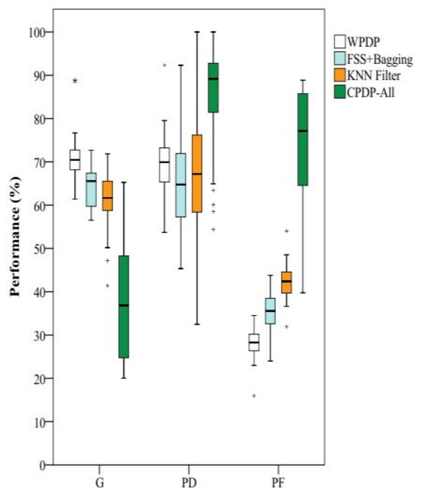
Figure 4. Comparison of G-measure under different parameter setting for feature subset selection.

We then compare the best performance our proposed method with those of the KNN filter and within project learning (WPDP). Fig. 5 illustrates the comparison details. For all three learners we tested, our proposed method produces relative higher *G-measure* values that the KNN filter. For prediction models using Naïve Bayes and Logistic Regression learner, our method produces lower *PD* as well as lower *PF*, and for prediction models using Random Forest learner, our method produces both higher *PD* and *PF*. An empirical conclusion observed in cross project defect prediction using static code metrics is that cross training data may generate prediction results with comparable or even higher *PD* at the cost of a higher *PF* [7]. A key task of cross project defect prediction is to find methods that can reduce the *PF*. Fig. 5 shows that our proposed methods can largely reduce *PF* while it does not cause big damage to *PD* if using Naïve Bayes or Logistic Regression learner. As for the Random Forest learner, again, we do not recommend it for cross project defect prediction because of its bad performance.

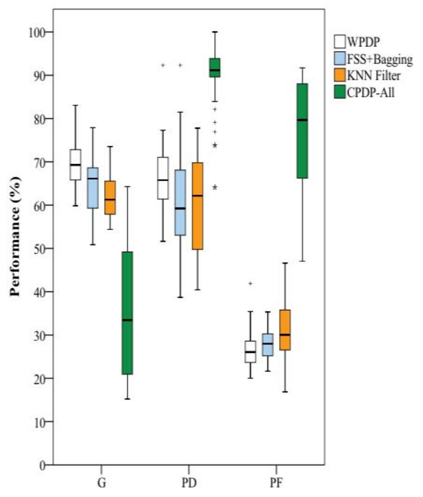
To further support our observation from Fig. 5, we employed Mann-Whitney U test to compare the performance of KNN filter and our method. Table III gives an overview of the significance test results, where "Win" indicates that performance of our method is better than KNN filter, "Loss" means that KNN filter is better, and "Tie" means no significant difference observed. The significance test results show that our proposed method performs better or equally with the KNN filter for most test sets.

TABLE III. Comparison of prediction performance between our proposed method and the KNN filter. This comparison result is based on Significance test results using Mann-Whitney U test at confidence level 0.95.

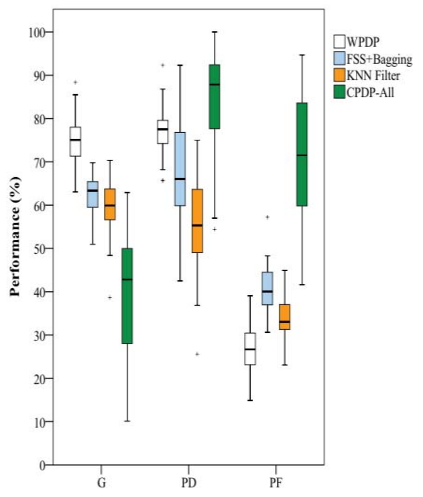
Learner	#Win	#Tie	#Loss
Naïve Bayes	22	8	4
Logistic Regression	15	12	7
Random Forest	17	12	5



(a) Naïve Bayes



(b) Logistic Regression



(c) Random Forest

Figure 5. Comparison of performance between our proposed method and the KNN filter, where FSS+Bagging presents cross project defect prediction using our proposed data selection method, and KNN Filter presents cross project defect prediction using the KNN filter.

Based on comparison results in Table III, we can generate our answer for the second research question:

RQ2: *Does our proposed approach perform better than existing cross-project defect prediction approaches in the literature?*

A2: *Compared with the KNN filter, our proposed method can generate better results for cross project defect prediction with regards to the G-measure. And for two out of the three learners tested, our method can largely reduce the PF without cause big damage to PD.*

Another concern of cross project defect prediction from multiple sources is that whether the learning approach is scalable. Given a test set comprising *N* instances and *F* features, and a set of training set with a total number of *M* instances. The runtime complexity of the KNN filter would be *O(M*N)* because it needs to compute the distance to every training instance for each test instance. In our proposed data selection approach, we convert the data distance measuring problem to be a binary classification problem whose complexity depends on the learning algorithm adopted. The Logistic Regression algorithm we adopted has been proved to be linear complex [32] thus the data similarity step of our method would have a runtime complexity *O(M+N)*. As for the feature subset selection step, we need a total of *O(F*(M+N))* runtime cost to compute the information gain of each feature. The bagging ensemble method we adopted is essentially an linear combination of individual predictors, thus it has linear runtime complexity with regard to the size of training data and test data, i.e., *O(M+N)*. Overall, our proposed method has a *O(F*(M+N))* runtime complexity, which is close to the linear complexity since for most defect prediction data sets the feature numbers are not high. Based on analysis above, we argue that our proposed data selection method is more scalable than the KNN filter. The runtime complexity of our method is approximately linear while that of the KNN filter is quadratic.

To further verify the scalability of our proposed method, we compared its runtime with that of the KNN filter. Fig. 6 illustrates comparison of the runtime for all 34 test sets. We observed that our proposed method can significantly reduce the computation cost when compared with the KNN filter. For example, for predictions using the Logistic Regression learner, the KNN filter needs 559 seconds to run while our method only needs 132 seconds.

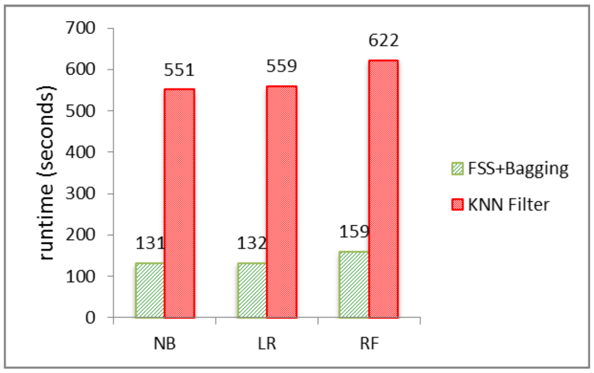


Figure 6. Comparison of runtime between our method and the KNN filter.

Based on analysis above, we can answer the third research question as follows:

RQ3: *Is our proposed approach more scalable than existing approaches?*

A3: *Comparison results of runtime show that our proposed method for cross project defect prediction is more scalable that the KNN filter.*

VI. DISCUSSION

A. Necessity of Cross Project Defect Prediction

Some studies argue that learning from other projects is a serious problem while some other argue that the performance of cross project defect prediction is comparable with that of within project defect prediction [5]-[7]. Our findings in this study are consist with what Turhan et al. found and support the conclusion that learning from other project without data filtering will cause very high *PF* [7]. Also, considering the expensive cost of project measurement and data maintains program [33] and the fact that local historical data are not available sometimes (e.g., the first release of a new project), we argue that cross project defect prediction is necessary.

Additionally, data filtering is needed because the characteristics of training data and the test data may be significantly different. For example, for the open-source projects and proprietary projects we investigated, there is a very intuitive difference: the ratios of defect-prone classes of proprietary projects are much lower than that of open-source projects.

B. Scalability of Defect Prediction Methods

By far, there are some attempts to solve the cross project defect prediction problem. One drawback of these existing methods is their computation cost. According to our experimental setting, managers of proprietary projects can learn from a large number of potential training sets, i.e., open-source projects, which will make the scalability problem more serious. Different from the KNN filter, the overall complexity of our proposed data selection method is almost linear, and the instance selection and feature subset selection operations largely reduce the size of training data (e.g., we only need to learn from 20%-40% features). We argue that scalability is a very crucial property that may decide the success of cross project defections in practice, and we should pay more attention to it.

C. Low Dimentionality of Defect Prediction Data

Another interesting observation of our experiments is that the low dimensionality phenomenon of cross project defect predictions. Fig. 4 shows we only need to learn from 20% - 40% features. Note that learning from fewer features can benefit both scalability of prediction models and the privacy and security of projects that generated the training data. The low dimensionality nature of defect prediction data was first systematically described by Menzies et al. [2], however, this study is the first one that reports the low dimensionality phenomenon of defect prediction in the cross project context.

VII. THREATS TO VALIDITY

The external threats concern the generality of this study. First, our experiment data consist of 68 data sets collected from 10 open-source projects and seven proprietary projects, which is considerably large compared with related studies in the defect prediction research field. We can not assure that these projects can represent all projects in industry. However, the various sources of data relatively reduce this risk. Projects examined in this study cover a large scope of applications including text/xml processing systems, search engines, source code integration/build tools, and management information systems et al. Second, all projects investigated in this study are developed in Java, and we are not sure that similar results can be observed on projects in other language.

Regarding internal threats, there are two aspects. First, the data similarity measuring method we adopted is an empirical one, and different settings (e.g., learner, sample size) may lead to different similarity measuring outputs. There are strictly defined data similarity measures such as MMD in the machine learning field [14]. However, considering their expensive computation cost, we prefer the empirical ways adopted in this study. Also, our feature subset selection method treats each feature independently, i.e., assess the stability of each feature by its information gain. It is possible that the interaction among features may make our stability assessment method biased. In our proposed data selection method, we learn from top *N* closest training sets and ensemble their prediction results using majority voting, which may help to reduce the bias caused by data similarity measuring.

Construct validity of this study mainly questions the model performance assessment approach and indicators we adopted. As mentioned in many other related studies, the conclusion of defect prediction depends on the performance assessment indicators [5], [30]. In this study we adopted *G-measure* as well as *PD* and *PF* to presents the prediction performance just as some widely cited studies did [2], [3].

VIII. CONCLUSION

The majority of cross defect prediction studies base their conclusions on experiments done using solely open-source projects. In other words they use them both as test and training data. Also, most existing cross project defect prediction methods are facing the problem of scalability. The problem here is, can we generalize these results to proprietary project in a scalable way? To answer this question, in this work we train on open-source data and test on proprietary data, and propose a new data selection method for cross project defect prediction. Empirical studies on large-scale data show that:

- Open-source project data can be used to build defect prediction models for proprietary projects despite the differences in data characteristics. Our experiment confirms the possibility of learning quality defect prediction models from open-source projects.
- Our proposed training data selection method can provide relatively better prediction results than the KNN filter at a lower computation cost. We employ data similarity measuring and feature subset selection to remove irrelevant and unstable parts of the training data. Our results demonstrate the effectiveness and scalability of the proposed method.

Our future work includes:

- Testing more data similarity measures to examine their effectiveness for cross project defect prediction
- More investigations on the size of essential training data, e.g., how many cross project training sets is enough to build quality defect prediction model.
- More investigations on the low dimensionality phenomenon of cross project defect prediction.

ACKNOWLEDGMENT

This research was supported in part by National Natural Science Foundation of China under Grant Nos. 61073044, 71101138 and 61003028; National Science and Technology Major Project under Grant Nos. 2012ZX01039-004; Beijing Natural Science Fund under Grant No.4122087. Also by NSF grant CCF: 1017330 and the Qatar/West Virginia University grant NPRP 09-12-5-2-470.

REFERENCES

[1] T. Ostrand, E. Weyuker, and R. Bell, “Predicting the location and number of faults in large software systems,” IEEE Transaction on Software Engineering, vol. 31, no. 4, pp. 340-355, 2005.

[2] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Transaction on Software Engineering, vol. 33, no. 1, pp. 2-13, 2007.

[3] T. Menzies, Z. Milton, and B. Turhan et al., “Defect prediction from static code features: current results, limitations, new approaches,” Automated Software Engineering, vol. 17, no. 4, pp. 375-407, 2010.

[4] S. Lessmann, B. Baesens, and C. Mues et al., “Benchmarking classification models for software defect prediction: a proposed framework and novel findings,” IEEE Transaction on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.

[5] F. Rahman, D. Posnett, and P. Devanbu, “Recalling the “Imprecision” of Cross-Project Defect Prediction,” in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 2012.

[6] T. Zimmermann, N. Nagappan, and H. Gall, “Cross-project defect prediction: a large scale experiment on data vs. domain vs. process,” in Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 91-100, 2009.

[7] B. Turhan, T. Menzies, and A. 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.

[8] Z. He, F. Shu, and Y. Yang et al., “An investigation on the feasibility of cross-project defect prediction,” Automated Software Engineering, vol. 19, pp. 167-199, 2012.

[9] 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.

[10] S. Watanabe, H. Kaiya, and K. Kaijiri, “Adapting a fault prediction model to allow inter language reuse,” in Proceedings of the 4th International Workshop on Predictive Models in Software Engineering, pp. 19-24 2008.

[11] F. Peters, T. Menzies, and L. Gong et al., “Balancing Privacy and Utility in Cross-Company Defect Prediction,” IEEE Transaction on Software Engineering, 2013. (in press)

[12] E. Kocaguneli and T. Menzies, “How to find relevant data for effort estimation?” in Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement, pp. 255-264, 2011.

[13] J. G. Moreno-Torres, T. Raeder, and R. Alaiz-Rodríguez et al., “A unifying view on dataset shift in classification,” Pattern Recognition, vol. 45, no. 1, pp. 521-530, 2012.

[14] A. Gretton, K. Borgwardt, and M. Rasch et al., “A Kernel Method for the Two-Sample Problem,” Advances in Neural Information Processing Systems 19, MIT Press, Cambridge, MA, pp.513-520, 2007.

[15] S. Hido, T. Ide, and H. Kashima et al., “Unsupervised Change Analysis Using Supervised Learning,” in Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data mining, pp. 148-159, 2008.

[16] A. Dries and U. Rückert, “Adaptive concept drift detection,” Statistical Analy Data Mining, vol. 2, no. 5-6, pp. 311-327, 2009.

[17] X. Song, M. Wu, and C. Jermaine et al. “Statistical Change Detection for Multi-Dimensional Data,” in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 667-676, 2007.

[18] N. Tatti, “Distances between Data Sets Based on Summary Statistics,” The Journal of Machine Learning Research, vol. 8, pp. 131-154, 2007.

[19] 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.

[20] J. Huang, A. Smola, and A. Gretton et al., “Correcting Sample Selection Bias by Unlabeled Data,” in Proceedings of the 19th International Conference on Neural Information Processing Systems, 2007.

[21] J. Blitzer, R. McDonald, and F. Pereira, “Domain Adaptation with Structural Correspondence Learning,” in Proceedings of the Conference on Empirical Methods in Natural Language, pp. 120-128, 2006.

[22] S. J. Pan, J.T. Kwok, and Q. Yang, “Transfer Learning via Dimensionality Reduction,” in Proceedings of the 23rd Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence, pp. 677-682, 2008.

[23] S. J. Pan, I. W. Tsang, and J. T. Kwok et al., “Domain Adaptation via Transfer Component Analysis,” IEEE Transaction on Neural Networks, vol. 22, no. 2, pp. 199-210, 2011.

[24] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, 2010.

[25] M. Jureczko, “Significance of different software metrics in defect prediction,” Software Engineering: An International Journal (SEIJ), vol. 1, no. 1, pp: 86-95, 2011.

[26] L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[27] D. Lewis, “Naive (bayes) at forty: The independence assumption in information retrieval,” in Proceedings of the 10th European Conference on Machine Learning in Machine Learning, pp. 4–15, 1998.

[28] W. Afzal, “Using faults-slip-through metric as a predictor of fault-proneness,” in Proceedings of the 2010 Asia Pacific Software Engineer- ing Conference, pp. 414-422, 2010.

[29] T. Menzies, B. Turhan, and A. Bener et al., “Implications of ceiling effects in defect predictors,” in Proceedings of the 4th international workshop on Predictive Models in Software Engineering, pp. 47-54, 2008.

[30] Y. Jiang, B. Cukic, and Y. Ma, “Techniques for evaluating fault prediction models,” Empirical Software Engineering, vol. 13, pp. 561-595, 2008.

[31] L. Breiman, “Bagging Predictors,” Machine Learning, vol. 24, no. 2, pp. 123-140, 1996.

[32] A. Y. Ng and M. I. Jordan, “On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes,” In Neural Information Processing Systems 14, 2002.

[33] N. E. Fenton and M. Neil, “Software metrics: successes, failures and new directions,” Journal of Systems and Software, vol. 47, no. 2-3, pp.149 – 157, 1999.