Int. j. inf. tecnol. (February 2022) 14(1):587–601
https://doi.org/10.1007/s41870-019-00299-6

ORIGINAL RESEARCH

Cross project defect prediction for open source software

Anushree Agrawal[1] · **Ruchika Malhotra**[1]

Received: 15 September 2017 / Accepted: 28 March 2019 / Published online: 6 April 2019

Abstract Software defect prediction is the process of identification of defects early in the life cycle so as to optimize the testing resources and reduce maintenance efforts. Defect prediction works well if sufficient amount of data is available to train the prediction model. However, not always this is the case. For example, when the software is the first release or the company has not maintained significant data. In such cases, cross project defect prediction may identify the defective classes. In this work, we have studied the feasibility of cross project defect prediction and empirically validated the same. We conducted our experiments on 12 open source datasets. The prediction model is built using 12 software metrics. After studying the various train test combinations, we found that cross project defect prediction was feasible in 35 out of 132 cases. The success of prediction is determined via precision, recall and AUC of the prediction model. We have also analyzed 14 descriptive characteristics to construct the decision tree. The decision tree learnt from this data has 15 rules which describe the feasibility of successful cross project defect prediction.

Keywords Cross project · Defect prediction · Software characteristics

✉ Anushree Agrawal
Anushreeagrawal.iet@gmail.com

Ruchika Malhotra
Ruchikamalhotra2004@gmail.com

[1] Department of Computer Engineering, Delhi Technological University, Delhi, India

1 Introduction

Defect prediction in software systems focuses on prediction of fault prone classes early in the software development life cycle. This helps in near to optimal allocation of testing and maintenance resources. Defect prediction works well if large amount of data is available to train the prediction model. However, if the data is not preserved or if we are dealing with the first release of the software system, no training data is available. Thus defect prediction based on historical data of same project is not always feasible.

Cross project defect prediction is the process of predicting defects in software systems using historical data of other projects [1]. Very few Studies are available in literature for cross project defect prediction and they show that this is a serious challenging task. In our work, we have attempted to study the feasibility of cross project defect prediction using open source software systems. The prediction model is build using logistic regression. In this work we have empirically investigated the following severe research paradigms.

1.1 Defect data of one project is likely to derive defects of another project

Various studies in the literature show that historical data from software repositories can be used in prediction of software defects for upcoming releases, but availability of this past defect data is not always possible. In this study, we have empirically validated that defect data from other projects can be used to identify the defective classes.

1.2 The potential defect predictors in cross project defect prediction

Cross project defect prediction is feasible in some cases, but this is not always possible. The major challenge in this field is how to identify the scenarios where cross project defect prediction is applicable. One solution to this problem given by Zimmerman [1] in his work is to study the relationship between the characteristics of the training and test set. In this work, we have studied 14 characteristics of software projects and illustrated this relationship with the help of decision tree where the characteristics determine the potential predictors.

1.3 The criteria for successful cross project defect prediction?

There are numerous studies in literature in the area of software defect prediction. The prediction model is build using statistical or machine learning methods and the efficiency of the model is evaluated using various measures as sensitivity, specificity, precision, recall, AUC etc. analyzing the various studies, we have built the model using logistic regression and chosen appropriate cut off values of precision, recall and AUC to accept or reject the model.

Rest of the paper is organized as follows: Sect. 2 is the related work in the context of cross project defect prediction. Numerous studies are present in literature for defect prediction model trained from previous release of the same project, but very few cross project studies have been done in literature. Section 3 explains the Research Methodology that is used for our experiment. Following this Sect. 4 is Result Analysis section that describes the cross project prediction results. We have shown the acceptance criteria of each cross project model and also the decision tree, which is learnt from data. In the last section, we provide the conclusion and related future work scopes on this project, which could be taken as a subject to be worked upon. We have also discussed the threats to validity in this section.

2 Related work

Numerous studies are available in literature in the area of software defect prediction. The aim of most of them is to study the feasibility of defect prediction from the historical data of same project. Prediction models are built using the statistical and machine learning methods. Radjenovic et al. have done a systematic literature review for software fault prediction models in their work [3]. In this work, the authors have searched seven digital libraries to identify the most commonly used set of software metrics in software fault proneness prediction. Gray and MacDonell have also compared the various techniques for software fault prediction models [4]. The authors have discussed the inherent limitations of the techniques used in defect prediction models. Careful attribute selection is very important for the success of a fault prediction model. The authors have investigated the impact of attribute selection on naïve bayes based fault prediction model in their work [5].

Very few studies are available in the area of cross project defect prediction. Turhan et al. have investigated the application of cross company defect data to build prediction model using static code features [9]. They have conducted their experiments on seven NASA and three SOFTLAB datasets. Zimmermann et al. have studied the feasibility of cross project defect prediction and validated it using several versions of open source software. They have conducted their study on apache tomcat, apache Derby, Eclipse, Firefox, Direct-X, IIS, Printing, Windows Clustering, Windows File system, SQL Server 2005 and Windows Kernel [1]. The results indicate that the relationship of characteristics between the projects permits cross project defect prediction in some cases. This relationship is analyzed with the help of decision trees.

He et al. have also empirically validated cross project defect prediction using defect data from PROMISE repository [7]. They have conducted the experiment on 34 releases of 10 open source projects. Ma et al. have proposed a novel learning algorithm 'Transfer Naïve Bayes' for cross company defect prediction [8]. They have exploited all the cross company data in training the model. The results are validated on NASA datasets and Turkish local software datasets. Gerardo et al. proposed the use of genetic algorithm to build a multi objective cross project defect prediction model [10]. They used public dataset from PROMISE repository to validate and produce a compromise between precision and recall for cross-project defect prediction.

S. Herbold proposed distance-based strategies for training data selection based on distributional characteristics [11]. They evaluated their work with 44 data sets obtained from 14 open source projects. The results indicate that this training data selection strategy improved the success rate of cross-project defect prediction. They also proposed a tool CrossPare to provide standards for cross project defect prediction [15]. The tool implemented few techniques proposed for cross-project defect predictions. CrossPare can be used for improving the assessment of results in cross project defect prediction studies.

Ryu et al. proposed a transfer learning based model to deal with the class imbalance problem which may decrease the prediction accuracy in cross project defect prediction studies [12]. They computed similarity weights of the training data based on the test data and applied it to Boosting algorithm considering the class imbalance. The

results are validated using NASA and SOFTLAB datasets. Ryu et al. also propose a multi objective naïve Bayes algorithm with Harmony search meta-heuristic algorithm [20]. The results indicate that the proposed approach shows similar prediction performance but better diversity compared to existing multi objective CPDP algorithms.

Panichella et al. conducted an empirical study on 10 open source software systems to analyze the similarity of different defect prediction models [13]. They proposed a combined approach that used the classification provided by different machine learning methods to improve the defect prediction results. They found that better prediction accuracy was achieved using the combined approach. Amasaki et al. conducted a study to identify the effects of the data simplification for CPDP methods [14]. They compared the predictive performance with and without applying data simplification on CPDP methods. They found that applying data simplification achieved improved results for cross-project selection. Satin et al. studied the combination of different classification algorithms for feature selection and data clustering [16]. They applied it to 1270 projects and built different cross-project prediction models. The authors reported that Naive Bayes algorithm obtained the best performance, with 31.58% of adequate predictions in 19 models. Zhang et al. investigated 7 algorithms integrating multiple machine learning classifiers to improve prediction results in cross project studies [17]. They performed experiments using 10 open source software systems from the PROMISE repository. They compared their results with CODEP [13] and found better results in terms of F-measure. Zhang et al. also compared the performance of unsupervised and supervised classifiers for cross project defect prediction using AEEEM, NASA and PROMISE datasets [21]. They propose connectivity-based classifiers as the potential solution for cross project defect prediction studies. The authors also investigated the effect of Log, Box-Cox and rank transformations in cross project defect prediction [23]. They found that all of these are comparable in terms of performance measures however these models do not exhibit same behavior on single entities. Peters et al. proposed a private multi-party sharing method for cross-project defect prediction [18]. Xia et al. propose a two-phase technique for cross defect prediction i.e. genetic algorithm phase and ensemble learning phase [19]. They performed experiments with 29 datasets from PROMISE repository and reported improved results when compared to literature. Hosseini et al. proposed Genetic Instance Selection (GIS) that optimizes combined measure of F-Measure and G-Mean [22]. They used 13 datasets from PROMISE repository for their experiments and concluded that search based instance selection is a promising solution for cross project defect prediction. Wu et al. proposed a semi-supervised dictionary learning technique for software defect prediction [24]. They used the labeled defect data and unlabeled data and performed their experiments using two public datasets. They found that the proposed technique was useful in identification of software defects. Poon et al. proposed a credibility theory based naïve bayes classifier based on reweighing mechanism [25]. Thus the source data adapts to the target distribution of data and preserves its patter as well. The results are promising and show significant improvement in prediction rate. Huang et al. proposed a three-stage algorithm for cross project defect prediction. They used the nearest neighbor algorithm for similarity identification and then applied the Bayes classifier [26]. Jing et al. proposed combination of improved Subclass discriminant analysis (ISDA) and semi supervised transfer component analysis as a solution for cross project defect prediction [27]. Goel et al. have conducted a systematic literature review on cross project defect prediction [28]. They found that the best practices for cross project defect prediction could not be established and more research needs to be carried out in heterogeneous CPDP to improve the prediction results.

From this study we observed that cross project defect prediction is feasible with careful selection of code quality features. The relationship among the various characteristics of the datasets should be carefully analyzed to choose the potential defect predictors. We have attempted to extend this study by empirical validation of cross project defect data using defect data of twelve open source software and twelve OO metrics [6]. The prediction model is build using logistic regression.

3 Research methodology

3.1 Data collection

We have analyzed the logs of latest version for software to identify the faulty classes. We have developed a tool, configuration management system (CMS) in java language to fetch these logs [2]. CMS offers features to analyze the changes amongst two versions of software as well as fetch logs from software project repositories and process them to obtain bug count. In this study we have used CMS to obtain faulty classes only. Figure 1 explains the data collection method of CMS.

3.1.1 Source code checkout

The first step in data collection process is to obtain the source code from the remote repository. For this, we create a local copy of the software. We connect to the CVS repository of the software by logging in into the system and

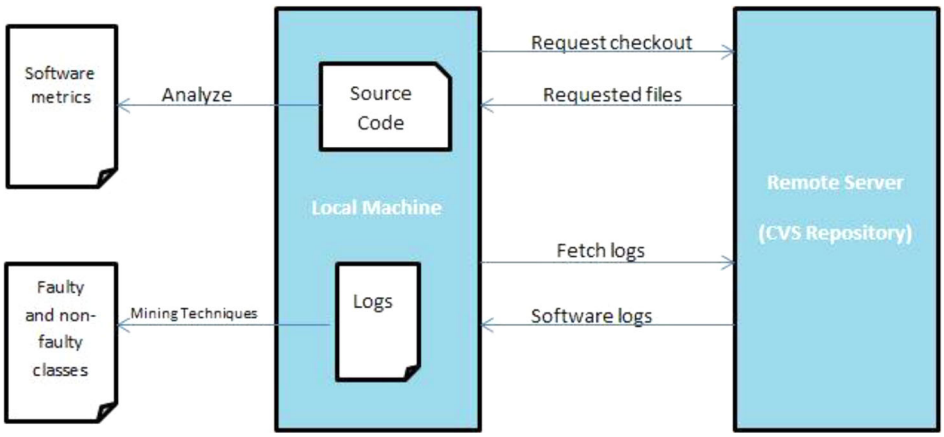

Fig. 1 Data collection methodology

then download the source code on our local machine. This is done with the help of CVS "checkout" command.

3.1.2 Extraction of bugs

After we make a local repository for the code, we can request the logs using "log" command. The server replies with the software logs in response to this command, which is a huge file. We apply text mining on this file and search for text pattern "bug" and "fix" in the logs. If any of the two keywords is found, the class is assumed to be faulty. We repeat the process for each file in the source code to identify all the faulty classes in the software.

3.1.3 Metrics calculation

We obtain the metrics for software with the help of "Understand" tool. This tool calculates the object oriented metrics for each class. We have calculated seven object oriented metrics for software.

3.1.4 Preparation of dataset

We integrate the metric and bug report to obtain the dataset. Preprocessing is done to remove the unnecessary data points. Now we apply logistic regression on the collected data to build the prediction model.

We have used 12 software systems for our study obtained from sourceforge.net. These datasets vary in domain of application, size and percentage of faulty classes, while the programming language of all datasets is Java. Table 1 lists the programming language, version used for our experiments, and the count and percentage of faulty classes for all software under study.

Amakihi: Amakihi supports the software testing activity of SDLC by helping the software developers in automation of test scripts. It consists of 98 classes where 44 are faulty [29].

Amber archer: Amber archer is a java class library to support corporate software development process. It consists of 693 java files with 9.7% of faulty classes [30].

Abbot: ABBOT is a java framework that is used to test UI for java applications. It consists of 330 java classes out of which 46.1% classes are faulty [31].

Apollo: It provides an editor and a compiler for data migration purpose for software systems. It consists of 292 java classes out of which 58 are faulty [32].

Avisync: Avisync is a utility developed in java language which is used to fix synchronization problems in audio/video while playing AVI files. It is also small software with 67 classes with 37.3% of faulty classes having one or more faults [33].

Jfreechart: Jfreechart is a chart library that can be used with java programs. It is developed in Java and we have used the version 1.0.0. It consists of 689 classes out of which 59.2% classes contain one or more faults [34].

Jgap: Jgap is a genetic programming component available as a Java framework. It is developed in java language. It consists of 173 classes out of which 35.3% (61) classes are having one or more than one faults [35].

Jtreeview: Jtreeview is a cross platform visualization tool which is used for visualization of gene expression data. It is developed in java language. We have studied version 1.0.0 of this software for our study and 184 out of 405 classes (45.4%) are found faulty [36].

Barcode4j: Barcode4j is available under the Apache license v2.0. It is a flexible generator of barcodes. We have used version 1.0 of this software for our experiments which consists of 170 classes out of which 31 classes had one or more than one faults [37].

Jtopen: It is a set of lightweight classes appropriate to be used on mobile devices. We have used v1.0 of this software for our study, which consists of 1527 classes out of which 27.9% classes are faulty [38].

Jung: JUNG provides a common and extendible language for the modeling, analysis, and visualization of data

Table 1 Software systems used for experiment

Dataset	Programming language	Version	No. of faulty classes	No. of non-faulty classes	Total no of classes	Percentage of faulty classes
Amakihi	Java	1.0alpha1	44	54	98	44.9
Amber archer	Java	1.1	67	626	693	9.7
Abbot	Java	1.0.0rc1	152	178	330	46.1
Apollo	Java	0.1	58	234	292	19.9
Avisync	Java	1.0	25	42	67	37.3
Jfreechart	Java	1.0.0	408	281	689	59.2
Jgap	Java	3.4.4	61	112	173	35.3
Jtreeview	Java	1.0.0	184	221	405	45.4
Barcode4j	Java	1.0	31	139	170	18.2
Jtopen	Java	1.0	426	1101	1527	27.9
Jung	Java	1.3	51	98	149	34.2
Geotag	Java	0.07	89	539	628	14.2

that can be represented as a graph or network. We have performed our experiments on JUNGv1.3, which consists of 51 faulty classes out of 149 [39].

Geotag: It is a portable; GUI based intelligent matching software system. We have used v 0.07 of this software for our study, which consists of 628 classes, 89 of which are faulty [40].

3.2 Prediction model

The prediction model is build using the logistic regression technique. Logistic regression is a type of probabilistic statistical classification model, which is used to predict a binary response from a binary predictor based on one or more predictor variables. It measures the relationship between the independent variable and the categorical independent variable. We have studied various object oriented software metrics and selected 12 of them to build our prediction model. Table 2 lists these software metrics. These metrics are the independent variables to construct the prediction variable and the binary dependent variable is fault proneness.

3.3 Descriptive statistics

We have calculated 14 indicators to describe the distribution of each metric in a training/test set. These indicators and their description are listed in Table 3. We combine these indicators with the metrics to make a set of (14 indicators $\times$ 12 metrics) 168 metric indicators. These 168 indicators describe the distributional characteristics of the training and test sets under study. We have listed these characteristics in Tables 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 for all the datasets.

3.4 Performance evaluation measures

3.4.1 Precision

Precision is the ratio of number of classes that are correctly classified as faulty and the no. of classes that are classified as faulty.

3.4.2 Recall

Recall is defined as the ratio of the number of classes that are correctly classified to the total no. of faulty classes.

3.4.3 Area under receiver operating characteristics (ROC) curve

ROC curve is a plot in between the true positives out of the total actual positives vs. the false positives out of the total actual negatives. Hence, ROC curve is a graphical plot between sensitivity and 1: specificity at varied discriminating thresholds. We define sensitivity and specificity as.

Sensitivity: Sensitivity or true positive rate is the fraction of true positives and total actual positives.

Specificity: It is the false positive rate or the fraction of false positives and total actual negatives subtracted from 1.

3.5 Construction of decision tree

Although cross project defect prediction works in several cases, but successful defect prediction is not feasible in all

Table 2 Metrics description

Metrics studied	Description
AVG_CC	There are a large number of functions or program modules in a project. The average of all such cyclomatic complexities is known as average cyclomatic complexity
CBO	CBO is measured only for object oriented systems and it is defined as the number of other classes that a class is coupled to
NOC	It is the count of number of immediate subclasses that inherit the class. This gives an idea about the influence of the class on software design
NIM	This is the count of total number of methods defined in a class that are only accessible through an object of that class
NIV	This is the count of total number of variables defined in a class that are only accessible through an object of that class
RFC	The response set (RS) of a class is a set of methods that can potentially be executed in response to a message received by an object of that class
NPM	It is the count of total public methods in a class
LOC	The total number of executable lines of code excluding blank lines and comments
MAX_CC	It is the maximum cyclomatic complexity possessed by any function or program in the entire software. This gives the information about most complex part of the project
DIT	DIT is the path length from root node to the farthest leaf node of the inheritance tree. The higher value of DIT denotes a greater number of classes that it inherits, making it complex to predict the class behavior
LCOM	It is the difference between method not having common attribute usage and methods having common attribute usage
WMC	WMC is defined as the weighted sum of the complexities of all the methods defined in a class

Table 3 Indicators of software attributes

Indicator	Description
Mean	The average value of the data points. It is given as $\mu = \sum_{i=1}^{n} \frac{x_i}{n}$
Median (Med)	The middle value in the sorted dataset
Mode (Mod)	The value with maximum occurrence in the dataset
Std. deviation (SD)	It measures the distance of data points from the mean. It is given as $\sqrt{\frac{\sum (x-\mu)^2}{n}}$
Variance (V)	It is a measure of variability and computed by squaring the std. deviation
Skewness (S)	It is the measure of asymmetry in the dataset
Kurtosis (K)	It is the measure of peakedness in the dataset
Minimum (min)	The minimum value among all the data points
Maximum (max)	The maximum value among all the data points
Range (R)	The numeric difference between the minimum and maximum
First quartile (1Q)	The first quartile is obtained by computing the median of the dataset and then re-computing the median of the lower half
Third quartile (3Q)	The third quartile is obtained by computing the median of the dataset and then re-computing the median of the upper half
Interquartile range (IQR)	The difference between the third quartile and the first quartile
Coff. Of variation (CoV)	It is given as the ratio of std. deviation to the arithmetic mean

cases. After studying the various combinations of training and testing datasets, we have constructed a decision tree to validate the relationship between feasibility of cross project defect prediction and distributional characteristics of training and testing datasets.

We have conducted our experiments on all possible permutations of the datasets. One set is chosen as training set, which is used to build the prediction model, and the remaining 11 sets are test sets. They are chosen one by one to evaluate the model. This process is repeated y choosing all the datasets as training sets one at a time. Thus we get

Table 4 Descriptive statistics of Amakihi dataset

Amakihi	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.7	1.0	1.0	1.7	2.9	3.5	15.7	11.0	1.0	12.0	1.0	2.0	1.0	0.96
CBO	2.0	1.0	0.0	2.9	8.8	3.4	15.6	18.0	0.0	18.0	0.0	3.0	3.0	1.45
NOC	0.3	0.0	0.0	0.8	0.7	3.2	11.5	5.0	0.0	5.0	0.0	0.0	0.0	2.88
NIM	6.1	4.0	1.0	7.3	53.7	3.8	23.3	57.0	0.0	57.0	1.0	9.0	8.0	1.20
NIV	1.4	1.0	0.0	2.3	5.3	2.3	5.3	11.0	0.0	11.0	0.0	2.0	2.0	1.58
RFC	9.2	8.0	4.0	7.9	62.7	2.9	16.3	59.0	1.0	60.0	4.0	13.0	9.0	0.86
NPM	0.2	0.0	0.0	0.5	0.3	2.8	8.5	3.0	0.0	3.0	0.0	0.0	0.0	2.62
LOC	82.2	44.5	5.0	147.1	21,638.0	5.8	44.0	1263.0	5.0	1268.0	17.0	83.0	66.0	1.79
MAX_CC	3.8	2.0	1.0	4.2	18.1	2.4	7.7	24.0	1.0	25.0	1.0	5.0	4.0	1.12
DIT	1.7	2.0	2.0	0.6	0.4	0.8	1.0	3.0	1.0	4.0	1.0	2.0	1.0	0.40
LCOM	51.0	64.0	0.0	35.5	1262.1	− 0.5	− 1.3	100.0	0.0	100.0	0.0	80.0	80.0	0.70
WMC	13.8	9.0	4.0	15.2	233.4	2.1	5.1	73.0	1.0	74.0	4.0	18.0	14.0	1.11

Table 5 Descriptive statistics of Amberarcher dataset

Amberarcher	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.2	1.0	1.0	0.8	0.6	3.0	13.8	7.0	0.0	7.0	1.0	1.0	0.0	0.6
CBO	2.6	2.0	0.0	3.2	10.2	2.1	5.4	20.0	0.0	20.0	0.0	4.0	4.0	1.2
NOC	0.5	0.0	0.0	2.0	4.1	9.3	109.9	30.0	0.0	30.0	0.0	0.0	0.0	4.5
NIM	0.3	0.0	0.0	1.3	1.6	14.4	288.9	27.0	0.0	27.0	0.0	0.0	0.0	4.1
NIV	0.4	0.0	0.0	1.4	1.9	5.9	41.6	14.0	0.0	14.0	0.0	0.0	0.0	3.5
RFC	10.3	8.0	2.0	10.9	118.4	2.8	10.5	76.0	0.0	76.0	3.0	13.0	10.0	1.1
NPM	0.6	0.0	0.0	1.5	2.4	5.0	34.7	16.0	0.0	16.0	0.0	0.0	0.0	2.7
LOC	42.3	23.0	11.0	50.8	2584.2	2.8	11.0	401.0	1.0	402.0	11.0	53.0	42.0	1.2
MAX_CC	2.4	1.0	1.0	2.4	5.9	2.9	11.4	20.0	0.0	20.0	1.0	3.0	2.0	1.0
DIT	1.9	2.0	1.0	1.0	0.9	0.7	− 0.5	4.0	1.0	5.0	1.0	3.0	2.0	0.5
LCOM	30.8	16.0	0.0	33.8	1140.4	0.4	− 1.4	100.0	0.0	100.0	0.0	64.0	64.0	1.1
WMC	8.4	5.0	2.0	9.9	98.7	2.8	10.9	83.0	0.0	83.0	2.0	10.0	8.0	1.2

Table 6 Descriptive statistics of Abbot dataset

Abbot	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.8	1.0	1.0	1.4	2.0	2.8	11.0	10.0	0.0	10.0	1.0	2.0	1.0	0.8
CBO	3.2	1.0	1.0	7.2	51.2	11.4	169.2	113.0	0.0	113.0	1.0	4.0	3.0	2.2
NOC	0.6	0.0	0.0	2.8	7.7	12.5	185.1	44.0	0.0	44.0	0.0	0.0	0.0	5.0
NIM	6.3	2.0	2.0	11.1	123.8	5.4	38.4	112.0	0.0	112.0	2.0	7.0	5.0	1.8
NIV	1.8	0.0	0.0	4.3	18.8	6.9	64.4	52.0	0.0	52.0	0.0	2.0	2.0	2.5
RFC	34.7	6.0	5.0	59.5	3544.1	2.2	3.5	221.0	0.0	221.0	3.0	35.3	32.3	1.7
NPM	1.1	0.0	0.0	5.2	27.3	13.2	204.6	85.0	0.0	85.0	0.0	1.0	1.0	5.0
LOC	82.5	25.5	5.0	203.8	41,543.9	7.9	85.0	2655.0	1.0	2656.0	9.0	83.0	74.0	2.5
MAX_CC	4.0	2.0	1.0	4.6	21.2	3.3	15.6	37.0	0.0	37.0	1.0	5.0	4.0	1.2
DIT	2.5	2.0	2.0	1.3	1.7	0.8	0.0	5.0	1.0	6.0	1.0	3.0	2.0	0.5
LCOM	30.4	0.0	0.0	36.1	1305.9	0.6	− 1.4	98.0	0.0	98.0	0.0	66.0	66.0	1.2
WMC	16.6	6.0	2.0	38.1	1453.5	6.3	47.9	369.0	0.0	369.0	2.0	15.3	13.3	2.3

Table 7 Descriptive statistics of Apollo dataset

Apollo	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.7	1.0	1.0	1.7	2.7	3.9	18.8	13.0	0.0	13.0	1.0	2.0	1.0	1.0
CBO	4.9	4.0	0.0	5.0	25.2	2.1	7.1	35.0	0.0	35.0	1.0	8.0	7.0	1.0
NOC	0.6	0.0	0.0	3.4	11.4	9.6	107.8	44.0	0.0	44.0	0.0	0.0	0.0	5.6
NIM	6.4	5.0	5.0	7.4	55.3	5.0	40.2	82.0	0.0	82.0	3.0	8.0	5.0	1.2
NIV	2.8	2.0	0.0	3.5	12.1	2.8	13.6	30.0	0.0	30.0	0.0	4.0	4.0	1.3
RFC	13.2	9.0	1a	11.9	141.0	1.6	4.1	83.0	0.0	83.0	4.0	20.5	16.5	0.9
NPM	0.2	0.0	0.0	0.8	0.6	5.1	30.3	7.0	0.0	7.0	0.0	0.0	0.0	3.8
LOC	69.5	39.5	17.0	97.1	9421.1	4.8	36.3	1022.0	2.0	1024.0	17.3	82.8	65.5	1.4
MAX_CC	4.3	3.0	1.0	5.4	28.6	4.0	22.1	48.0	0.0	48.0	1.0	5.0	4.0	1.2
DIT	1.9	2.0	2.0	0.8	0.7	0.8	1.0	4.0	1.0	5.0	1.0	2.0	1.0	0.4
LCOM	39.7	44.0	0.0	32.2	1036.7	0.0	− 1.4	100.0	0.0	100.0	0.0	68.0	68.0	0.8
WMC	13.8	9.0	5.0	19.5	379.1	5.9	53.5	229.0	0.0	229.0	5.0	16.0	11.0	1.4

Table 8 Descriptive statistics of Avisync dataset

Avisync	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.1	1.0	1.0	0.4	0.2	2.1	6.4	3.0	0.0	3.0	1.0	1.0	0.0	0.4
CBO	3.1	1.0	0.0	4.8	22.5	2.0	4.0	21.0	0.0	21.0	0.0	4.0	4.0	1.5
NOC	0.6	0.0	0.0	1.5	2.3	3.4	14.0	9.0	0.0	9.0	0.0	0.0	0.0	2.5
NIM	7.7	6.0	1.0	7.7	59.9	1.3	1.2	32.0	0.0	32.0	1.0	11.0	10.0	1.0
NIV	2.1	1.0	0.0	2.8	7.9	2.2	5.6	14.0	0.0	14.0	0.0	3.0	3.0	1.4
RFC	14.9	8.0	5.0	12.8	162.9	0.9	− 0.6	44.0	0.0	44.0	5.0	24.0	19.0	0.9
NPM	1.6	0.0	0.0	4.1	17.0	3.8	15.2	23.0	0.0	23.0	0.0	2.0	2.0	2.6
LOC	46.2	32.0	5.0	55.9	3124.0	2.2	4.8	247.0	4.0	251.0	5.0	61.0	56.0	1.2
MAX_CC	2.3	1.0	1.0	2.5	6.3	3.5	17.0	17.0	0.0	17.0	1.0	3.0	2.0	1.1
DIT	2.4	2.0	1.0	1.4	1.8	0.5	− 1.0	4.0	1.0	5.0	1.0	4.0	3.0	0.6
LCOM	68.0	81.0	100.0	35.0	1222.7	− 1.0	− 0.2	100.0	0.0	100.0	57.0	100.0	43.0	0.5
WMC	11.1	7.0	1.0	12.4	153.3	1.8	3.3	58.0	0.0	58.0	1.0	15.0	14.0	1.1

Table 9 Descriptive statistics of Jfreechart dataset

Jfreechart	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.5	1.0	1.0	0.9	0.9	3.2	14.3	8.0	0.0	8.0	1.0	2.0	1.0	0.6
CBO	4.5	2.0	1.0	6.4	40.5	3.9	31.9	81.0	0.0	81.0	1.0	6.0	5.0	1.4
NOC	0.3	0.0	0.0	1.3	1.8	6.3	45.8	14.0	0.0	14.0	0.0	0.0	0.0	4.3
NIM	9.9	5.0	4.0	15.2	230.1	5.5	43.6	172.0	0.0	172.0	4.0	11.0	7.0	1.5
NIV	2.2	1.0	0.0	4.9	23.7	5.1	35.1	46.0	0.0	46.0	0.0	2.5	2.5	2.2
RFC	33.8	7.0	5.0	62.3	3876.5	2.5	5.1	264.0	1.0	265.0	5.0	27.0	22.0	1.8
NPM	0.5	0.0	0.0	1.0	1.1	2.6	7.5	7.0	0.0	7.0	0.0	0.0	0.0	2.1
LOC	126.5	74.0	5.0	186.0	34,600.4	4.8	33.5	2148.0	4.0	2152.0	42.0	133.0	91.0	1.5
MAX_CC	5.3	2.0	2.0	6.5	42.8	3.0	11.6	51.0	0.0	51.0	2.0	6.0	4.0	1.2
DIT	2.0	2.0	2.0	0.9	0.8	1.4	2.6	5.0	1.0	6.0	2.0	2.0	0.0	0.4
LCOM	38.9	41.0	0.0	37.9	1433.3	0.1	− 1.7	100.0	0.0	100.0	0.0	75.0	75.0	1.0
WMC	21.9	9.0	8.0	38.1	1449.4	6.0	53.5	489.0	0.0	489.0	6.0	22.0	16.0	1.7

Table 10 Descriptive statistics of Jgap dataset

Jgap	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.3	1.0	1.0	0.7	0.5	2.0	7.1	5.0	0.0	5.0	1.0	2.0	1.0	0.6
CBO	4.2	4.0	5.0	4.4	19.0	3.2	16.0	34.0	0.0	34.0	1.0	5.0	4.0	1.0
NOC	0.6	0.0	0.0	3.7	13.7	8.8	82.3	39.0	0.0	39.0	0.0	0.0	0.0	6.0
NIM	8.8	7.0	7.0	9.4	88.5	3.1	13.5	65.0	0.0	65.0	3.0	10.0	7.0	1.1
NIV	2.3	1.0	0.0	3.6	13.0	3.1	12.7	25.0	0.0	25.0	0.0	3.0	3.0	1.6
RFC	31.9	12.0	2a	31.9	1015.7	0.4	− 1.7	83.0	0.0	83.0	4.0	71.0	67.0	1.0
NPM	0.2	0.0	0.0	0.8	0.6	7.3	65.9	8.0	0.0	8.0	0.0	0.0	0.0	4.6
LOC	74.3	46.0	36.0	109.8	12,059.7	5.3	36.8	1011.0	3.0	1014.0	25.0	77.0	52.0	1.5
MAX_CC	3.4	2.0	2.0	3.6	12.7	3.0	12.2	26.0	0.0	26.0	1.0	4.0	3.0	1.1
DIT	1.7	1.0	1.0	0.8	0.6	0.6	− 1.2	2.0	1.0	3.0	1.0	2.0	1.0	0.5
LCOM	76.6	84.0	100.0	27.0	730.1	− 1.4	1.7	100.0	0.0	100.0	64.0	100.0	36.0	0.4
WMC	15.4	10.0	8.0	21.4	455.8	4.5	26.1	179.0	0.0	179.0	5.0	17.0	12.0	1.4

Table 11 Descriptive statistics of Jtreeview dataset

Jtreeview	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.6	1.0	1.0	1.2	1.4	3.1	12.5	10.0	0.0	10.0	1.0	2.0	1.0	0.8
CBO	3.5	2.0	0.0	4.8	22.8	3.8	24.0	48.0	0.0	48.0	1.0	4.0	3.0	1.4
NOC	0.2	0.0	0.0	1.0	0.9	6.2	51.2	11.0	0.0	11.0	0.0	0.0	0.0	3.9
NIM	8.1	5.0	1.0	10.0	99.8	3.3	17.2	92.0	0.0	92.0	2.0	10.0	8.0	1.2
NIV	3.2	2.0	0.0	4.6	20.8	2.7	10.0	33.0	0.0	33.0	0.0	4.0	4.0	1.4
RFC	13.8	8.0	1.0	18.3	333.4	2.7	8.4	104.0	0.0	104.0	3.0	17.0	14.0	1.3
NPM	0.9	0.0	0.0	1.7	3.0	2.9	9.5	12.0	0.0	12.0	0.0	1.0	1.0	2.0
LOC	99.5	54.0	10a	131.5	17,297.0	3.7	19.7	1153.0	3.0	1156.0	27.0	124.5	97.5	1.3
MAX_CC	4.0	3.0	1.0	3.6	13.3	1.8	3.1	20.0	0.0	20.0	1.0	5.0	4.0	0.9
DIT	1.9	2.0	2.0	0.7	0.5	0.7	1.2	4.0	1.0	5.0	1.0	2.0	1.0	0.4
LCOM	41.4	50.0	0.0	35.0	1225.4	0.0	− 1.6	100.0	0.0	100.0	0.0	75.0	75.0	0.9
WMC	16.0	8.0	1.0	21.1	443.5	3.1	12.9	153.0	0.0	153.0	4.0	20.0	16.0	1.3

Table 12 Descriptive statistics of barcode4j dataset

barcode4j	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.8	1.0	1.0	1.3	1.6	2.0	4.5	7.0	1.0	8.0	1.0	2.0	1.0	0.7
CBO	3.1	3.0	0.0	2.7	7.4	0.8	0.0	11.0	0.0	11.0	1.0	5.0	4.0	0.9
NOC	0.3	0.0	0.0	1.2	1.4	5.0	28.5	9.0	0.0	9.0	0.0	0.0	0.0	3.5
NIM	4.8	3.0	1.0	4.6	20.8	1.7	3.4	24.0	0.0	24.0	2.0	6.0	4.0	1.0
NIV	1.1	0.0	0.0	2.2	4.7	2.7	8.0	12.0	0.0	12.0	0.0	1.0	1.0	2.0
RFC	9.8	7.0	3.0	9.8	95.8	1.6	1.9	41.0	1.0	42.0	3.0	12.0	9.0	1.0
NPM	0.8	0.0	0.0	2.1	4.2	3.8	17.7	15.0	0.0	15.0	0.0	1.0	1.0	2.5
LOC	65.7	43.0	4.0	83.0	6886.0	4.7	34.9	788.0	3.0	791.0	18.0	87.3	69.3	1.3
MAX_CC	4.4	3.0	1.0	4.6	21.6	2.6	10.1	32.0	1.0	33.0	1.0	6.0	5.0	1.1
DIT	1.8	2.0	2.0	0.7	0.5	0.6	− 0.1	3.0	1.0	4.0	1.0	2.0	1.0	0.4
LCOM	31.6	0.0	0.0	37.1	1379.0	0.5	− 1.6	96.0	0.0	96.0	0.0	75.0	75.0	1.2
WMC	12.2	7.0	1.0	13.6	184.1	2.4	7.0	77.0	1.0	78.0	3.0	17.0	14.0	1.1

Table 13 Descriptive statistics of Jtopen dataset

Jtopen	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.8	1.0	1.0	2.0	3.8	6.7	75.6	34.0	0.0	34.0	1.0	2.0	1.0	1.1
CBO	5.1	3.0	0.0	6.2	38.3	2.5	12.0	72.0	0.0	72.0	1.0	7.0	6.0	1.2
NOC	0.4	0.0	0.0	2.4	5.5	10.7	144.0	39.0	0.0	39.0	0.0	0.0	0.0	5.5
NIM	10.1	5.0	1.0	16.0	256.7	5.4	46.6	217.0	0.0	217.0	2.0	11.0	9.0	1.6
NIV	3.6	1.0	0.0	5.6	31.0	3.2	15.5	53.0	0.0	53.0	0.0	5.0	5.0	1.6
RFC	20.8	15.0	7.0	22.4	502.8	2.9	13.4	217.0	0.0	217.0	6.0	27.0	21.0	1.1
NPM	0.9	0.0	0.0	1.7	3.0	4.2	26.2	20.0	0.0	20.0	0.0	1.0	1.0	2.0
LOC	151.4	74.	35.	248.1	61,559.4	5.2	40.2	3135	1.0	3136	35.0	163	128	1.6
MAX_CC	6.1	4.0	1.0	9.7	94.8	6.0	52.1	135.0	0.0	135.0	1.0	7.0	6.0	1.6
DIT	2.0	2.0	2.0	1.0	0.9	0.8	0.1	4.0	1.0	5.0	1.0	3.0	2.0	0.5
LCOM	73.6	84.0	100.0	28.8	829.3	− 1.3	0.8	100.0	0.0	100.0	62.0	95.0	33.0	0.4
WMC	23.7	11.0	1.0	38.9	1513.1	4.5	29.7	443.0	0.0	443.0	4.0	28.0	24.0	1.6

Table 14 Descriptive statistics of Jung dataset

Jung	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.6	1.0	1.0	1.1	1.2	2.0	4.3	6.0	0.0	6.0	1.0	2.0	1.0	0.7
CBO	5.1	4.0	0.0	4.6	21.4	0.9	0.6	23.0	0.0	23.0	1.0	8.0	7.0	0.9
NOC	0.4	0.0	0.0	1.0	1.0	3.2	11.4	6.0	0.0	6.0	0.0	0.0	0.0	2.6
NIM	7.2	4.0	1.0	7.6	57.2	1.7	3.0	40.0	0.0	40.0	2.0	10.0	8.0	1.1
NIV	2.2	1.0	0.0	2.9	8.2	2.4	7.0	16.0	0.0	16.0	0.0	3.0	3.0	1.3
RFC	17.6	9.0	4.0	18.4	339.3	1.2	0.1	66.0	0.0	66.0	4.0	30.5	26.5	1.1
NPM	0.5	0.0	0.0	1.1	1.2	3.1	11.0	7.0	0.0	7.0	0.0	0.0	0.0	2.4
LOC	68.2	40.0	4.0	67.1	4506.3	1.3	0.9	276.0	2.0	278.0	16.5	105.5	89.0	1.0
MAX_CC	3.9	3.0	1.0	3.4	11.6	1.9	5.5	22.0	0.0	22.0	1.0	5.0	4.0	0.9
DIT	1.8	1.0	1.0	1.0	1.1	1.3	1.1	4.0	1.0	5.0	1.0	2.0	1.0	0.6
LCOM	35.6	35.0	0.0	34.0	1158.5	0.2	− 1.5	100.0	0.0	100.0	0.0	68.0	68.0	1.0
WMC	14.7	9.0	1.0	14.6	213.6	1.7	4.0	87.0	0.0	87.0	4.0	21.5	17.5	1.0

Table 15 Descriptive statistics of Geotag dataset

Geotag	Mean	Med	Mod	SD	V	S	K	R	Min	Max	1Q	3Q	IQR	CoV
AVG_CC	1.8	1.0	1.0	1.7	3.0	2.9	11.9	15.0	0.0	15.0	1.0	2.0	1.0	0.9
CBO	2.7	1.0	1.0	4.2	17.7	4.8	38.2	52.0	0.0	52.0	1.0	3.0	2.0	1.5
NOC	0.3	0.0	0.0	1.6	2.7	8.7	87.3	19.0	0.0	19.0	0.0	0.0	0.0	4.8
NIM	5.4	3.0	1.0	7.7	58.6	4.9	35.8	88.0	0.0	88.0	1.0	6.0	5.0	1.4
NIV	1.7	0.0	0.0	3.3	10.6	4.1	22.5	29.0	0.0	29.0	0.0	2.0	2.0	2.0
RFC	11.5	5.0	2.0	14.4	207.1	2.0	4.1	88.0	0.0	88.0	2.0	14.0	12.0	1.3
NPM	0.7	0.0	0.0	1.9	3.6	5.6	44.1	22.0	0.0	22.0	0.0	1.0	1.0	2.8
LOC	72.1	35.0	10.0	127.3	16,200.4	7.8	98.5	2039.0	2.0	2041.0	15.0	79.5	64.5	1.8
MAX_CC	4.3	2.0	1.0	6.9	47.6	8.2	109.9	113.0	0.0	113.0	1.0	5.0	4.0	1.6
DIT	1.9	2.0	2.0	1.1	1.1	1.6	3.0	5.0	1.0	6.0	1.0	2.0	1.0	0.5
LCOM	37.4	44.0	0.0	36.2	1312.3	0.2	− 1.5	100.0	0.0	100.0	0.0	70.0	70.0	1.0
WMC	12.9	6.0	2.0	22.3	498.2	7.8	103.0	365.0	0.0	365.0	3.0	14.0	11.0	1.7

132 (12 × 11) combinations from 12 datasets. Precision, recall and AUC are analyzed to predict whether prediction is possible or not. If precision > 0.6 and recall > 0.7 and AUC > 0.6, then we assume that prediction is possible, otherwise not. Analyzing the acceptance criterion of various prediction models that are available in literature chooses these cut off values. Choosing these cut off values of precision, recall and AUC, prediction was found possible in 35 out of 132 permutations. Then we used the distributive characteristics of these datasets to build the decision node of the decision tree and the leaf node tells whether prediction is possible or not.

To construct the decision tree, we have used weka 3.6.10. Random tree algorithm is used to construct the decision tree with 10 fold cross validation on the dataset. The dataset is constructed in the following manner: first we List all the distributive characteristics for all the metrics for the training data set followed by the distributive characteristics of test dataset. The last column is a binary variable, which tells prediction is possible for this permutation, or not. Assuming we have m distribution characteristics for n metrics, the total number of columns in the dataset will be 2(m × n) +1. In our case, m = 14 and n = 12, hence the total number of columns in the dataset = 337. The number of rows is equal to the number of permutations of the training and test sets.

The procedure for construction of dataset for decision tree is shown in Fig. 2. The prediction model is built by training from a software system and tested on all remaining datasets. The result is marked "yes" if the criterion for successful prediction is satisfied else "no". Now we calculate distributive characteristics for all metrics of both train and test sets and combine them with the prediction result as shown in Fig. 2. This gives one row of the combined dataset. Now we repeat the process for all combinations to complete the dataset for learning of the decision tree.

4 Results and findings

4.1 Experimental results

We generated 132 train-test instances from the various combinations of the datasets. Out of these 132 instances, 35 were successful with the values of precision, recall and AUC greater than the cut off values. Thus we get only 26.5% successful cross project defect prediction scenarios. The best prediction results are observed with Amberarcher as test set and various training sets. The highest values of precision recall and AUC are obtained with barcode4j and Geotag as training sets and Amberarcher as test set. Precision and recall for both these models is greater than 80% and AUC is greater than 70%. Table 16 lists the successful train-test combinations and corresponding values of precision, recall and AUC.

The size of the decision tree learnt from these train test instances is 75. It consists of 38 leaf nodes out of which 15

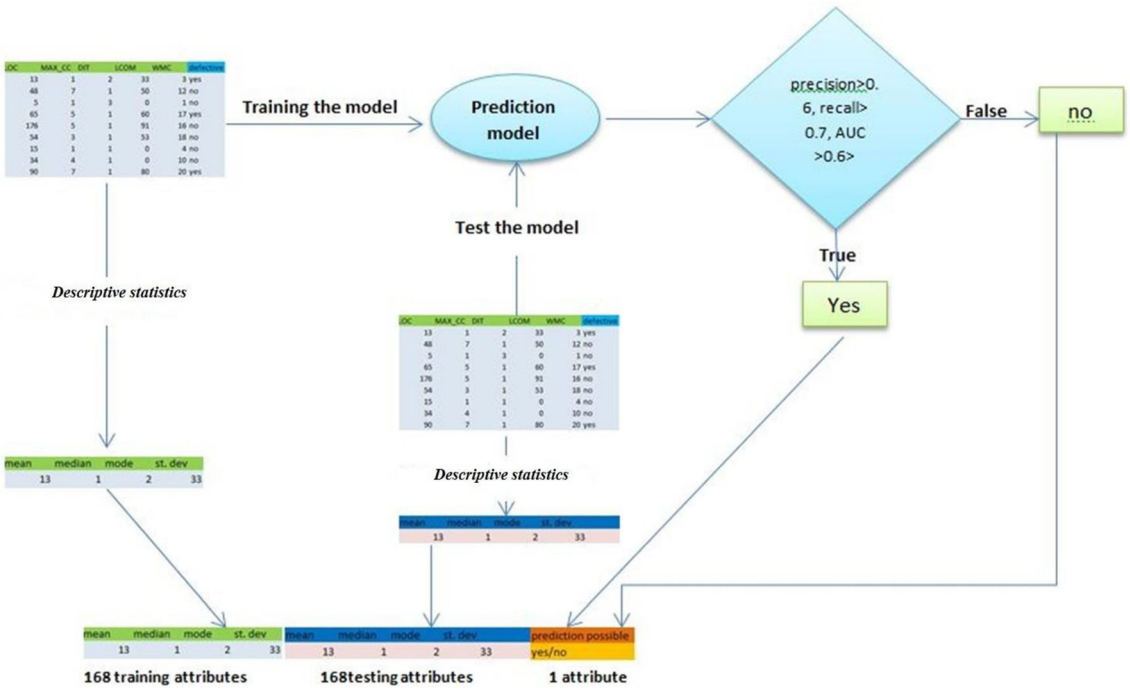

Fig. 2 Generation of training-test instance from the dataset combination

Table 16 Successful prediction results

Training	Testing	Precision	Recall	AUC
Amakihi	Amberarcher	0.848	0.791	0.63
Amakihi	Barcode4j	0.74	0.706	0.721
Amberarcher	Abbot	0.763	0.7	0.793
Amberarcher	Apollo	0.743	0.733	0.648
Amberarcher	Avisync	0.697	0.701	0.763
Amberarcher	barcode4j	0.769	0.782	0.775
Abbot	Jtreeview	0.738	0.738	0.797
Apollo	Amberarcher	0.842	0.886	0.754
Apollo	Abbot	0.763	0.7	0.895
Apollo	Jgap	0.717	0.705	0.78
Apollo	Barcode4j	0.769	0.818	0.809
Avisync	Amakihi	0.724	0.724	0.754
Jfreechart	Amberarcher	0.857	0.848	0.656
Jfreechart	Abbot	0.777	0.776	0.87
Jfreechart	Jtreeview	0.72	0.721	0.786
Jgap	Amberarcher	0.832	0.89	0.335
Jgap	Abbot	0.792	0.721	0.713
Jgap	Apollo	0.764	0.781	0.713
Jtreeview	Abbot	0.811	0.809	0.901
Jtreeview	Barcode4j	0.797	0.724	0.788
Barcode4j	Amberarcher	0.86	0.84	0.711
Barcode4j	Abbot	0.739	0.7	0.826
Barcode4j	Apollo	0.707	0.709	0.678
Barcode4j	Avisync	0.708	0.701	0.729
Jung	Amberarcher	0.854	0.851	0.62
Jung	Apollo	0.755	0.767	0.604
Jung	barcode4j	0.82	0.841	0.67
Jung	Geotag	0.802	0.841	0.504
Geotag	Amberarcher	0.859	0.887	0.709
Geotag	Apollo	0.761	0.801	0.673
Geotag	Barcode4j	0.788	0.824	0.812
Amberarcher	Geotag	0.781	0.788	0.596
Apollo	Geotag	0.797	0.83	0.687
Jgap	Geotag	0.776	0.728	0.535
Barcode4j	Geotag	0.787	0.815	0.708

are labeled "yes" and 23 are labeled "no". We performed 10-fold cross validation and observed precision 74.7%, recall 74.2% and AUC 67.9%. The decision tree is built using random tree algorithm. Table 17 lists the top 3 rules derived from the decision tree for successful cross project defect prediction. The support indicates the no. of instances which satisfy the rule. Only 37 out of 336 project characteristics are found significant in the construction of decision tree. 24 of these 37 deciding characteristics are of training set and the rest 13 of test set. These characteristics are compared with a cut off value at each deciding node and the value decides the class whether "yes" or "no".

4.2 Discussion of results

From the results obtained from our experiments, some of the common observations we concluded are:

- Software with lower percentage of defective classes has a very large set of potential defect predictors.
- Defects for large software systems cannot be predicted by relatively smaller software systems.
- Datasets with huge difference in the number of classes cannot be used in cross project defect prediction.

Table 18 lists the datasets which can be used for identification of defective classes for each of the training dataset. Here we can see that Jtopen is not useful in defect proneness prediction of any of the software under study. Amber archer, Barcode4j and Jung are potential predictors for 5 and 4 datasets respectively. Abbot and Avisync are predictors for only 1 dataset while jtopen for none.

Figure 3 shows the diagrammatic representation of the potential predictors for software. The X-axis shows the test dataset and Y-axis shows the count of the potential predictors. This helps in the relative study of the potential predictors. Amber archer has the highest number of predictors while jfreechart, jtopen and jung can't be predicted by any of the training sets. However, if we relax some of the acceptance criterion, we obtain better results with these software systems. Apollo, avisync, barcode4j and geotag also have ample training sets. Thus we can see that 9 out of 12 software systems under study can be successfully predicted by one or more training sets.

4.3 Threats to validity

One of the major threats to validity of our work is the acceptance criteria of the successful model. We have selected three parameters for successful model i.e. precision, recall and AUC. The selection of these parameters is based on previous studies in literature about defect prediction and our own analysis. However, the acceptance criteria may vary depending on various factors. In such a case, some of our observations and conclusions may change.

Another threat is the selection of static code metrics to build the defect prediction model. Studies in literature show that these metrics can be used for defect prediction models, but it is not always the case. The appropriate selection of these metrics may vary depending on the dataset. A subset of these metrics is found significant in a large number of studies. Thus we can conclude that our

Table 17 Top 3 rules for successful prediction learnt from DT

Rule	Support
Kurtosis_DIT_test > = 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test < 12.5 AND Median_LCOM_train < 57 AND Range_CBO_train < 66.5 AND Variance_LOC_train < 16,748.7	8
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test > = 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train < 56.22 AND Skewness_WMC_train > = 2.28	7
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test > = 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train > = 56.22 AND Range_avg_cc_train > = 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test < 0.5 AND Mode_CBO_train > = 0.5	3

Table 18 Performance of training sets

Training dataset	Prediction possible for
Amakihi	Amber archer, Barcode4j
Amber archer	Abbot, Apollo, Avisync, Barcode4j, Geotag
Abbot	Jtreeview
Apollo	Amber archer, Abbot, Jgap, Barcode4j, Geotag
Avisync	Amakihi
Jfreechart	Amber archer, Abbot, Jtreeview
Jgap	Abbot, Apollo, Geotag
Jtreeview	Abbot, Barcode4j
Barcode4j	Amber archer, Abbot, Apollo, Avisync, Geotag
Jung	Amber archer, Apollo, Barcode4j, Geotag
Geotag	Amber archer, Apollo, Barcode4j

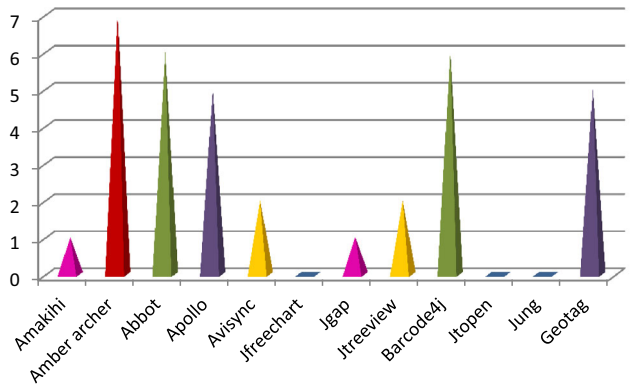

Fig. 3 potential defect predictors for dataset under study

experiments and observations may vary depending on the selection of these static code metrics.

5 Conclusions and future work

5.1 Conclusions

Cross project defect prediction is the process of learning from one project to improve another project. It is applicable to the Software Development Life Cycle to improve the software quality and make the software system more reliable and gain more confidence and customer's satisfaction. Also choosing more than one prediction model trained by different sets will increase the confidence of the prediction results. This is not possible in the traditional defect prediction method because model is trained from the previous release of same system. Training the model with different data will increase the reliance on prediction results. This will increase the reliability, traceability, usability and maintainability of the software systems and will help in mitigating software crisis.

In this work, the prediction model is built with this using logistic regression. We conducted our experiments on 12 open source projects. 132 combinations of train-test instances are generated from these 12 projects and the feasibility of cross project defect prediction is analyzed. The results show that cross project defect prediction is not always feasible. Only 35 out of 132 instances exhibit successful cross project prediction behavior in our experiments. Thus careful selection of training set needs to be done in order to identify defective classes correctly. The decision tree, constructed in our experiment learns from the distributive characteristics of software projects to generate rules for successful defect prediction. This may help in selection of appropriate training sets.

Chances of successful cross project defect prediction are more likely for comparable number of classes in the training and test sets. It is observed that jtopen; with very high number of classes and high LOC than other software systems is neither a good training nor a test set. Cross project defect prediction may provide acceptable results, if careful selection of training set is done.

5.2 Future work

Previous studies show that application of some machine learning algorithms builds better prediction models than statistical method. We may apply machine learning methods as bagging, naïve bayes etc. to build the prediction model instead of logistic regression in future. This may

improve the performance of the model as well as decision tree.

Our experiments for cross project do not take into account the programming language of the software under study. All the projects under study are developed using same programming language i.e. java. We may extend our work where combinations of different programming languages are taken and verify if the prediction works as well in such scenarios or not. We may also extend the work on real life corporate software. The process followed and the complexity of industrial software is different from that of open source software and hence we may take these features also into account. This will make the application of cross project defect prediction more realistic and applicable to software industry.

References

1. Zimmermann T, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction
2. Malhotra R, Agrawal A (2014) CMS tool. ACM SIGSOFT Softw. Eng. Notes 39(1):1–5
3. Radjenović D, Heričko M, Torkar R, Živkovič A (2013) Software fault prediction metrics: a systematic literature review. Inf Softw Technol 55(8):1397–1418
4. Gray R, Macdonell SG (1997) A comparison of techniques for developing predictive models of software metrics. Inf Softw Technol 5849(96):6–7
5. Mishra B, Shukla KK (2011) Impact of attribute selection on defect proneness prediction in OO software. In: 2011 2nd Int. Conf. Comput. Commun. Technol., pp 367–372
6. Chidamber Shyam R, Kemerer Chris F (1994) A Metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493
7. He Z, Shu F, Yang Y, Li M, Wang Q (2011) An investigation on the feasibility of cross-project defect prediction. Autom Softw Eng 19(2):167–199
8. Ma Y, Luo G, Zeng X, Chen A (2012) Transfer learning for cross-company software defect prediction. Inf Softw Technol 54(3):248–256
9. Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. Empir Softw Eng 14(5):540–578
10. Canfora G, De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2013) Multi-objective cross-project defect prediction. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luembourg, pp 252–261
11. Steffen H (2013) Training data selection for cross-project defect prediction.In: 9th International Conference on Predictive Models in Software Engineering, ACM, New York, USA, p 10
12. Ryu D, Choi O, Baik J (2014) Improving prediction robustness of VAB-SVM for cross-project defect prediction. In: IEEE 17th International Conference on Computational Science and Engineering, Chengdu, pp 994–999
13. Panichella R, Oliveto R, De Lucia A (2010) Cross-project defect prediction models: L'Union fait la force. Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), Antwerp, pp 164–173
14. Amasaki S, Kawata K, Yokogawa T (2015) Improving cross-project defect prediction methods with data simplification. In: 41st Euromicro Conference on Software Engineering and Advanced Applications, Funchal, pp 96–103
15. Herbold S (2015) CrossPare: a tool for benchmarking cross-project defect predictions. In: 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), Lincoln, NE, pp 90–96
16. Satin RFP, Wiese IS, Ré R (2015) An exploratory study about the cross-project defect prediction: impact of using different classification algorithms and a measure of performance in building predictive models. In: Latin American Computing Conference (CLEI), Arequipa, pp 1–12
17. Zhang Y, Lo D, Xia X, Sun J (2015) An empirical study of classifier combination for cross-project defect prediction. In: IEEE 39th Annual Computer Software and Applications Conference, Taichung, pp 264–269
18. Peters F, Menzies T, Layman L (2015) LACE2: better privacy-preserving data sharing for cross project defect prediction. IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, pp 801–811
19. Xia X, Lo D, Pan SJ, Nagappan N, Wang X (2016) HYDRA: massively compositional model for cross-project defect prediction. IEEE Trans Softw Eng 42(10):977–998
20. Ryu D, Baik J (2016) Effective multi-objective naïve Bayes learning for cross-project defect prediction. Appl Soft Comput 49:1062–1077
21. Zhang F, Zheng Q, Zou Y, Hassan AE (2016) Cross-project defect prediction using a connectivity-based unsupervised classifier. In: IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, pp 309–320
22. Hosseini S, Turhan B, Mantyla M (2016) Search based training data selection for cross project defect prediction. In: The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, ACM, New York, USA, p 10
23. Zhang F, Keivanloo I, Zou Y (2017) Data transformation in cross-project defect prediction. Empir Softw Eng 22(6):3186–3218
24. Fei W et al. (2017) Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In: IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, pp 195–197
25. Poon WN, Bennin KE, Huang J, Phannachitta P, Keung JW (2017) Cross-project defect prediction using a credibility theory based naive Bayes classifier. In: IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, pp 434–441
26. Huang S, Wu Y, Ji H, Bai C (2017) A three-stage defect prediction model for cross-project defect prediction. In: International conference on dependable systems and their applications (DSA), Beijing, pp 169–169
27. Jing XY, Wu F, Dong X, Xu B (2017) An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans Softw Eng 43(4):321–339
28. Goel L, Damodaran D, Khatri SK, Sharma M (2017) A literature review on cross project defect prediction. In: 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON), Mathura, pp 680–685
29. http://amakihi.sourceforge.net/. Accessed 10 Aug 2017
30. http://sourceforge.net/projects/amberarcher/. Accessed 10 Aug 2017
31. http://abbot.sourceforge.net/doc/overview.shtml. Accessed 10 Aug 2017

32. http://sourceforge.net/projects/startec-apollo. Accessed 10 Aug 2017
33. http://sourceforge.net/projects/avisync/. Accessed 10 Aug 2017
34. http://sourceforge.net/projects/jfreechart/. Accessed 10 Aug 2017
35. http://sourceforge.net/projects/jgap/. Accessed 10 Aug 2017
36. http://sourceforge.net/projects/jtreeview/. Accessed 10 Aug 2017
37. http://sourceforge.net/projects/barcode4j/. Accessed 10 Aug 2017
38. http://sourceforge.net/projects/jt400/. Accessed 10 Aug 2017
39. http://sourceforge.net/projects/jung/. Accessed 10 Aug 2017
40. http://sourceforge.net/projects/geotag/. Accessed 10 Aug 2017