International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 7 (2015) pp. 17375-17391

http://www.ripublication.com

Product Metrics Based Predictive Classification of Software Using RAR Mining and Naive Bayes Approach

Kumudha. P, Assistant Professor(SG),CSE
Venkatesan. R, Professor, CSE
Radhika Engimuri, PG Scholar,CSE
Coimbatore Institute of Technology, Anna University,Coimbatore-641014,Tamilnadu
PSG College of Technology, Anna University,Coimbatore-641004,Tamilnadu
Coimbatore Institute of Technology, Anna University,Coimbatore-641014,Tamilnadu
kumudha.cit.cse@gmail.com, rve@cse.psgtech.ac.in,radhi.eg@gmail.com
9944970398 9789219522 7708936665

Abstract

To improve Software quality, it is essential for software developers to identify defective software modules at any phase of Software Development Life Cycle (SDLC). Many machine learning based classification models were designed and are still getting improved to solve the problem of defect prediction. The effectiveness of these models is influenced mainly by two key quality of data factors – set of software metrics used to build the models and proportion of defect-prone instances in the software measurement data set. In this paper, we proposed a classification model which is a combination of Naive Bayes and relational association rules. The classifier discovers relational association rules on the metrics data based on user defined confidence and support during training stage and integrates using Naive Bayes at testing stage to predict whether a software module is defective or non defective. An experimental evaluation of the proposed model is carried out on open source bug prediction dataset containing object oriented metrics. The proposed model performed better with class imbalance data and is compared with existing machine learning based techniques – Naive Bayes, Bagging and One R. The results obtained are positive having better improvement than existing mainly for the evaluation measures – Recall and F-Measure. The proposed model has shown high True positive rate of defective modules which signifies the model capability of identifying defective modules as defective than any other classifiers used in the experiment. This confirms the potential of the proposal.

Key-Words: Data Mining, Association Rules, Naive Bayes, Software Defect Prediction.

Introduction

Software quality is considered to be of great importance in the area of software engineering. In order to increase the efficiency and the quality of software modules, software defect prediction is used to identify defect prone modules and helps in achieving high software reliability. Any project having many defects lacks quality and thus techniques and methodologies for predicting the defects can help in reducing the faultiness during software testing process which results in high quality software product. Prediction models with input as software metrics, can predict the number of defective software modules. Software metrics are attributes which includes process, productor source code metrics of the software system. There are many metrics that have proved their value for system maintenance and modification and hence selecting metrics plays a key role in software defect prediction. For our work, we have used one set of dataset that is made public by D'Ambros et al. to evaluate bug prediction techniques [1], [2]. Basically, the dataset includes seventeen product based object oriented metrics collected at class-level for Java based Systems. Another set of product metrics includes static code metrics that are available in PROMISE Repository [3].

In Association rule mining [4], frequently occurring attribute value conditions are searched in a dataset and used for prediction purpose thereby neglecting non frequent attribute values. Ordinal association rules [5]are a particular type of association rules that specifies ordinal relationship i.e., numerical ordering between attributes for a certain percentage of records. However, in real world scenarios, there might not exist ordinal relationship all the time between attributes in which case ordinal association rules are not strong enough to describe the different domains and relationships between the attributes. In such situations, ordinal association rules are not strong enough to describe data consistency. Consequently, relational association rules were introduced in [6] in order to capture various kinds of relationships between record attributes.

This paper proposes novel approach based on Naive Bayes using relational association rules. Naive Bayes is simple yet powerful. The primary Naive Bayes assumption is that for a given class, the features are conditionally independent. Even if the features are not independent, each feature is considered as independent in terms of how it contributes to the classification of the set. By combining Relational Association Rules with Naive Bayes approach the results obtained are promising.

The paper is organized as follows. We start with related work and background in section 2. Next, Section 3 introduces our methodology by including system design, Training, classification process and measurement technique. An experimental evaluation with result analysis is reported in section 4. Conclusions and future work are outlined in section 5.

Related Work and Background

Related Work

This section briefly describes the various studies done in the field of the association rule mining, relationships between object oriented metrics and its impact in defect prediction models. Association rule mining [7] identifies interesting relationships among data items and is used as unsupervised learning scenarios. The discovery of these relationships can help in many decision making processes. Association rule mining finds those rules that satisfy some minimum support and minimum confidence constraints. Many extensions of association rules are discovered for classification. One such is CBA(Classification Based Association) method presented in [8] where association rules are mined for pre determined target class. CBA2 is an extension of CBA where association rules predict different classes having different minimum support to solve data imbalance problem. A novel classification model based on relational association rule mining has been proposed in [9] to predict whether a software module is defective or not. The Defect Prediction Relational Association Rules (DPRAR) classifier defines set of relations between the metric values during training process. At the classification stage, a new software entity is declared as positive or negative based on measurement how 'close' the entity is to the positive instances and also how 'far' it is from negative one. The obtained results shown that DPRAR classifier performed better than existing classifiers – CBA2, OneR and Bagging.

Apart from rule-based methods, many different machine learning algorithms are developed to solve software defect prediction problem. Menzies et al., [10], evaluated the Naive Bayes classifier, OneR and J48 and concluded that Naive Bayes produced the best results on the 8 used NASA datasets. Another paper [11] presents Text categorization using combination of association rules and Naive Bayes where in instead of using words, word relation is used to derive feature set from pre-classified text documents. Naïve Bayes Classifier is then applied on derived features for final categorization. The obtained results were promissory with probabilistic nature of Naive Bayes approach.

Metrics that are playing key role as feature set also been analysed and applied to compare different approaches. Ref. [12] evaluated the object-oriented metrics given by Chidamber and Kemerer, and few other static code metrics for couple of open source projects. The empirical study is to find out nature of relationship between these metrics and defects. In another literature study [13], results shows that 49% of the metrics used in defect prediction models are Object-oriented metrics when compared to 27% of traditional source code metrics or 24% of processmetrics.Chidamber and Kemerer's (CK) object oriented metrics were most frequently used compared to traditional size and complexity metrics. Menzies.,[14] described the value ofusing McCabes and Halstead static code attributes to learn defect predictor models. These static code attributes are module based metrics, easy to use and are widely being used

Background

The objective of relational association rule mining is to find relationships between features that hold over large percentage of records. In a binary classification problem, if an attribute A is in ">" relation with attribute B for majority of positive instances then a record in which attribute A holds "<" relation with B may be belongs to negative instance. It may not mean very much if only one rule including B is not fulfilled, but it increases the likelihood that the instance inquestion belongs to the negative class if many such rules are broken.

The following will briefly review the concept of relational association rules, as well as the mechanism for identifying therelevant relational association rules that hold within a dataset.

throughx$_n$, Baye's theorem states the following relationship:

$$P\left(\frac{\mathrm{c}}{\mathrm{x}}\right) = \frac{\mathrm{P}\left(\frac{\mathrm{x}}{\mathrm{c}}\right)\ \mathrm{P(c)}}{\mathrm{P(x)}} \tag{1}$$

Let R = $\{\boldsymbol{r_1}, \boldsymbol{r_2}, \ldots . \boldsymbol{r_n}\}$ be a set of instances in relational model where each instance is characterized by list of *m* attributes, $(\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_m})$**.** We denote by $\boldsymbol{\Phi}(\boldsymbol{r_j}, \boldsymbol{a_i})$ as the value of attribute $\boldsymbol{a_i}$ for the instance $\boldsymbol{r_j}$. Each attribute $\boldsymbol{a_i}$ takes value from a domain $\boldsymbol{D_i}$. The domain $\boldsymbol{D_i}$ represents the type of data that a class can have and different classes represents different data. Between two domains $\boldsymbol{D_i}$ and $\boldsymbol{D_j}$ relations can be defined as: less or equal (<=) and greater than (>). We denote by M the set of all possible relations that can be defined on $\boldsymbol{D_i x D_j}$.

A relational association rule [6] is an expression $(\boldsymbol{a_{i_1}}, \boldsymbol{a_{i_2}}, \boldsymbol{a_{i_3}}, \ldots, \boldsymbol{a_{i_l}})$ => $(\boldsymbol{a_{i_1}}\boldsymbol{\mu_1}\boldsymbol{a_{i_2}}\boldsymbol{\mu_2}\boldsymbol{a_{i_3}}\ldots\ldots\boldsymbol{\mu_{l-1}}\boldsymbol{a_{i_l}})$ where $\{\boldsymbol{a_{i_1}}, \boldsymbol{a_{i_2}}, \boldsymbol{a_{i_3}}, \ldots, \boldsymbol{a_{i_l}}\} \subseteq \mathbf{A} = (\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_m})$**,** $\boldsymbol{a_{i_j}} \neq \boldsymbol{a_{i_k}}$**,** $\boldsymbol{j}$ **,** $\boldsymbol{k}$ **= 1...***l***,** $\boldsymbol{j} \neq \boldsymbol{k}$ and$\boldsymbol{\mu_1}$ $\in$ *M*is a relation over$\boldsymbol{D_{i_j}} \times \boldsymbol{D_{i_{j+1}}}$, $\boldsymbol{D_{i_j}}$is the domain of the attribute$\boldsymbol{a_{i_j}}$, *If*:

1. $\{\boldsymbol{a_{i_1}}, \boldsymbol{a_{i_2}}, \boldsymbol{a_{i_3}}, \ldots, \boldsymbol{a_{i_l}}\}$occur together(are non-empty) in *S%* of n instances then we call *S* the support of the rule, and
2. We denote by $\boldsymbol{R'} \subseteq \boldsymbol{R}$the set of instances where $\{\boldsymbol{a_{i_1}}, \boldsymbol{a_{i_2}}, \boldsymbol{a_{i_3}}, \ldots, \boldsymbol{a_{i_l}}\}$ occur together and relations $\boldsymbol{\Phi}\ (\boldsymbol{r_j}, \boldsymbol{a_{i_1}})\boldsymbol{\mu_1}\ \boldsymbol{\Phi}\ (\boldsymbol{r_j}, \boldsymbol{a_{i_2}})$ **,** $\boldsymbol{\Phi}\ (\boldsymbol{r_j}, \boldsymbol{a_{i_2}})\boldsymbol{\mu_2}\ \boldsymbol{\Phi}\ (\boldsymbol{r_j}, \boldsymbol{a_{i_3}})$ **,...**$\boldsymbol{\Phi}(\boldsymbol{r_j}, \boldsymbol{a_{i_{l-1}}})\boldsymbol{\mu_{l-1}}\boldsymbol{\Phi}(\boldsymbol{r_j}, \boldsymbol{a_{i_l}})$hold for each instance $\boldsymbol{r_j}$from$\boldsymbol{R'}$**:** then we call $\boldsymbol{c} = |\boldsymbol{R'}|/|\boldsymbol{R}|$ the confidence of the rule**.**

The number of attributes in the rule is called as the length of a relational association rule. Maximum length could be at most equal to 'm' number of attributes. Any Relational association rule is interesting if its support S is greater than or equal to a user specified minimum support and its confidence c is greater than or equal to a user specified minimum confidence.

Naive Bayes method[15] is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable C and a dependent feature vector x_1

Where P (c|x) is the posterior probability of class(target) for given predictor (attribute), P(c) is the prior probability of class, P(x|c) is the likelihood which is the probability of predictor given class and P(x) is the prior probability of predictor.

Advantages of Naive Bayes [15]
Probabilistic classifier based on applying Baye's theorem and is easy to implement. It is optimistic to use even when there is dependent relation between attributes.With a small amount of training data, Naive Bayes will converge more quickly to a solution than any other machine learning classifiers. Naive Bayes can work with both discrete and continuous features together in the same data set.

Methodology

In this section we introduce a novel method for detecting software modules with defects, based on relational association rule mining integrating with Naive Bayes approach, called DRAR_NB (Defect Prediction using combination of Relational Association rules and Naive Bayes Approach). The combination of association rules with Naive Bayes on Text Categorization [16] has already demonstrated that instead of words, word relation i.e. association rules from these words has derived meaningful word sets as feature set for classification. Naive Bayes is then used on derived feature set for final target class categorization. In our proposed work, we extend the concept of association rules and Naive Bayes to capture relevant relationship between metrics using binary relations and then categorizing target class 'defective or non-defective' with simple probabilistic Naive Bayes model.

Proposed Design

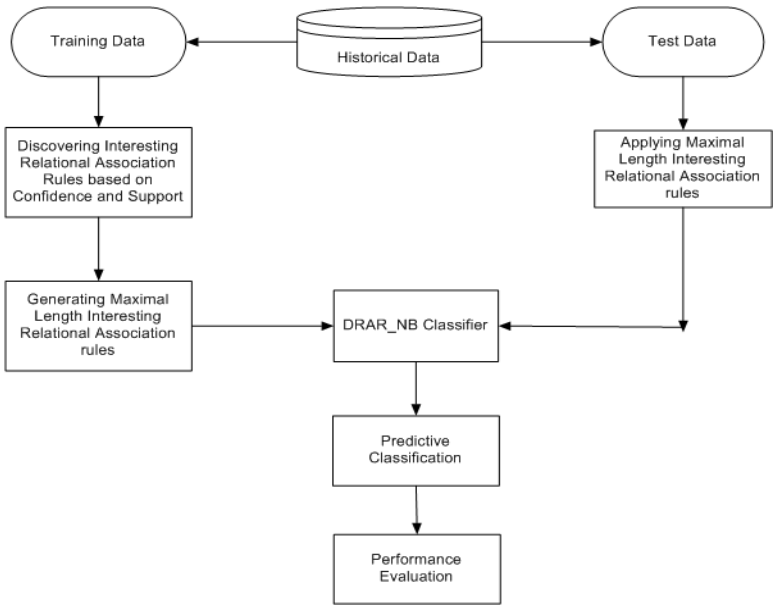

Figure 1: Flow Chart of The Proposed Design DRAR_NB

Training

As shown in Fig. 1, the historical data is divided into training dataset and test data set. The data represents software metrics that are relevant for deciding if a software module is defective or non-defective. Relational association rules are discovered in training dataset containing defective and non-defective instances using minimum thresholds of confidence and support set by the user. The possible relations defined between two software metrics are <= and >. We have considered that the relations are not defined between zero valued software metrics.

Below are detailed steps to follow in training process:

1. Determine from training dataset, using DRAR algorithm[9], the set of interesting relational association rules having minimum support and confidence.
2. Discovery of interesting rulesstarts with two attributes (called as two attribute set) and extends to generate three attribute set by considering two attribute set as input. Similarly discovery of interesting rules carried out until total number of attributes.
3. Identify Maximal length Interesting rules in each rule set by comparing first rule set with second rule set and pick those rules from first rule set that doesn't extend in second rule set.

Let us consider the java code example shown in Fig.2:

```
Public class Class_ex1 { public class Class_ex2 {
        publicstaticintattr1; private staticint attr3;
        publicstaticintattr2; private statici nt attr4;
public static void method1() public static void method4()
        { {
                attr1 = 0; Class_ex1.attr1 =0;
        method2();      Class_ex1.attr2 =0;
Class_ex1.method1();
        } }
Public static void method2() public static void method5()
        { {
                attr2 = 0; attr3 = 0;
                attr1 = 0; attr4 = 0;
        } }
Public staticvoidmethod3() public static void method6()
        { {
                attr2 = 0; attr3 = 0;
                attr1 = 0;                      method4();
                method1(); method5();
                method2();
        } }
} }
```

Figure 2: Code Example

In the above example, we considered software entity can be either an application class or a method from an application class. The software metrics considered in this example are :

1. Depth in Inheritance Tree(DIT)
2. Number of Children(NOC)
3. Fan-In(FI)
4. Fan-Out(FO)

Using the above mentioned software metrics, each software entity from the system presented in Fig.2. can be represented as a 4-dimensional vector, having as components the values of the considered metrics. The corresponding dataset is given in

Table 1: Sample Dataset

Entity	DIT	NOC	FI	FO
Class_ex1	1	0	3	1
Class_ex2	1	0	0	2
method1	1	0	2	1
method2	1	0	2	0
method3	1	0	0	2
method4	1	0	1	1
method5	1	0	1	0
method6	1	0	0	2

Table 2: Interesting Relational Association Rules

Length	Rule	Confidence
2	DIT>NOC	1
2	NOC < FI	0.625
2	NOC < FO	0.75
2	FI > FO	0.5
3	DIT > NOC < FI	0.625
3	DIT > NOC < FO	0.75
3	NOC < FI > FO	0.5
4	DIT > NOC < FI > FO	0.5

Table 3: Maximal Length Interesting Relational Association Rules

Length	Rule	Confidence
3	DIT > NOC < FO	0.75
4	DIT > NOC < FI > FO	1

DRAR algorithm has been implemented on sample dataset (**Table 1**) with minimum support 0.9 and minimum confidence 0.4. Minimum values of support and confidence are identified by examining dataset at multiple values and setting one that generates enough patterns for the study. These parameters also help in pruning search space and vary from dataset to dataset. The discovery of interesting relational rules on sample dataset for length 2, 3 and 4 based are shown in **Table 2**.The interesting relational rules of length 2, 3 and 4 are called as two rule set, three rule set and 4 rule set respectively. **Table 3** shows the maximal length interesting relational association rules for rule lengths 2, 3 and 4 respectively. The maximal length interesting rules are obtained by eliminating rules of a length '*l*' if they are subset of '*l+1'* length. In the above example, all four rules of length '2' are eliminated as they became subset of rules of length '3'. Similarly rules 'DIT > NOC < FI' and 'NOC < FI > FO' of length '3' are eliminated as they exists in next level length 'DIT > NOC < FI > FO'. Thus maximal length interesting relational association rules get reduced to 2 from total number of interesting relational association rules which is 8.

Similarly maximal length interesting relational association rules extends upto total number of attributes based on rules satisfying minimum confidence and support.

1. Determine defect likelihood for each rule by calculating how many instances in the training dataset satisfied the rule against total defective instances. The obtained value is stored as defect likelihood for the rule.
2. Determine non-defect likelihood for each rule by calculating how many instances in the training dataset satisfied the rule against total non-defective instances. The obtained value is stored as non-defect likelihood for the rule.

Classification

At the classification stage, after the training was completed if a new software entity 'e' has to be classified as defective or non-defective, then the proposed algorithm is [Fig.3]:

Algorithm DRAR_NB is

//Input: Dataset R consisting of software module to be classified as defective or non-defective.

Rule sets beginning from two rule set to maximum rule sets.

// Output: Software module classified as defective or non-defective.

// Declarations:

Defect_Lkhd= Defective likelihood of the rules

Prior_prob_Defect= Prior probability of defective class

Non_Defect_Lkhd= Non-Defective likelihood of the rules

Prior_prob_Non_Defect = Prior probability of non-defective class

Prior_prob_rules= Prior probability of rules

Step 1 : For each rule set

Step 2: **Read each rule in a rule set**

Step 3: Scan rule on to the new instance

Step 4: **If** rule is satisfied

Then

Step 5: Multiply defect likelihood of the rule with existing value [default is 1]
Step 6: Multiply non-defect likelihood of the rule with existing value [default is 1]
Step 7: **Repeat steps 3 to 6 until end of all rules in the rule set.**
Step 8: Determine prior Prior_prob_Defect as
Count of defective instances / total number of input training instances
Step 9: Determine prior Prior_prob_Non_Defect as
Count of non-defective instances / total number of input training instances.
Step 10: Determine Prior_prob_rules as
Number of rules satisfied by the new instance / total number of rules in a rule set.
Step 11: Determine posterior probability of a defective class as
(Defect_Lkhd (step 5) * Prior_prob_Defect (step 8))/ Prior_prob_rules (step 10)
Step 12: Determine posterior probability of a non-defective class as
(Non_Defect_Lkhd(step 6) * Prior_prob_Non_Defect(step 9))/ Prior_prob_rules (step 10)
Step 13: If (posterior probability of defective class > posterior probability of non-defective class)
Then Increment score_positive by 1
Otherwise
Increment score_negative by 1.
Step 14: Repeat steps 2 to 13 until end of all rule sets.
Step 15: If score_positive>score_negative then declare new instance as defective otherwise declare new instance as non-defective.

Figure 3: DRAR_NB Algorithm

The steps 11 and 12are based on Bayes theorem specified in section 2.2. Maximal length Relational association rules are used to determine posterior probabilities of a target class which is defective or non-defective.

Measurement Technique

The prediction model used in the experiment outputs probability of defect proneness for each module that has been tested. To classify module as defective, one can use minimum threshold and minimum support set by the user and different choices of thresholds will give varying rates of false positives and false negatives. For a binary classification task, the confusion matrix for the two possible outcomes (positive and negative) is computed. The confusion matrix [17] shown in **Table 4** consists of
TP - The number of actual positive instances predicted as positive.
FP - The number of actual negative instances predicted as positive.
TN - The number of actual negative instances predicted as negative.
FN - The number of actual positive instances predicted as negative.

Table 4: Confusion Matrix

	Actual		
Predicted		Defect-Prone	Non-Defect-Prone
	Defect-Prone	True Positive	False Positive
	Non-Defect-Prone	False Negative	True Negative

The following evaluation measures are used for defect prediction in this paper:

Accuracy of the model is the proportion of correct classifications (true positives and true negatives) from overall number of cases.

$$\text{Accuracy} = (TP + TN)/ (TP + FP + TN + FN) \quad (2)$$

Ma et al. [18] observed that in a highly class imbalanced data set with very few defective modules, accuracy is not a useful performance measure because accuracy shown to be high even with poor detection of positive cases.

Precision measures proportion of correct positive classifications (true positives) from cases that are predicted as positive.

$$\text{Precision} = TP/ (TP + FP) \quad (3)$$

Recall measures proportion of correct positive classifications (true positives) from cases that are actually positive.

$$\text{Recall} = TP/ (TP + FN) \quad (4)$$

F- Measure is a harmonic mean of precision and recall and is defined as:

$$\text{F-measure} = (2 * \text{Recall} * \text{Precision})/ (\text{Recall} + \text{Precision}) \quad (5)$$

Experimental Evaluation

This section aims at experimentally evaluating proposed method DRAR_NB for defect prediction using relational association rules and provides a comparison with existing classifiers – Naive Bayes, Bagging and One R.

Data Collection

One set of datasets chosen in the experiment consists of object oriented product metrics describing the size and design complexity of software module. The proposed model is compared with other classifiers to determine how the efficiency varies in predicting software entity as defective or non-defective. The projects studied are open source projects from Eclipse- Bug Prediction Dataset ("http://bug.inf.usi.ch/ download.php").

Table 5 shows latest version of eclipse bug prediction datasets from different software systems used in the experiment. **Table 6** shows product based object oriented metrics that are used in the experiment.

Table 5: Eclipse Open Source Projects

Open Source Project	Description	#Classes	% Defective
Lucene	Text Search Engine Library	691	9.3
Eclipse_JDT	Provides the tool plug-ins that	997	21

Table 6: Product Metrics – Object Oriented

Metrics	**Description**
CBO	Coupling between object class
DIT	Depth of inheritance tree
FI	Fan IN-Number of classes that reference the class
FO	Fan Out-Number of classes referenced by the class
LCOM	Lack of cohesion in methods
NOC	Number of children
NOA	Number of attributes
NOAI	Measure of Attributes Inherited
NLOC	Number of lines of code
NOM	Number of Methods
NOMI	Number of Methods inherited
NOPRA	Number of private attributes
NOPRM	Number of private methods
NOPA	Number of public attributes
NOPM	Number of public methods
RFC	Response for class
WMC	Weighted methods per class

Another set of datasets used in the experiment is based on static McCabe and Halstead static code metrics which is made publicly available, often classed NASA Datasets. These are also referred as PROMISE Repository [3] and is freely available for anyone who wanted to build or test defect prediction models. **Table 7** shows description of PROMISE repository datasets – CM1 and PC1 that are studied in our experiment. **Table 8** shows static code attributes from PROMISE Repository and data is available in the link "http://promise.site.uottawa.ca/SERepository

Table 7: PROMISE Repository

Open Source Project	Description	# Modules	% Defective
CM1	NASA spacecraft instrument written in "C"	439	9.5
PC1	Flight software for earth orbiting satellite written in C.	947	15.2

Table 8: Static Code Metrics

Metrics	**Description**
LOC	McCabe's lines of code in software module
v(g)	Measure McCabe Cyclomatic Complexity
ev(g)	McCabe Essential Complexity
iv(g)	McCabe Design Complexity
N(N1 + N2)	Halstead Total number of Operators and Operands
V	Halstead Volume
L	Halstead Program length
D	Halstead Measure difficulty
I	Halstead Measure Intelligence
E	Halstead Measure Effort
B	Halstead Error Estimate
T	Halstead Time Estimator
Locode	Halstead's Number of lines in software module
Locomment	Halstead's Number of comments
Loblank	Halstead's Number of blank lines
Locodeand comment	Number of codes and comments
uniq_op	Unique Operators
uniq_opnd	Unique Operands
total_op	Total Operators
total_opnd	Total Operands
Branchcount	Number of branch count
Defects	Class describing software module is defective or not

Result Analysis

The implementation work of proposed classifier DRAR_NB is compared with Naive Bayes, one R classifier [19] and Bagging classifier[20].To run the experiments stratified ten-fold cross-validation was used. Cross-validation is a standard evaluation

measure for calculating error rate on data in machine learning and performance measures presented in section 3.4are used.

Observations have shown that threshold value of confidence and support plays a key role in overall prediction process. Quality of results varies based on high or low confidence value set.

Fig.4 shows comparison between DRAR_NB and other classifiers on datasets containing object oriented product based metrics. The graphs reflect data related to defective class. Results for all the three datasets namely LUCENE (Fig.4a)and ECLIPSE_JDT (Fig.4b) are considered for evaluation. The minimum support value has been set to '1' representing that none of the attributes are left empty in any instance of '*R*' and hence total '*n*' instances in the dataset will contribute for defect prediction. The minimum threshold value of confidence varies from dataset to dataset. Range of values studied for setting minimum confidence in three datasets are 0.3 to 0.9 and among them obtained balanced results at 0.4 for datasets Lucene and Eclipse_ JDT. Balanced results doesn't consider results that are turned out completely to either True positive (TP Rate = 1) or True negative (TN Rate = 1). Minimum value 0.4 also ensures search space not to be pruned to a large extent which may result in lack of quality on prediction model. The comparison results of classification based on proposed work DRAR_NB has shown significant improvement over other classifiers in terms of performance measures of Recall(equation4)

Fig.5shows comparison between DRAR_NB and other classifiers on datasets containing static code metrics. The graphs reflect data related to defective class. Results for all the three datasets namely CM1 (Fig.5a) and PC1(Fig. 5b) are considered for evaluation. The minimum support value has been set to '1' and confidence is set to 0.2 for the two datasets. The comparison results reflect DRAR_NB has performed significant improvement in terms of Recall which determines the True positive rate. The results obtained are similar to those of object oriented product based metrics.

Table 9 shows comparative results for all open source projects considered for evaluation, the Accuracy(equation 2), Precision(equation 3), Recall(equation 4) and F-measures(equation 5) obtained for DRAR_NB and classifiers – Naive Bayes, Bagging and One R.. Data reflects for defective class. As confidence, support and Length are not applicable to all other classifiers, they are marked with NA(Not Applicable). For each project, the best result through DRAR_NB is marked with bold characters. Results clearly show that DRAR_NB outperforms all other classifiers in identifying defective modules on all datasets. With respect to True Positive rate (Recall), Lucene dataset has achieved more than double times of better improvement through DRAR_NB when compared to Naive Bayes and Bagging. OneR classifier has only 7.8 % of True positive rate compared to 67.2 % of DRAR_NB. Eclipse_JDT obtained double the performance over Naive Bayes and OneR, 8% of improvement over Bagging.

The datasets CM1 and PC1 also obtained double the performance over Naive Bayes and OneRwith respect to True Positive rate i.e. recall. Bagging has '0' true positive rate for CM1 and performed poor in PC1 with precision as '1' and F-Measure as '0.2'.

Table 9: Comparative Results (In **%**).

Open source Project	classifiers	confidence	Support	Length	Accuracy	True Positve (Recall)	Precision	F-Measure
Lucene								
	DRAR_NB	0.4	1	Any	73.5	**67.2**	24.2	32
	Naïve Bayes	NA	NA	NA	85.8	25	21	24.6
	Bagging	NA	NA	NA	92.7	23.4	93.8	37.5
	OneR	NA	NA	NA	91.1	7.8	71.4	14.1
Eclipse_JDT								
	DRAR_NB	0.4	1	Any	73.7	**72.8**	42.1	53.3
	Naive Bayes	NA	NA	NA	82.8	34.5	66.4	45.4
	Bagging	NA	NA	NA	92	67.5	92.1	77.9
	OneR	NA	NA	NA	85.2	43.2	74.8	54.8
CM1								
	DRAR_NB	0.2	1	Any	56	**89.1**	18.1	30.1
	Naive Bayes	NA	NA	NA	84.2	32.6	28.3	30.3
	Bagging	NA	NA	NA	89.5	0	0	0
	OneR	NA	NA	NA	91.1	28.3	68.4	40
PC1								
	DRAR_NB	0.2	1	Any	61	**79.4**	12.6	21.7
	Naive Bayes	NA	NA	NA	89.9	36.5	29.5	32.6
	Bagging	NA	NA	NA	94	11.1	1	0.2
	OneR	NA	NA	NA	94	20.6	68.4	31.7

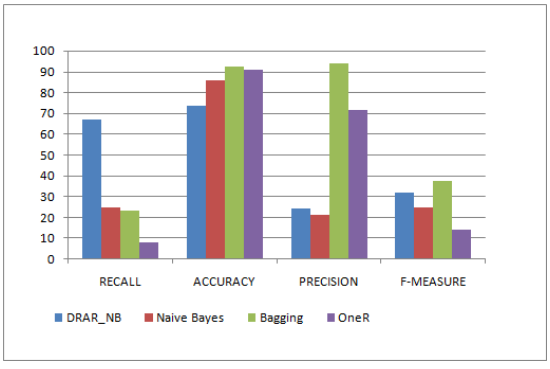

Figure 4a: Dataset – LUCENE, conf = 0.4

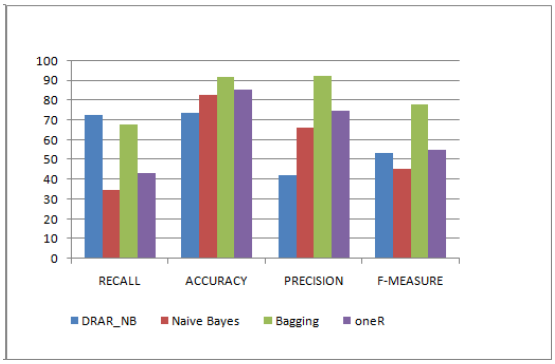

Figure 4b: Dataset – ECLIPSE_JDT, conf = 0.4

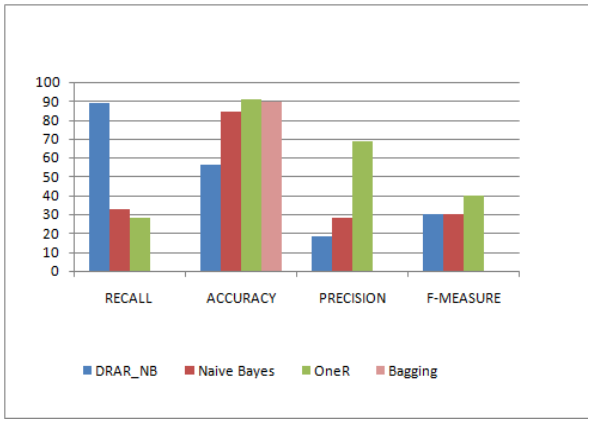

Figure 5a: Dataset – CM1, conf = 0.2

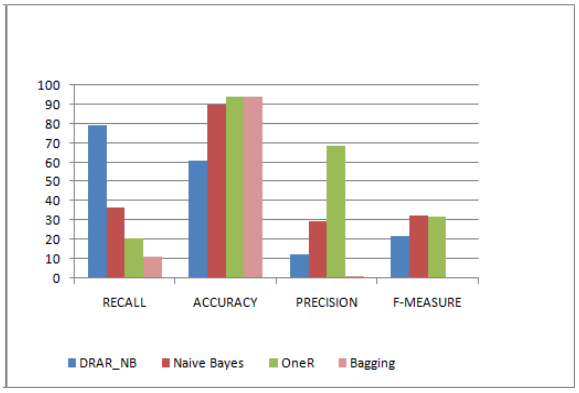

Figure 5b: Dataset – PC1, conf = 0.2

Conclusions and Future Work

We have introduced in this paper a classification model based on discovering relational association rules and using probabilistic approach to detect software module

that is likely to be defective in the software systems. Experiments were conducted in order to predict defective software modules, and the obtained results have shown that our classifier is better than existing classifiers that are already applied for software defect prediction. The datasets considered for experiment are eclipse and PROMISE open source projects – Lucene, Eclipse_JDT, CM1 and PC1 with latest version.

Future work in product metrics based predictive classification of software using relational association rules and Naive Bayes will be made to include process metrics at different stages of SDLC and determine the improvement effectiveness. We will investigate of extending the proposed model DRAR_NB by combining it with other machine learning based predictive models. We will also analyze how the length of the rules and the confidence of the relational association rules discovered in the training data may influence the accuracy of the classification task.

References

[1] Marco D'Ambros, Michele Lanza, and RomainRobbes. "*Evaluating defect prediction approaches: a benchmark and an extensive comparison*". Journal of Empirical Software Engineering.To appear.

[2] Marco D'Ambros, Michele Lanza, and RomainRobbes. "*An extensive comparison of bug prediction approaches*". In 7th Working Conference on Mining Software Repositories(MSR), pages 31–41, 2010.

[3] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, June 2012 <http://promisedata.googlecode.com>.

[4] B. Minaei-Bidgoli, R. Barmaki, M. Nasiri: "*Mining numerical association rules via multi-objective genetic algorithms*", Inform. Sci. 233 (2013) 15–24.

[5] A.Marcus, J.I. Maletic, and K-I. Lin, "*Ordinal Association Rules for Error Identification in Data Sets*", in Proceedings of Tenth International Conference on Information and Knowledge Management (CIKM2001), ACM, New York, NY, USA, pp. 589–591, Apr 2001.

[6] G. Serban, A. Campan, I.G. Czibula, "*A programming interface for finding relational association rules*", Int. J. Comput., Commun. Control I (S.) (2006) 439–444.

[7] E. Baralis, L. Cagliero, T. Cerquitelli, P. Garza, "*Generalized association rule mining with constraints*", Inform. Sci. 194 (2012) 68–84.

[8] B. Liu, W. Hsu, Y. Ma, "*Integrating classification and association rule mining*", in: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD), 1998, pp. 80–86.

[9] Gabriela Czibula , Zsuzsanna Marian, IstvanGergelyCzibula, *"Software defect prediction using relational association rule mining",* Information Sciences vol no.264, pp.260–278, Jan 2014.

[10] T. Menzies, J. Greenwald, A. Frank, "*Data mining static code attributes to learn defect predictors*", IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.

[11] M Kamruzzaman and Chowdhury Mofizur Rahman, "*Text Categorization using Association Rule and Naive Bayes Classifier*", Asian Journal of Information Technology, Vol. 3, No. 9, pp 657-665, Sep. 2004.

[12] Pradeep Singh, K.D.Chaudhary, ShrishVerma, "*An Investigation of the Relationships between Software Metrics and Defects*", International Journal of Computer Applications (0975 – 8887),Volume 28– No.8, August 2011.

[13] DanijelRadjenovic, MarjanHericko, Richard Torkar and Ales Zivkovic, "*Software fault prediction metrics: A systematic literature review*", Elsevier, Information and software Technology, pp. 1397–1418, Mar 2013.

[14] Paul Jenkins and P. Radivojac."*Naive Bayes Clasifiers*".Machine Learning, CSCI-B555,45(1), 2001, pp 5-32.

[15] S.Stehman, "*Selecting and interpreting measures of thematic classification accuracy*", Rem. Sens. Environ. 62 (1) (1997) 77–89.

[16] Y. Ma and B. Cukic."*Adequate and precise evaluation of quality models in software engineering studies*". In Proceedings of the 29th ICSE Workshops, ICSEW '07,pages 68–, Washington, DC, USA, 2007. IEEE Computer Society.

[17] V.U.B. Challagulla, F.B. Bastani, I.-L. Yen, R.A. Paul, "*Empirical assessment of machine learning based software defect prediction techniques"*, in: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 263–270.

[18] A.S. Haghighi, M.A. Dezfuli, S. Fakhrahmad, "*Applying mining schemes to software fault prediction: a proposed approach aimed at testcost reduction*",in: Proceedings of the World Congress on Engineering 2012, WCE 2012, vol. I, IEEE Computer Society, Washington, DC, USA, 2012, pp. 1–5.

[19] Tina. R. Patil and S.S. Sherekar, "*Performance Analysis of Naive Bayes and J48 Classification Algorithm for Data Classification*". In International Journal of Computer science and Applications, vol.6, No.2, Apr 2013.