ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

DPRAR: Defect Prediction Using Relational Associational Rules

S. Sivaprasad [1], V.Jyothsna[2]

sriram.sivaprasad@gmail.com jyothsna1684@gmail.com

[1] Department of IT, Sree Vidyanikethan Engineering College, Tirupathi, India.
[2] Department of IT, Sree Vidyanikethan Engineering College, Tirupathi, India

Abstract: Software defect prediction focus to automatically discover defective software modules, in order to help software testers focus their time and effort on those modules which are likely to contain faults. Many machine learning algorithms have been used for this classification task. A novel software defect prediction method, called DPRAR, which uses relational association rules to classify modules for predicting whether a software module is or is not defective. Relational association rules are an expansion of ordinal association rules, which are a particular type of association rules that describe numerical orderings between attributes that usually occur over a dataset. DPRAR analyses how close the module is for faulty instances and the non-faulty instances. An experimental evaluation of the proposed model on the open source NASA datasets, as well as an evaluation to similar existing approaches is provided. The obtained results show that DPRAR classifier over performs, for most of the considered evaluation measures which conforms potential than the existing machine learning based techniques for defect prediction.

Keywords – Software defect prediction, Software Metrics, Relational Associational rules, Data mining.

1. INTRODUCTION

Software quality -is considered of great importance in the software engineering field. Thus, building software of high quality is very expensive task. Consequently in order to increase the efficiency and usefulness of quality declaration and testing, software defect prediction is used to find out defect-prone modules in a forthcoming version of a software system and help assign the effort on those modules.

Association rule mining–finding attribute value conditions that occur repeatedly together in a dataset. Ordinal association rules are described as an exact type of association rules. Particular a set of records represented as set of attributes, the ordinal association rules identify ordinal relationships between verification attributes that seize for a particular percentage of the records. But, in real world datasets, attributes with special domains and relationships between them, other than ordinal, do really exist. Sometimes, ordinal association rules are not strong sufficient to describe data regularities. Accordingly, relational association rules were introduced in [24] in order to be able to capture different kinds of relationships between record attributes.

The paper invests a novel classification model for the problem of defect prediction, based on the plan of discovering relational association rules within a dataset. To find whether a software module is defective or not is of main role for the maintenance and development of software systems, as developers are always concerned in improving the software quality. The results found by evaluating the classification model proposed in this paper do verify that applying relational association rule mining for defect detection is capable and indicate the potential of our suggestion. Moreover, the use of relational association rules in discovery software entities as being defective or not, is a new approach.

2. MOTIVATION

Identifying the software entities like classes, modules, methods, and functions that are defective is of main importance as it facilitates progress software development and maintenance. Although many models have been proposed in the software defect prediction literature, this problem has not been completely solved so for researchers are still focus on developing more accurate defect predictors. Recent results show that researchers should focus on improving the quality of the data in order to overcome the limits of the existing software prediction model.

Relational association rules-- [24] were introduced as an expansion to association rules, in order to be capable to discover different kinds of relations or correlations that exist between data in large datasets. A software module as of a software system can be characterized by a set of significant software metrics values. These software metrics may be applicable for deciding if a module is defective or not. Accordingly, a software module can be visualized as a high dimensional vector and the complete software system can be represented as a dataset consisting of the high dimensional vectors equivalent to the system's software modules. Within this dataset, where the records are the (high dimensional) software modules and the features are the software measurements, significant information can be extracted from the software metrics values characterizing the modules. Dissimilar types of relationships between the numerical feature values can be distinct and a relational association mining process can be performed on the dataset representing the software system. Such a mining process can offer interesting patterns that would be useful for predicting if a software module is defective or not.

Our suggestion, we have started from the perception that when deciding if a software entity is defective, relational association rules may be

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

efficient, as relationships between the software metrics values characterizing the software entities may be applicable. These relationships may communicate quantitative information that may exist in the vector characterizing a software entity. It is possible that these relationships could provide important information regarding defective entities. While many methods for software defect prediction do exist within the software engineering text, recent researches are still passed out for proposing more accurate software defect predictors and for overcoming the drawbacks and limitations of the presented models. Relational association rule mining has not been applied so far for predicting if a software entity is defective or not. Thus in this paper at developing a novel method based on relational association rules, whose usefulness will be shown through the experimental results.

3. DEFECT PREDICTION

In that the aim of presenting the problem of defect prediction and its importance, as well as presented machine learning based approaches for solving the consider problem.

3.1. Problem statement and relevance

The automated evaluation of software, in conditions of defect prediction, is of main importance to the software engineering community and researchers are endlessly focusing on building accurate and responsible predictors using legacy data. In order to bring high quality software on time, software project managers, quality managers and software developers require to always focusing on detect and correct software defects at all stages of the development process. Defects such as faults or bugs represent a main issue in planning the on-time-delivery and the quality of the released product mainly during the maintenance and development of a software project. The product quality is highly connected with the defects. So high importance for developers and project managers to declare a software development with as only some errors as possible. In this direction, software defect prediction helps in finding, tracking and resolving software anomaly that might have an achieve on human security and lives, particularly in safety dangerous systems. Defect prediction also allows changes to be made previously in the software life-cycle, assuring this way a lesser software cost and improving customer satisfaction .A software defect shows any error or deficiency in a software object or a software process and a major focus is on **finding** those defects that manipulate project or product performance. Several software defect predictors use software metrics to measure the software quality in order to predict software defects. Consequently, software defect prediction is the duty of **classifying** software modules into the fault-prone and the non-fault-prone ones by using metric-based classification. Mainly defect prediction techniques used in planning are based on past data, hence rely on supervised classification.

3.2 RELATED WORK

Although association rules are generally used in an unsupervised learning situation, different extensions for classification are presented in the future. One of them is the CBA method, presented in, where class association rules, i.e. association rules whose following is a class label, are mined. Presents an extension of this method, called CBA2, where rules predicting various classes can have a various minimum support, to solve the data imbalance problem.

Ma et al. use in the CBA2 method for finding defective software modules. Experiments on the NASA datasets are performed and comparisons with additional rule based classification methods are given. The authors also inspect whether the association rule sets that were provided based on the data from one software project can be used to calculate defective software modules in other, similar software projects.

Rodriguez et al. introduce in a Subgroup Discovery (SD) algorithm named EDER-SD means Evolutionary Decision Rules for Subgroup Discovery i.e based on evolutionary estimation and generates rules describing only fault-prone modules. The results performed on datasets from NASA showed that the EDER-SD algorithm shows well in the majority cases when compared to three other well related SD algorithms. Moreover rule-based methods, many different machine learning algorithms have been useful to the problem of defect prediction. Menzies et al., in which they compute the Naive Bayes classifier (NBC), OneR and J48. They have also experimented with different filters and finished that, on average, logarithmic filtering and Naive Bayes formed the best results on the 8 used NASA datasets. Presents a literature study about a pair of methods that are often used for defect prediction: small equations, machine learning methods and defect density prediction models. Likewise, Kaur et al. in shortly current clustering, categorization and association rule mining as software defect prediction methods.

Challagulla et al. evaluate some special predictor models on four dissimilar real-time software defect data sets that were in use from the NASA repository. The experimental results have shown that a grouping of 1-rule classification and Instance-based Learning using regularity based Subset Evaluation technique provides a relatively enhanced reliability in accuracy prediction compared to other models.

Haghighi et al. gives a comparative analysis of 37 dissimilar classifiers in fault detection systems and use the NASA datasets for performing an experiment. The results show that, on average, the Bagging classifier achieved a higher performance and accuracy evaluate to the others.

A dissimilarity-based semi-supervised learning method, called ROCUS, is presented by Jiang et al. i. They use semi supervised learning because in defect prediction there is a limited degree of labeled data, whereas gathering unlabeled data is easy. They are also use under-sampling to solve the class variation problem. ROCUS is calculated on eight datasets from the NASA repository and the results are compared to extra methods which are either semi-supervised methods that do not take into consideration the extreme data, or class-imbalance learning methods that cannot utilize unlabeled data.

4. RELATIONAL ASSOCIATION RULES: BACKGROUND

In relational association rule mining the goal is to find several relationships between the attributes that be liable to hold over a large percentage of records, in binary classification problem, then attribute A is in relation with attribute B for a large number of positive instances, next a record in which attribute A is not in relative with attribute B may be a negative instance, possibly will not mean very much if only one rule including B is not satisfied, but it increases the possibility that the instance in query belongs to the unconstructive class if many such rules are broken. The following will briefly evaluate the theory of relational association rules, as well as the method for identifying the appropriate relational association rules that hold within a dataset. Let R={r_1,r_2,,…r_n}be a set of instances (entities or records in the relational model),anywhere every instance is discriminate by a list of m attributes, (a_1,. . . . a_m). We denote by Þ (r_j, a_j) the cost of attribute a_i for the instance r_j . Every attribute a_i takes values from a domain D_i, which contains the empty value denoted by ε. Between two domains D_i and D_j relations can be defined (not necessarily ordinal relations), such as: less or equal (≤) (or), equal (=), greater or equal (≥), etc. We represent by M the set of all possible relations that can be defined on D_i x D_j. A relational association rule is an expression (a_{i1} , a_{i2} , a_{i3} , . . . a_{il}) => (a_{i1} μ_1 a_{i2} μ_2 a_{i3}…. μ_{l-1} a_{il}), where {a_{i1},a_{i2},a_{i3},...,a_{il} }A={a_1,….,a_m}, ai_j ≠ai_k, j, k=1…..j ≠k and μ € M is a relation over Di_j*Di_{j+1},Di_j is the domain of the attribute ai_j .if:

a)ai_1,ai_2,ai_3,….ai_l occur concurrently (are non-empty) in s% of the n instances, then we represent s the support of the rule, and

b)we denote by $\dot{R}$⊆ R the set of instances where ai_1,ai_2,ai_3,….ai_l occur together and the relations Þ(r_j,ai_1) μ_1 Þ(r_j,ai_2), Þ(r_j,ai_2) μ_2 Þ(r_j,ai_3)….. Þ(r_j,ai_{l-1}) μ_{l-1} Þ(r_j,ai_l) hold for each instance r_j from R˙;then we call c=|R˙|/|R| the confidence of the rule.

We call the size of a relational association rule the number of attributes in the rule. The size of a relational association rule can be at mainly equivalent to the number of the attributes relating the data. The users usually need to discover interesting relational association rules that embrace in a dataset, they are concerned in relational rules which embrace in a minimum number of instances i.e. , rules with support at least s_{min}, and confidence at smallest amount c_{min} (s_{min} and c_{min} are user-provided thresholds).

We defined a relational association rule in R interesting if it's maintain s is greater than or equal to(≥) a user-specified minimum support, s_{min}, and its confidence c is larger than or equivalent to a user-specified minimum confidence, c_{min}. We have previous introduced in [6] an A-Priori [1] like algorithm, called DOAR (Discovery of Ordinal Association Rules), that capably finds all ordinal association rules (i.e. relational association rules in which the relations are ordinal) of any span, that hold over a dataset.

In the following a lengthy explanation of the discovering interesting ordinal association rules will be given [6]. The method of discovering interesting ordinal association rules in a dataset will be complete in our approach towards identifies relational association rules. DOAR algorithm identifies ordinal association rules are using an iterative process that consists in length-level invention of candidate rules, followed by the authentication of the candidates for minimum support and confidence compliance. DOAR algorithm performs various passes over the dataset R. First pass, it calculates the support and confidence of the 2-length rules and determines which of them are motivating, (i.e. verify the minimum support and confidence requirement). Each subsequent pass over the data contains two phases. The first phase starts with a beginning set of (k-1)-length (k ≥ 3) interesting rules, found in the before pass. This set is used to generate new possible k-length interesting rules, called candidate rules.

The candidate building process is a key element of the DOAR algorithm. In the second phase, a scan in excess of the R data is performed in order to calculate the actual support and confidence of the aspirant rules. At the conclusion of this step, the algorithm maintains the rules that are deemed interesting (have minimum support and assure the confidence requirements), will be used in the next iteration. The process ends when no new interesting rules were found in the newest iteration.

The DOAR algorithm considerably prunes the exponential search space of all achievable interesting ordinal association rules, suitable to the candidate generation method. The candidate generation restricts the look for those regions of the find space where it is feasible that motivating rules may exist, pruning out all the regions where it is not possible to find any interesting rules. The investigate space reduction depends on the data being analyzed. The better the number of interesting rules in the dataset is, the larger the size of the candidate sets will be. We have established that the proposed algorithm is exact and complete. We have shown that it capably explores the investigate space of the possible rules. Additional the DOAR algorithm and its notional validation are given in [6].

The DOAR algorithm is total in our approach towards the DRAR algorithm (Discovery of Relational Association Rules) for discovery interesting relational association rules, i.e. association rules which are

capable to capture different types of relationships between record attributes. Our present implementation provides two functionalities:
Finds all interesting relational association rules of any size.
Finds all maximal interesting relational association rules of any size, i.e. if an attractive rule r of a certain length can be extended with one attribute and it leftovers interesting , only the extended rule is kept.

5. DISCOVERY OF ORDINAL ASSOCIATION RULES -DOAR

The new algorithm, called as DOAR
(Discovery of Ordinal Association Rules), to determine all the *interesting* ordinal rules of *any length* in a data set. Algorithm is forced by the Apriori algorithm for determining Boolean association rules in a transactional data set. Particularly, rules identification is an iterative procedure that consists in length-level generation of candidate rules, followed by the authentication of the candidates for minimum support and confidence fulfillment.
The DOAR algorithm performs several passes over the data set *R*. The first pass, it evaluates the support and assurance of the 2-length rules and determines which of them are motivating, i.e., validate minimum support and confidence requirement. In each subsequent pass over the data, we found with a seed set of interesting rules, found in the previous pass. We use this set to produce new achievable interesting rules, called *candidate rules*, and we estimate the real support and confidence of these candidates in the scan of the data, by the end of this step, we maintain the rules that are deemed attractive, which will be used in the after iteration. The process stops when no new attractive rules were found in the most recent iteration. The remainder of this section explains in details and formalizes the most important steps of the algorithm, discusses the complexity of the algorithm.

The DOAR Algorithm

DOAR algorithm contains following steps:
• *Ck* is the set of *k*-length candidate rules ,a *k*-length Candidate rule is a series of incomplete orderings between *k* attributes, $2 \leq k \leq m$;
• *Lk* is the place of the *k*-length interesting (i.e., support And confidence larger than or equivalent with *min_s* and *min_c*, correspondingly) ordinal rules found by DOAR. It will be proved that *Lk* is equivalent to the set of all *k*-length interesting ordinal association rules presented in data, $2 \leq k \leq m$. The DOAR algorithm starts by generating *C2*, calculating the support and confidence for each candidate rule in *C2*, and formative *L2*. Intended for the set $M = \{\leq, =, \geq\}$ of partial ordering relations between attributes and the binary candidate rules (*C2*) are generated. The *L2* set is determined by a scan of the data and is the initial point of the following steps in the iterative process engaged by DOAR.
Every iteration consists of two phases:
• Initial, DOAR generates the *k*-length candidate rules Set, *Ck* ($k \geq 3$), by means of the set of (*k-1*)-length interesting rules, *Lk-1*. The candidate creation process is the input element of the algorithm.
• In that case, a scan of the *R* data set is perform, as computing the support and the confidence of each candidate rule in *Ck*. The candidates in *Ck* that have minimum support and convince the confidence requirements are attractive ordinal association rules and consequently are integrated in L_k.
At every iteration, candidates are generated by the *GenCandidates* function (see [6]). The *GenCandidates* task has dispute the L_{k-1} set of (*k-1*)-length interesting rules and precedes C_k, a superset of the set of the exciting *k*-length rules. The elements of C_k are sequences of incomplete orderings between *k* attributes, defined as candidate *k*-length rules. *GenCandidates* performs the candidates in C_k in the following manner. Every unordered couple of rules ($rule_1$, $rule_2$), $rule_1$, $rule_2 \in L_{k-1}$, which satisfy one of the formats below, is combined into a candidate rule *c*. To simplify the information in these formulas, we only write from each rule the incomplete orderings sequence (i.e., the right hand side of the rule).
.

6. METHODOLOGY

In that we introduce a novel supervised technique for detecting software entities with the defects, based on Relational Association Rule Mining, called as DPRAR (Defect Prediction using Relational Association Rules).

THEORETICAL MODEL

The main idea of this approach is to describe the entities (classes, modules, methods, functions) of a software structure as a multidimensional vector and whose elements are the values of different software metrics applied to the specified entity. In order to give a formal description, we regard as that a software system S is a group of components (i.e entities) $S=\{s_1,s_2,\ldots,s_n\}$. It is well known that software metrics are broadly used to measure the software quality. As we mean to identifying software entities having defects, we regard as a set of software metrics (the aspect set in a vector space model based approach) applicable for deciding if a software entity is or not defective. Therefore, we have a feature set of software metrics $SM=\{sm_1,sm_2,\ldots,sm_k\}$ and thus every entity si €S from the software system can be indicated as a k-dimensional vector, having as components the values of the software metrics from SM, $s_i=(S_{i1},S_{i2},\ldots S_{ik})$ (s_{ij} represents the value of the software metric sm_j applied to the software entity S_i).

7. OUR APPROACH

In that mostly focus on is a binary classification problem. There are two achievable classes, represented in the following by ‘‘+’’ and ‘‘-’’. By ‘‘+’’ we indicate the class corresponding to software entities that having defects, also the entities that be in the right place to the

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

''+'' class will be referred to as positive instances or defects, by means of ''-'' we denote the class equivalent to software entities that are not defective, and the entities that be in the right place to the ''-'' class will be referred to as unconstructive instances or non-defects.

The main task of our approach is the subsequent. In a supervised learning scenario for predicting faulty software entities, two sets contain positive and negative instances are given. The vector space model, these datasets consist of k-dimensional software entities as of a software system. These are the sets will be used for training the classifier. For the period of training, the DRAR algorithm will be used. We discover in the training datasets all the attractive relational rules, with value to the user-provided support and confidence thresholds. Once the training was completed, when a new occurrence (software entity) has to be classified (as ''+'' or ''-''), we explanation as follows. Considering the rules exposed in training in the set of positive and negative instances, contains two scores, $score_+$ (indicating the similarity degree of the instance to the positive class) and $score_-$ (indicating the parallel degree of the instance to the negative class), and is computed. If $score_+$ is greater than $score_-$, then the query occurrence will be classified as a positive instance, or else it will be classified as a negative instance.

The procedure takes place in two phases that reveal the principles of a supervised learning algorithm called training and testing. For the duration of training a classification model will be built, in testing, the model built during training will be applied for classifying an hidden instance. The same as mentioned above, we consider for training two datasets: DS_+ consisting of positive k-dimensional instances (software entities that are defective) and DS_- consisting of negative k-dimensional instances (software instances are not defective). These datasets be use in the training step of the DPRAR classifier and a classification model consisting of the exposed relational association rules is constructing. By the classification time, as a new instance (software entity) e has to be classified, the model academic during the training step will be used for computing the parallel degrees of the instance e to the positive and negative classes, i.e. to expect if the query instance is or not defective. In favor of classifying software entity is defective or not, the following steps will be performed:

1. Data pre-processing.
2. Training/building the DPRAR classifier.
3. Testing and classification.

These are following will describe these steps.

7.1 DATA PRE- PROCESSING

In that, the training data are scaled to [0,1] and a numerical analysis is passed out on the training datasets DS_+ and DS_- in organize to discover a separation of features that are correlated with the target output. The numerical analysis on the features is performed in arrange to reduce the dimensionality of the input data, through eliminating features which do not broadly influence the output value.

To find out the dependencies between features and the objective output, the Spearman's rank correlation coefficient [24] is used. A Spearman correlation of 0 between two variables are X and Y indicates that there is no movement for Y to also increase or decrease while X increases. The Spearman correlation of 1 or -1 result when the two variables should be compared is monotonically related, still if their relationship is not linear. At the numerical analysis step we take away from the feature set those features that have no major influence on the target output, i.e. are to some amount correlated with it.

In order to choose which features to eliminate, explanation as follows. For every feature (software metric) sm_i €SM we compute the Spearman correlation (cor (sm_i, target)) among the feature and the target output (defect or non-defect). Let us specify by m the average is a value and stdeV the standard deviation of the correlations among all features and the target output. We regard as that a feature sm_i is slightly interrelated with the target classification output and will be eliminated from the feature set if the total value of the correlation is less than m - stdeV, i.e. abs (cor (smi, target)) < m – stdeV. The dataset pre-processed likewise we can be used for construction the relational association rule based classification model.

7.2 TRAINING

In training, we describe a set of relations between the feature values that are used in the relational association rule mining process. Exactly, we are focusing on identify relations between two software metrics , dealings that would be relevant for deciding but a software entity is defective or not, and accordingly would be useful in the mining process. After the relations are distinct, the interesting relational association rules are revealed in the training datasets. Exactly, the training contains of the following steps:

- By the DRAR algorithm we can determine DS_+, the set of RAR_+ relational association rules consists of minimum support and confidence.
- By using DRAR algorithm we can calculate DS_- the set RAR_- of relational association rules consists of minimum support and confidence.
- For each rule r from the sets RAR_+ and RAR- strong-minded as indicated above, the support (denoted by supp(r)) and the confidence (denoted by conf (r)) of the rule are computed. We indicate in the following by ratio(r) the value obtained by separating the confidence of the rule to its support, i.e.

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

$$\text{Ratio (r)} = \frac{conf(r)}{supp(r)}.$$

7.3 CLASSIFICATION

In classification phase, once the training was completed and the DPRAR was built, once a new software entity e has to be classified, we calculate the scores score$_+$ (e) (the similarity of e to the positive class) and score$_-$ (e) (the similarity of e to the Negative class). In calculating these scores we started based on the perception that the similarity of an instance e to the positive class, for example, is very possible to be influenced by the rules from RAR+ that are verified in the entity e but also by the rules from RAR_ that are not confirmed in the entity e. In this way, score+ measures simply how ‘‘close’’ the entity is to the positive instances, even though how ‘‘far’’ it is from the negative ones.

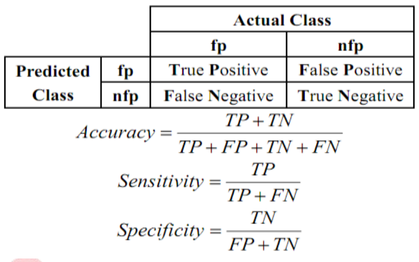
We suggest the following steps for calculating the scores:

- Find out n$_+$ as the average values of ratio(r) for each rule r from RAR$_+$ that is verified in the entity e and n$_-$ as the average values of ratio(r) for each rule r from RAR- that is not verified in the entity e.
- Calculate score$_+$ as score+ = n$_+$+ n-.
- Determine m$_-$ as the average values of ratio(r) for each rule r from RAR- that is verified in the entity e and m+ as the average values of ratio(r) for each rule r from RAR+ that is not verified in the entity e.
- Calculate score- as score$_-$ = m$_-$ + m$_+$.

The above offered score computation method takes into consideration the strength of the unproven and proven rules (by using the value of ratio, which increase as the confidence of the rule increases), although there are other potential for score calculation as well as using only the number of these rules, or calculating one single score, which can be changed into a class label with the use of a threshold. In the future we will examine extra score calculation formulas. At the classification stage of a new instance e if score+ > score_ then instance e will be specified as a positive instance (Defect), otherwise it will classify as a negative instance (non-defect).

7.4 TESTING

In testing by using ‘‘leave-one-out’’ methodology we can evaluate the performance our classifier. As for a binary classification task, the confusion matrix for the two achievable outcomes (negative and positive) is calculated. The confusion matrix will be shown below.



		Actual Class	
		fp	nfp
Predicted Class	fp	True Positive	False Positive
	nfp	False Negative	True Negative

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

$$Sensitivity = \frac{TP}{TP+FN}$$

$$Specificity = \frac{TN}{FP+TN}$$

Fig. 1: Confusion matrix and performance metrics for discrete classifiers.

The Confusion matrix consists of the number of true positives (TP) (TP – The no of real positive instances predicted as positive), the number of false positives (FP) (FP – the number of real negative instances predicted as positive), the no of true negatives (TN) (TN – the no of actual negative instances predicted as negative) and the no of false negatives (FN) (FN – the no of actual positive instances predicted as negative).

The result gives us different evaluation measures those values are computed based on the values from the confusion matrix. Why, in order to better compare our method to the presented ones, we are going to use in this paper a combination of the measures that be used in the previous to calculate software defect predictors.

When taking into consideration the values calculated from the confusion matrix, the subsequent estimate measures for defect detectors will be used in this paper:

1. The classification accuracy (represented by “Acc”) communicates to the percentage of instances that are classified correct (or) wrong by a classifier,

   $$\text{i.e.,}\quad \text{Acc} = \frac{TP+TN}{TP+TN+FP+FN}.$$

2. The probability of detection (represented by Pd), the classifier computes the amount of actual positives which are predicted positive,

   $$\text{i.e., Pd} = \frac{TP}{\text{TP+FN}}.$$

3. The specificity of the classifier (represented by “Spec”) calculates the proportion o actual negatives which are predicted negative,

   $$\text{i.e., Spec} = \frac{\text{TN}}{\text{TN+FP}}.$$

4. The classification precision (represented by “Prec”) procedures the proportion of predicted positives which are real positive,

   $$\text{i.e., Prec} = \frac{\text{TP}}{\text{TP+FP}}.$$

5. The Area under the ROC curve measure (AUC) is represented as one of the best estimation measure to compare different classifiers and it is recommended as the most important accuracy indicator for relative studies in software defect prediction. The ROC (Receiver Operating Characteristics) curve is a two-dimensional design of sensitivity vs. (1- specificity). ROC curves are generally constructed for classifiers which, in its place of directly returning the class of an instance and return a score that is transformed into a label using a threshold. Some cases, special (sensitivity, 1-specificity) pairs are obtained for every threshold, which are represented on the ROC curve. In case of classifiers returning the class directly, the ROC space has a single point. In this point can be linked to the points at (0, 0) and (1, 1), hence producing a curve, intended for which the AUC measure can be computed.

The ROC curves constructed for our preferably, detectors have high Pd; specificity and AUC. These measures have to be maximized in order to achieve better detectors. In experimental part of the paper (Section 8), these evaluation measures will be used for comparing the results given by the DPRAR classifier to the results of the classifiers previously existing in the software engineering literature.

.

8. EXPERIMENTAL EVALUATION

The experimentally evaluating our approach for defect discovery using relational association rules, as well as providing a comparison with other existing related approaches. The case studies used in our experiment, the method used, as well as the gained results are showed in the following. The datasets used in our experiments are open source and available at [13], a software engineering repository made publicly available in order to support repeatable, verifiable, and improvable analytical models of software engineering. These are the 13 public fault data repositories, from those we will use 10, frequently called NASA datasets, and were initially published at NASA's Independent Verification and Validation (IV&V) Facility website [12]. They were taken over by the PROMISE (Predictor Models In Software Engineering) repository, which has recently moved to a original address, and the older one is no longer available. A latest study initiate that out of 208 defect prediction studies 58 used at least one NASA dataset. In 2011 Gray et al. Describe that these datasets need serious data cleaning earlier than analysis, because they contain duplicated and unpredictable instances, mainly the Promise version of the datasets. Based on that Shepperd et al. in [18] first present that the datasets on the unique IV&V website and the ones at the Promise site are different both in number of instances and number of attributes. Then, they recognize possible problems with attributes and instances and present an algorithm that cleans the data; equally the implementation and the cleaned datasets are available online at the NASA – Software Defect Datasets webpage [13]. There are really two cleaned versions for each datasets: DS'- where duplicated and inconsistent instances are kept, and DS''- where duplicated and inconsistent instances are eliminated as well. These cleaned datasets are presently available in the Promise repository [9] as well. In all our analysis we have used the DS'' version of the datasets, taken from [13].

In our evaluation we are focusing on detecting software modules that are likely to be defective, thus an entity is measured to be a module, which can be a function, procedure or method, depending on the programming language. We declare that the DPRAR classifier is general, and it can also be used for detecting potential defective application classes, subprograms, etc., if a proper representation of these entities is provided.

The methodology represented as applied for each case study. The first stage, the data pre-processing stage that depends on the measured dataset will be complete for each case study. The previous steps of DPRAR, specifically building the DPRAR classifier and the testing step are useful. The datasets pre-processed as indicated above, are used for building the DPRAR classifier. In support of all the experiments, we have measured two possible relations between the software metrics characterize a software entity: ≤ and > (we have measured that the relations are not defined between zero valued software metrics) and we execute the classification algorithm with minimum support threshold $S_{min} = 0.9$ and dissimilar values for the minimum confidence thresholds for the dataset contains positive and negative instances. The minimum confidence threshold consider for the dataset DS_+ is denoted by c^+_{min} and the minimum confidence threshold considered for the dataset DS_- is denoted by c^-_{min}.

While conducting the case studies, we used a software framework that we have planned for binary classification, based on the finding of interesting relational association rules. This interface implements the DRAR algorithm (a variation of the DOAR algorithm previously introduced in [6]) developed for detecting relational association rules in a dataset.

8.1. The CM1 dataset

The CM1 dataset shows a NASA spacecraft instrument written in the C programming language. It contains 42 positive instances i.e. defects and 285 negative instances i.e. non-defects, meaning that

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

there are 12.84% positive instances and 87.16% negative instances. Every instance has 37 features and the class label.

8.1.1. DPRAR results

Fig. 2 represents the complete values of the correlations between the features (software metrics) and the target output (defects or correct) designed for the CM1 dataset.

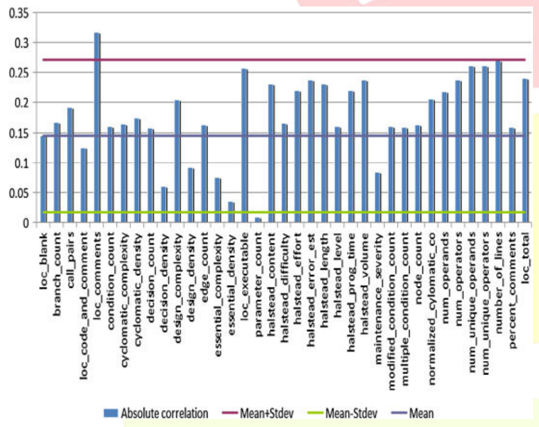


Fig. 2. Correlations for the CM1 dataset.

Case study	C^+ Min	C^- min	Len	Acc	Pd	Spec	Prec	Acc
CM1	0.92	0.94	Any	0.87	0.92	0.86	0.5	0.89
KC1	0.8	0.82	2	0.82	0.81	0.82	0.62	0.82
KC3	0.88	0.96	2	0.83	0.88	0.81	0.52	0.85
MC2	0.96	0.99	Any	0.89	0.77	0.96	0.91	0.86

Table .1.Obtained results for datasets

8.2. The KC1 dataset

The KC1 dataset contains data in favour of a C++ system implementing storage management for receiving and processing ground data. It contains 314 positive instances (defects) and 869 negative instances (non-defects), meaning that there are 26.54% positive instances i.e. defects and 73.46% negative instances i.e. non defects. Every instance has 21 features and the class label.

8.2.1. DPRAR results

Fig. 4 shows the absolute values of the correlations between the features (software metrics) and the target output (defects or correct) for the KC1 dataset. As a result of the analysis indicated we concluded that the third feature (software metric) loc_code_and_comment is somewhat correlated with the target output and it should therefore be removed from the feature set.

Table 1 presents the most excellent result obtained by the DPRAR classifier for the KC1 dataset (pre-processed as indicated below). We should point out that the maximal interesting relational association rules of various lengths (i.e. 2-length rules vs. any length rules) were measured for this case study collectively with different values for the minimum confidence thresholds.

8.3. The KC3 dataset

The KC3 dataset contains data about a system written in Java for processing and delivery of dependency metadata. It contains 36 positive instances (defects) and 158 negative instances (non-defects), meaning that there are 18.56% positive instances and 81.44% negative instances. Each instance has 39 features and the class label.

8.3.1. DPRAR results

As a result of the analysis indicated, we concluded that there is no slightly correlated feature (software metric) the output for the KC3 dataset. Consequently, the feature set remained the same and no features were removed from it.

Table 1 presents the best result obtained by the DPRAR classifier for the KC3 dataset (pre-processed as shown below). We should declare that the maximal interesting relational association rules of various lengths (i.e. 2-length rules vs. any length rules) were measured for this case study together with different values for the minimum confidence thresholds.

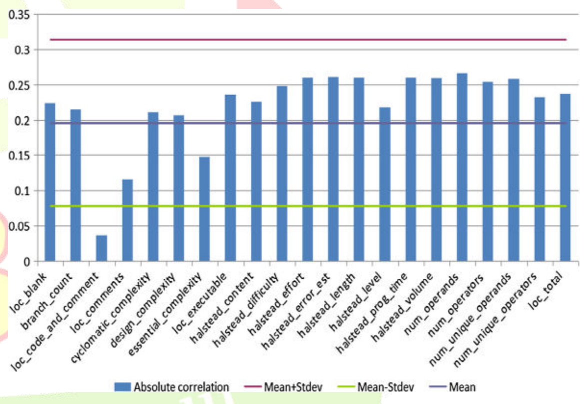


Fig.3. Correlations for the KC1 dataset.

8.4. The MC2 dataset

The MC2 dataset contains data regarding a video direction system, written in C/C++. It consists of 44 positive instances (defects) and 81 negative instances (non-defects) that contain 35.2% positive instances and 64.8% negative instances. Each instance has 39 features and the class label.

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

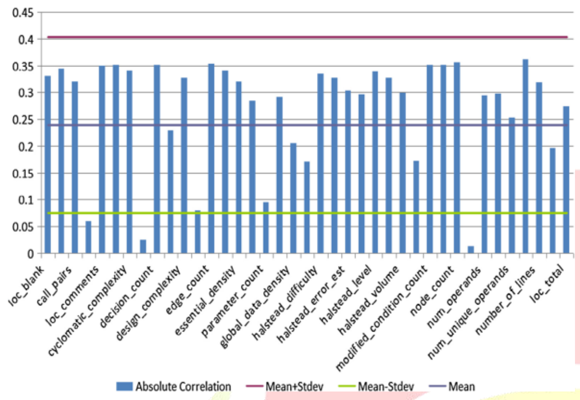


Fig. 4. Correlations for the MC2 dataset

8.4.1 DPRAR results

Fig. 4 represents the complete values of the correlations among the features (software metrics) and the target outputs (defect or correct) for the MC2 dataset.

As a result of the analysis indicated that features (software metrics) 4 (loc code and comment), 8 (cyclomatic density) and 32 (normalized cyclomatic complexity) are somewhat correlated with the target Output and they should consequently removed from the feature set.

Table 1 presents the best result obtained by the DPRAR classifier for the MC2 dataset (pre-processed as indicated below). We should mention that the maximal interesting relational association rules of different lengths (2-length rules between any length rules) were considered for this case study mutually with dissimilar values for the minimum confidence thresholds length rules) were considered for this case study collectively with dissimilar values for the minimum confidence thresholds.

9. Discussion

In that we aim at analyzing the method proposed in this paper by emphasizing its advantages and drawbacks, at the same time comparing the DPRAR classifier to other similar approaches existing in the software engineering defect detection literature.

The subsequent provides a comparison between the DPRAR method introduced in this paper and the CBA2 method, the 1R classifier, the Bagging classifier and the EDER-SD. The foremost reason for selecting the CBA2; 1R; Bagging and the EDER-SD methods for evaluation is that they were applied on datasets from the NASA repository [36], accordingly a evaluation of the obtained results is potential is the majority cases. One more reason is that CBA2 is a classification method based on association rule mining (DPRAR), EDER-SD is rule based (as DPRAR is) and 1R and Bagging were identified as the classifiers with the maximum accuracy among the classifiers that were experimented on the NASA datasets. And also we can conduct the comparison based on before results, that it will show DPRAR classifier better performance.

10. Conclusions and future work

We establish in this paper a classification model based on relational association rule finding for detecting in software systems software entities that are possible to be defective. Experiments were conducted in order to detect defective software modules, and the achieve results have shown that our classifier (DPRAR) is better than, or comparable to the classifiers previously useful for software defect detection, indicating the potential of our proposal.

Further work in the relational association rules invention will be made in order to discover and regard as different types of relations between the software metrics, relations that may be related in the mining process. We will also examine how the size of the rules and confidence of the relational association rules discovered in the training data may influence the accuracy of the classification task. Instructions to hybridize our classification model, by combining it with other machine learning based predictive models [11] will be considered too. We also plan to extend our model considering fuzzy relational association rules [14] and investigate their usefulness in software defect detection.

11. REFERENCES

1. R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufman Publishers Inc., San Francisco, CA, USA, 1994, pp. 487–499.

2. M. Baojun, K. Dejaeger, J. Vanthienen, B. Baesens, Software defect prediction based on association rule classification, Open Access publications from Katholieke Universiteit Leuven urn:hdl:123456789/296322, Katholieke Universiteit Leuven (February 2011).

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

4. G.D. Boetticher, Advances in Machine Learning Applications in Software Engineering, IGI Global, 2007 (Ch. Improving the Credibility of Machine Learner Models in Software Engineering).

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 13, April 2016

5. L.C. Briand, W.L. Melo, J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Trans. Softw. Eng. 28 (7) (2002) 706–720.

6. A. Campan, G. Serban, T.M. Truta, A. Marcus, An algorithm for the discovery of arbitrary length ordinal association rules, DMIN (2006) 107–113.

7. D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, The misuse of the NASA metrics data program data sets for automated software defect prediction,in: Proceedings of the Evaluation and Assesment in Software Engineering, 2011, pp. 96–103.

8. A. Marcus, J.I. Maletic, K.-I. Lin, Ordinal association rules for error identification in data sets, in: Proceedings of the Tenth International Conference on Information and Knowledge Management, CIKM '01, ACM, New York, NY, USA, 2001, pp. 589–591.

9. 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>..

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

11. T.M. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.

12. NASA independent verification& validation facility http://www.nasa.gov/centers/ivv/home/index.html>.

13. NASA software defect datasets <http://nasa-softwaredefectdatasets.wikispaces.com/>.

14. Bin Pe, Suyun Zhao, Hong Chen, Xuan Zhou, Dingjie Chen, FARP: Mining fuzzy association rules from a probabilistic quantitative database, Inform. Sci. 237 (2013) 242–260.

15. N.J. Pizzi, A fuzzy classifier approach to estimating software quality, Inform. Sci. 241 (2013) 1–11.

16. M.S. Rawat, S.K. Dubey, Software defect prediction models for quality improvement: a literature study, Int. J. Comp. Sci. Iss. 9 (2) (2012) 288–296.

17. D. Rodriguez, R. Ruiz, J.C. Riquelme, J.S.N. Aguilar-Ruiz, Searching for rules to detect defective modules: a subgroup discovery approach, Inform. Sci. 191 (2012) 14–30.

18. M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: some comments on the NASA software defect data sets, IEEE Trans. Softw. Eng. 99 (2013) 1(PrePrints).

19. F. Simon, F. Steinbruckner, C. Lewerentz, Metrics based refactoring, in: CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2001, pp. 30–38.

20. Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect proneness prediction framework, IEEE Trans. Softw. Eng. 37 (3) (2011) 356–370.

21. S. Stehman, Selecting and interpreting measures of thematic classification accuracy, Rem. Sens. Environ. 62 (1) (1997) 77–89.

22. P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, first ed., Addison-Wesley, Longman Publishing Co., Inc., Boston, MA, USA, 2005.

23. 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.

24. C. Spearman, The proof and measurement of association between two things, Am. J. Psychol. 15 (1904) 72–101.