STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 4, 2013

ON THE SOFTWARE METRICS INFLUENCE IN RELATIONAL ASSOCIATION RULE-BASED SOFTWARE DEFECT PREDICTION

ZSUZSANNA MARIAN

Abstract. Software defect prediction tries to automatically identify defective software modules, in order to help software testers focus their time and effort on those modules which are likely to contain faults. So far many different machine learning algorithms have been used for this classification task. We have introduced a new software defect prediction method, called DPRAR, which uses relational association rules to classify modules, represented by a vector of software metric values, as faulty or non-faulty. In this paper we investigate how different feature elimination techniques influence the results of the DPRAR method. We also consider two methods for computing the scores for a module, which are two values showing how close the module is to the faulty instances and the non-faulty instances. Experiments on an open source dataset as well as comparisons to related work are provided.

1. Introduction

Software systems are becoming more complex nowadays, made of many components, with a great number of connections among them. In such complex systems a thourough manual analysis and testing is usually impossible due to time and size constraints. Still, releasing good quality and bug free (or with as few bugs as possible) software is crucial for many organizations. This is why different intelligent methods are often used, which can suggest those modules, classes or components which are more likely to contain errors, thus helping developers select which parts to focus on during testing. Such methods often use the value of software metrics, and try to find anomalies in these values.

In [7] we have presented and experimentally analysed a novel model for software defect prediction using relational association rule-based classification.

Received by the editors: October 8, 2013.

2010 *Mathematics Subject Classification.* 62H30, 68N99.

1998 *CR Categories and Descriptors.* H.2.8 [**Database Management**]:Database Applications - Data Mining; D.2.4 [**Software Engineering**]: Software/Program Verification;

Key words and phrases. relational association rules, software metrics, software defect prediction.

This model, presented in more detail in Section 2.2, uses a representation where each module is described by a set of software metrics, also called features, and it has a preprocessing step during which different software metrics can be chosen to be eliminated from further steps of the model. In this paper we analyse the effect of using different feature elimination methods on the results of defect prediction. We will present three different feature elimination methods and compare their results to the case when no feature is eliminated.

The model presented in [7] computes a score for every module, and the prediction is based on this score. After determining the best feature elimination method, we will also investigate the results of using a different formula for computing this score.

The rest of this article is organized as follows. Section 2 presents the background for this paper, in two subsections: Section 2.1 presents some similar approaches from the literature in the field of defect prediction and Section 2.2 shortly presents the DPRAR model, a relational association rule-based classification model. Section 3 first presents the datasets used for the experiment (Section 3.1), followed by the description of the comparison criteria (Section 3.2) and finally the description of the measures used for evaluating the results (Section 3.3). Section 4 presents the results of the experiments (Section 4.1) and a comparison to the results reported in related work from the literature (Section 4.2). Finally, Section 5 draws the conclusions of the paper and identifies directions for future work.

2. Background

2.1. **Related Work.** Software defect prediction is a well researched field, with many different approaches presented in the literature. In this section we will give a short description of some methods from the literature that are either similar to our method (i.e. they are using association rules) or they are tested on the same dataset as our method, so a direct comparison of the results will be possible.

According to our knowledge, no one has used relational association rules for software defect prediction before, but other association rule-based methods are presented in the literature. One example is the CBA method, presented in [13], where so-called class association rules are mined (these are association rules whose consequent is the class label). An extension of this method, the CBA2 method, is presented in [14], where different minimum support can be used for rules predicting different classes, thus trying to solve the class imbalance problem. In [4] this CBA2 method is used for defect prediction and results of experiments on different Nasa datasets are presented.

In [16] Rodriguez et al. present an algorithm, called *EDER-SD* (Evolutionary Decision Rules for Subgroup Discovery), which generates only rules describing the faulty modules. Experiments performed on different Nasa datasets revealed that this method is better than other Subgroup Discovery algorithms.

Besides rule-based methods, different other machine learning algorithms were also applied for software defect prediction. For example, in [10] 37 different classifiers are compared on five Nasa datasets. The conclusion of the paper is that there is no single classifier that performs best for every dataset, but, on average, bagging has the best performance. Menzies et al. compare three classifiers, OneR, J48 and Naive Bayes, and different filters in [15] out of which Naive Bayes with a log filter produced the best results. Challagulla et al. compare 11 classifiers in [5] and conclude that there is no method which performs best for every dataset, but, in general, OneR and Instance Based Learning perform well.

Another direction for software defect prediction is the use of disagreement-based semi-supervised learning methods. One such method is ROCUS, presented in [11], which builds several classfiers for the labeled examples and then tries to assign labels for the rest of the examples based on majority voting among the classifiers. It also uses under-sampling to handle the class imbalance problem. Another similar method is presented in [12], called ACoForest, which uses active-learning, which means that it can suggest which examples to label (the ones on which the classifiers mostly disagree), so in a software defect prediction scenario, it can suggest which modules should be tested.

Since defect prediction datasets are usually imbalanced, i.e. there are a lot more negative examples than positive ones, many methods try to handle this problem, too. Besides CBA2 and ROCUS, which try to solve this problem, there are other methods which focus on this aspect. One such method is presented in [21], where the imbalanced binary dataset is first transformed into a balanced multiclass dataset and the classifiers are trained on this new dataset. Wang and Yao in [22] first evaluate different existing class imbalance learning methods, and for the best one, AdaBoost, they introduce a new dynamic version, which can automatically adjust its parameters during training.

2.2. **The DPRAR model.** The DPRAR (*Defect Prediction using Relational Association Rules*) model is a novel supervised method for detecting defective software entities, introduced in [7]. One important part of this approach is to represent the entities of the software system, which can be methods, classes, modules and so on, as a multidimensional vector, where each element of the vector is the value of a software metric (feature), computed for the given entity. The main problem in defect detection is to determine whether such an entity contains defects or not.

Such a problem can be considered a binary classification problem, with two classes, defective (denoted in our case by "+") and non-defective (denoted by "-"). In order to classify an entity (represented as a multidimensional vector of software metric values) as defective or not, the following three steps are performed:

- Data pre-processing;
- Training/building the DPRAR classifier;
- Classification/testing.

During the first step, *data pre-processing*, the software metric values are scaled to the [0,1] interval, and a statistical analysis is carried out to determine if some of the software metrics should be eliminated from further analysis. During this analysis the Spearman's rank correlation of each software metric to the target value is computed and, based on the result, some software metrics can be eliminated, reducing the dimensionality of data. In this paper we will analyse how different feature elimination methods influence the result of classification. The elimination methods used in this analysis are presented in Section 3.2.

Before presenting the second step, we will shortly describe what *relational association rules* are, as presented in [17]: let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of *instances*, in our case modules, where each instance is characterized by a list of m attributes $(a_1, \ldots, a_m)$. We denote by $\Phi(r_j, a_i)$ the value of attribute a_i for the instance r_j. Each 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 ($\leq$), equal ($=$), greater or equal ($\geq$), etc. We denote by M the set of all possible relations that can be defined on $D_i \times D_j$.

A *relational association rule* [17] is a sequence $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell})$ such that $a_{i_j}\ \mu_{i_j}\ a_{i_{j+1}}$ holds for each $1 \leq j \leq l-1$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_m\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $\mu_i \in M$ is a relation over $D_{i_j} \times D_{i_{j+1}}$, D_{i_j} is the domain of the attribute a_{i_j}. If:

a) $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together (are non-empty) in $s\%$ of the n instances, then we call s the *support* of the rule,
and
b) we denote by $R' \subseteq R$ the set of instances r_j where $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together and $\Phi(r_j, a_{i_k})\ \mu_k\ \Phi(r_j, a_{i_{k+1}})$ holds for each $1 \leq k \leq l-1$ for each instance r_j from R'; then we call $c = |R'|/|R|$ the *confidence* of the rule.

The *length* of a relational association rule is the number of attributes in the rule, which can be at least two and at most the number m of attributes. In any dataset a huge number of relational association rules can be found, but we are

usually interested only in rules with at least s_{min} support and c_{min} confidence, also called interesting rules. Both s_{min} and c_{min} are user defined thresholds. In [6] we have presented an Apriori [3] like algorithm, called $DRAR$, which can efficiently find all relational association rules in a dataset.

During the second step, the DPRAR classifier is built. Since software defect prediction is a supervised classification task, we need a set of training data, which is divided into two subsets: DS_+, consisting of those instances from the training data which are defective, and DS_-, consisting of the non-defective instances. On these sets relational association rule mining is performed, which will give two sets of relational association rules, denoted by RAR_+ (rules mined from DS_+) and RAR_- (rules mined from DS_-). For each rule r from these two sets, a value called $ratio(r)$, is assigned, obtained by dividing the rule's confidence to its support: $\frac{conf(r)}{supp(r)}$.

The third step of the model consists of the classification of a new entity. At this step, we will compute two scores, $score_+$ and $score_-$, which try to capture the similarity of the entity to the defective (through $score_+$) and the non-defective (through $score_-$) instances. In [7] these scores for an entity e were computed in the following way:

- Compute n_+ as the average values of $ratio(r)$ for each rule $r \in RAR_+$ that is verified by the entity e, and compute n_- as the average values of $ratio(r)$ for each rule $r \in RAR_-$ that is not verified by the entity e.
- Calculate $score_+$ as $score_+ = n_+ + n_-$.
- Compute m_- as the average values of $ratio(r)$ for each rule $r \in RAR_-$ that is verified by the entity e, and compute m_+ as the average values of $ratio(r)$ for each rule $r \in RAR_+$ that is not verified by the entity e.
- Calculate $score_-$ as $score_- = m_+ + m_-$.

We can see that $score_+$ measures not only how "close" the entity is to the positive instances, but also how "far" it is from the negative ones. When the two scores are computed, if $score_+ > score_-$ we will classify the entity as defective, otherwise it will be classified as non-defective. Besides analysing the effect of different feature elimination methods, this paper will also investigate the effect of using a different formula for computing the value of $score_+$ and $score_-$, presented in Section 3.2.

3. Experiments

This section will describe the experiments that we performed both for feature elimination and score computation. It also presents the dataset used for these experiments, and the measures used for evaluating the results.

3.1. **Datasets.** In order to be able to compare our results to other results reported in the literature, we have decided to use one of the Nasa datasets for this study. These datasets are open source and publicly available at [2], in order to encourage the development of repeatable, verifiable and improvable predictive models. These datasets were originally published at Nasa's Independent Verification and Validation (IV & V) Facility website [1], but are no longer available there. In 2011 Gray et al. in [9] describe that, despite their popularity, these datasets should be used carefully, because they need serious data cleaning before use, because they contain a large number of duplicated and inconsistent instances. While Gray et al. focuses more on the effect of these duplicated and inconsistent instances on the performance of predictive models, Shepperd et al. in [18] present, among others, an algorithm that cleans the data. Both the implementation of the algorithm, and the cleared datasets are available at the NASA - Software Defect Datasets webpage [2].

While in [7] we have used 10 out of the 13 Nasa datasets, in this paper we will perform our experiments on only one of these datasets, called PC3. This dataset contains data about modules from a flight software for earth orbiting satellite, written in C. It consists of 134 positive instances (defects) and 943 negative instances (non-defects), namely the class distribution is 12.4% instances are positive and 87.6% instances are negative. Each instance has 37 features and the class label.

3.2. **Comparison criteria.** As mentioned in Section 2.2, the DPRAR model has a preprocessing step, when a statistical analysis is carried out, which can lead to the elimination of different software metrics. Obviously, we would like to eliminate those software metrics that do not significantly influence the output value. In order to do so, for every software metric, we computed the Spearman's rank correlation coefficient [19] between the software metric and the class label. A Spearman correlation of 0 between two variable X and Y indicates that there is no tendency for Y to increase or decrease when X increases. A Spearman correlation of 1 or -1 results when the two variables being compared are monotonically related, even if their relationship is not linear.

After computing the Spearman correlation of each software metric to the class label, we compute the mean (denoted by m) and standard deviation (denoted by $stdev$) of these correlations as well. The correlations, as well as three lines, denoting the value $m + stdev$, m and $m - stdev$ are presented on Figure 1.

In the following we will present those four different feature elimination methods that are experimentally analysed in this paper:

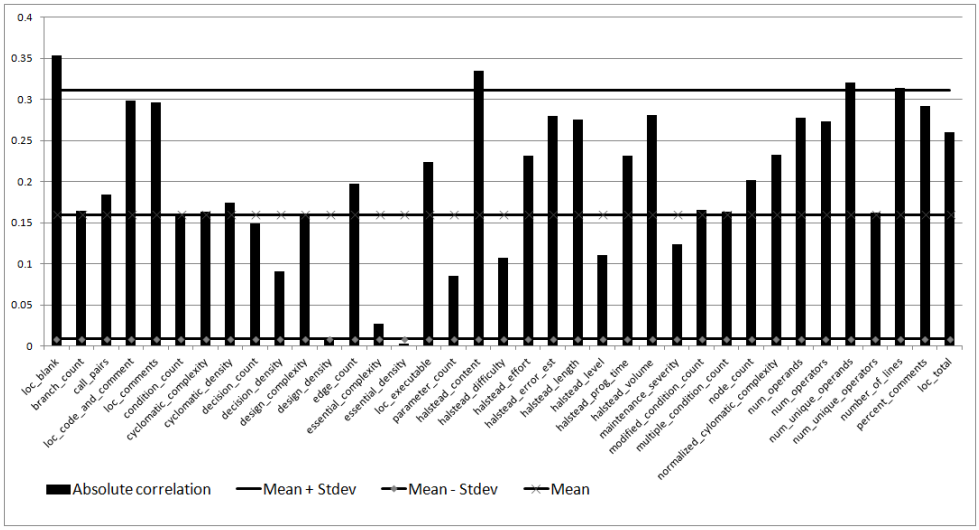

FIGURE 1. Correlations for **PC3** dataset

(1) **Case 1.** The simplest version is to keep all the features. In case of the PC3 dataset, this means that all 37 features will be used to generate the relational association rules.
(2) **Case 2.** To eliminate only those features, whose correlation is lower than $m - stdev$. This is the variant that we used in [7]. In case of the PC3 dataset, this means that only one feature is removed, feature number 15, called *essential density*.
(3) **Case 3.** To eliminate only those features, whose correlation is higher than $m + stdev$. In case of the PC3 dataset, this means that four features will the eliminated: the first feature, *loc blank*, feature number 18, *halstead content*, feature number 33, *num unique operands* and feature number 35, *number of lines*.
(4) **Case 4.** To eliminate both features with correlation lower than $m - stdev$ and higher than $m+stdev$. In case of the PC3 dataset, this means eliminating the features mentioned for the two previous cases, namely: *essential density*, *loc blanc*, *halstead content*, *num unique operands*, *number of lines*.

Besides determining which of the above described four feature elimination cases gives the best results, we want to experiment with the score calculation formula as well. As presented in Section 2.2, for a module to be classified as defective or not, we compute two values: $score_+$ and $score_-$. In [7], $score_+$ is computed by taking into consideration the *ratio* of rules from RAR_+ verified

by the module and the *ratio* of the rules from RAR_-, not verified by the module. $Score_-$ is computed similarly.

We will experiment with a different score computation formula, where, for an entity e, $score_+$ and $score_-$ are computed in the following way:

$$score_+(e) = \frac{nrverif_+(e) + nrneverif_-(e)}{|RAR_+| + |RAR_-|} \tag{1}$$

$$score_-(e) = \frac{nrverif_-(e) + nrneverif_+(e)}{|RAR_+| + |RAR_-|} \tag{2}$$

where $nrverif_+(e)$ is the number of rules from RAR_+ that are verified by the entity e, and $nrneverif_-(e)$ is the number of rules from RAR_- that are not verified by e. In the rest of this paper we will denote this score computation method as $score_{number}$, while the other method, presented in Section 2.2, will be called $score_{ratio}$.

3.3. **Evaluation measures.** In this section we present the measures used to evaluate the results of the experiments. For every experiment, we used a "leave-one-out" cross-validation methodology. After evaluating each entity, we compute the *confusion matrix* [20], a two-by-two matrix, which consists of the number of *true positives* (TP - the number of positive entities that were classified by the DPRAR method as defective), *true negatives* (TN - the number of negative entities that were classified negative), *false positives* (FP - the number of negative entities classified as positive) and *false negatives* (FN - the number of positive instances classified as negative).

In the literature different evaluation measures, whose value can be computed based on these four values (TP, TN, FP, FN), are presented. In this paper we are going to use three of them, *precision*, *recall* and AUC:

- *Precision*, denoted by *Prec*, denotes the proportion of predicted positives which are actual positives, i.e. $Prec = \frac{TP}{TP+FP}$.
- *Recall*, also called *probability of detection*, is the proportion of actual positives, which are predicted positives, i.e. $Recall = \frac{TP}{TP+FN}$.
- AUC or *Area Under the ROC Curve* is often considered the best evaluation measure for comparing different classifiers [8]. The ROC (Receiver Operating Characteristics) curve is a two-dimensional plot of *recall* versus (1 - *specificity*). ROC curves are usually constructed for classifiers which, instead of directly returning the class of an instance, return a score, which is transformed into the class label using a threshold. For different values of the threshold, different (*recall*, 1 - *specificity*) pairs are obtained, which are represented on the ROC

curve. As its name suggests, AUC measures the area under this curve. In case of classifiers that return the class directly, like the DPRAR classifier, the ROC space has one single point. As presented in [8], this point can be linked with the points at (0,0) and (1,1), thus producing a curve for which the AUC value can be computed.

For all three of these measures, the maximum value is 1, which represents a perfect classifier (no false negatives and false positives).

4. Results and Discussion

This section presents the results of the performed experiments, together with an analysis of these results, and a comparison to other results reported in the literature for the PC3 dataset.

4.1. **Results and analysis.** The results of the experiments for the four feature elimination cases presented in Section 3.2 are presented on Figure 2. For all four cases only rules of length 2 were mined, with a minimum support threshold of 0.9. The minimum confidence threshold differs for the two classes, for DS_+ a threshold of 0.95 and for DS_- a threshold of 0.995 was used. For computing the scores, the $score_{ratio}$ formula was used.

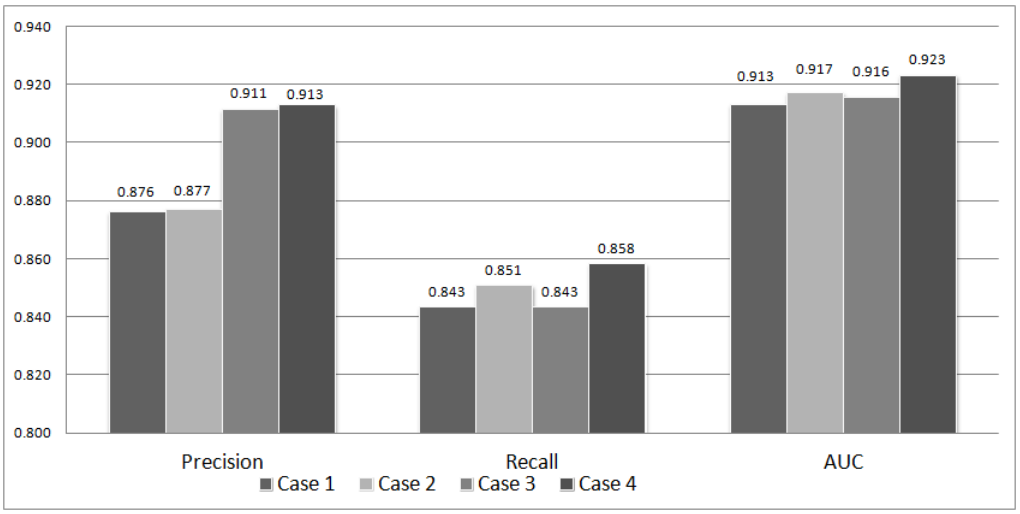

Figure 2. The results for the four different feature elimination methods.

From this Figure we can see that Case 4 has the highest value for each measure. This is the case, when five software metrics are removed from the analysis, both the ones with correlation less than $m - stdev$ and the ones with correlation greater than $m + stdev$.

The worst performance, considering all three measures, is achieved for Case 1, the case when no software metrics are removed. In case of Recall and AUC this difference is quite small, 0.015 and 0.01 respectively, but for Precision the difference is 0.037. This suggests that using any feature elimination method improves the classifier.

For the other two cases the comparison is not so easy. Considering Precision, Case 3 is a lot better than Case 2 (by 0.34), but considering Recall and AUC Case 2 has better values, even if in case of AUC the difference is only 0.001. In order to compare them better, we have looked at the number of *false negatives*. False negatives are those instances which are actually positive, but predicted as negative. In case of defect prediction, *false negatives* represent faulty modules, classified as non-faulty. This is an error which, for defect prediction, is more important than *false positives*, non-faulty modules classified as faulty. Considering *false negatives*, Case 2 is better than Case 3, because it has 20 *false negatives*, while Case 3 has 21. Case 4, having the best values for all measures, has only 19.

For the best case, Case 4, we have performed another experiment, using the $score_{number}$ formula for computing the score. The results of this experiment, compared to the results for Case 4, but with $score_{ratio}$ are presented on Figure 3.

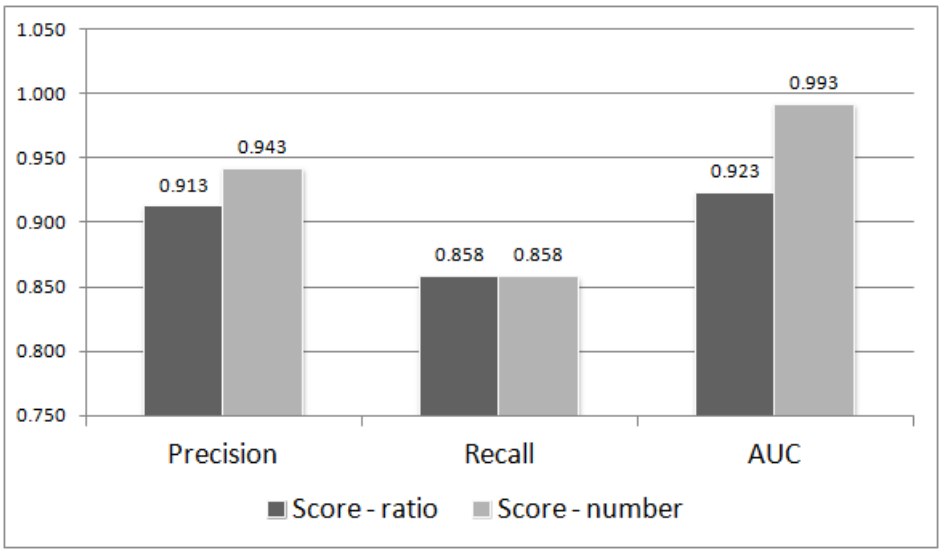

FIGURE 3. The results for Case 4 with the two different score computation formulas.

Method	Precision	Recall	AUC
DPRAR - $Score_{ratio}$	0.913	**0.858**	0.923
DPRAR - $Score_{number}$	**0.943**	**0.858**	**0.925**
CBA2	n/a	0.255	0.821
ROCUS	n/a	n/a	0.795
Random Forest with one - against - one coding	n/a	n/a	0.85
Dynamic AdaBoost.NC	n/a	n/a	0.816

TABLE 1. Comparative results.

From Figure 3 we can see that both score computation formula leads to the same Recall. Considering the other two measures, $score_{number}$ has higher values than $score_{ratio}$, even if the difference is quite small. Considering all these, we can conclude, that using $score_{number}$ instead of $score_{ratio}$ improves the performance of the DPRAR method for the PC3 dataset.

4.2. **Comparison to Related Work.** In this section we will present a short comparison of the results of the *DPRAR* method to other methods. Unfortunately, we can not compare our method to all methods presented in the Related Work section, because:

- Not all methods are tested on Nasa datasets, and even in case of the methods that are tested on Nasa datasets, not all of them use the PC3 dataset, that we have used.
- Not all papers report the same evaluation measures that we use. AUC is often reported, but Precision and Recall are not, instead other different measures are used, and usually there is no measure used in every paper.

Still, even for cases when the PC3 dataset was used, and the value of at least one measure used by us is reported, an exact comparison is problematic. One reason for this is the fact that we have used, as mentioned in Section 3.1, the cleaned version of the PC3 dataset, but others did not. As presented by an experiment in [9], using the uncleaned version, which contains duplicates, can lead to better performance, because training and test sets are not perfectly separated, they can contain common elements. Another difference can be in the testing methods, we have used a *leave-one-out* cross-validation method, while others used different methods.

The comparison to other methods is presented in Table 1. For each measure, the bold value represents the highest value for the given measure. n/a stands for *not available*, for measures that were not reported for a given method.

The first two rows from Table 1 report the results of our method, for Case 4, for the two different score computation formulas. The third row presents the values reported for the CBA2 method in [4]. The next line contains the values for the ROCUS method, as reported in [11]. We would like to mention that results for four different parameter settings are reported for ROCUS, and we used here the highest AUC achieved in these experiments. The fifth line contains the best results reported in [21] for a Random Forest algorithm, using one-against-one coding. Finally, the last line presents the results of the Dynamic AdaBoost.NC algorithm introduced in [22].

From Table 1 we can conclude that the best method, considering Precison, Recall and mainly AUC, is DPRAR with the $score_{number}$ score computation, while the second is DPRAR with $score_{ratio}$.

5. Conclusions and Further Work

In this paper we have presented an experimental study about the effect of different feature elimination techniques for a relational association rule-based software defect prediction method, called DPRAR, introduced in [7]. All used feature elimination techniques were based on the Spreaman's rank correlation of the features to the class label. We have presented four different cases for feature elimination, and concluded that the best results, in terms of Precision, Recall and AUC, are achieved when both features with correlation less than $m - stdev$ (mean correlation minus standard deviation of all correlations) and features with correlation higher than $m + stdev$ are eliminated.

An important part of the DPRAR method is the score computation, since the final prediction (i.e. whether an instance is predicted as faulty or not) is directly based on this score. In [7] we have used a score computation based on the support and confidence of rules. Here we performed an additional experiment, with a different score computation formula, which considers only the number of verified and unverified rules. This experiment showed that this second score computation leads to better results than the one used in [7].

In the future we would like to perform experiments on other Nasa datasets as well, to further investigate both the attribute elimination strategy and this different score computation method.

References

[1] Nasa independent verification & validation facility, http://www.nasa.gov/centers/ivv/home/index.html.
[2] Nasa software defect datasets, http://nasa-softwaredefectdatasets.wikispaces.com/.
[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very*

Large Data Bases, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] Ma Baojun, Karel Dejaeger, Jan Vanthienen, and Bart 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.

[5] Venkata U. B. Challagulla, Farokh B. Bastani, I-Ling Yen, and Raymond 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, pages 263–270, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Detecting software design defects using relational association rule mining. *Knowledge and Information Systems*, 2013. Under review.

[7] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Software defect prediction using relational association rule mining. *Information Sciences*, 2013. Under review.

[8] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[9] David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce 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*, pages 96–103, 2011.

[10] A.A. Shahrjooi Haghighi, M. Abbasi Dezfuli, and S.M. Fakhrahmad. Applying mining schemes to software fault prediction: A proposed approach aimed at test cost reduction. In *Proceedings of the World Congress on Engineering 2012 Vol I*, WCE 2012,, pages 1–5, Washington, DC, USA, 2012. IEEE Computer Society.

[11] Yuan Jiang, Ming Li, and Zhi-Hua Zhou. Software defect detection with rocus. *Journal of Computer Science and Technology*, 26(2):328–342, 2011.

[12] Ming Li, Hongyu Zhang, Rongxin Wu, and Zhi-Hua Zhou. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, 19(2):201–230, 2012.

[13] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 80–86, 1998.

[14] Bing Liu, Yiming Ma, and Ching-Kian Wong. *Data Mining for Scientific and Engineering Applications*, chapter Classification Using Association Rules: Weaknesses and Enhancements. Kluwer Academic, 2001.

[15] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[16] D. Rodríguez, R. Ruiz, J. C. Riquelme, and J. S. Aguilar-Ruiz. Searching for rules to detect defective modules: A subgroup discovery approach. *Inf. Sci.*, 191:14–30, May 2012.

[17] Gabriela Serban, Alina Câmpan, and Istvan Gergely Czibula. A programming interface for finding relational association rules. *International Journal of Computers, Communications & Control*, I(S.):439–444, June 2006.

[18] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect data sets. *IEEE Transactions on Software Engineering*, 99(PrePrints):1, 2013.
[19] C. Spearman. The proof and measurement of association between two things. *Amer. J. Psychol.**15***, pages 72–101, 1904.
[20] S. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, October 1997.
[21] Zhongbin Sun, Qinbao Song, and Xiaoyan Zhu. Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(6):1806–1817, 2012.
[22] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2):434–443, 2013.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu St., 400084 Cluj-Napoca, Romania

E-mail address: marianzsu@cs.ubbcluj.ro