Searching for rules to detect defective modules: A subgroup discovery approach

D. Rodríguez a,⇑, R. Ruiz b, J.C. Riquelme c, J.S. Aguilar–Ruiz b

a Department of Computer Science, University of Alcalá, Ctra. Barcelona, Km. 31.6, 28871 Alcalá de Henares, Madrid, Spain
b School of Engineering, Pablo de Olavide University, Ctra. Utrera km. 1, 41013 Seville, Spain
c Department of Computer Science, University of Seville, Avda. Reina Mercedes s/n, 41012 Seville, Spain

Article info
Article history:
Available online 18 February 2011

Keywords:
Defect prediction
Subgroup discovery
Imbalanced datasets
Rules

Abstract
Data mining methods in software engineering are becoming increasingly important as they can support several aspects of the software development life-cycle such as quality. In this work, we present a data mining approach to induce rules extracted from static software metrics characterising fault-prone modules. Due to the special characteristics of the defect prediction data (imbalanced, inconsistency, redundancy) not all classification algorithms are capable of dealing with this task conveniently. To deal with these problems, Subgroup Discovery (SD) algorithms can be used to find groups of statistically different data given a property of interest. We propose EDER-SD (Evolutionary Decision Rules for Subgroup Discovery), a SD algorithm based on evolutionary computation that induces rules describing only fault-prone modules. The rules are a well-known model representation that can be easily understood and applied by project managers and quality engineers. Thus, rules can help them to develop software systems that can be justifiably trusted. Contrary to other approaches in SD, our algorithm has the advantage of working with continuous variables as the conditions of the rules are defined using intervals. We describe the rules obtained by applying our algorithm to seven publicly available datasets from the PROMISE repository showing that they are capable of characterising subgroups of fault-prone modules. We also compare our results with three other well known SD algorithms and the EDER-SD algorithm performs well in most cases.

1. Introduction

Software Quality remains an important topic of research within the software engineering community. There are many definitions of software quality, but in this context we refer to software reliability, which is generally defined as the probability of failure-free software operation for a specified period of time in a given environment [43]. One way of improving software reliability and guiding the testing effort is through static metrics [15] capable of predicting fault-prone modules.

There is a wide range of defect prediction techniques using statistical methods and more recently, data mining techniques (see Section 5). This is due to the creation of a number of publicly available data repositories obtained from real projects that allow researchers and practitioners to apply data mining techniques. Examples of such repositories include PROMISE1 and

⇑ Corresponding author. Tel.: +34 918856639; fax: +34 918856646.
E-mail addresses: drg@ieee.org, daniel.rodriguezg@uah.es (D. Rodríguez), robertoruiz@upo.es (R. Ruiz), riquelme@lsi.us.es (J.C. Riquelme), aguilar@upo.es (J.S. Aguilar–Ruiz).

1 http://promisedata.org/.

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.01.039
FLOSSMetrics\(^2\). There are however several issues that need to be considered when applying data mining techniques to defect prediction data.

First, datasets can be imbalanced. In fact, like most datasets in defect prediction, the datasets used in this work are highly imbalanced, i.e., samples of non-defective modules vastly outnumber the defective ones. In this situation, many data mining algorithms generate poor models because they try to optimize the overall accuracy but perform badly in classes with very few samples. For example, if the number of non-defective samples outnumbers the defective samples by 90%, an algorithm that always predicts a module as non-defective will obtain a very high accuracy. As a result, many data mining algorithms obtain biased models that do not take into account the minority class (defective modules in this case).

Second, although in theory having more attributes could provide further discriminant power, experience with data mining algorithms shows just the opposite [36]. Removing irrelevant, redundant or noisy data provides immediate benefits including performance improvements with respect to speed, predictive accuracy and comprehensibility of the results.

Finally, another problem when applying data mining techniques is that data can have duplicated (i.e., identical attribute values including the class) and contradictory cases (i.e., instances with same attribute values but the class).

Recently a new set of descriptive induction algorithms categorised as Subgroup Discovery (SD) algorithms [30,58,59] have been proposed to discover statistically different subgroups of data with respect to a property of interest, making these types of algorithms suitable for extracting knowledge from imbalanced datasets.

In this work, we tackle the defect prediction problem through a descriptive induction process using SD. The objective is to generate useful models represented through rules characterising fault–prone modules. The induced rules allow us to determine software metrics and their thresholds that increase the probability of detecting fault–prone modules. The rules are obtained with an Evolutionary Algorithm (EA), called EDER-SD (Evolutionary Decision Rules for Subgroup Discovery). EAs have the advantage that it is possible to optimise different fitness functions, such as precision, recall or coverage, depending on the characteristics of the domain knowledge from experts.

Empirical work was performed using seven publicly available NASA datasets (CM1, KC1, KC2, KC3, MC2, MW1 and PC1) related to software defect prediction from the PROMISE repository [5]. The induced rules from EDER-SD show that our technique generates understandable and useful models that can be used by project managers or quality assurance personnel to guide the testing effort and improve the quality of software development projects. EDER-SD is also compared with three other standard techniques in SD, performing well in most cases.

The organization of the paper is as follows. Section 2 presents the most relevant concepts related to the process and techniques used in this work. Section 3 explains the modifications to a hierarchical classification algorithm to be adapted to subgroup discovery in defect prediction. Section 4 describes the experimental work and discusses the results. Section 5 describes the related work in defect prediction. Finally, Section 6 concludes the paper and outlines future research work.

2. Background

Data mining techniques can be grouped into predictive and descriptive depending on the problem at hand. From the predictive point of view, patterns are found to predict future behaviour. In fault prediction, it would correspond to the generation of classification models to predict whether a software module will be defective based on metrics from historical project data. From the descriptive point of view, the idea is to find patterns capable of characterising the data represented in such a way that domain experts can understand them. A comprehensive framework for data mining is described by Peng et al. [46].

The application of data mining in the context of defect prediction aims to extract useful and applicable knowledge from the datasets obtained from the software modules in such a way that it can be used for decision making. There are different possible representations of the learning–based models including decisions trees, decision rules, rules with exceptions, fuzzy rules, artificial neural networks, among others. However, when the output needs to be directly interpreted by end–users (e.g., project manager, quality assurance manager, testers), the readability and understandability of the representation needs to be considered (for example, in the case of rules, hierarchical rules are much harder to understand than non–hierarchical rules). In this respect, we selected rules as a representation as they are considered far more simple and intuitive than other representations.

In the following subsections we describe data mining concepts used in this work.

2.1. Reduction of imbalanced datasets

The existence of irrelevant and redundant features in the datasets has negative impact in most data mining algorithms. As we are dealing with datasets of collected metrics from modules, we need to consider the following. First, the larger the number of metrics collected per module, the larger the need of data samples to ensure the quality of the learned patterns due to statistical variability between patterns of different class. This problem is known as the curse of dimensionality [17].

Second, redundant or irrelevant features may mislead learning algorithms or cause them to overfit the data [36]. Hence, the obtained classifier is in general less accurate than the one learned from the relevant data. Conversely, a dataset with less dimensionality can in most cases improve the accuracy of models, generate simple, understandable models and the data mining algorithms can be run faster.

\(^2\) http://flossmetrics.org/.
As stated previously like most datasets in defect prediction, the datasets used in this work are highly imbalanced, i.e., samples of non–defective modules vastly outnumber the cases of defective modules. Under this situation, when the balanced data is not considered, many learning algorithms generate distorted models for which (i) the impact of some factors can be hidden and (ii) the prediction accuracy can be misleading. This is due to the fact that most data mining algorithms assume balanced datasets. When dealing with imbalanced datasets, there are two alternatives, either (i) sampling or balancing techniques: over–sampling algorithms aimed at balancing the class distribution increasing the minority class, or under–sampling algorithms that change the class removing instances from the majority classes; and (ii) to apply algorithms that are robust to this problem [51,23,35,16].

A few authors have applied Feature Subset Selection (FSS) techniques to software engineering data. In the case of effort estimation, it has been reported that reduced datasets improve the estimation accuracy [29,9,34,8]. It is known, however, that feature selection algorithms do not perform well with imbalanced datasets, resulting in a selection of metrics that cannot be adequate for the learning algorithms, decreasing the quality and usefulness of the rules. In previous work, we analysed the application of FSS to the datasets used in this work [50]. In particular, we applied CFS (Correlation–based Filter Selection) a feature selection algorithm based on non–linear correlations, CNS (Consistency–based filter selection) and wrappers [36]. Although the classification accuracy increased using FSS, few common metrics were selected from the datasets. This can also be observed in other works such as [12]. From the software engineering point of view, these very heterogeneous results were confusing when used to generate predictive models, for example, giving greater importance to metrics such as the number of blank lines or number of commented lines than more intuitive metrics such as complexity. Another alternative to the use of FSS is to consider weights in conjunction with the attributes. For example, Turhan and Bener [53] reports positive results with the application of Infogain [49] to the naïve Bayes classifiers in software defect prediction.

### 2.2. Subgroup discovery

Subgroup Discovery (SD) aims to find subgroups of data that are statistically different given a property of interest [30,58,59,21]. SD lies between predictive (finding rules given historical data and a property of interest) and descriptive tasks (discovering interesting patterns in data). An important difference with classification tasks is that the SD algorithms only focus on finding subgroups (e.g., inducing rules) for the property of interest and do not necessarily describe all instances in the dataset.

In general, subgroups are represented through rules with the form \( \text{Cond} \rightarrow \text{Class} \) having as consequent \( \text{Class} \) a specific value of an attribute of interest. The antecedent \( \text{Cond} \) is usually composed of a conjunction of attribute–value pairs through relational operators. Discrete attributes can have the form of \( \text{att} = \text{val} \) or \( \text{att} \neq \text{val} \) and for continuous attributes ranges need to be defined, i.e., \( \text{val}_1 \leq \text{att} \leq \text{val}_2 \).

An important aspect of SD is how to measure the quality of the rules that define the subgroups in order to both (i) guide the search process of the SD algorithms and (ii) to compare them. To do so, we first describe standard measures used in classification and later modifications to those measures in SD.

One common way to evaluate the performance of classifiers is through the values of the confusion matrix [19]. Table 1 shows the possible outcomes for two classes; \( \text{True Positives (TP)} \) and \( \text{True Negatives (TN)} \) are respectively the number of positive and negative instances correctly classified, \( \text{False Positives (FP)} \) is the number of negative instances misclassified as positive, and \( \text{False Negatives (FN)} \) is the number of positive instances misclassified as negative.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>Negative</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

- Type I error (False alarm)
- Recall = Sensitivity = \( \frac{TP}{TP+FN} \)
- Specificity = \( \frac{TN}{TN+FP} \)

There is a trade–off between \( \text{true positive rate} \) and \( \text{true negative rate} \) as the objective is to maximise both metrics. They can be combined to form single metrics. For example, the predictive accuracy (Acc) is defined as:

\[ \text{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \]
Another widely used metric when measuring the performance of classifiers is the \( f - \text{measure} \) \cite{57} as an harmonic median of these two proportions:

\[
f - \text{measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}.
\] (2)

where \( \text{precision} = \frac{TP}{TP + FN} \) is the proportion of positive predictions that are correct and \( \text{recall} \) is the \( TP \), previously defined.

There are also some classification measures adapted to rules and SD. We next describe the most widely used measures for SD evaluation \cite{19,21}:

- **Coverage of a rule (Cov)** is the percentage of instances covered by a rule of the induced set of rules

\[
\text{Cov}(R_i) = p(\text{Cond}) = \frac{n(\text{Cond})}{N} = \frac{TP + FP}{N},
\] (3)

where \( R_i \) is a single rule, \( n(\text{Cond}) \) is the number of instances covered by condition \( \text{Cond} \) and \( N \) is the total number of instances.

- **The Support of a rule** refers to the ratio between the number of instances satisfying both the antecedent and the consequent part of a rule and the total number of instances.

\[
\text{Sup}(R_i) = \frac{n(\text{Cond} \cdot \text{Class})}{N} = \frac{TP}{N},
\] (4)

where the \( n(\text{Cond} \cdot \text{Class}) \) corresponds to the \( TP \) and \( N \) is the total number of instances.

- **The Specificity** is the proportion of negative cases correctly classified.

\[
\text{Spec}(R_i) = \frac{-n(\text{Cond} \cdot \text{Class})}{-n(\text{Class})} = \frac{TN}{FP + TN},
\] (5)

where the \( -n(\text{Cond} \cdot \text{Class}) \) corresponds to instances which do not satisfy both condition and target class (\( TN \)). The \( n(\text{Class}) \) is the number of instances that satisfy the target class and \( -n(\text{Class}) \) is the number of those that do not satisfy the target class.

- **The Complexity** refers to the number of tests or antecedents (conjunction attribute-value pairs) in the condition (\( \text{Cond} \)) of a single rule.

- **Confidence (Conf)**, also known as Precision or Positive Predictive Value (PPV) of a rule is the percentage of positive instances of a rule, i.e. relative frequency of the number of instances satisfying the both the \( \text{Cond} \) and the target \( \text{Class} \) and the number of instances satisfying the condition.

\[
\text{Conf}(R_i) = \frac{n(\text{Cond} \cdot \text{Class})}{n(\text{Cond})} = \frac{TP}{TP + FP}.
\] (6)

- **Rule Unusualness** is measured through the Weighted Relative Accuracy (WRAcc).

\[
\text{WRAcc}(R_i) = \frac{n(\text{Cond})}{N} \cdot \left( \frac{n(\text{Cond} \cdot \text{Class})}{n(\text{Cond})} - \frac{n(\text{Class})}{N} \right).
\] (7)

This measure represents a trade–off between the coverage of a rule, i.e., its generality \( (p(\text{Cond})) \) and its accuracy gain \( (p(\text{Cond} \cdot \text{Class}) - p(\text{Class})) \).

- **Significance for a rule** is measured by the likelihood ratio of a rule.

\[
\text{Sig}(R_i) = 2 \cdot \sum_{k=1}^{n_v} n(\text{Cond} \cdot \text{Class}_k) \cdot \log \frac{n(\text{Cond} \cdot \text{Class}_k)}{n(\text{Class}_k) \cdot p(\text{Cond})}.
\] (8)

where \( n_v \) is the number of values of the target class. Therefore, considering a binary problem as in this case:

\[
\text{Sig}(R_i) = 2 \cdot \left( \text{TP} \cdot \log \frac{\text{TP}}{\text{Def} \cdot \left( \frac{\text{TP} + \text{FP}}{N} \right)} + \text{TN} \cdot \log \frac{\text{TN}}{\text{NonDef} \cdot \left( \frac{\text{TP} + \text{FP}}{N} \right)} \right).
\] (9)

where \( \text{Def} \) is the number of faulty modules in the dataset and \( \text{NonDef} \) is the number of non-defective modules contained in the dataset.

- **Lift** (also known as interest) measures how many times more often the \( \text{Cond} \) and the \( \text{Class} \) occur together than expected if they were statistically independent.

\[
\text{Lift}(R_i) = \frac{\text{Conf}(R_i)}{\text{Sup}(\text{Cond})} = \frac{p(\text{Cond} \cdot \text{Class})}{p(\text{Cond}) \cdot p(\text{Class})} = \frac{\text{TP} \cdot N}{(\text{TP} + \text{FP}) \cdot \text{Def}}.
\] (10)
Currently, a number of SD algorithms have been proposed since the concept was introduced by Wrobel [58] with the EXPLORA algorithm. A comprehensive survey of SD algorithms can be found in Herrera et al. [21]. In this work, we also compare our algorithm, EDER-SD, described in the next section with the following classical algorithms.

The Subgroup Discovery algorithm SD [18] is a covering rule induction algorithm that using beam search aims to find rules that maximise \( q_g = \frac{TP}{TP + FP} \), where \( g \) is a generalisation parameter that allow us to control the specificity of a rule, i.e., the balance between the complexity of a rule and its accuracy.

The CN2-SD [32] algorithm is an adaptation of the CN2 classification rule algorithm [11]. CN2-SD, like the original algorithm, consists of a search procedure based on beam search but the CN2-SD algorithm uses unusualness (WRAcc) as a quality measure of the induced rules and incorporates weights into the samples. Discretisation is also required for continuous attributes and an entropy based discretisation method (entropy MDL – Minimum Description Length) is used internally by the algorithm so different rules can have different ranges for the same attribute. Therefore, there is no need to discretise continuous attributes as a preprocessing step.

APRIORI-SD [24] is also an adaptation of the APRIORI-C [22] which in turn is a modification for classification tasks of the well-known rule APRIORI association algorithm [1].

There are other approaches to SD that do not fit with the representation needed for defect prediction (e.g. Železný and Lavrač [56] describe how relational rule learning is adapted to subgroup discovery) or improvements to these algorithms that are out of the scope of this paper (e.g., Cano et al. [7] make the previous CN2-SD algorithm scalable to large size datasets using instance selection).

### 3. EDER-SD

In this work, we propose EDER-SD (Evolutionary Decision Rules for Subgroup Discovery), an Evolutionary Algorithm (EA) [41] to characterise the minority class. EDER-SD is a robust algorithm written in C++ capable of dealing with imbalanced data that generates rules only for the defective modules, the class we are interested in.

To do so, we modified HIDER (HIerarchical DEcision Rules) [2], a sequential covering EA that produces a hierarchical set of rules, i.e., an instance will be classified by the \( i \)-th rule if it does not match the conditions of the \( (i-1) \)-th precedent rules (Fig. 1). The rules are sequentially obtained until the search space is totally covered.

In order to apply EAs to optimisation problems, we need to (i) select an internal representation of the space to be searched in and (ii) define a function that assigns fitness to candidate solutions. Both components are of paramount importance for the successful application of the EAs to the problem of interest.

The representation of an individual, i.e., a rule in our case, in HIDER is a tuple of real values as is shown in Fig. 2. The \( l_i \) and \( u_i \) values represent the interval for the \( i \)-th attribute. The last position is the label representing the Class. When \( l_i \) for an attribute \( a_i \) is equal to its minimum value (\( l_i = \min(a_i) \)) such constrain will not be part of the rule and equally when the \( u_i \) value for an attribute is equal to its maximum value (\( u_i = \max(a_i) \)). For example, in the first case the rule would be \([l_i, v]\) and in the second case \([v, u_i]\), where \( v \) is any value within the range of the attribute. If both values are equal to their respective boundaries, it means that the attribute is not relevant and will not appear in the rule. EDER-SD maintains the same representation for individuals.

HIDER follows an Iterative Rule Learning methodology such that in each iteration an execution of the EA is performed to induce a rule [55]. Instances covered by the induced rule after an execution of the EA are removed from the dataset for the next execution so that only new instances will be covered (and as a consequence, it produces the hierarchy due to fact that rules need to be applied in the induced order). This process is repeated until all instances are covered. HIDER is an algorithm that induces a set of decision rules for all classes. Therefore, in the random generation of the individuals of the initial

![Fig. 1. Hierarchical set of rules (HIDER).](image)

![Fig. 2. Representation of rules.](image)
population individuals are selected to cover all existing labels, and it is the evolutionary process which will provide the best option (rule) in each iteration.

The main differences between EDER-SD and HIDER so that EDER-SD is suitable for subgroup discovery are as follows. The first one is in the process of selecting the initial population at the beginning of each evolutionary process. HIDER generates individuals (rules) for each class, however EDER-SD does not search for rules to classify the all data but only to describe one particular class. In EDER-SD, the generation of each rule in the initial population randomly selects an instance that corresponds to the class of interest (a defective module in this case). From this instance, a rule is generated so that each interval \([l_i, u_i]\) includes the attribute values for that instance. More formally, let \(e\) be an instance from the training dataset with label \(\text{def}_\text{module}\) and attribute values \((a_1, a_2, \ldots, a_n)\). Then, \(n\) weights are randomly generated \(w_i \in [0, 1]\). Finally, a rule is generated as follows:

\[
\begin{align*}
l_i &= a_i - w_i \cdot \text{range}(a_i), \\
u_i &= a_i + w_i \cdot \text{range}(a_i), \\
class &= \text{def}_\text{module},
\end{align*}
\]

where \(\text{range}(a_i)\) is the \(\max(a_i) - \min(a_i)\) for each attribute.

The second difference lies in the fact that HIDER removes the covered instances from the training data after each execution of the EA (i.e., the instances covered by the rule selected) and as a consequence, the induced rules are hierarchical. EDER-SD, on the other hand, does not remove instances from the training file but those instances covered by the rule are penalised. Adding weights to instances instead of removing them produces the induction of more complex rules (i.e., with larger number of conditions) increasing the precision of the rules. This can also generate pyramidal rules, where the first rule (with fewer conditions) covers a large number of instances (high support) but with low precision. The rest of the rules in the pyramid keep adding conditions decreasing the support but increasing the precision.

The final difference between HIDER and EDER-SD is related to the fitness function. While in the original HIDER, the fitness function used is to maximise accuracy, in EDER-SD the fitness function can be any of the metrics used in SD, such as \(W\text{RAcc}\) or \(\text{Lift}\). Thus, rules are adapted to whatever measure is the most adequate in the domain. Fig. 3 shows the EDER-SD algorithm where the classification algorithm was transformed into an algorithm for the extraction of subgroups to generate descriptive models based on rules.

---

**Procedure** EDER-SD(E,R)

\(E\) is instance set \(\cup\) Weights
\(R\) is the set of rules
\(c\) is the target class
\(P\) is the population

\[
\begin{align*}
\text{\(R := \emptyset\)} \\
\text{\(E' := E\)} \\
\text{while \(|E'| > |\{e \in E | \text{class}(e) == c\}|\)} \\
\text{\(r := \text{EvoAlg}(E')\)} \\
\text{\(R := R \oplus \{r\}\)} \\
\text{\(E' := \text{modifyWeights}(E', R)\)} \\
\end{while}
\]

**Procedure** EvoAlg(E)

\(i := 0\)
\(P_0 := \text{Initialise}(E)\)
\(E\text{valuation}(P_0, E)\)

\[
\text{while } i < \text{num\_generations} \\
\text{\(i := i + 1\)} \\
\text{for } j \in \{1, \ldots, |P_{i-1}|\} \\
\text{\(x := \text{Selection}(P_{i-1}, i, j)\)} \\
\text{\(P_i := P_i \oplus \text{Recombination}(x, P_{i-1}, i, j)\)} \\
\text{end\_for} \\
\text{E\text{valuation}(P_i, E)} \\
\text{end\_while} \\
\text{return \text{best\_of}(P_i)}
\]

---

*Fig. 3.* EDER-SD algorithm.
The EDER-SD procedure works over a set of weighted instances whose weights are initialised to one. Each execution of the evolutionary algorithm \( \text{EvoAlg} \) generates one rule, \( r \), which is added to the total set of rules, \( R \). The weights of the covered instances are decreased by 10\% each time and will be used to evaluate the fitness of posterior rules (Evaluation procedure within \( \text{EvoAlg} \)). The functions inside the \( \text{EvoAlg} \) procedure are the classical evolutionary algorithm functions:

- The \textit{Initialise} function generates an initial set of rules covering a set of the target class (as previously explained).
- The \textit{Evaluation} function assigns to each rule a value according to its fitness. As stated previously, different evaluation functions can be selected for each execution such as accuracy, sensitivity, significance, \( f \)-measure, lift or WRAcc. In order to calculate such measures, the weighted instances are used for the \( n(\text{Cond}) \) and \( -n(\text{Cond}) \) expressions. The \( n(\text{Cond}) \) represents the sum of weights that satisfy the condition (antecedent of the rule) and the \( n(\text{Cond} \cdot \text{Class}) \) is the sum of those instances that satisfy the condition and belong to the target class. Therefore, during the first iteration of the \( \text{EvoAlg} \) procedure, all instances are equally considered. However, as more iterations are performed, the fitness values of the instances already covered are penalised.
- The \textit{Selection} function selects the rules in a generation to be recombined according to their fitness measure. As in the HIDER algorithm, EDER-SD uses the roulette-wheel algorithm.
- The \textit{Recombination} function is the crossover operator. We have also applied the real code crossovers [2], an extension of BLX-alpha adapted to individuals coded as interval [13].
- Finally, the \textit{best_of} function returns the best rule according to the fitness measure used.

4. Experimental work

In this section we firstly describe the datasets used in this work. Secondly, we present the induced rules characterising defective modules for each of the datasets. Thirdly, we compare the rules induced using EDER-SD and three other well-known SD algorithms as well as a validation considering a splitting criterion. Finally, threats to validity of the empirical work are considered.

4.1. Datasets

In this paper, we have used the CM1, KC1, KC2, KC3, MC2, MW1 and PC1 datasets available in the PROMISE repository [5], to generate models for defect classification. These datasets were created from projects carried out at NASA.

Table 2 shows the number of instances for each dataset, the number of defective, non-defective modules and their percentage, number of duplicates, inconsistencies (equal values for all attributes of an instance but the class) and programming language. It can be observed that all datasets are highly imbalanced, varying from approximately 7\% to 20\% with a large number of duplicate instances.

All datasets contain the same 22 attributes composed of 5 different metrics for lines of code, 3 McCabe metrics [37], 4 base Halstead metrics [20] and 8 derived Halstead metrics that have been discarded (see Subsection 2.1), a branch–count, and the last attribute is problems with 2 classes (whether a module has reported defects). Table 3 summarizes the metrics selected from the datasets in this study.

The McCabe metrics are based on the count of the number of paths contained in a program based on its graph. To find the complexity, the program, module or method of class in an object oriented program is represented as a graph, and its complexity is calculated as \( n(g) = e - n + 2 \), where \( e \) is the number of edges of the graph and \( n \) is the number of nodes in the graph.

The cyclomatic complexity metric measures quantity, but McCabe also defined \textit{essential complexity}, \( e(V(g)) \), to measure the quality of the code (penalising what is known as \textit{spaghetti} code). Structured programming only requires sequences, selection and iteration structures, and the essential complexity is calculated in the same manner as cyclomatic complexity but from a simplified graph where such structures have been removed. The \textit{design complexity} metric (\( \text{dit}(g) \)) is similar but takes into account the calls to other modules.

The other metrics used in this experiment are the Halstead’s \textit{Software Science} metrics. They are based on simple counts of tokens grouped into (i) \textit{operators} such as keywords from programming languages, arithmetic operators, relational operators and logical operators and (ii) \textit{operands} that include variables and constants.

These sets of metrics (both McCabe and Halstead) have been used for quality assurance during (i) development to obtain quality measures, code reviews etc., (ii) testing to focus and prioritize testing effort, improve efficiency etc. and (iii) and maintenance as indicators of comprehensibility of the modules etc. Generally, the developers or maintainers use rules of thumb or threshold values to keep modules, methods etc. within certain range. For example, if the cyclomatic complexity (\( n(g) \)) of a module is between 1 and 10, it is considered to have a very low risk of being defective; however, any value greater than 50 is considered to have an unmanageable complexity and risk. For the essential complexity (\( e(V(g)) \)), the threshold suggested is 4 etc. Although these metrics have been used for long time, there are no clear thresholds, for example, although McCabe suggests a threshold of 10 for \( n(g) \), NASA’s in–house studies for this metric concluded that a threshold of 20 can be a better predictor of a module being defective.

\(^3\) \url{http://mdp.ivv.nasa.gov/}.
4.2. Rules found with EDER-SD

We next show the most relevant rules obtained with our tool EDER-SD for each dataset. In order to run the genetic algorithms, we need to define a number of parameters. For all datasets, the population size was 100 individuals and each execution of the evolutionary algorithm ran 100 generations. We must also define a minimum support which is approximately 10% of the number of defective modules contained in the dataset. The rules presented here are the result of several executions with different fitness functions but for the sake of brevity and space, only the most relevant ones are shown.

4.2.1. CM1 dataset

As it can be seen in Table 4 in relation to the CM1 dataset, the ratio between defective and non-defective modules for the first rule is about 26% (22/84). Although it seems a low value, it is worth noting that the percentage of unbalance of this dataset is 10%, with only 49 defective modules out of 498 modules contained in the dataset. Therefore, the probability of finding a defective module has been increased considerably. This rule was obtained by maximising \( \text{sensitivity}(0.44) \).

The second and third rules cover fewer modules than the first rule but the ratio between defective and non-defective modules increases to 38% and 43% respectively. These rules were obtained optimising \( \text{accuracy}(0.89) \) or \( \text{lift}(4.35) \) and rule 3 can also be obtained maximising the \( f/C_0 \) measure (0.36).

Rules 4 to 6 show the effect of decreasing the weights of instances already covered by the EA, adding a new condition to the precedent rule and showing a pyramidal effect. Rule 4 (obtained by maximising the \( f/C_0 \) measure) just considers \( \text{LoC} \) as single condition achieving a precision of 38%, which means that almost 40% of the modules with more than 71 \( \text{LoC} \) will be defective. Such a threshold is relatively close to the 60 \( \text{LoC} \) suggested by the McCabe IQ tool and the NASA repository.  

---

4 http://www.mccabe.com/.
being defective. The limits found by EDER-SD are 93 and 17 respectively, maximising specific- 
ity (0.4), decreases (41 and 30 modules respectively out of the 498 contained in the dataset). These rules can be obtained maximising 

In rules 5 and 6, when new conditions are added, precision increases (64% and 67% for rules 5 and 6 respectively) but support decreases (41 and 30 modules respectively out of the 498 contained in the dataset). These rules can be obtained maximising precision (0.4), WRAcc (0.25) or significance (11.17).

4.2.2. KC1 dataset

As shown by the first two rules in Table 5, modules with a large number of LoC or uniqOp have a higher probability of being defective. The limits found by EDER-SD are 93 and 17 respectively, maximising specificity (0.98). Although the number of defective modules covered by the rules is larger than the non defective, both rules have low support as they cover a relatively small number of modules: 73 for the first rule with a single condition (93 

EDER-SD also found rules combining lines of code and complexity. For modules with large complexity but a relatively small number of LoC, the probability of the module being defective increases. For example, for rule 3 in Table 5 (4 \leq t(g) and 69 \leq totalOpnd), its ratio is 58% (76 out of 130). However, when the size of the module is limited to 78 LoC, the ratio of defective modules is 72%. In other words, rule 4 states that small modules with high complexity tend to be fault-prone. This rule was obtained by maximising precision (0.73) whereas rule 3 was obtained by maximising either accuracy (0.86) or significance (54.48).

The combined threshold values for eV(g) and t(g) complexity metrics are 3 and 4 respectively (rule 5 obtained maximising sensitivity (0.3)), achieving a ratio of 38% for defective modules. Adding a new constraint about unique operators (17 \leq uniqOp) to these complexity values increases the ratio to 50% with more than 70 modules covered by the rule. For this dataset, the ratio of defective vs. non defective modules is just 15%, therefore the probability of finding a defective module also increases considerably when compared with random selection.

Other rules found by EDER-SD relate the number of branches with unique operators. The rule (5 \leq branchCount) covers 204 defective modules out of 598 modules (34%) but if this threshold is increased to 9, the ratio also increases to 40%. However, when considering both conditions (9 \leq branchCount \land 9 \leq uniqOp), the rule covers 194 defective modules (almost 2/3 of the 313 defective samples included in the dataset) with ratio of 60%. That is to say that modules with branch count and unique operator values larger than 9 have a 60% of probability of being defective. The last three rules can be obtained by maximising the $f$–measure (0.44), significance (54.94) or WRAcc (0.05).

4.2.3. KC2 dataset

The first rule for the KC2 dataset in Table 6 combines three parameters (LoC, eV(g) and branchCount), which are very close to the thresholds suggested by the McCabe IQ tool and the empirical values from the NASA repository (60, 4 and 19.

### Table 5
Selected EDER-SD rules for KC1 dataset.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$93 \leq LoC \land 17 \leq t(g) \land 17 \leq branchCount$</td>
<td>40</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>$93 \leq LoC \land 17 \leq uniqOp$</td>
<td>39</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>$4 \leq t(g) \land 69 \leq totalOpnd$</td>
<td>76</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>$4 \leq eV(g) \land 69 \leq totalOpnd \land LoC \leq 78$</td>
<td>27</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>$3 \leq eV(g) \land 4 \leq t(g)$</td>
<td>100</td>
<td>159</td>
</tr>
<tr>
<td>6</td>
<td>$3 \leq eV(g) \land 4 \leq t(g) \land 17 \leq uniqOp$</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>$5 \leq branchCount$</td>
<td>204</td>
<td>394</td>
</tr>
<tr>
<td>8</td>
<td>$9 \leq branchCount$</td>
<td>134</td>
<td>196</td>
</tr>
<tr>
<td>9</td>
<td>$9 \leq branchCount \land 9 \leq uniqOp$</td>
<td>134</td>
<td>194</td>
</tr>
</tbody>
</table>

### Table 6
Selected EDER-SD rules for KC2 dataset.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$58 \leq LoC \land 5 \leq eV(g) \land 17 \leq branchCount$</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>$17 \leq uniqOp$</td>
<td>85</td>
<td>78</td>
</tr>
<tr>
<td>3</td>
<td>$17 \leq uniqOp \land 30 \leq totalOpnd$</td>
<td>82</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>$17 \leq uniqOp \land 50 \leq totalOpnd$</td>
<td>70</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>$30 \leq totalOpnd$</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>6</td>
<td>$71 \leq totalOpnd$</td>
<td>51</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>$120 \leq totalOpnd$</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>$9 \leq t(g)$</td>
<td>46</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>$9 \leq t(g) \land 4 \leq eV(g)$</td>
<td>42</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>$9 \leq t(g) \land 4 \leq eV(g) \land 5 \leq i(t(g)$</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>$9 \leq t(g) \land 4 \leq eV(g) \land 5 \leq i(t(g) \land 75 \leq totalOpnd$</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>$9 \leq t(g) \land 4 \leq eV(g) \land 5 \leq i(t(g) \land uniqOp \leq 24 \land 75 \leq totalOpnd$</td>
<td>31</td>
<td>4</td>
</tr>
</tbody>
</table>
4.2.5. MC2 dataset

The rules are composed of a large number of conditions (with the exception of rules 4 and 7). In this case, adding the $50 \leq \text{totaloperands}$ condition decreases the number of modules covered from 163 (rule 2) to 111 (rule 4), but rule’s precision is increased from 52% (82 out of 163) to 63% (60 out of 111). Rules 2 to 4 were obtained maximising the $f –\text{measure}$ (0.64), accuracy (0.85) or WRAcc (0.99), rules 5 and 6 sensitivity (0.78), and specificity (0.99) for rule 7.

Rules 8 to 12 do not modify the limits but keep adding conditions to the precedent rule creating the pyramidal effect previously mentioned. Again, there is a trade-off between support and precision, slightly reducing the number of modules covered by the rule, increases the probability of the module being defective. These rules were obtained maximising the lift (4.32).

4.2.4. KC3 dataset

Table 7 shows the selected rules for the KC3 dataset. The first two rules are obtained using measures that favour high support, e.g., $TP_r$ or WRAcc. With such measures, the rules obtained cover a large number of defective modules (32 out of 43) but penalising accuracy and specificity measures. On the other hand, the rest of the rules try to maximise the latter measures resulting in a very low rate of false positives at the expense of a low support (only seven defective modules). In addition, the rules are composed of a large number of conditions (with the exception of rules 4 and 7).

4.2.5. MC2 dataset

The selected rules for the MC2 dataset are shown in Table 8. As with the KC3 dataset, the first three rules maximise WRAcc or $TP_r$, while the remaining rules maximise Specificity or Precision. However, it is worth noting that the differences between the accuracy values in both sets of rules are very small, i.e., 0.6 to 0.7 for the first set and 0.7 to 0.75 for the second one. Rules 6, 7 and 8 maximise accuracy and significance resulting in similar values for $\text{UniqOp}$ and $\text{UniqOpnd}$.

4.2.6. MW1 dataset

Table 9 shows the results for the MW1 dataset. Again, the first two rules maximise WRAcc and $TP_r$, and the rest of the rules lift or precision. However, the differences of the values of the accuracy measure are also very small (0.88 for the first of rules and 0.94 for the rest of the rules). The same occurs in specificity (0.9 vs. 0.98) or significance (12 vs. 15). As with the previous dataset, rules with very low values of false positives are composed of a large number of conditions.

### Table 7

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$3 \leq \text{if}(g) \land 12 \leq \text{uniqOp} \land 40 \leq \text{totalOp}$</td>
<td>32</td>
<td>94</td>
</tr>
<tr>
<td>2</td>
<td>$20 \leq \text{loc} \land 3 \leq \text{if}(g)$</td>
<td>25</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>$79 \leq \text{loc} \land 10 \leq \text{if}(g) \land \text{uniqOp} \leq 25 \land \text{totaloperands} \leq 308 \land \text{branchCount} \leq 41$</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$79 \leq \text{loc} \land 10 \leq \text{if}(g) \land \text{uniqOp} \leq 25$</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>$3 \leq \text{if}(g) \land \text{vOp}(g) \leq 3 \land \text{uniqOp} \leq 20 \land 32 \leq \text{UniqOpnd}$</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$88 \leq \text{loc} \land 20 \leq \text{uniqOp} \land \text{UniqOpnd} \leq 106 \land \text{branchCount} \leq 41$</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$\text{vOp}(g) = 1 \land 58 \leq \text{UniqOpnd}$</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>$57 \leq \text{loc} \land \text{vOp}(g) \leq 3 \land 58 \leq \text{UniqOpnd}$</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>$\text{vOp}(g) = 1 \land \text{if}(g) \leq 12 \land \text{uniqOp} \leq 14 \land 40 \leq \text{totalOp} \land \text{totaloperands} \leq 39$</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>$10 \leq \text{if}(g) \land \text{uniqOp} \leq 25 \land 48 \leq \text{UniqOpnd} \land 22 \leq \text{branchCount}$</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 8

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2 \leq \text{if}(g) \land 15 \leq \text{uniqOp}$</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>$15 \leq \text{uniqOp}$</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>$5 \leq \text{if}(g) \land 2 \leq \text{if}(g)$</td>
<td>30</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>$14 \leq \text{uniqOp} \land 11 \leq \text{UniqOpnd} \land \text{totaloperands} \leq 38$</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>$9 \leq \text{loc} \land 15 \leq \text{UniqOpnd} \land 31 \leq \text{totalOp}$</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$32 \leq \text{loc} \land 5 \leq \text{vOp}(g) \land 2 \leq \text{if}(g) \land 18 \leq \text{uniqOp}$</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>$8 \leq \text{if}(g) \land 3 \leq \text{if}(g)$</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>$8 \leq \text{if}(g) \land 137 \leq \text{totaloperands}$</td>
<td>17</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>$\text{totaloperands} \leq 24 \land 11 \leq \text{totaloperands} \land \text{branchCount} \leq 1$</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
4.2.7. PC1 dataset

Table 10 shows rules for the PC1 dataset. The first rule shows the threshold found by EDER-SD relating for the cyclomatic complexity and metrics related to operands and operators. This rule covers a large proportion of the defective modules (62%) contained in the dataset but provides a low precision of about 16%. It is worth noting that this dataset is the most imbalanced one with only 77 samples of defective modules out of 1109 (7% of the total) and therefore when this rule applies, the probability of finding a defective module is more than double than using random selection.

The second rule establishes the limits for the design complexity (\( \text{i}x(g) \)) and the number of branches (\( \text{branchCount} \)). Like the previous one, this rule covers a large number of the defective modules (53% of the total) but has low precision (14%).

EDER-SD found two limits for the number of lines of code \( \text{LoC} \) 64 and 84, as shown by the third and fourth rules. The latter limit is more restrictive providing higher precision with a relatively low support as it covers 16 defective modules out of 77 (21% of the total). Finally, the rest of the rules are further examples of rules providing high precision but low support.

4.2.8. General observations across all datasets

Each dataset provides examples of concrete applications within a domain (real-time, instruments, control systems) and programming language (C, C++ or Java). Thus, it is difficult to extrapolate the rules found for any of the datasets. It is possible, however, to observe some common trends. As it can be observed from Tables 4–10, almost all induced rules are formed by conditions providing lower limits for each of the metrics that compose the rules. As expected, we found defective modules for high values of the metrics (e.g., lines of code, complexity, number of operands, operators and branches). This is especially true for rules with high support values in each of the datasets. After analysing the rules, it can be seen in general that the limit for \( \text{LOC} \) is 60; for the complexity measures, the limit of \( \text{i}x(g) \) is between 4 and 9, and for \( \text{i}x(g) \) is from 2 to 4. The number of operators and operands have wider ranges (\( \text{uniqOp} \): 10–15, \( \text{uniqOpnd} \): 17–38, \( \text{totalOp} \): 40, \( \text{totalOpnd} \): 29–69). Finally, the lower limit for the \( \text{branchCount} \) metric is between 5 and 7. On the rare occasions where an upper limit appears, the number of defective modules covered by those rules (support) is very low, which means that they are very specific rules adjusted to a particular subset of the data (e.g., high complex modules with very few operators) and they are also very difficult to generalise.

It is worth noting the consistency of the values established by EDER-SD for the lower threshold values in relation to those obtained by McCabe IQ tool and the NASA MDP web site (\( \text{LOC} \): 60, \( \text{i}x(g) \): 10, \( \text{evG} \); 4, \( \text{i}x(g) \); 7, \( \text{uniqOp} \); 20, \( \text{uniqOpnd} \); 20, \( \text{totalOp} \); 30, \( \text{totalOpnd} \); 30, \( \text{branchCount} \); 19) but such thresholds set by the McCabe IQ tool are meant to be used individually. However, EDER-SD thresholds are delineated in several dimensions (metrics), allowing better adjustment of the values that lead to faulty modules.

4.3. Comparison of EDER-SD with other algorithms

As stated previously, defect prediction datasets are in general highly imbalanced and with a large number of inconsistencies (duplicates or contradictory cases). To deal with these problems using SD, we have developed an evolutionary algorithm called EDER-SD that aims to find simple and easily understandable rules (few conditions) but capable of predicting
preformed better in five out of the seven datasets for most evaluation measures. The best results were obtained in the algorithm in 39 out of 70 evaluation measures, similar results in 14 and worst results in 17 evaluation measures. It also equal than the other algorithms in their respective metrics). Table 14 shows that the EDER-SD performed better than the SD it obtained maximum values (the No. of pluses, minuses and equals in Table 14 shows if EDER-SD performed better, worst or optimise different functions. EDER-SD performed better than the other three algorithms counting the number of times that can be used before removed, $k$ covered by a rule is not enough to fulfil the quality criteria. Furthermore, the APRIORI-SD algorithm not only discretises con-

Tables 11–13 show the set of rules induced for the KC2 dataset using the default parameters used in the Orange tool (minimal support = 5%; minimal precision = 80%; beam width = 20; generalization parameter, $g = 5$ and No. of times a covered instance can be used before removed, $k = 5$). We do not show the rules for rest of the datasets for the sake of brevity and space. These three algorithms are deterministic when the parameters are fixed.

The first observation is the disparity in the number of rules induced. The CN2-SD algorithm generates very few rules for all datasets (just two or three rules) but those rules are in general quite good. In order to validate EDER-SD against these classical SD algorithms from a machine learning perspective, we also used a splitting criterion. We divided the datasets into training and testing with two-thirds and one-third of the samples respectively. Table 14 shows the comparative results for the maximum values obtained for the evaluation measures described in Section 2.2 using all algorithms and testing datasets, $\text{beam width} = 20$. generalization parameter, $g = 5$ and No. of times a covered instance can be used before removed, $k = 5$). The reason may reside in the fact that datasets are so highly imbalanced that rules do not achieve the minimum support and precision required, i.e., the number of samples covered by a rule is not enough to fulfil the quality criteria. Furthermore, the APRIORI-SD algorithm not only discretises continuous attributes but discrete ones are also binerised.

Tables 11–13 show the set of rules induced for the KC2 dataset using the default parameters used in the Orange tool (minimal support = 5%; minimal precision = 80%; beam width = 20; generalization parameter, $g = 5$ and No. of times a covered instance can be used before removed, $k = 5$). We do not show the rules for rest of the datasets for the sake of brevity and space. These three algorithms are deterministic when the parameters are fixed.

The first observation is the disparity in the number of rules induced. The CN2-SD algorithm generates very few rules for all datasets (just two or three rules) but those rules are in general quite good. In order to validate EDER-SD against these classical SD algorithms from a machine learning perspective, we also used a splitting criterion. We divided the datasets into training and testing with two-thirds and one-third of the samples respectively. Table 14 shows the comparative results for the maximum values obtained for the evaluation measures described in Section 2.2 using all algorithms and testing datasets, where the $+$, $-$ and $=$ corresponds to EDER-SD performing better, worst or equal to other algorithm respectively. The reason for using maximum values is that all algorithms induce a very different number of rules (CN2-SD induces only two or three rules for these datasets). Therefore, it would not be fair to compare average values. Furthermore, the different algorithms optimise different functions. EDER-SD performed better than the other three algorithms counting the number of times that it obtained maximum values (the No. of pluses, minuses and equals in Table 14 shows if EDER-SD performed better, worst or equal than the other algorithms in their respective metrics). Table 14 shows that the EDER-SD performed better than the SD algorithm in 39 out of 70 evaluation measures, similar results in 14 and worst results in 17 evaluation measures. It also preformed better in five out of the seven datasets for most evaluation measures. The best results were obtained in the

### Table 11
SD induced rules for the KC2 dataset.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$eV(g) &gt; 4 \land totalOpnd &gt; 117$</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>$iV(g) &gt; 8 \land uniqOpnd &gt; 34 \land eV(g) &gt; 4$</td>
<td>31</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>$loc &gt; 100 \land uniqOpnd &gt; 34 \land eV(g) &gt; 4$</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>$loc &gt; 100 \land iV(g) &gt; 8 \land eV(g) &gt; 4$</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>$loc &gt; 100 \land iV(g) &gt; 8 \land totalOpnd &gt; 117$</td>
<td>27</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>$iV(g) &gt; 8 \land uniqOp &gt; 11 \land totalOp &gt; 80$</td>
<td>33</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>$iV(g) &gt; 8 \land uniqOpnd &gt; 34$</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>totalOpnd &gt; 117</td>
<td>31</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>$loc &gt; 100 \land iV(g) &gt; 8$</td>
<td>31</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>$eV(g) &gt; 4 \land iV(g) &gt; 8$</td>
<td>32</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>$eV(g) &gt; 4 \land uniqOpnd &gt; 34$</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>$loc &gt; 100 \land eV(g) &gt; 4$</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>$iV(g) &gt; 8 \land uniqOp &gt; 11$</td>
<td>34</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>$eV(g) &gt; 4 \land totalOp &gt; 80 \land iV(g) &gt; 6 \land uniqOp &gt; 11$</td>
<td>46</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>$iV(g) &gt; 8 \land totalOp &gt; 80$</td>
<td>35</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>$eV(g) &gt; 4 \land totalOp &gt; 80 \land uniqOp &gt; 11$</td>
<td>46</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>$eV(g) &gt; 4 \land totalOp &gt; 80 \land iV(g) &gt; 6$</td>
<td>47</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>$loc &gt; 100 \land uniqOpnd &gt; 34$</td>
<td>34</td>
<td>9</td>
</tr>
<tr>
<td>19</td>
<td>$eV(g) &gt; 4 \land totalOp &gt; 80$</td>
<td>47</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>$iV(g) &gt; 8$</td>
<td>35</td>
<td>13</td>
</tr>
</tbody>
</table>

### Table 12
CN2-SD induced rules for the KC2 dataset.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>uniqOpnd &gt; 34 \land eV(g) &gt; 4</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>totalOp &gt; 80 \land eV(g) &gt; 4</td>
<td>47</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>uniqOp &gt; 11</td>
<td>86</td>
<td>117</td>
</tr>
</tbody>
</table>

fault-prone modules. In this subsection, we now compare EDER-SD with three other well-known SD algorithms widely cited in the literature, SD, CN2-SD and APRIORI-SD (see Section 2.2). These algorithms have been implemented in the Orange data mining toolkit\(^5\) as part of a SD plug-in\(^6\).

An important issue with the application of the APRIORI-SD algorithm is that we were only capable of inducing rules for the KC2 dataset which corresponds with the more balanced one (20%). For the rest of the datasets the algorithm could not find any rules, no matter the variations in the parameters of the algorithm. The reason may reside in the fact that datasets are so highly imbalanced that rules do not achieve the minimum support and precision required, i.e., the number of samples covered by a rule is not enough to fulfil the quality criteria. Furthermore, the APRIORI-SD algorithm not only discretises continuous attributes but discrete ones are also binerised.

Tables 11–13 show the set of rules induced for the KC2 dataset using the default parameters used in the Orange tool (minimal support = 5%; minimal precision = 80%; beam width = 20; generalization parameter, $g = 5$ and No. of times a covered instance can be used before removed, $k = 5$). We do not show the rules for rest of the datasets for the sake of brevity and space. These three algorithms are deterministic when the parameters are fixed.

The first observation is the disparity in the number of rules induced. The CN2-SD algorithm generates very few rules for all four datasets (just two or three rules) but those rules are in general quite good. In order to validate EDER-SD against these classical SD algorithms from a machine learning perspective, we also used a splitting criterion. We divided the datasets into training and testing with two-thirds and one-third of the samples respectively. Table 14 shows the comparative results for the maximum values obtained for the evaluation measures described in Section 2.2 using all algorithms and testing datasets, where the $+$, $-$ and $=$ corresponds to EDER-SD performing better, worst or equal to other algorithm respectively. The reason for using maximum values is that all algorithms induce a very different number of rules (CN2-SD induces only two or three rules for these datasets). Therefore, it would not be fair to compare average values. Furthermore, the different algorithms optimise different functions. EDER-SD performed better than the other three algorithms counting the number of times that it obtained maximum values (the No. of pluses, minuses and equals in Table 14 shows if EDER-SD performed better, worst or equal than the other algorithms in their respective metrics). Table 14 shows that the EDER-SD performed better than the SD algorithm in 39 out of 70 evaluation measures, similar results in 14 and worst results in 17 evaluation measures. It also preformed better in five out of the seven datasets for most evaluation measures. The best results were obtained in the

---


PC1 dataset (the most imbalanced one) where EDER-SD outperformed the SD in all the evaluation measures. Similar results can be observed when comparing EDER-SD with the CN2-SD algorithm. It outperformed CN2-SD in 35 evaluation measures, obtained similar values in 10 occasions, and 25 values were worst. EDER-SD almost always performed better in the specificity, accuracy and lift measures and obtained similar or better results in WRAcc. The EDER-SD algorithm also performed better than the APRIORI-SD algorithm with the KC2 dataset and MC2 datasets (the only two that we were able to obtain results) for all evaluation measures with the exception of precision which was very similar. Taking into account all evaluation measures, EDER-SD performed better in 98 measures, similarly in 24 and worst in 43 out of the 160 evaluation measures.

Table 13
APRIORI-SD induced rules for the KC2 dataset.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th># Def</th>
<th># Non Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(v(G) &gt; 6 \land i_r(G) &gt; 8 \land uniqOp &gt; 11 \land totalOp &gt; 80)</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>(i_r(G) &gt; 8 \land uniqOp &gt; 11 \land totalOp &gt; 80 \land branchCount &gt; 11)</td>
<td>34</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>(LoC &gt; 100 \land uniqOp &gt; 11 \land uniqOpnd &gt; 34 \land totalOp &gt; 117)</td>
<td>31</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>(LoC &gt; 100 \land uniqOp &gt; 34 \land totalOp &gt; 80 \land totalOpnd &gt; 117)</td>
<td>31</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>(uniqOp &gt; 11 \land uniqOpnd &gt; 34 \land totalOp &gt; 80 \land totalOpnd &gt; 117)</td>
<td>31</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>(eV(g) &gt; 4 \land uniqOp &gt; 11 \land uniqOpnd &gt; 34)</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>(v(G) &gt; 6 \land eV(g) &gt; 4)</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>(eV(g) &gt; 4 \land branchCount &gt; 11)</td>
<td>37</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>(v(G) &gt; 6 \land i_r(G) &gt; 8 \land branchCount &gt; 11)</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>(LoC &gt; 100 \land uniqOp &gt; 11 \land uniqOpnd &gt; 34 \land totalOp &gt; 80)</td>
<td>34</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>(eV(g) &gt; 4 \land totalOp &gt; 80)</td>
<td>47</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 14
Comparison of EDER-SD with other SD algorithms.

<table>
<thead>
<tr>
<th>WRAcc</th>
<th>Cov</th>
<th>Sup</th>
<th>Acc</th>
<th>Sig</th>
<th>Prec</th>
<th>TP</th>
<th>Spec</th>
<th>Lift</th>
<th>fmsr</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDER-SD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM1</td>
<td>.025</td>
<td>.175</td>
<td>.042</td>
<td>.880</td>
<td>3.027</td>
<td>.357</td>
<td>.438</td>
<td>.947</td>
<td>3.705</td>
</tr>
<tr>
<td>KC1</td>
<td>.057</td>
<td>.287</td>
<td>.101</td>
<td>.859</td>
<td>2.767</td>
<td>.596</td>
<td>.657</td>
<td>.980</td>
<td>3.878</td>
</tr>
<tr>
<td>KC2</td>
<td>.090</td>
<td>.333</td>
<td>.155</td>
<td>.839</td>
<td>11.301</td>
<td>.769</td>
<td>.771</td>
<td>.978</td>
<td>3.824</td>
</tr>
<tr>
<td>KC3</td>
<td>.034</td>
<td>.268</td>
<td>.059</td>
<td>.935</td>
<td>6.219</td>
<td>.643</td>
<td>.643</td>
<td>1</td>
<td>10.928</td>
</tr>
<tr>
<td>MC2</td>
<td>.062</td>
<td>.389</td>
<td>.167</td>
<td>.759</td>
<td>3</td>
<td>.529</td>
<td>.529</td>
<td>1</td>
<td>3.176</td>
</tr>
<tr>
<td>MW1</td>
<td>.027</td>
<td>.119</td>
<td>.037</td>
<td>.941</td>
<td>3.07</td>
<td>.455</td>
<td>.455</td>
<td>1</td>
<td>12.273</td>
</tr>
<tr>
<td>PC1</td>
<td>.024</td>
<td>.259</td>
<td>.041</td>
<td>.943</td>
<td>7.125</td>
<td>.577</td>
<td>.577</td>
<td>1</td>
<td>14.231</td>
</tr>
<tr>
<td>SD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM1</td>
<td>.023</td>
<td>.313</td>
<td>.048</td>
<td>.855</td>
<td>2.067</td>
<td>.278</td>
<td>.5</td>
<td>.933</td>
<td>2.882</td>
</tr>
<tr>
<td>KC1</td>
<td>.050</td>
<td>.213</td>
<td>.083</td>
<td>.859</td>
<td>21.843</td>
<td>.600</td>
<td>.537</td>
<td>.975</td>
<td>3.906</td>
</tr>
<tr>
<td>KC2</td>
<td>.079</td>
<td>.322</td>
<td>.144</td>
<td>.839</td>
<td>9.854</td>
<td>.750</td>
<td>.714</td>
<td>.986</td>
<td>3.729</td>
</tr>
<tr>
<td>KC3</td>
<td>.042</td>
<td>.523</td>
<td>.078</td>
<td>.856</td>
<td>2.84</td>
<td>.857</td>
<td>.857</td>
<td>.928</td>
<td>2.277</td>
</tr>
<tr>
<td>MC2</td>
<td>.066</td>
<td>.815</td>
<td>.278</td>
<td>.741</td>
<td>2.171</td>
<td>.882</td>
<td>.882</td>
<td>.973</td>
<td>2.541</td>
</tr>
<tr>
<td>MW1</td>
<td>.030</td>
<td>.111</td>
<td>.037</td>
<td>.933</td>
<td>5.542</td>
<td>.455</td>
<td>.455</td>
<td>.992</td>
<td>9.205</td>
</tr>
<tr>
<td>PC1</td>
<td>.016</td>
<td>.097</td>
<td>.022</td>
<td>.897</td>
<td>4.341</td>
<td>.308</td>
<td>.308</td>
<td>.948</td>
<td>3.558</td>
</tr>
<tr>
<td>CN2-SD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM1</td>
<td>.029</td>
<td>.325</td>
<td>.06</td>
<td>.88</td>
<td>1.719</td>
<td>.238</td>
<td>.625</td>
<td>.973</td>
<td>2.47</td>
</tr>
<tr>
<td>KC1</td>
<td>.052</td>
<td>.418</td>
<td>.117</td>
<td>.859</td>
<td>21.843</td>
<td>.588</td>
<td>.759</td>
<td>.975</td>
<td>3.829</td>
</tr>
<tr>
<td>KC2</td>
<td>.071</td>
<td>.391</td>
<td>.149</td>
<td>.839</td>
<td>9.257</td>
<td>.667</td>
<td>.743</td>
<td>.950</td>
<td>3.314</td>
</tr>
<tr>
<td>KC3</td>
<td>.040</td>
<td>.346</td>
<td>.072</td>
<td>.856</td>
<td>2.84</td>
<td>.786</td>
<td>.786</td>
<td>.928</td>
<td>2.268</td>
</tr>
<tr>
<td>MC2</td>
<td>.042</td>
<td>.278</td>
<td>.13</td>
<td>.685</td>
<td>.769</td>
<td>.412</td>
<td>.412</td>
<td>.838</td>
<td>1.588</td>
</tr>
<tr>
<td>MW1</td>
<td>.032</td>
<td>.148</td>
<td>.044</td>
<td>.904</td>
<td>3.489</td>
<td>.545</td>
<td>.545</td>
<td>.952</td>
<td>4.909</td>
</tr>
<tr>
<td>PC1</td>
<td>.029</td>
<td>.322</td>
<td>.051</td>
<td>.927</td>
<td>4.763</td>
<td>.731</td>
<td>.731</td>
<td>.997</td>
<td>3.558</td>
</tr>
<tr>
<td>APRIORI-SD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KC2</td>
<td>.043</td>
<td>.091</td>
<td>.061</td>
<td>.831</td>
<td>7.3</td>
<td>.681</td>
<td>.306</td>
<td>.963</td>
<td>3.384</td>
</tr>
<tr>
<td>MW2</td>
<td>.027</td>
<td>.12</td>
<td>.065</td>
<td>.694</td>
<td>1.018</td>
<td>.559</td>
<td>.206</td>
<td>.919</td>
<td>1.775</td>
</tr>
</tbody>
</table>
In the case of software prediction, there is an ongoing discussion about the evaluation of defect prediction models [60.39]. Menzies et al. suggest the use of probability of detection \( (pd) \) and probability of false alarm \( (pf) \) arguing that a low precision acceptable in this domain and with the datasets used in this work. In any case, there is a trade-off between these two measures as stated by Menzies et al., increasing the recall, also increases the pf, and viceversa. Also, Khoshgoftaar and Seliya [28] chose to to minimise Type II errors \( \left( \text{a fault-prone error misclassified as non-fault-prone} \right) \) in accordance with a project manager of the system studied. When using EDER-SD in comparison with other tools, it is possible to generate rules according to the criteria of project managers or quality engineers.

We do not apply HIDER or other classification techniques due to the fact that quality measures are related to accuracy and this measure might not always be the most appropriate when data are imbalanced. Furthermore, in the case of hierarchical classification rules, those are harder to interpret and apply by domain experts than the rules obtained using EDER-SD. For example, Vandecruys et al. [54] show the rules obtained using their AntMiner + tool for the PC1 and KC1 datasets. AntMiner + extracts hierarchical classification rules, a chain of if \( \ldots \) then \( \ldots \) else \( \ldots \) which are harder to interpret and apply than the rules obtained using EDER-SD. For example, Table 15 shows the AntMiner + rules for the CM1 dataset. Although the first rule is easy to interpret, for the rest of the rules it becomes increasingly harder to extract useful knowledge. For modules not covered by the first rule we need to apply the second one and so on and so forth. Therefore, the number of conditions to consider is 3 for the first rule, 3 + 4 for the second rule, 3 + 4+5 for the third rule etc. Furthermore, the rules presented by Vandecruys to identify faulty modules have in general a low support and and the high level of accuracy for the Antminer + rules seems to be consequence of the final "else" default branch, which covers the non–defective modules.

4.4. Threats to validity

There are some threats to validity that need to be considered in this study as in all empirical studies.

**Construct validity** is the degree to which the variables used in the study accurately measure the concepts they to measure. Although there seems to be an agreement about the practical usefulness of static metrics, there are critics to their effectiveness as predictors of quality. Here, we can also highlight the point that is difficult to avoid an unfair comparison between SD algorithms as they induce a different number of rules, use different quality measures etc. For the comparison, the rules were obtained using the default parameters provided by the tool but other parameters could generate better sets of rules.

**Internal validity** is the degree to which conclusions can be drawn. This work consisted in a small number of datasets and all came from the same domain. There is some consistency among the attributes used within each dataset but they vary among datasets and so do the thresholds. In this work, we have generated a set of rules for each dataset but from a practical point of view, it could be interesting to generalise the rules across the datasets in order to facilitate their application by project managers or quality engineers. One approach to do this is to join all datasets and generate generic rules even if some degree of performance is lost in individual datasets. It can be observed that the threshold values across (the different datasets are close to each other (e.g., LoC varies between 60 and 80; \( \text{ist}(g) \) between 3 and 5, etc.).

**External validity** is the degree to which the results of the research can be generalised to the population under study and other research settings. According to Menzies et al. [40], the NASA repository can be generalised to the industry in general.

<table>
<thead>
<tr>
<th>Table 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>AntMiner + rules for the PC1 dataset [54].</td>
</tr>
<tr>
<td>if LOC_Blank &gt;= 16 and LOC_Code_And_Comment &gt;= 2 and Normalized_Cyclomatic_Complexity &gt;= 0.17 then class = Erroneous module else if LOC_Code_And_Comment &gt;= 1 and LOC_Comments &gt;= 5 and Normalized_Cyclomatic_Complexity &gt;= 0.23 and Num_Unique_Operands &gt;= 38 then class = Erroneous module else if Halstead_Content &gt;= 50.37 and Halstead_Error_Est &gt;= 0.6 and LOC_Blank &gt;= 16 and LOC_Comments &gt;= 14 and LOC_Executeable &gt;= 53 then class = Erroneous module else if Halstead_Content &gt;= 50.37 and LOC_Blank &gt;= 16 and LOC_Code_And_Comment &gt;= 2 and LOC_Comments &gt;= 14 and Normalized_Cyclomatic_Complexity &gt;= 0.08 then class = Erroneous module else if Halstead_Content &gt;= 50.37 and LOC_Code_And_Comment &gt;= 2 and LOC_Comments &gt;= 5 and Normalized_Cyclomatic_Complexity &gt;= 0.17 then class = Erroneous module else class = Correct module</td>
</tr>
</tbody>
</table>
However, it is probably better to calibrate the rules to different domains or organisations. Finally, as with other empirical studies, this approach needs to be replicated with further datasets and SD algorithms.

5. Related work

Initially, some statistical approaches were proposed to deal with defect prediction. For example, Munson and Khoshgoftaar [42] explore discriminant analysis techniques on two commercial datasets composed with many of the Halstead and McCabe metrics used in this work. Basili et al. [4] analysed the applicability of Chidamber and Kemerer’s Object Oriented set of metrics [10] with logistic regression to predict fault-prone code classes. Khoshgoftaar and Allen [27] also analysed logistic regression extended with prior probabilities and of misclassification costs.

More recently, a number of researchers have focused on machine learning approaches. Khoshgoftaar et al. [26] described the use of neural networks for quality prediction. The authors used a dataset from a telecommunications system and compare the neural networks results with a non-parametric model. Also, Khoshgoftaar et al. [25] applied regression trees as classification model to the same problem.

However, there are still large discrepancies regarding the assessment of the goodness of the different techniques and the reasons for such discrepancies [44,60,39]. For example, Lessmann et al. [33] compare 22 classifiers grouped into statistical, nearest neighbour methods, neural networks, support vector machine, decision trees and ensemble methods over ten datasets from the NASA repository. The authors discuss several performance metrics such as TP, and FP, but advocate the use of AUC as the best indicator to compare the different classifiers. Arisholm et al. [3] compare different data mining techniques (classification tree algorithm (C4.5), a coverage rule algorithm (PART), logistic regression, back–propagation neural work and support vector machines) over 13 releases of a Telecom middleware software developed in Java using three types metrics: (i) object oriented metrics, (ii) delta measures, amount of change between successive releases, and (iii) process measures from a configuration management system. The authors concluded that although there are no significant differences regarding the techniques used, large differences can be observed depending on the criteria used to compare them. The authors also propose a cost–effectiveness measure based on the AUC and number of statements so that larger modules are more expensive to test. The same approach of module size in conjunction with the AUC as evaluation measure has been explored by Mende and Koschke [38] using NASA datasets and three versions of Eclipse’7 and random forests [6] as classification technique. Kuru and Liu [31] use the C4.5 [49] implementation of Weka for defect prediction on the NASA datasets to analyse the relationships between defects and module size. Khoshgoftaar and Seliya [28] recognise the problem of imbalanced data and use Case-based Reasoning to deal with this problem, considering Type I error when a non-faulty module is classified as faulty and Type II error occurs when a faulty module is classified as non-faulty. This approach is also considered by Ostrand and Weyuker [45].

Also in this respect but in the domain of cost estimation, Shepperd and Kadoda [52] analyse the influence of different data characteristics (dataset size; number, type and independence of features; and type of distribution, in.) using simulated data over a number of different types of classifiers (regression, rules induction, nearest neighbour and neural networks). The authors conclude there is no best classifier as the characteristics of the data highly affect the outcomes.

There are a number of other works using subsets of the NASA repository. Peng et al. [47] propose a performance metric to evaluate the merit of classification algorithms using a broad selection of classification algorithms and performance measures. The experimental results, using 13 classification algorithms with 11 measures over 11 software defect datasets, indicate that the classifier which obtains the best result for a given dataset according to a given measure may perform poorly on a different measure. The results of the experiment indicate that support vector machines, k-nearest neighbor algorithm and C4.5 algorithm ranked the top three classifiers. Menzies et al. [40] applied J48 (the Weka implementation of the C4.5) and Naïve Bayes to several datasets of the PROMISE repository for defect prediction. The authors concluded that such technique can be used as good defect estimators and suggest bound exploration as part of future work. Elish and Elish [12] applied the Support Vector Machine (SVM) technique to a subset of the NASA repository. The authors concluded that SVM is capable of improving or at least obtaining similar result than other techniques such as logistic regression, neural networks, Bayesian networks or decision trees. Recently, Peng et al. [48] have also analysed ten NASA datasets using four Multicriteria Decision Making methods to rank classification algorithms, highlighting that the boosting of CART and the boosting of C4.5 decision tree are ranked as the most appropriate algorithms to deal with defect prediction.

Finally, some authors criticize the use of only static metrics with statistical techniques as an approach to defect prediction. For example, Fenton and Neil [14] advocate the use of Bayesian network approaches as a probabilistic technique to estimate defects among other parameters.

6. Conclusions and future work

In this work, we applied Subgroup Discovery (SD), a data mining approach used to find groups of statistically different data given a property of interest, to the problem of software defect prediction. SD is a sensible approach to deal with imbalanced, inconsistent or redundant data.

---

7 http://www.eclipse.org/.
To do so, we developed a genetic algorithm, EDER-SD (Evolutionary Decision Rules for Subgroup Discovery), used to induce rules describing only fault-prone modules. EDER-SD has the advantage of working with continuous variables as the conditions of the rules are defined using intervals. Furthermore, the fitness function of the genetic algorithm can be adapted to optimise the most suitable quality measure of the domain. EDER-SD was applied to seven publicly available datasets from the PROMISE repository. The results show that the induced rules are capable of characterising subgroups of fault-prone modules. We also found that many of the thresholds found by the metrics are close to those defined in the literature or the McCabe IQ tool used to obtain the datasets. Furthermore, the simplicity of the induced rules and their readability facilitates the application of this approach helping project managers or quality engineers with testing and software quality assurance activities. We also compared EDER-SD with three other well-known SD algorithms. The results of the comparison showed the advantages of been able to adapt the fitness function of the evolutionary algorithm as (in general) EDER-SD did perform better than the other algorithms.

In relation to future work, we will further explore this approach with other datasets and different metrics, for example, datasets with object oriented metrics. There is also room for improvement of the genetic algorithm, for example exploring further quality measures or multiobjective approaches.

Acknowledgements

The authors are grateful to the anonymous reviewers and Prof Rachel Harrison for their useful comments. This work has been supported by the projects TIN2007-68084-C02-00 and TIN2010-21715-C02-01 (Spanish Ministry of Education and Science). D. Rodríguez carried out part of this work as a visiting research fellow at Oxford Brookes University, UK.

References