Information and Software Technology 51 (2009) 1365–1376

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A hybrid heuristic approach to optimize rule-based software quality estimation models

D. Azar, H. Harmanani *, R. Korkmaz

Department of Computer Science and Mathematics, Lebanese American University, Byblos 1401 2010, Lebanon

ARTICLE INFO

Article history:
Received 1 March 2009
Received in revised form 18 May 2009
Accepted 18 May 2009
Available online 18 June 2009

Keywords:
Software quality
Search-based software engineering
Soft computing

ABSTRACT

Software quality is defined as the degree to which a software component or system meets specified requirements and specifications. Assessing software quality in the early stages of design and development is crucial as it helps reduce effort, time and money. However, the task is difficult since most software quality characteristics (such as maintainability, reliability and reusability) cannot be directly and objectively measured before the software product is deployed and used for a certain period of time. Nonetheless, these software quality characteristics can be predicted from other measurable software quality attributes such as complexity and inheritance. Many metrics have been proposed for this purpose. In this context, we speak of estimating software quality characteristics from measurable attributes. For this purpose, software quality estimation models have been widely used. These take different forms: statistical models, rule-based models and decision trees. However, data used to build such models is scarce in the domain of software quality. As a result, the accuracy of the built estimation models deteriorates when they are used to predict the quality of new software components. In this paper, we propose a search-based software engineering approach to improve the prediction accuracy of software quality estimation models by adapting them to new unseen software products. The method has been implemented and favorable result comparisons are reported in this work.

1. Introduction

With the complexity of object-oriented (OO) software products on the rise, it is becoming crucial to evaluate the quality of the software products during the different stages of development in order to reduce time, effort and money. The quality of a software is evaluated in terms of characteristics such as maintainability, reusability, stability, etc. Though most of these characteristics cannot be measured before the software is used for a certain period of time, they can be deduced from several measurable software attributes such as cohesion, coupling and size. For this purpose, several metrics that capture such attributes have been proposed in the literature [7,17,15,10,27,26,33,32], and [35]. Examples of such metrics include number of children of a class in an object-oriented system (NOC), and number of methods (NOM). Software quality estimation models build a relationship between the desired software quality and the measurable attributes. These models are either statistical models (for example, regression models [21,30]) or logical models [29,19]. The latter have been extensively used because of their white-box nature as they provide practitioners with the prediction label as well guidelines to attain it. Logical models can take the form of decision trees or rule sets. In general, they suffer from degradation of their prediction when they are applied to new/unseen data. This is largely due to the lack of a representative sample that can be drawn from available data in the domain of software quality. Unlike other fields where public repositories abound with data, software quality data is usually scarce. The reason is that not many companies systematically collect information related to software quality. Furthermore, such information is normally considered confidential and in the case where a company is willing to make it public, usually only the resulting model is published. The latter is company-specific and is difficult to generalize, to cross-validate or to re-use it.

Search-based software engineering, first coined by Harman et al. [25], is an emerging field that applies metaheuristic search techniques to software engineering problems. The field has recently witnessed intense activity [16,37], and shown a lot of promise when applied to various software engineering problems including project management [2,4,14], prediction in software engineering management [20,31], project planning and quality assessment [1,13,31], and software testing [24,34,44]. Typically, search methods such as local search, simulated annealing, genetic algorithms, and genetic programming are used as sampling techniques [23]. However, to the best of our knowledge, not much work was reported using *hybrid metaheuristic approaches* nor using *tabu search* in search-based software engineering.

* Corresponding author.
E-mail address: haidar@acm.org (H. Harmanani).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.05.003

This paper presents a search-based software engineering approach that consists of building a better model from already-existing ones. This can be seen as combining the expertise of several expert models, built from common domain knowledge, and adapting the resulting models to context-specific data. To validate our technique, we use the specific problem of predicting the stability of object-oriented (OO) software components (classes, in this context). During its evolution, a software component undergoes various modifications due to changes in requirements, error detections, changes in the environment, etc. It is important for the software component to remain "usable" in the new changed environment. In [5], we presented a similar approach using genetic algorithms (GA). The results were promising. In this work, we present a hybrid approach to solve the above model. The approach is hybrid on two levels. It combines the strengths of different heuristics (genetic algorithms, tabu search and simulated annealing), and it works on two levels of optimization (rule set level and rule level). Results show that our hybrid heuristic outbeats C4.5 as well as the genetic algorithm presented in [5]. The remainder of this paper is organized as follows. In Section 2, we give an overview of the related work in the field. In Section 3, we state the problem and the objective of the work. In Section 4, we give an overview of the heuristics used in our approach. We also describe how we instantiate elements of these heuristics to our problem. In Section 6, we explain the experiments that we perform and the obtained results. Finally, in Section 7, we conclude with a brief recollection of the technique and future paths.

2. Related work

Logical estimation models have been widely used in the domain of software quality. Selby and Porter [42] have used machine learning to build such models in the form of decision trees as early as 1988. Later on, such models were very popular in the field. Mao et al. [36] used C4.5 to build models that predict reusability of a class in an object-oriented software system from metrics for inheritance, coupling and complexity. The authors proposed the use of estimation models as guidelines for future software development. Basili et al. [8] use C4.5 to build models that estimate the cost of rework in a library of reusable components. De Almeida et al. [3] use C4.5 rule sets to predict the average isolation effort and the average effort. Briand et al. [10] investigated the relationship between most of the existing coupling and cohesion measures defined at the level of a class in an object-oriented system on one hand, and fault-proneness on the other hand. In all the cases, the models were not very useful in predicting the quality characteristic of unseen software. The main reason is that the data used to build the models is scarce so the models become hard to generalize. Various search-based software engineering approaches have been also proposed in the domain of software quality. For example, Azar and Precup [5] and Bouktif et al. [9] presented two genetic algorithm-based approaches that optimize the accuracy of these models on new data. One approach relies on the recombination of several models into new ones. The other one relies on the adaptation of a single model to a new data set. Both approaches outbeat C4.5 when tested on the stability of classes in an object-oriented software system with the recombining approach showing the highest performance [6]. Pedrycz and Succic [39] represent classifiers as hyperboxes and uses genetic algorithms to modify these hyperboxes (and the underlying classifiers). Similarly to our approach, this technique preserves the interpretability of the classifiers and can easily be extended to a problem with multiple classification labels. However, the data that is used to validate this approach uses different metrics than those reported in this work, and is concerned with software maintenance rather than stability. Vivanco [43] uses a genetic algorithm to improve a classifier accuracy in identifying problematic components. The approach relies on the selection of metrics that are more likely to improve the performance of predictive models.

3. Problem statement and objective

The problem notation originates from the machine learning formalism. A *data set* is a set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of *n instances* or *cases* where $x_i = \langle a_1, a_2, \ldots, a_d \rangle \in \Re^d$ is an *attribute vector* of *d attributes*, and $y_i \in C$ is a *classification label*. In the particular problem that we are considering, a case represents a software component (a class in an OO software system). The attributes $(a_1, \ldots, a_d)$ are metrics (such as number of methods, number of children, etc.) that are considered to be relevant to the software quality factor being predicted (stability). The label y_i represents the software quality factor. In this problem, $y_i \in \{0(\text{stable}), 1(\text{unstable})\}$.

A *classifier* is a function $f : \Re^d \mapsto C$ that predicts the label y_i of any attribute vector x_i. In the framework of supervised learning, it is assumed that (vector, label) pairs are random variables (X, Y) drawn from a fixed but unknown probability distribution, and the objective is to find a classifier f with a low error (misclassification) rate. Since the data distribution is unknown, both the selection and the evaluation of f must be based on the data set D. For this purpose, D is partitioned into two parts, the *training set* D_{train} and the *testing set* D_{test}. Most learning algorithms take the training set as input and search the space of classifiers for one that minimizes the error on D_{train}. The output classifier is then evaluated on the testing sample D_{test}. Examples of learning algorithms that use this principle are the back-propagation algorithm for feed forward neural nets [41] and C4.5 [40]. In our experiments, we use *10-fold cross validation* that allows us to use the whole data set for training and to evaluate the error probability more accurately.

In this paper, we focus on rule-based classifiers i.e. classifiers that take the form of rule sets. A rule-based classifier is a disjunction of conjunctive rules and a default classification label. Fig. 1 illustrates a rule-based classifier that predicts the stability of a component (a class in an object-oriented software system) based on the metrics number of classes used by a member function (CUBF), number of parents (NOP) and number of used classes (CUB). The example model that has three rules and a default classification label. The first rule estimates that if the number of classes used by a member function (CUBF) in the designated class is greater than 7 then the class is unstable (1). The second rule classifies a class with number of parents (NOP) greater than 2 as unstable. The third rule classifies a class with the number of used classes (CUB) less than or equal to 2 and number of parents (NOP) less than or equal to 2 as stable (0). The classification is sequential. The first rule (toward the top) whose left hand side is satisfied by a case fires. If no such rule exists, the default classification label is used to classify the case (default class 1).

The model has an accuracy that can be measured using the correctness of the classifier (percentage of cases correctly classified) or its Youden's J_{index}, $J(R)$, (average correctness per class label) as follows Eq. (1):

$$J(R) = \frac{1}{k} \sum_{i=1}^{k} \frac{n_{ii}}{\sum_{j=1}^{k} n_{ij}}. \tag{1}$$

Rule 1: CUBF $> 7 \rightarrow 1$
Rule 2: NOP $> 2 \rightarrow 1$
Rule 3: CUB $\leq 2 \wedge$ NOP $\leq 2 \rightarrow 0$
Default class: 1

Fig. 1. Example rule-based classifier.