Information Sciences 259 (2014) 555–570

Contents lists available at ScienceDirect

Information Sciences

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

Software quality assessment using a multi-strategy classifier

Taghi M. Khoshgoftaar [a,*], Yudong Xiao [a], Kehan Gao [b]

[a] Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, United States
[b] Department of Mathematics and Computer Science, Eastern Connecticut State University, Willimantic, CT 06226, United States

ARTICLE INFO

Article history:
Available online 3 December 2010

Keywords:
Rule-based model
Case-based learning
Genetic algorithm
Multi-strategy classifier
Software quality classification

ABSTRACT

Classifying program modules as fault-prone or not fault-prone is a valuable technique for guiding the software development process, so that resources can be allocated to components most likely to have faults. The rule-based classification and the case-based learning techniques are commonly used in software quality classification problems. However, studies show that these two techniques share some complementary strengths and weaknesses. Therefore, in this paper we propose a new multi-strategy classification model, RB2CBL, which integrates a rule-based (RB) model with two case-based learning (CBL) models. RB2CBL possesses the merits of both the RB model and CBL model and restrains their drawbacks. In the RB2CBL model, the parameter optimization of the CBL models is critical and an embedded genetic algorithm optimizer is used. Two case studies were carried out to validate the proposed method. The results show that, by suitably choosing the accuracy of the RB model, the RB2CBL model outperforms the RB model alone without overfitting.

1. Introduction

Software reliability is an important attribute of high-assurance and mission-critical systems. Identifying which software modules, during the software development process, are likely to be faulty is a challenging but valuable technique for improving software quality. Software quality assurance efforts can be focused on those program modules that are either high-risk or likely to have a high number of faults. A variety of classification and prediction approaches have been proposed in recent years [3,4,15–18,23,27,35]. Among them, rule-based classification and case-based reasoning techniques are commonly used in software quality classification problems. However, studies [2,8,14] show that these two techniques share some complementary strengths and weaknesses. In the field of data mining and machine learning, there exists a methodology that combines two or more basic learning approaches into an algorithm that may behave more appropriately than any basic method alone. This methodology is termed as an "empirical multi-strategy learning technique" [28,29]. KBNGE [37] and RISE [8] systems are examples of this type.

In Wettschereck [37], a hybrid method that combines the nearest-hyperrectangle algorithm and the *k*-Nearest Neighbor algorithm, called KBNGE, was introduced for improved classification accuracy. Results from eleven domains showed that KBNGE achieved generalization accuracies similar to the *k*-Nearest Neighbor algorithm at improved classification speed. KBNGE is a fast and easy to use inductive learning algorithm that gives very accurate predictions in a variety of domains and represents the learned knowledge in a manner that can be easily interpreted by the user. In Domingos [8], a unification

* Corresponding author. Tel.: +1 561 297 3994; fax: +1 561 297 2800.
E-mail addresses: taghi@cse.fau.edu (T.M. Khoshgoftaar), yudongxiao@gmail.com (Y. Xiao), gaok@easternct.edu (K. Gao).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2010.11.028

of two widely-used empirical approaches, rule induction and instance-based learning, was described. Theoretical analysis showed this approach to be efficient. It was implemented in the RISE 3.1 system. In an extensive empirical study, RISE consistently achieves higher accuracies than state-of-the-art representatives of both its parent approaches (PEBLS and CN2), as well as a decision tree learner (*C4.5*). Lesion studies show that each of RISE's components is essential to this performance. Most significantly, in 14 of the 30 domains studied, RISE is more accurate than the best of PEBLS and CN2, showing that a significant synergy can be obtained by combining multiple empirical methods.

A multi-strategy system can consist of a global procedure which calls different algorithms as sub-procedures, or a simple algorithm that can behave and adapt properly to different situations. Many combination or unification schemes are possible. In this paper, we present a novel classification model, RB2CBL, which cascades a rule-based (RB) model with two case-based learning (CBL) models. RB2CBL integrates the advantages of both RB and CBL models and restrains their drawbacks. Next, we will simply introduce the background and characteristics of the RB model (*C4.5*) and the CBL model as well as the possibility of the collaboration between the two types of the models.

Rule-based classification methods extract classification rules from a training dataset and use them to predict the category of the dependent variable for the incoming unlabelled instances. Once the rules are obtained, RB models work fast. Among RB models, decision trees [6,13,16,35] are especially attractive in the data mining and software quality modeling fields for several reasons. First, due to their intuitive representation, the result is easy to understand by humans [6,26]. Second, decision trees do not require many parameter settings from the user and thus are especially suited for exploratory knowledge discovery. Third, decision trees can be constructed relatively quickly [10,33]. In addition, the accuracy of decision trees is comparable or superior to other classification models [24]. In fact, the *C4.5* [30] decision tree model has become a very popular machine learning method because it produces an accurate and fast classifier.

In a case-based reasoning (CBR) system [17,22], by calculating the *distance* or *similarity* between the test data and the fit (training) data points, an unlabelled data point in the test dataset is predicted to be the same class as the most similar training data points are. The CBR model uses a lazy evaluation method, i.e., it classifies a data point only when immediately needed. Accordingly the CBR model has no overhead and new training data can be added in at any time. It is very flexible and, theoretically speaking, suited for dynamically changing data.

CBR systems have been used to solve real-world problems in many fields [3,5,12,17,19,25,36]. However, CBR algorithms have several deficiencies [2]:

- They are computationally expensive because they save and compute similarities to all the training cases.
- They are intolerant of noise.
- They are intolerant of irrelevant features.
- They are sensitive to the choice of the similarity function.

For CBR systems, as learning progresses, the learner's knowledge about certain parts of the input space increases, and examples in the "well-understood" portion of the space become less useful.

In the *Probably Approximately Correct* (PAC) model [9,32,34], learning algorithms need to approximately double the number of seen examples in order to halve their error rate. However, for conservative algorithms, since the number of examples actually used for learning is proportional to the error rate, the number of new examples used by the algorithm each time it wishes to halve its error rate remains (approximately) constant. Thus, the number of examples actually used to achieve some error rate is logarithmic rather than linear.

Along this line, Aha et al. described a framework and methodology of case-based learning (CBL) [1,2] that generates class predictions using only specific instances. This approach extends the nearest neighbor algorithm, which has large storage requirements. It described how storage requirements can be significantly reduced with, at most, minor sacrifices in learning rate and classification accuracy. While the storage-reducing algorithms perform well on some real-world databases, its performance degrades rapidly with increase of the level of attribute noise in training instances. Therefore, Aha et al. extended it with a significance test to distinguish noisy instances. The extended algorithm's performance degrades gracefully with increasing noise levels and compares favorably with a noise-tolerant decision tree algorithm.

Two induction paradigms with largely complementary strengths and weaknesses are rule-based induction and the case-based learning algorithm. Rule-based induction systems often succeed in identifying small sets of highly predictive features and making effective use of statistical measures to combat noise [8]. However, they can only form axis-parallel frontiers in the instance space, and they have trouble recognizing exceptions, or in general small, low-frequency sections of the space [14]; this is known as the small disjuncts problem. Furthermore, their general-to-specific (separate and conquer) search strategy causes them to suffer from the splintering problem: as induction progresses, the amount of data left for further learning dwindles rapidly, leading to wrong decisions or insufficient specialization due to lack of adequate statistical support. On the other hand, the CBL method can form complex, non-axis-parallel frontiers, and be well-suited to handling clustered distributed datasets and exceptions, but can be very vulnerable to noise and irrelevant features. Therefore, we create a new multi-strategy classification model which attempts to overcome the limitations of the RB and CBL models while maintaining the benefits of both models.

The rest of the paper continues with Section 2 in which the proposed multi-strategy classifier is described. Sections 3 and 4 present two case studies and finally we draw our conclusion in Section 5, including suggestions for future work.

2. RB2CBL: multi-strategy classifier

2.1. Structure and methodology of RB2CBL

The multi-strategy classifier, RB2CBL, consists of two parts: a rule-based model, RB, and two case-based learning models, CBL_p and CBL_n, as shown in Fig. 1, where the rule-based classifier can be any kind of rule-based algorithm, such as a decision tree (*C4.5*).

All instances in the dataset are classified into two predefined classes by the RB model: positive class *P*, and negative class *N*. For a given dataset, comparing the actual class with the predicted class, each instance of the dataset will fall into one of the four categories as shown in Table 1. They are true positive (*TP*), false positive (*FP*), true negative (*TN*), and false negative (*FN*), among which *TP* and *TN* are correct classifications while *FP* and *FN* are wrong classifications.

Initially, the rule-based model, RB, is built from a given fit dataset D_{fit}. RB classifies D_{fit} into two classes: positive class D_p and negative class D_n. Then the CBL models, CBL_p and CBL_n, are built using D_p and D_n, respectively.

When the fit dataset passes through the RB model, the instances are most likely to be correctly classified into the classes D_p or D_n. If an instance s is misclassified, it will be put into the CBL case libraries, L_p or L_n, as a new case. So, the CBL case libraries store the misclassified instances. After that, if the same instance, s, passes through the RB2CBL again, the CBL algorithm will identify this misclassification and re-classify the instance. Thus, when the RB2CBL model has been built, it always accurately (with 100% accuracy) classifies the fit dataset if the fit dataset does not contain contradictory instances, where the contradictory instances are the instances which have identical attributes but are labelled as different classes. The algorithm is described in Figs. 2 and 3.

Once the classifier RB2CBL is built, for each instance, s, of the normal (test) dataset, RB2CBL will predict its class according to the following process. First, an instance s is classified into one of two classes, D_p and D_n, by the model, RB, then s is re-checked by CBL_p (if $s \in D_p$) or CBL_n (if $s \in D_n$) for some necessary modifications, and finally s is classified into the class, *prediction*(s). The algorithm is described in Fig. 4. For example, suppose an instance, s, of the test dataset passes through RB2CBL and is put into, say, class D_p by RB. Next, s will be compared with the instances in the case library L_p to see whether there is a similar instance in the case library. If the similar instance exists, then s would probably be re-classified into the class D_n.

In Fig. 4, there are two parameters α_p and α_n that are used to control the similarity of a given point and the point in the case library of CBL. The values of the parameters α_p and α_n are very crucial. When $\alpha_p, \alpha_n = 0$, the behavior of CBL_p and CBL_n is equivalent to the pure rule-based model. When $\alpha_p, \alpha_n > 0$, each case in the case libraries L_p and L_n will cover a circular region. Any potential point s which falls into this region will be considered similar to that case and should have the same class as it does. As a result, s is re-classified by the CBL models. When α_p and α_n get larger, the region will cover a larger neighborhood, which may make too many potential points be re-classified and increase the misclassification rate. This phenomenon is

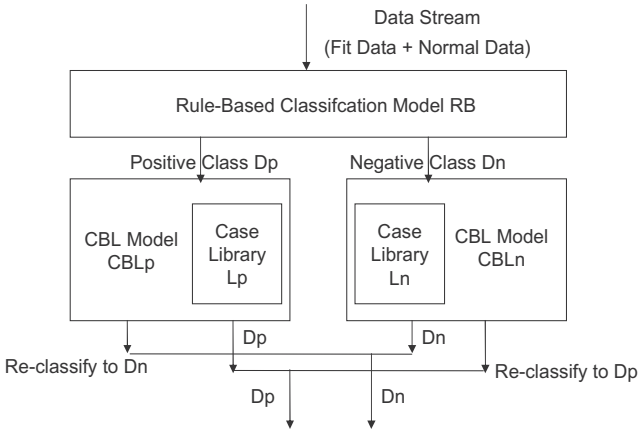

Fig. 1. The architecture of RB2CBL.

Table 1
Two group classification.

Symbol	Description
TN	Negative instance labelled as negative
FN	Positive instance labelled as negative
TP	Positive instance labelled as positive
FP	Negative instance labelled as positive

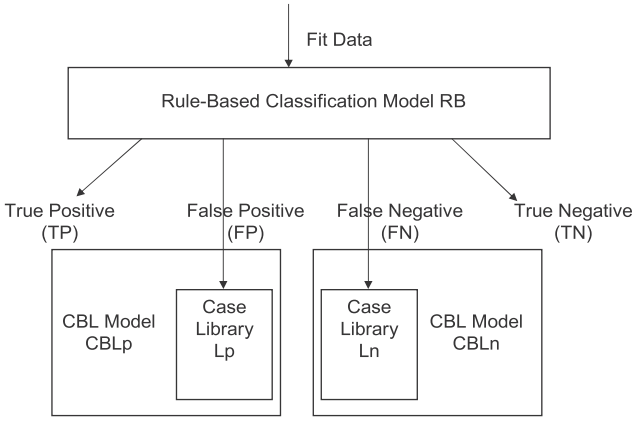

Fig. 2. The process of building CBL_p and CBL_n.

1. **Procedure Build_classifer_RB2CBL**(D_{fit})
2. Build_rule_based_model_RB(D_{fit}); //Build model RB from fit dataset D_{fit}
3. Classify D_{fit} to two class D_p and D_n by RB;
4. Build_CBLp(D_p); //Build CBL_p model from dataset D_p
5. Build_CBLn(D_n); //Build CBL_n model from dataset D_n
6. **Procedure Build_CBLp**(D_p)
7. for each point $s \in D_p$
8. modify_case_lib_Lp(s);
9. **Procedure Build_CBLn**(D_n)
10. for each point $s \in D_n$
11. modify_case_lib_Ln(s);
12. **Procedure modify_case_lib_Lp**(s)
13. if $s \in FP$, then put s into case library L_p;
14. **Procedure modify_case_lib_Ln**(s)
15. if $s \in FN$, then put s into case library L_n;

Fig. 3. Building a RB2CBL model.

called *single side overfitting* because it causes only the points that are classified into class D_p to change to class D_n in model CBL_p and class D_n to change to class D_p in model CBL_n. As the values of α_p and α_n increase, the single side overfitting that occurs becomes more severe, but it can be adjusted independently by changing the parameters α_p and α_n in the CBL models separately. In other words, if the values of the parameters α_p and α_n are properly set, then CBL_p and CBL_n will correct some misclassified instances and finally reduce the total misclassification rate.

2.2. Algorithm of CBL

In this study, an algorithm which is modified from Aha's *IB*3 [2] is used. The case library L_p keeps only misclassified cases *FP* in D_p, and L_n keeps misclassified cases *FN* in D_n for all cases of the fit dataset. The basic 1-nearest-neighbor algorithm is used.

During the model training process, the work history of each misclassified instance in the case libraries, L_p and L_n, is recorded. The record adds one for each correct re-classification and subtracts one for each incorrect re-classification. The record is kept for each case in the case libraries L_p and L_n. The parameters thd_p and thd_n are the thresholds of case libraries L_p and L_n, respectively. For the prediction on the test dataset, an instance will be re-classified if it falls into a neighbor of a case and the record of that case is greater than the threshold thd_p (for class P) or thd_n (for class N). Fig. 4 also presents this CBL algorithm.

2.3. Optimization by genetic algorithm

The genetic algorithm (GA) [11,20,21] is known as a common and effective tool to optimize parameters for highly discontinuous functions. A GA starts from a set of initial solutions, and uses biologically inspired evolution mechanisms to derive new and possibly better solutions. It starts from initial population P_0, and generates a sequence of populations $P_1, \ldots, P_n$, by

1. **Procedure Work_process_of_RB2CBL**(s)
2. Classify_by_RB(s); //s falls into one of two classes D_p and D_n;
3. if $s \in D_p$ then Classify_by_CBLp(s); //re-classify s by CBL_p
4. if $s \in D_n$ then Classify_by_CBLn(s); //re-classify s by CBL_n
5. **Procedure Classify_by_CBLp**(s)
6. if $\exists\ t \in L_p$: $distance(s,t) \leq \alpha_p$ and Record(t)> thd_p, then $prediction(s) \in D_n$;
7. else $prediction(s) \in D_p$;
8. **Procedure Classify_by_CBLn(** s **)**
9. if $\exists\ t \in L_n$: $distance(s,t) \leq \alpha_n$ and Record(t)> thd_n, then $prediction(s) \in D_p$;
10. else $prediction(s) \in D_n$;
11. **Procedure Record(** t **)** // This is for training process, where $t \in L_p$ or $t \in L_n$
12. for $\forall\ s \in D_p$, if $prediction(s) \in D_n$ and $actual\_class(s) \in D_n$, // t is used in $prediction(s)$
13. then record++;
14. else if $prediction(s) \in D_n$ and $actual\_class(s) \in D_p$
15. then record−−;
16. for $\forall\ s \in D_n$, if $prediction(s) \in D_p$ and $actual\_class(s) \in D_p$, // t is used in $prediction(s)$
17. then record++;
18. else if $prediction(s) \in D_p$ and $actual\_class(s) \in D_n$
19. then record−−;
20. return record;

Fig. 4. Multi-strategy classification of RB2CBL.

using three types of operations within the population: *crossover*, *mutation*, and *reproduction*. The elements of the population are called *chromosomes* and the fitness of each chromosome is measured by a *fitness function*. Each chromosome consists of a set of *genes*. For each generation, the algorithm selects some of the chromosomes and uses crossover or mutation on them with some given probabilities. *Crossover* mixes genes and *mutation* randomly changes some genes. Each pair of chromosomes creates a new pair. Each generation inherits some chromosomes from the last generation and accepts some newly created chromosomes by a given probability. This mechanism of the transportation of evolving genetic material to a future generation is called *reproduction*. The fitter chromosomes have a higher chance of being inherited into the next generation. The algorithm stops when a certain criterion is satisfied or a fixed number of generations is reached.

For the case studies of this paper, we used `Galib v2.4`, a C++ library of genetic algorithm components, as the GA engine in the model development process. `Galib` contains a set of C++ genetic algorithm objects. The library includes tools for using genetic algorithms to do optimization in any C++ program using a variety of representations and genetic operators. The source code and related documents of `Galib` can be obtained for free at http://lancet.mit.edu/ga/.

The RB2CBL model uses the genetic algorithm to optimize parameters α_p, α_n, thd_p, and thd_n during the modeling process. Therefore, an extra dataset, called *opt*, is needed for *fitness* evaluation. The *opt* dataset can be provided stand-alone if there are enough labelled data instances available, otherwise the *opt* data is identical to the *fit* data and *n*-fold cross-validation has to be used.

In order to optimize the parameters α_p, α_n, thd_p, and thd_n, the genetic algorithm needs to minimize the objective function (1) defined later. Therefore, the genome (chromosome) is represented as an array of 4-real values with each element defined on an allele set that corresponds to its location in the genome. Four arrays each comprised of 3-real values: $\langle maximum, minimum, interval\rangle$ are assigned to the parameters α_p, α_n, thd_p, and thd_n, respectively. Each defines a data range and changing interval allowed for a specific parameter. At each run of the evolution, the GA engine only picks up the next value from the corresponding allele set for each parameter α_p, α_n, thd_p, or thd_n. But, as the combination of the four values, the next value of genome, $\langle \alpha_p, \alpha_n, thd_p, thd_n\rangle$, will be globally determined by the GA algorithm and the assessment result of the objective function.

One aspect that GA users may notice is the fact that a GA, working on the population of bit strings, has no knowledge of the semantics associated with these bits. It only contacts the environment through the fitness function. The fitness function is very crucial to the GA. It enforces selection pressure in the GA and guides the evolution. In practice, the objective function is more likely to be stated as minimizing some cost function rather than maximizing some profit function. In the GA-engine, mapping an objective function to a fitness function is carried out automatically. The only thing needed to be taken care of is the objective function.

The modeling method may not always create accurate predictions. In the context of two-group software quality classification modeling, a software program module would be classified as *fault-prone* or *not fault-prone*. Therefore, two types of misclassifications can occur, Type I and Type II. A Type I misclassification (*FP*) is when a model classifies a software module as fault-prone which is actually not fault-prone, while a Type II misclassification (*FN*) is when a model classifies a software

module as not fault-prone and which is actually fault-prone. The objective in our case study is to balance the Type I and Type II misclassification rates with the Type II error being as low as possible, as well as to minimize total misclassification rate. To achieve this goal, the objective function F_{obj} was designed as

$$F_{obj} = \left(\frac{t_1}{t_2 e + 0.001} + \frac{t_2 e}{t_1 + 0.001}\right) + d * (t_1 + t_2 e), \quad (1)$$

where t_1 and t_2 are the Type I and Type II error rates[1], respectively. The terms in the first parenthesis of Eq. (1) are used to balance the two types of errors. The 'e' is the weight of the Type II error, which is normalized with respect to the weight of the Type I error. If $e = 1$, then the Type I and Type II errors are of the same weights. Since the Type II error is more severe, the weight of a Type II error should be greater than the weight of a Type I error, which is 1. But on the other hand, to obtain equal balance between the two types of errors and also to make the Type II error rate less than the Type I error rate, the weight 'e' should be a number slightly greater than 1. In the case study, e was set to 1.1. We also tried using other values between 1 and 1.5 for 'e', but 1.1 was the most appropriate one. A small constant, 0.001, is added to the denominators of the two quotients to avoid dividing by zero. The term $d * (t_1 + t_2 e)$ is used to minimize the total misclassification errors, where the 'd' is a constant weight term. In our case study, $d = 1$, which means the two requirements, i.e., "equal balance between the Type I and the Type II error rates" and "lowest total error rates", were given the same weights. If $t_1 = t_2 = 0$, then $F_{obj} = 0$. However, this situation rarely happens in practice. If $t_1 \neq 0, t_2 \neq 0$, the minimal value of terms in the first parenthesis of Eq. (1) is equal to 2 when $t_1 = t_2$.

3. Case Study I

3.1. System description

This case study involves data collection efforts from the initial releases of two large Windows©-based embedded system applications used primarily for customizing the configuration of wireless telecommunications products. The two embedded applications (written in C++) provide similar functionalities, and contain common source code. The main difference between them is the type of wireless product that each supports. Both systems are comprised of over 1400 source code files and each of the files contained more than 27 million lines of codes.

Software inspection metrics were obtained by observing the configuration management systems of the applications. The problem reporting system tracked and recorded problem statuses. Information such as how many times a source file was inspected prior to system tests was logged in its database. Software metrics obtained reflected aspects of source files, and consequently, a module in this case study is comprised of a single source file. Many software metrics were collected to record information such as fault severity, inspection time, and major/minor errors. However, only a few primitive metrics were selected for modeling purposes since they provided the most relevant and concise information pertaining to the project.

The fault data collected for this case study represent the faults discovered during system tests. Upon preprocessing and cleaning the collected data, i.e., removal of outliers and illogical data instances, 1211 modules remained. Over 66% of modules (809) were observed to have no faults, and the remaining 402 modules had at least one or more faults. The five software metrics used for reliability modeling in this case study are presented in Table 2. The product metrics used are statement metrics for the source files. They primarily indicated the number of lines of source code prior to the coding phase (i.e., auto-generated code) and just before system tests. The inspection metric, *INSP*, was obtained from problem reporting systems of the two embedded applications. The whole dataset was split into two parts: FIT and TEST. Two thirds of the instances (807) were assigned to the fit dataset and the remaining one third was assigned to the test dataset. In order to avoid biased results due to the random selection process, the original dataset was randomly split 50 times to obtain 50 pairs of the FIT and TEST datasets. Empirical studies were then performed on the 50 split combinations. Since the RB2CBL model needs an extra dataset to optimize the parameters, each FIT dataset is further equally divided into two parts: *fit* (404) and *opt* (403), for building the RB2CBL model and optimizing its parameters, respectively. The TEST dataset is used for model evaluation.

3.2. Procedures

The experiment was implemented using the following steps:

1. Preprocess the dataset. In the original dataset, the dependent variable represents the number of faults of a program module. In order to classify the modules as fault-prone (*fp*) or not fault-prone (*nfp*), the dependent variable has to be converted into the class labelled *P* (*fp*) or *N* (*nfp*) by some given threshold value. Threshold-selection is an important decision since it can affect the usefulness of the classification model. For this system, it was decided that modules with 1 or more faults were considered as *fp*, and *nfp* if modules had no faults.
2. Build the RB2CBL model using the algorithm described in Figs. 3 and 4.
   - Build the RB (*C4.5*) model using the *fit* dataset.
   - Build the case libraries L_p and L_n using the *fit* dataset.

[1] $t_1 = \frac{|FN|}{|TP|+|FN|}$ and $t_2 = \frac{|FP|}{|FP|+|TN|}$ where $|\cdot|$ stands for the size of a given set.

Table 2
Software metrics.

Metrics	Description
B_LOC	Number of lines of code for the source file version prior to the coding phase, i.e., auto-generated code
S_LOC	Number of lines of code for the source file version delivered to system tests
B_COM	Number of lines of commented code for source file version prior to coding phase, i.e., auto-generated code
S_COM	Number of lines of commented code for source file version delivered to system tests
INSP	Number of times the source file was inspected prior to system tests

- Set the history record for the case libraries L_p and L_n using the *fit* and the *opt* datasets.
- Use the embedded GA-optimizer on the *opt* dataset to search for the optimized values of parameter α_p, α_n, thd_p, and thd_n.

3. Evaluate the RB2CBL model using the *test* dataset.

3.3. Results and analysis

RB models have mechanisms to control the termination of the model building and tree pruning process. For example, *C4.5* will stop its tree branch splitting in a subtree when the number of remaining data instances on the tree node is less than a given "minimal number of objects" (M_{obj}). This value will control the accuracy of the model. In practice, greater accuracy on the fit dataset does not always mean better prediction. In fact, greater accuracy on the fit dataset implies higher possibility of overfitting that may occur on the test dataset. Therefore, scrutiny and trade-off are needed in real-world applications. In order to inspect the effects of different accuracies of the RB models, three types of RB models were built: *LOOSE_RB*, *MID_RB*, and *TIGHT_RB*, which represent the models with low, medium, and high classification accuracies on the fit dataset, respectively. In our case study, these three types of models were instantiated via the *Cost Sensitive C4.5* model with M_{obj} = 30,5, and 1, respectively.

The *Cost Sensitive C4.5* model has other adjustable parameters, such as *CF* and *cost_ratio*. The parameter *CF* is used to control the level of pruning of the decision tree and *cost_ratio* gives relative cost between *FN* and *FP* [7,31]. In our case studies, the two parameters have been optimized manually to adjust the RB model and make the misclassification rates balanced and minimized. The dataset and parameters involved in the experiment are as follows:

1. Datasets: $|fit|$ = 404, $|opt|$ = 403 and $|test|$ = 404. The 50 splits of dataset were preprocessed by *threshold* = 1.
2. The parameters of the RB model (*Cost Sensitive C4.5*): M_{obj} was fixed at 30 for *LOOSE_RB*, 5 for *MID_RB* and 1 for *TIGHT_RB*. The preferred parameter values for *CF* and *cost_ratio* were obtained by manually optimization, i.e., *CF* = 0.25 and *cost_ratio* = 1.9.
3. The parameters of the CBL models: α_p, α_n, thd_p, and thd_n were optimized by the embedded GA-optimizer during RB2CBL modeling and the Euclidean distance was adopted as the similarity function in the CBL models.
4. The main parameters of the GA are as follows:
   - *Replacement percentage* = 0.5.
   - *Crossover probability* = 0.9.
   - *Mutation probability* = 0.08.
   - *Number of generations* = 4000.
   - *Size of population* = 200.
   - *Number of best solutions* = 1.
   - *Number of runs* = 3.

The genetic algorithm needs only a single measurement to evaluate a single individual when compared to the other individuals. The objective function provides this measurement. It is important to note the distinction between the fitness and objective scores. The objective score is the value returned by the objective function. It is the original performance evaluation of a genome. The fitness score, on the other hand, is a transformed measure used by the genetic algorithm to determine the fitness of individuals for mating. The fitness score is typically obtained by a linear scaling of the original objective scores, but you can define any mapping you would like, or no transformation at all. The genetic algorithm uses the fitness scores to do selection. In Galib, the translation from objective score to fitness score is implemented automatically.

The results of the *LOOSE_RB*, *MID_RB* and *TIGHT_RB* models are presented in Tables 3–5, respectively. The misclassification rates of Type I and Type II over the fifty data splits are reported, where t_1 and t_2 represent the Type I and Type II misclassification rates in percentage (%), respectively. Besides, the RB2CBL-related parameters α_p, α_n, thd_p, and thd_n that were tuned through GA are provided for all the preferred models. The bottom rows highlighted with bold provide the average and standard deviation values across the fifty splits. To evaluate the performance of the models, two measures, i.e., classification accuracy and the balance degree (absolute difference) of Type I and Type II error rates, were considered. In fact, the objective function (1) combines these two aspects. It returns a single value to the GA for the parameter evaluation and evaluation for

Table 3
LOOSE_RB model – *C4.5* with M_{obj} = 30.

Data split	Parameters				*fit*				*test*			
	α_p	α_n	thd_p	thd_n	RB (*C4.5*)		RB2CBL		RB (*C4.5*)		RB2CBL	
					t_1%	t_2%	t_1%	t_2%	t_1%	t_2%	t_1%	t_2%
1	0.1653	0.032	−167	43	37.55	11.19	23.42	20.9	38.89	9.7	24.44	19.4
2	0.0024	0.0001	−10	−12	28.62	17.78	19.33	12.59	25.19	24.81	20.74	17.29
3	0.0034	0.0157	−11	11	20.07	30.6	18.59	18.66	22.59	29.1	25.19	20.9
4	0.0032	0.0175	−12	10	19.7	32.59	22.68	24.44	20.52	29.63	19.78	20
5	0.1979	0.0035	−306	2	24.72	25	19.1	18.38	25.09	27.07	20.66	20.3
6	0.0028	0.1413	1	4	24.44	27.82	24.81	21.8	25.93	28.36	25.56	20.15
7	0.0121	0.0174	−14	12	20.07	42.54	20.82	17.91	21.85	41.79	19.63	26.12
8	0.061	0.002	−52	−1	27.51	24.63	18.59	17.91	27.14	28.89	18.96	24.44
9	0.1548	0.0816	−66	8	18.28	43.7	23.88	24.44	19.26	36.57	25.56	26.12
10	0.0225	0.0085	−7	8	39.03	11.19	21.19	17.91	35.45	13.97	19.03	22.06
11	0.0064	0.0167	−2	24	35.96	15.33	20.97	26.28	37.73	9.23	19.78	19.23
12	0.1107	0.0053	4	−50	37.5	12.98	20.22	18.32	35.58	13.87	20.6	21.17
13	0.0006	0.0264	1	14	20.68	29.2	18.42	17.52	19.63	31.34	19.26	17.91
14	0.0003	0.0135	−1	6	24.16	13.33	18.59	11.85	28.41	20.45	19.19	16.67
15	0.0723	0.0008	−151	2	28.52	20.9	18.89	22.39	28.57	19.23	19.41	19.23
16	0.132	0.01	−7	7	32.72	21.37	25	22.14	30.97	23.53	23.88	24.26
17	0.0029	0.0133	−13	2	38.95	14.6	20.97	18.98	34.34	12.32	15.09	13.77
18	0.0003	0.0027	−379	7	21.21	32.37	20.45	20.86	21.11	36.57	17.78	23.88
19	0.0285	0.0006	−7	2	28.62	23.13	20.45	18.66	32.1	22.56	28.04	22.56
20	0.0006	0.0062	0	1	46.3	13.53	21.48	18.8	43.22	16.79	21.61	18.32
21	0.0003	0.0017	2	2	28.68	21.21	23.16	19.7	24.26	17.56	16.91	15.27
22	0.0173	0.1067	−6	1	19.56	30.3	19.93	18.18	18.01	32.87	21.07	23.78
23	0.0219	0.0002	−20	2	13.79	36.36	24.14	19.58	15.67	31.11	20.9	19.26
24	0.148	0.0014	−144	1	34.7	19.26	22.01	19.26	32.72	25.76	22.06	28.79
25	0.0747	0.0007	−254	1	26.84	19.08	20.96	18.32	26.81	13.28	19.2	15.63
26	0.0633	0.0011	−64	−25	16	30.47	21.82	18.75	17.11	35.46	25.1	19.86
27	0.0058	0.0155	5	12	27.1	21.13	24.43	17.61	25.86	22.14	22.43	18.57
28	0.0106	0.0016	1	−21	36.12	14.29	22.43	18.57	37.78	13.43	24.81	15.67
29	0.1288	0.0309	−178	45	37.17	7.46	20.07	20.15	34.32	9.77	21.03	27.82
30	0.0024	0.1786	2	41	24.44	36.09	24.81	21.8	25.84	43.8	26.59	29.2
31	0.0017	0	−1	−56	28.46	21.17	14.98	18.25	25.18	23.26	17.88	14.73
32	0.0775	0.1935	−254	41	26.01	26.15	26.37	24.62	25.75	27.94	25.75	26.47
33	0.1827	0.0022	−57	4	28.09	22.79	22.47	22.79	29.59	24.09	25.09	24.09
34	0.1119	0	−247	−3	29.7	18.25	23.31	19.71	30.8	18.75	20.29	20.31
35	0.0722	0.0235	−156	14	28.73	11.72	20.36	17.19	29.26	14.18	21.48	17.16
36	0.0028	0.0111	−452	8	24.91	26.87	18.22	16.42	26.39	31.85	21.56	22.22
37	0.0248	0.0041	−6	10	24.54	27.61	21.56	18.66	22.43	22.7	22.81	17.73
38	0.0174	0.001	−10	−1	34.6	16.31	23.57	19.86	32.25	14.96	16.67	14.96
39	0.0055	0.0036	0	5	19.57	35.43	15.58	15.75	20.59	38.64	20.96	24.24
40	0.1583	0.0387	−126	46	42.8	8.33	25.46	21.21	42.96	5.97	24.81	20.15
41	0.0044	0.0115	0	4	30.11	29.1	25.28	23.13	29.75	25.6	27.96	21.6
42	0.0477	0.028	−65	20	32.73	18.25	20.14	23.02	30.65	14.08	19.54	20.42
43	0.0071	0.0022	−2	2	27.69	18.18	19.62	16.78	26.09	17.97	21.38	21.09
44	0.0357	0.0065	−1	1	21.45	21.09	16	15.63	23.6	27.01	20.6	23.36
45	0.0641	0.0065	−44	2	24.44	34.06	19.55	26.09	22.85	26.47	19.48	19.12
46	0.0088	0.0085	−4	8	21.35	28.68	19.85	18.38	21.77	24.06	21.77	15.04
47	0.0463	0.0009	−32	−44	32.22	13.43	20.74	17.16	33.21	15.22	20.38	17.39
48	0.0001	0.0001	−3	−34	24.24	25.9	20.83	20.14	23.62	21.8	19.56	13.53
49	0.1296	0.0306	−21	23	51.11	5.26	26.67	21.8	52.35	5.51	26.35	20.47
50	0.1694	0.0086	−17	2	34.78	8.59	21.74	14.06	33.58	12.32	19.62	16.67
Avg.	0.052	0.023	−67	4	**28.53**	**22.33**	**21.28**	**19.47**	**28.29**	**22.67**	**21.56**	**20.37**
Stdev.	0.061	0.043	108	21	**7.81**	**9.16**	**2.66**	**3.07**	**7.33**	**9.29**	**3.00**	**3.91**

each round of loop of the GA. For the purpose of illustration, we separate it into two parts, i.e., *absolute difference* (*AD*) and *total of Type I and Type II error rates* (*TE*). They are defined as follows:

$$AD = |t_1 - t_2|, \tag{2}$$

$$TE = t_1 + t_2, \tag{3}$$

where $|\cdot|$ stands for the absolute value of a given number. A smaller *AD* means a better balance result between the Type I and Type II error rates and a smaller *TE* represents better accuracy of classification. It is worthwhile noting that these two measures should be considered together for our high-assurance embedded system when training and evaluating models. Emphasizing one over the other could cause misleading for software quality assurance practice.

Table 4
MID_RB model – *C4.5* with $M_{obj} = 5$.

Data split	Parameters				*fit*				*test*			
	α_p	α_n	thd_p	thd_n	RB (*C4.5*)		RB2CBL		RB (*C4.5*)		RB2CBL	
					t_1%	t_2%	t_1%	t_2%	t_1%	t_2%	t_1%	t_2%
1	0.1499	0.0091	−41	1	23.42	13.43	16.36	13.18	22.59	17.16	17.41	18.4
2	0.0019	0.1527	−6	−3	17.84	14.07	17.84	14.56	17.04	20.3	17.41	15.54
3	0.001	0.1615	−7	34	18.22	20.15	19.33	16.16	17.04	23.88	18.15	19.15
4	0.0076	0.1189	−7	3	21.93	29.1	22.68	22.13	18.28	22.06	18.66	16.65
5	0.1937	0.1092	−393	3	17.23	17.52	17.98	16.52	17.34	19.7	18.08	17.18
6	0.0028	0.0026	−1	3	19.26	17.16	17.41	18.4	20.74	19.55	17.41	16.29
7	0.1032	0.0001	−343	−55	18.59	26.67	19.33	24.19	18.15	21.8	18.89	15.54
8	0.0681	0.0762	−16	2	17.1	20.74	21.19	16.78	17.47	23.88	18.59	16.16
9	0.003	0.0051	−9	1	26.87	17.78	19.4	16.78	26.67	16.42	24.44	13.18
10	0.1404	0.0061	−388	1	20.07	10.45	14.13	10.94	20.52	17.65	15.3	18.85
11	0.0944	0.004	−170	−9	19.1	25	17.6	16.65	21.25	19.85	19.41	16.56
12	0.0005	0.0089	−61	−9	21.32	23.66	17.28	15.79	20.22	25.55	16.1	25.28
13	0.0073	0.0061	−1	0	19.55	23.19	17.67	25.09	17.41	17.29	15.93	14.79
14	0.1811	0.0012	−136	2	22.3	12.59	19.33	12.33	25.46	19.7	21.03	18.7
15	0.0527	0.0239	−38	8	20	24.06	19.63	18.55	20.15	20.61	20.51	16.56
16	0.0061	0.0001	−1	−9	26.47	15.27	18.75	15.03	26.49	13.24	19.4	12.97
17	0.0371	0.0036	−17	−5	28.84	25	22.85	19.59	23.4	24.46	20.38	21.3
18	0.0032	0.0127	−17	2	13.64	37.41	23.48	22.02	12.22	31.34	23.7	16.91
19	0.1967	0.0115	−18	5	21.56	18.52	18.22	19.74	25.83	17.42	21.4	17.18
20	0.179	0.0174	−86	−3	25.19	16.54	20.74	17.8	23.08	13.74	16.12	17.32
21	0.0696	0.0173	0	0	22.43	19.85	17.28	15.03	21.32	15.91	15.81	14.91
22	0.0063	0.0006	−9	2	14.76	22.73	18.82	16.42	12.26	28.67	15.71	21.38
23	0.0027	0.0258	−13	43	14.94	22.38	16.86	20.68	17.54	20	17.91	16.78
24	0.1233	0.0007	−304	−1	26.49	16.3	20.15	15.3	25	19.7	19.85	19.45
25	0.0099	0.1053	−2	−2	21.32	21.97	21.32	20.21	18.12	18.9	17.39	16.32
26	0.1193	0.0042	0	2	16.36	19.53	15.27	13.84	17.87	27.66	16.73	23.82
27	0.0002	0.0072	−1	4	17.94	21.13	17.56	18.01	20.15	20.71	18.63	16.86
28	0.0002	0.0034	3	0	22.43	17.14	18.63	15.43	19.63	17.91	15.56	16.16
29	0.0923	0.0003	−19	2	23.05	15.56	20.07	15.3	20.66	15.91	16.97	14.15
30	0.0129	0.0001	−19	2	18.52	22.56	17.78	17.05	20.22	27.74	19.1	24.55
31	0.1643	0.0008	−147	2	20.6	17.65	16.48	15.18	19.34	16.15	16.42	15.92
32	0.0082	0.1216	−4	40	17.58	25.95	19.78	19.61	17.54	23.7	19.03	17.52
33	0.0739	0.1676	−43	0	11.99	27.01	20.22	24.55	15.36	31.62	20.97	20.32
34	0.0038	0.1117	−1	1	18.05	21.01	19.55	16.39	19.57	21.26	19.57	18.69
35	0.0122	0.0024	−6	1	19.27	24.22	20	17.75	17.78	16.42	19.26	10.94
36	0.0069	0.0022	1	−7	23.79	10.45	14.87	11.69	20.07	21.48	13.75	22.7
37	0.1914	0.1586	−229	−4	17.1	24.44	16.36	24.19	20.15	18.57	19.77	17.57
38	0.0302	0	−4	1	20.91	19.29	19.01	16.14	15.22	17.97	13.77	15.41
39	0.0301	0.0002	−18	1	26.09	19.69	19.2	15.54	27.21	19.7	23.9	17.94
40	0.0004	0.005	−373	−55	17.71	13.53	16.61	14.04	23.33	13.53	17.04	15.54
41	0.0898	0.0006	−275	2	24.16	17.04	24.91	13.81	19.71	21.77	19.35	17.55
42	0.1106	0.0037	−225	2	14.75	33.33	17.63	22.02	18.01	32.39	22.99	22.24
43	0.0823	0.0002	−27	−32	22.31	16.78	18.08	15.08	22.83	13.28	19.2	12.28
44	0.073	0.0005	−34	−2	18.18	24.22	18.91	16.97	16.48	30.66	16.1	24.55
45	0.068	0.041	−10	2	16.54	23.36	18.05	16.52	16.1	27.01	17.6	22.36
46	0.073	0.0089	−373	2	18.35	20.44	18.35	15.06	18.45	18.18	16.97	15.67
47	0.0016	0.0229	4	11	24.44	18.8	19.26	16.29	23.4	15.11	18.11	16.27
48	0.0129	0.0058	0	2	20.83	10.71	18.18	11.86	22.14	12.12	15.13	11.88
49	0.0246	0.0297	−2	3	23.33	13.43	15.93	19.9	25.99	13.49	16.25	14.08
50	0.1522	0.0178	−11	−5	19.2	11.02	14.49	13.17	19.25	19.42	14.72	22.02
Avg.	0.062	0.032	−78	0	**20.24**	**20.00**	**18.62**	**17.11**	**20.04**	**20.45**	**18.24**	**17.61**
Stdev.	0.064	0.051	124	16	**3.68**	**5.66**	**2.19**	**3.42**	**3.50**	**5.13**	**2.47**	**3.36**

Figs. 5–7 present the two measures for the *LOOSE_RB*, *MID_RB* and *TIGHT_RB* models on both *fit* and *test* datasets across all fifty data splits, respectively. They intuitively reflect the performance of two types of the models (RB vs RB2CBL). Table 6 is a summary of the results on the three models: *LOOSE_RB*, *MID_RB* and *TIGHT_RB*. From the figures and Table 6, some important facts are observed.

1. The *LOOSE_RB* model got the most significant improvement from the RB model to the RB2CBL model. This can be observed from Fig. 5(b) and (d), where the RB2CBL curves are under the RB curves. In addition, Table 6 shows that the Type I and Type II misclassification rates decreased 7.25% and 2.86%, respectively, for the *fit* dataset, and 6.72% and 2.3%, respectively, for the *test* dataset. Therefore, the largest gain highlighted with bold shown on the last column of the table was obtained: 10.12% and 9.02% for the *fit* and *test* datasets, respectively.

Table 5
TIGHT_RB model – *C4.5* with $M_{obj} = 1$.

Data split	Parameters				*fit*				*test*			
	α_p	α_n	thd_p	thd_n	RB (*C4.5*)		RB2CBL		RB (*C4.5*)		RB2CBL	
					t_1%	t_2%	t_1%	t_2%	t_1%	t_2%	t_1%	t_2%
1	0.1871	0.1535	−155	−13	16.73	15.67	16.36	14.18	16.67	18.66	17.78	18.66
2	0.0041	0.0223	−6	2	16.73	16.42	15.99	15.67	15.19	23.13	15.19	20.9
3	0.02	0.064	−29	12	13.01	23.88	20.45	20.9	15.93	23.13	22.96	21.64
4	0.0303	0.1873	−23	38	17.84	35.07	27.51	29.1	14.93	22.79	26.12	16.91
5	0.0617	0.0009	−83	2	17.98	20.59	17.6	17.65	14.76	27.82	15.13	24.81
6	0.0036	0.0559	−204	2	12.96	27.07	21.11	21.05	14.44	26.12	24.44	22.39
7	0.0097	0.003	−17	3	19.33	20.15	17.1	17.16	18.89	18.66	18.52	17.91
8	0.0993	0.0445	−217	1	13.75	25.37	17.84	17.91	13.01	22.22	20.45	22.22
9	0.1923	0.0149	−333	2	20.15	20.74	17.91	17.78	19.63	17.91	20	18.66
10	0.0012	0.1196	−35	7	15.61	17.91	17.84	16.42	17.16	22.06	19.03	22.06
11	0.0155	0.019	−89	2	13.48	19.85	20.22	18.38	14.65	16.79	20.15	16.79
12	0.0014	0.0196	−180	12	16.18	29.77	22.43	22.9	14.61	27.01	20.22	23.36
13	0.1063	0.1274	−290	−2	11.28	32.85	23.31	21.9	8.52	32.84	23.7	21.64
14	0.04	0.0049	−258	−2	14.13	20.15	19.33	18.66	18.45	22.56	22.88	18.8
15	0.0003	0.1684	−1	43	19.63	20.3	19.63	19.55	18.68	19.08	18.68	19.08
16	0.1631	0.0019	−189	2	20.22	17.56	17.65	17.56	21.27	20.59	17.91	20.59
17	0.0324	0.0065	−171	−4	15.73	22.79	21.72	22.06	14.72	25.9	21.51	25.9
18	0.041	0.0849	−113	1	17.8	20.86	20.08	18.71	14.81	21.64	15.93	19.4
19	0.0001	0.0439	−2	22	20.45	22.39	20.45	19.4	23.25	21.8	23.62	20.3
20	0.1657	0.0134	−15	1	20.74	18.05	18.15	18.05	19.78	17.56	19.41	18.32
21	0.0109	0.0092	−11	2	18.38	19.85	18.01	16.79	17.28	15.15	19.49	16.67
22	0.0607	0.1294	−120	−3	15.87	21.97	21.03	19.7	11.88	25.87	17.62	28.67
23	0.0959	0.1135	−208	37	12.26	23.94	19.54	19.72	12.31	25.74	21.64	20.59
24	0.1187	0.0102	−74	31	17.16	17.04	17.16	17.04	15.07	24.24	15.07	24.24
25	0.0363	0.0646	−13	3	19.49	23.66	19.85	19.85	15.58	19.53	18.12	18.75
26	0.1955	0.0001	−103	1	17.09	17.97	16.73	16.41	17.11	21.28	16.73	16.31
27	0.0088	0.0003	−188	−28	14.5	26.24	23.66	21.28	12.55	24.82	20.53	19.15
28	0.0027	0.0021	−120	−6	23.95	14.29	19.01	16.43	22.59	13.43	16.3	17.91
29	0.0036	0.0002	0	2	20.07	18.66	17.84	17.91	18.45	19.55	17.34	19.55
30	0.0024	0.0993	3	21	16.67	17.29	16.67	16.54	18.73	26.28	18.73	25.55
31	0.0031	0.1809	−111	40	17.98	24.26	22.85	22.06	15.33	23.08	21.53	21.54
32	0.0036	0.0238	−16	1	16.85	26.15	19.41	19.23	15.3	27.21	20.15	22.79
33	0.0055	0.1132	−67	1	13.48	22.06	21.35	19.12	17.6	29.93	23.6	31.39
34	0.019	0.061	−14	0	15.04	29.2	21.05	21.17	14.13	27.34	19.57	22.66
35	0.0741	0.001	−194	4	22.55	21.09	21.09	21.09	21.11	13.43	20.74	13.43
36	0.1989	0.0018	−229	−40	19.33	12.69	16.36	14.18	14.5	23.7	13.38	25.93
37	0.0033	0.0025	−156	3	16.36	24.63	21.93	22.39	20.53	16.31	26.62	16.31
38	0.0015	0.1462	−7	28	19.39	21.43	20.53	20.71	16.3	21.88	19.2	21.88
39	0.0065	0.1346	−11	24	22.83	23.62	22.83	22.05	23.53	20.45	23.9	18.94
40	0.0013	0.0264	−25	0	12.92	18.18	16.97	16.67	16.67	16.42	20.74	17.91
41	0.1727	0.0044	−404	−1	22.68	14.93	18.59	15.67	19	21.6	15.77	22.4
42	0.1897	0.1309	−104	35	23.02	20	23.02	20	23.75	25.87	23.75	25.87
43	0.115	0.0057	−314	5	19.23	24.48	20.77	20.98	17.75	23.44	18.48	21.09
44	0.0056	0.01	−290	−15	12.73	21.09	18.91	17.19	13.48	32.85	22.85	29.2
45	0.0007	0.0523	−429	1	15.41	18.98	19.17	18.98	17.6	19.71	20.97	19.71
46	0.0032	0.0156	−186	−11	11.99	27.21	19.85	19.85	14.76	30.08	25.83	26.32
47	0.12	0.0788	−250	−2	11.48	24.81	23.33	21.05	11.7	24.46	23.4	20.14
48	0.105	0.0001	−270	1	20.08	15.11	17.42	15.11	19.56	18.05	17.34	18.05
49	0.0923	0.0033	−96	3	20.74	16.54	16.67	16.54	18.77	13.39	18.05	14.17
50	0.1502	0.0221	−217	0	19.2	11.81	14.86	13.39	18.11	18.71	16.23	22.3
Avg.	0.060	0.052	−133	5	**17.25**	**21.33**	**19.58**	**18.88**	**16.81**	**22.24**	**19.95**	**21.00**
Stdev.	0.068	0.058	115	16	**3.33**	**4.84**	**2.50**	**2.78**	**3.25**	**4.68**	**3.15**	**3.75**

2. The *TIGHT_RB* model never showed improvement of the classification accuracy for both *fit* and *test* datasets over the RB2CBL model. This can also be observed from Fig. 7(b) and (d), where the two model curves are at the same horizontal level. Even though the Type II error was slightly reduced, the Type I error was increased. In addition, a small amount of overfitting occurs from *fit* to *test*. More specifically, the Type II error rate of RB2CBL increased over 2% from *fit* to *test*. However, the TIGHT_RB model got the best predictive accuracy when the RB models were used alone (highlighted with bold).
3. The RB2CBL model in the *MID_RB* group showed the highest accuracy. The Type I and Type II misclassification rates are 18.62% and 17.11%, respectively, for the *fit* dataset and 18.24% and 17.61%, respectively, for the *test* dataset. Therefore, the total of the Type I and Type II error rates is the lowest among all the models: 35.72% and 35.84% for the *fit* and *test* datasets, respectively (highlighted with bold).

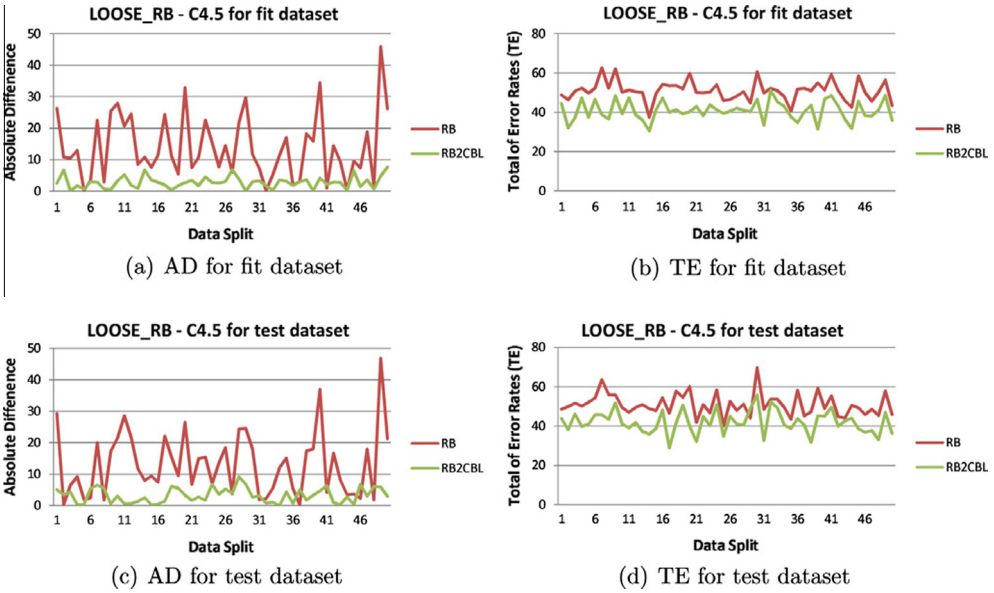

Fig. 5. *LOOSE_RB* model – RB vs. RB2CBL.

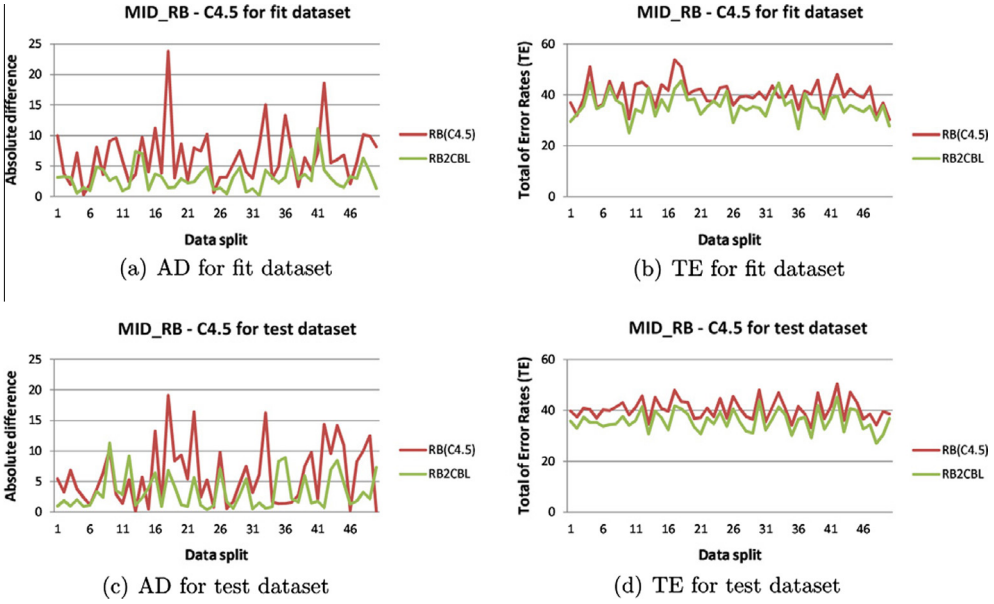

Fig. 6. *MID_RB* model – RB vs. RB2CBL.

4. The balances between the Type I and Type II errors of the RB2CBL models are significantly improved as compared with their respective pure RB models. Furthermore, by inspecting the bottom line in Tables 3–5, it is found that the deviation of the error rates among 50 splits were unanimously improved (decreased) when the RB model was enhanced to the RB2CBL model. This can also be observed from Figs. 5(a) and (c), 6(a) and (c), and 7(a) and (c), where the RB2CBL curves are less skewed than the RB curves.

Among the three types of RB models, the *LOOSE_RB* model had the lowest accuracy and thereby left more space to the CBLs for improvement. As a result, significant accuracy improvement was obtained. In contrast, the *TIGHT_RB* model had the highest classification accuracy on the *fit* dataset but overfitting was likely to occur on the *test* dataset. There is little space left for the CBL's adjustment. This situation may lead to both the RB and RB2CBL models overfitting and the unexpected final result. In fact, a midpoint exists between the *LOOSE* and *TIGHT* models, which is a tradeoff between the RB model and the CBL

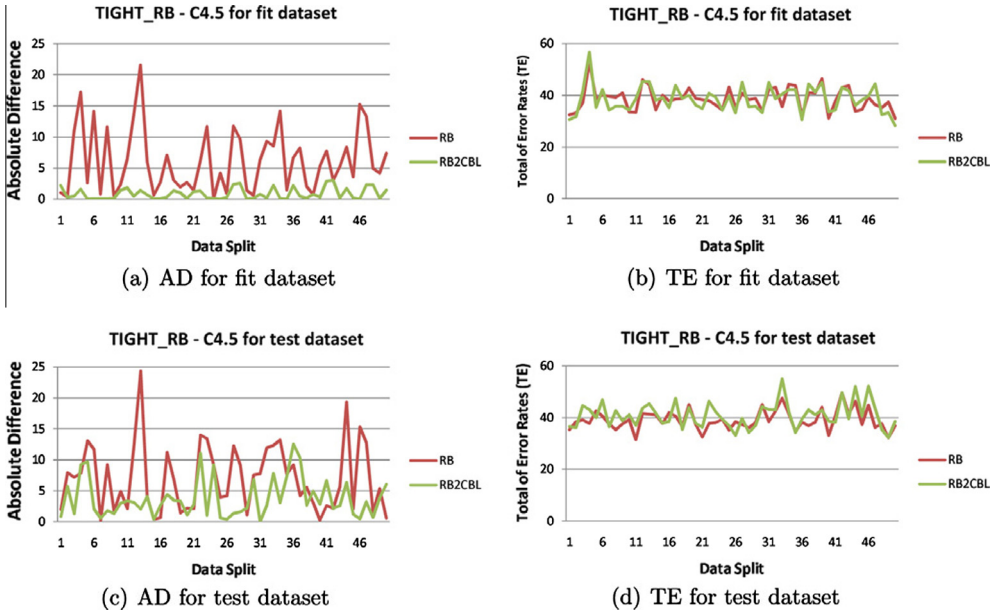

Fig. 7. *TIGHT_RB* model – RB vs. RB2CBL.

Table 6
Summary: RB vs. RB2CBL.

Model type (M_{obj})	Data set	RB (*C4.5*)				RB2CBL				*Gain*[a] %
		t_1%	t_2%	*AD*%	*TE*%	t_1%	t_2%	*AD*%	*TE*%	
LOOSE_RB (30)	*fit test*	28.53	22.33	13.95	50.86	21.28	19.47	2.81	40.74	**10.12**
		28.29	22.67	13.15	50.95	21.57	20.37	3.2	41.93	**9.02**
MID_RB (5)	*fit test*	20.24	20	6.62	40.24	18.62	17.11	3.09	**35.72**	4.52
		20.04	20.45	5.98	40.49	18.24	17.61	3.3	**35.84**	4.65
TIGHT_RB (1)	*fit test*	17.25	21.33	6.22	**38.58**	19.58	18.88	0.87	38.46	0.12
		16.81	22.24	7.02	**39.05**	19.95	21	3.77	40.94	−1.89

[a] $Gain = TE_{RB} - TE_{RB2CBL}$.

models. An RB2CBL model on the midpoint will build its RB with suitable accuracy without overfitting, which leaves enough space to bring CBLs into full play. It will create the best prediction accuracy and balance compared to the other (*LOOSE* or *TIGHT*) RB2CBL models.

4. Case Study II

In Case Study I, the RB2CBL model was applied for an industrial scale dataset and the impressive results with improved classification accuracy and no overfitting were obtained. However, the general behavior for RB2CBL in different types of datasets needs to be further investigated. In this case study, we explore them using two groups of simulated datasets.

4.1. Simulated datasets

Most of the rule based algorithms are based on the separate and conquer paradigm. For that reason these algorithms are capable of finding axis-parallel frontiers for a given dataset. On the other hand, the CBL approach is good at dealing with non-axis-parallel frontiers and clustered or clumped data points scattered on a dataset. To investigate the effects of the RB2CBL model, one of the effective ways could be intensively scattering clustered data points on a known dataset and evaluating its classification results. For doing so, the following steps were performed:

1. Randomly take a set of data points, R, without redundancy, from the original dataset of Case Study I.
2. For each data point $r \in R$, create some data points with the same or different classes as r in a neighborhood around. The number of data points created, the size of the neighborhood, and the class of each point are all determined randomly within the given domains.

3. Insert the artificial data points generated from step 2 on the original dataset by some given proportion.

Two groups of datasets were obtained in this way. Group SIMUL20 includes 10 datasets and each of them was randomly created with 20% extra clustered data points inserted. The statistics on 10 datasets could eliminate or reduce the bias outcomes due to a single dataset used alone. Similarly, Group SIMUL10 includes 10 datasets and each of them was randomly created with 10% extra clustered data points inserted. Then, we split each dataset into *fit*, *opt* and *test* the same way as we did in the first case study.

Table 7
Experiment for SIMUL20.

Data set	Split no.	RB (*C4.5*)				RB2CBL				*Gain*[a] %
		t_1%	t_2%	*AD%*	*TE%*	t_1%	t_2%	*AD%*	*TE%*	
fit	1	29.77	21.02	8.75	50.79	21.39	19.11	2.28	40.50	10.29
	2	45.95	14.65	31.30	60.60	24.86	23.57	1.29	48.43	12.17
	3	31.21	18.47	12.74	49.68	21.10	17.83	3.27	38.93	10.75
	4	34.39	21.66	12.73	56.05	22.83	22.29	0.54	45.12	10.93
	5	29.77	25.48	4.29	55.25	24.86	21.66	3.20	46.52	8.73
	6	40.17	18.47	21.70	58.64	24.86	23.57	1.29	48.43	10.21
	7	23.99	24.20	0.21	48.19	22.25	19.11	3.14	41.36	6.83
	8	29.48	19.11	10.37	48.59	20.52	20.38	0.14	40.90	7.69
	9	31.50	20.38	11.12	51.88	21.39	18.47	2.92	39.86	12.02
	10	31.79	28.03	3.76	59.82	26.59	23.57	3.02	50.16	9.66
	Avg.	32.80	21.15	**11.70**	**53.95**	23.07	20.96	**2.11**	**44.02**	**9.93**
test	1	32.28	22.29	9.99	54.57	25.65	24.20	1.45	49.85	4.72
	2	43.80	14.01	29.79	57.81	26.51	22.93	3.58	49.44	8.37
	3	33.43	19.75	13.68	53.18	23.92	19.11	4.81	43.03	10.15
	4	36.60	19.11	17.49	55.71	24.21	23.57	0.64	47.78	7.93
	5	30.55	20.38	10.17	50.93	27.95	20.38	7.57	48.33	2.60
	6	37.46	15.92	21.54	53.38	23.34	22.29	1.05	45.63	7.75
	7	25.65	22.29	3.36	47.94	24.21	19.11	5.10	43.32	4.62
	8	31.99	20.38	11.61	52.37	25.07	22.93	2.14	48.00	4.37
	9	34.01	16.56	17.45	50.57	25.65	13.38	12.27	39.03	11.54
	10	32.56	19.75	12.81	52.31	27.95	14.65	13.30	42.60	9.71
	Avg.	33.83	19.04	**14.79**	**52.88**	25.45	20.26	**5.19**	**45.70**	**7.18**

[a] $Gain = TE_{RB} - TE_{RB2CBL}$.

Table 8
Experiment for SIMUL10.

Data set	Split no.	RB (*C4.5*)				RB2CBL				*Gain*[a] %
		t_1%	t_2%	*AD%*	*TE%*	t_1%	t_2%	*AD%*	*TE%*	
fit	1	27.36	19.33	8.03	46.69	20.97	18.49	2.48	39.46	7.23
	2	28.27	23.53	4.74	51.80	22.80	21.85	0.95	44.65	7.15
	3	25.53	26.05	0.52	51.58	24.01	22.69	1.32	46.70	4.88
	4	29.48	24.37	5.11	53.85	23.40	21.85	1.55	45.25	8.60
	5	27.36	31.09	3.73	58.45	27.36	26.05	1.31	53.41	5.04
	6	34.04	18.49	15.55	52.53	24.92	22.69	2.23	47.61	4.92
	7	26.75	28.57	1.82	55.32	25.84	23.53	2.31	49.37	5.95
	8	24.01	18.49	5.52	42.50	17.33	15.97	1.36	33.30	9.20
	9	24.32	26.05	1.73	50.37	23.10	21.01	2.09	44.11	6.26
	10	26.14	21.01	5.13	47.15	22.19	20.17	2.02	42.36	4.79
	Avg.	27.33	23.70	**5.19**	**51.02**	23.19	21.43	**1.76**	**44.62**	**6.40**
test	1	26.67	21.01	5.66	47.68	23.64	21.85	1.79	45.49	2.19
	2	27.88	20.17	7.71	48.05	23.94	19.33	4.61	43.27	4.78
	3	27.58	22.69	4.89	50.27	25.45	26.05	0.60	51.50	−1.23
	4	29.70	24.37	5.33	54.07	22.42	26.05	3.63	48.47	5.60
	5	25.15	26.89	1.74	52.04	25.76	21.01	4.75	46.77	5.27
	6	34.24	14.29	19.95	48.53	22.73	19.33	3.40	42.06	6.47
	7	27.27	23.53	3.74	50.80	25.76	18.49	7.27	44.25	6.55
	8	26.06	16.81	9.25	42.87	18.48	13.45	5.03	31.93	10.94
	9	23.33	21.01	2.32	44.34	23.33	18.49	4.84	41.82	2.52
	10	25.45	19.33	6.12	44.78	23.33	20.17	3.16	43.50	1.28
	Avg.	27.33	21.01	**6.67**	**48.34**	23.48	20.42	**3.91**	**43.91**	**4.44**

[a] $Gain = TE_{RB} - TE_{RB2CBL}$.

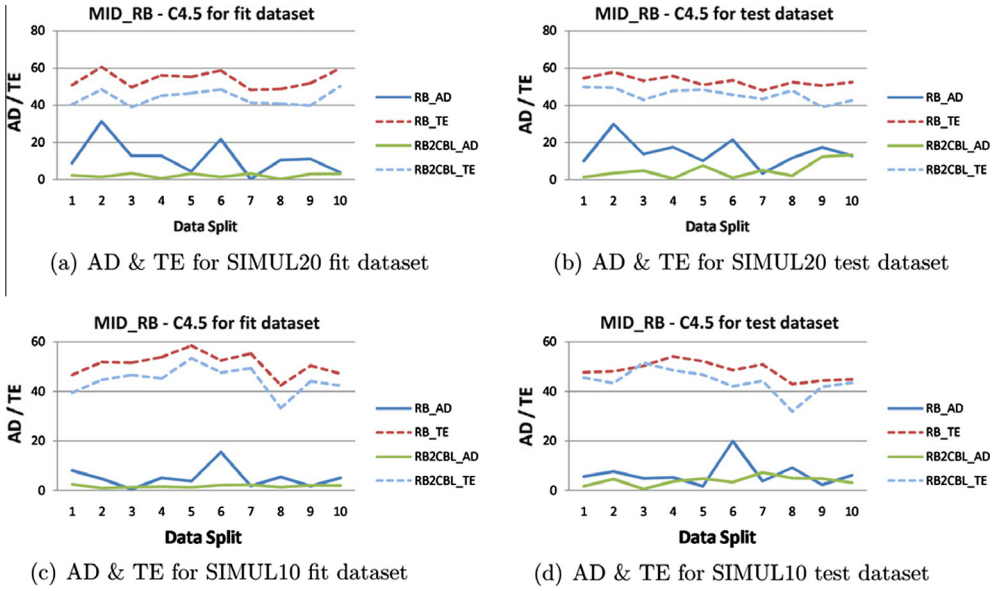

Fig. 8. *MID_RB* model – RB vs. RB2CBL for the simulated datasets.

4.2. Results and analysis

The experiments were carried out the same way as presented in Case Study I, however, we used the *MID_RB* only this time, since we obtained better result when using the *MID_RB* model for Case Study I. The parameters involved in the experiment are summarized as follows:

- The parameters of the RB model (*Cost Sensitive C4.5*): $CF = 0.25$,$M_{obj} = 5$, *cost_ratio* = 2.7 for SIMUL20 datasets and *cost_ratio* = 3.0 for SIMUL10 datasets. (These parameters have been optimized manually when balancing the Type I and Type II misclassification rates for the RB model.)
- The parameters of the *CBL* models: α_p, α_n, thd_p, and thd_n were optimized by the embedded GA-optimizer during RB2CBL modeling and the Euclidean distance was used as the similarity function for the CBL models.
- The parameters of the *GA*: same as those presented in Case Study I.

Tables 7 and 8 present the experimental results for the data groups SIMUL20 and SIMUL10, respectively. The average performance (in terms of AD and TE) of RB and RB2CBL and the Gain over the 10 data splits are also presented and highlighted with bold. Note that the values of the Type I and Type II misclassification rates themselves in Tables 7 and 8 are incomparable to each other, because the datasets SIMUL20 and SIMUL10 have different sizes and different proportions for extra artificial data points inserted. Fig. 8 is a graphical representation of the performance for RB models against RB2CBL models. Some notable improvements from stand-alone RB to the cascaded RB2CBL model can be observed.

- The different accuracy improvements have been made when the RB model was enhanced to the RB2CBL model for two groups of datasets. The Type I misclassification rates on average were significantly decreased for both fit and test datasets: 9.73% and 8.38% respectively for SIMUL20 and 4.14% and 3.85% respectively for SIMUL10. The Type II misclassification rates remained similar. Therefore, the improvement of the Type I misclassification rate dominates the gain value shown in the last column of the tables.
- The balance between the Type I and Type II errors was significantly improved from pure RB to the cascaded RB2CBL model for both SIMUL20 and SIMUL10. This can also be seen from the different fluctuations of the *AD* curves in Fig. 8.

The results of this study demonstrate that the more small disjuncts a dataset contains, the more improvement the RB2CBL model would make.

5. Conclusion

By cascading an RB and two CBL models, a new classifier architecture, RB2CBL, is proposed. RB2CBL integrates the advantages of both RB and CBL models and restrains their drawbacks.

In the first case study, a full-scale industrial software system has been processed by the RB2CBL model. Three types of models, *LOOSE_RB*, *MID_RB*, and *TIGHT_RB*, were inspected. The experiments suggest that, by suitably choosing the accuracy

of the RB model, the best result of the RB2CBL model can be obtained without overfitting. Also, it outperforms the best result of the RB model alone.

To gain a better understanding of the RB2CBL model, an additional case study on two groups of simulated datasets was conducted. The experiments on the simulated datasets certified our observation: RB2CBL models are good at dealing with small disjuncts, which is the drawback of the original RB model.

In this paper, our discussion of the rule-based system is limited to decision trees. The performance analysis of different rule-based systems is an interesting research topic. Also, the performance comparison of the experimental results between RB2CBL and output results of other modeling methods besides the conventional RB will be studied. Furthermore, the two-group classification problem for the RB2CBL model could be extended to a multi-group classification problem. These issues will be investigated in our future work.

Acknowledgments

We thank the Guest Editor, Dr. Marek Reformat, and the anonymous reviewers for their constructive comments and suggestions. We are grateful to the current and former members of the Empirical Software Engineering and Data Mining and Machine Learning Laboratories at Florida Atlantic University for their reviews and comments.

References

[1] D.W. Aha, Case-based learning algorithms, in: Proceedings of the DARPA Case-Based Reasoning Workshop, 1991, pp. 147–158.
[2] D.W. Aha, D. Kibler, M. Albert, Instance-based learning algorithms, Machine Learning 6 (1991) 37–66.
[3] D.W. Aha, M. Molineaux, M. Ponsen, Learning to win: case-based plan selection in a real-time strategy game, AI Magazine 27 (2006) 75–84.
[4] S. Ali, B. Pang, K. Tackle, Rule based base classifier selection for bagging algorithm, in: Proceedings of the 2008 International Conference on Data Mining, Las Vegas, NV, USA, 2008, pp. 26–29.
[5] E.R. Bareiss, B.W. Porter, C.C. Wier, Protos: an examplar-based learning apprentice, in: Proceedings of the 4th International Workshop on Machine Learning, 1987, pp. 12–23.
[6] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Chapman and Hall, New York, 1984.
[7] N.V. Chawla, C45 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure, Workshop on Learning from Imbalanced Datasets, vol. II, ICML, Washington DC, 2003.
[8] P. Domingos, Unifying instance-based and rule-based induction, Machine Learning 24 (1996) 141–168.
[9] Y. Freund, R. Schapire, Experiments with a new boosting algorithm, in: Proceedings of the 13th International Conference on Machine Learning, 1996, pp. 148–156.
[10] J. Gehrke, R. Ramakrishnan, V. Ganti, Rainforest – a framework for fast decision tree construction of large datasets, Data Mining and Knowledge Discovery 4 (2000) 127–162.
[11] D.E. Goldgerg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, 1989.
[12] K.M. Gupta, D.W. Aha, P. Moore, Case-based collective inference for maritime object classification, in: Proceedings of the 8th International Conference on Case-Based Reasoning, Seattle, WA, USA, 2009, pp. 434–449.
[13] K. Haynes, X. Liu, W. Mio, Recognition using rapid classification tree, in: Proceedings of International Conference on Image Processing 2006, Atlanta, GA, USA, 2006, pp. 2753–2756.
[14] R.C. Holte, L.E. Acker, B.W. Porter, Concept learning and the problem of small disjuncts, in: International Joint Conferences on Artificial Intelligence, 1989, pp. 813–818.
[15] Y. Jiang, J. Lin, B. Cukic, T. Menzies., Variance analysis in software fault prediction models, in: Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering, Bangalore-Mysore, India, 2009, pp. 99–108.
[16] T.M. Khoshgoftaar, E.B. Allen, J. Deng, Using regression trees to classify fault-prone software modules, IEEE Transactions on Reliability 51 (2002) 455–462.
[17] T.M. Khoshgoftaar, B. Cukic, N. Seliya, Predicting fault-prone modules in embedded systems using analogy-based classification models, International Journal of Software Engineering and Knowledge Engineering 12 (2002) 201–221.
[18] T.M. Khoshgoftaar, Y. Liu, A multi-objective software quality classification model using genetic programming, IEEE Transactions on Reliability 56 (2007) 237–245.
[19] T.M. Khoshgoftaar, N. Seliya, Software quality estimation with case-based reasoning, Advances in Computers 62 (2004) 250–293.
[20] T.M. Khoshgoftaar, Y. Xiao, K. Gao, Multi-objective optimization by CBR GA-optimizer for module-order modeling, in: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), Banff, Alberta, Canada, 2004, pp. 220–225.
[21] T.M. Khoshgoftaar, Y. Xiao, K. Gao, Applying genetic algorithm based optimizer to software quality classification, in: Proceedings of the Twelfth ISSAT International Conference on Reliability and Quality in Design, Chicago, Illinois, USA, 2006, pp. 139–143.
[22] M. Lenz, B. Bartsch-Sporl, H. Burkhard, S. Wess (Eds.), Case-Based Reasoning Technology, from Fundations to Applications, Springer Company, 1998.
[23] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (2008) 485–496.
[24] T.-S. Lim, W.-Y. Loh, Y.-S. Shih, An empirical comparison of decision trees and other classification methods, Technique report 979, Department of Statistics, University of Wisconsin, Madison (1997).
[25] J.-J. Lin, Pattern recognition of fabric defects using case-based reasoning, Textile Research Journal 80 (2010) 794–802.
[26] M. Mehta, R. Agrawal and J. Rissanen, SLIQ: a fast scalable classifier for data mining, in: Proceedings of 5th International Conference on Extending Database Technology, 1996, pp. 18–32.
[27] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (2007) 2–13.
[28] R. Michalski, G. Tecuci (Eds.), Machine Learning: A Multistrategy Approach, Morgan Kaufman, 1994.
[29] R.S. Michalski (Ed.), Multistrategy Learning, Kluwer Academic Publishers, 1993.
[30] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman, San Mateo, CA, 1993.
[31] J.R. Quinlan, Improved use of continuous attributes in c4.5, Journal of Artificial Intelligence Research 4 (1996) 7790.
[32] R.E. Schapire, The strength of weak learnability, Machine Learning 5 (1990) 197–227.
[33] J.C. Shafer, R. Agrawal, M. Mehta, SPRINT: a scalable parallel classifier for data mining, in: Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India, 1996, pp. 544–555.
[34] A.L. Strehl, L. Li, E. Wiewiora, J. Langford, M.L. Littman, PAC model-free reinforcement learning, in: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 2006, pp. 881–888.

[35] D.E. van de Vlag, A. Stein, Incorporating uncertainty via hierarchical classification using fuzzy decision trees, IEEE Transactions on Geoscience and Remote Sensing 45 (2007) 237–245.
[36] C.A. Visaggio, F. de Rosa, A system for managing security knowledge using case based reasoning and misuse cases, Journal of Universal Computer Science 15 (2009) 3059–3078.
[37] D. Wettschereck, A hybrid nearest-neighbor and nearest-hyperrectangle algorithm, in: F. Bergadano, L.D. Raedt (Eds.), Proceedings of European Conference on Machine Learning, 1994, pp. 323–335.