Knowledge-Based Systems 91 (2016) 263–274

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Three-way decisions based software defect prediction

Weiwei Li [a,b,*], Zhiqiu Huang [a], Qing Li [c]

[a] College of Astronautics, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[b] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[c] College of Command Information System, PLA University of Science and Technology, Nanjing 210007, China

ARTICLE INFO

Article history:
Received 28 December 2014
Revised 24 September 2015
Accepted 29 September 2015
Available online 19 October 2015

Keywords:
Software defect classification
Software defect ranking
Three-way decisions

ABSTRACT

Based on a two-stage classification method and a two-stage ranking method on three-way decisions, this paper introduces a three-way decisions framework for cost-sensitive software defect prediction. For the classification problem in software defect prediction, traditional two-way decisions methods usually generate a higher classification error and more decision cost. Here, a two-stage classification method that integrates three-way decisions and ensemble learning to predict software defect is proposed. Experimental results on NASA data sets show that our method can obtain a higher accuracy and a lower decision cost. For the ranking problem in software defect prediction, a two-stage ranking method is introduced. In the first stage, all software modules are classified into three different regions based on three-way decisions. A dominance relation rough set based ranking algorithm is next applied to rank the modules in each region. Comparison experiments with 6 other ranking methods present that our proposed method can obtain a better result on FPA measure.

1. Introduction

Software defect prediction attempts to predict the defect proneness of new software modules with the historical defect data. It plays an important role in improving the quality of software systems [2,16,22]. The prediction technology is supposed to estimate the number of defects and to locate the position of each defect. Unfortunately, it is still hard to obtain the exact number or position of all defects based on current methods. Instead, since more information can be extracted from the historical defect data, two types of machine learning tasks are usually applied in software defect prediction: classification and ranking [65]. In a classification problem, new incoming software modules will be classified into defect-prone and not-defect-prone categories, with no differentiation between how bad the defect ones will be. In a ranking problem, new incoming modules will be sorted in descending order according to their numbers of predicted defects. The purpose of ranking is to identify the modules most likely to contain the largest numbers of defects and to allocate more testing resources for them.

In existing work for software defect classification, most researchers adopted many popular classifications methods directly, such as Naive Bayes [53], SVM [18], decision tree [39], and neural networks [31]. Furthermore, by considering the misclassification cost, some studies regarded the problem as a cost-sensitive learning task [50], and employed cost-sensitive learning methods including boosted neural network [80] and cost-boosting [34]. With or without cost-sensitive learning, all these studies assumed the underlying classification is binary and employed two-way decisions methods. A two-way decisions method is a kind of "immediately" decision method, which means it can provide the classification result directly and quickly. However, a quick result is usually accompanied by a high classification error. For example, assume the conditional probability of one module being a defect is 0.51, which is computed by the classification model, then its conditional probability of being a not-defect is 0.49. According to the majority principle in simple two-way decisions methods, this module will be classified into the defect-prone category directly. For these "vague" modules with conditional probabilities around 0.5, it has a high probability to classify them into wrong categories followed by more misclassification cost.

In existing work for software defect ranking, most researchers combined the defect-prone modules with the not-defect-prone modules together and ranked them based on the same attributes. A common method is applying regression models [8,9,13,24,33]. Regression methods implicitly assume that there exists a quantitative relation between the class label and the part of (or the whole of) attributes. However, in most applications, there exists a qualitative relation between the class label and the attributes only, and the quantitative hypothesis is too restrict to be satisfied.

* Corresponding author at: College of Astronautics, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China. Tel.: +8613584061768.
E-mail addresses: liweiwei@nuaa.edu.cn, jiaxiuyi@gmail.com (W. Li), zqhuang@nuaa.edu.cn (Z. Huang), lqhyt1994@126.com (Q. Li).

http://dx.doi.org/10.1016/j.knosys.2015.09.035
0950-7051/

In this paper we aim to solve the above problems. The three-way decisions theory is applied into the software defect prediction problem. The theory of three-way decisions is an extension of the common two-way decisions [69]. The essential ideal of three-way decisions is described in terms of a ternary classification according to evaluations of a set of criteria [71]. In widely used two-way decisions methods, an object will be accepted to classify into a particular category when it satisfies the criteria, and it will be rejected to classify into the particular category when it does not satisfy the criteria. By considering the uncertainty in many situations, three-way decisions methods classify objects based on the set of criteria with some degree, and use thresholds on the degrees of satisfiability to make one of three decisions: (a) accept an object as satisfying the set of criteria if its degree of satisfiability is above a certain level; (b) reject the object by treating it as not satisfying the criteria if its degree of satisfiability is below another level; and (c) neither accept nor reject the object but opt for a deferment. The primary advantage of applying three-way decisions over two-way decisions is that three-way decisions can choose some potentially misclassified objects for further-exam, which may lead to a lower classification error and less misclassification cost.

For software defect classification task, a two-stage classification method based on three-way decisions is proposed in this paper. In the first stage, a base classifier will be trained to compute the conditional probability of each module. After comparing to the thresholds, the modules with acceptance decision are classified into the defect-prone category, the modules with rejection decision are classified into the not-defect-prone category, and the left modules with deferment decisions wait for further-exam. In the second stage, for deferment modules, several classifiers will be combined as ensembles to provide a two-way decisions result. The experimental results on NASA data sets show that our proposed method can produce a higher classification accuracy and less misclassification cost.

For software defect ranking task, a two-stage ranking method is also proposed in this paper. In the first stage, all modules will be classified into three regions based on three-way decisions. In the following stage, a dominance relation rough set based algorithm will be introduced to rank the modules in each region, respectively. Several comparing experimental results validate the efficiency of our proposed method.

The rest of this paper is organized as follows. Some related work of software defect classification, software defect ranking, and three-way decisions are reviewed in Section 2. Section 3 presents the two-stage software defect classification method based on three-way decisions. Section 4 gives the two-stage software defect ranking method. All experiments are discussed in Section 5. Finally, we draw the conclusion in Section 6.

2. Related work

2.1. Software defect classification

An import work in software defect classification task is to construct defect-related feature space. For the defect-related features, many approaches have been proposed based on diverse information [16], such as code metrics [56], process metrics [23], previous defects [38] and so on [1,60].

Several individual studies reported that LOC data performs well in software defect prediction [77,83]. D'Ambros et al. [14] reported that change coupling correlates with defects. Nagappan and Ball [55] applied code churn together with dependency metrics to predict defect-prone modules. Khoshgoftaar and Seliya [35] considered 14 process metrics in their paper. Weyuker et al. [64] constructed a defect-count prediction model using a number of process measures in addition to structural measures. Besides existing studies which considered different features, some researches also focused on the feature selection on sets of metrics seems to improve the performance [7,32,50,62].

Another important work in software defect classification task is the construction or selection of the classification models or methods. Many statistical models and machine learning methods were applied in existing studies. Olague et al. [58] applied logistic regression to predict defects. Mizuno and Kikuno [54] reported that Orthogonal Sparse Bigrams Markov models were best suited to defect prediction based on their study. Bibi et al. [6] applied Regression via Classification method in software defect prediction. Khoshgoftaar et al. [37] built a classification tree using TREEDISC algorithm to predict whether a module was likely to have defects discovered by customers, or not, based on software product, process, and execution metrics. Classical machine learning algorithms were also widely used for software defect prediction such as Naive Bayes [53], Random Forests [52], and C4.5 [39].

Some comparison works among different methods were also studied by many researchers. Briand et al. [10] compared traditional regression techniques with multivariate adaptive regression splines. Khoshgoftaar and Seliya [36] compared 7 models in their work, and these models were evaluated against each other by comparing a measure of expected misclassification cost. Vandecruys et al. [63] compared ant colony optimization against C4.5, SVM, logistic regression, K-nearest neighbour, RIPPER and majority vote. Kanmani et al. [31] compared two variants of artificial neural networks against logistic regression and discriminant analysis. Guo et al. [20] compared 27 modeling techniques through the WEKA tool. Arisholm et al. [2] reported that their comprehensive performance comparison revealed no predictive differences between the 8 modeling techniques they investigated. Catal [12] reported that Random Forests provided the best prediction performance for large data sets and Naive Bayes was the best prediction algorithm for small data sets in terms of AUC evaluation parameter.

More literature reviews of existing studies on software defect classification techniques can be found in [2,11,12,22,65].

2.2. Software defect ranking

A scenario that is more useful in practice is to rank the classes by the predicted number of defects they will exhibit [16]. Given a predicted rank-order, one can focus on the top of the list due to the limitation of testing resources.

Many researches on ranking methods have been done. Based on their study at Ericsson, Ohlsson and Alberg [57] suggested that it was possible to predict the most defect-prone modules before coding has started. Khoshgoftaar and Allen [33] introduced a module-order model for defect prediction, and found that module-order models gave management more flexible reliability enhancement strategies than classification models. Ostrand et al. [5,59] predicted the expected number of defects by a negative binomial regression model, and then produced a predicted ranking of the units from most faulty to least faulty. Based on their ranking result, they could focus on the top $N\%$ of the files only [60]. Yang et al. [67] proposed a learning-to-rank algorithm for constructing defect prediction models.

Some researchers also considered the problem of how to measure the ranking performance. D'Ambros et al. [15] applied Spearman's correlation coefficient between the list of classes ranked by number of predicted defects and number of actual defects to measure the performance. In their further study [16], cumulative lift charts was employed as the evaluation measure. Weyuker et al. [65] defined a fault-percentile-average metric. Several comparison studies between different ranking methods on different performance measures were also reviewed in [65,66].

2.3. Three-way decisions

There are only two kinds of actions in a two-way decisions based software defect classification: accept the module as defect-prone one or reject the module as defect-prone one. In many applications, it is not suitable to make a direct two-way decisions because of the uncertainty or incompleteness of given information. To conquer this problem, the three-way decisions method is considered in this paper. Compared to the two-way decisions method including acceptance and rejection decisions only, an additional deferment decision is incorporated in the three-way decisions method. The reason of applying the deferment decision is that less information cannot support the two-way decisions result, and the advantage of incorporating the deferment decision is that it can decrease the misclassification cost.

The theory of three-way decisions, which describes the human cognitive process during decision making, have drawn a lot of attentions in recent years. By reviewing the related literature, we classify current studies on three-way decisions into two groups.

One group concentrates on the theoretical analysis or model construction of three-way decisions. Yao [69,70,73] analyzed the superiority of three-way decisions in probabilistic rough set models from a theoretical perspective. Li et al. [41] combined misclassification cost and test cost to determine the three-way decisions. Deng and Yao [17] studied three-way decisions based on the information theory. Azam and Yao [3] proposed a game-theoretic based three-way decisions framework. Liu et al. [49] reported a dynamic three-way decisions model and reviewed several function based three-way decisions. Liang and Liu [46] studied three-way decisions with interval-valued decision-theoretic rough sets. Hu [25] defined a three-way decisions space which can derive existing three-way decisions models. Li [40] and Yao [72] proposed that one could apply a sequential three-way decisions model to obtain a final two-way result. Jia et al. [26,29] addressed an optimization representation of decision-theoretic rough set models by considering the minimization of the three-way decisions cost. The attribute reduction in three-way decisions were also studied by several researchers [27,30,42,74,79]. Luo et al. [51] studied the dynamic maintenance of three-way decision rules. Several researchers studied the multiple class problem based on three-way decisions theoretic rough set model [47,48,81].

The other group concentrates on the application of three-way decisions. The essential ideas of three-way decisions are commonly used in many domains, such as hypothesis testing [69], decision making in environment management [19], information retrieval [45]. Yu et al. [75] analyzed the clustering problem based on three-way decisions. Jia et al. [28] and Zhou et al. [82] applied three-way decisions into email spam filtering. Li et al. [44] constructed a hierarchical framework for text classification based on three-way decisions. Yao and Azam [68] built a web-based medical decision support system for three-way medical decision making. Both Azam and Yao [4] and Zhang and Min [76] applied three-way decisions into recommender systems.

To the best of our knowledge, there is still no reported work on the three-way decisions based software defect prediction.

3. Two-stage software defect classification method

In this section, the basic notions about three-way decisions will be introduced followed by the two-stage software defect classification method.

3.1. Cost-sensitive learning based on three-way decisions

Typically, software defect prediction can be seen as a kind of cost-sensitive classification problem. Two kinds of misclassification costs exist in classical two-way decisions methods. One is the cost of misclassifying a not-defect-prone module into the defect-prone category.

Table 1
A classical cost function matrix for software defect prediction based on two-way decisions.

	Actual defect-prone	Actual not-defect-prone
Predicted defect-prone	C_{PP}	C_{PN}
Predicted not-defect-prone	C_{NP}	C_{NN}

Table 2
Two-way decisions method for software defect classification problem.

Notation	Explanation in software defect classification problem
x	Testing module;
$\Omega = \{\omega_1, \omega_2\}$	Two possible states for x: defect-prone or not-defect-prone;
$\mathcal{A} = \{a_1, a_2\}$	Two possible actions for x: classify x into defect-prone category or not-defect-prone category;
$p(\omega_j\|x)$	The conditional probability of x being in defect-prone state or not-defect-prone state;
$\lambda(a_i\|\omega_j)$	The cost for taking action a_i when the state of x is ω_j;

The other is the cost of labelling a defect-prone module as a not-defect-prone one. The former brings the waste of time and testing resource. The latter means we will lose the opportunity of correcting the defect module. Generally, the first kind of misclassification cost is less than the second one in real applications. Table 1 shows the classical cost function matrix for software defect prediction.

In Table 1, C_{PP} denotes the cost of classifying an actual defect module into the defect-prone category, and C_{NN} denotes the cost of classifying an actual not-defect module into the not-defect-prone category. Usually, it is assumed that the correct classification does not generate misclassification cost, which means $C_{PP} = C_{NN} = 0$. C_{PN} denotes the cost of classifying a not-defect module into the defect-prone category, and C_{NP} denotes the cost of classifying an actual defect module into the not-defect-prone category. In a classical cost-sensitive learning problem, C_{PN} is usually not equal to C_{NP}.

For the cost function matrix shown in Table 1, it can be described from the decision-theoretic perspective in the followings. Assume classification is a kind of action, and the actual class label of a module represents its state, then the cost function denotes the cost for taking the action with the current state. In Bayesian decision theory, let $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ be a finite set of m possible actions, and $\Omega = \{\omega_1, \ldots, \omega_s\}$ be a finite set of s states. Let $p(\omega_j|x)$ be the conditional probability of x being in state ω_j, and $\lambda(a_i|\omega_j)$ denote the cost for taking action a_i when the state is ω_j, then the expected cost associated with action a_i is given by:

$$\mathcal{R}(a_i|x) = \sum_{j=1}^{s} \lambda(a_i|\omega_j) \cdot p(\omega_j|x) \tag{1}$$

For the software defect classification problem, the detail of representing two-way decisions method is summarized in Table 2.

The theory of three-way decisions proposed by Yao [69] is constructed based on the notions of the acceptance, rejection and deferment, which can be directly generated by the three regions of probabilistic rough sets. The rules generated by the positive region are used to make a decision of acceptance, the rules generated by the negative region are used to make a decision of rejection, and the rules generated by the boundary region are used for making a decision of deferment. For the software defect classification problem, the detail of representing three-way decisions method is summarized in Table 3.

The cost function matrix of three-way decisions is shown in Table 4.

According to Eq. 1, we can derive the expected cost based on three-way decisions:

$$\begin{aligned}
\mathcal{R}(a_P|x) &= \lambda_{PP} \cdot p(X|x) + \lambda_{PN} \cdot p(X^c|x) \\
\mathcal{R}(a_B|x) &= \lambda_{BP} \cdot p(X|x) + \lambda_{BN} \cdot p(X^c|x) \\
\mathcal{R}(a_N|x) &= \lambda_{NP} \cdot p(X|x) + \lambda_{NN} \cdot p(X^c|x)
\end{aligned} \tag{2}$$

Table 3
Three-way decisions method for software defect classification problem.

Notation	Explanation in software defect classification problem
x	Testing module;
$\Omega = \{X, X^c\}$	X: x is a defect-prone module, X^c: x is a not-defect-prone module;
$\mathcal{A} = \{a_P, a_N, a_B\}$	a_P: accept x as defect-prone module $x \in \mathrm{POS}(X)$, a_N: reject x as defect-prone module $x \in \mathrm{NEG}(X)$, a_B: x needs further-exam $x \in \mathrm{BND}(X)$;
$p(X\|x)$	The conditional probability of x being in X;
$p(X^c\|x)$	The conditional probability of x being in X^c, $p(X\|x) + p(X^c\|x) = 1$;
$\lambda_{*P} = \lambda(a_*\|X)$	$* = P, N$ or B, the costs for taking action a_P, a_N or a_B when the state of x is X;
$\lambda_{*N} = \lambda(a_*\|X^c)$	$* = P, N$ or B, the costs for taking action a_P, a_N or a_B when the state of x is X^c;

Table 4
The cost function matrix for software defect prediction based on three-way decisions.

	X (actual defect-prone)	X^c (actual not-defect-prone)
a_P (predicted defect-prone)	λ_{PP}	λ_{PN}
a_B (needs further-exam)	λ_{BP}	λ_{BN}
a_N (predicted not-defect-prone)	λ_{NP}	λ_{NN}

Based on Bayesian minimizing decision cost procedure, the following rules of three-way decisions can be derived:

(P) If $\mathcal{R}(a_P|x) \le \mathcal{R}(a_B|x)$ and $\mathcal{R}(a_P|x) \le \mathcal{R}(a_N|x)$, then $x \in \mathrm{POS}(X)$;
(B) If $\mathcal{R}(a_B|x) \le \mathcal{R}(a_P|x)$ and $\mathcal{R}(a_B|x) \le \mathcal{R}(a_N|x)$, then $x \in \mathrm{BND}(X)$;
(N) If $\mathcal{R}(a_N|x) \le \mathcal{R}(a_P|x)$ and $\mathcal{R}(a_N|x) \le \mathcal{R}(a_B|x)$, then $x \in \mathrm{NEG}(X)$;

Consider a special kind of cost functions with $\lambda_{PP} \le \lambda_{BP} < \lambda_{NP}$ and $\lambda_{NN} \le \lambda_{BN} < \lambda_{PN}$, that is, the cost of classifying an object x belonging to X into the positive region $\mathrm{POS}(X)$ is less than or equal to the cost of classifying x into the boundary region $\mathrm{BND}(X)$, and both of these cost are strictly less than the cost of classifying x into the negative region $\mathrm{NEG}(X)$. The reverse order of cost is used for classifying an object not in X. Since $p(X|x) + p(X^c|x) = 1$, we have the following decision rules:

(P) If $p(X|x) \ge \alpha$ and $p(X|x) \ge \gamma$, decide $x \in \mathrm{POS}(X)$;
(B) If $p(X|x) \le \alpha$ and $p(X|x) \ge \beta$, decide $x \in \mathrm{BND}(X)$;
(N) If $p(X|x) \le \beta$ and $p(X|x) \le \gamma$, decide $x \in \mathrm{NEG}(X)$;

where the parameters α, β, and γ are defined as:

$$\alpha = \frac{(\lambda_{PN} - \lambda_{BN})}{(\lambda_{PN} - \lambda_{BN}) + (\lambda_{BP} - \lambda_{PP})},$$
$$\beta = \frac{(\lambda_{BN} - \lambda_{NN})}{(\lambda_{BN} - \lambda_{NN}) + (\lambda_{NP} - \lambda_{BP})},$$
$$\gamma = \frac{(\lambda_{PN} - \lambda_{NN})}{(\lambda_{PN} - \lambda_{NN}) + (\lambda_{NP} - \lambda_{PP})}. \quad (3)$$

Each rule is defined by two out of the three parameters. If the following condition on the cost functions [69] is applied:

$$\frac{(\lambda_{NP} - \lambda_{BP})}{(\lambda_{BN} - \lambda_{NN})} > \frac{(\lambda_{BP} - \lambda_{PP})}{(\lambda_{PN} - \lambda_{BN})}, \quad (4)$$

we have $0 \le \beta < \gamma < \alpha \le 1$. In this case, after tie-breaking, the following simplified rules are obtained:

(P) If $p(X|x) \ge \alpha$, decide $x \in \mathrm{POS}(X)$;
(B) If $\beta < p(X|x) < \alpha$, decide $x \in \mathrm{BND(X)}$;
(N) If $p(X|x) \le \beta$, decide $x \in \mathrm{NEG}(X)$.

3.2. Three-way decisions based two-stage classification approach to cost-sensitive software defect prediction

Based on three-way decisions, a two-stage software defect classification method is proposed. The flow chart of the method is shown in Fig. 1.

In the first stage, all test modules will be classified into three different regions on the basis of the comparison result between their conditional probabilities and the pair of decision thresholds. The modules in positive region are labeled as defect-prone ones, the modules in negative region are labeled as not-defect-prone ones, and the modules in boundary region are deferred for further-exam. In the second stage, for those deferment modules, an ensemble learning algorithm is proposed to obtain the final two-way result [43].

As the purpose of any decisions is to provide a clear result for users, and the reason of incorporating deferment decision is that the given information is uncertain or incomplete, we can convert the three-way decisions result to a two-way decisions result by adding more information. There are two kinds of methods to add information: one is adding data and the other is adding "experts".

For the method of adding data, we can further classify it into two groups: the first group is adding features or attributes, and the second group is adding training samples. Both two groups of adding data can change the conditional probabilities of some modules. For example, in the software defect classification task, assume there are two attributes *number_operators* and *number_code_lines* only, then a lot of modules will be classified into the boundary region. To conquer this problem, other attributes such as *cyclomatic_complexity* and *design_complexity* can be added to describe the modules. More modules will be classified into the positive or negative regions as their conditional probabilities may increase or decrease. The method of adding attributes is called sequential decisions [40,72]. Besides adding attributes, adding training samples can also change the conditional probabilities of some modules. For those modules with their conditional probabilities between β and α, their updated conditional probabilities may be greater than α or less than β, then they will be classified into the positive or negative regions.

The second method of adding information is adding "experts". If an expert cannot decide whether the module is a defect-prone one or not, we can ask more experts to decide it. In the software defect classification task, the classifier can be seen as the expert, then the uncertainty decisions problem can be solved by adding classifiers. Ensemble learning will be an intuitive and reasonable approach because the generalization ability of an ensemble is usually much stronger than that of a single classifier. Several classifiers will be combined to vote for the final class label. In the voting stage, the most frequent class label appeared will be the final result. If there exists a tie situation, for example, each class label appeared once, select one randomly as the final result. The detail of the two-stage classification algorithm for software defect prediction is presented in Algorithm 1 .

3.3. Discussion on the efficiency of proposed method

As we have summarized in related work, many machine learning methods were directly applied in software defect classification task. All these methods can be seen as a kind of two-way decisions based method. A module has to be classified into the defect-prone category or the not-defect-prone category by the two-way decisions based classifier. Then the accuracy depends on the classifier's performance mostly. For those modules with both defect-related and not-defect-related characteristics, which are not easy to be classified correctly. The two-way decisions based classification method may bring more misclassification errors and more costs.

Our proposed method is based on three-way decisions. The main advantage of three-way decisions solution is that it allows the possibility of refusing to make a direct decision, which means it can

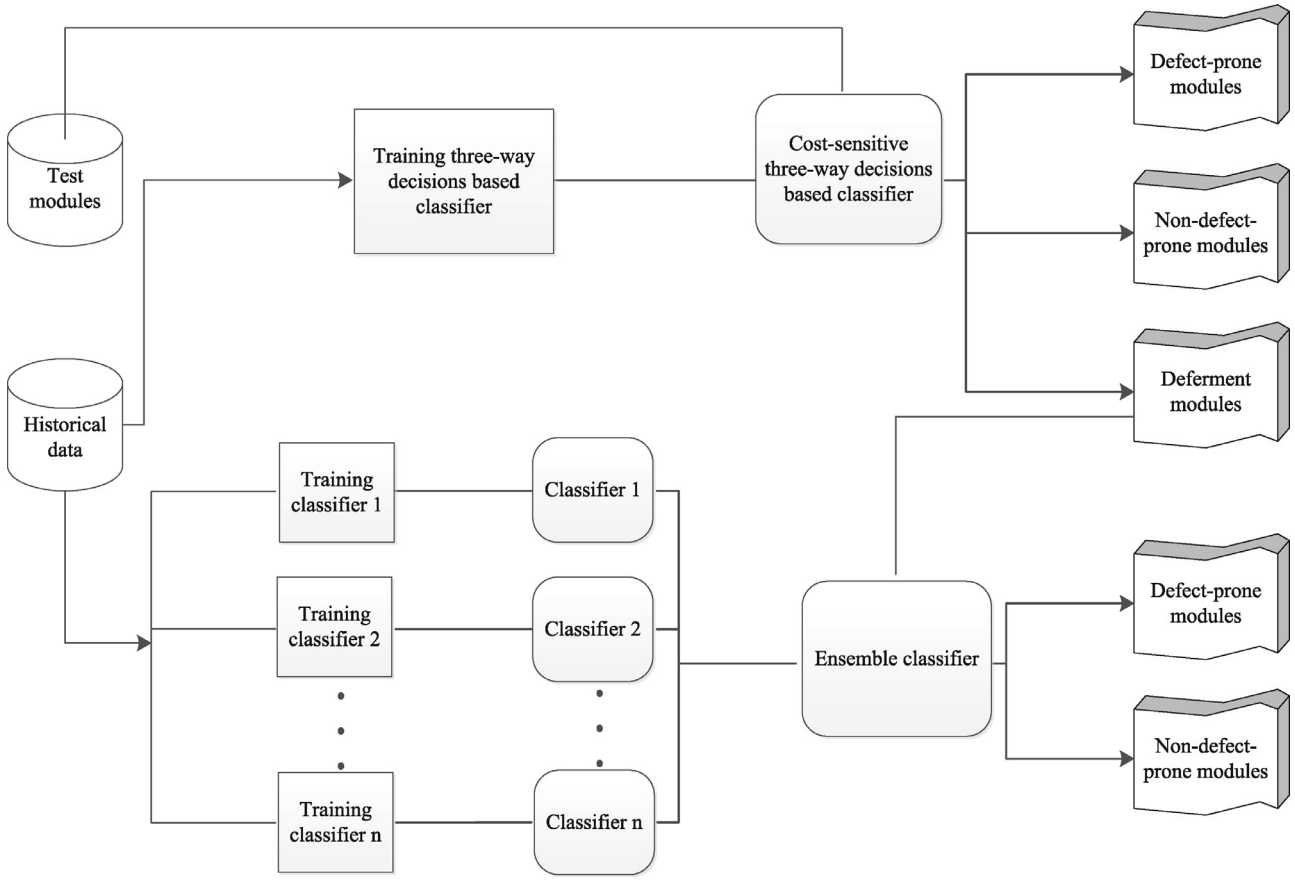


Fig. 1. The flow chart of the three-way decisions based two-stage classification approach to cost-sensitive software defect prediction.

Algorithm 1 Three-way decisions based two-stage classification algorithm to cost-sensitive software defect prediction.

Input:
Training samples $\mathcal{D}$;
A cost matrix λ_{ij};
Multiple classifiers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_T$;
A test module x.

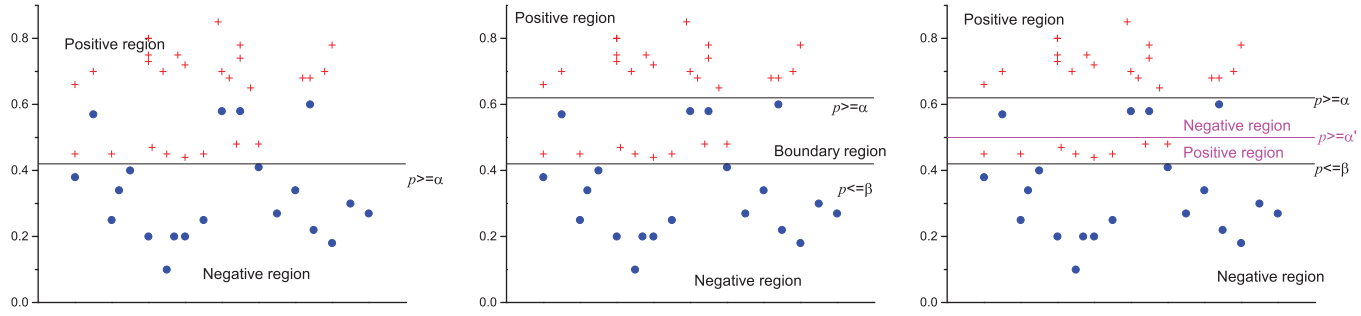
Output:
Predicted class label $H(x) \in \{X, X^c\}$, X: defect-prone, X^c: not-defect-prone.

Compute α, β from λ_{ij};
for each $t \in [1, T]$ **do**
  $h_t = \mathcal{C}_t(\mathcal{D})$; % Train a learner h_t by applying $\mathcal{C}_t$ to training samples $\mathcal{D}$
end for
$p_b(X|x) = p(h_b(x))$; % Use h_b as the base learner to get the probability of being defect-prone
if $p_b(X|x) > \alpha$ **then**
  $H(x) = X$; %predicted as a defect-prone one
else if $p_b(X|x) < \beta$ **then**
  $H(x) = X^c$; %predicted as a not-defect-prone one
else
  for each $t \in [1, T]$ **do**
    $L_t = h_t(x)$; % Using each classifier to compute the class label L_t
  end for
  $H(x) = vote\{L_1, L_2, \ldots, L_T\}$; % Select the most frequently class label, pick one randomly if tie situation exists
end if
return $H(x)$;

convert some potential misclassifications into rejections [28], and these modules will be further-examined. Furthermore, to deal with the deferment modules and obtain a two-way result, since we do not have any additional information about the attributes or samples, we adopt ensemble learning method in the second stage. As the generalization ability of an ensemble is usually much stronger than that of a single classifier, compared to classical two-way decisions based methods with single classifier, our proposed three-way decisions based two-stage classification approach can get a lower error rate and a lower misclassification cost.

Fig. 2 is an example to explain the superiority of three-way decisions based two-stage classification approach. Fig. 2(a) shows the classification result of classical two-way decisions methods. Assume a classifier Cf_1 can provide the conditional probability of each module, denoted as p. For the sake of simplicity, the classification mechanism is also assumed that modules with their conditional probabilities are greater than or equal to a threshold α will be classified into the positive region, other modules with $p < \alpha$ are classified into the negative region. From Fig. 2(a) we can see that there exist some modules which are misclassified into the wrong region and this kind of misclassification could not be avoided by the classifier. The coverage of a two-way decision method is usually equal to 1 as it make immediate decisions.

Fig. 2 (b) shows the classification result of classical three-way decisions methods. Assume a classifier Cf_2 has following classification mechanism: if $p \geq \alpha$, modules will be classified into the positive region; if $p \leq \beta$, modules will be classified into the negative region; otherwise, modules will be classified into the boundary region. Compared to two-way decisions methods, the advantage of three-way decisions methods is that three-way decisions methods usually have a higher accuracy because modules in the boundary region are not



(a) Two-way decisions method (b) Three-way decisions method (c) Three-way decisions based two-stage method

Fig. 2. A simple example to show the superiority of three-way decisions based two-stage method.

considered in its calculation formula (see Eq. 10). However, the coverage of a three-way decision method is usually less than 1.

Fig. 2 (c) shows the classification result of a three-way decisions based two-stage method. In the first stage, a three-way decision based classifier Cf_2 is applied to get three regions. In the second stage, another classifier (or classifiers) Cf_i is (are) adopted to provide two-way decisions based classification for the boundary region (simply denoted as $p \geq \alpha'$). Compared to two-way decisions methods, the two-stage method has a higher accuracy with a same coverage value.

4. Two-stage software defect ranking method

4.1. Three-way decisions based two-stage ranking approach to cost-sensitive software defect prediction

In this section, a two-stage software defect ranking method will be introduced. In the first stage, a three-way decisions based classifier is applied to classify test modules into the positive region, boundary region and negative region, respectively. For these three different regions, an intuitive situation is assumed that the number of defects for each module in the positive region is greater than the number of defects for each module in the boundary region, and the number of defects for each module in the boundary region is greater than the number of defects for each module in the negative region. In the second stage, a dominance relation rough set based ranking method is applied for each region and the results from three different regions are ranked at last. For example, assume following results are obtained in the second stage: $x_1 \succ x_2$ in the positive region, $x_4 \succ x_5$ in the boundary region, and $x_3 \succ x_6$ in the negative region, then the final ranking list is $x_1 \succ x_2 \succ x_4 \succ x_5 \succ x_3 \succ x_6$.

For the first stage, as we have described a lot in Section 3.1, we will not explain more in this part. For the second stage, the dominance relation rough set based ranking method is introduced as follows [61].

Definition 1. (U, C, F) is a continuous value information system, where $U = \{x_i, x_2, \ldots, x_n\}$ is a finite nonempty set of objects, $C = \{c_1, c_2, \ldots, c_m\}$ is a set of attributes describing the objects, $F = \{f_l : U \to V_l (l \leq m)\}$ is the set of functions mapping from objects to attributes, and V_l is the domain of a_l.

For each set of attributes $B \subseteq C$, $R_B^{\leq}$ is the dominance relation in the given information system if

$$R_B^{\leq} = \{(x_i, x_j) \in U^2 \mid f_l(x_i) \leq f_l(x_j), \forall a_l \in B\}. \quad (5)$$

$(x_i, x_j) \in R_B^{\leq}$ denotes x_j dominates x_i with respect to B. The dominating class of x_i is defined as:

$$[x_i]_B^{\leq} = \{x_j \mid (x_i, x_j) \in R_B^{\leq}\}. \quad (6)$$

$[x_i]_B^{\leq}$ is the set of objects dominating x_i with respect to B.

Table 5
An information system.

U	c_1	c_2	c_3
x_1	1	2	1
x_2	3	2	2
x_3	1	1	2
x_4	2	1	3
x_5	3	3	2
x_6	3	2	3

For dominating classes $[x_i]_B^{\leq}$ and $[x_j]_B^{\leq}$, the inclusion degree of x_i dominating x_j is defined as:

$$R_B(x_i, x_j) = \frac{|\sim[x_i]_B^{\leq} \cup [x_j]_B^{\leq}|}{n}, \quad (7)$$

where $|\cdot|$ denotes the cardinality of the given set, and $\sim X = U - X$.

Then the inclusion degree of x_i with respect to B can be computed by:

$$R_B(x_i) = \frac{1}{n-1} \sum_{j \neq i} R_B(x_i, x_j). \quad (8)$$

If $R_B(x_i) > R_B(x_j) > R_B(x_k)$, the ranking list is $x_i \succ x_j \succ x_k$.

The computation procedure of the ranking method can be shown by the following example [78].

Example 1. Table 5 is an information system (U, C, F), where $U = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, $C = \{c_1, c_2, c_3\}$, $V_l = \{1, 2, 3\}$.

The dominating classes of all objects can be obtained as follows:

$$[x_1]_B^{\leq} = \{x_1, x_2, x_5, x_6\}, \quad [x_2]_B^{\leq} = \{x_2, x_5, x_6\},$$
$$[x_3]_B^{\leq} = \{x_2, x_3, x_4, x_5, x_6\}, \quad [x_4]_B^{\leq} = \{x_4, x_6\},$$
$$[x_5]_B^{\leq} = \{x_5\}, \quad [x_6]_B^{\leq} = \{x_6\}.$$

Based on Eq. 7, the following matrix of inclusion degrees can be derived. The element x_{ij} in the matrix represents the inclusion degree of x_i dominating x_j.

$$\mathbf{R}_B = [x_{ij}] = \begin{pmatrix} 1 & 0.83 & 0.83 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 0.67 & 0.67 & 0.67 \\ 0.67 & 0.67 & 1 & 0.5 & 0.33 & 0.33 \\ 0.83 & 0.83 & 1 & 1 & 0.67 & 0.83 \\ 1 & 1 & 1 & 0.83 & 1 & 0.83 \\ 1 & 1 & 1 & 1 & 0.83 & 1 \end{pmatrix}$$

Then the inclusion degree of each x_i can be obtained:

$$R_B(x_1) \approx 0.63, \quad R_B(x_2) = 0.8, \quad R_B(x_3) = 0.5,$$
$$R_B(x_4) \approx 0.83, \quad R_B(x_5) \approx 0.93, \quad R_B(x_6) \approx 0.97.$$

The final ranking result is $x_6 \succ x_5 \succ x_4 \succ x_2 \succ x_1 \succ x_3$.

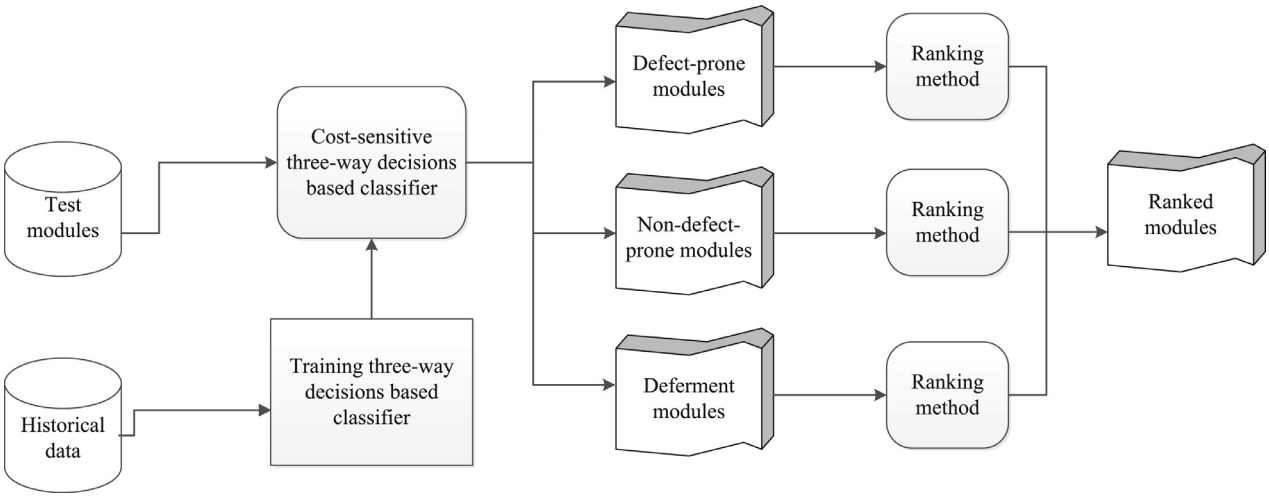


Fig. 3. The flow chart of the three-way decisions based two-stage ranking approach to cost-sensitive software defect prediction.

The flow chart of the three-way decisions based two-stage ranking approach to cost-sensitive software defect prediction is shown in Fig. 3.

The detail of the two-stage ranking algorithm for software defect prediction is presented in Algorithm 2 .

Algorithm 2 Three-way decisions based two-stage ranking algorithm to cost-sensitive software defect prediction.

Input:
Training samples $\mathcal{D}$;
A cost matrix λ_{ij};
A classifiers $\mathcal{C}$;
The dominance relation rough set based ranking method $\mathcal{DR}$;
Test modules $O = \{o_1, o_2, \ldots, o_n\}$.

Output:
Ranking list of O.

Compute α, β from λ_{ij};
$h = \mathcal{C}(\mathcal{D})$; % Train a learner h by applying $\mathcal{C}$ to training samples $\mathcal{D}$
for each $t \in [1, n]$ **do**
  $p(o_t) = p(h(o_t))$; % Use h as the learner to get the probability of being defect-prone
  if $p(o_t) > \alpha$ **then**
    o_t is added into the positive region *POS*;
  else if $p(o_t) < \beta$ **then**
    o_t is added into the negative region *NEG*;
  else
    o_t is added into the boundary region *BND*;
  end if
end for
rank the modules in *POS* by $\mathcal{DR}$: R_P;
rank the modules in *BND* by $\mathcal{DR}$: R_B;
rank the modules in *NEG* by $\mathcal{DR}$: R_N;
combine R_P, R_B and R_N into the final result R_O with respect to $R_P \succ R_B \succ R_N$;
return R_O;

4.2. Discussion on the efficiency of proposed method

Most existing ranking methods for software defect ranking problem are regression methods, which are used to predict the "exact" number of defects. These regression methods may ignore the order information among different modules and could not obtain a good ranking result. Besides, some non-regression methods were also discussed [66]. In their work, the defect-prone modules and the not-defect-prone modules are treated equally. For defect-prone modules and not-defect-prone modules, they may have different characteristics, thus it is not suitable to rank them together under the same evaluating measure. In our proposed method, these two kinds of different modules are classified into three different regions and ranked separately.

We can explain the superiority of our two-stage based method in two ways. Firstly, our method has a good tolerance for noise affection. In existing studies, the defect-prone modules and the not-defect-prone modules are ranked together. In this case, the not-defect-prone modules may bring negative influence on the ranking of defect-prone modules. For example, assume x_1 and x_2 are two actually defect-prone modules with $x_2 \succ x_1$, x_3 and x_4 are two actually not-defect-prone modules with $x_3 \succ x_4$. Based on a particular ranking method, we may have following result: $x_1 \succ x_3, x_1 \succ x_4, x_3 \succ x_2, x_4 \succ x_2, x_3 \succ x_4, x_2 \succ x_1$. In order to eliminate the contradiction, affected by the not-defect-prone modules x_3 and x_4, the final ranking result may be obtained as $x_1 \succ x_3 \succ x_4 \succ x_2$. However, in our proposed method, all modules are classified into different regions in the first stage, and the modules in negative region cannot affect the modules in positive region. In other words, those predicted not-defect-prone modules will not induce the contradiction of ranking order among predicted defect-prone modules.

Secondly, our method has a good convenience. Since an important reason of ranking defects is to select top N% defects, if the size of the positive region is greater than N in our method, we can focus on the ranking of modules in positive region only and ignore other two regions.

5. Experiments

In this section, several comparison experiments on software defect classification and ranking problems are implemented.

5.1. Comparison experiments on software defect classification

5.1.1. Data sets

There are 11 NASA data sets used in our experiments. All these data sets which contain satellite data, simulation data, and ground station data, etc., can be achieved from the public PROMISE Repository[1]. Several attributes are applied to describe software modules,

[1] http://promisedata.googlecode.com

Table 6
Brief description of the data sets(I: # instances, A: # attributes, P: percentage of defects).

Data sets	Description	Language	I	A	P(%)
cm1	Spacecraft instrument	C	498	22	9.83
jm1	Real-time predictive ground system	C	7782	22	21.4
kc2	Storage management for ground data	C++	522	21	20.49
kc3	Storage management for ground data	JAVA	458	39	9.38
mc2	Video guidance system	C++	61	39	32.29
mw1	A zero gravity experiment related to combustion	C++	403	37	7.69
pc1	Flight software for earth orbiting satellite	C++	1109	21	6.94
pc2	Flight software for earth orbiting satellite	C++	745	37	2.147
pc3	Flight software for earth orbiting satellite	C++	1077	38	12.4
pc4	Flight software for earth orbiting satellite	C++	1458	38	12.2
pc5	Flight software for earth orbiting satellite	C++	17186	39	3.0

such as McCabe measure, Halstead measure, number of operators, number of code lines, etc. Information of data sets are summarized in Table 6.

The percentage of defects in each data set shows that software defect classification problem is a kind of imbalance learning problem.

5.1.2. Experimental setting

Three groups of comparison experiments are implemented in this part. To simplify the denotation, we use TS3WD to represent the three-way decisions based two-stage method, 3WD to represent the classical three-way decisions based method, 2WD to represent the classical two-way decisions method, and ELM to represent an ensemble learning method.

The first group of experiment, named 3WD VS. 2WD, compares the results of the three-way decisions based method and the two-way decisions based method. Naive Bayes (NB) is adopted as the base classifier in both 3WD and 2WD because it can provide the conditional probability directly. The conditional probability of module x being a defect-prone one can be computed as:

$$p(X|x) = \frac{p(x|X) \cdot p(X)}{p(x|X) \cdot p(X) + p(x|X^c) \cdot p(X^c)}. \tag{9}$$

The second group of experiment, named TS3WD VS. 2WD, compares the efficiency of TS3WD and 2WD. For 2WD and the first stage of TS3WD, same experimental setting as shown in group 3WD VS. 2WD is applied. For the second stage of TS3WD, as the final label of x is voted by all ensemble classifiers, it is enough to obtain the two-way decisions result rather than the exact value of $p(X|x)$ for each classifier. There are 5 different classifiers in the ensemble learning procedure: NB (Naive Bayes), C4.5 (decision tree), KNN (K nearest neighbor), SVM (support vector machine), and RBF (neural network). All these classifiers are implemented based on WEKA [21] with version 3.5, with all parameters set to their default values.

The third group of experiment, named TS3WD VS. EML, compares the efficiency of TS3WD and classical ensemble learning method. For EML, all above mentioned 5 classifiers are implemented as two-way decisions based classifiers to vote for the final label of each x. For TS3WD, all classifiers are tested as the base classifier each time. Since C4.5, KNN, SVM and RBF do not provide conditional probability of each module directly, their sigmoid functions are used to compute the conditional probability, while the default sigmoid functions are provided by WEKA.

In each series of experiments, ten times 10-fold cross validation are performed. Concretely, 10-fold cross validation is repeated for ten times with randomly generated cost matrices, and the average results are recorded.

5.1.3. Measures

The measures of comparison performance include accuracy, F-measure and misclassification cost. The accuracy is defined as following:

$$accuracy = \frac{N_{PP} + N_{NN}}{N_{PP} + N_{NN} + N_{PN} + N_{NP}}. \tag{10}$$

N_{PP} denotes the number of actual defect modules being classified into the defect-prone category. N_{NN} denotes the number of actual not-defect modules being classified into the not-defect-prone category. N_{PN} denotes the number of actual not-defect modules being classified into the defect-prone category. N_{NP} denotes the number of actual defect modules being classified into the not-defect-prone category.

The coverage value is defined as following:

$$coverage = \frac{N_{PP} + N_{NN} + N_{PN} + N_{NP}}{N_{PP} + N_{NN} + N_{PN} + N_{NP} + N_{BP} + N_{BN}}, \tag{11}$$

where N_{BP} denotes the number of actual defect modules being classified into the boundary region and N_{BN} denotes the number of actual not-defect modules being classified into the boundary region. For two-way decisions methods, we have $coverage = 1$ because it does not generate boundary region, $N_{BP} = N_{BN} = 0$. For three-way decisions methods, the value of *coverage* is usually less than 1.

Based on the accuracy and coverage, F-measure is defined as following:

$$F = \frac{2 \cdot accuracy \cdot coverage}{accuracy + coverage}. \tag{12}$$

The decision cost can be defined by considering the cost matrix shown in Table 4.

$$cost = \lambda_{PN} \cdot N_{PN} + \lambda_{NP} \cdot N_{NP} + \lambda_{BN} \cdot N_{BN} + \lambda_{BP} \cdot N_{BP}. \tag{13}$$

The definition tells us that three-way decisions methods must consider the cost generated from deferment rules.

5.1.4. Results and analysis

Table 7 presents the performances of the three-way decisions based Naive Bayes and the two-way decisions based Naive Bayes on accuracy, F-measure and decision cost, respectively. The best performance of each comparison is boldfaced.

For the experiments of 3WD VS. 2WD, we have the following conclusion and analysis:

- For the accuracy measure, the three-way decisions approach can obtain a higher value than the two-way decisions approach. The reason can be two-fold: On one hand, for those modules whose conditional probabilities are between β and α, which could be misclassified with a high probability, the three-way decisions approach classified them into the boundary region waiting for further examination while the two-way decisions approach assigned them exact labels immediately. On the other hand, from the Eq. 10 we can see that although the value of numerator $(N_{PP} + N_{NN})$ is reduced in the three-way decisions approach, the value of denominator (can be represented by $|U| - N_{BP} - N_{BN}$, while $|U|$ denotes

Table 7
The comparison results of the three-way decisions based Naive Bayes (3WD) and the two-way decisions based Naive Bayes (2WD) on aerospace software data sets. The best performance is boldfaced.

	Accuracy		*F*-measure		Decision cost	
Data sets	3WD	2WD	3WD	2WD	3WD	2WD
cm1	**0.8253**	0.8229	0.9004	**0.9029**	**2.3644**	2.4638
jm1	**0.7835**	0.7819	0.8769	**0.8776**	**52.6842**	53.8592
kc2	**0.8376**	0.8352	0.9092	**0.9102**	**3.7804**	3.8632
kc3	**0.7894**	0.7892	0.8754	**0.8822**	**1.4919**	1.5759
mc2	0.7162	**0.7192**	0.8321	**0.8367**	**1.1220**	1.1383
mw1	**0.8225**	0.8135	0.8966	**0.8972**	**1.9763**	2.1394
pc1	**0.8870**	0.8808	**0.9373**	0.9366	**3.1114**	3.3816
pc2	**0.9194**	0.9072	**0.9533**	0.9514	**2.6973**	3.2319
pc3	**0.4019**	0.3565	**0.5482**	0.5256	**28.8311**	35.7230
pc4	**0.8748**	0.8704	0.9194	**0.9307**	**6.8538**	7.8289
pc5	**0.9647**	0.9642	0.9816	**0.9818**	**23.2620**	23.6651
average	**0.8020**	0.7946	0.8755	**0.8757**	**11.6523**	12.6246

Table 8
The comparison results of the three-way decisions based two-stage method (TS3WD) and the classical two-way decisions method (2WD) on aerospace software data sets. The best performance is boldfaced.

	Accuracy		*F*-measure		Decision cost	
data sets	TS3WD	2WD	TS3WD	2WD	TS3WD	2WD
cm1	**0.8223**	0.8210	0.9015	**0.9017**	**2.4859**	2.5110
jm1	**0.7823**	**0.7823**	**0.8778**	**0.8778**	**71.5940**	71.7599
kc2	**0.8351**	0.8349	0.9097	**0.9100**	3.4461	**3.4438**
kc3	**0.7864**	0.7822	**0.8781**	0.8778	**1.6916**	1.7451
mc2	**0.7174**	0.7160	0.8313	**0.8345**	**1.4952**	1.5047
mw1	**0.8190**	0.8128	**0.8986**	0.8967	**1.7972**	1.8864
pc1	**0.8851**	0.8820	**0.9387**	0.9373	**3.2662**	3.4001
pc2	**0.9180**	0.9120	**0.9563**	0.9540	**2.3588**	2.6003
pc3	**0.4253**	0.3543	**0.5845**	0.5232	**22.8784**	27.0534
pc4	**0.8765**	0.8717	**0.9340**	0.9315	**7.4281**	7.6010
pc5	**0.9647**	0.9643	**0.9820**	0.9818	**25.0446**	25.1999
average	**0.8029**	0.7939	**0.8811**	0.8751	**13.0442**	13.5187

the number of all samples) is also reduced. Usually, the denominator reduced more than the numerator, which means the three-way decisions approach can have a higher accuracy.

- In the three-way decisions approach, $(N_{BP} + N_{BN}) \geq 0$ owing to the incorporating of deferment decision, thus the value of *coverage* is usually less than 1 followed by reducing the value of *F*-measure. In most situations shown in Table 7, two-way decisions can obtain a better result than three-way decisions on *F*-measure.
- For decision cost, although the three-way decisions approach has additional decision cost generated by deferment decision, it still superior to corresponding two-way decisions approach on decision cost because three-way decisions is constructed on the basis of minimizing Bayesian decision cost principle, and the experimental result also indicates the conclusion.
- The 10 groups of cost functions used in our experiments are randomly generated. Under these different groups of cost functions, the three-way decisions approach can get a consistent performance on all three measures, which means the superiority of three-way decisions is independent on particular cost functions. However, the number of modules being classified into the boundary region is decided by the pair of thresholds, users can achieve a proper or even the best performance on particular measure by adjusting the pair of thresholds.

Table 8 presents the comparison result of TS3WD VS. 2WD. We can obtain the following result: TS3WD is absolutely superior to 2WD on both accuracy and decision cost measures, and TS3WD is also better than 2WD on *F*-measure on most data sets.

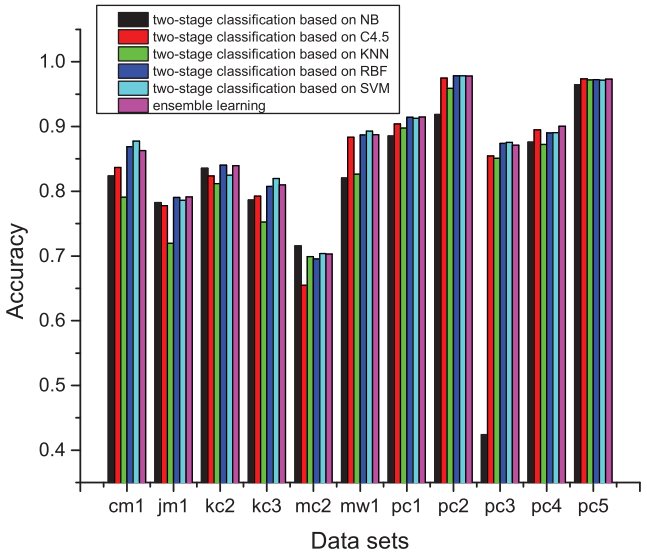


Fig. 4. The comparison results of the three-way decisions based two-stage method (TS3WD) and the ensemble learning method (EML) on accuracy.

For the experiments of TS3WD VS. 2WD, we have the following observation and analysis:

- The purpose of TS3WD is to deal with the deferment modules and convert the three-way decisions result to a two-way decisions result. TS3WD has the same value 1 on *coverage* with 2WD, since $(N_{BP} + N_{BN}) = 0$ through ensemble learning, then the value of *F*-measure will be improved with respect to the increasing of accuracy value. Note that all values appeared in Table 8 are the average results based on 10 groups of cost functions, then each value of accuracy does not correspond to the value of *F*-measure.
- Compared to 2WD, TS3WD improved 1.1% on accuracy and 0.7% on *F*-measure only, which is not a "so great" result. But on decision cost, the improvement is 3.5%, which means TS3WD is an efficient method indeed for the cost-sensitive aerospace software data.

In TS3WD, the interval size of the pair of thresholds determines the number of deferment modules, moreover, it determines the effect of ensemble learning in the whole method. If the interval size of the pair of thresholds is 0, which means there is no deferment module, the ensemble learning in second stage will be ignored and TS3WD will be converted to 2WD totally. If the thresholds are set as $\beta < 0$ and $\alpha > 1$, the first stage in TS3WD will be ignored and TS3WD will be converted to an ensemble learning method (EML) totally. Fig. 4 shows the detailed comparison results of TS3WD VS. EML on accuracy measure.

From Fig. 4 we can conclude the following observation and analysis:

- Generally speaking, ensemble learning methods have a better generalization ability rather than a single classifier. However, it cannot declare that EML have a better classification accuracy rather than other TS3WD methods. In this experiment, TS3WD based on SVM can obtain the best classification result on 5 data sets, while EML obtains the best result on 3 data sets. TS3WD based on RBF can obtain the best result on 2 data sets, and both TS3WD based on NB and TS3WD based on C4.5 obtain the best result on 1 data set. As TS3WD contains two stages of classification, the efficiency of TS3WD will be influenced by the results of two stages. From the result shown in Fig. 4, we may conclude that the efficiency of TS3WD is most influenced by the efficiency of its base classifier. TS3WD with a better base classifier can get a better classification accuracy rather than EML.

Table 9
Brief description of the data sets (I: # instances, A: # attributes, P: percentage of defects).

Data sets	Source	I	A	P(%)
file2.0	Eclipse_File2.0 [84]	6729	198	14.5
file2.1	Eclipse_File2.1 [84]	7888	198	10.8
file3.0	Eclipse_File3.0 [84]	10593	198	14.8
package2.0	Eclipse_Package2.0 [84]	377	207	50.4
package2.1	Eclipse_Package2.1 [84]	434	207	44.7
package3.0	Eclipse_Package2.0 [84]	661	207	47.4
jdt	Eclipse JDT Core [16]	997	212	20.7
equinox	Equinox Framework [16]	324	212	39.8
lucene	Apache Lucene [16]	691	212	9.3
mylyn	Mylyn [16]	1862	212	13.2
pde	Eclipse PDE UI [16]	1497	212	14

Table 10
10 groups of thresholds.

id	α	β
1	0.95	0.05
2	0.9	0.1
3	0.85	0.15
4	0.8	0.2
5	0.75	0.25
6	0.7	0.3
7	0.65	0.35
8	0.6	0.4
9	0.55	0.45
10	0.5	0.5

- In some real applications, it is not free to train more classifiers on more samples, which means it may bring some test cost or other kinds of cost. By considering this situation, TS3WD has an advantage over EML on reducing cost and running time.

Since TS3WD and EML have same coverage value, the comparison results on *F*-measure and accuracy will deduce same conclusion, then the comparison experiment on *F*-measure is not considered in this group.

5.2. Comparison experiments on software defect ranking

5.2.1. Data sets

There are 11 data sets with exact number of defects used in our ranking experiments. These data sets are coming from [16,84]. Information of data sets are summarized in Table 9. More detail of these data sets can also be found in [66].

5.2.2. Comparison algorithms

Yang [66] proposed a LTR (learning-to-rank) algorithm for software defect ranking, and compared LTR algorithm with other 5 ranking algorithms, such as LR (Logistical Regression [33]), NBR (Negative Binomial Regression [24]), RF (Random Forest [8]), BART (Bayesian Additive Regression Trees [13]), and RP (Recursive Partitioning [9]). In our experiments, our method will be compared with all these 6 algorithms. More details about these 6 algorithms can be found in [66].

5.2.3. Measures and parameters

A metric called FPA (Fault-Percentile-Average) is applied as the ranking measure in our experiments. The FPA is described as follows [65].

Consider a data with K modules $f_1, f_2, \ldots, f_K$, listed in increasing order of predicted numbers of defects, so f_K is the module predicted to have the most defects. Let n_k be the actual number of defects in module k, and let $N = n_1 + n_2 + \cdots + n_K$ be the total number of defects in the data. For any m in $1, \ldots, K$, the proportion of actual defects in the top m predicted modules is

$$P_m = \frac{1}{N} \sum_{k=K-m+1}^{K} n_k. \tag{14}$$

The FPA is then the average of the P_m:

$$\text{FPA} = \frac{1}{K} \sum_{m=1}^{K} \frac{1}{N} \sum_{k=K-m+1}^{K} n_k. \tag{15}$$

As both Menzies and Yang applied Information Gain to select the top 3 attributes as the evaluating metrics in their experiments, the same procedure is adopted in our experiments in order to compare with them at the same level. 10 groups of thresholds which are shown in Table 10 are tested.

In each series of experiments, ten times 10-fold cross validation are performed. Concretely, 10-fold cross validation is repeated for ten times with 10 groups of thresholds, and the average results are recorded. C4.5 is applied as the three-way decisions based classifier.

5.2.4. Results and analysis

Table 11 presents the comparison results of our ranking method and other 6 ranking methods on FPA. In this table, the values of FPA are in the form of "mean ± standard deviation", and the best performance of each row is boldfaced.

In detail, pairwise *t*-tests at 95% significance level indicate that on FPA, our two-stage ranking method performs significantly better than LTR on 3 data sets, better than RF on 4 data sets, better than RP on 8 data sets, better than BART on 4 data sets, better than NBR on 2 data sets and better than LR on 4 data sets. These comparisons are summarized in Table 12, which presents the win/tie/loss counts of our method on the row over other methods on the column.

From Tables 11 and 12, we have the following observation:

- Except the data set "equinox", our ranking method is always significantly better than or at least comparable to other 6 ranking methods.

Table 11
The comparison results of our method and other ranking methods on FPA. The best performance is boldfaced.

Data sets	Our method	LTR	RF	RP	BART	NBR	LR
file2.0	0.796 ± 0.027	0.805 ± 0.021	**0.814 ± 0.018**	0.749 ± 0.035	0.782 ± 0.030	0.796 ± 0.023	0.804 ± 0.022
file2.1	**0.795 ± 0.017**	0.765 ± 0.022	0.758 ± 0.021	0.722 ± 0.025	0.748 ± 0.025	0.749 ± 0.024	0.765 ± 0.022
file3.0	**0.813 ± 0.016**	0.788 ± 0.016	0.783 ± 0.017	0.731 ± 0.025	0.756 ± 0.023	0.775 ± 0.021	0.787 ± 0.017
pack2.0	**0.813 ± 0.016**	0.769 ± 0.063	0.756 ± 0.067	0.676 ± 0.087	0.746 ± 0.067	0.771 ± 0.061	0.766 ± 0.067
pack2.1	**0.803 ± 0.023**	0.776 ± 0.062	0.758 ± 0.066	0.724 ± 0.073	0.722 ± 0.098	0.774 ± 0.062	0.774 ± 0.062
pack3.0	**0.834 ± 0.017**	0.815 ± 0.031	0.800 ± 0.033	0.745 ± 0.051	0.792 ± 0.037	0.813 ± 0.031	0.807 ± 0.035
jdt	**0.826 ± 0.042**	0.822 ± 0.047	**0.826 ± 0.039**	0.799 ± 0.045	0.803 ± 0.045	0.820 ± 0.054	0.824 ± 0.044
equinox	0.767 ± 0.038	0.805 ± 0.039	0.804 ± 0.040	0.768 ± 0.051	0.781 ± 0.048	0.804 ± 0.040	**0.806 ± 0.038**
lucence	0.831 ± 0.061	**0.842 ± 0.063**	0.836 ± 0.072	0.748 ± 0.098	0.770 ± 0.113	0.828 ± 0.073	0.841 ± 0.065
mylyn	0.683 ± 0.081	0.735 ± 0.055	0.737 ± 0.057	0.684 ± 0.057	0.692 ± 0.072	**0.740 ± 0.053**	**0.740 ± 0.053**
pde	0.773 ± 0.062	0.779 ± 0.051	0.774 ± 0.057	0.697 ± 0.063	0.720 ± 0.083	**0.780 ± 0.050**	0.748 ± 0.06
average	**0.794 ± 0.036**	0.791 ± 0.043	0.786 ± 0.044	0.731 ± 0.055	0.756 ± 0.058	0.786 ± 0.045	0.787 ± 0.044

Table 12
Summary of the comparison (win/tie/loss) of our ranking method and other 6 methods under pairwise *t*-tests at 95% significance level.

	on FPA					
	LTR	RF	RP	BART	NBR	LR
Our ranking method	3/7/1	4/6/1	8/3/0	5/6/0	2/8/1	4/6/1

- The final average results show us that our method is the best ranking method, while LTR is the second one and RP is the worst one.

For the superiority of our proposed method, we have following analysis and discussion:

- Regression methods focus on predicting the exact number of defects in modules, which may ignore the order information among different modules. The comparison results show that regression methods could not obtain a good ranking result. Two non-regression methods including our proposed method and LTR obtains the best 2 ranking result.
- Compared to LTR, the advantage of our proposed method is that we rank defect-prone modules and not-defect-prone modules separately. Consequently, this strategy could eliminate the possibly negative influence of not-defect-prone modules on ranking defect-prone modules.
- The purpose of software defect ranking is to help users focus on the modules with more defects and allocate the testing resources effectively and efficiently. In our two-stage ranking method, the positive region in the first-stage contains the most defect-prone modules, which can be allocated more testing resources.

6. Conclusion and further work

Software defect prediction is a very important technology for improving the quality of software systems. For the majority of software defect prediction studies, software defect classification and software defect ranking are the most effective methods. In this paper, a two-stage classification method and a two-stage ranking method for cost-sensitive software defect prediction based on three-way decisions are proposed.

In the two-stage classification method, to overcome the disadvantage of classical two-way decisions classification methods, which means a high misclassification error and a high decision cost for ambiguous objects, a three-way decisions classification is considered in the first stage. All modules are classified into three different regions, while the positive region contains the defect-prone modules, the negative region contains the not-defect-prone modules and the boundary region contains the deferment modules waiting for further examination, respectively. In the second stage, an ensemble learning method is applied to classify those deferment modules to obtain the final two-way results. Several comparison experiments are also finished on NASA data. The experimental result shows that our proposed method can obtain a lower misclassification error and a lower decision cost.

In the two-stage ranking method, all modules are classified by a three-way decisions based classifier first. In the second stage, the modules in different regions are ranked separately. A dominance relation rough set based ranking method is applied. Experimental results on several data sets show the efficiency of our proposed method.

The contribution of our work is that we applied three-way decisions into the software defect prediction. How to convert three-way decisions into two-way decisions by considering adding attributes or samples in an incremental way will be an important and interesting future work.

Acknowledgments

The authors would like to thank reviewers of the paper for their constructive comments. This work is supported by the National Natural Science Foundation of China under Grant no. 61272083.

References

[1] E. Arisholm, L.C. Briand, M.J. Fuglerud, Data mining techniques for building fault-proneness models in telecom java software, in: Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering, 2007, pp. 215–224.
[2] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, J. Syst. Softw. 83 (1) (2010) 2–17.
[3] N. Azam, J.T. Yao, Analyzing uncertainties of probabilistic rough set regions with game-theoretic rough sets, Int. J. Approx. Reason. 55 (1) (2014a) 142–155.
[4] N. Azam, J.T. Yao, Game-theoretic rough sets for recommender systems, Knowl-Based Syst. 72 (2014b) 96–107.
[5] R.M. Bell, T.J. Ostrand, E.J. Weyuker, Looking for bugs in all the right places, in: Proceedings of the 2006 International Symposium on Software Testing and Analysis, 2006, pp. 61–72.
[6] S. Bibi, G. Tsoumakas, I. Stamelos, I.P. Vlahavas, Software defect prediction using regression via classification, in: Proceedings of IEEE International Conference on Computer Systems and Applications, 2006, pp. 330–336.
[7] C. Bird, N. Nagappan, H. Gall, B. Murphy, P. Devanbu, Putting it all together: using socio-technical networks to predict failures, in: Proceedings of the 20th International Symposium on Software Reliability Engineering, 2009, pp. 109–119.
[8] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[9] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth and Brooks, Monterey, CA, 1984.
[10] L.C. Briand, J. Wust, Empirical studies of quality models in object-oriented systems, Adv. Comput. 59 (2002) 97–166.
[11] C. Catal, Software fault prediction: a literature review and current trends, Expert Syst. Appl. 38 (4) (2011) 4626–4636.
[12] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Syst. Appl. 36 (4) (2009) 7346–7354.
[13] H.A. Chipman, E.I. George, R.E. McCulloch, Bart: Bayesian additive regression trees, Ann. Appl. Stat. 4 (1) (2010) 266–298.
[14] M. D'Ambros, M. Lanza, R. Robbes, On the relationship between change coupling and software defects, in: Proceedings of the 16th Working Conference on Reverse Engineering, 2009, pp. 135–144.
[15] M. D'Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches, in: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, 2010, pp. 31–41.
[16] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empir. Softw. Eng. 17 (4) (2012) 531–577.
[17] X.F. Deng, Y.Y. Yao, An information-theoretic interpretation of thresholds in probabilistic rough sets, in: Proceedings of the 7th International Conference on Rough Sets and Knowledge Technology, 2012, pp. 369–378.
[18] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, J. Syst. Software 81 (5) (2008) 649–660.
[19] R. Goudey, Do statistical inferences allowing three alternative decisions give better feedback for environmentally precautionary decision-making? J. Environ. Manage. 85 (2) (2007) 338–344.
[20] L. Guo, Y. Ma, B. Cukic, H. Singh, Robust prediction of fault-proneness by random forests, in: Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004, pp. 417–428.
[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The weka data mining software: an update, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 10–18.
[22] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE T. Softw. Eng. 38 (6) (2012) 1276–1304.
[23] A.E. Hassan, Predicting faults using the complexity of code changes, in: Proceedings of the 2009 International Conference on Software Engineering, 2009, pp. 78–88.
[24] J.M. Hilbe, Negative Binomial Regression, Cambridge University Press, 2011.
[25] B.Q. Hu, Three-way decisions space and three-way decisions, Inf. Sci. 281 (2014) 21–52.
[26] X.Y. Jia, W.W. Li, L. Shang, J.J. Chen, An optimization viewpoint of decision-theoretic rough set model, in: Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology, 2011, pp. 457–465.
[27] X.Y. Jia, W.H. Liao, Z.M. Tang, L. Shang, Minimum cost attribute reduction in decision-theoretic rough set models, Inf. Sci. 219 (2013) 151–167.
[28] X.Y. Jia, L. Shang, Three-way decisions versus two-way decisions on filtering spam email, in: Transactions on Rough Sets XVIII, 2014, pp. 69–91.
[29] X.Y. Jia, Z.M. Tang, W.H. Liao, L. Shang, On an optimization representation of decision-theoretic rough set model, Int. J. Approx. Reason. 55 (1) (2014) 156–166.
[30] X.Y. Jia, L. Shang, B. Zhou, Y.Y. Yao, Generalized attribute reduct in rough set theory, knowl-based syst., 2015, doi:10.1016/j.knosys.2015.05.017
[31] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, Object-oriented software fault prediction using neural networks, Inform. Softw. Tech. 49 (5) (2007) 483–492.

[32] T.M. Khoshgoftaar, K. Gao, N. Seliya, Attribute selection and imbalanced data: problems in software defect prediction, in: Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, 1, 2010, pp. 137–144.
[33] T.M. Khoshgoftaar, E.B. Allen, A comparative study of ordering and classification of fault-prone software modules, Empir. Softw. Eng. 4 (2) (1999) 159–186.
[34] T.M. Khoshgoftaar, E. Geleyn, L. Nguyen, L. Bullard, Cost-sensitive boosting in software quality modeling, in: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, 2002, pp. 51–60.
[35] T.M. Khoshgoftaar, N. Seliya, Analogy-based practical classification rules for software quality estimation, Empir. Softw. Eng. 8 (4) (2003) 325–350.
[36] T.M. Khoshgoftaar, N. Seliya, Comparative assessment of software quality classification techniques: An empirical case study, Empir. Softw. Eng. 9 (3) (2004) 229–257.
[37] T.M. Khoshgoftaar, X.J. Yuan, E.B. Allen, W.D. Jones, J.P. Hudepohl, Uncertain classification of fault-prone software modules, Empir. Softw. Eng. 7 (4) (2002) 297–318.
[38] S. Kim, T. Zimmermann, E. Whitehead Jr, A. Zeller, Predicting faults from cached history, in: Proceedings of the 2007 International Conference on Software Engineering, 2007, pp. 489–498.
[39] A.G. Koru, H.F. Liu, An investigation of the effect of module size on defect prediction using static measures, ACM SIGSOFT Software Engineering Notes 30 (4) (2005) 1–5.
[40] H.X. Li, X.Z. Zhou, B. Huang, D. Liu, Cost-sensitive three-way decision: a sequential strategy, in: Proceedings of the 8th International Conference on Rough Sets and Knowledge Technology, 2013, pp. 325–337.
[41] H.X. Li, X.Z. Zhou, J.B. Zhao, B. Huang, Cost-sensitive classification based on decision-theoretic rough set model, in: Proceedings of the 7th International Conference on Rough Sets and Knowledge Technology, 2012, pp. 379–388.
[42] H.X. Li, X.Z. Zhou, J. Zhao, D. Liu, Attribute reduction in decision-theoretic rough set model: a further investigation, in: Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology, 2011, pp. 466–475.
[43] W.W. Li, Z.Q. Huang, X.Y. Jia, Two-phase classification based on three-way decisions, in: Proceedings of the 8th International Conference on Rough Sets and Knowledge Technology, 2013, pp. 338–345.
[44] W. Li, D.Q. Miao, W.L. Wang, N. Zhang, Hierarchical rough decision theoretic framework for text classification, in: Proceedings of the 9th IEEE International Conference on Cognitive Informatics, 2010, pp. 484–489.
[45] Y. Li, C. Zhang, J.R. Swan, An information filtering model on the web and its application in jobagent, Knowl-Based Syst. 13 (5) (2000) 285–296.
[46] D.C. Liang, D. Liu, Systematic studies on three-way decisions with interval-valued decision-theoretic rough sets, Inf. Sci. 276 (2014) 186–203.
[47] P. Lingras, M. Chen, D.Q. Miao, Rough multi-category decision theoretic framework, in: Proceedings of the 3rd International Conference on Rough Sets and Knowledge Technology, 2008, pp. 676–683.
[48] D. Liu, T.R. Li, J.B. Zhang, Incremental updating approximations in probabilistic rough sets under the variation of attributes, Knowl-Based Syst. 73 (2015) 81–96.
[49] D. Liu, T.R. Li, D.C. Liang, Three-way decisions in dynamic decision-theoretic rough sets, in: Proceedings of the 8th International Conference on Rough Sets and Knowledge Technology, 2013, pp. 291–301.
[50] M.X. Liu, L.S. Miao, D.Q. Zhang, Two-stage cost-sensitive learning for software defect prediction, IEEE T. Reliab. 63 (2) (2014) 676–686.
[51] C. Luo, T.R. Li, H.M. Chen, Dynamic maintenance of approximations in set-valued ordered decision systems under the attribute generalization, Inf. Sci. 257 (2014) 210–228.
[52] Y. Ma, L. Guo, B. Cukic, A statistical framework for the prediction of fault-proneness, in: Advances in Machine Learning Applications in Software Engineering, 2007, pp. 237–260.
[53] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE T. Softw. Eng. 32 (1) (2007) 2–13.
[54] O. Mizuno, T. Kikuno, Training on errors experiment to detect fault-prone software modules by spam filter, in: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2007, pp. 405–414.
[55] N. Nagappan, T. Ball, Using software dependencies and churn metrics to predict field failures: An empirical case study, in: Proceedings of First International Symposium on Empirical Software Engineering and Measurement, 2007, pp. 364–373.
[56] N. Nagappan, T. Ball, A. Zeller, Mining metrics to predict component failures, in: Proceedings of the 2006 International Conference on Software Engineering, 2006, pp. 452–461.
[57] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE T. Softw. Eng. 22 (12) (1996) 886–894.
[58] H.M. Olague, L.H. Etzkorn, S. Gholston, S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-roneness of object-oriented classes developed using highly iterative or agile software development processes, IEEE T. Softw. Eng. 33 (6) (2007) 402–419.
[59] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE T. Softw. Eng. 31 (4) (2005) 340–355.
[60] T.J. Ostrand, E.J. Weyuker, How to measure success of fault prediction models, in: Proceedings of the Fourth International Workshop on Software Quality Assurance: in Conjunction with the 6th ESEC/FSE Joint Meeting, 2007, pp. 25–30.
[61] G.F. Qiu, H.Z. Li, L.D. Xu, W.X. Zhang, A knowledge processing method for intelligent systems based on inclusion degree, Expert Syst. 20 (4) (2003) 187–195.
[62] S. Shivaji, E.J. Whitehead Jr, R. Akella, S. Kim, Reducing features to improve code change-based bug prediction, IEEE T. Softw. Eng. 39 (4) (2013) 552–569.
[63] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M.D. Backer, R. Haesen, Mining software repositories for comprehensible software fault prediction models, J. Syst. Softw. 81 (5) (2008) 823–839.
[64] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Using developer information as a factor for fault prediction, in: Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering, 2007, pp. 1–7.
[65] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Comparing the effectiveness of several modeling methods for fault prediction, Empir. Softw. Eng. 15 (3) (2010) 277–295.
[66] X.X. Yang, Metrics-Based Software Defect Prediction, University of Science and Technology of China, 2013 Phd thesis.
[67] X.X. Yang, K. Tang, X. Yao, A learning-to-rank algorithm for constructing defect prediction models, in: Proceedings of Intelligent Data Engineering and Automated Learning, 2012, pp. 167–175.
[68] J.T. Yao, N. Azam, Web-based medical decision support systems for three-way medical decision making with game-theoretic rough sets, IEEE T. Fuzzy Syst. 23 (1) (2015) 3–15.
[69] Y.Y. Yao, Three-way decisions with probabilistic rough sets, Inf. Sci. 180 (3) (2010) 341–353.
[70] Y.Y. Yao, The superiority of three-way decisions in probabilistic rough set models, Inf. Sci. 181 (6) (2011) 1080–1096.
[71] Y.Y. Yao, An outline of a theory of three-way decisions, in: Proceedings of the 8th International Conference on Rough Sets and Current Trends in Computing, 2012, pp. 1–17.
[72] Y.Y. Yao, Granular computing and sequential three-way decisions, in: Proceedings of the 8th International Conference on Rough Sets and Knowledge Technology, 2013, pp. 16–27.
[73] Y.Y. Yao, The two sides of the theory of rough sets, Knowl-Based Syst. 80 (2015) 66–77.
[74] Y.Y. Yao, Y. Zhao, Attribute reduction in decision-theoretic rough set models, Inf. Sci. 178 (17) (2008) 3356–3373.
[75] H. Yu, C. Zhang, G.Y. Wang, A tree-based incremental overlappingclustering method using the three-way decision theory, knowl-based syst, Knowledge-Based Systems (2015), doi:10.1016/j.knosys.2015.05.028.
[76] H.R. Zhang, F. Min, Three-way recommender systems based on random forests, knowl-based syst, Knowledge-Based Systems (2015), doi:10.1016/j.knosys.2015.06.019.
[77] H.Y. Zhang, An investigation of the relationships between lines of code and defects, in: Proceedings of 2009 IEEE International Conference on Software Maintainance, 2009, pp. 274–283.
[78] W.X. Zhang, G.F. Qiu, Uncertain Decision Making Based on Rough Sets, Tsinghua University Press, 2005.
[79] X.Y. Zhang, D.Q. Miao, Reduction target structure-based hierarchical attribute reduction for two-category decision-theoretic rough sets, Inf. Sci. 277 (2014) 755–776.
[80] J. Zheng, Cost-sensitive boosting neural networks for software defect prediction, Expert Syst. Appl. 37 (6) (2010) 4537–4543.
[81] B. Zhou, Multi-class decision-theoretic rough sets, Int. J. Approx. Reason. 55 (1) (2014) 211–224.
[82] B. Zhou, Y.Y. Yao, J.G. Luo, Cost-sensitive three-way email spam filtering, J. Intell. Inf. Syst (2013) 1–27.
[83] Y.M. Zhou, B.W. Xu, H. Leung, On the ability of complexity metrics to predict fault-prone classes in object-oriented systems, J. Syst. Software 83 (4) (2010) 660–674.
[84] T. Zimmermann, R. Premraj, A. Zeller, Predicting defects for eclipse, in: Proceedings of the 29th International Conference on Software Engineering Workshops, 2007, pp. 76–82.