Information Sciences 191 (2012) 3–13

Contents lists available at ScienceDirect

Information Sciences

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

User preferences based software defect detection algorithms selection using MCDM

Yi Peng [a,*], Guoxun Wang [a], Honggang Wang [b]

[a] School of Management and Economics, University of Electronic Science and Technology of China, Chengdu 610054, China
[b] Department of Electrical and Computer Engineering, University of Massachusetts, Dartmouth, USA

ARTICLE INFO

Article history:
Received 10 January 2010
Received in revised form 12 April 2010
Accepted 17 April 2010
Available online 24 April 2010

Keywords:
Algorithm selection
Classification algorithm
Knowledge-driven data mining
Multi-criteria decision making (MCDM)
Software defect detection

ABSTRACT

A variety of classification algorithms for software defect detection have been developed over the years. How to select an appropriate classifier for a given task is an important issue in Data mining and knowledge discovery (DMKD). Many studies have compared different types of classification algorithms and the performances of these algorithms may vary using different performance measures and under different circumstances. Since the algorithm selection task needs to examine several criteria, such as accuracy, computational time, and misclassification rate, it can be modeled as a multiple criteria decision making (MCDM) problem. The goal of this paper is to use a set of MCDM methods to rank classification algorithms, with empirical results based on the software defect detection datasets. Since the preferences of the decision maker (DM) play an important role in algorithm evaluation and selection, this paper involved the DM during the ranking procedure by assigning user weights to the performance measures. Four MCDM methods are examined using 38 classification algorithms and 13 evaluation criteria over 10 public-domain software defect datasets. The results indicate that the boosting of CART and the boosting of C4.5 decision tree are ranked as the most appropriate algorithms for software defect datasets. Though the MCDM methods provide some conflicting results for the selected software defect datasets, they agree on most top-ranked classification algorithms.

1. Introduction

Data mining and knowledge discovery (DMKD) has made remarkable progress during the past three decades [54]. It utilizes methods, algorithms, and techniques from many disciplines, including statistics, databases, machine learning, pattern recognition, artificial intelligence, data visualization, and optimization [31]. One of the major tasks in DMKD is classification. Researchers in a variety of fields have created a large number of classification algorithms, such as decision tree, neural networks, Bayesian network, linear logistic regression, Naïve Bayes, and K-nearest-neighbor. How to select the most appropriate algorithms for a given task is an important issue in DMKD.

The algorithm selection problem is actually a central issue in many fields, including Artificial Intelligence, Operations Research, and Machine Learning [13,37,63,64]. As early as 1976, Rice (1976) formalized the problem of algorithm selection as abstract models with five ingredients: the problem space, the feature space, the criteria space, the algorithm space, and the performance measures [58]. The machine learning community presents the algorithm selection as a learning problem and focuses on the classification algorithm selection problem [2,61]. Berrer included user preferences in learning algorithm ranking schemes [6]. Nakhaeizadeh and Schnabl (1997) proposed data envelopment analysis (DEA) approach to take both

* Corresponding author.
E-mail address: pengyi@uestc.edu.cn (Y. Peng).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.04.019

positive and negative properties of data mining algorithms into consideration when ranking classification algorithms [50]. Rokach (2009) suggests that the algorithm selection can be considered as a multiple criteria decision making (MCDM) problem and encourages researchers to utilize MCDM methods to systematically choose the appropriate algorithm [3,59].

As Wolpert and Macready pointed out in their No Free Lunch (NFL) theorem that there exists no single algorithm that could achieve the best performance for all measures for a given problem domain [70]. Thus, a list of classification algorithm ranking is more useful than providing the best performed algorithm for a particular task [6]. In addition, the preferences of users play an important role in algorithms evaluation and selection. One way to get users involved in the algorithm selection procedure is to allow them to assign priorities to performance measures. Since MCDM methods can satisfy both requirements, they have great potential in the area of ranking classification algorithms.

Inspired by these previous works, this paper introduces a set of MCDM methods to rank classification algorithms for software defect prediction. As an useful software testing tool, software defect prediction can help detect software faults in an early stage, which facilitates efficient test resource allocation, improves software architecture design, and reduces the number of defective modules [44]. Software defect prediction can be modeled as a two-group classification problem by categorizing software units as either fault-prone (*fp*) or non-fault-prone (*nfp*) using historical data. Researchers have developed many classification models for software defect prediction (for example, see [14,20,29,35,39,44,47,49,54,56]).

This paper conducts an empirical study to evaluate a selection of classification algorithms using 13 performance measures over 10 public-domain datasets from the NASA Metrics Data Program (MDP) repository [19]. The classification results are then analyzed using four MCDM methods in order to rank the classifiers for software defect prediction task.

The rest of this paper is organized as follows: Sections 2 and 3 briefly describe the selected classification algorithms and MCDM methods, respectively. Section 4 presents details of the experimental study and analyzes the results; Section 5 summarizes the paper.

2. Classification algorithms

This section gives a short description of classification algorithms and ensemble learning algorithms used in the experimental study. Twelve classifiers which represent five categories of classifiers (i.e., trees, functions, Bayesian classifiers, lazy classifiers, and rules) and four ensemble learning algorithms were implemented in WEKA [68].

For trees category, we chose classification and regression tree (CART) [12], Naïve Bayes tree [41], and C4.5 [57]. Functions category includes linear logistic regression [15], radial basis function (RBF) network [7], sequential minimal optimization (SMO) [55,65], and Neural Networks [68]. Bayesian classifiers category includes Bayesian network [66] and Naïve Bayes [28]. K-nearest-neighbor (KNN) was chosen to represent lazy classifiers [24]. For rules category, decision table and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) rule induction were selected [22].

Ensemble learning algorithms construct a set of classifiers and then combine the results of these classifiers using some mechanisms to classify new data records [26]. Experimental results have shown that ensembles are often more accurate, more robust to the effects of noisy data, and achieve lower average error rate than any of the constituent classifiers [25,36,63,67]. Thus this paper includes four ensemble methods (bagging, boosting, stacking, and voting) in the experimental study.

Bagging combines multiple outputs of a learning algorithm by taking a plurality vote to get an aggregated single prediction [11]. In this study, bagging is generated by averaging probability estimates [68]. Similar to bagging, boosting method also combines the different decisions of a learning algorithm to produce an aggregated prediction [34,62]. In boosting, however, weights of training instances change in each iteration to force learning algorithms to put more emphasis on instances that were predicted incorrectly previously and less emphasis on instances that were predicted correctly previously [27]. This study chooses the adaptive boosting (AdaBoost) algorithm in the experiment [34].

In addition to bagging and boosting, two other meta-learning methods: stacking and voting are included in the experiment. Stacking generalization is a scheme for minimizing the generalization error rate of one or more learning algorithms and can combine different types of learning algorithms [68,69]. Voting is a simple average of multiple classifiers probability estimates provided by WEKA [68].

3. MCDM methods

Ranking of classification algorithms normally need to examine several criteria, such as accuracy, computational time, and misclassification rate. Therefore algorithm selection can be modeled as multiple criteria decision making (MCDM) problems [53,59]. As mentioned heretofore, algorithm ranking is a useful strategy for choosing the appropriate classifier and the preferences of users are important in algorithm ranking [6]. Some existing MCDM methods are able to rank classifiers based on multiple performance measures and take the preferences of users into the ranking process. This section introduces four MCDM methods, i.e., DEA, TOPSIS, ELECTRE, and PROMETHEE, and explains how they can be used to rank classification algorithms.

3.1. Data Envelopment Analysis (DEA)

Charnes, Cooper, and Rhodes developed data envelopment analysis (DEA) to evaluate the efficiency of decision making units (DMUs) through identifying the efficiency frontier and comparing each DMU with the frontier [20]. Since DEA is able

to estimate efficiency with minimal prior assumptions [42,45], it has a comparative advantage to approaches that require a priori assumptions, such as standard forms of statistical regression analysis [21]. During the past thirty years, various DEA extensions and models have been developed and established themselves as powerful analytical tools [23].

The original DEA model presented by Charnes, Cooper, and Rhodes [20] is called "CCR ratio model", which uses the ratio of outputs to inputs to measure the efficiency of DMUs. Assume that there are n DMUs with m inputs to produce s outputs. x_{ij} and y_{rj} represent the amount of input i and output r for DMU_j (j = 1,2,...,n), respectively. Then the ratio-form of DEA can be represented as:

$$\begin{aligned} \max \quad & h_0(u,v) = \sum_r u_r y_{ro} / \sum_i v_i x_{io}, \\ \text{subject to} \quad & \sum_r u_r y_{rj} / \sum_i v_i x_{ij} \leqslant 1 \quad \text{for} \quad j = 1,\ldots,n, \\ & u_r, v_i \geqslant 0 \quad \text{for all} \quad i \quad \text{and} \quad r. \end{aligned}$$

where the u_r's and the v_i's are the variables and the y_{ro}'s and x_{io}'s are the observed output and input values of the DMU to be evaluated (i.e., DMU_o), respectively [23]. The equivalent linear programming problem using the Charnes–Cooper transformation is [23]:

$$\begin{aligned} \max \quad & z = \sum_{r=1}^{s} \mu_r y_{ro}, \\ \text{subject to} \quad & \sum_{r=1}^{s} \mu_r y_{rj} - \sum_{i=1}^{m} v_i x_{ij} \leqslant 0, \\ & \sum_{i=1}^{m} v_i x_{io} = 1, \\ & \mu_r, v_i \geqslant 0. \end{aligned}$$

Banker, Charnes, and Cooper introduced the BCC model by adding a constraint $\sum_{j=1}^{n} \lambda_j = 1$ to the CCR model [5]. These models can be solved using the simplex method for each DMUs. DMUs with value of 1 are efficient and others are inefficient.

Nakhaeizadeh and Schnabl proposed to use DEA approach in data mining algorithms selection [50]. They argued that in order to make an objective evaluation of data mining algorithms that all the available positive and negative properties of algorithms are important and DEA models are able to take both aspects into consideration. Positive and negative properties of data mining algorithms can be considered as output and input components in DEA, respectively. For example, the overall accuracy rate of a classification algorithm is an output component and the computation time of an algorithm is an input component. Using existing DEA models, it is possible to give a comprehensive evaluation of data mining algorithms.

In this paper, the CCR and BCC models are utilized to rank a wide selection of classification algorithms. In the experiment, input components include mean absolute error, false positive rate, false negative rate, training time, and test time. Output components include the area under receiver operating characteristic (AUC), overall accuracy, F-measure, Kappa statistic, true positive rate, true negative rate, precision, and recall.

3.2. Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)

Technique for order preference by similarity to ideal solution (TOPSIS) was initially developed by Hwang and Yoon (1981) to rank alternatives over multiple criteria [38]. TOPSIS finds the best alternatives by minimizing the distance to the ideal solution and maximizing the distance to the nadir or negative-ideal solution [51].

Since its first introduction, a number of extensions and variations of TOPSIS have been developed over the years [1,16,18,40]. In the experimental study of this paper, the following TOPSIS procedure adopted from Opricovic and Tzeng and Olsonwas used [51,52]:

Step 1: calculate the normalized decision matrix. The normalized value r_{ij} is calculated as:

$$r_{ij} = x_{ij} / \sqrt{\sum_{j=1}^{J} x_{ij}^2}, \quad j = 1,\ldots,J; \quad i = 1,\ldots,n.$$

where J and n denote the number of alternatives and the number of criteria, respectively. For alternative A_j, the performance measure of the ith criterion C_i is represented by x_{ij}.

Step 2: develop a set of weights w_i for each criterion and calculate the weighted normalized decision matrix. The weighted normalized value v_{ij} is calculated as:

$$v_{ij} = w_i x_{ij}, \quad j = 1,\ldots,J; \quad i = 1,\ldots,n.$$

where w_i is the weight of the ith criterion, and $\sum_{i=1}^{n} w_i = 1$.

Step 3: find the ideal alternative solution S^+, which is calculated as:

$$S^+ = \{v_1^+, \ldots, v_n^+\} = \{(\max_j v_{ij} | i \in I'), (\min_j v_{ij} | i \in I'')\},$$

where I' is associated with benefit criteria and I'' is associated with cost criteria.

Step 4: find the negative-ideal alternative solution S^-, which is calculated as:

$$S^- = \{v_1^-, \ldots, v_n^-\} = \{(\min_j v_{ij} | i \in I'), (\max_j v_{ij} | i \in I'')\},$$

Step 5: calculate the separation measures, using the n-dimensional Euclidean distance. The separation of each alternative from the ideal solution is calculated as:

$$D_j^+ = \sqrt{\sum_{i=1}^{n} (v_{ij} - v_i^+)^2}, \quad j = 1, \ldots, J.$$

The separation of each alternative from the negative-ideal solution is calculated as:

$$D_j^- = \sqrt{\sum_{i=1}^{n} (v_{ij} - v_i^-)^2}, \quad j = 1, \ldots, J.$$

Step 6: calculate a ratio R_j^+ that measures the relative closeness to the ideal solution and is calculated as:

$$R_j^+ = D_j^- / \left(D_j^+ + D_j^-\right), \quad j = 1, \ldots, J.$$

Step 7: rank alternatives by maximizing the ratio in Step 6.

3.3. Elimination and choice expressing reality (ELECTRE)

ELECTRE stands for *ELimination et Choix TRaduisant la REalité* (ELimination and Choice Expressing the REality) and was first proposed by Roy [60] to choose the best alternative from a collection of alternatives. Over the last four decades, a family of ELECTRE methods has been developed, including ELECTRE I, ELECTRE II, ELECTRE III, ELECTRE IV, ELECTRE IS, and ELECTRE TRI.

There are two main steps of ELECTRE methods: the first step is the construction of one or several outranking relations; the second step is an exploitation procedure that identifies the best compromise alternative based on the outranking relation obtained in the first step [33]. This paper uses ELECTRE I in the experimental study following the solution procedure, which was summarized by Milani [48]:

Step 1: define a concordance and discordance index set for each pair of alternatives $A_j$and A_k, $j, k = 1, \ldots, m; i \neq k$

Step 2: calculate a global concordance index C_{ki} and a global discordance index D_{ki} for each pair of alternatives by summing the decision maker's (DM) weights.

Step 3: choose a global concordance threshold c and a global discordance threshold d to perform the global concordance and discordance tests.

Step 4: test both global concordance and global discordance thresholds:

$$\text{if } C_{ik} \geqslant c \quad \text{and} \quad D_{ik} \leqslant d, \text{ an outranking relation is judged as true,}$$

Step 5: complete the two tests for all pairs of alternatives. The preferred alternatives are those outrank more being outranked.

3.4. Preference Ranking Organisation Method for Enrichment of Evaluations (PROMETHEE)

Brans (1982) proposed the PROMETHEE I and PROMETHEE II in 1982 [8]. The PROMETHEE methods use pairwise comparisons and outranking relationships to choose the best alternatives. Specifically, they compute positive and negative preference flows for each alternative and help the DM to make final selection. The positive preference flow indicates how an alternative is outranking all the other alternatives and the negative preference flow indicates how an alternative is outranked by all the other alternatives [10]. While PROMETHEE I obtains partial ranking, PROMETHEE II provides a complete ranking. The ranking generated by PROMETHEE I is partial because it does not compare conflicting actions [9]. On the other hand, PROMETHEE II ranks alternatives according to the net flow which equals to the balance of the positive and the negative preference flows. The higher the net flow, the better the alternative [10]. Since the purpose of this paper is to build a ranking of classification algorithms, PROMETHEE II is selected. The PROMETHEE II procedure described by Brans and Mareschal was used in the experimental study [10]:

Step 1: define aggregated preference indices.

Let $a, b \in A$, and let:

$$\begin{cases} \pi(a,b) = \sum_{j=1}^{k} p_j(a,b) w_j, \\ \pi(b,a) = \sum_{j=1}^{k} p_j(b,a) w_j. \end{cases}$$

where A is a finite set of possible alternatives $\{a_1, a_2, \ldots, a_n\}$, k represents the number of evaluation criteria and w_j is the weight of each criterion. Arbitrary numbers for the weights can be assigned by the DM. The weights are then normalized to ensure that $\sum_{j=1}^{k} w_j = 1$. $\pi(a,b)$ indicates how a is preferred to b and $\pi(b,a)$ indicates how b is preferred to a. $P_j(a,b)$ and $P_j(b,a)$ are the preference functions for alternatives a and b.

Step 2: calculate $\pi(a,b)$ and $\pi(b,a)$ for each pair of alternatives of A.
Step 3: define the positive and the negative outranking flow as follows:

The positive outranking flow:

$$\phi^+(a) = \frac{1}{n-1} \sum_{x \in A} \pi(a,x).$$

The negative outranking flow:

$$\phi^-(a) = \frac{1}{n-1} \sum_{x \in A} \pi(x,a),$$

Step 4: compute the net outranking flow for each alternative as follows:

$$\phi(a) = \phi^+(a) - \phi^-(a).$$

When $\phi(a) > 0$, a is more outranking all the alternatives on all the evaluation criteria. When $\phi(a) < 0$, a is more outranked.

4. Experimental study

The experiment is designed to rank classification algorithms using the four MCDM methods described in the previous section in the application domain of software defect prediction. The following subsections describe the performance measures, data sources, experimental design, and the results.

4.1. Performance measures

There are an extensive number of performance measures for classification. Commonly used performance measures in software defect classification are accuracy, precision, recall, *F*-measure, the area under receiver operating characteristic (AUC), and mean absolute error [17,30,44,46]. Besides these popular measures, this work includes seven other classification measures. The following paragraphs briefly describe these measures.

- Overall accuracy: Accuracy is the percentage of correctly classified modules [36]. It is one the most widely used classification performance metrics.

$$\text{Overall accuracy} = \frac{TN + TP}{TP + FP + FN + TN}.$$

- True positive (TP): TP is the number of correctly classified fault-prone modules. TP rate measures how well a classifier can recognize fault-prone modules. It is also called sensitivity measure.

$$\text{True positive rate/sensitivity} = \frac{TP}{TP + FN}.$$

- False positive (FP): FP is the number of non-fault-prone modules that is misclassified as fault-prone class. FP rate measures the percentage of non-fault-prone modules that were incorrectly classified.

$$\text{False positive rate} = \frac{FP}{FP + TN}.$$

- True negative (TN): TN is the number of correctly classified non-fault-prone modules. TN rate measures how well a classifier can recognize non-fault-prone modules. It is also called specificity measure.

$$\text{True negative rate/specificity} = \frac{TN}{TN + FP}.$$

- False negative (FN): FN is the number of fault-prone modules that is misclassified as non-fault-prone class. FN rate measures the percentage of fault-prone modules that were incorrectly classified.

$$\text{False negative rate} = \frac{FN}{FN + TP}.$$

- Precision: This is the number of classified fault-prone modules that actually are fault-prone modules.

$$\text{Precision} = \frac{TP}{TP + FP}.$$

- Recall: This is the percentage of fault-prone modules that are correctly classified.

$$\text{Recall} = \frac{TP}{TP + FN}.$$

- *F*-measure: It is the harmonic mean of precision and recall. *F*-measure has been widely used in information retrieval [4].

$$F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

- AUC: ROC stands for Receiver Operating Characteristic, which shows the tradeoff between TP rate and FP rate [36]. AUC represents the accuracy of a classifier. The larger the area, the better the classifier.
- Kappa statistic (KapS): This is a classifier performance measure that estimates the similarity between the members of an ensemble in multi-classifiers systems [43].

$$\text{KapS} = \frac{P(A) - P(E)}{1 - P(E)}.$$

P(A) is the accuracy of the classifier and P(E) is the probability that agreement among classifiers is due to chance.

$$P(E) = \frac{\sum_{k=1}^{c}\left(\left[\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,k)C(i,j)\right] \cdot \left[\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,j)C(i,k)\right]\right)}{m^2},$$

m is the number of modules and$c$ is the number of classes. $f(i, j)$ is the actual probability of i module to be of class j. $\sum_{i=1}^{m} f(i,j)$ is the number of modules of class j. Given threshold θ, $C_\theta(i, j)$ is 1 if j is the predicted class for i obtained from P(i, j); otherwise it is 0 [32].
- Mean absolute error (MAE): This measures how much the predictions deviate from the true probability. P(i, j) is the estimated probability of i module to be of class j taking values in [0,1] [32].

$$\text{MAE} = \frac{\sum_{j=1}^{c}\sum_{i=1}^{m} |f(i,j) - P(i,j)|}{m \cdot c}.$$

- Training time: the time needed to train a classification algorithm or ensemble method.
- Test time: the time needed to test a classification algorithm or ensemble method.

4.2. Data sources

The data used in this study are 10 public-domain software defect datasets provided by the NASA IV& V Facility Metrics Data Program (MDP) repository. The NASA website gives brief descriptions of each MDP dataset [19]:

- CM1: This dataset is from a science instrument written in a C code with approximately 20 kilo-source lines of code (KLOC). It contains 505 modules.
- JM1: This dataset is a real-time C project containing about 315 KLOC. There are eight years of error data associated with the metrics and has 2012 modules.
- KC3: This dataset is about the collection, processing and delivery of satellite metadata. It is written in Java with 18 KLOC and has 458 modules.
- KC4: This dataset is a ground-based subscription server written in Perl code containing of 25 KLOC with 125 modules.
- MC1: This dataset is about a combustion experiment that is designed to fly on the space shuttle written in C & C++ code containing 63 KLOC. There are 9466 modules.
- MW1: This dataset is about a zero gravity experiment related to combustion written in C code containing 8 KLOC, with 403 modules.
- PC1: This dataset is flight software from an earth orbiting satellite that is no longer operational. It contains 40 KLOC of C code with 1107 modules.
- PC2: This dataset is dynamic simulator for attitude control systems. It contains 26 KLOC of C code with 5589 modules.
- PC3: This dataset is flight software from an earth orbiting satellite that is currently operational. It has 40 KLOC of C code with 1563 modules.

- PC4: This dataset is flight software from an earth orbiting satellite that is currently operational. It has 36 KLOC of C code with 1458 modules.

In two-class software defect prediction, modules with non-empty defect_id are labeled as *fp* and modules with empty defect_id are labeled as *nfp*.

4.3. Experimental design

The experiment was carried out according to the following process:

Input: a software defect dataset
Output: Ranking of classifiers
Step 1: Prepare target datasets: select and transform relevant features; data cleaning; data integration.
Step 2: Train and test the selected classification models on a randomly sampled partitions (i.e., 10-fold cross-validation) using WEKA 3.7 [68].
Step 3: Evaluate classification models using DEA, TOPSIS, ELECTRE I, and PROMETHEE II. All the MCDM methods are implemented using MATLAB.
Step 4: Generate four separate tables of the final ranking of classifiers provided by each MCDM method.
END

Note that each of bagging and boosting is applied to the twelve individual classifiers to generate twelve ensembled outputs, while each of stacking and voting is applied to the twelve classifiers to produce one prediction. Therefore there are total of thirty-eight classifiers.

Among the four MCDM methods used in the experiment, TOPSIS, ELECTRE I, and PROMETHEE II can take user preferences into the ranking procedure by assigning weights to criteria or performance measures. For the DEA method, the weights are determined for each classification algorithm during the computation of LP problem [50].

For TOPISIS, ELECTRE I, and PROMETHEE II, weights for each criterion or performance measure are defined according to the results from previous research and use the scale ranges from 1 to 9 with increasing importance. Number 1, 3, 5, 7 and 9 represent equal, moderate, strong, very strong, and extreme importance, respectively; while 2, 4, 6 and 8 indicate intermediate values [71]. AUC is assigned a weight of 9 to indicate that it is extremely important since previous studies have proved that it is the most informative and objective measurement of predictive accuracy in software defect prediction [32,44]. Mean absolute error, overall accuracy, and *F*-measure are assigned a weight of 7. False positive rate, false negative rate, Kappa statistic, true positive rate, true negative rate, precision, and recall are assigned a weight of 5. Training time and test time are assigned a weight of 1. The weights are then normalized and the sum of all weights equal to 1.

4.4. Discussion of results

The ranking of classifiers generated by DEA, TOPSIS, ELECTRE I, and PROMETHEE II are summarized in Table 1–4, respectively. For each classifier listed on the leftmost column, the corresponding output value is presented. The classifiers are named following the formats of WEKA.

Table 1 summarizes the results of the CCR and BCC models for the thirty-eight classifiers. The efficient algorithms identified by CCR and BCC models only differ at two instances: linear logistic regression (functions.Logistic) and bagging of decision table (rules.DecisionTable.Bagging). For inefficient algorithms, both models produce similar results.

The ranking results produced by the TOPSIS for the software defect datasets are summarized in Table 2. Boosting of CART (trees.SimpleCart.Adaboost), boosting of C4.5 decision tree (trees.J48.Adaboost), and K-nearest neighbours classifier (lazy.IBK) are ranked as the top-three classifiers according to the computed ratio that measures the relative closeness of each classifier to the ideal solution.

Table 3 summarizes the evaluation results of ELECTRE I. Boosting of CART (trees.SimpleCart.Adaboost), bagging of C4.5 decision tree (trees.J48.Bagging), and boosting of C4.5 decision tree (trees.J48.Adaboost) are preferred classifiers. In experiments, we arbitrary set the threshold of the concordance test to 0.95. However, as all data has been normalized to value between 0 to 1, we arbitrary set the threshold of the discordance test to 0.1.

Table 4 represents the ranking results generated by PROMETHEE II. Boosting of CART (trees.SimpleCart.Adaboost), boosting of C4.5 decision tree (trees.J48.Adaboost), and bagging of C4.5 decision tree (trees.J48.Bagging) have the highest net flows, which are computed as the balance of the positive and the negative preference flows.

Table 1–4 indicate that DEA, TOPSIS, ELECTRE I, and PROMETHEE II provide similar top-ranked classification algorithms and differ about some classifiers for software defect datasets. For example, they all consider boosting of CART and boosting of C4.5 decision tree as the top two classifiers. However, they have different opinions on Naïve Bayes (bayes.NaiveBayes) and Bayesian network (bayes.BayesNet). Both classifiers are efficient according to the results of DEA, but they are not efficient in TOPSIS, ELECTRE I, and PROMETHEE II. In fact, both classifiers are ranked quite low in TOPSIS and PROMETHEE II.

Table 1
Results of the DEA approach.

DMU/Classifier	CCR	BCC	DMU/Classifier	CCR	BCC
bayes.BayesNet.Adaboost	0.67	0.68	rules.DecisionTable.Bagging	0.74	1.00
bayes.NaiveBayes.Adaboost	0.70	0.74	rules.JRip.Bagging	0.83	0.84
functions.Logistic.Adaboost	0.66	0.67	trees.J48.Bagging	1.00	1.00
functions.MultilayerPerceptron.Adaboost	0.69	0.71	trees.NBTree.Bagging	1.00	1.00
functions.RBFNetwork.Adaboost	0.55	0.58	trees.SimpleCart.Bagging	1.00	1.00
functions.SMO.Adaboost	0.55	0.56	bayes.BayesNet	1.00	1.00
lazy.IBk.Adaboost	0.91	0.93	bayes.NaiveBayes	1.00	1.00
rules.DecisionTable.Adaboost	0.78	0.79	functions.Logistic	0.84	1.00
rules.JRip.Adaboost	1.00	1.00	functions.MultilayerPerceptron	0.68	0.71
trees.J48.Adaboost	1.00	1.00	functions.RBFNetwork	0.74	0.75
trees.NBTree.Adaboost	0.93	0.93	functions.SMO	0.67	0.71
trees.SimpleCart.Adaboost	1.00	1.00	lazy.IBk	1.00	1.00
bayes.BayesNet.Bagging	1.00	1.00	rules.DecisionTable	0.86	0.99
bayes.NaiveBayes.Bagging	0.75	0.76	rules.JRip	0.90	0.91
functions.Logistic.Bagging	0.65	0.67	trees.J48	1.00	1.00
functions.MultilayerPerceptron.Bagging	0.72	0.72	trees.NBTree	0.86	0.87
functions.RBFNetwork.Bagging	0.58	0.62	trees.SimpleCart	0.92	0.93
functions.SMO.Bagging	0.58	0.62	meta.Stacking	0.95	0.96
lazy.IBk.Bagging	1.00	1.00	meta.Vote	0.88	0.89

Table 2
Results of the TOPSIS approach.

DMU/Classifier	TOPSIS	DMU/Classifier	TOPSIS
trees.SimpleCart.Adaboost	0.9791	functions.MultilayerPerceptron.Adaboost	0.668286
trees.J48.Adaboost	0.969652	rules.JRip	0.658249
lazy.IBk	0.8816	functions.Logistic	0.569089
rules.JRip.Adaboost	0.871979	functions.Logistic.Bagging	0.567159
trees.NBTree.Adaboost	0.867601	rules.DecisionTable.Bagging	0.561985
lazy.IBk.Bagging	0.84845	functions.Logistic.Adaboost	0.558039
lazy.IBk.Adaboost	0.846795	rules.DecisionTable	0.546139
trees.J48.Bagging	0.819102	bayes.BayesNet.Adaboost	0.524538
rules.DecisionTable.Adaboost	0.810651	functions.SMO.Adaboost	0.46479
meta.Stacking	0.805535	functions.SMO.Bagging	0.453615
meta.Vote	0.800557	functions.SMO	0.448293
trees.J48	0.789733	bayes.BayesNet.Bagging	0.447375
trees.NBTree.Bagging	0.774144	bayes.BayesNet	0.440049
trees.SimpleCart.Bagging	0.772255	functions.RBFNetwork.Adaboost	0.41195
trees.NBTree	0.728909	bayes.NaiveBayes	0.356969
functions.MultilayerPerceptron.Bagging	0.705117	bayes.NaiveBayes.Bagging	0.356266
rules.JRip.Bagging	0.701797	bayes.NaiveBayes.Adaboost	0.332357
trees.SimpleCart	0.695209	functions.RBFNetwork.Bagging	0.3255
functions.MultilayerPerceptron	0.686196	functions.RBFNetwork	0.32445

Table 3
Results of the ELECTRE I approach.

DMU/Classifier	ELECTRE	DMU/Classifier	ELECTRE
bayes.BayesNet.Adaboost	0	lazy.IBk.Bagging	0
bayes.NaiveBayes.Adaboost	0	rules.DecisionTable.Bagging	0
functions.Logistic.Adaboost	0	rules.JRip.Bagging	0
functions.MultilayerPerceptron.Adaboost	0	trees.J48.Bagging	1
functions.RBFNetwork.Adaboost	0	trees.NBTree.Bagging	0
functions.SMO.Adaboost	0	trees.SimpleCart.Bagging	0
Lazy.IBk.Adaboost	0	bayes.BayesNet	0
rules.DecisionTable.Adaboost	0	bayes.NaiveBayes	0
rules.JRip.Adaboost	0	functions.Logistic	0
Trees.J48.Adaboost	1	functions.MultilayerPerceptron	0
Trees.NBTree.Adaboost	0	functions.RBFNetwork	0
Trees.SimpleCart.Adaboost	1	functions.SMO	0
bayes.BayesNet.Bagging	0	lazy.IBk	0
bayes.NaiveBayes.Bagging	0	rules.DecisionTable	0
functions.Logistic.Bagging	0	rules.JRip	0
functions.MultilayerPerceptron.Bagging	0	trees.J48	0
functions.RBFNetwork.Bagging	0	trees.NBTree	0
functions.SMO.Bagging	0	trees.SimpleCart	0

Table 4
Results of the PROMETHEE II approach.

DMU/Classifier	PROMETHEE	DMU/Classifier	PROMETHEE
Trees.SimpleCart.Adaboost	0.935458	rules.JRip	−0.09802
Trees.J48.Adaboost	0.822509	functions.MultilayerPerceptron	−0.10044
Trees.J48.Bagging	0.732957	rules.DecisionTable	−0.1545
meta.Stacking	0.599032	functions.MultilayerPerceptron.Adaboost	−0.16015
Trees.SimpleCart.Bagging	0.575635	functions.Logistic	−0.19161
rules.JRip.Adaboost	0.571601	functions.Logistic.Bagging	−0.25938
Trees.NBTree.Bagging	0.570795	functions.Logistic.Adaboost	−0.33683
Trees.NBTree.Adaboost	0.561113	bayes.BayesNet.Adaboost	−0.39653
Lazy.IBk	0.50948	bayes.BayesNet.Bagging	−0.42719
Lazy.IBk.Bagging	0.501412	bayes.BayesNet	−0.45623
rules.DecisionTable.Adaboost	0.393304	functions.SMO.Adaboost	−0.48689
Trees.J48	0.351351	functions.SMO.Bagging	−0.57886
Lazy.IBk.Adaboost	0.335216	functions.RBFNetwork.Adaboost	−0.5837
rules.JRip.Bagging	0.201694	functions.SMO	−0.61799
meta.Vote	0.198064	functions.RBFNetwork.Bagging	−0.62888
rules.DecisionTable.Bagging	0.153691	functions.RBFNetwork	−0.65551
functions.MultilayerPerceptron.Bagging	0.143203	bayes.NaiveBayes	−0.69342
Trees.NBTree	0.119	bayes.NaiveBayes.Bagging	−0.70069

5. Conclusion remarks

Inconsistencies exist in different studies and the performances of learning algorithms may vary using different performance measures and under different circumstances. The selection of an appropriate classification algorithm is an important and difficult task. This paper proposed to use MCDM methods (i.e., DEA, ELECTRE I, TOPSIS, and PROMETHEE II) to evaluate and rank a selection of classification algorithms using a set of performance measures for software defect prediction. Since the preferences of the decision maker (DM) play an important role in algorithm evaluation and selection, this paper involved user's preferences during the ranking procedure by assigning weights to evaluation criteria. It started by introducing the classification algorithms and the MCDM methods. Then the experiment, which used 38 classification algorithms and 13 evaluation criteria over 10 software defect datasets, was described.

The experimental results indicate that *the boosting of CART* and *the boosting of C4.5 decision tree* are ranked as the most appropriate algorithms for software defect datasets. The second observation is that the four MCDM methods generate similar top-ranked classification algorithms while produce different ranking for some classifiers for the selected software defect datasets. The third observation is that TOPSIS and PROMETHEE II may be more appropriate than DEA and ELECTRE I for the given task since they provide a complete ranking of algorithms.

Acknowledgements

A short 5 pages version of this paper has been appeared at the proceeding of the 2nd International Conference on Software Engineering and Data Mining.

This research has been supported by Grants from the National Natural Science Foundation of China under the Grant No. 70901011, No. 70901015, No. 70921061, No. 70531040; the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

References

[1] M.A. Abo-Sinna, A.H. Amer, Extensions of TOPSIS for multi-objective large-scale nonlinear programming problems, Applied Mathematics and Computation 162 (2005) 243–256.
[2] D. Aha, Generalizing from case studies: a case study, in: Proceedings of the Nineth International Conference on Machine Learning, 1992, pp. 1–10.
[3] N. Ahmad, P. Laplante, Using the analytical hierarchy process in selecting commercial real-time operating systems, International Journal of Information Technology and Decision Making 8 (1) (2009) 151–168.
[4] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison Wesley, 1999.
[5] R. Banker, A. Charnes, W.W. Cooper, Some models for estimating technical and scale inefficiencies in data envelopment analysis, Management Science 30 (1984) 1078–1092.
[6] H. Berrer, I. Paterson, J. Keller, Evaluation of machine-learning algorithm ranking advisors, in: P. Brazdil, A. Jorge (Eds.), Proceedings of the PKDD2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions, 2000, pp. 1–13.
[7] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[8] J.P. Brans, L'ingénièrie de la décision; elaboration d'instruments d'aide à la décision. La méthode PROMETHEE, in: R. Nadeau, M. Landry (Eds.), L'aide à la décision: Nature, Instruments et Perspectives d'Avenir, Presses de l'Université Laval, Québec, Canada, 1982, pp. 183–213.
[9] J.P. Brans, B. Mareschal, 1994, How to decide with PROMETHEE, available at: <http://www.visualdecision.com/Pdf/How%20to%20use%20PROMETHEE.pdf>.
[10] J.P. Brans, B. Mareschal, PROMETHEE methods, in: J. Figueira, V. Mousseau, B. Roy (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, New York, 2005, pp. 163–195.

[11] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.
[12] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, California, 1984.
[13] C. Castiello, G. Castellano, A. Fanelli, MINDFUL: a framework for meta-inductive neuro-fuzzy learning, Information Sciences 178 (16) (2008) 3253–3274.
[14] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (8) (2009) 1040–1058.
[15] S. le Cessie, J.C. Houwelingen, Ridge estimators in logistic regression, Applied Statistics 41 (1) (1992) 191–201.
[16] Y. Cha, M. Jung, Satisfaction assessment of multi-objective schedules using neural fuzzy methodology, International Journal of Production Research 41 (8) (2003) 1831–1849.
[17] V.U.B. Challagulla, F.B. Bastani, I.Y. Raymond, A. Paul, Empirical assessment of machine learning based software defect prediction techniques, International Journal on Artificial Intelligence Tools 17 (2) (2008) 389–400.
[18] T.-Ch. Chu, Facility location selection using fuzzy TOPSIS under group decisions, International Journal of Uncertainty, Fuzziness & Knowledge-Based Systems 10 (6) (2002) 687–701.
[19] M. Chapman, P. Callis, W. Jackson, Metrics Data Program, NASA IV and V Facility, 2004, <http://mdp.ivv.nasa.gov/>.
[20] A. Charnes, W.W. Cooper, E. Rhodes, Measuring the efficiency of decision making units, European Journal of Operational Research 2 (6) (1978) 429–444.
[21] L. Cherchye, T. Post, Methodological advances in DEA: a survey and an application for the Dutch electricity sector, Statistica Neerlandica 57 (4) (2003) 410–438.
[22] W.W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufman, 1995, pp. 115–123.
[23] W.W. Cooper, L.M. Seiford, J. Zhu, Data envelopment analysis: history, models and interpretations, in: W.W. Cooper, L.M. Seiford, J. Zhu (Eds.), Handbook on Data Envelopment Analysis, Kluwer Academic Publisher, Boston, 2004. Chapter 1, 1–39.
[24] B.V. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, 1991.
[25] T.G. Dietterich, Machine learning research: four current directions, AI Magazine 18 (1997) 97–136.
[26] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Machine Learning 40 (2) (2000) 139–157.
[27] T.G. Dietterich, Ensemble methods in machine learning, in: J. Kittler, F. Roli (Eds.), First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science, vol. 1857, Springer Verlag, New York, 2000, pp. 1–15.
[28] P. Domingos, M. Pazzani, On the optimality of the simple Bayesian classifier under zero-one loss, Machine Learning 29 (203) (1997) 103–130.
[29] K. El-Emam, S. Benlarbi, N. Goel, S.N. Rai, Comparing case-based reasoning classifiers for predicting high risk software components, Journal of Systems and Software 55 (3) (2001) 301–310.
[30] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, Journal of Systems and Software 81 (5) (2008) 649–660.
[31] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery: an overview, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 1–34.
[32] C. Ferri, J. Hernandezorallo, R. Modroiu, An experimental comparison of performance measures for classification, Pattern Recognition Letters (2009) 27–38. January 2009.
[33] J. Figueira, V. Mousseau, B. Roy, Eletre methods, ELECTRE methods, in: J. Figueira, V. Mousseau, B. Roy (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, New York, 2005, pp. 133–153.
[34] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Proceedings of 13th International Conference on Machine Learning, Morgan Kaufman, San Francisco, 1996, pp. 148–156.
[35] K. Ganesan, T.M. Khoshgoftaar, E.B. Allen, Case-based software quality prediction, International Journal of Software Engineering and Knowledge Engineering 10 (2) (2000) 139–152.
[36] J. Han, M. Kamber, Data Mining: Concepts and Techniques, second ed., Morgan Kaufman, 2006.
[37] J.V. Hansen, Combining predictors: comparison of five meta machine learning methods, Information Sciences 119 (1–2) (1999) 91–105.
[38] C.L. Hwang, K. Yoon, Multiple Attribute Decision Making Methods and Applications, Springer, Berlin Heidelberg, 1981.
[39] T.M. Khoshgoftaar, E.B. Allen, J. Deng, Using regression trees to classify fault-prone software modules, IEEE Transactions on Reliability 51 (4) (2002) 455–462.
[40] G. Kim, C. Park, K.P. Yoon, Identifying investment opportunities for advanced manufacturing system with comparative-integrated performance measurement, International Journal of Production Economics 50 (1997) 23–33.
[41] R. Kohavi, Scaling up the accuracy of Naïve Bayes classifiers: a decision tree hybrid, in: E. Simoudis, J.W. Han, U. Fayyad (Eds.), Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, Portland, OR. Menlo Park, CA, 1996, pp. 202–207.
[42] M. Koksalan, C. Tuncer, A DEA-based approach to ranking multi-criteria alternatives, International Journal of Information Technology and Decision Making 8 (1) (2009) 29–54.
[43] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley, 2004.
[44] 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 (4) (2008) 485–496.
[45] H. Li, L. Ma, Ranking decision alternatives by integrated DEA, AHP and gower plot techniques, International Journal Of Information Technology & Decision Making 7 (2) (2008) 241–258.
[46] C. Mair, G. Kadoda, M. Leflel, L. Phapl, K. Schofield, M. Shepperd, S. Webster, An investigation of machine learning based prediction systems, Journal of Systems Software 53 (1) (2000) 23–29.
[47] T. Menzies, J. DiStefano, A. Orrego, R. Chapman, Assessing predictors of software defects, in: Proceedings of Workshop Predictive Software Models, 2004.
[48] A.S. Milani, A. Shanian, C. El-Lahham, Using different ELECTRE methods in strategic planning in the presence of human behavioral resistance, Journal of Applied Mathematics and Decision Sciences 2006 (2006), doi:10.1155/JAMDS/2006/10936. Article ID 10936, 19 pages.
[49] J.C. Munson, T.M. Khoshgoftaar, The detection of fault-prone programs, IEEE Transactions on Software Engineering 18 (5) (1992) 423–433.
[50] G. Nakhaeizadeh, A. Schnabl, Development of multi-criteria metrics for evaluation of data mining algorithms, in: Proceeding of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97), Newport Beach, California, August 14–17, 1997, pp.37–42.
[51] D.L. Olson, Comparison of weights in TOPSIS models, Mathematical and Computer Modelling 40 (7–8) (2004) 721–727.
[52] S. Opricovic, G.H. Tzeng, Compromise solution by MCDM methods: a comparative analysis of VIKOR and TOPSIS, European Journal of Operational Research 156 (2004) 445–455.
[53] Y. Peng, G. Kou, G. Wang, H. Wang, F. Ko, Empirical evaluation of classifiers for software risk management, International Journal of Information Technology and Decision Making 8 (4) (2009) 749–768.
[54] Y. Peng, G. Kou, Y. Shi, Z. Chen, A descriptive framework for the field of data mining and knowledge discovery, International Journal of Information Technology and Decision Making 7 (4) (2008) 639–682.
[55] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schotolkopf, C.J.C. Burges, A. Smola (Eds.), Advances in Kernel Methods-Support Vector Learning, MIT press, 1998, pp. 185–208.
[56] T.S. Quah, Estimating software readiness using predictive models, Information Sciences 179 (4) (2009) 430–445.
[57] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman, 1993.

[58] J. Rice, The algorithm selection problem, Advances in Computers 15 (1976) 65–118.
[59] L. Rokach, (2009) Ensemble-based classifiers, Artificial Intelligence Review, 19 November 2009, DOI 10.1007/s10462-009-9124-7, published online.
[60] B. Roy, "Classement et choix en presence de points de vue multiples (la methode ELECTRE)" R.I.R.O 8, 1968, pp. 57–75.
[61] K.A. Smith-Miles, Cross-Disciplinary perspectives on meta-learning for algorithm selection, ACM Computing Surveys 41 (1) (2008), doi:10.1145/1456650.1456656. December 2008, <http://doi.acm.org/10.1145/1456650.1456656>.
[62] R. Schapire, The strength of weak learnability, Machine Learning 5 (2) (1990) 197–227.
[63] K.M. Ting, Z. Zheng, A study of AdaBoost with Naïve Bayesian classifiers: weakness and improvement, Computational Intelligence 19 (2) (2003) 186–200.
[64] A. Ulaş, M. Semerci, O. Yıldız, E. Alpaydın, Incremental construction of classifier and discriminant ensembles, Information Sciences 179 (9) (2009) 1298–1318.
[65] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, USA, 1995.
[66] S.M. Weiss, C.A. Kulikowski, Computer Systems that Learn: Classification and Predication Methods from Statistics, Neural Nets, Machine Learning and Expert Systems, Morgan Kaufmann, 1991.
[67] T. Wilson, J. Wiebe, R. Hwa, Recognizing strong and weak opinion clauses, Computational Intelligence 22 (2) (2006) 73–99.
[68] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, second ed., Morgan Kaufman, San Francisco, 2005.
[69] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241–259.
[70] D.H. Wolpert, W.G. Macready, No Free Lunch Theorems for Search, Technical Report SFI-TR-95-02-010 (Santa Fe Institute), 1995.
[71] F. Zahedi, The analytic hierarchy process-a survey of the method and its applications, Interfaces 16 (4) (1986) 96–108.