Information Sciences 191 (2012) 31–46

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining

Lean Yu *

School of Management and Economics, University of Electronic Science and Technology of China, Chengdu 610054, China
MADIS, Institute of Systems Science, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
Hangzhou Key Laboratory of E-Business and Information Security, Hangzhou Normal University, Hangzhou 310036, China

ARTICLE INFO

Article history:
Available online 6 October 2011

Keywords:
Asymmetric weighted least squares support vector machine
Evolutionary programming
Ensemble learning algorithm
Software repository mining

ABSTRACT

In this paper, a novel evolutionary programming (EP) based asymmetric weighted least squares support vector machine (LSSVM) ensemble learning methodology is proposed for software repository mining. In this methodology, an asymmetric weighted LSSVM model is first proposed. Then the process of building the EP-based asymmetric weighted LSSVM ensemble learning methodology is described in detail. Two publicly available software defect datasets are finally used for illustration and verification of the effectiveness of the proposed EP-based asymmetric weighted LSSVM ensemble learning methodology. Experimental results reveal that the proposed EP-based asymmetric weighted LSSVM ensemble learning methodology can produce promising classification accuracy in software repository mining, relative to other classification methods listed in this study.

1. Introduction

Identifying faults or errors in software modules is one of the most important tasks for software engineers and managers. On the one hand, this enables software managers to focus testing activities on modules classified as fault-prone and thus improving software quality. On the other hand, this reduces testing expenditure significantly and greatly improves profitability of software companies [29]. However, it is difficult to identify faults or errors in software modules because the factors that affect the software development process are hard to be estimated beforehand. Thus, software managers are often confronted with software projects that contain errors or inconsistencies and exceed budget and time limits [29]. For this purpose, software mining, an emerging interdisciplinary field, is used to solve some of these issues by extracting knowledge from past project data that may be applied to future software projects, thus supporting software development and improving the software quality.

Because software mining techniques allow software engineers and managers to allocate their testing budgets to those modules that are most likely to contain errors, based on the results of software mining algorithms, software mining is naturally used as an important tool to find possible bugs or errors in software modules to improve the quality of software systems. Obviously, software mining is a far more efficient approach than equally allocating the testing budget to the whole software system. On the one hand, if some software modules are classified as fault-prone, more attention would be paid to modules that may contain errors, thus improving the quality of software systems. On the other hand, if some software

* Address: MADIS, Institute of Systems Science, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China.
E-mail address: yulean@amss.ac.cn

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.09.034

modules are classified as non-fault-prone, a considerable amount of extensive testing effort on these modules can be avoided, thus saving software testing cost. Usually, software measurements in software projects are usually conducted at the level of program functions, subroutines or methods. Thus in the remainder of this paper, a software module refers to a program function, subroutine or method [29].

In the existing software engineering literature, many commonly used single data mining techniques such as linear regression, nonlinear regression, belief networks, clustering and artificial intelligence techniques are widely applied to software fault prediction. For example, Fenton and Neil [7] applied Bayesian belief networks to predict software defects and Graves et al. [8] utilized linear regression model and nonlinear weighted time damp model to predict fault incidences in terms of software changes in 1999. Likewise, Shepperd et al. [25] used linear regression model to construct a prediction system for software engineers in 2000. In 2004, Zhong et al. [40] used *k*-means clustering techniques to classify software modules to enhance quality and Ying et al. [33] applied association analysis technique to predict source code changes by mining change history and obtained good performance. Similarly, Zimmermann et al. [41] and Song et al. [27] used the association analysis to predict software changes and defect associations in 2005 and 2006, respectively. In the same way, Williams and Hollingsworth [31] and Zhang and Pham [39] adopted statistical analysis models to predict software faults and software field failure rate individually, while Song et al. [26] used gray relational analysis to select feature subsets and predict software efforts in an early stage of development. In 2008, Vandecruys et al. [29] proposed an ant colony optimization (ACO)-based classification technique to construct comprehensible software fault prediction models and obtained good performance. Peng et al. [20] presented 13 single classifiers to predict software defects in software development and they found support vector machines (SVM), C4.5 algorithm, and *k*-nearest-neighbor (KNN) algorithm to be the top three classifiers for 11 public domain software defect datasets. In 2009, Catal and Diri [1] applied some new computational intelligence approaches such as artificial immune system and random forests to predict software defects and investigated the effect of data size, metrics sets and feature selection techniques on software fault prediction problems. Recently, Peng et al. [21] adopted multi-criteria decision-making (MCDM) techniques and user preference information to discover software defects, while Park et al. [17] designed a polynomial function-based neural network predictor to detect software defects and obtained good performance. Lessmann et al. [12] proposed a comparative framework for software defects prediction and conducted many large-scale experiments. The experimental results revealed that there are no significant performance differences among the top-17 classifiers.

Although almost all classification techniques can be used to predict software defects, some hybrid or combined classifiers, which integrate two or more single classification methods, have shown higher classification accuracy than any individual techniques for software mining tasks. For example, MacDonell and Shepperd [13] combined linear regression, case-based reasoning (CBR) and opinions of experts to optimize the effort for fault prediction in software project management in 2003. Similarly, Khoshgoftaar and Seliya [11] integrated CBR, majority voting and data clustering techniques to classify modules for quality estimation and Dick et al. [3] integrated fuzzy clustering and principal components analysis (PCA) for software fault prediction in the same year. In 2004, Menzies et al. [14] integrated linear regression and rule induction techniques to mine software repositories for enhancing project planning and resource allocation. In 2007, Kanellopoulos et al. [10] used *k*-means clustering and multiple minimum support (MMS) Apriori association analysis to acquire some useful rules to support software maintenance by mining program source code. In 2009, Chang et al. [2] integrated in-process software defect prediction with association mining techniques to discover defect patterns so as to improve the quality of software systems. Meantime, Quah [22] applied neural network and genetic algorithm (GA) to predict software defects and to estimate software readiness. Recently, Peng et al. [19] proposed an AHP-based ensemble evaluation method to predict software defects and the experimental results revealed that the ensemble method can improve classification performance of software defects prediction. Generally speaking, these hybrid or combined approaches have performed better than single data mining techniques. It should be noted that in previous studies, about two-thirds of the papers have used private datasets, i.e., datasets are not publicly accessible. It should be clear, however, that using publicly available datasets can offer important advantages [29]. For example, public datasets can be used for comparisons of existing and new data mining and knowledge discovery [18] techniques. For this reason, this study adopts the most commonly used state-of-the-art data classification techniques, and applies them to publicly accessible datasets for comparison purpose.

Furthermore, inspired by hybrid or combined learning techniques, this study proposes an evolutionary programming (EP) [4] based asymmetric weighted least squares support vector machines (AW-LSSVM) ensemble learning methodology for software mining. In this methodology, an asymmetric weighted least squares support vector machines (AW-LSSVM) is proposed as a generic learner to construct an efficient ensemble predictor for software fault mining. Then the results from different generic learners are aggregated into an ensemble output using an evolutionary programming (EP) [4] technique.

The main motivations of this paper reflect the following threefold: (1) to show how to construct an EP-based asymmetric weighted LSSVM ensemble learning methodology; (2) to show how to identify software faults or errors using the EP-based asymmetric weighted LSSVM ensemble learning methodology; and (3) to display how various methods compare in their performance in identifying software faults. This paper mainly describes the process of building the proposed EP-based asymmetric weighted LSSVM ensemble learning methodology and its applications in software repository mining, while comparing its performance with some commonly used state-of-the-art classification techniques in terms of different evaluation criteria.

The rest of this paper is organized as follows: In the next section, an asymmetric weighted least squares support vector machines (AW-LSSVM) model is briefly proposed. In Section 3, the process of building the EP-based AW-LSSVM ensemble

learning methodology is presented in detail. For illustration purpose, empirical results of testing on two publicly available software datasets are reported in Section 4. The conclusions are contained in Section 5.

2. Asymmetric weighted least squares support vector machines

Suppose there is a dataset $\{x_i,y_i\}$ $(i = 1,2,\ldots,N)$ with N samples where $x_i \in R^N$ is the ith input pattern and y_i is its corresponding observed result, and it is a binary variable. In software faults mining models, x_i denotes the attributes of software metrics, and y_i is the observed outcome of software modules. If there are some faults or errors in software modules, $y_i = 1$, or else $y_i = -1$. The support vector machine (SVM) model first maps the input data into a high-dimensional feature space through a mapping function $\phi(\cdot)$ and finds the optimal separating hyperplane with minimal classification errors. The separating hyperplane can be represented below:

$$z(x) = w^T\phi(x) + b = 0 \tag{1}$$

where w is the normal vector of the hyperplane and b is the bias, which is a scalar.

Suppose $\phi(\cdot)$ is a nonlinear function that maps the input space into a higher dimensional feature space. If the dataset is linearly separable in this feature space, the classifier should be constructed below

$$\begin{cases} w^T\phi(x_i) + b \geqslant 1 & \text{if } y_i = 1 \\ w^T\phi(x_i) + b \leqslant -1 & \text{if } y_i = -1 \end{cases} \tag{2}$$

which is equivalent to

$$y_i(w^T\phi(x_i) + b) \geqslant 1, \quad i = 1,\ldots,N \tag{3}$$

In order to deal with linear inseparable data, the previous analysis can be generalized by introducing some nonnegative variables $\xi_i \geqslant 0$, such that (3) is modified as

$$\begin{cases} y_i[w^T\phi(x_i) + b] \geqslant 1 - \xi_i, & i = 1,\ldots,N \\ \xi_i \geqslant 0, & i = 1,\ldots,N \end{cases} \tag{4}$$

Nonnegative variables ξ_i in (4) are those for which data point x_i does not satisfy (3). Thus the term $\sum_{i=1}^{N}\xi_i$ can be considered as a measure of the amount of misclassification, i.e. tolerable misclassification errors.

According to the structural risk minimization (SRM) principle, the risk bound is minimized by formulating the following optimization problem.

$$\begin{cases} \text{Minimize} & J(w,b;\xi_i) = \frac{1}{2}w^Tw + C\sum_{i=1}^{N}\xi_i \\ \text{Subject to} & y_i(w^T\phi(x_i) + b) \geqslant 1 - \xi_i, \quad i = 1,\ldots,N \\ & \xi_i \geqslant 0, \quad i = 1,\ldots,N \end{cases} \tag{5}$$

where C is a penalty parameter controlling the trade-off between classification margin and tolerable misclassification errors.

Searching the optimal hyperplane in (5) is a quadratic programming (QP) problem [30]. When a large-scale QP problem is computed, it may lead to a high computational cost. For this purpose, Suykens and Vandewalle [28] proposed a least squares version of SVM. In the least squares support vector machine (LSSVM) classifier, the following optimization problem for LSSVM [28] can be formulated

$$\begin{cases} \text{Minimize} & J(w,b;\xi_i) = \frac{1}{2}w^Tw + \frac{C}{2}\sum_{i=1}^{N}\xi_i^2 \\ \text{Subject to} & y_i(w^T\phi(x_i) + b) = 1 - \xi_i, \quad i = 1,\ldots,N \end{cases} \tag{6}$$

In (6), penalty parameter C determines the trade-off between classification margin and tolerable misclassification errors, as similar to (5). With the increase of C, the relative importance of misclassification errors will increase, relative to the classification margin, and vice versa. Usually, in standard SVM and LSSVM, penalty parameter C is equal to weight of misclassification error function [30], and of least squares error function [28]. That is, penalty parameter C is a constant or a fixed value in standard SVM and LSSVM.

However, many practical applications have shown that a fixed penalty parameter C is unsuitable for some classification tasks. In some cases, different classes might have different importance for different classification tasks, i.e. higher penalties should be given to classes with higher importance [34]. In the case of software repository mining, more penalties should be given to the class of fault-prone software modules, taking into account the fact that erroneous software modules lead to more costs for software company incurring by delivering fault-prone software modules. Not only will the software company spend extra money and human resources to correct the erroneous software modules, but the reputation of the software company will also be damaged. As costs associated with misclassification of a fault-prone software module are clearly higher than the costs associated with misclassification of a non-faulty software module [29], it is natural to assign different penalty parameters to different classes. For this purpose, different penalty parameters for different classes in the above LSSVM formulation should be used to obtain robust classification results. Accordingly, the following form can be formulated

$$\begin{cases} \text{Minimize} & J(w,b;\xi_i) = \frac{1}{2}w^T w + \frac{C^+}{2}\sum_{i=1}^{N^+}\left(\xi_i^+\right)^2 + \frac{C^-}{2}\sum_{i=1}^{N^-}\left(\xi_i^-\right)^2 \\ \text{Subject to} & y_i^+\left(w^T\phi\left(x_i^+\right)+b\right) = 1-\xi_i^+, \quad i=1,\ldots,N^+ \\ & y_i^-\left(w^T\phi\left(x_i^-\right)+b\right) = 1-\xi_i^-, \quad i=1,\ldots,N^- \end{cases} \tag{7}$$

where C^+ and C^- are penalty parameters for positive and negative classes, N^+ and N^- are the number of positive and negative classes, respectively.

Using (7), one can define a Lagrangian function as follows:

$$\begin{aligned} L\left(w,b,\xi_i^+,\xi_i^-,C^+,C^-;\boldsymbol{\alpha}_i^+,\boldsymbol{\alpha}_i^-\right) = {} & \frac{1}{2}w^T w + \frac{C^+}{2}\sum_{i=1}^{N^+}\left(\xi_i^+\right)^2 + \frac{C^-}{2}\sum_{i=1}^{N^-}\left(\xi_i^-\right)^2 - \sum_{i=1}^{N^+}\boldsymbol{\alpha}_i^+\left[y_i^+\left(w^T\varphi\left(x_i^+\right)+b\right)-1+\xi_i^+\right] \\ & - \sum_{i=1}^{N^-}\boldsymbol{\alpha}_i^-\left[y_i^-\left(w^T\varphi\left(x_i^-\right)+b\right)-1+\xi_i^-\right] \end{aligned} \tag{8}$$

where $\boldsymbol{\alpha}_i^+$ and $\boldsymbol{\alpha}_i^-$ are Lagrangian multipliers of positive and negative classes, respectively. The condition for optimality in (7) can be obtained from (8), as shown below

$$\begin{cases} \frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{N^+}\boldsymbol{\alpha}_i^+ y_i^+\varphi\left(x_i^+\right) + \sum_{i=1}^{N^-}\boldsymbol{\alpha}_i^- y_i^-\varphi\left(x_i^-\right) \\ \frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N^+}\boldsymbol{\alpha}_i^+ y_i^+ + \sum_{i=1}^{N^-}\boldsymbol{\alpha}_i^- y_i^- = 0 \\ \frac{\partial L}{\partial \xi_i^+} = C^+\xi_i^+ - \boldsymbol{\alpha}_i^+ = 0 \Rightarrow \xi_i^+ = \boldsymbol{\alpha}_i^+/C^+, \quad i=1,2,\ldots,N^+ \\ \frac{\partial L}{\partial \xi_i^-} = C^-\xi_i^- - \boldsymbol{\alpha}_i^- = 0 \Rightarrow \xi_i^- = \boldsymbol{\alpha}_i^-/C^-, \quad i=1,2,\ldots,N^- \\ \frac{\partial L}{\partial \boldsymbol{\alpha}_i^+} = 0 \Rightarrow y_i^+\left[w^T\varphi(x_i^+)+b\right]-1+\xi_i^+ = 0, \quad i=1,2,\ldots,N^+ \\ \frac{\partial L}{\partial \boldsymbol{\alpha}_i^-} = 0 \Rightarrow y_i^-\left[w^T\varphi(x_i^-)+b\right]-1+\xi_i^- = 0, \quad i=1,2,\ldots,N^- \end{cases} \tag{9}$$

After elimination of w and ξ_i in (9), the solution is given by the following set of linear equations:

$$\begin{cases} \sum_{i,j=1}^{N^+}\boldsymbol{\alpha}_i^+ y_i^+ y_j^+\varphi\left(x_i^+\right)^T\varphi\left(x_j^+\right) + \sum_{i=1,j=1}^{N^+,N^-}\boldsymbol{\alpha}_i^- y_i^+ y_j^-\varphi\left(x_i^+\right)^T\varphi\left(x_j^-\right) + by_i^+ + \left(\boldsymbol{\alpha}_i^+/C^+\right) - 1 = 0 \\ \sum_{i=1,j=1}^{N^+,N^-}\boldsymbol{\alpha}_i^+ y_i^+ y_j^-\varphi\left(x_i^+\right)^T\varphi\left(x_j^-\right) + \sum_{i,j=1}^{N^-}\boldsymbol{\alpha}_i^- y_i^- y_j^-\varphi\left(x_i^-\right)^T\varphi\left(x_j^-\right) + by_i^- + \left(\boldsymbol{\alpha}_i^-/C^-\right) - 1 = 0 \\ \sum_{i}^{N^+}\boldsymbol{\alpha}_i^+ y_i^+ + \sum_{i}^{N^-}\boldsymbol{\alpha}_i^- y_i^- = 0 \end{cases} \tag{10}$$

Using the Mercer condition, the kernel function can be defined as $K(x_i,x_j) = \varphi(x_i)^T\varphi(x_j)$, for $i,j = 1,2,\ldots,N^+(N^-)$. Typical kernel functions include linear kernel $K(x_i,x_j) = x_i^T x_j$, polynomial kernel $K(x_i,x_j) = \left(x_i^T x_j + 1\right)^d$, Gaussian or RBF kernel $K(x_i,x_j) = \exp\left(-\|x_i - x_j\|^2/\sigma^2\right)$, and MLP kernel $K(x_i,x_j) = \tanh\left(\beta x_i^T x_j + \theta\right)$, where d, σ, β and θ are kernel parameters specified by users beforehand [30]. Accordingly, using the matrix form, linear equations in (10) can be rewritten as

$$\begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{\Lambda} & \mathbf{Y}^+ \\ \boldsymbol{\Lambda} & \boldsymbol{\Theta} & \mathbf{Y}^- \\ (\mathbf{Y}^+)^T & (\mathbf{Y}^-)^T & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ 0 \end{bmatrix} \tag{11}$$

where b is a scalar, $\boldsymbol{\Omega}$, $\boldsymbol{\Theta}$,$\boldsymbol{\Lambda}$, $\mathbf{Y}^+$, $\mathbf{Y}^-$, $\boldsymbol{\alpha}^+$, $\boldsymbol{\alpha}^-$ and $\mathbf{1}$ are defined as follows,

$$\boldsymbol{\Omega} = y_i^+ y_j^+\varphi\left(x_i^+\right)^T\varphi\left(x_j^+\right) + (1/C^+)I \tag{12}$$

$$\boldsymbol{\Theta} = y_i^- y_j^-\varphi\left(x_i^-\right)^T\varphi\left(x_j^-\right) + (1/C^-)I \tag{13}$$

$$\boldsymbol{\Lambda} = y_i^+ y_j^-\varphi\left(x_i^+\right)^T\varphi\left(x_j^-\right) \tag{14}$$

$$\mathbf{Y}^+ = \left(y_1^+, y_2^+, \ldots, y_{N^+}^+\right)^T \tag{15}$$

$$\mathbf{Y}^- = \left(y_1^-, y_2^-, \ldots, y_{N^-}^-\right)^T \tag{16}$$

$$\boldsymbol{\alpha}^+ = \left(\alpha_1^+, \alpha_2^+, \ldots, \alpha_{N^+}^+\right)^T \tag{17}$$

$$\boldsymbol{\alpha}^- = \left(\alpha_1^-, \alpha_2^-, \ldots, \alpha_{N^-}^-\right)^T \tag{18}$$

$$\mathbf{1} = (1,1,\ldots,1)^T \tag{19}$$

where I is a unit matrix in Eqs. (12) and (13). From (12) and (13), the $\mathbf{\Omega}$ and $\mathbf{\Theta}$ are positive definite, and solution of Lagrangian multipliers $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ can be obtained from Eq. (11), i.e.

$$\boldsymbol{\alpha}^+ = \frac{(1 - b\mathbf{Y}^+)\mathbf{\Lambda} - (1 - b\mathbf{Y}^-)\mathbf{\Omega}}{\mathbf{\Lambda}^2 - \mathbf{\Omega}\mathbf{\Theta}} \tag{20}$$

$$\boldsymbol{\alpha}^- = \frac{(1 - b\mathbf{Y}^-)\mathbf{\Lambda} - (1 - b\mathbf{Y}^+)\mathbf{\Theta}}{\mathbf{\Lambda}^2 - \mathbf{\Omega}\mathbf{\Theta}} \tag{21}$$

Substituting (20) and (21) into the third matrix equation of (10), we can obtain value of b, as shown in (22).

$$b = \frac{\mathbf{\Lambda}[(\mathbf{Y}^+)^T + (\mathbf{Y}^-)^T] - \mathbf{\Omega}(\mathbf{Y}^+)^T - \mathbf{\Theta}(\mathbf{Y}^-)^T}{\mathbf{\Lambda}[(\mathbf{Y}^+)(\mathbf{Y}^+)^T + (\mathbf{Y}^-)(\mathbf{Y}^-)^T] - \mathbf{\Omega}(\mathbf{Y}^-)(\mathbf{Y}^+)^T - \mathbf{\Theta}(\mathbf{Y}^+)(\mathbf{Y}^-)^T} \tag{22}$$

Substituting (22) into (20) and (21), Lagrangian multipliers $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$ can be obtained. Accordingly, the solution of w can be obtained from the first equation in (9). Using w and b, the separating hyperplane in (1) can easily be determined.

The main advantages of the AW-LSSVM can be summarized as follows. First of all, it requires fewer prior assumptions about the input data than are required in statistical approaches, such as normal distribution and continuity, as similar to standard SVM [30]. Second, it can perform nonlinear mapping from an original input space into a high dimensional feature space, in which it constructs a linear discriminant function to replace the nonlinear function in the original input space. This characteristic also solves the dimension disaster problem because its computational complexity is not dependent on the sample's dimension. Third, it attempts to learn the separating hyperplane to maximize the classification margin, thereby implementing structural risk minimization and realizing good generalizability. Fourth, a distinct trait of LSSVM is that it can further lower computational complexity by transforming a quadratic programming problem into a linear equation group problem. Thus the computational process is simplified immensely and computational costs might be reduced when large-scale problems are needed to be computed. Finally, an important advantage of AW-LSSVM is that it can deal with cases where different classes have different importance levels for specified classification tasks, thereby making the proposed AW-LSSVM more suitable for some real-world problems. These important characteristics also make AW-LSSVM preferable for many practical applications.

3. Formulation of EP-based LSSVM ensemble learning

In this section, a four-stage evolutionary programming (EP) based asymmetric weighted LSSVM ensemble learning methodology is proposed for software repository mining. The basic idea of EP-based AW-LSSVM ensemble originated from using all the valuable information hidden in multiple LSSVM classifiers, each of which can contribute to improvement of generalization. In the proposed EP-based AW-LSSVM ensemble learning methodology, the main task at the first stage is to preprocess the original dataset, including input feature selection, data normalization and data partition. In the second stage, multiple single AW-LSSVM classifiers are created for diversification. In the third stage, the created multiple single AW-LSSVM classifiers are used for training and learning to produce the classification results. In the final stage, classification results of single AW-LSSVM classifiers are combined into an aggregated output in terms of evolutionary programming (EP) algorithm. The general architecture of the multistage EP-based AW-LSSVM ensemble learning methodology is illustrated in Fig. 1.

3.1. Data preprocessing

In the modeling of this study, data preprocessing is performed. This procedure is composed of three steps: input feature selection, data normalization and data partition, elaborated as follows.

(1) *Input feature selection.* In order to make sure that only relevant variables are included in the datasets, we adopt an input feature selection procedure using a χ^2-based filter approach [29]. In this approach, the observed frequencies of all possible combinations of values for the class and variables are first measured. Accordingly, theoretical frequencies are computed, assuming complete independence between the variables and the class. Then the hypothesis of equal odds provides a χ^2 test statistic; higher value allows one to more confidently reject the zero hypothesis of equal odds. Thus these values allow one to rank the variables in terms of predictability [29].

(2) *Data normalization.* All input variables are linearly scaled to [0,1], as in (23), to avoid dominance of attributes with greater numeric values over those with smaller values. Note that in (23), $i = 1,2,\ldots,N$ is the number of training samples and $k = 1,2,\ldots,m$ denote the number of input feature variables

$$x'_{ik} = \frac{x_{ik} - \min_i\{x_{ik}\}}{\max_i\{x_{ik}\} - \min_i\{x_{ik}\}}, \quad i = 1,2,\ldots,N;\ k = 1,2,\ldots,m \tag{23}$$

(3) *Data division.* The sample data is randomly divided into two parts, training set and testing set, which are used for model training and model testing purposes, respectively.

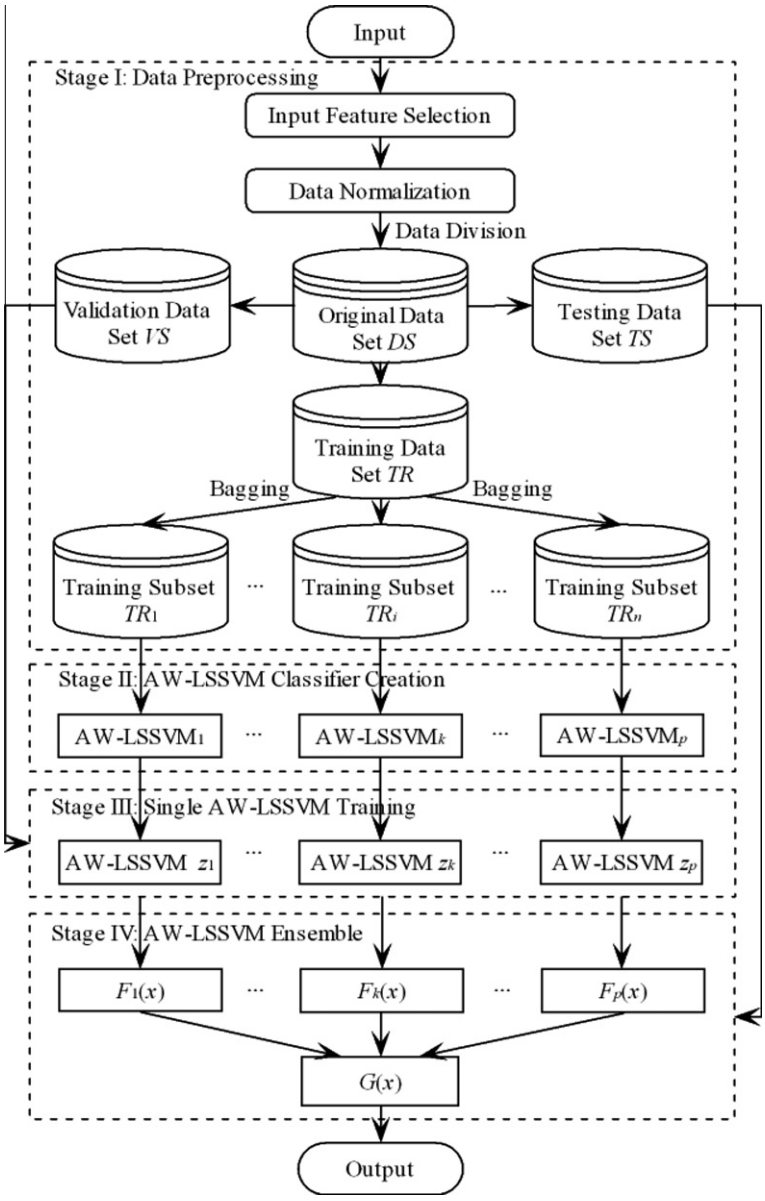

Fig. 1. General process of EP-based AW-LSSVM ensemble learning methodology.

3.2. Creating diverse AW-LSSVM classifiers

According to the definition of effective ensemble classifiers by Hansen and Salamon [9], 'a necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are accurate and diverse.' Generally, an effective ensemble classifier consisting of diverse models with much disagreement is more likely to have a good generalization performance in terms of the principle of bias-variance trade-off [36]. Therefore, generating diverse models is crucial for constructing an effective ensemble learning approach. For AW-LSSVM model, several methods have been investigated for generation of ensemble members making different errors. Such methods basically relied on varying the training samples and parameters related to the AW-LSSVM design. In particular, some main diversity strategies can be divided into the following three aspects [38].

(1) *Different training datasets.* Because different data often contain different information, different data can generate diverse models with dissimilarities. Usually, in machine learning models, training dataset is used to construct a concrete model and thus different training datasets will be an important source of diversity if the training dataset at hand

is functionally or structurally divisible into some distinct training subsets. In order to achieve diverse models, six typical techniques [32] that can create diversity, such as bagging, noise injection, cross-validation, stacking, boosting and input decimation [32], are used to obtain diverse training datasets. In this paper, the bagging algorithm [38] is used to create diverse training datasets.

(2) *Different kernel functions.* In a AW-LSSVM model, the kernel function has an important effect on generalization performance of AW-LSSVM. Hence, using different kernel functions in AW-LSSVM models can also create diverse AW-LSSVM models. In the AW-LSSVM model, linear kernel function, polynomial kernel function, MLP kernel function and RBF kernel function are typical kernel functions [30] that can create different AW-LSSVM learners.

(3) *Different model parameters.* In a AW-LSSVM model, there are two classes of typical parameters: penalty parameter (C^+ and C^-) and kernel parameters. By changing the AW-LSSVM model's parameters, different AW-LSSVM models with high levels of disagreement can be produced.

Using the above strategies, different AW-LSSVM classifiers can be obtained. In this paper, the bagging algorithm is used to generate different training subsets to create diverse AW-LSSVM classifiers. In addition, the AW-LSSVM model is selected as the generic classifier in this paper due to its distinct advantages. With these different training datasets, diverse AW-LSSVM classifiers are produced.

3.3. Single AW-LSSVM training

After determining diverse AW-LSSVM models, the next step is to train the different AW-LSSVM models to produce different output results, using different training datasets. In this study, the proposed AW-LSSVM model, described in the previous section, is used as a generic classifier.

The generic idea of AW-LSSVM is to maximize the margin hyperplane in the feature space and to consider the category importance simultaneously. Similar to other supervised learning methods, an underlying theme of the AW-LSSVM is to learn from data, which adopts the structural risk minimization (SRM) principle from computational learning theory [30]. Usually, AW-LSSVM can be used for regression and classification tasks. In this paper, we focus on the classification problem. In mathematical form, learning result of the single AW-LSSVM classifier $F(x)$ can be represented as

$$F(x) = sign(z(x)) = sign(w^T \cdot \varphi(x) + b) \tag{24}$$

where *sign* is a sign function and other symbols are identical to Eq. (1).

3.4. Ensemble learning of multiple AW-LSSVM classifiers

When multiple diverse classifiers are produced, the subsequent task is to combine results of different classifiers into an aggregated output in an appropriate ensemble learning strategy. Although the single AW-LSSVM model may have good performance, it is sensitive to data samples and parameter settings, i.e. each single AW-LSSVM model may have some biases when different data samples and different parameters are used. One effective way to reduce the bias is to integrate these AW-LSSVM classifiers into an ensemble output for final results. In previous studies, ensemble learning technique has been found to be an efficient technique for achieving high classification performance, especially when the development of a powerful single classification model requires considerable efforts.

However, before combining these AW-LSSVM classifiers, some strategies for selecting AW-LSSVM classifiers must be noted. Generally, these strategies can be divided into two categories: (i) producing an exact number of AW-LSSVM classifiers; and (ii) overproducing AW-LSSVM classifiers and then selecting a subset of these overproduced AW-LSSVM classifiers [32,38].

In the first strategy, several common ensemble approaches, e.g., boosting [24], can be employed to generate the exact number of diverse classifiers for ensemble purpose. That is, no selection process is used in this strategy, and all classifiers produced are combined into an aggregated output. In the second strategy, the main aim is to create a large number of classifier candidates and then choose some most diverse classifiers for ensemble purpose. The selection criterion includes some error measures, such as mean squares errors (MSE) and modified MSE, which were introduced in detail by Yu et al. [37,38]. Because the first strategy is based upon the idea of creating diverse LSSVM classifiers at the early stage of ensemble design, it is more effective than the second strategy when availability of some powerful computational resources is limited. The main reason why the second strategy is not preferred is that it cannot avoid consuming large computational time and storage space when creating a large number of classifier candidates, some of which are to be later discarded [32,38].

Generally, there are many ensemble approaches to combine different classification results in existing literature. Typically, majority voting and weighted averaging are two popular ensemble approaches. The main objective of majority voting strategy is to determine the vote of the majority of the population of classifiers. To the best of our knowledge, majority voting is one of the most widely used ensemble approaches for classification problems because it is easy-to-implement. In majority voting, each classifier has the same weight and voting by ensemble members determines the final result. Usually, it takes over half of ensemble members to agree in order for a result to be accepted as the final output of the ensemble, regardless

of diversity and accuracy of each classifier's generalizability. In mathematical form, the result of the final ensemble output $G(x)$ can be represented as

$$G(x) = sign\left(\sum_{k=1}^{p} F_k(x)/p\right) \tag{25}$$

where $F_k(x)$ is the output result of the kth classifier presented in Eq. (24) and p is the number of classifiers. Although this approach is easy to be used and implemented, there are several serious problems associated with majority voting. First of all, it ignores the fact that some classifiers that lie in a minority sometimes do produce the correct results. Second, if too many inefficient and uncorrelated classifiers are considered, the vote of the majority may lead to worse results than the ones obtained by using a single classifier. Third, it does not consider the difference in expected performance when they are employed in particular circumstances, such as plausibility of outliers. At the stage of combination, it ignores the existence of diversity, which is the purpose of the ensemble [32,37,38].

Weighted averaging is where the final output result is calculated in terms of a single classifier's performance, and a weight is attached to each single classifier's output. The sum of the total weight is one and each classifier is entitled to a portion of this total weight, based on its performance. A typical binary classification example is to use validation examples to determine the ensemble weight. If *FP* represents the number of observed negative class instances misclassified as positive by a classifier, *TP* denotes the number of correctly classified positive instances that belong to positive class; *FN* represents the number of observed positive instances mistakenly classified as negative, while *TN* represents the number of correctly classified negative class instances that belong to the negative class. Some common measures used to evaluate the performance of the classifier are defined as follows:

$$\text{Type I Accuracy} = \text{Specificity} = \frac{TN}{FP + TN} \tag{26}$$

$$\text{Type II Accuracy} = \text{Sensitivity} = \frac{TP}{TP + FN} \tag{27}$$

$$\text{Total Accuracy (TA)} = \frac{TP + TN}{TP + FP + TN + FN} \tag{28}$$

Each AW-LSSVM classifier is trained by a training dataset and is verified by the validation samples. If the total accuracy of validation samples of classifier k is denoted by TA_k, weight of LSSVM classifier k, denoted by w_k, can be calculated as

$$w_k = \frac{TA_k}{\sum_{k=1}^{p} TA_k} \tag{29}$$

Then the final ensemble output value $G(x)$ of this strategy is

$$G(x) = sign\left(\sum_{k=1}^{p} w_k \cdot F_k(x)\right) \tag{30}$$

where w_k is the kth weight for kth LSSVM classifier defined in Eq. (29); other symbols are identical to Eq. (25).

The above *TA*-based weighted averaging ensemble learning approach is a typical class of ensemble methods; determination of the weight is heavily dependent on validation samples. When there is no validation subset in data division, this ensemble approach is useless. In addition, this method cannot be used when the classes are continuous. For this reason, an additive method that permits continuous aggregation of classifications should be preferred. In this paper, we propose an evolutionary programming (EP) based approach [4] to maximize classification accuracy.

Suppose there are p classifiers and let c_{ij} be the classification results that classifier j,$j = 1,2,\ldots,p$ makes of sample i, $i = 1,2,\ldots,N$. Without loss of generality, we assume there are only two classes (e.g., fault-prone and non-fault-prone modules in the case of software repository mining) in data samples, i.e. $c_{ij} \in \{0,1\}$ for all i,j. Let $C_i^w = sign\left(\sum_{j=1}^{p} w_j c_{ij} - \theta\right)$ be the ensemble classification of data sample i, where w_j is the weight assigned to classifier j, θ is a confidence threshold and $sign(\cdot)$ is a sign function. For software fault classification problem, the software testing engineer can adjust confidence threshold θ to change the final classification results. Only when the ensemble output is larger than the threshold, the corresponding module can be classified as a non-fault module. Let $A_i(w)$ be the associated accuracy of classification:

$$A_i(w) = \begin{cases} a_1 & \text{if } C_i^w = 0 \text{ and } C_i^s = 0 \\ a_2 & \text{if } C_i^w = 1 \text{ and } C_i^s = 1 \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

where C_i^w is the classification result of the ensemble classifier, C_i^s is the actual observed class of data sample itself, and a_1 and a_2 are Type I and Type II accuracy, respectively.

The current problem is how to formulate an optimal combination of classifiers for ensemble classification. A natural idea is to find the optimal combination of weights $w^* = \left(w_1^*, w_2^*, \ldots, w_p^*\right)$ by maximizing total classification accuracy, including Types I and II accuracy. Usually, classification accuracy can be estimated through k-fold cross-validation (CV) technique.

Using the principle of total classification accuracy maximization, the above problem can be summarized as an optimization problem:

$$\begin{cases} \max_{w} A(w) = \sum_{i=1}^{M} A_i(w) \\ \text{s.t. } C_i^w = sign\left(\sum_{j=1}^{p} w_j c_{ij} - \theta\right), \quad i = 1, 2, \ldots M \\ A_i(w) = \begin{cases} a_1 & \text{if } C_i^w = 0 \text{ and } C_i^s = 0 \\ a_2 & \text{if } C_i^w = 1 \text{ and } C_i^s = 1 \\ 0 & \text{otherwise} \end{cases} \end{cases} \tag{32}$$

where M is the size of cross-validation set and other symbols are similar to the above notations.

Since constraint C_i^w is a nonlinear threshold function and $A_i(w)$ is a step function, optimization methods assuming differentiability of the objective function may have some problems. Therefore, the above problem cannot be solved with classical optimization methods. For this reason, an evolutionary programming (EP) algorithm [4] is proposed to solve the optimization problem indicated in (32) because EP is a useful method of optimization when other techniques such as gradient descent or direct analytical discovery are not possible. For the above problem, the EP algorithm [16,37] is described as follows:

(1) Create an initial set of L solution vectors $w_r = (w_{r1}, w_{r2}, \ldots, w_{rp})$, $r = 1, 2, \ldots, L$ for the above optimization problems by randomly sampling interval $[x, y]$, $x, y \in R$. Each population or individual w_r can be seen as a trial solution.
(2) Evaluate the objective function of each of the vectors $A(w_r)$. Here $A(w_r)$ is called the fitness of w_r.
(3) Add a multivariate Gaussian vector $\Delta_r = N(0, G(A(w_r)))$ to vector w_r to obtain $w_r' = w_r + \Delta_r$, where G is an appropriate monotone function. Re-evaluate $A(w_r')$. Here $G(A(w_r))$ is called mutation rate and w_r' is called an offspring of individual w_r.
(4) Define $\bar{w}_i = w_i,\ \bar{w}_{i+L} = w_i',\ i = 1, 2, \ldots, L,\ \overline{C} = \bar{w}_i,\ i = 1, 2, \ldots, 2L$. For every $\bar{w}_j,\ j = 1, 2, \ldots, 2L$, choose q vectors $\bar{w}_*$ from $\overline{C}$ at random. If $A(\bar{w}_j) > A(\bar{w}_*)$, assign $\bar{w}_j$ as a "winner".
(5) Choose L individuals with a higher number of "winners" $w_i^*,\ i = 1, 2, \ldots, L$. If the stop criteria are not fulfilled, let $w_r = w_i^*,\ i = 1, 2, \ldots, L$, generation = generation + 1 and go to Step 2.

Using the above EP algorithm, an optimal combination, w^*, of classifiers that maximizes the total classification accuracy, is formulated. Accordingly, the final ensemble results are also produced by the EP algorithm in the form of Eq. (30).

3.5. Overall process of EP-based AW-LSSVM ensemble learning methodology

According to the previous descriptions, the overall process of the proposed EP-based AW-LSSVM ensemble learning methodology can be summarized into the following four stages.

Stage I: *Data preprocessing.* In this stage, three steps: input feature selection, data normalization and data division, are included, as described in Section 3.1.
Stage II: *Diverse AW-LSSVM classifier creation.* To construct an effective ensemble learning system, multiple diverse AW-LSSVM classifiers with high levels of disagreement are used as classifiers for modeling purpose.
Step III: *Single AW-LSSVM classifier training.* When a single AW-LSSVM classifier is produced, these classifiers can be trained with a training subset obtained from Stage I. Accordingly, single classification results can be obtained.
Step IV: *Multi-classifier ensemble learning.* After generalization results of individual classifiers are produced, an additive EP-based ensemble learning approach is used to fuse outputs of individual classifiers into an aggregated output.

In order to verify the effectiveness of the proposed EP-based AW-LSSVM ensemble learning methodology, two publicly available software datasets are used as testing targets for empirical analysis in the following section.

4. Experimental results

In this section, two publicly available software datasets from NASA software projects [15] are used to test the performance of the proposed AW-LSSVM ensemble learning methodology. For comparison purposes, six individual classification models (logistic regression (LogR), k-nearest neighbor (KNN), C4.5, standard support vector machine (SVM), standard least squares SVM (LSSVM) and asymmetric weighted LSSVM (AW-LSSVM)) and two main ensemble classification models (majority voting and *TA*-based weighted averaging) are also used in the experiments. In particular, LSSVM ensemble learning is also performed to compare the advantages of the proposed AW-LSSVM ensemble learning. In these commonly used state-of-the-art classification techniques, the k-nearest neighbor (KNN) model classifies a data instance by only considering the k most similar data points in the training dataset. In this study, k is set to one, i.e. $k = 1$. C4.5 is a popular decision tree builder [23] where each leaf assigns a class label to observations. Each of these leaves can be represented by a rule, and standard pruning factors are used, i.e. a confidence factor of 0.25. In the standard SVM [30] model, the Gaussian kernel function is used and all

other parameters, including penalty parameter C and kernel parameters, are determined by the grid search method [37]. Similar settings are used in the standard LSSVM classifier. In the asymmetric weighted LSSVM classifier with Gaussian kernel function, as costs associated with misclassification of a faulty software module are much higher than the costs associated with incorrect classification of a non-faulty software module, the penalty on misclassification of faulty software modules should clearly be larger than the penalty on misclassification of non-faulty software modules. In this study, penalty parameters of faulty modules and non-faulty modules are set to 10,000 and 100, respectively. In addition, kernel parameter σ^2 is set to 5.

In ensemble learning methods, majority voting, weighted averaging and evolutionary programming (EP) based ensemble approaches are performed by using Eqs. (25), (30) and (32). In particular, 30 different training subsets are produced by using the bagging algorithm [37,38]. In this way, 30 diverse ensemble members are produced. The basic settings of the ensemble members are similar to the single LSSVM and AW-LSSVM models, as previously mentioned.

In software mining, the classification task is to distinguish between erroneous and non-faulty software modules. For this purpose, classification accuracy in the testing set (out-of-sample) is used as performance evaluation criterion. Typically, three commonly used evaluation criteria: Type I accuracy (Specificity), Type II accuracy (Sensitivity) and Total accuracy, defined in Eqs. (26)–(28), are used to measure classification performance.

In order to rank all models, the receiver operating characteristic (ROC) curve [6] and the area under the ROC curve (AUC) [5] are used as another two performance measures. The ROC graph is a useful technique for ranking models and visualizing their performance. Usually, ROC is a two-dimensional graph in which *sensitivity* is plotted on the *Y*-axis and 1-*specificity* is plotted on the *X*-axis, as illustrated in Fig. 2.

Fig. 2 shows ROC curves of two different models, labeled *A* and *B*. To perform the model ranking task, a common method is to calculate the area under the ROC curve, abbreviated as AUC [5]. Since the AUC is a portion of the area of the unit square, its value is always between 0 and 1. Fig. 2 shows AUCs of two different models with different fillings. Particularly, AUC of Model *A* is the area of the skewed line, while AUC of Model *B* is the area of the shaded part. Generally, a model with a large AUC will have a good average performance. For example, in this figure, AUC of Model *A* is larger than that of Model *B*. Thus, the performance of Model *A* is better than that of Model *B*. However, it is possible for a large AUC model to perform worse than a small AUC model in a specific region of ROC space. Fig. 2 illustrates an example of this. Model *A* is generally better than Model *B*, except at (1-specificity) > 0.7, where Model *B* has a slight advantage. But AUC can well describe the general behavior of the classification model because it is independent of any cutoff or misclassification costs used for obtaining a class label. Due to this characteristic, it is widely used for performance comparison in practice [35]. More details about ROC curve and AUC can be found in [5,6].

4.1. Dataset I: KC1

Experimental dataset KC1, in this subsection, is a subsystem of a large ground control system in NASA projects. KC1 contains 43,000 lines of codes (LOC) written in C++. After deleting data with missing attribute values, we obtained 1571 modules, of which 319 cases were faulty modules and 1252 cases were non-faulty modules.

In terms of the overall process of the proposed LSSVM ensemble learning methodology, we first used a χ^2-based filter approach [29] to perform input feature selection; 10 metrics were retained from 25 metrics (Table 1). Note that the description of some metrics can be referred to the NASA MDP website (http://mdp.ivv.nasa.gov/mdp_glossary.html). Then all data

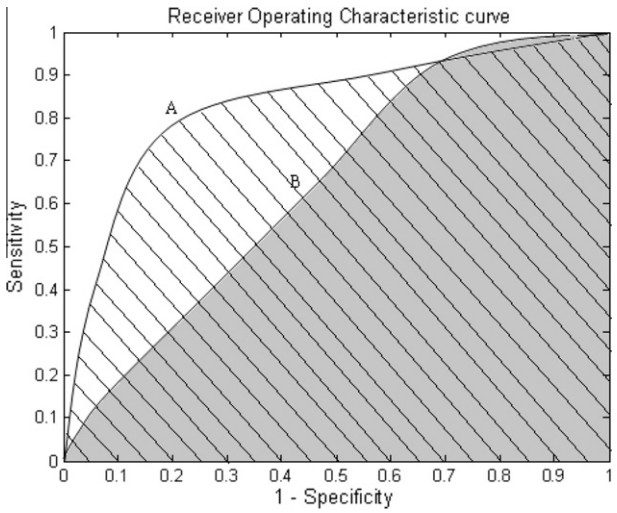

Fig. 2. ROC curve and AUC for two different models.

were normalized into the interval [0,1]. In the data division step, we made a 70%/30% stratified split up of the original dataset into training and test datasets. In addition, one third of the training datasets are set apart for validation purposes.

According to the previous experimental design and model settings, the final classification results are presented in Table 2. Note that classification results of four single classification models and majority voting based ensemble learning method are obtained from Table 6 of Vandecruys et al. [29].

From Table 2, several important findings can be inferred.

(1) In terms of the total accuracy and AUC criteria, the proposed EP-based AW-LSSVM ensemble learning methodology performs the best, followed by the TA-based AW-LSSVM ensemble learning model and the EP-based LSSVM ensemble learning model; the individual KNN model is the worst, indicating that the proposed EP-based AW-LSSVM ensemble learning methodology has a good generalization capability in software defects detection. This phenomenon is explained by the following four reasons. First of all, aggregating multiple diverse classifiers into a group consensus can remedy the shortcomings of any individual classifier, thus increasing classification reliability. Secondly, the proposed EP-based AW-LSSVM ensemble learning methodology utilizes evolutionary programming (EP) algorithm to determine ensemble weights, using the principle of classification accuracy maximization. This EP algorithm may help improve the performance of ensemble learning effectively. Third, the asymmetric weights in AW-LSSVM model can increase classification performance of the single AW-LSSVM classifier, thus improving the ensemble classification performance greatly. Finally, aggregation of different results can reduce variance of generalization error and, therefore, produce a more robust result than the individual models.

(2) In terms of Type I accuracy and Type II accuracy, ensemble learning approaches do not provide the best performance. From Table 2, KNN is the best classifier in terms of Type I accuracy and C4.5 performs the best in terms of Type II accuracy. In addition, the volatility of Type I accuracy is much higher than that of Type II accuracy in terms of variance. The main reasons causing these phenomena are still unknown, which is worth further exploring in the future.

(3) Of the six single models, asymmetric weighted LSSVM model performs the best, followed by single LSSVM, C4.5, logistic regression and standard SVM; the KNN is the worst in terms of total accuracy and AUC criteria. Using two tailed *t*-test, we find that the differences between performance of AW-LSSVM and LSSVM (C4.5, logR and SVM) are insignificant at 5% significance. The reasons of this phenomenon are unknown, which is worth exploring further in the future. However, the difference between AW-LSSVM and KNN is significant at one percent significance. The main reasons are twofold. On the one hand, the KNN classifier is unsuitable for this dataset. On the other hand, the AW-LSSVM can obtain higher classification accuracy due to the use of different asymmetric weights in AW-LSSVM.

(4) In AW-LSSVM ensemble models, the proposed EP-based AW-LSSVM ensemble learning methodology and TA-based AW-LSSVM ensemble learning methodology consistently outperform the majority voting based AW-LSSVM ensemble learning methodology. Using two tailed *t*-test, it is easy to find that the difference between performance of EP-based AW-LSSVM and TA-based AW-LSSVM ensemble learning methodologies are insignificant at one percent significance from the viewpoint of total accuracy, while the difference between EP-based AW-LSSVM and majority-voting based AW-LSSVM ensemble learning methodologies is significant at 5% significance, from the perspective of Type I accuracy. This implies that the proposed EP-based AW-LSSVM ensemble model is a promising approach to software fault identification.

(5) In LSSVM ensemble models, the proposed EP-based LSSVM and TA-based LSSVM ensemble learning methodologies consistently outperform the majority voting based AW-LSSVM ensemble learning methodology. The main reason lies in that the weight-based ensemble learning model is superior to majority voting based ensemble learning model, as previous mentioned in Section 3. However, there is no significant difference among the three LSSVM ensemble learning models, according to statistical testing results of the three models.

(6) Comparing single models with ensemble models, it is not hard to find that the ensemble learning approach is generally better than the single learning models. The main reason is that integrating multiple diverse models can remedy the shortcomings of any individual methods, thereby increasing classification accuracy.

(7) Comparing LSSVM ensemble learning with AW-LSSVM ensemble learning method, it is easy to find that performance of the AW-LSSVM ensemble learning methodology is much better than that of the LSSVM ensemble learning methodology in terms of Type I accuracy. This demonstrates the effectiveness of AW-LSSVM classifiers, which is due to the reason that more weights are often assigned to the more important classes.

Table 1
The selected metrics after input feature selection in KC1 dataset.

No.	Metrics	No.	Metrics
1	LOC total	6	Halstead prog time
2	Halstead Error Est	7	Halstead effort
3	Halstead volume	8	Num operators
4	Halstead length	9	Num operands
5	Halstead difficult	10	Num unique operands

Table 2
Classification results on KC1 dataset with different methods.

Category	Model	Strategy	Type I (%)	Type II (%)	Total (%)	AUC
Single	LogR		18.75	94.95	79.45	0.7913
	KNN		75.00	51.06	55.93	0.5525
	C4.5		18.75	95.48	79.87	0.7914
	SVM		26.04	92.82	79.24	0.7889
	LSSVM		27.59	91.85	80.20	0.7976
	AW-LSSVM		41.67	90.67	80.68	0.8011
Ensemble	LSSVM	Voting	28.28	90.03	78.56	0.7803
	LSSVM	TA	29.32	91.64	80.85	0.8025
	LSSVM	EP	29.79	92.59	81.25	0.8107
	AW-LSSVM	Voting	32.29	91.20	79.19	0.7882
	AW-LSSVM	TA	43.75	90.93	81.32	0.8116
	AW-LSSVM	EP	46.88	93.33	83.86	0.8328

(8) Focusing on the three ensemble strategies, the EP strategy performs better than the TA-based strategy and majority voting strategy for both LSSVM ensemble learning and AW-LSSVM ensemble learning. This reveals that the proposed EP strategy is a feasible and promising way to combine multiple diverse classifiers into an aggregated classifier.

4.2. Dataset II: PC4

In this subsection, the used dataset are PC4, describing a flight project software written in C and used for an earth orbiting satellite, which can be found in NASA MDP repository [15]. It contains detailed information of 36,000 lines of codes (LOC). After deleting some data with missing attributes, we obtained 1347 modules, in which including 178 faulty and 1169 non-faulty modules. After removing some irrelevant variables using a χ^2-based filter approach [29], 13 metrics are retained from 42 metrics for classification purpose, which are described as follows:

(1) LOC total.
(2) LOC Comments.
(3) Halstead Error Est.
(4) Halstead volume.
(5) Halstead length.
(6) Halstead level.
(7) Num operands.
(8) Condition count.
(9) Modified condition count.
(10) Multiple condition count.
(11) Cyclomatic complexity.
(12) Normalized cyclomatic complex.
(13) Call pairs.

Descriptions of some metrics can be referred to the NASA MDP website (http://mdp.ivv.nasa.gov/mdp_glossary.html). All metrics are scaled into the interval [0,1]. In the data division process, we made a 70%/30% stratified split up of the original dataset into training and testing datasets. In addition, one third of the training datasets are set aside for validation purpose.

Table 3
Classification results on PC4 dataset with different methods.

Category	Model	Strategy	Type I (%)	Type II (%)	Total (%)	AUC
Single	LogR		32.08	97.44	88.86	0.8812
	KNN		49.06	89.74	84.65	0.8429
	C4.5		35.85	94.02	86.39	0.8598
	SVM		16.98	98.86	88.15	0.8785
	LSSVM		25.32	94.78	88.66	0.8823
	AW-LSSVM		47.17	95.44	89.11	0.8885
Ensemble	LSSVM	Voting	29.27	94.21	88.35	0.8813
	LSSVM	TA	30.35	95.76	90.09	0.8989
	LSSVM	EP	36.74	96.01	90.21	0.8991
	AW-LSSVM	Voting	45.28	95.16	88.61	0.8819
	AW-LSSVM	TA	52.83	96.01	90.35	0.8994
	AW-LSSVM	EP	54.72	96.87	91.34	0.9101

In the ensemble learning methodology, the bagging algorithm [37,38] is used to produce 30 different training subsets; other settings are similar to previous descriptions in the first dataset. According to the previous experiment design, the final computational results are shown in Table 3.

As similar to Table 2, some similar conclusions can be obtained from Table 3. Particularly, this dataset again confirms that the proposed EP-based AW-LSSVM ensemble learning model is suitable for software defects detection, implying that it is a very promising solution for software repository mining. A visual explanation for performance comparison of different models is illustrated with ROC space in Fig. 3.

From Table 3 and Fig. 3, several important conclusions can be drawn, as summarized below.

First of all, the proposed EP-based AW-LSSVM ensemble learning model performs the best in terms of Type I accuracy, total accuracy and AUC, indicating that the additive EP-based AW-LSSVM ensemble learning models can consistently outperform other individual classification models in software fault identification, implying the strong capability of the multistage EP-based AW-LSSVM ensemble learning methodology in software repository mining. An interesting and exceptional phenomenon is that the performance of the single SVM classifier is the best among all classifiers, followed by the single LogR classifier, in terms of Type II accuracy. The reason is not known, and is worth exploring further in the future.

Second, among the three AW-LSSVM ensemble learning methodologies, the proposed EP-based AW-LSSVM ensemble learning model performs the best, followed by the TA-based AW-LSSVM ensemble learning model and the majority voting based ensemble learning model in terms of all four evaluation criteria. This finding is consistent with the results of previous experiments. However, through two-tail paired *t*-test, the average difference in performance of the three ensemble learning methodologies is insignificant at 10% significance. It is also obvious that performances of the three ensemble models are quite close in terms of the total accuracy. However, performances of EP-based AW-LSSVM ensemble learning and TA-based AW-LSSVM ensemble learning methods are significantly better than the majority voting based AW-LSSVM ensemble learning model at five percent level. Furthermore, of the three AW-LSSVM ensemble learning approaches, the majority voting based AW-LSSVM ensemble learning method is the worst. The main reason is that it ignores the existence of diversity, which is the motivation of the ensemble learning method [37]. Similar conclusions can be found in outcomes of application of the three LSSVM ensemble learning methodologies.

Third, comparing AW-LSSVM ensemble learning and LSSVM ensemble learning method, it is easy to find that performance of the AW-LSSVM ensemble learning methodology is much better than the LSSVM ensemble learning model. This result is consistent with the previous experimental example and further confirms the effectiveness of the asymmetric weights of LSSVM classifier.

Fourth, compared with the three different ensemble strategies between AW-LSSVM and LSSVM ensemble learning methods, two important conclusions can be found. On the one hand, the AW-LSSVM ensemble learning methodology performs better than the LSSVM ensemble learning methodology for the three different ensemble strategies consistently. The main

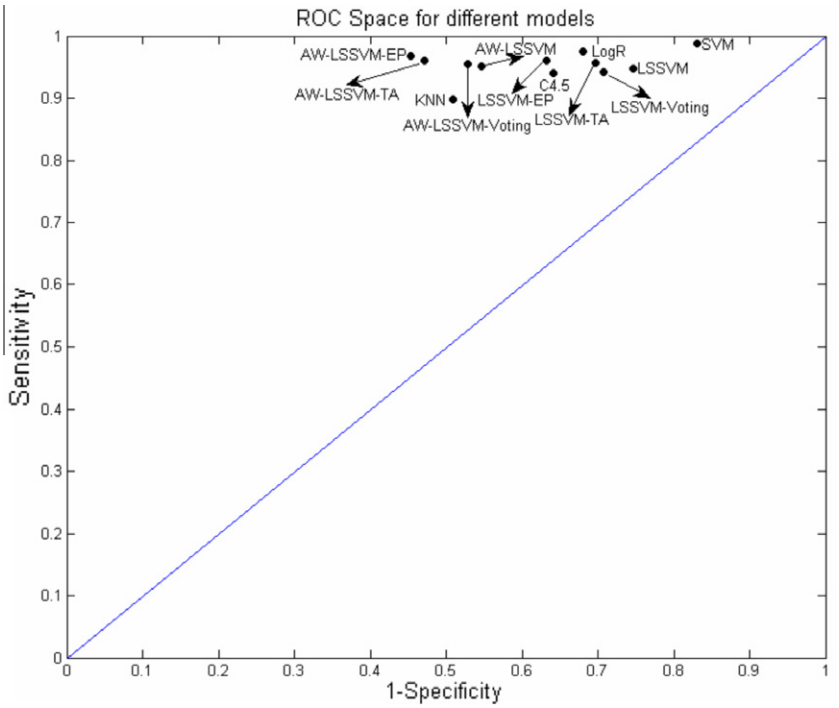

Fig. 3. A graphic performance comparison of different models in PC4 dataset.

reason is that the AW-LSSVM classifier can assign higher weights to important classes. On the other hand, performance of the EP-based ensemble strategy is better than the other two ensemble strategies. The main reason lies in its advantages of the EP-based ensemble learning strategy, as mentioned in Section 3.

Fifth, among the six single models, the AW-LSSVM model performs the best, followed by single logR, standard LSSVM, standard SVM and C4.5; KNN is the worst from the criterion of Type I accuracy, as well as total accuracy. This conclusion is slightly different from the previous experiment. The possible reason has two aspects. On the one hand, different methods may have different classification capabilities for different datasets. On the other hand, different datasets may have different classification properties. The two possible reasons lead to this interesting result.

Sixth, of the six single models, total accuracy of the AW-LSSVM model is better than that of other single models. The main reason for this is that the AW-LSSVM model utilizes the principle of structural risk minimization (SRM), thereby increasing the generalization capability. Furthermore, Type I accuracy of the proposed AW-LSSVM model is much better than that of the standard LSSVM, LogR, C4.5 and standard SVM. The main reason is that the AW-LSSVM model assigns higher penalties for fault-prone modules, increasing the classification capability of LSSVM. However, Type I accuracy of KNN model is the best among the five single models. The reason leading to this is unknown and confirmation of this phenomenon requires further research.

Finally, comparing ensemble classifiers and the single classifiers, it is not difficult to find that the performances of ensemble classifiers are generally better than single classifiers. The main reason is the advantages of ensemble classifiers, as mentioned in Sections 1 and 3.

4.3. Further discussion

The above two practical datasets verify the effectiveness of the proposed EP-based AW-LSSVM ensemble learning methodology. From total accuracy and AUC measurements, we can easily judge which model performs the best and which model performs the worst. However, there are some important issues about the proposed EP-based AW-LSSVM ensemble learning methodology, as are shown below.

Firstly, limitations of the proposed EP-based AW-LSSVM ensemble learning methodology need to be noted. On the one hand, the AW-LSSVM classifier requires that the number of positive samples be equal to the number of negative samples in the training phase, as Eq. (22) revealed. That is, the AW-LSSVM classifier is a class of the balanced learning machine. When the number of positive samples is seriously unequal to the number of negative samples, performance of the AW-LSSVM classifier is affected adversely.

Secondly, the proposed EP/TA-based AW-LSSVM/LSSVM ensemble learning methodologies are superior to majority voting based AW-LSSVM/LSSVM ensemble learning methods in terms of all evaluation criteria (Type I accuracy, Type II accuracy, total accuracy, AUC and ROC curve) in the previous two numerical experiments. This implies the effectiveness of weighted averaging for ensemble learning. However, performance improvements of the proposed EP-based AW-LSSVM/LSSVM ensemble learning methodology are only marginally smaller, relative to TA-based AW-LSSVM and LSSVM ensemble learning methods, in the previous two numerical experiments, suggesting that it may be possible to further improve the EP algorithm for ensemble weight determination.

Finally, due to introduction of the asymmetric penalty factor, Type I accuracy of ensemble learning methodologies and single AW-LSSVM model has been greatly increased. This is the expected result of introduction of the asymmetric penalty factor because the benefits associated with Type I accuracy are much higher than the benefits associated with Type II accuracy. However, this does not mean that Type II accuracy can be neglected. Actually, we can adjust the penalty factor to emphasize Type II accuracy. In the software fault classification task, a classification technique that classifies all software modules as erroneous might well result in high quality software, but the testing costs will be unbearably high. A trade-off has to be made in order to get a high Type I accuracy combined with a reasonable Type II accuracy. This allows software managers to efficiently allocate testing budgets to fault-prone software modules, thereby increasing the quality of delivered software systems.

5. Conclusions

Due to the huge accumulated and still growing mass of software repositories, software defects mining has attracted much research interest from both academic and industrial communities. A more accurate, consistent and robust software fault classification technique can significantly reduce testing costs for the software industry.

In this study, a multi-stage EP-based AW-LSSVM ensemble learning methodology is proposed for software fault classification problem. For the purposes of verification, two publicly available software repository datasets were used in order to test effectiveness and classification power. The empirical results reported in the experiments clearly show that the proposed EP-based AW-LSSVM ensemble learning approach can consistently outperform other comparable models, including the five ensemble learning models and the six single classification techniques. The obtained results reveal that the proposed EP-based AW-LSSVM ensemble learning model generally performs the best in terms of Type I accuracy and total accuracy, with a reasonable Type II accuracy, implying that the proposed EP-based AW-LSSVM ensemble learning model can be used as a feasible and promising solution to improve accuracy of software fault classification. In addition, as a promising methodology,

the proposed EP-based AW-LSSVM ensemble learning model can also be applied to other classification problems, which will be investigated in the future research.

Acknowledgements

The author would like to express their sincere appreciation to the Editor-in-Chief and the Guest Editor as well as the four independent referees for making valuable comments and suggestions. Their comments have helped improve the quality of the paper immensely. This work is partially supported by grants from the National Science Fund for Distinguished Young Scholars (NSFC No. 71025005), the National Natural Science Foundation of China (NSFC No. 90924024), the Knowledge Innovation Program of the Chinese Academy of Sciences and the Open Project of Hangzhou Key Laboratory of E-Business and Information Security, Hangzhou Normal University. The author gratefully acknowledges the support of K.C. Wong Education Foundation, Hong Kong.

References

[1] 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.
[2] C.P. Chang, C.P. Chu, Y.F. Yeh, Integrating in-process software defect prediction with association mining to discover defect pattern, Information and Software Technology 51 (2009) 375–384.
[3] S. Dick, A. Meeks, M. Last, H. Bunke, A. Kandel, Data mining in software metrics databases, Fuzzy Sets and Systems 145 (2003) 81–110.
[4] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, 2003.
[5] D. Faraggi, B. Reiser, Estimation of the area under the ROC curve, Statistics in Medicine 21 (2002) 3093–3106.
[6] T. Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (2006) 861–874.
[7] N. Fenton, M. Neil, A critique of software defect prediction models, IEEE Transactions on Software Engineering 25 (1999) 675–689.
[8] T. Graves, A. Karr, J. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (1999) 653–661.
[9] L.K. Hansen, P. Salamon, Neural network ensembles, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (1990) 993–1001.
[10] Y. Kanellopoulos, C. Makris, C. Tjortjis, An improved methodology on information distillation by mining program source code, Data and Knowledge Engineering 61 (2007) 359–383.
[11] T. Khoshgoftaar, N. Seliya, Analogy-based practical classification rules for software quality estimation, Empirical Software Engineering 8 (2003) 325–350.
[12] 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.
[13] S. MacDonell, M. Shepperd, Combining techniques to optimize effort predictions in software project management, Journal of Systems and Software 66 (2003) 91–98.
[14] T. Menzies, J. Di Stefano, C. Cunanan, R. Chapman, Mining repositories to assist in project planning and resource allocation, in: Proceedings of the International Workshop on Mining Software Repositories, 2004, pp. 75–79.
[15] NASA MDP repository, 2004. <http://mdp.ivv.nasa.gov/>.
[16] I. Olmeda, E. Fernandez, Hybrid classifiers for financial multicriteria decision making: the case of bankruptcy prediction, Computational Economics 10 (1997) 317–335.
[17] B.J. Park, S.K. Oh, W. Pedrycz, The design of polynomial function-based neural network predictors for detection of software defects, Information Sciences, in press, doi:10.1016/j.ins.2011.01.026.
[18] 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.
[19] Y. Peng, G. Kou, G. Wang, W. Wu, Y. Shi, Ensemble of software defect predictors: an AHP-based evaluation method, International Journal of Information Technology and Decision Making 10 (1) (2011) 187–206.
[20] Y. Peng, G. Kou, G.X. Wang, H.G. Wang, F.I.S. Ko, Empirical evaluation of classifiers for software risk management, International Journal of Information Technology and Decision Making 8 (4) (2009) 749–767.
[21] Y. Peng, G. Wang, H. Wang, User preferences based software defect detection algorithms selection using MCDM, Information Sciences 191 (2012) 3–13.
[22] T.S. Quah, Estimating software readiness using predictive models, Information Sciences 179 (2009) 430–445.
[23] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.
[24] R.E. Schapire, The strength of weak learnability, Machine Learning 5 (1990) 197–227.
[25] M. Shepperd, M. Cartwright, G. Kadoda, On building prediction systems for software engineers, Empirical Software Engineering 5 (2000) 175–182.
[26] Q. Song, S. Shepperd, C. Mair, Using grey relational analysis to predict software effort with small data sets, in: Proceedings of the 11th IEEE International Software Metrics Symposium, 2005, pp. 10–35.
[27] Q. Song, S. Shepperd, M. Cartwright, C. Mair, Software defect association mining and defect correction effort prediction, IEEE Transactions on Software Engineering 32 (2006) 69–82.
[28] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Processing Letters 9 (3) (1999) 293–300.
[29] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, R. Haesen, Mining software repositories for comprehensible software fault prediction models, Journal of Systems and Software 81 (2008) 823–839.
[30] V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
[31] C. Williams, J. Hollingsworth, Automatic mining of software repositories to improve bug finding techniques, IEEE Transactions on Software Engineering 31 (2005) 466–480.
[32] S. Yang, A. Browne, Neural network ensembles: combining multiple models for enhanced performance using a multistage approach, Expert Systems 21 (2004) 279–288.
[33] A. Ying, G. Murphy, R. Ng, M. Chu-Carroll, Predicting source code changes by mining change history, IEEE Transactions on Software Engineering 30 (2004) 574–586.
[34] L. Yu, S.Y. Wang, J. Cao, A modified least squares support vector machine classifier with application to credit risk analysis, International Journal of Information Technology and Decision Making 8 (4) (2009) 697–710.
[35] L. Yu, S.Y. Wang, K.K. Lai, An intelligent-agent-based fuzzy group decision making model for financial multicriteria decision support: the case of credit scoring, European Journal of Operational Research 195 (2009) 942–959.
[36] L. Yu, S.Y. Wang, K.K. Lai, Credit risk assessment with a multistage neural network ensemble learning approach, Expert Systems with Applications 34 (2008) 1434–1444.
[37] L. Yu, S.Y. Wang, K.K. Lai, L.G. Zhou, Bio-Inspired Credit Risk Analysis: Computational Intelligence with Support Vector Machines, Springer, Berlin, 2008.

[38] L. Yu, W.Y. Yue, S.Y. Wang, K.K. Lai, Support vector machine based multiagent ensemble learning for credit risk evaluation, Expert Systems with Applications 37 (2010) 1351–1360.
[39] X. Zhang, H. Pham, Software field failure rate prediction before software deployment, Journal of Systems and Software 79 (2006) 291–300.
[40] S. Zhong, T. Khoshgoftaar, S. Naeem, Analyzing software measurement data with clustering techniques, IEEE Intelligent Systems 19 (2004) 20–27.
[41] T. Zimmerman, P. Weissgerber, S. Diehl, A. Zeller, Mining version histories to guide software changes, IEEE Transactions on Software Engineering 31 (2005) 429–445.