Contents lists available at ScienceDirect

Chemometrics and Intelligent Laboratory Systems

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

The boosting: A new idea of building models

Dong-Sheng Cao [a], Qing-Song Xu [b], Yi-Zeng Liang [a,*], Liang-Xiao Zhang [a], Hong-Dong Li [a]

[a] Research Center of Modernization of Traditional Chinese Medicines, Central South University, Changsha 410083, PR China
[b] School of Mathematical Sciences and Computing Technology, Central South University, Changsha 410083, PR China

ARTICLE INFO

Article history:
Received 24 February 2009
Received in revised 3 September 2009
Accepted 17 September 2009
Available online 6 October 2009

Keywords:
AdaBoost
Gradient boosting (GB)
Classification and Regression Tree (CART)
Bagging
Ensemble learning

ABSTRACT

The idea of boosting deeply roots in our daily life practice, which constructs the general aspects of how to think about chemical problems and how to build chemical models. In mathematics, boosting is an iterative reweighting procedure by sequentially applying a base learner to reweighted versions of the training data whose current weights are modified based on how accurately the previous learners predict these samples. By using different loss criteria, boosting copes with not only classification problems but also regression problems. In this paper, the basic idea and algorithms of commonly used boosting are discussed in detail. The applications to two datasets are conducted to illustrate the significant performance of boosting.

1. Introduction

Boosting, based on an ensemble of individual models, is one of the most powerful learning methods introduced in the last ten years. It stems from the PAC (probably approximately correct) learning framework [1] or the concept of ensemble learning [2,3] and was originally designed for classification problems. Based on the PAC theory, Kearns and Valiant [4] were the first to pose the question of whether a "weak" learning algorithm that performs just slightly better than random guessing can be "boosted" into an arbitrarily accurate "strong" learning algorithm. Such a question forms the foundation of boosting. The underlying idea of boosting is a procedure that combines the outputs of many "weak" learners to produce a powerful "committee'. A weak learner is an algorithm whose error rate is only slightly better than random guessing. Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a (weighted) vote of their predictions [5–7]. Nevertheless, it should be noted that the ensemble obtained through boosting can reduce the variance and bias of a model simultaneously. The first provable polynomial-time boosting algorithm which suffered from certain practical drawbacks was developed in 1990 by Schapire [8] in the PAC learning framework. The AdaBoost algorithm, introduced in 1995 by Freund and Schapire [9], solved many of the practical difficulties of the earlier boosting algorithms. AdaBoost, as the most popular boosting procedure, has since drawn much attention [10–16]. In order to find out why AdaBoost, in most cases, performs well, Friedman et al. [17,18] analyzed AdaBoost statistically, derived the exponential criterion, and showed that it estimated the log-odds of the class probability. The connection between boosting and additive models not only helps to explain why boosting is an effective classification method but also opens the door to the application to many other statistical prediction problems. Moreover, there are actually several implementations of boosting, such as Friedman's gradient boosting [19] (GB), stochastic variants of boosting [20] (SGB), and many others [21–23]. By minimizing different loss functions, boosting can deal with not only the classification problems but also the regression problems. A significant feature of boosting is that it can accurately capture the nonlinear structure of data by adding a set of base learners adaptively. Much has been written about the success of boosting in producing more accurate prediction [24–26]. So, it was soon applied to many research fields [27–47]. All these development and applications of boosting make it a very popular approach.

Recently, boosting has also aroused chemist's concern because of its desired properties. It is so attractive that many chemists have focused on its basic studies and especially on its applications, e.g. bioinformatics [31–40], near infrared spectroscopy [48,49], quantitative structure–activity/property relationship [50–55] (QSAR/QSPR) study, protein structure/function prediction [56], mass spectrometry analysis [57]. The main idea of boosting rooted in our daily life practice may give us a brand-new thought process of how to think about chemical problems or how to build chemical models. More and more papers have been published to report the successful application of boosting in the chemical field. So it is necessary for chemists to have much in-depth discussion on the basic idea of boosting.

The present paper introduces the main idea of boosting algorithms and reports two applications in chemical research fields. In this paper, we intend to present a somewhat basic point of view illustrating the

* Corresponding author.
E-mail address: yizeng_liang@263.net (Y.-Z. Liang).

0169-7439/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.chemolab.2009.09.002

main idea of boosting algorithms rather than attempt a full treatment of all available algorithms. So, there is not much more theoretical derivation in this paper. We place more emphasis on the implementations of commonly used boosting algorithms such as AdaBoost, gradient boosting etc. This paper is organized as follows. We start by introducing the basic development situation of boosting, notation used in this paper and the basic idea of Classification and Regression Tree (CART) in the first three sections. Then, the concept of bagging is developed to give a comprehensive understanding and comparison with boosting in Section 4. The general idea of boosting is further introduced through our daily life experiences in Section 5. A simple nonlinear example is illustrated to interpret how boosting works in mathematics. Subsequently, some commonly used boosting methods such as discrete AdaBoost, gradient boosting and their implementations are introduced in detail for classification and regression problems, respectively. Some attention will be devoted to questions of model selection, *i.e.*, how to choose the parameters in the boosting approaches. We discuss some key properties of boosting in Section 6, which may be very helpful to our in-depth understanding of the boosting technique. Finally, two experimental results are provided to demonstrate the significant features and performance of boosting in Section 8.

2. Notation

Let us first introduce some notations used in this paper. In order to conveniently understand the below mentioned symbols, some chemical interpretations are assigned. The data matrix **X** has *N* observations in the rows and *p* variables in the columns. Each observation or sample represents one experiment, one molecule or one spectrum etc. Each variable or feature can be seen as some experiment factor or molecular feature. In addition, matrices and vectors are denoted by bold capital and lowercase characters, respectively, while scalar variables are shown in italic letters. In this paper, the terms, namely learner, classifier and base function, are used interchangeably.

3. Classification and Regression Tree

In this section, we briefly introduce the concept of Classification and Regression Tree (CART), because most weak learners used two-terminal node tree (stump). CART, proposed by Breiman et al., is a nonparametric statistical technique [58]. The goal of CART is to explain the response **y** by selecting some useful variables from a large pool of variables. The tree is generated in a recursive binary way, resulting in nodes connected by branches. A node, which is partitioned into two new nodes, is called a parent node. The new nodes are called child nodes. A terminal node is a node that has no child nodes. A CART procedure is generally made up of three steps. In the first step, the full tree is built using a binary split procedure. The full tree is an overgrown model, which closely describes the training set and generally shows overfitting. In the second step, the overfitted model is pruned. This procedure generates a series of less-complex trees, derived from the full tree. In the third step, the optimal tree is chosen using a cross-validation procedure. Herein, stump is a two-terminal node tree which divides the data into two classes using some optimal variable (a single axis-oriented split producing the best performance). A simple example using CART to solve a binary classification problem will be discussed in some detail in Section 5.1.

4. Bagging

Two of the popular techniques for constructing ensembles are bootstrap aggregation (bagging) and the AdaBoost family of algorithms (boosting). Both of these methods create an ensemble of classifiers by resampling the data, which are then combined by majority voting, although they have substantial differences. So it is necessary to dwell on how the bagging technique works before we discuss the boosting technique which goes a step further.

Bagging [59] is a "bootstrap" ensemble method that creates individuals for its ensemble by training each classifier on a random redistribution of the training set. So, it incorporates the benefits of both bootstrap and aggregating approaches. Fig. 1 shows the scheme of the bagging procedure. It can clearly be seen from Fig. 1 that many bootstrap samples firstly are generated from the original training set. Bootstrap [60], is based on random sampling with replacement. Consider firstly the regression problem, suppose we want to fit a model to our training data **Z**, obtaining the prediction $f(\mathbf{x})$ at input **x**. Firstly, *N* samples are generated by randomly drawing with replacement, where $\mathbf{Z}^* = \{(\mathbf{x}_1^*, y_1^*), (\mathbf{x}_2^*, y_2^*), \ldots, (\mathbf{x}_N^*, y_N^*)$. It is worth noting that many of the original samples may be repeated in the resulting training set while others may be left out (approximately $1/e \approx 37\%$ of all the samples are not presented in the bootstrap sample). The main idea of the aggregating approach actually means combining multiple models. For each bootstrap sets $\mathbf{Z}^{*t}$, $t = 1, 2, \ldots, T$,

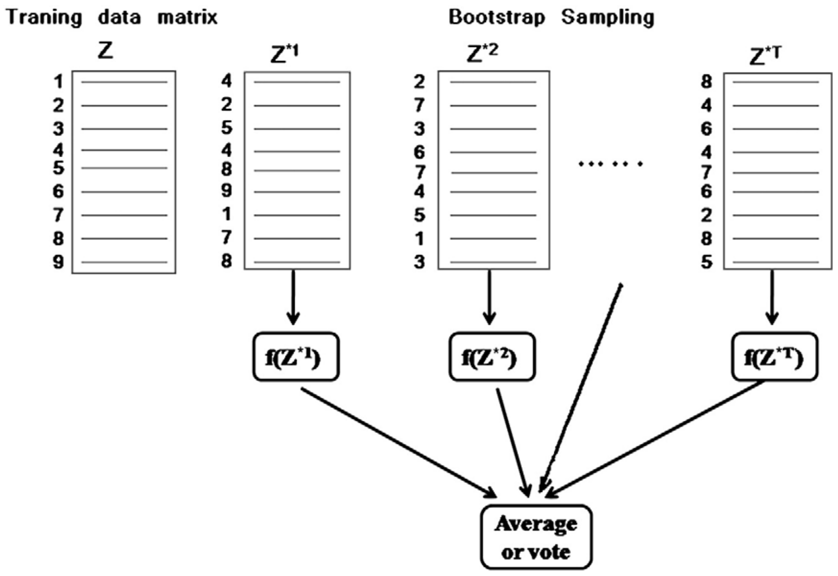

Fig. 1. The schematic illustration of the bagging procedure. *T*: the number of iterations. Average *T* predicted values for regression; Cast a Vote of *T* predicted values for classification.

one model is fitted, giving prediction $f(\mathbf{x})$. The estimation of bagging, averaging the predictions over a collection of bootstrap observations, is given by:

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} \hat{f}^{*t}(\mathbf{x}) \tag{1}$$

It is noticeable that combining models never makes the root mean square of residuals worse, at the population level. Bagging can be effective, especially, according to Breiman, for "unstable" learning algorithms for which a small change in the datasets results in a large change in the computed results [59]. Bagging can dramatically reduce the variance of unstable procedures like decision trees and neural networks, leading to an improved prediction, because averaging reduces variance and leaves bias unchanged.

A simple regression example with one predictor was constructed to illustrate how bagging captures the true regression function. Herein, $y_i = 2 + 4x_i + \varepsilon_i$, $i = 1, \ldots, n$. where ε_i is a noise term distributed according to a Gaussian distribution of zero mean and unit variance and each independent x_i is uniformly distributed in [0,1]. All x_i's are independent from all ε_i's. The true target function $F_{\text{true}}(x_i) = 2 + 4\ x_i$, $x_i \in [0, 1]$. We use stumps (a two-terminal node regression tree) as base learners and average the predicted results of 50 iterations. For this simple example, the bagging stumps estimate is displayed in Fig. 2A. From Fig. 2A, one can see that the estimate from bagging is still a rather crude and erratic approximation of the target function. Compared with the performance of bagging, Fig. 2B gives the fitting results of boosting stumps which we will introduce in detail soon. It is noticeable from the plots that the boosting method is more suitable to model the true target function.

5. Boosting

5.1. The boosting: general idea

As a matter of fact, the general idea of boosting has an intrinsic connection to our daily life experiences. We use it all the time! Generally speaking, we hope to improve our confidence that we make a correct decision in our daily life. In fact, making a highly correct decision is certainly a difficult task. How can we address this question in face of complex problems? We always seek for additional opinions before making a final decision, subsequently weigh various opinions and combine these opinions through some thought process to reach a final decision [61]. In all of opinions, some of them may give a high attention in that they come from some experts' decisions. We often trust the decisions of the experts much more, who can provide us with the more informative opinions to help us to make correct decisions. However, not all experts can give a correct decision. How can we reduce the risk of incorporating wrong expert decisions/opinions in our own decision making? In most of cases, combining various opinions by some certain thought process may reduce the risk of an individual opinion which may bring an unfortunate selection. The averaging or weighing combination may or may not exceed the performance of the best opinion in the set of opinions, but it can undoubtedly reduce the overall risk of making a final decision. Furthermore, complex problems are just too difficult for us to solve one time in our daily life. That is to say, we are hard to obtain the best answer directly when facing with a complex problem (the best answer is even unknown to us). In general, we have an intuitive idea: we usually take stepwise manner and firstly look for and deal with some easy-to-solve subproblems and subsequently figure out the hard-to-solve ones. Boosting just adopts these strategies to achieve a more accurate decision. These simple strategies result in dramatic improvements in performance for most of the problems.

As discussed above, boosting is a general and effective method for producing an accurate prediction rule by combining rough and moderately rough rules of thumb. Given a classification problem, the goal, of course, is to generate a rule that makes the most accurate predictions possible on new test samples. To apply boosting, there are two fundamental questions that must be solved: first, how should each weak classifier be chosen? Second, once we have collected many weak classifiers, how should the weak classifiers be combined into a single one? Subsequently, we will show how boosting solves the above questions mathematically. In order to elucidate how the boosting technique works briefly, let's consider a binary classification problem in 2-D space. The data were class labeled with two classes: +1 (circle) and −1 (triangle). The classes cannot be linearly separated (see Fig. 3). We choose the split along either x_1 or x_2 that produces the smallest training misclassification error as our classifier stump. To compare the difference between boosting with stumps and CART, Fig. 4A shows the results of using CART to classify the simple example. We can clearly see from Fig. 4A that CART adopts recursive binary partitions to classify all the training data. Firstly, the space is split into two regions via some split point achieving the best performance. Each split can be considered as a stump. Then, one or both of these regions are split into two more regions, and this process is continued until some stopping rule is achieved. For the above example, a correct classification is achieved by three splits. It can be seen from this example that CART makes use of many stumps without weights to recursively classify the training data. However, this is different from boosting which combines many weighted stump-predictions to reach the final aggregate prediction. Fig. 4B shows the overall process of the

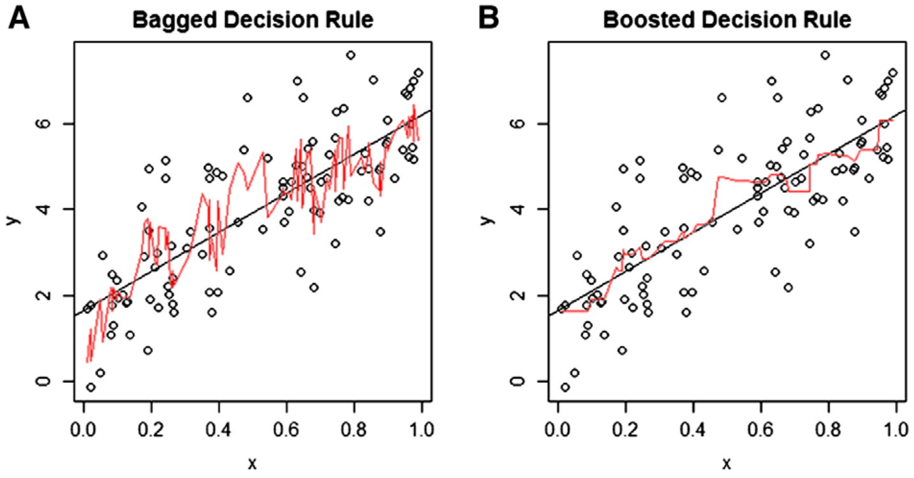

Fig. 2. A simple regression example with one predictor. Base learner: stump . Left panel (A): decision function estimated from bagging. Right panel (B): decision function estimated from boosting. The target function is indicated in black line. Boosting and bagging are done with50 iterations.

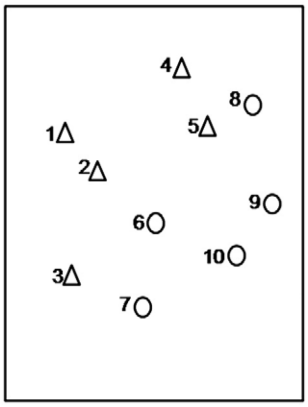

Fig. 3. The simulated data for a two-class problem marked by +1(circle) and −1(triangle).

AdaBoost algorithm. Initially all of the weights were set to $\omega_{1i}=1/10$ ($i=1, 2, \ldots, 10$), so that the first step simply trains the classifier in the usual way. The line (h_1) along x_2 direction splits the samples into two parts and two samples (solid black triangle) are misclassified. After the first iteration, the samples misclassified are given high weights ($\omega_{2i}=0.25$ for samples misclassified, $\omega_{2i}=0.0625$ for samples correctly classified) so that the next classifier could focus on these samples misclassified. In order to emphasize this, in Fig. 4B the size of the sample symbols are scaled according to their misclassification *i.e.*, samples with larger weights have larger symbols. To achieve the minimal weighted misclassification error rate, the adjustment of sample weights forces the next classifier to correctly classify the samples with larger weights. Thus, the line (h_2) along x_1 direction splits the weighted samples into two parts, in which there are still two samples (solid black circle) misclassified with higher weights ($\omega_{3i}=0.25$ for samples misclassified in this step; $\omega_{3i}=0.143$ for samples misclassified in the first step; $\omega_{3i}=0.0375$ for samples all correctly classified in the above two steps). With the help of these weights, the third split can happen along x_2 direction on the right side (see Fig. 4). It is worth noting that as iterations proceed, those samples misclassified by the previous step classifier have their weights increased, whereas the weights are decreased for those classified correctly. Finally, the predictions from all classifiers are then combined through a weighted majority vote to produce the final prediction:

$$G(\mathbf{x}) = sign(\sum_{t=1}^{T} c_t G_t(\mathbf{x})) \tag{2}$$

Here $c_1, c_2, \ldots, c_t$ (the confidences, explained in the next section) are computed by the boosting algorithm and weigh the contribution of each respective classifier $G_t(\mathbf{x})$. Their effect is to give higher influence to the more accurate classifiers in the sequence. In the example, the weights of three classifiers are 0.69, 0.97, and 1.28, respectively. Fig. 5 shows the schematic illustration of the boosting procedure for classification. In the plot, it can clearly be seen that the AdaBoost algorithm calls this weak classifier repeatedly, each time providing it with training examples with different weight distribution. After many rounds, the boosting algorithm combines these weak rules into a single prediction rule that, hopefully, will be much more accurate than any one of the weak rules.

A classification example was constructed to elucidate that boosting can dramatically increase the performance of a very weak classifier. In the context of this example, the features $\mathbf{x}_1$ and $\mathbf{x}_2$ are derived from the standard independent Gaussian distribution, and the deterministic target y is defined by:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{2} \mathbf{x}_i^2 < 1.64 \\ -1 & \text{otherwise} \end{cases}$$

Fig. 6A shows the simulated data for a two-class problem marked by +1(dot) and −1(plus). There are 500 training samples, with approximately 250 samples in each class, and 500 test samples. Here a stump (a two-terminal node classification tree) is selected as the weak classifier. The test error rate for boosting with stumps as a function of the number of iterations is shown in Fig. 6B. From the plot, we can clearly see that a single stump applied to the training dataset yields a very poor test set error rate of 45.8%, compared with 50% for random guessing. However, as the boosting iterations proceed the error rate steadily decreases, reaching 5.2% after 278 iterations. Thus, boosting this simple weak classifier reduces its prediction error rate by almost a factor of nine. It also outperforms a single large classification tree (error rate 7.8%).

5.2. Boosting for classification

The commonly used boosting method for classification problems is Freund's AdaBoost in chemistry fields. There are three types of AdaBoost algorithms, discrete AdaBoost, real AdaBoost and gentle AdaBoost included. In addition, the AdaBoost.MH algorithm can be applied for multi-class classification. The algorithms are described in detail in Ref [17,48]. In this section the discrete AdaBoost algorithm is discussed.

Consider a training dataset with N samples belonging to two classes. The two classes are labeled as $y\in\{-1, 1\}$. The discrete AdaBoost algorithm consists of the following steps:

1. Assign initial equal weights to each sample in the original training set: $w_i^1=1/N$, $i=1, 2, \ldots, N$

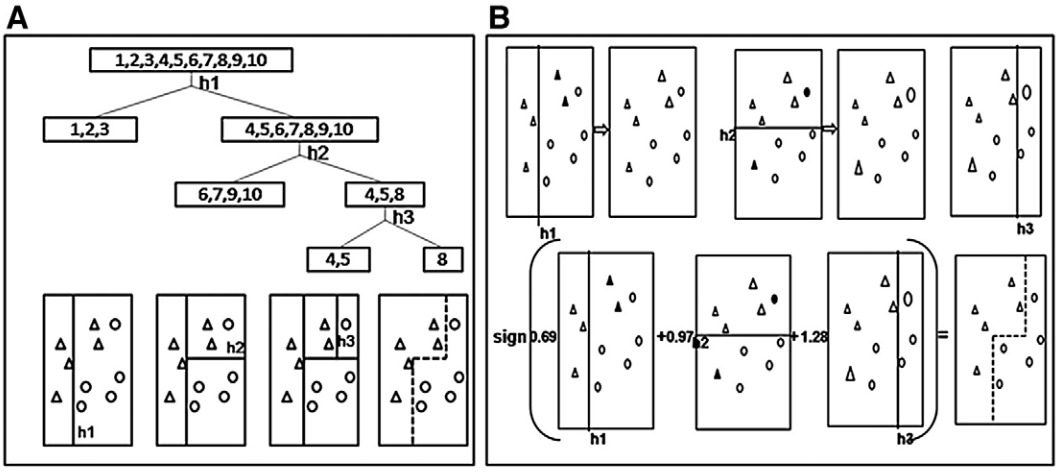

Fig. 4. (A) The overall classification process of CART. (B) The overall iteration process for AdaBoost.

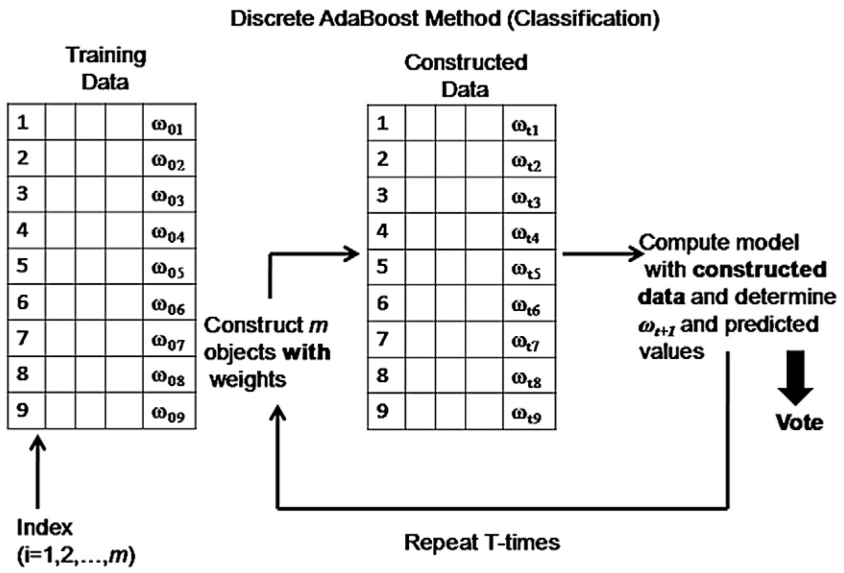
Discrete AdaBoost Method (Classification)
Training Data
Constructed Data
Construct m objects with weights
Compute model with constructed data and determine ω_{t+1} and predicted values
Vote
Index (i=1,2,…,m)
Repeat T-times

Fig. 5. The schematic illustration of the boosting procedure for classification. m: number of objects. T: the number of iterations. ω_{ti} : the weight of the i-th object at t-th iteration. Vote: Cast a Vote of T predicted values for classification.

2. For iterations $t = 1, 2, \ldots, T$:
   (a) Select a dataset with N samples from the original training set using weighted resampling. The chance for a sample to be selected is related to its weight. A sample with a higher weight has a higher probability to be selected.
   (b) Obtain a learner $f(\mathbf{x})$ from the resampled dataset.
   (c) Apply the learner $f(\mathbf{x})$ to the original training dataset. If a sample is misclassified, its error $err = 1$, otherwise its error is 0.
   (d) Compute the sum of the weighted errors of all training samples.

$$err^t = \sum_{i=1}^{N} (w_i^t \times err_i^t) \tag{3}$$

The confidence index of the learner $f(\mathbf{x})$ is calculated as:

$$c^t = \log((1-err^t) / err^t) \tag{4}$$

The lower the weighted error made by the learner $f(\mathbf{x})$ on the training samples is, the higher the confidence index of the learner $f(\mathbf{x})$ is.
   (e) Update the weight of all original training samples.

$$w_i^{t+1} = w_i^t \exp(c^t err_i^t) \tag{5}$$

The weights of the samples that are correctly classified are unchanged while the weights of the misclassified samples are increased.
   (f) Re-normalize w so that $\sum_{i=1}^{N} w_i^{t+1} = 1$
   (g) $T = t + 1$, if err<0.5 and $t<T$, repeat steps (a)–(f); otherwise, stop and $T = t - 1$. After T iterations in step 2, there are T learners $f^t(\mathbf{x})$, $t = 1, 2, \ldots, T$.

The performance of Discrete AdaBoost is evaluated by a test set. For a sample j of the test set, the final prediction is the combined prediction obtained from the T learners. Each prediction is multiplied by the confidence index of the corresponding learner $f(\mathbf{x})$. The higher the confidence index of a learner $f(\mathbf{x})$ is, the higher its role in the final decision is.

$$y_j = sign(\sum_{t=1}^{T} c^t f^t(\mathrm{x}_j)) \tag{6}$$

From the above algorithm, we can see that several interesting features of the algorithm are worth noting. The AdaBoost algorithm generates a set of weak learners, and combines them through weighted majority voting of the classes predicted by the individual weak learners. Each weak learner is obtained using samples drawn from an iteratively updated distribution of the entire training samples. Thus, AdaBoost maintains a distribution or set of weights over the original training set $\mathbf{Z}$ and adjusts these weights after each classifier is learned by weak learners. The adjustments increase the weight of samples that are misclassified by weak learners and decrease the weight of samples that are correctly classified. Hence, consecutive classifiers gradually focus on those increasingly hard-to-classify samples. Eq. (4) describes the confidence index of the weak learner. The higher the confidence index is, the more accurate its corresponding weak learner is. A large confidence index indicates that the corresponding weak learner plays a more important role in final decision. AdaBoost adopts such a strategy called weighted majority voting to weigh a set of weak learners. The idea behind the strategy is very intuitive and effective. Those classifiers that have shown high performance during training are rewarded with higher voting weights than others. Note that

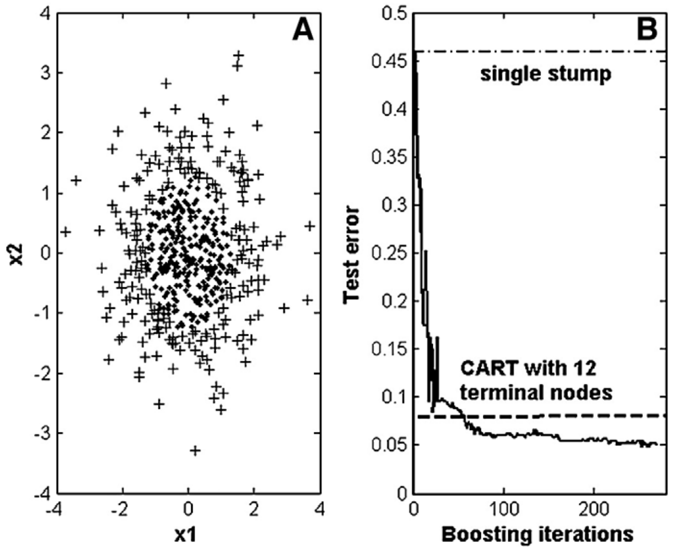

Fig. 6. (A) The simulated data for a two-class problem marked by +1(dot) and −1 (plus). (B)Test error rate for boosting with stumps as a function of the number of iterations. Also shown are the test error rates for single stump and CART with 12 terminal nodes.

the confidence index is larger than 0. This guarantees that the weak learner is better than random guessing. Eq. (5) describes the training set distribution update rule. Each distribution update of the training set is controlled by the weights of the training samples. The distribution weights of the samples that are correctly classified are unchanged while the distribution weights of the misclassified samples are increased. Thus, iteration by iteration, AdaBoost updates the distribution of the training samples in order to give a high attention toward those increasingly difficult samples. Once the preset number of iterations is reached, AdaBoost is ready to classify the unlabeled test samples.

Practically, AdaBoost have many advantages. It is fast, simple and easy to program. It has no parameters to tune (except for the number of iterations T). Moreover, it also requires no prior knowledge about the weak classifier and can flexibly be implemented with any weak classification method. A nice property of AdaBoost is its ability to identify outliers [62], *i.e.*, samples that are either mislabeled in the training data, or which are inherently ambiguous and hard to classify. Because AdaBoost focuses on the samples that are more difficult to classify, the samples with the highest weights often turn out to be outliers.

5.3. Boosting for regression

There are two methods used to establish the boosting regression model. The first one is by forward stage-wise additive modeling, which modifies the target values to effectively fit residuals. The second one is by imitating classification and changing sample weights to emphasize those which were poorly regressed on previous stages of the fitting process [63,64]. Since the latter method is similar to the boosting for classification, we mainly focus on the forward stage-wise additive model in the section.

Generally speaking, an additive model can be expressed as:

$$F(\mathrm{x}) = \beta_1 f_1(\mathrm{x}) + \beta_2 f_2(\mathrm{x}) + \ldots + \beta_T f_T(\mathrm{x}) = \sum_{t=1}^{T} \beta_t f_t(\mathrm{x}) \tag{7}$$

where, β_t, $t = 1, 2, \ldots, T$ are the expansion coefficients, typically $\beta_t = 1$, and the functions $f_t(\mathbf{x})$ are usually chosen as simple functions with input $\mathbf{x}$. The boosting method aims at finding a latent function $F(\mathbf{x})$ that minimizes a loss function such as squared-error loss etc between response $\mathbf{y}$ and estimated values $F(\mathbf{x})$ (in most cases, the latent function is not known to us). So, it attempts to seek for a simple function $f(\mathbf{x})$ at each iteration.

$$\min_{f(\mathrm{x})} \sum_{i=1}^{N} \| y_i - F_{t-1}(\mathrm{x}_i) - f_t(\mathrm{x}_i) \|^2 \tag{8}$$

where t is the number of the $f(\mathbf{x})$ added. Thus, each simple function can be generated by fitting previous residuals with the original training set $\mathbf{X}$.

The algorithm of boosting based on the forward stage-wise additive strategy is the following:

1. Initially fit the training data using a basic regression model $f(\mathbf{X})$, expressed as $\hat{y}_1 = f_1(\mathbf{X})$.

   Calculate the residual : $\mathrm{y}_{\mathrm{res}} = \mathrm{y} - v_1 \hat{\mathbf{y}}_1$ (9)

   Here $0 < v < 1$; v is a shrinkage parameter which may be a constant or not. This shrinkage parameter can effectively prevent overfitting problems. Only the v part of the fitted values is extracted at each step.
2. For $t = 2, \ldots, T$, repeat the following steps:
   (a) Fit the current residual $\mathbf{y}_{\mathrm{res},t-1}$ using basic regression model: $\hat{y}_t = f_t(\mathbf{X})$.
   (b) Update the residual,

$$\mathbf{y}_{\mathrm{res},t} = \mathbf{y}_{\mathrm{res},t-1} - v_t \hat{\mathbf{y}}_t \tag{10}$$

In this step, only the v part of the fitted $\hat{\mathbf{y}}_t$ is used as useful regression information.

3. The final prediction is

$$\mathrm{y}_{\mathrm{pre}} = v_1 \hat{\mathrm{y}}_1 + v_2 \hat{\mathrm{y}}_2 + \ldots + v_{T-1} \hat{\mathrm{y}}_{T-1} + v_T \hat{\mathrm{y}}_T = \sum_{t=1}^{T} v_t f_t(\mathbf{X}) \tag{11}$$

The basic idea of boosting as an additive model is to sequentially construct additive regression models by fitting a basic regression model to the current residuals that are not fitted by previous models (see Appendix A). The current residuals are the gradient of the loss function (e.g. a squared-error function) being minimized at current step. Initially, a basic regression model is generated by fitting the response $\mathbf{y}$ with the predictor $\mathbf{X}$, and the initial residual of $\mathbf{y}$ is calculated. Hence, iteration by iteration, boosting only considers the current residual of $\mathbf{y}$ and always fits current residual of $\mathbf{y}$ with the original $\mathbf{X}$. Thus, boosting sequentially adds a series of basic regression models which are simple to implement. Each constructed model is shrunk by a shrinkage parameter v, which is a positive value less than 1. The v acts as a weight for current model which prevents possible overfitting by constraining the fitting process [65]. Each time the whole $\mathbf{X}$ is used to extract information correlated to the residuals of $\mathbf{y}$, but the information extracted is not 100% used. Only v part is used, and the rest, $1 - v$, part is put back to avoid overfitting. The sequential adding is repeated, resulting in many weighted basic regression models. Once a preset number T of iterations is reached, a single test set will be used to predict. The final prediction is the combination of these weighted basic regression models.

The aforementioned method is termed gradient boosting (GB). In order to improve the accuracy and execution speed of GB, Friedman [20] incorporated randomization analogous to bagging into the procedure and exploited a new algorithm in GB, which lead to the well-known stochastic gradient boosting (SGB). At each iteration of SGB, a fraction η of the training data are sampled at random (without replacement) from the full training dataset. These randomly selected subsamples are subsequently used in place of the full samples to fit the base learner and compute the model update for current iteration. The rest of the algorithm is identical to the gradient boosting algorithm. According to Friedman's research, a typical value for η can be 1/2 for large number N of the training observations. The implementation of sampling strategy not only reduces the computing time by the same fraction η, but also actually obtains a more accurate model in most cases.

Instead of the above squared-error loss function other robust loss functions with accompanying gradient functions can be implemented. Table 1 summarizes the gradients of commonly used loss functions. For squared-error loss, the negative gradient is just the ordinary residual (the above algorithm). For absolute-error loss, the negative gradient is the sign of the residual. Thus, the signs of the current residuals are fitted with original predictor $\mathbf{X}$ at current step. For Huber M-regression, the negative gradient is the combination between the above two (see Table 1).

In order to well elucidate the features of boosting as an additive model, we now investigate the effect of boosting in a simple quadratic model.

$$y_i = 2 + 3x_i^2 + \varepsilon_i \qquad i = 1, \ldots, n$$

where, ε_i is a noise term distributed according to a Gaussian distribution of zero mean and unit variance and each independent x is uniformly distributed in [0, 1]. All x_i's are independent from all ε_i's. The true target function $F_{\mathrm{true}}(x_i) = 2 + 3x_i^2, x_i \in [-2, 2]$. We use stumps as base learners and run squared-error loss boosting algorithm for regression from the above function. Fig. 7 shows the fitting results of 50 boosting iterations. From Fig. 7, we can clearly see that the boosting method adaptively captures the nonlinear structure of the target function. Compared with the target function indicated in black quadratic curve, the estimated function generated by boosting with stumps

Table 1
Gradients for commonly used loss functions.

Loss function		Gradient
Squared-error loss	$1/2[y_i - F(x_i)]^2$	$y_i - F(x_i)$
Absolute-error loss	$\lvert y_i - F(x_i)\rvert$	$\text{sign}[y_i - F(x_i)]$
Huber	$1/2[y_i - F(x_i)]^2$ for $\lvert y_i - F(x_i)\rvert \le \delta$	$y_i - F(x_i)$ for $\lvert y_i - F(x_i)\rvert \le \delta$
	$\lvert y_i - F(x_i)\rvert$ for $\lvert y_i - F(x_i)\rvert > \delta$	$\delta(y_i - F(x_i))$ for $\lvert y_i - F(x_i)\rvert > \delta$

δ is the α-th quantile $\{\lvert y_i - F(x_i)\rvert\}$.

consists of many piecewise lines approximating the true function. This is a consequence of using tree-based models such as CART that produces discontinuous constant models.

5.4. Model selection

The parameter optimization is a very important aspect for establishing a good model. Besides the parameters of the base function $f(\mathbf{x})$ (there are different parameters for different base functions), two regularization parameters, namely shrinkage parameter v and the number of iterations T, control the degree-of-fit in the boosting model (In AdaBoost, T is only decided parameter.). The choice of these parameters has a crucial effect on the performance. If one does not choose them properly, one will not achieve the excellent performance reported in many papers. Model selection techniques can provide principled ways to select proper model parameters [66–74]. Herein, we make use of well-known cross-validation or independent validation set techniques to select proper boosting parameters. In boosting, controlling the contribution of single base learner through v may help to reduce or spread around the influence of noise caused by additional factors and subsequently improve the fitting and predictive accuracy. The optimal value v that achieves the best performance is usually small (less than 0.1). In practice, to obtain the better prediction results, a suitable value for v can be estimated through cross-validation. For the number of iterations T, controlling the value of T can regulate the degree to which expected loss on the training data can be minimized. However, a large T can lead to overfitting resulting in degraded future model performance. Generally speaking, the best value for T can be estimated by some model selection method, such as cross-validation. A convenient way to determine T is to monitor prediction error as a function of T on a validation set. This seems analogous to the early stopping rule often used in the neural network modeling. According to Copas' study [65], it has often been found that regularization through shrinkage provides superior results to that obtained by restricting the number of iterations.

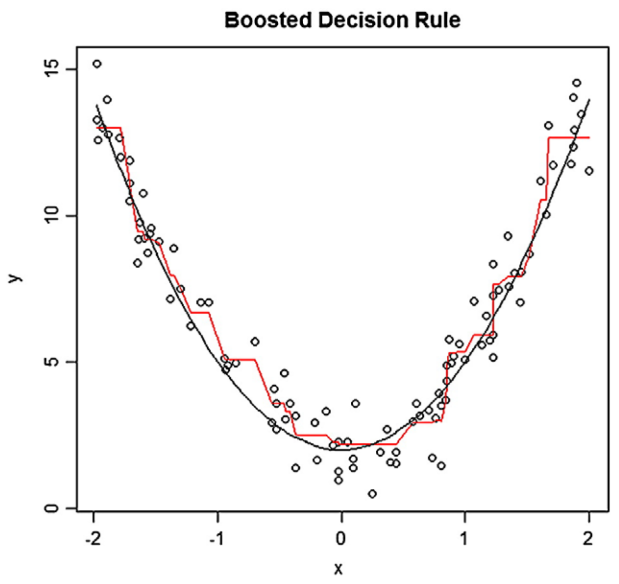

Fig. 7. Boosting stumps estimate based on a typical sample (circles). The target function is indicated in black curve. Boosting is done with50 iterations.

6. Further considerations

In this section, we will discuss some key problems related to boosting, which may be very helpful to our in-depth understanding of the boosting technique. (1) In general, good performance of ensemble methods can be expected when the individual learners are diverse. The model diversity formed the cornerstone of ensemble learning [75]. For boosting, the model diversity may be obtained by using different training data to sequentially train individual learners. Such datasets are often generated through weighted resampling techniques where training data subsets are drawn randomly with probability proportional to their weights. There are two ways that AdaBoost can use these weights to construct a new training set **Z*** to give a weak learner. (a) Boosting by sampling: samples are drawn with replacement from **Z** with probability proportional to their weights. Samples with high probability will occur more often than those with low probability, while some samples may not occur in the constructed training samples at all although their probability is not zero. (b) Boosting by weighting: weak learners can accept a weighted training set directly. With such an algorithm, the entire training set **Z** with associated weights is given to the weak learner. Both methods have been shown to be very effective. (2) A crucial property of boosting is that boosting appears to be more resistant to overfitting. Recently, Massart used a boosting algorithm combined with PLS (BPLS) to predict seven chemical datasets and showed that BPLS is relatively resistant to overfitting [49]. Overfitting is a commonly observed situation where the learner performs well on training data, but has a large error on the test data [76]. Overfitting is usually attributed to memorizing the data, or learning the noise in the data. Much to our surprise, boosting can continuously decrease the generalization error even after training error reaches zero. Schapire et al. later provided an explanation to this phenomenon based on the so-called margin theory. The details of margin theory can be found in ref [12]. Generally speaking, the margin of a sample **x** can be considered as its distance from the decision boundary. The further away a sample is from the boundary, the larger its margin, and hence the higher the confidence of the classifier in correctly classifying this sample. Margins are usually used in the context of support vector machine (SVM). In fact, SVMs are based on the idea of maximizing the margins between samples and the decision boundary. They aim at finding a subset of samples, called support vectors, which are closest to the decision boundary. The support vectors can be used to define the margin that separates the classes. Similar to the margin of SVMs, Schapire et al. used a slightly different definition of a margin for AdaBoost and showed that AdaBoost also boosts the margins. That is to say, it can find the decision boundary having maximum distances to all samples. In the context of AdaBoost, the margin of a sample is simply the difference between the total votes. It receives from correctly identifying classifiers and the maximum vote received by any incorrect class. They showed that the ensemble error is bounded with respect to the margin, but is independent of the number of classifiers. The concept of a margin for AdaBoost and SVMs is conceptually illustrated in Fig. 8. (3) Bagging and boosting being non-Bayesian methods have, to a certain extent, some similarity with Bayesian model [18]. A general form for Bagging, Boosting and Bayesian models can be expressed as:

$$\hat{F}(\mathbf{x}) = \sum_{t=1}^{T} w_t E(\mathbf{y} \mid \mathbf{x}, \hat{\theta}_t) \tag{12}$$

where, $\hat{\boldsymbol{\theta}}_t$ is a set of sub-model parameters and w_t the weight of the t-th model. $E(*)$ is the expectation value at some point. For Bayesian models the $w_t = 1/T$, and the average estimates the posterior mean by sampling $\boldsymbol{\theta}_t$ from the posterior distribution. That is to say, the Bayesian approach fixes the data and perturbs the parameters, according to the current estimate of the posterior distribution. For bagging, $w_t = 1/T$ and $\hat{\boldsymbol{\theta}}_t$ is the parameter vector refit to bootstrap samples of the training data. That

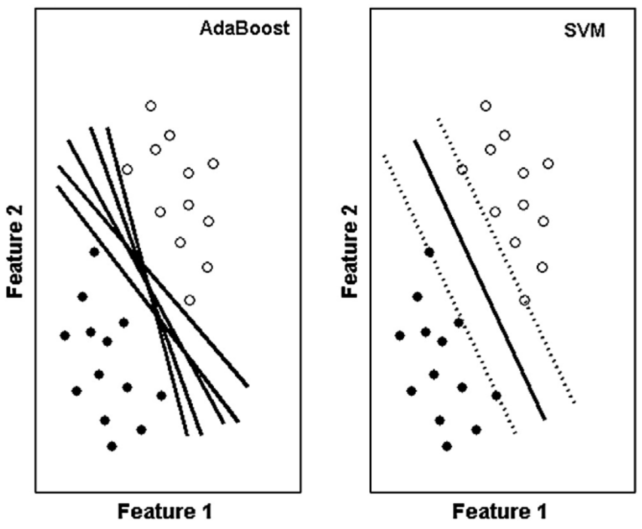

Fig. 8. Both SVMs and AdaBoost maximize the margin between samples of different classes.

is to say, bagging perturbs the data in an *i.i.d* (independent and identically distributed) fashion and then re-estimates the model to give a new set of model parameters. At the end, a simple average of the model predictions from different bagged samples is computed. For boosting, all weights are equal to 1, but $\hat{\theta}_t$ is typically selected in a nonrandom sequential fashion to constantly improve the fit. That is to say, boosting fits a model that is additive in the models of each individual basic learner, which are learned using non *i.i.d* samples. The comparison of the three methods can help us gain a deeper understanding of the boosting concept. (4) Compared with bagging, boosting has a significant advantage. Boosting can not only reduce the variance of the weak learner by averaging several weak learners obtained from different subsamples of the training samples, but also reduce the bias of the weak learner by forcing the weak learner to concentrate on hard-to-classify samples [11,77]. (5) The underlying idea of booting is deeply rooted in our daily life decision making. It provides chemists with new important aspects of how to think about chemical problems and build chemical models. In practice, certain chemical problems are just too difficult for a given individual leaner to solve. More specifically, the form of the target (modeled) function may be too complex to model with a single model, or the target function may lie outside the selected model function space (wrong model). Typical examples in chemistry are QSAR/QSPR studies [78]. Because of the diversity of chemicals, the truth may be a very complex function form (nonlinear) and it seems therefore hard to construct an accurate integral QSAR/QSPR model generalizing the diverse datasets. However, finding many relatively linear functions can be a lot easier than finding a single, highly accurate nonlinear function. Thus, such questions might resort to sequentially combining appropriate simple linear functions to approximate the latent nonlinear function.

7. Experiment

In this section, the boosting algorithm was illustrated by two moderate sized datasets. Both datasets were randomly partitioned into a training set consisting of about three-fourths of the data, with the remaining data being used as the test set for demonstrating the performance of the algorithm. All the datasets are summarized in Table 2. For near infrared spectroscopy (NIR) data, Zhang's BoostPLS[49] was used to construct the regression model. For classification data, the base learner is a stump. For the datasets, the average validation error values of 10 repeated model runs are reported.

In this study gbm package (coded in R language) was employed to construct the boosting model [79]. In addition, all other programs needed such as PLS and BoostPLS etc are coded in Matlab.

Table 2
The training set and test set for classification and regression problems.

	Classification		Regression
	Heart disease data		NIR data
	Class 1	Class 2	–
Training set	89	113	170
Test set	33	35	56

7.1. Heart disease data for classification

This is a dataset from the UCI Machine Learning group [80]. It contains 270 samples belonging to 2 classes. Each sample is characterized by 13 attributes. In this study, 202 samples out of the 270 samples are randomly chosen to construct the training set, and the remaining ones construct the test sets (Table 2).

7.2. Near infrared data for regression

This dataset [81] collected near infrared spectra of 246 diesel samples with 20 outliers. The response value is the boiling point at 50% recovery (BP50). This data was originally obtained at Southwest Research Institute (SWRI) [82] on a project sponsored by the U.S. Army. In this case, only the 226 normal samples are used for the boosting model. The 226 samples were randomly divided into the training set and test set for the construction and evaluation of the boosting model (Table 2).

8. Results and discussions

8.1. Heart disease data

For classification problems, Friedman's stochastic gradient boosting (SGB) based on Bernoulli criterion is employed to construct the classification model. SGB has three primary meta-parameters: the number of iterations T, the shrinkage ν and the fraction η of the training data. These parameters were discussed in the Section 5.4 model selection. Herein, we use $\nu = 0.01$, $\eta = 0.5$ (default value). The best value for T should be estimated by 10-fold cross-validation. In addition, there are the meta-parameters associated with the selection of base learners. For our study, we select tree as the base learner. Thus, the number L of terminal nodes is the primary meta-parameter of this base learner. The best choice of its value depends strongly on the nature of the real problem. Generally speaking, the number L of terminal nodes is relatively small so that we can obtain a weak classifier.

Additionally, we compare the performance of boosting (SGB) with that of two commonly used methods: partial least squares discriminant analysis (PLS-DA) and Bagging. PLS-DA is a standard technique used for classification problems in chemometrics. It is usually used as a reference method to test the other modeling methods. In bagging, multiple trees are grown independently on bootstrap samples of the training data, and the predictions are obtained by aggregating the outputs of all the trees (in classification, all the trees vote). The results of every method described above are shown in Table 3. In the table, we reported means of the performance statistics for 10 replicates of train error, 10-fold cross-validation error and an independent test error. From Table 3, it is shown that bagging tree has an overall poor

Table 3
Misclassification rates and RMSE of four methods on two datasets.

	Heart disease data				NIR data		
	Train	CV	Test		Train	CV	Test
Boosting tree	0.114	0.151	0.147	BoostingPLS	1.9570	2.6626	2.4091
Bagging tree	–	0.188	0.186	BaggingPLS	2.1788	2.8449	2.5429
PLS-DA	0.149	0.163	0.162	PLS	2.1838	3.2182	2.6037

predictive error of 0.188 for cross-validation set and 0.186 for individual test set. The results of PLS-DA predictions are better than ones for bagging tree (0.163 for cross-validation set and 0.162 for individual test set). However, boosting tree achieves the best performance of 0.151 for cross-validation set and 0.147 for individual test set among these three methods. In this process, we firstly determined the optimal number of trees (675) for the ensemble by running SGB within 10-fold cross-validation. Once the number of trees was determined, we trained the final ensemble using all the training data. Fig. 9 shows the train and CV error rates as a function of the number of boosting with stumps. From Fig. 9, it is shown that the train and CV error is seen to decrease monotonically with increasing the number of boosting iterations more rapidly during the early stages and then leveling off to be nearly constant as iterations increase. From the final model, we extracted the variable importance for all variables, and an individual test set was used to check the performance of the final model. The variable importance of SGB is plotted in Fig. 10. It can be seen that variables 3, 12 and 13 are the most features which have the highest differentiating power for the two classes using GBM. For three classification methods, we can clearly see that boosting gets the best predictive results. The reason for this may lie in that boosting captures the most suitable nonlinear structure of the heart disease data by maximizing the margin between two classes. To summarize: boosting tree can be used as a very promising modeling method due to its good properties.

8.2. NIR data

Partial least squares (PLS) is a basic and powerful tool in analytical chemistry, especially, it is more suitable to obtain a regression model for high collinear data such as near infrared (NIR) spectroscopic data. It can reduce the number of variables in original data to relative small number of uncorrelated variables called latent variables to some extent. In this study, PLS is used as a reference method and the comparative analysis between PLS and boosting is conducted to give some insight into these two methods. Moreover, bagging PLS is also used as an ensemble method to compare the performance of boosting. In our study, the boostPLS algorithm which is the combination of boosting and PLS is used to construct the regression model for NIR data. In the boostPLS algorithm, there are two important parameters affecting the performance of the final model, including the number T of iterations and the shrinkage value ν. According to Zhang's study, a lower ν may also yield an acceptable predictive results but a much higher T may then be needed, which is not economic in computation time. So the higher value of ν which we select is 0.9. The number T of iterations is chosen by 10-fold cross-validation (here the optimal value of T is 2187). Table 3 lists the predictive results of three methods. From Table 3, it is shown that the performance of two ensemble methods outperforms that of a single PLS model and boostPLS gives the best results among all the methods. Its root mean square error of fitting (RMSEF), root mean square error of prediction (RMSEP) and root mean square error of testing (RMSET) are 1.9570, 2.6626 and 2.4091, respectively. The predictive results of BoostPLS are shown in Fig. 11. Fig. 12 shows RMSEP for the cross-validation set and RMSET for the individual test set. From Fig. 12, we can see RMSEP and RMSET have a significant downtrend against the number of iterations and the minimum value of RMSEP occurs at the 2187th iteration. RMSEP against the number of latent components (LCs) using the PLS model and BoostPLS model are plotted in Fig. 13. From Fig. 13A, it can clearly be seen that RMSEP begins to increase after 12 LCs. Therefore, 12 LCs are used to construct the PLS model using the training set. Fig. 13B shows RMSEP for the cross-validation set as a function of the number of BoostPLS iterations. Compared with Fig. 13A, Fig. 13B suggests that BoostPLS is somewhat resistant to overfitting. For the better performance of BoostPLS, it may be indicated that the BoostPLS not only cope

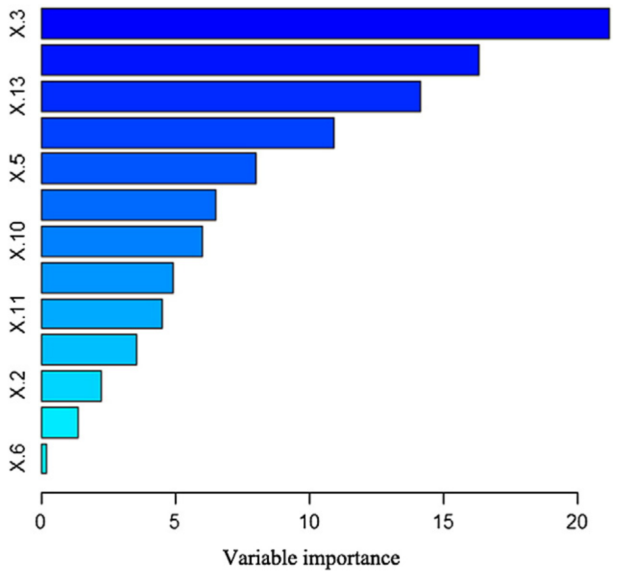

Fig. 10. The relative importance of the variables for the heart disease data.

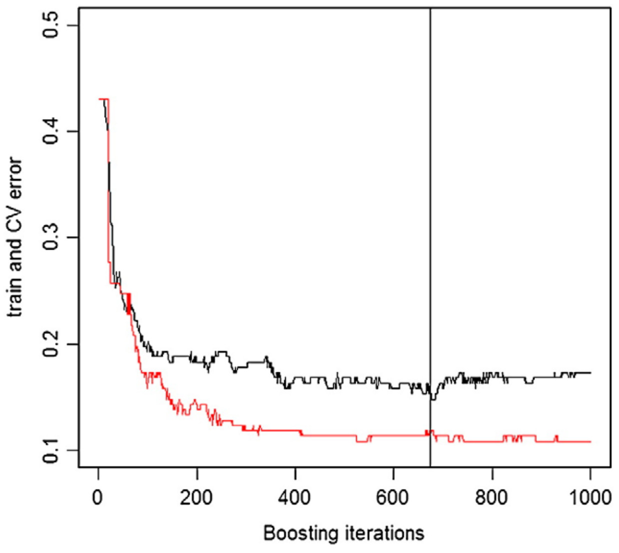

Fig. 9. Train and CV error rates as a function of the number of boosting with stumps for the heart disease data (red: train error; black: CV error). Vertical line indicates the optimal iteration number of 675. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

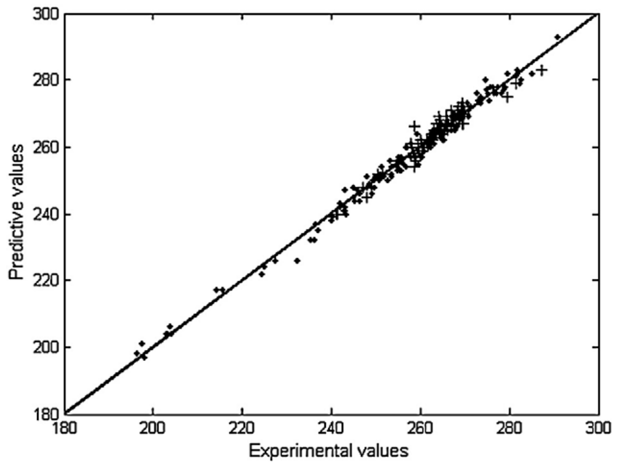

Fig. 11. The fitting and predictive values versus the experimental values using the BoostPLS algorithm on the NIR data (dot: the training set, plus: the test set).

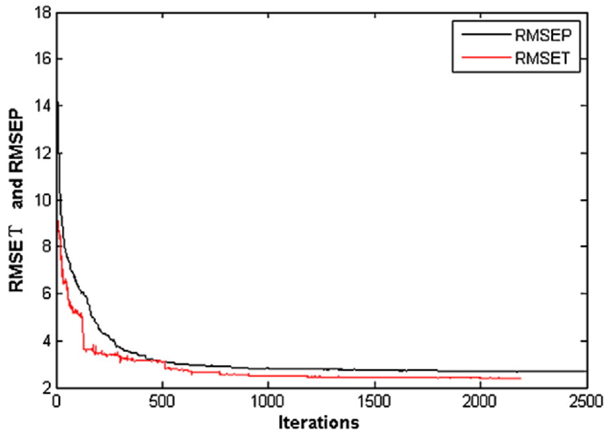

Fig. 12. RMSEP for the cross-validation set and RMSET for the individual test set using the BoostPLS algorithm on the NIR data.

with the linear relationship between data matrix **X** and response value **y**, but also is capable of modeling the nonlinear part of the correlation between them to certain extent due to the additive nature of BoostPLS. PLS can't work well in that it is inherently a linear modeling method, which makes it impossible to well interpret the nonlinear part of the data. However, we can clearly see that BoostPLS have its limitation and disadvantages when coping with the spectral dataset. BoostPLS is the combination of many simple models and this makes it difficult for the researches to interpret the model. In conclusion, BoostPLS can be seen as a competitive and promising method when modeling the nonlinear NIR dataset.

9. Conclusion

The goal of the present article was to give a simple introduction into the exciting field of ensemble learning methods, boosting in particular. We only briefly touched the ensemble learning theory — omitting many theoretical details — and instead focused on how to use and work with the boosting algorithms. Boosting is a general method for improving the accuracy of any given algorithm. The idea of boosting deeply roots in our daily life experiences. We unconsciously use it all the time! Generally speaking, combining various opinions through certain thought process can always improve our confidence of right decisions and reduce the risk of wrong selections. In mathematics, it works by sequentially applying a classification algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers thus produced. Boosting can usually improve the performance of a single weak learner. Besides, a very significant advantage of boosting is that the nonlinear relationship in classification and regression can be modeled to large extent in that it is considered as a way of fitting an additive expansion in a set of elementary base learners.

On the other hand, some caveats are certainly worth noting. The performance of boosting on a particular problem clearly relies on the particular data and the choice of the weak learner. In some cases boosting may fail to perform well, especially for the data with noises. Boosting seems to be especially susceptible to noise[16]. According to the aforementioned examples, we can clearly see that the boosting procedure outperforms the other commonly used methods. However, we keep in mind "no free lunch" theorem which states that no single method can all obtain the best results in any circumstance. Nevertheless, the results show that the boosting method is generally competitive with or superior to other commonly used methods. It is believed that the boosting method will become very more and more popular in chemistry due to its easy implementation and high predictive performance.

10. Software source

There are many packages in R language used to establish the boosting model. R is a free software used for statistical computing [83]. The packages used for boosting include gbm package (http://cran.r-project.org/web/packages/gbm/index.html), ada package (http://cran.r-project.org/web/packages/ada/index.html), mboost package (http://cran.r-project.org/web/packages/mboost/index.html) and RWeka package (http://cran.r-project.org/web/packages/RWeka/index.html) etc. These packages can directly be used to solve various chemical problems. BoostPLS is coded in Matlab 6.5. We can provide its package if the readers need it. Please feel free to contact with corresponding author.

Acknowledgements

This work is financially supported by the National Nature Foundation Committee of PR China (Grants No. 20875104 and No. 10771217), the International Cooperation Project on Traditional Chinese Medicines of the Ministry of Science and Technology of China (Grant No. 2007DFA40680). The studies meet with the approval of the university's review board. We also would like to thank the reviewers for their useful discussions, comments and suggestions throughout this entire work.

Appendix A

Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. For classification, the base functions are the individual classifiers. More generally, basis function expansions take the form:

$$\hat{F}(\mathrm{x}) = \sum_{t=1}^{T} \beta_t f(\mathrm{x}, \mathrm{a}_t) \qquad \text{(A1)}$$

where β_t, $t=1, 2, \ldots, T$ are the expansion coefficients, and the functions $f(\mathbf{x}, \mathbf{a}_t)$ are usually chosen to be simple functions of the multivariate argument $\mathbf{x}$, characterized by a set of parameters $\mathbf{a}_t$. Typically, these models are fit by minimizing a loss function ψ averaged over the training data, such as squared-error or a likelihood-based loss function.

$$(\beta_t, \mathrm{a}_t) = \arg\min_{\beta, a} \sum_{i=1}^{N} \Psi(y_i, \sum_{t=1}^{T} \beta_t f_t(\mathrm{x}; \mathrm{a}_t)) \qquad \text{(A2)}$$

For many loss functions ψ and basis functions $f(\mathbf{x}, \mathbf{a}_t)$, complex numerical optimization methods must be applied to solve the above

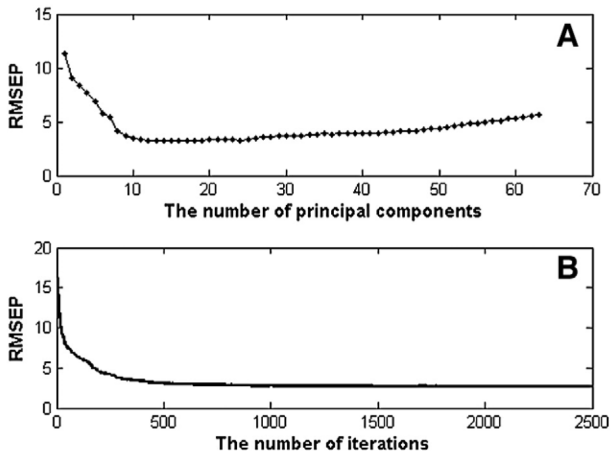

Fig. 13. RMSEP of 10-fold cross-validation versus the number of principal components using the PLS model (A). RMSEP of 10-fold cross-validation versus the number of iterations using the BoostPLS model (B).

formula. However, forward stage-wise modeling should approximate the solution to the above formula by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added.

$$(\beta_t, \mathbf{a}_t) = \arg\min_{\beta,a} \sum_{i=1}^{N} \Psi(y_i, F_{t-1}(\mathbf{x}_i) + \beta f(\mathbf{x}_i, \mathbf{a}_t)) \tag{A3}$$

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \beta f(\mathbf{x}; \mathbf{a}_t) \tag{A4}$$

The formulas above are called "boosting". For classification problems, y is the class label and $\psi(y, F)$ is either an exponential criterion or negative binomial log-likelihood. For regression problems, y is continuous values and $\psi(y, F)$ can be chosen from a variety of functions e.g. squared-error; absolute-error and Huber loss functions. The function $f(\mathbf{x}, \mathbf{a})$ is called a "weak learner" or "base learner".

For a particular loss ψ such as squared-error and a base learner $f(\mathbf{x}, \mathbf{a})$, the solution to the Eqs. (A3) and (A4) can be obtained in the following way: suppose that the current approximation $F_{t-1}(\mathbf{x})$ at the t-th iteration, the function $\beta f(\mathbf{x}_i, \mathbf{a}_t)$ can be regarded as the best greedy step towards the optimal solution $F^*(\mathbf{x})$. So the selection of the function $f(\mathbf{x}, \mathbf{a})$ should be most highly correlated with the negative gradient $-g_t(\mathbf{x})$ over the data distribution. Thus, the function $\beta f(\mathbf{x}_i, \mathbf{a}_t)$ can take place of the negative gradient $-g_t(\mathbf{x})$:

$$a_t = \arg\min_{\beta,a} \sum_{i=1}^{N} [-g_t - \beta f(\mathbf{x}_i, \mathbf{a}_t)]^2 \tag{A5}$$

where, for squared-error loss criterion, the negative gradient can be replaced with the residuals after every iteration $\mathbf{r}_t = \mathbf{y} - F_{t-1}(\mathbf{x})$. Subsequently, the line search is performed:

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{N} [y_i - F_{t-1} - \rho f(\mathbf{x}_i, \mathbf{a}_t)]^2 \tag{A6}$$

and the approximation updated

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t f(\mathbf{x}, \mathbf{a}_t) \tag{A7}$$

Thus, for any $f(\mathbf{x}, \mathbf{a})$ for which a least-squares algorithm exists, one can use this approach to minimize any loss in conjunction with forward stage-wise additive modeling.

References

[1] L.G. Valiant, A theory of the learnable, Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, ACM, 1984.
[2] T.G. Dietterich, Lecture Notes in Computer Science 1857 (2000) 1–15.
[3] D. Opitz, R. Maclin, Journal of Artificial Intelligence Research 11 (1999) 169–198.
[4] M. Kearns, L.G. Valiant, Journal of the Association for Computing Machinery 41 (1994) 67–95.
[5] L. Breiman, Annals of Statistics 26 (1998) 801–824.
[6] L. Breiman, Neural Computation 11 (1999) 1493–1517.
[7] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, Cambridge, 2006.
[8] R.E. Schapire, Machine Learning 5 (1990) 197–227.
[9] Y. Freund, Information and Computation 12 (1995) 252–285.
[10] Y. Freund, R.E. Schapire, Journal of computer and system sciences 55 (1997) 119–139.
[11] Y. Freund, R.E. Schapire, Machine learning, Proceedings of the Thirteenth International Conference, 1996, pp. 148–156.
[12] R.E. Schapire, Y. Freund, P. Bartlett, W.S. Lee, Annals of Statistics 26 (1998) 1651–1686.
[13] R.E. Schapire, Y. Singer, Machine Learning 37 (1999) 297–336.
[14] J.R. Quinlan, Bagging, Boosting, and C4.5, Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1, 1996, pp. 725–730.
[15] E. Bauer, R. Kohavi, Machine Learning 36 (1999) 105–139.
[16] T.G. Dietterich, Machine Learning 40 (2000) 139–157.
[17] J.H. Friedman, T. Hastie, R. Tibshirani, Annals of Statistics 28 (2000) 337–374.
[18] J.H. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning: Data Mining, Inference and Prediction, Springer-Verlag, New York, 2008.
[19] J.H. Friedman, Annals of Statistics 29 (2001) 1189–1232.
[20] J.H. Friedman, Computational Statistics & Data Analysis 38 (2002) 367–378.
[21] Y. Freund, R. Iyer, R.E. Schapire, Y. Singer, Journal of Machine Learning Research 4 (2004) 933–969.
[22] Y. Freund, 12th Annual Conference on Computational Learning Theory, Santa Cruz, California, 1999, pp. 293–318.
[23] M. Assaad, R. Bon, H. Cardot, Information Fusion 9 (2008) 41–55.
[24] R.E. Schapire, Workshop on Nonlinear Estimation and Classification, Berkeley, CA, Berkeley, CA, 2001, pp. 149–171.
[25] R.E. Schapire, M. Rochery, M. Rahim, N. Gupta, IEEE Transactions on Speech and Audio Processing 13 (2005) 174–181.
[26] S. Borra, A. Ciaccio, Computational Statistics & Data Analysis 38 (2002) 407–420.
[27] E. Cesario, F. Folino, A. Locane, G. Manco, R. Ortale, Knowledge and Information Systems 15 (2008) 285–320.
[28] W.C. Lin, M. Oakes, J. Tait, International Workshop on Content-based Multimedia Indexing, IEEE, London, ENGLAND, 2008, pp. 176–183.
[29] T. Choudhury, J.M. Rehg, V. Pavlovic A., 16th International Conference on Pattern Recognition (ICPR), IEEE Computer Soc, Quebec City, Canada, 2002, pp. 789–794.
[30] J.W. Robinson, Health Services Research 43 (2008) 755–772.
[31] J. Gertheiss, G. Tutz, Bioinformatics 25 (2009) 1076–1077.
[32] P. Geurts, M. Fillet, D. Seny, M.-A. Meuwis, M. Malaise, M.-P. Merville, L. Wehenkel, Bioinformatics 21 (2005) 3138–3145.
[33] H.Z. Li, Y.H. Luan, Bioinformatics 21 (2005) 2403–2409.
[34] M. Dettling, P. Buhlmann, Bioinformatics 19 (2003) 1061–1069.
[35] P.Y. Hong, X.S. Liu, X. Lu, J.S. Liu, W.H. Wong, Bioinformatics 21 (2005) 2636–2643.
[36] H. Binder, A. Allignol, M. Schumacher, J. Beyersmann, Bioinformatics 25 (2009) 890–896.
[37] T. Hothorn, P. Buhlmann, Bioinformatics 22 (2006) 2828–2829.
[38] H. Saigo, T. Uno, K. Tsuda, Bioinformatics 23 (2007) 2455–2462.
[39] C.-C. Lin, Y.-S. Tsai, Y.-S. Lin, T.-Y. Chiu, Bioinformatics 23 (2007) 3374–3381.
[40] B.L. Wu, T. Abbott, D. Fishman, W. McMurrary, G. Mor, K. Stone, D. Ward, K. Williams, H.Y. Zhao, Bioinformatics 19 (2003) 1636–1643.
[41] J.-H. Hong, S.-B. Cho, Pattern Recognition 42 (2009) 1761–1767.
[42] H. Takahashi, Y. Murase, T. Kobayashi, H. Hongda, Biochemical Engineering Journal 33 (2007) 100–109.
[43] J.-W. Hsieh, Y.-T. Hsu, Pattern Recognition 41 (2008) 3078–3091.
[44] H. Altun, G. Polat, Expert Systems with Applications 36 (2009) 8197–8203.
[45] I. Zitouni, H.-K.w. Jeff Kuo, C.-H. Lee, Speech Communication 41 (2003) 647–661.
[46] X.Z. Liu, L. Zhang, M.J. Li, H.J. Zhang, D.X. Wang, Pattern Recognition 38 (2005) 887–901.
[47] N. Shafik, G. Tutz, Computational Statistics & Data Analysis 53 (2009) 2453–2464.
[48] M.H. Zhang, Q.S. Xu, F. Daeyaert, P.J. Lewi, D.L. Massart, Analytica Chimica Acta 544 (2005) 167–176.
[49] M.H. Zhang, Q.S. Xu, D.L. Massart, Analytical Chemistry 77 (2005) 1423–1431.
[50] P. He, K.-T. Fang, Y.-Z. Liang, B.-Y. Li, Analytica Chimica Acta 543 (2005) 181–191.
[51] E. Deconinck, M.H. Zhang, D. Coomans, Y. Vander Heyden, Journal of Chemical Information and Modeling 46 (2006) 1410–1419.
[52] Y.-P. Zhou, J.-H. Jiang, European Journal of Pharmaceutical Sciences 28 (2006) 344–353.
[53] E. Deconinck, M.H. Zhang, D. Coomans, Y.V. Heyden, Journal of Chemometrics 21 (2007) 280–291.
[54] Q.S. Xu, M. Daszykowski, B. Walczak, F. Daeyaert, M.R. de Jonge, J. Heeres, L.M.H. Koymans, P.J. Lewi, H.M. Vinkers, P.A. Janssen, D.L. Massart, Chemometrics and Intelligent Laboratory Systems 72 (2004) 27–34.
[55] V. Svetnik, T. Wang, C. Tong, A. Liaw, R.P. Sheridan, Q. Song, Journal of Chemical Information and Modeling 45 (2005) 786–799.
[56] Y.-D. Cai, K.-Y. Feng, W.-C. Lu, Journal of Theoretical Biology 238 (2006) 172–176.
[57] P. He, C.-J. Xu, Y.-Z. Liang, K.-T. Fang, Chemometrics and Intelligent Laboratory Systems 70 (2004) 39–46.
[58] L. Breiman, J.H. Friedman, R.A. Olsen, C.J. Stone, Classification and Regression Trees, Wadsworth International Group, Minterey, CA, 1984.
[59] L. Breiman, Machine Learning 24 (1996) 123–140.
[60] B. Efron, R.J. Tibshirani, An Introduction to the Bootstrap, Chapman & Hall, 1993.
[61] R. Polikar, IEEE Circuits and Systems Magazine 6 (2006) 21–45.
[62] N. Cheze, J.M. Poggi, Journal of Statistical Research of Iran 3 (2006) 1–21.
[63] Y. Freund, R.E. Schapire, Journal of Computer and System Sciences 55 (1997) 119–139.
[64] H. Drucker, Improving regressors using boosting techniques, In Machine Learning — International Workshop then Conference, 1997.
[65] J.B. Copas, Journal of the Royal Statistical Society. Series B (Methodological) 45 (1983) 311–354.
[66] J. Shao, Journal of the American Statistical Association 88 (1993) 486–494.
[67] P. Laud, J. Ibrahim, Journal of the Royal Statistical Society. Series B 57 (1995) 247–262.
[68] M.H. Pesaran, The Review of Economic Studies 41 (1974) 153–171.
[69] S. Geisser, W. Eddy, Journal of the American Statistical Association 74 (1979) 153–160.
[70] A. Raftery, Sociological Methodology 25 (1995) 111–163.
[71] H. Bozdogan, Psychometrika 52 (1987) 345–370.
[72] J. Shao, Journal of the American Statistical Association 91 (1996) 655–665.
[73] Q.-S. Xu, Y.-Z. Liang, Chemometrics and Intelligent Laboratory Systems 56 (2001) 1–11.
[74] Q.S. Xu, Y.Z. Liang, Y.P. Du, Journal of Chemometrics 18 (2004) 112–120.
[75] L.I. Kuncheva, C.J. Whitaker, Machine Learning 51 (2003) 181–207.
[76] D.M. Hawkins, Journal of Chemical Information and Computer Sciences 44 (2004) 1–12.
[77] E.B. Kong, T. Dietterich, Proceedings of the Twelfth International Conference on Machinde Learning, 1995, pp. 313–321.
[78] D.S. Cao, Y.Z. Liang, Q.S. Xu, H.D. Li, X. Chen, A new strategy of outlier detection for QSAR/QSPR, Journal of Computational Chemistry, 2009, DOI:10.1002/jcc.21351.
[79] http://cran.r-project.org/web/packages/gbm/index.html.
[80] http://www.ics.uci.edu/ mlearn/MLRepository.html.
[81] http://software.eigenvector.com/Data/SWRI/index.html.
[82] http://www.swri.org/.
[83] http://www.r-project.org/.