Information Technology Journal 9 (3): 535-544, 2010
ISSN 1812-5638

Ensemble Numeric Prediction of Nearest-Neighbor Learning

Liang He, Qinbao Song, Junyi Shen and Zhen Hai
Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China

Abstract: The accuracy of various instances can hardly be ensured inherently by existing *k*-NN prediction schemes, suffering from the single *k* mechanism. To address this problem, an ensemble *k*-NN numeric prediction algorithm, Bs*k*-NN, is proposed. On the basis of boosting principle, a series of base *k*-NN predictors is constructed firstly by Bs*k*-NN. Then the instance-relevant combination of each individual base predictor presents the final composite estimate. The weight of each predictor changes adaptively with respect to the distinct features of different unknown instances. Attribute selection is introduced into Bs*k*-NN as well to optimize the proximity measurement and to perturb the stable training set. Since the requirement that various instances demand specific prediction schemes to match with has been taken into account thoroughly, the defect of fixed *k* nearest-neighbor prediction is rectified consequently. Moreover, Bs*k*-NN is compatible with datasets of any kinds of attributes, discrete, continuous or mixed. The experimental results on public datasets show that Bs*k*-NN outperforms the traditional *k*-NN prediction and the improvement is statistically significant according to the paired *t*-test.

Key words: *k*-NN learning, numeric prediction, boosting, ensemble

INTRODUCTION

Classification and numeric prediction are important research topics in data mining. A dataset in such tasks often consists of a set of descriptive attributes *x* and one special target attribute *y*. With this designation, the tuple (*x*, *y*) refers to an instance of the dataset. The value of *y* is confined to be categorical in classification, while it is continuous in numeric prediction. Sometimes the numeric prediction is also termed prediction for short. The *k*-Nearest-Neighbor (*k*-NN) algorithm is an instance-based learning scheme for both classifying and predicting applications, which classifies an instance in accordance with a majority vote of its *k* nearest neighbors or predicts it by the weighted sum of the estimates of those *k* neighbors.

Extensive investigations of *k*-NN learning have been carried out in the past, mainly focusing on the improvement of *k*-NN classifier. To calculate the distance precisely and pick up proper neighbors, a text categorization *k*-NN method with weight adjusting assigned different weights to the attributes of training set (Han *et al*., 2001). Based on Tabu search heuristic, hybrid approach of simultaneous attribute selection and attribute weighting for *k*-NN learning was presented by Tahir *et al*. (2007). The combination of multiple *k*-NN classifiers with different distance functions improved the performance of *k*-NN method as well (Yamada *et al*., 2006). Yang *et al*. (2005) proposed a novel *k*-NN approach with semantic distance calculation in terms of domain ontology. To accelerate the neighbor search in a high-dimensional metric space, the data was divided into small partitions and indexed by B^+-tree structure (Jagadish *et al*., 2005). Another reduced method introduced gray relational structure and instance pruning approach into *k*-NN, so that the classification can be performed by fewer training instances at lower cost (Huang, 2006).

The works about *k*-NN learning typically concentrate on the improvement of accuracy and the efficiency of neighbor search. Although these efforts have promoted *k*-NN in some senses, a common problem remains: a fixed number of neighbors are referenced in the classifying no matter what the unknown instance is. As a result, the distinct features of each individual instance tend to be ignored indeed by the existing *k*-NN methods, which make the overall accuracy hardly being ensured. Meanwhile, a similar situation arises in *k*-NN numeric prediction, suffering more from this weakness. Because the target attribute of the instances in numeric prediction may have numerous continuous values rather than several categories, compared with classification, the estimation turns to be extremely sensitive to the reference neighbors. In order to achieve better accuracy, we should try to resort to different *k*-NN prediction models and select

Corresponding Author: Liang He, Department of Computer Science and Technology,
School of Electronic and Information Engineering, Xi'an Jiaotong University,
Xianning West Road 28#, Xi'an 710049, Shaanxi, China Tel: 0086-29-83062244

appropriate quantity of neighbors thoroughly, especially when the unknown instance changes. In this study we accordingly attempt to resolve this problem incurred by the fixed *k* mechanism in numeric prediction.

In contrast with the fixed *k* prediction, if unknown instances consult varying numbers of neighbors with respect to the features of themselves, it is quite possible that each one obtains perfect accuracy. Whereas this intention also faces some challenges. First, it is quite impractical to realize personalized prediction just through a single *k*-NN predictor. Varying numbers of reference neighbors imply diverse schemes rather than a fixed one. It is necessary to initialize a series of predictors with substantial differences among each other. By operating on the unknown instance, respectively, the predictors produce independent outcomes and the weighted sum of them gives the final estimate. Furthermore, the weight distribution keeps flexible to underline the specific predictors and overlook the others according to the current unknown instance. However, straightforward learning on the original dataset could only establish one predictor; thus we have to provide sufficient training sets prior to the learning procedure, which brings the second problem. As an instance-based learning scheme, *k*-NN is more suited for smaller datasets in practice, taking computational cost into account. The challenge need to be addressed to construct diverse training sets with only one small original dataset available. The ensemble methodology puts forward a comprehensive solution that transforms weak learning methods into strong ones by a combination strategy. Presently, boosting is a prominent one of those practical ensemble methods.

AdaBoost, the classical algorithm of boosting family, is initially proposed for binary classification (Freund and Schapire, 1997). Successive AdaBoost.M1 and AdaBoost.M2 for multiple-label classification are also presented meanwhile. After then, some improved boosting algorithms based on ECOC (Error Correcting Output Codes) technique emerged, such as AdaBoost. OC (Schapire, 1997) and AdaBoost.MO (Schapire and Singer, 1999). In fact, boosting is applied not only to improve the classification schemes, but also to enhance the regression (numeric prediction). By transforming numeric prediction into multi-class classification and further binary classification, the representative algorithm AdaBoost.R (Freund and Schapire, 1997) submits an original recipe for regression problem. Some other boosting methods for regression have been proposed in succession (Breiman, 1999; Ridgeway *et al*., 1999; Kégl, 2003). Although, these methods have promoted numeric prediction theoretically in some ways, they could hardly serve in practice for the reason of computational complexity. Hence, we turn to a lightweight boosting regression method (Drucker, 1997) to facilitate the *k*-NN prediction.

According to boosting, training sets are sampled iteratively from the original dataset in terms of the instance weight distribution. Then, the base leaning model is built in turn on each sampling set. Within this procedure, those instances of worse accuracy receive heavier weights to maximize the chance of being picked up in the next sampling. Consequently, we are able to establish a series of base *k*-NN predictors in like manner. An ensemble *k*-NN prediction algorithm, Bs*k*-NN (Boosted *k*-NN), is therefore proposed in this study, by which radical changes have been made to the traditional *k*-NN learning. Meanwhile, attribute selection treatment is also introduced into Bs*k*-NN on account of unequal importance of different attributes to dissimilar instances and *k* values. The distance calculation on selected attribute subset optimizes the proximity measurement and brings more performance gains.

BOOSTING-BASED ENSEMBLE *k*-NN PREDICTION

Boosting: Boosting is an algorithm-independent method used to improve the accuracy of weak learners. According to this method, a series of base models can be built iteratively by particular data mining algorithm. While classifying or predicting an unknown instance, boosting combines all the outputs contributed by each individual model separately into final result through voting or averaging. The specific weight adjustment mechanism guarantees the training procedure of diverse base models. Firstly, a sampling training set is drawn with replacement from the original dataset, with regard to the weight distribution assigned in advance. Secondly, a base model is obtained from this sampling set and used to classify or predict all the instances in original dataset, whose performance controls the weight update entirely. Those instances that are classified or predicted incorrectly will have their weights increased, while those classified or predicted correctly will be assigned lower weights. After that, resampling from the original dataset produces another new training set for the next iteration. The subsequent base models always focus on the hard instances. By maintaining a measure of hardness with each instance, boosting provides an elegant way of generating a series of base models that complement one another. However, not all the weak learners could be improved by boosting under any circumstances, because this effective ensemble method only accepts unstable algorithms. There are different ways to realize boosting for classification and numeric prediction, the policies about weight adjustment and base models combination being the major aspects to distinguish the variants.

Ensemble model of *k*-NN prediction: The ensemble *k*-NN model consists of a sequence of base *k*-NN predictors, where each one is constructed by the following two steps:

- Sample a new training set from the original dataset
- Formulate a base *k*-NN predictor compatible with the sampling set

Bootstrap technique is chosen to perform the sampling process in step 1 owing to its facility. The instance weight determines the probability of being sampled. Before the training procedure, each instance in original dataset has an equally initial weight of 1/N, where N is the number of all the instances. After each iteration, all the weights will be recalculated by the performance of the latest base predictor.

The base predictor formulated in step 2 is described by two parameters: the first one *k* represents the number of nearest neighbors to learn from in predicting; the second one refers to the attribute subset corresponding to the given value of *k*, which achieves more accurate Euclidean distance measurement. Additionally, the ensemble *k*-NN learning also benefits from the unstable training sets perturbed by attribute selection. The main purpose of training on the sampling dataset is solely to find out the appropriate *k* and attribute subset. Obviously, the two parameters of various base predictors always differ from each other as the sampling sets keep changing in the iterations.

Figure 1 shows the procedure of building an ensemble of base *k*-NN predictors. Here D is the original training set and t is the number of iterations with an initial value of 1. Within the tth iteration, a new base *k*-NN predictor h_t is constructed on the sampling set D_t, drawn from D according to the weight vector w_t. Following the overall error, the weight vector w_t is recalculated to w_{t+1} To produce a series of base *k*-NN predictors $h_1 \ldots h_T$, the iteration repeats until some termination criterion is satisfied. We discuss the details of weight update and iteration control in the subsection of weight update.

Once the ensemble is obtained, by which an arbitrary unknown instance (x_p, y_p) could be predicted in accordance with the manner illustrated in Fig. 2. Each individual predictor h_t (t=1...T) produces an estimated value $\hat{y}_t$ to y_p, so that the weighted sum of all these values gives the final composite prediction. The coefficient C_t (t = 1...T) is specified in the subsection of combination of base predictors.

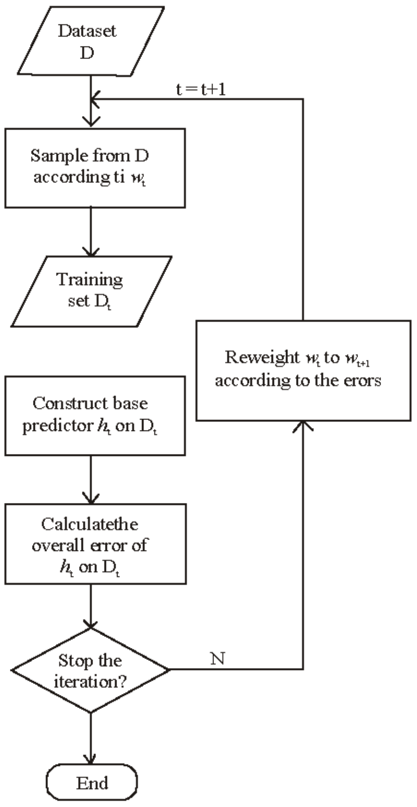
Dataset D
t = t+1
Sample from D according ti w_t
Training set D_t
Reweight w_t to w_{t+1} according to the erors
Construct base predictor h_t on D_t
Calculatethe overall error of h_t on D_t
Stop the iteration?
N
End

Fig. 1: Procedure of building an ensemble *k*-NN model

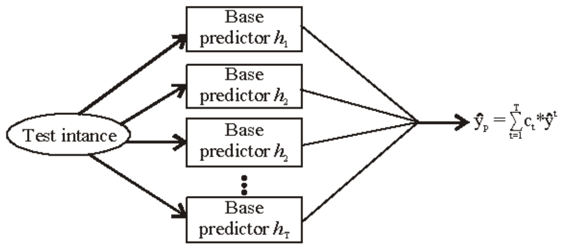

Fig. 2: Prediction by the ensemble *k*-NN model

Construction of base predictor h_t: The first essential for constructing a base predictor is the attribute selection within each iteration.

When searching for nearest neighbors, the distance measurement on appropriate attribute subset instead of on the entire set may help not only in evaluating the proximity explicitly, but also in reducing the computational cost in practice. There are many different kinds of techniques for attribute selection including wrapper, filter, etc. (Guyon and Elisseeff, 2003). Among these ones, a well-known method belonging to the wrapper technique, backward elimination, is chosen to implement the attribute selection for the training of base *k*-NN predictor, because of its simplicity and practicability. Following the backward elimination, each attribute is tentatively removed from the original dataset to formulate different candidate subsets. Then, the performance of the given learning algorithm (here is the *k*-NN) is evaluated on the subsets sequentially. The one producing the most performance gains is selected to be the new full set of attributes, on which the same process repeats continuously. If there is no more attribute available to make further improvements after being removed from the current subset, the selection comes to the end. Backward elimination is a standard greedy search procedure and guarantees to find out a locally optimal subset of attributes.

The second question for constructing base *k*-NN predictor is how to choose a proper value of *k*.

In order to predict a certain instance (x, y) in D_t during the training procedure, the contributions of the *k* nearest neighbors (x_1, y_1), ..., (x_k, y_k) are weighted by their distances to (x, y), respectively. Let d_i be the Euclidean distance between (x, y) and the ith neighbor (x_i, y_i), $i \in 1 \ldots k$, so that y is estimated as follows, where w_i denotes the weight of y_i:

$$d'_i = \frac{1}{d_i}, \quad i = 1 \ldots k \tag{1}$$

$$w_i = \frac{d'_i}{\sum_{i=1}^{k} d'_i}, \quad i = 1 \ldots k \tag{2}$$

$$\hat{y} = \sum_{i=1}^{k} w_i \cdot y_i \tag{3}$$

Hence, the error of (x, y) is:

$$\varepsilon = \left| \frac{y - \hat{y}}{y} \right| \tag{4}$$

It can be seen that the value of *k*, namely the number of referenced neighbors, dominates the estimated value of y. Thus, the mean error of D_t will also differ if the value of *k* alters. In other words, the best *k* and its attribute subset chosen by the training procedure should optimize the mean accuracy on D_t.

Although, the necessity of optimized *k* has been illustrated above, there is another outstanding question: What is the search space for seeking out the best value of *k* and its attribute subset?

In principle, the *k* neighbors of (x, y) ought to be found from the original dataset D. Unfortunately, the sampling rules of D_t shatter this notion. The training set D_t is drawn in fact from the original dataset D in terms of

```
Function BaseTrain( $D_t$ , D )
1  for  $k$=1…$k_{max}$//$K_{max}$ is a predefined integer
                of maximum neighbors
2     Predict each instance in $D_t$ in terms of Eqs. (3) and
      (4), with $k$ nearest neighbors. Calculate the mean error
      as min_k_error;
3     for  u=1…|U| //|U| is the number of attributes of U
4       U'=U-U(u); //remove the uth attribute from U
5       D'=$D_t$ (D');//only the attributes in U' are reserved
6       D'=D(D');
7       Predict each instance of $D'_t$ in terms of Eq. (3)
        and (4), where the neighbors are searched from
        D'. Calculate the mean error as $\varepsilon_k$ ($\mu$);
8   end for
9     $\varepsilon_k$ ($\mu$), $\mu$=1…|U|
10    if $\varepsilon_k$= min_k_error  then
11      U=$U_{min}$://$U_{min}$ is the attribute set corresponding to $\varepsilon_k$
12      min_k_error=$\varepsilon_k$, go to 3;
            //continue looking for better attribute subset
13    else
14      Terminate the attribute selection. The minimum
        mean error for $k$ is just min_k_error;
15    end if
16    err(k)=min_k_error;
17  end for
18  $K_t$=$\arg_k$min (err (k)), $k$=1…$K_{max}$;
19  Return base predictor $h_t$($K_t$, $A_t$, $E_t$). The parameters refer
    to $k$, attribute subset and error vector of $D$ in sequence.
```

Fig. 3: Realization of function BaseTrain (Dt, D)

weighted bootstrap. In this way any instance in D_t must be an element of D with no exception of (x, y), which means an error-free prediction of all the instances in D_t as long as *k* is set to be one. At that time the nearest neighbor is just the instance itself in D. Although such a base predictor h_t owns a significant accuracy, it is useless in practice due to the loss of generalization capability. To avoid this, the search space is switched to D' = D-(x, y).

Now we intend to realize a function BaseTrain(D_t, D) for the training of base *k*-NN predictor h_t. Letting U denote the whole attribute set of D, the function is given in Fig. 3.

Here, E_t is a relative error vector of the original dataset D predicted by the leave-one-out approach, according to the parameters of h_t. Furthermore, this vector is also used to reweight the instances of D by the end of each iteration.

Weight update: The training set D_{t+1} for the next base predictor h_{t+1} is sampled from the reweighted set D. Thus, the performance of a base predictor can be seen as reflection of its training set, so that the explicit weight update of D_t is a critical step to build the successful base predictor h_{t+1}.

When classifying an instance, the result holds nothing but two possibilities that the instance is classified correctly or incorrectly, namely the predicted value is either equal to the actual one or not. The accuracy of a classifier is evaluated by a straightforward count of the instances classified correctly. As a result, the weight update in classification by boosting principle becomes a cushy job as AdaBoost has done. On the contrary, the situation is more complicated in numeric prediction tasks, because the accuracy of a predictor can not be judged directly by an accumulated count like in classification. For this reason, the instance reweighting must be rectified to assort with the numeric prediction. The resort is to convert a prediction into infinite binary classification tasks, which determine whether the predicted values are greater or smaller than the actual one, then the boosting principle works. Although, various theoretical boosting algorithms for numeric prediction have been proposed in the past, most of them could hardly be applied practically for the sake of complexity. With respect to a feasible boosting regression algorithm (Drucker, 1997), the weight adjustment of D benefits from the inspirations as well.

Once the relative errors have been mapped into the interval [0, 1] via some kind of loss function, the weights of instances in D are updated in terms of the weighted average loss. The loss of an arbitrary instance (x, y) may be calculated by any one of the equations below:

$$L_1 = \frac{|\hat{y} - y|}{D} \tag{5}$$

$$L_2 = \frac{|\hat{y} - y|^2}{D^2} \tag{6}$$

$$L_3 = 1 - \exp(\frac{-|\hat{y} - y|}{D}) \tag{7}$$

where, $D = \sup|\hat{y}_i - y_i|$, i = 1…N. Obviously, the loss L_1 and L_2 always lie in [0, 1], so that the inequality $L_2 = L_1$ also holds to the same instance, which implies slight adjustment of L_2. In addition, L_3 only covers a sub internal of [0, 1] for the reason of $L_1 \in [0,1]$ and $0 \leq L_3 \leq 1\text{-}e^{-1}$. Besides the three ones, any other monotonically increasing function with a range of [0, 1] is also able to work as a loss functions. Following Eq. 8-11, the ith instance in D is consequently reweighted:

$$\bar{L}_t = \sum_{i=1}^{N} w_t(i)\, L_t(i) \tag{8}$$

$$\beta_t = \frac{\bar{L}_t}{1 - \bar{L}_t} \tag{9}$$

$$w_t'(i) = w_t(i) \cdot \beta_t^{1-L_t(i)} \tag{10}$$

$$w_{t+1}(i) = \frac{w'_{t+1}(i)}{\sum_{i=1}^{N} w'_{t+1}(i)} \tag{11}$$

In Eq. 8, the variables w_t(i) and L_t(i) denote the weight and loss of the ith instance, respectively upon the base predictor h_t. However, $\bar{L}_t$ is the weighted average loss of D produced by h_t. In Eq. 9, β_t is the confidence level of h_t. A smaller β_t refers to lower weighted mean error and better confidence level. Each instance weight is re-calculated and normalized by Eq. 10 and 11. According to the regenerated weight vector, the training set D_{t+1} for the next base predictor h_{t+1} is then sampled from D. Whenever the average loss $\bar{L}_t$ goes to exceed 0.5, the iteration should be terminated since the performance of the mapped binary classifier is even inferior to a random choice.

Combination of base predictors: So far, the ensemble of base *k*-NN predictors has been established. The last problem to be addressed is how to combine these ones to predict an unknown instance? As we have described previously, the weighted sum of each base predictor's individual estimate comes to be the final outcome. Thus, the weight distribution, i.e., the coefficients of base predictors, plays a significant role in the combination. The regular strategy designates fixed weight to each base predictor with regard to its performance in training, by which the weight distribution keeps immutable, no matter what the unknown instance is like. However, the reality is that each base predictor has its distinctive capability merely on certain kind of instances. The overall accuracy is therefore going to be undesirable, according to the fixed combination.

An instance-relevant combination approach is presented here for the ensemble, by which only such base predictors suitable for the current instance are dynamically assigned heavier weights with emphasis, whereas those not suitable are given lighter weights. In this manner, the weight distribution is adaptively refined with various incoming unknown instances. In contrast, the regular approach discussed above produces an instance-irrelevant distribution.

To combine the estimated values of an instance (x_s, y_s), its closest neighbor must be located from D in advance. Note that the distance is measured successively on varying attribute subsets offered by the base predictors. In such a way, each base predictor is able to determine one closest neighbor. The dissimilarity of all the attribute subsets makes these closest neighbors possibly different from each other. Letting (x_{near}, y_{near}) be the closest neighbor of (x_s, y_s) searched by h_t ($t \in 1 \ldots T$), we take advantage of this instance in formulating the coefficient C_t for h_t, based on the assumption that (x_s, y_s) may potentially achieve a satisfying accuracy as long as (x_{near}, y_{near}) could be predicted precisely by h_t. There are two concerns about the formulation of C_t: the error of

$(\boldsymbol{x}_{near}, y_{near})$ generated by h_t and the distance between $(\boldsymbol{x}_s, y_s)$ and $(\boldsymbol{x}_{near}, y_{near})$. The former may be taken easily from the parameter E_t of the previous function BaseTrain( ), the latter even available while locating $(\boldsymbol{x}_{near}, y_{near})$. Especially, C_t will be assigned a greater value in the case of both slighter error of $(\boldsymbol{x}_{near}, y_{near})$ and closer distance between $(\boldsymbol{x}_s, y_s)$ and $(\boldsymbol{x}_{near}, y_{near})$. Otherwise, C_t turns to be lower. According to various base predictors, the closest neighbors of different unknown instances will always change, so that the coefficients respond adaptively to underline the promising base predictors. Inevitably, the instance-relevant combination demands more time cost, compared with the instance-irrelevant one. However, it is worthwhile for higher overall accuracy.

We summarize the practical steps of instance-relevant combination as follows:

- Search out the closest neighbor of the unknown instance $(\boldsymbol{x}_s, y_s)$ from the original dataset D and use $d_{near}(t)$ to denote their distance calculated on the attribute subset of h_t, $t \in 1\ldots T$
- Assume that e_t represents the error of the closest neighbor by h_t, $e_t \in E_t$. We normalize the contribution of both $d_{near}(t)$ and e_t to be the coefficient C, by following equations:

$$C_t' = \frac{d_{near}(t) \cdot e_t}{|A_t|} \tag{12}$$

$$C_t' = \frac{1}{C_t'}, \tag{13}$$

$$C_t = \frac{C_t'}{\sum_{t=1}^{T} C_t'}. \tag{14}$$

The symbol A_t in Eq. 12 indicates the attribute subset of h_t, whereas $|A_t|$ refers to the number of attributes that A_t holds. The distance $d_{near}(t)$ has to be divided by $|A_t|$ in Eq. 12 due to inconsistent number of attributes among the base predictors.

Bs*k*-NN algorithm and its complexity: By now the critical issues of ensemble *k*-NN prediction have been fully addressed. We continue to realize a specialized numeric prediction algorithm named Bs*k*-NN (Boosted *k*-NN), a complete implementation presented in Fig. 4.

Bs*k*-NN algorithm consists of four modules: lines 1-3 initialize the variables used in Bs*k*-NN; lines 5-6 draw the sampling set and train a base *k*-NN predictor; the instance weights are updated with respect to the accuracy of h_t in lines 7-12; at last the test instance is predicted by the ensemble in lines 13-19.

```
Algorithm:Bsk-NN numeric prediction
Input:dataset D,test instance $(x_s,y_s)$
Output:estimated value $\hat{y}_s$
1  t=1;
2  Initialize the instance weights:
   $w_1$ (i)=1/N,i=1…N,N=|D|;
3  $\bar{L}_i$ =0;
4  while $\bar{L}_i \le 0.5$
   {
5    Draw the training set $D_t$ from D in terms of Bootstrap
     and weight vector $w_t$, $|D_t| = N$;
6    Build the base predictor $h_t$ on $D_t$:
         ( $K_t$, $A_t$, $E_t$)=BaseTrain( $D_t$, D );
7    Calculate the loss $L_t$(i) of each instance in D with Eq.
     (5), i=1…N;
8    Calculate the weighted average loss $\bar{L}_i$ with Eq. (8);
9    if $\bar{L}_i \le 0.5$ then
10     Reweight and normalize the instances in D with
       Eqs. (9)-(11);
11     t=t+1;
12   end if
   }
13 T=t-1; //T is the number of base predictors
14 for j=1…T
15   Calculate the estimated value y(j) of $(x_s,y_s)$ on $h_j$;
16   Calculate the coefficient C"(j) of $h_j$ with Eqs. (12)
     and (13);
17 end for
18 Normalize C"(j) to C(j) in terms of Eq. (14), j=1…T ;
19 $\hat{y}_s = \sum_{j=1}^{T} C(j) \cdot y(j)$
```

Fig. 4: Bs*k*-NN numeric prediction algorithm

It is necessary to examine the computational complexity of Bs*k*-NN, each iteration of which involves two phases of dataset sampling and base predictor construction. The cost of bootstrap sampling is negligible owing to its simplicity. For this reason, we restrict our attention to the latter phase, which is the dominating part of the cost. For a given *k* and its training set with d initial attributes, firstly we consider an extreme case that none of the attributes will be eliminated by the selection treatment. In other words, the overall accuracy on this training set of d attributes is actually better than that on any other set of d-1 attributes. Although, this situation rarely occurs, it does exist. To prove the best performance of the full attributes, the accuracy of any reduced training set with d-1 attributes must be verified, according to backward elimination. There are totally d possible datasets of d-1 attributes, generated by removing each attribute from the full set in sequence. Hence, intensive training procedures are required to repeat d+1 times on all the candidate datasets, i.e., d times on the sets of d-1 attributes and once on the set of d attributes. More generally, suppose a resulting set of d-i attributes after the selection, $i \in 0\ldots d\text{-}1$, where, i accounts for the number of attributes having been eliminated. In this situation, the training procedure needs to run 1+d+(d-1)+...+(d-i) = 1+(2d-i)(i+1)/2 times. It is clear that there are a total of 2^d-1 candidate datasets with different numbers of attributes. Among these ones, the probability that a dataset of d-i attributes is chosen to be the result is,

$$\binom{d}{d-i}\Big/(2^d-1)$$

Thus, the training procedure has to be performed S times on the average to find out the right attributes, where S is:

$$\begin{aligned} S &= \frac{\sum_{i=0}^{d-1}\binom{d}{d-i}\left(1+\frac{(2d-i)}{2}(i+1)\right)}{2^d-1} \\ &= \frac{\sum_{i=0}^{d-1}\binom{d}{i}}{2^d-1}+\frac{\sum_{i=0}^{d-1}\binom{d}{i}\frac{(2d-i)}{2}(i+1)}{2^d-1} \\ &= 1+\frac{\sum_{i=0}^{d-1}\binom{d}{i}\frac{(2d-i)}{2}(i+1)}{2^d-1} \end{aligned} \qquad (15)$$

Since, we have:

$$\begin{aligned} \sum_{i=0}^{d-1}\binom{d}{i}\frac{(2d-i)}{2}(i+1) &= \frac{1}{2}\sum_{i=0}^{d-1}\binom{d}{i}(2d+(2d-1)i-i^2) \\ &= \frac{1}{2}\left[2d\sum_{i=0}^{d-1}\binom{d}{i}+(2d-1)\sum_{i=0}^{d-1}\binom{d}{i}i-\sum_{i=0}^{d-1}\binom{d}{i}i^2\right] \\ &= \frac{1}{2}\left[\frac{3}{4}d^2 2^d+\frac{5}{4}d2^d-d^2-d\right] \end{aligned} \qquad (16)$$

substitute this result into Eq. 15:

$$\begin{aligned} S &= 1+\frac{1}{2}\left(\frac{\frac{3}{4}d^2 2^d+\frac{5}{4}d2^d-d^2-d}{2^d-1}\right) \\ &= \frac{3}{8}d^2+\frac{5}{8}d-\frac{1}{8}\frac{d^2-d}{2^d-1}+1 \end{aligned} \qquad (17)$$

The distance calculation takes the most computational cost within each training procedure and leads to a complexity of O (n^2), where, n is the number of instances in training set. Taking the Eq. 17 into account, the attribute selection treatment corresponding to a given *k* has the complexity O (Sn^2) = O (d^2n^2). Letting *k* refer to the predefined maximum value of *k*, i.e., the maximum number of reference neighbors, the complexity of an iteration is therefore O (Kd^2n^2). Then, the overall complexity increases to O (TKd^2n^2) for the purpose of establishing an ensemble of T base predictors. Compared with the training iterations, the cost of base predictors' combination need not be considered for its magnitude. The tradeoff between accuracy and complexity always stands there. However, the computational complexity of Bs*k*-NN may be possibly reduced by some techniques of accelerating neighbor search, optimizing attribute selection and so on. After all, this ensemble is a feasible solution to promoting the *k*-NN numeric prediction, especially for the datasets not too large.

EXPERIMENTAL RESULTS

Settings: Experiments on five public datasets are made in this section to evaluate the performance of Bs*k*-NN. These test datasets involve different fields about automobile, computer and cloud seeding, available from the website of WEKA (2008)-a kind of well-known open source software for data mining. We normalize the datasets initially and skip few instances for the reason of missing values. Bs*k*-NN is actually able to cope with this by an alternative distance measurement of overlooking those attributes. Because this treatment may interfere with effective attribute selection, we remove those instances. Besides, the absence of a very few instances will not bring adverse effects to the verification of Bs*k*-NN. After the preprocessing, the five datasets auto93, autoHorse, autoPrice, Cloud and CPU comprise 82, 159, 155, 108, 209 instances, respectively and 23, 26, 16, 7, 8 attributes in sequence.

Also we realize a traditional *k*-NN numeric prediction algorithm as a comparison, where the only parameter, the value of *k*, is determined by a leave-one-out learning process. The selected *k* produces best overall accuracy on the training set.

Methods: Cross validation is a principal performance evaluation approach for both classification and numeric prediction, overshadowing other ones like holdout, bootstrap and random subsampling. Hence, a standard 3-fold cross validation is applied in our experiments. All the test datasets are firstly split into 3 subsets of approximately equal size. Then, each subset is used for testing in turn and the reminders are used for training by both Bs*k*-NN and traditional *k*-NN. In spite of the usual partition of ten folds, a 3-fold cross validation is used here because of the concern that the large number of folds would produce smaller partitions, which potentially brings on biased assessment in virtue of insufficient test instances within each partition, especially for a small dataset and instance-based *k*-NN learning. Thus, 3-fold cross validation is preferred here. The randomness of both weighted sampling and cross-validation partitions may very likely incur different parameters for the base

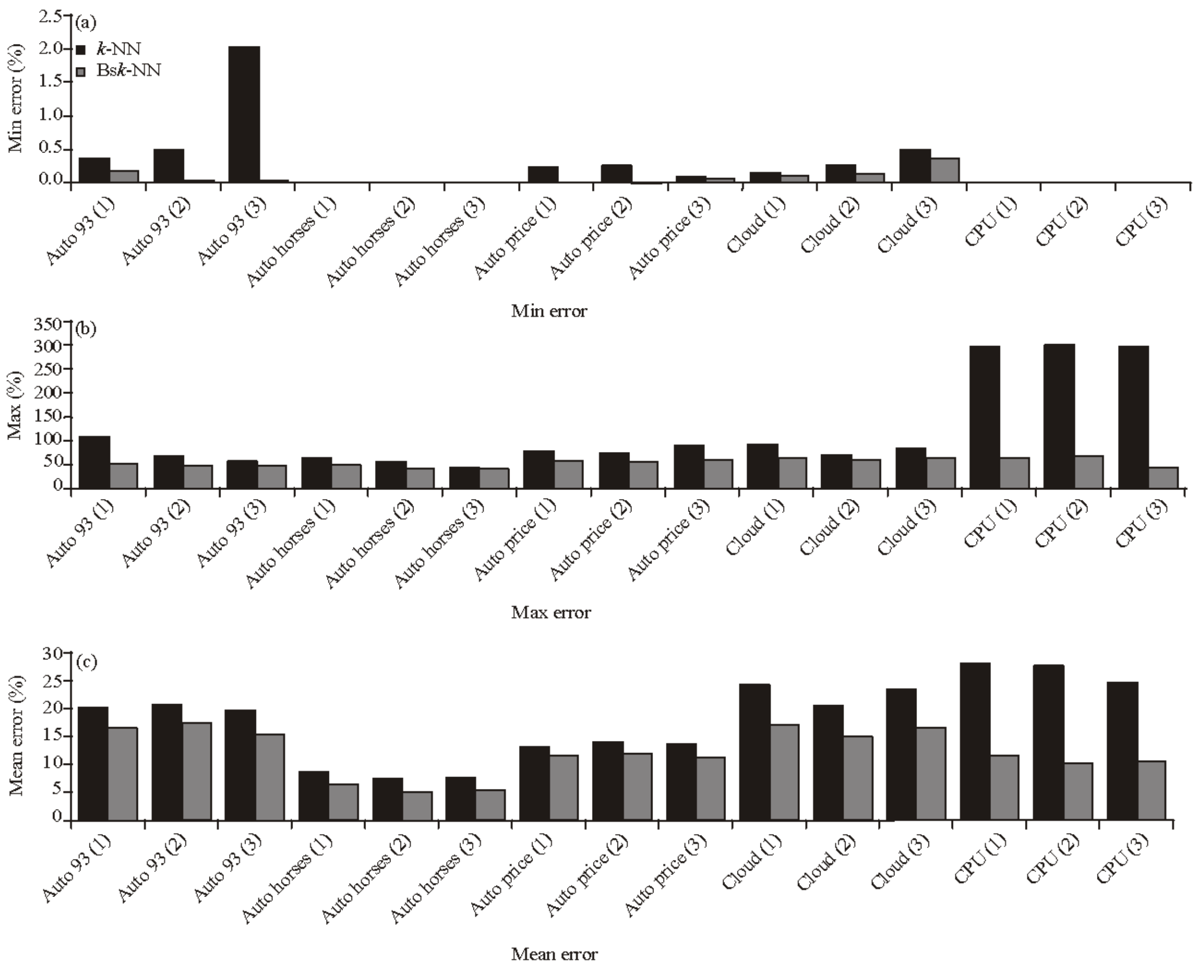

Fig. 5: Experimental results on public datasets of Bs*k*-NN and *k*-NN prediction (a) min error, (b) max error and (c) mean error

predictor and therefore inconsistent accuracies. Accordingly, the experiments are repeated three times on each dataset in case of occasional evaluation from only once results.

Results: The experimental results on five public datasets are illustrated by histograms in Fig. 5a-c. We mainly focus on the comparison among minimum, maximum and mean relative error of both Bs*k*-NN and traditional *k*-NN prediction. The number in the brackets followed the dataset name in Fig. 5 refers to the experiment round.

It can be seen that all the three kinds of relative errors have been decreased by Bs*k*-NN, more or less. The minimum error of *k*-NN on auto93 dataset is rather low, ranging from 0.35 to 1.99% within three test rounds, whereas Bs*k*-NN makes further efforts to cut it down to an interval of 0.02-0.17%. On the same dataset, traditional *k*-NN generates large maximum errors, even exceeding 100% in round 1, due to its defect of fixed *k* mechanism. Nevertheless, the maximum error is also reduced effectively by Bs*k*-NN. The most significant improvement is the reduced mean error, implying an overall effectiveness of Bs*k*-NN prediction. Similar results appear on the other four test datasets as well. Because some instances in autoPrice and CPU have duplicate values on target attribute, both *k*-NN and Bs*k*-NN could predict them precisely, so that the zero minimum errors in Fig. 5a can be explained. The most prominent promotion within the experiments emerges on CPU dataset: *k*-NN incurs tremendous maximum error lager than 300% in each test round while Bs*k*-NN depresses it significantly to be no longer greater than 70%. In the meantime, the overall performance is boosted strongly. As a result, the mean accuracy of Bs*k*-NN improves 11.00~63.54% beyond the traditional *k*-NN prediction on the five test datasets.

Table 1: p-value of paired t-test

	p-value		
Dataset	Round 1	Round 2	Round 3
auto93	0.0225	0.0209	0.0031
autoHorse	0.0011	0.0013	0.0041
autoPrice	0.0149	0.0018	0.0041
Cloud	0.0001	0.0002	0.0001
CPU	0.0000	0.0000	0.0000

So far we are not very sure yet that whether the observed improvement is caused by the solid difference between the two algorithms or just a chance effect in the estimation. Thus, an additional paired *t*-test continues to be performed. Let M_i denote the difference between the two estimated values of the ith instance predicted by *k*-NN and Bs*k*-NN respectively, i = 1...N. The differences $M_1...M_N$ are assumed to be independent of each other and M_i obeys normal distribution N (μ_M,σ_M). The right-tail *t*-test is taken as follows:

$$H_0: \mu M \leq 0, \qquad H_1: \mu M > 0,$$

where, the null hypothesis H_0 means the accuracy of the traditional *k*-NN is greater than or equal to Bs*k*-NN and the alternative hypothesis H_1 against H_0 shows the contrary. The p-values of paired *t*-test in each experimental round are given in Table 1. Apparently, all these figures are far less than 0.05. In particular, the figures of CPU dataset are very close to zero, which is consistent with the most significant performance improvement on the same dataset. Hence, the null hypothesis is rejected and the alternative one is accepted at a significance level of α = 0.05. We conclude that the improvement of Bs*k*-NN prediction is statistically significant on these test datasets.

CONCLUSIONS

As a feasible and effective lazy learning scheme, *k*-NN has been widely used in a mass of classification problems and relatively small numbers of numeric prediction tasks as well. However, the weakness of *k*-NN caused by a single *k* parameter prevents us from moving forward to improve its overall accuracy.

An ensemble *k*-NN prediction algorithm named Bs*k*-NN is proposed. According to this algorithm, diverse base *k*-NN predictors are established based on various training sets sampled from the original dataset. The composite outcome of an unknown instance is the weighted sum of all the estimated values predicted by each base *k*-NN predictor respectively, where those fit better for the unknown instance tend to be assigned heavier weights. It is conceivable that for most of the unknown instances there are suitable base predictors. Thus, the fixed *k* prediction without considering individual features of distinct instances will not happen. The performance improvement of *k*-NN leaning comes to be most likely credible. We have explored all the essential issues of Bs*k*-NN including ensemble *k*-NN model, base *k*-NN predictor training, instances reweighting and base predictors combination. Besides, we have also introduced attribute selection into Bs*k*-NN to perturb the stable datasets and to optimize the neighbor search.

We have made a series of experiments on different public datasets to compare the performance between Bs*k*-NN and traditional *k*-NN prediction. The results show that Bs*k*-NN actually outperforms the latter and the minimum, maximum and mean errors are all reduced. Additional paired t-test claims the statistical significance of the improvement.

Moreover, Bs*k*-NN prediction may be performed on all kinds of datasets, including attributes that are categorical, numeric, or mixed. It is another distinguishing characteristic of Bs*k*-NN.

ACKNOWLEDGMENTS

This study is supported by the National Science Foundation of China NSFC under the Grant NO.90718024 and the Science Foundation of Xi'an Jiaotong University (2009).

REFERENCES

Breiman, L., 1999. Prediction games and arcing algorithms. Neural Comput., 11: 1493-1517.

Drucker, H., 1997. Improving regressors using boosting techniques. Proceedings of the 14th International Conference on Machine Learning, July 08-12, Morgan Kaufmann Publisher Inc., pp: 107-115.

Freund, Y. and R.E. Schapire, 1997. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55: 119-139.

Guyon, I. and A. Elisseeff, 2003. An introduction to variable and feature selection. J. Mach. Learn. Res., 3: 1157-1182.

Han, E.H., G. Karypis and Y. Kumar, 2001. Text categorization using weight adjusted *k*-nearest neighbor classification. Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, April 16-18, Springer-Verlag, Hong Kong, China, pp: 53-65.

Huang, P.C., 2006. A novel gray-based reduced NN classification method. Pattern Recognation, 39: 1979-1986.

Jagadish, H.V., B.C. Ooi, K.L. Tan, C. Yu and R. Zhang, 2005. Distance: An adaptive B_+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst., 30: 364-397.

Kégl, B., 2003. Robust regression by boosting the median. Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop, Aug. 24-27, Springer Verlag, pp: 258-272.

Ridgeway, G., D. Madigan and T. Richardson, 1999. Boosting methodology for regression problems. Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics, Jan. 06-09, Morgan Kaufmann Publishers, pp: 152-161.

Schapire, R.E., 1997. Using output codes to boost multiclass learning problems. Proceedings of the 14th International Conference on Machine Learning, July 08-12, Morgan Kaufmann Publishers Inc., pp: 313-321.

Schapire, R.E. and Y. Singer, 1999. Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37: 297-336.

Tahir, M.A., P. Bouridane and F. Kurugollu, 2007. Simultaneous feature selection and feature weighting using Hybrid Tabu Search/*k*-nearest neighbor classifier. Pattern Recognation Lett., 28: 438-446.

WEKA, 2008. Weka Machine Learning Project. University of Waikato, New Zealand.

Yamada,T., K. Yamashita, N. Ishii and K. Iwata, 2006. Text classification by combining different distance functions with weights. Proceedings of the 7th ACIS International Conference on Software Engineering, Artificial, Intelligence, Networking and Parallel/Distributed Computing, June 19-20, IEEE Computer Society, pp: 85-90.

Yang, L., C. Zuo and Y.G. Wang, 2005. *k*-nearest neighbor classification based on semantic distance. Chinese J. Software, 16: 2054-2062.