Jan. 2005, Vol.20, No.1, pp.48–54 J. Comput. Sci. & Technol.

Adapt Bagging to Nearest Neighbor Classifiers

Zhi-Hua Zhou and Yang Yu

National Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, P.R. China

E-mail: zhouzh@nju.edu.cn

Received September 19, 2004; revised November 12, 2004.

Abstract It is well-known that in order to build a strong ensemble, the component learners should be with high diversity as well as high accuracy. If perturbing the training set can cause significant changes in the component learners constructed, then Bagging can effectively improve accuracy. However, for stable learners such as nearest neighbor classifiers, perturbing the training set can hardly produce diverse component learners, therefore Bagging does not work well. This paper adapts Bagging to nearest neighbor classifiers through injecting randomness to distance metrics. In constructing the component learners, both the training set and the distance metric employed for identifying the neighbors are perturbed. A large scale empirical study reported in this paper shows that the proposed BagInRand algorithm can effectively improve the accuracy of nearest neighbor classifiers.

Keywords bagging, data mining, ensemble learning, machine learning, Minkowsky distance, nearest neighbor, value difference metric

1 Introduction

Ensemble learning algorithms train multiple component learners and then combine their predictions. Since the generalization ability of an ensemble could be significantly better than that of a single learner, ensemble learning has been a hot topic in the past years[1]. It is well-known that in order to produce a strong ensemble, the component learners should preserve high diversity as well as high accuracy[2,3]. Although different algorithms may utilize different schemes to achieve the diversity, the most often used one is to perturb the training sets.

In one of the most famous ensemble learning algorithms, i.e., Bagging[4], many samples are generated from the original data set via *bootstrap sampling*[5] and then a component learner is trained from each of these samples, whose predictions are combined via *majority voting*. It is evident that the bootstrap sampling process is a specific mechanism for perturbing the training set. As Breiman indicated[4], for unstable learners such as decision trees and neural networks, perturbing the training set can cause significant changes in the component learners constructed, therefore Bagging can effectively improve the accuracy. However, for stable learners such as nearest neighbor classifiers, perturbing the training set can hardly produce diverse component learners, therefore Bagging can hardly work. So, although Bagging is a powerful ensemble learning algorithm, it is difficult to be applied to nearest neighbor classifiers to obtain good performance, notwithstanding nearest neighbor classifiers are very useful in real-world applications[6,7].

In this paper, a new variant of Bagging named BagInRand (Bagging with Injecting Randomness) is proposed, which constructs diverse component nearest neighbor classifiers through perturbing both the training set and the distance metrics employed in measuring the distance between different instances. A large scale empirical study involving twenty data sets, eight configurations of k values of nearest neighbor classifiers and nine configurations of ensemble sizes is reported in this paper, which shows that BagInRand can effectively improve the accuracy of nearest neighbor classifiers.

The rest of this paper is organized as follows. Section 2 proposes the BagInRand algorithm. Section 3 presents the empirical study. Section 4 concludes the paper.

2 BagInRand

In the middle of the 1990's, Krogh and Vedelsby[2] presented a famous equation $E = \overline{E} - \overline{A}$, where E is the generalization error of an ensemble, and $\overline{E}$ and $\overline{A}$ are the average generalization error and average ambiguity of the component learners, respectively. This equation clearly discloses that the more accurate and the more diverse the component learners are, the better the ensemble is. Therefore, in order to develop a good ensemble, the component learners should be with high diversity as well as high accuracy. Unfortunately, measuring diversity is not straightforward because there is no generally accepted formal definition, and so using it effectively for building better ensembles is still a problem to be solved[3]. Therefore, generating diverse but accurate component learners remains a trick at present, which is the key of most ensemble learning algorithms.

Bagging[4] achieves the diversity through perturbing the training set. For a given data set D, t samples $D_1, D_2, \ldots, D_t$ are generated by bootstrap sampling.

*Regular Paper

Supported by the National Outstanding Youth Foundation of China under Grant No.60325207, the Fok Ying Tung Education Foundation under Grant No.91067, and the Excellent Young Teachers Program of MOE of China.

Note that the data distribution held by a sample, say D_i, is usually different from that of D. Therefore the component learners trained from these samples can be anticipated to be diverse. Although this scheme is quite effective for unstable learners such as decision trees and neural networks, it is not so useful in dealing with stable learners such as nearest neighbor classifiers. This is not difficult to understand because as Breiman[4] indicated, given N training examples, the probability that the i-th training example is selected $0, 1, 2, \ldots$ times is approximately Poisson distributed with $\lambda = 1$. The probability that the i-th example will occur at least once is $1 - (1/e) \approx 0.632$. If there are t bootstrap samples in a 2-class problem, then a test instance may change classification only if at least one of its nearest neighbors in the training set is not in at least half of the t samples. This probability is given by the probability that the number of heads in t tosses of a coin with probability 0.632 of heads is less than $0.5t$. As t gets larger, this probability gets very small.

Since the scheme of perturbing training set does not work for nearest neighbor classifiers, in order to build a good ensemble, we have to try other schemes for introducing diversity. In fact, as indicated by Dietterich[1], besides perturbing training set there are several other schemes that could help improve the diversity, among which is the scheme of injecting randomness into the learning algorithm. Such a scheme has been applied by Kolen and Pollack[8] to neural networks through setting different initial weights for different networks, by Kwok and Carter[9] and Dietterich[10] to C4.5 decision trees through introducing some randomness to the selection of tests for splitting tree nodes, and by Ali and Pazzani[11] to relational learner FOIL through randomly selecting some good candidate rule conditions. These studies inspired us to try the scheme of injecting randomness in building ensembles of nearest neighbor classifiers.

Nearest neighbor classifier has no separate training phase, but when a new instance is given to be classified, the classifier will identify several neighboring training examples for this new instance and then usually use the *majority voting* method to produce the classification. The Euclidean distance is generally used to measure the distance between different instances described by continuous attributes. Actually, the Euclidean distance is a special case of Minkowsky distance, with order 2. That is, in the formal definition of Minkowsky distance shown in (1), p is set to 2 for obtaining Euclidean distance. Here $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two instances described by d-dimensional continuous attribute vectors.

$$\begin{aligned} \mathit{Minkowsky}_p(\boldsymbol{x}_1, \boldsymbol{x}_2) =& (|\boldsymbol{x}_{11} - \boldsymbol{x}_{21}|^p + |\boldsymbol{x}_{12} - \boldsymbol{x}_{22}|^p \\ &+ \cdots + |\boldsymbol{x}_{1d} - \boldsymbol{x}_{2d}|^p)^{1/p}. \end{aligned} \tag{1}$$

Through setting different values to p, different distance metrics can be obtained. In general, the smaller the value of p, the more robust the resulting distance metric to data variations; while the larger the value of p, the more sensitive the resulting distance metric to variations. That means these resulting distance metrics may help identify different vicinities of a given instance. So, it is possible to use them to help construct diverse component nearest neighbor classifiers.

However, Minkowsky distance can hardly deal with categorical attributes. Fortunately, VDM[12] can be a good complement. Let $N_{a,u}$ denote the number of training examples holding value u on categorical attribute a, $N_{a,u,c}$ denote the number of training examples belonging to class c and holding value u on a, and C denote the number of classes. The distance between two values u and v on a can be computed by a simplified version of VDM shown in (2), where q is usually set to 1 or 2.

$$\mathit{VDM}_q(u, v) = \sum_{c=1}^{C} \left| \frac{N_{a,u,c}}{N_{a,u}} - \frac{N_{a,v,c}}{N_{a,v}} \right|^q. \tag{2}$$

A new distance metric can be developed through combining the Minkowsky distance and VDM in the way as (3) shows, where the first j attributes are categorical while the remaining $(d - j)$ ones are continuous. Here the distance is called *Minkovdm* distance. It is evident that such a distance metric can deal with both continuous and categorical attributes, and both p and q can be perturbed to help construct diverse component nearest neighbor classifiers.

$$\begin{aligned} \mathit{Minkovdm}_{p,q}(\boldsymbol{x}_1, \boldsymbol{x}_2) =& (\mathit{VDM}_q(\boldsymbol{x}_{11}, \boldsymbol{x}_{21}) + \cdots + \\ & \mathit{VDM}_q(\boldsymbol{x}_{1j}, \boldsymbol{x}_{2j}) + |\boldsymbol{x}_{1,j+1} - \boldsymbol{x}_{2,j+1}|^p \\ & + \cdots + |\boldsymbol{x}_{1d} - \boldsymbol{x}_{2d}|^p)^{1/p}. \end{aligned} \tag{3}$$

Table 1. Pseudo-Code Describing the BagInRand Algorithm

```
BagInRand(x, D, Y, k, t, Ω1, Ω2)
  Input: x: Instance to label
         D: Data set
         Y: Label set {y1, y2, ..., yn}
         k: Number of neighbors to query
         t: Trials of bootstrap sampling
         Ω1: Distance order set {p1, p2, ..., pl}
         Ω2: Distance order set {q1, q2, ..., qm}
  for i ∈ {1..n} do count_i ← 0
  for i ∈ {1..t} do
    D_i ← BootstrapSample(D);
    p_i ← RandomSelect(Ω1);
    q_i ← RandomSelect(Ω2);
    Z ← Neighbor(x, k, D_i, Minkovdm_{p_i,q_i});
        %% k nearest neighbors of x are identified in D_i
        %% with the distance metric Minkovdm_{p_i,q_i}
    i* ← arg max_i  Σ_{y_i ∈ Y, z ∈ Z: z.label = y_i}  1;
        %% y_{i*} is the label held by majority instances in Z
    count_{i*} ← count_{i*} + 1
        %% y_{i*} receives the vote of a component learner
  end of for
  Output: x.label ← y_{arg max_i (count_i)}
        %% x.label is determined via majority voting
```

The BagInRand algorithm works through bootstrap sampling the original data set first. For each sample, it then randomly selects a value of p from a set Ω_1 and a value of q from a set Ω_2. The selected p and q values are then used to substantiate the Minkovdm distance defined in (3), which results in a component nearest neighbor classifier on the sample. Finally, the classification of a given instance is determined by majority voting of the predictions made by the component classifiers constructed on different samples. The empirical study reported later shows that such a simple algorithm works well on nearest neighbor classifiers.

The pseudo-code of BagInRand is shown in Table 1. Note that compared with the Bagging algorithm for nearest neighbor classifiers, BagInRand has two more parameters to set, that is, Ω_1 and Ω_2. But since Ω_2 is usually set to $\{1, 2\}$ and Ω_1 can be easily set to a random set of real values, the BagInRand algorithm is almost as easy as Bagging to use.

3 Empirical Study

A large scale empirical study is performed to test BagInRand. Twenty data sets from the UCI Machine Learning Repository[13] are used, where the original instances with missing values are removed. Information on the experimental data sets is tabulated in Table 2.

Table 2. Experimental Data Sets

Data set	Size	Attribute		Class
		Categorical	Continuous	
anneal	898	6	32	6
auto	159	15	10	7
balance	625	4	0	3
breast	277	0	9	2
breast-w	683	9	0	2
credit-a	653	6	9	2
credit-g	1,000	7	13	2
diabetes	768	8	0	2
glass	214	9	0	7
heart	270	13	0	2
heart-c	296	6	7	5
ionosphere	351	34	0	2
iris	150	4	0	3
lymph	148	3	15	4
segment	2,310	19	0	7
sonar	208	60	0	2
soybean	562	0	35	19
vehicle	846	18	0	4
vote	232	0	16	2
vowel	990	10	3	11

On each data set, ensembles of eight kinds of k-nearest neighbor classifiers are tested, where the value of k is set to 3, 5, 7, ..., 17, respectively. Nine ensemble sizes are tried, including 3, 5, 7, ..., 19. Since 10 times 10-fold cross validation is performed, in total 7,200 BagInRand ensembles are tested on each data set. For comparison, Bagging is tested in the experiments. Also, three other ensemble algorithms are included in the comparison. The first one is *InRand*, which is almost the same as BagInRand except that it does not utilize bootstrap sampling. That is, the InRand algorithm attempts to construct diverse component nearest neighbor classifiers through only injecting randomness to distance metrics. The second one is *BagInRandRAW*, i.e., BagInRand with Random Attribute Weights. As its name suggests, this algorithm utilizes all the mechanisms used in BagInRand and besides, for each component nearest neighbor classifier it assigns a random weight in the range of $[0.0, +1.0]$ to each input attribute. Therefore, different input attributes will have different degrees of impact in computing the distances, while attributes with zero weight will be excluded. The third one is *BagRAW*, i.e., Bagging with Random Attribute Weight, which is almost the same as BagInRandRAW except that it does not inject randomness to distance metrics. Ω_1 used by BagInRand, InRand and BagInRandRAW is set to $\{1, 2, 3\}$, while Ω_2 is set to $\{1, 2\}$.

Note that this paper focuses on adapting the Bagging algorithm to nearest neighbor classifiers through introducing diversity from different aspects. As for building ensembles of nearest neighbor classifiers, there are at least two effective algorithms. One is the *subspace* algorithm that trains different component learners from different attribute subspaces[14], which has also been presented as the MFS (Multiple Feature Subsets) algorithm in [15]; the other is the *kNN moderating* algorithm which controls the sampling process and marginalizes the *kNN* estimates using the Bayesian prior[16]. In fact, the BagRAW algorithm attempts to introduce some merits of the subspace algorithm to adapt the Bagging algorithm to nearest neighbor classifiers while BagInRandRAW attempts to further improve BagInRand with these merits. On the other hand, since the nearest neighbor classifiers used in this paper make predictions through majority voting instead of soft voting among training examples, merits of the *kNN* moderating algorithm are not exploited.

Since there are eight k values and nine ensemble sizes tested on 20 data sets, reporting on the average 10 times 10-fold cross validation results requires a table with 8,640 table entries ($8 \times 9 \times 20 \times 6$), or 180 figures ($9 \times 20$) each depicting for each data set the error of the six algorithms according to different k values under a fixed ensemble size and another 160 figures (8×20) each depicting for each data set the error of the six algorithms according to different ensemble sizes under a fixed k value. It is evident that it will be too tedious to present these detailed experimental results in the paper. So, only the summaries are given below. ①

If the 10 times 10-fold cross validation result of an ensemble is significantly better than that of a single k-

①The detailed experimental results can be accessed at http://cs.nju.edu.cn/people/zhouzh/zhouzh.files/publication/annex/jcst05-expdata.rar.

Table 3. Comparing Bagging with Single Nearest Neighbor Classifier (win/tie/lose)

k	$sz=3$	$sz=5$	$sz=7$	$sz=9$	$sz=11$	$sz=13$	$sz=15$	$sz=17$	$sz=19$
3	*3/1/16*	4/5/11	4/8/8	4/10/6	4/12/4	4/13/3	6/11/3	7/10/3	10/7/3
5	*1/5/14*	*2/6/12*	3/8/9	2/11/7	2/12/6	5/9/6	6/10/4	7/9/4	9/7/4
7	*0/6/14*	*0/9/11*	*1/10/9*	*0/14/6*	2/14/4	4/11/5	4/12/4	4/12/4	7/9/4
9	*2/7/11*	3/9/8	3/11/6	2/13/5	4/11/5	6/10/4	5/12/3	4/13/3	6/11/3
11	3/7/10	3/11/6	4/12/4	6/13/1	6/11/3	7/12/1	7/12/1	7/10/3	7/12/1
13	3/7/10	3/9/8	2/13/5	3/15/2	5/13/2	4/14/2	4/14/2	4/14/2	6/11/3
15	*3/5/12*	3/7/10	5/7/8	5/11/4	6/11/3	5/11/4	5/11/4	5/12/3	6/11/3
17	*3/4/13*	*1/10/9*	2/11/7	2/14/4	2/13/5	3/12/5	3/12/5	4/10/6	5/9/6

Table 4. Comparing InRand with Single Nearest Neighbor Classifier (win/tie/lose)

k	$sz=3$	$sz=5$	$sz=7$	$sz=9$	$sz=11$	$sz=13$	$sz=15$	$sz=17$	$sz=19$
3	6/10/4	8/8/4	9/7/4	10/3/7	9/3/8	8/8/4	9/6/5	9/9/2	8/8/4
5	7/4/9	6/8/6	9/5/6	9/6/5	9/6/5	9/6/5	9/5/6	9/5/6	7/9/4
7	6/9/5	6/7/7	8/4/8	8/5/7	9/4/7	8/6/6	8/6/6	7/9/4	11/4/5
9	8/7/5	10/5/5	9/6/5	8/7/5	7/7/6	10/6/4	10/6/4	11/5/4	12/4/4
11	9/6/5	10/5/5	9/6/5	10/5/5	10/5/5	**12/5/3**	10/6/4	11/5/4	12/3/5
13	8/7/5	9/8/3	9/9/2	8/8/4	11/4/5	11/4/5	9/6/5	11/4/5	10/6/4
15	9/6/5	8/9/3	10/6/4	11/6/3	10/7/3	11/6/3	11/4/5	12/4/4	12/4/4
17	8/8/4	8/8/4	8/7/5	10/5/5	10/7/3	10/6/4	**10/8/2**	10/6/4	10/6/4

Table 5. Comparing BagInRand with Single Nearest Neighbor Classifier (win/tie/lose)

k	$sz=3$	$sz=5$	$sz=7$	$sz=9$	$sz=11$	$sz=13$	$sz=15$	$sz=17$	$sz=19$
3	3/6/11	6/7/7	7/8/5	10/6/4	**10/9/1**	**8/11/1**	**12/7/1**	**11/7/2**	**12/6/2**
5	*1/7/12*	5/8/7	5/10/5	7/9/4	**8/11/1**	**11/7/2**	**13/5/2**	**14/3/3**	**13/5/2**
7	*3/4/13*	4/7/9	4/10/6	8/7/5	7/7/6	9/7/4	9/8/3	11/6/3	9/8/3
9	4/5/11	6/9/5	7/9/4	**10/8/2**	**11/7/2**	**10/9/1**	**12/6/2**	**12/6/2**	**12/6/2**
11	4/8/8	5/8/7	9/7/4	**9/10/1**	**10/10/0**	**10/10/0**	**10/9/1**	**12/7/1**	**11/8/1**
13	5/7/8	8/8/4	8/9/3	**11/8/1**	8/9/3	**12/7/1**	**10/8/2**	**12/6/2**	**11/7/2**
15	4/9/7	6/9/5	10/6/4	8/8/4	**10/9/1**	**11/8/1**	**11/8/1**	**12/7/1**	**11/7/2**
17	4/8/8	4/10/6	9/4/7	10/6/4	10/6/4	**11/8/1**	**11/7/2**	10/7/3	11/6/3

Table 6. Comparing BagInRandRAW with Single Nearest Neighbor Classifier (win/tie/lose)

k	$sz=3$	$sz=5$	$sz=7$	$sz=9$	$sz=11$	$sz=13$	$sz=15$	$sz=17$	$sz=19$
3	*1/0/19*	*1/2/17*	*1/4/15*	*3/2/15*	*3/4/13*	4/4/12	6/3/11	7/3/10	6/4/10
5	*0/1/19*	*1/0/19*	*1/0/19*	*1/1/18*	*1/3/16*	*1/2/17*	*2/3/15*	*3/3/14*	*2/4/14*
7	*0/0/20*	*0/1/19*	*0/1/19*	*0/2/18*	*0/4/16*	*1/5/14*	*2/3/15*	*2/4/14*	*2/6/12*
9	*0/1/19*	*0/2/18*	*1/4/15*	*1/4/15*	*1/4/15*	*1/4/15*	*1/6/13*	*1/6/13*	*1/6/13*
11	*0/1/19*	*1/4/15*	*1/3/16*	*2/2/16*	*1/4/15*	*1/4/15*	*1/6/13*	*1/8/11*	*3/4/13*
13	*1/1/18*	*1/4/15*	*2/3/15*	*4/3/13*	*3/3/14*	*3/5/12*	*3/5/12*	4/4/12	4/4/12
15	*1/2/17*	*1/4/15*	*3/3/14*	*3/3/14*	*4/3/13*	*3/4/13*	*5/2/13*	*5/2/13*	4/6/10
17	*1/3/16*	*2/4/14*	*1/5/14*	*3/3/14*	*4/2/14*	*2/5/13*	*4/2/14*	5/3/12	5/2/13

Table 7. Comparing BagRAW with Single Nearest Neighbor Classifier (win/tie/lose)

k	$sz=3$	$sz=5$	$sz=7$	$sz=9$	$sz=11$	$sz=13$	$sz=15$	$sz=17$	$sz=19$
3	*2/3/15*	*2/7/11*	5/5/10	6/5/9	8/6/6	8/6/6	8/9/3	12/4/4	10/6/4
5	*0/2/18*	*2/3/15*	*2/5/13*	4/8/8	6/8/6	4/10/6	7/8/5	7/9/4	8/7/5
7	*0/3/17*	*1/6/13*	5/4/11	5/7/8	5/5/10	4/7/9	6/6/8	10/2/8	6/6/8
9	*1/4/15*	*2/6/12*	4/7/9	6/3/11	5/8/7	7/5/8	8/6/6	7/5/8	6/7/7
11	*2/3/15*	*2/3/15*	4/5/11	5/6/9	4/7/9	5/7/8	6/5/9	6/7/7	8/5/7
13	*2/2/16*	*3/4/13*	3/6/11	4/9/7	5/6/9	5/6/9	6/7/7	7/8/5	6/8/6
15	*2/2/16*	*3/3/14*	*3/3/14*	*2/7/11*	4/6/10	4/7/9	5/5/10	7/4/9	7/4/9
17	*2/3/15*	*2/4/14*	*3/3/14*	*2/5/13*	*3/7/10*	3/9/8	4/8/8	4/6/10	7/4/9

nearest neighbor classifier on a data set (pairwise two-tailed t-test at 0.05 significance level), then the ensemble algorithm is deemed to win single classifier for one time. The win/tie/lose appearances of Bagging, InRand and BagInRand are summarized in Tables 3 to 7, where "sz" denotes ensemble size. Boldfaced and italicized table entries respectively indicate that the ensemble algorithm is significantly better or worse than single nearest neighbor classifier (sign test at 0.05 significance level).

Table 3 shows that Bagging is never significantly better than single classifier, and when the ensemble is relatively small ($sz \leqslant 7$), Bagging is often significantly worse than single classifier. This confirms Breiman's result[4] that Bagging cannot work on nearest neighbor classifiers. Table 4 shows that although InRand is never significantly worse than single classifier, it is rarely significantly better. This reveals that the scheme of injecting randomness to distance metrics does not work well

on nearest neighbor classifiers either. Table 5 shows that although there are rare cases where BagInRand is significantly worse than single nearest neighbor classifiers ($sz = 3$ and $k = 5$ or 7), it is often significantly better than them when $sz \geqslant 11$. This discloses an interesting fact, that is, although neither perturbing the training set nor injecting randomness to the learning algorithm works well solely on nearest neighbor classifiers, the combination of them can work well.

Unfortunately, Tables 6 and 7 show that neither BagInRandRAW nor BagRAW is significantly better than single nearest neighbor classifier. Even worse, BagInRandRAW seems almost always significantly worse than single classifier, so does BagRAW when the ensemble is relatively small ($sz \leqslant 7$). These observations suggest that simply introducing random attribute weight to Bagging is not helpful. We guess that this might due to the following reasons. Perturbing the input attributes, such as the subspace algorithm, can usually successfully generate quite diverse component *kNN* classifiers. However, the accuracy of the component classifiers is often not good. This might be because in an original training set there are usually some attributes irrelevant to the learning target, which might interfere with the learning on the relevant attributes, especially considering that *kNN* classification relies on the computation of distances in attribute spaces. When random attribute weight is directly exerted on many bootstrap samples, the influence of irrelevant attributes might increase to such a degree that the accuracy of the component learners is injured so much that the ensemble could not work. However, this conjecture has to be justified in the future. Since BagInRandRAW and BagRAW are not effective and BagInRand has not exploited random attribute weight, in the following analyses we do not consider BagInRandRAW and BagRAW further.

Tables 3 to 5 also suggest that relatively big ensemble sizes might be beneficial to all these three ensemble algorithms, because as ensemble size increases, the number of times that Bagging is significantly worse than single classifier tends to become smaller, the number of times that BagInRand is significantly better than single classifier tends to become bigger, and although InRand is almost always comparable to single classifier, the number of data sets it wins tends to become bigger. In addition, Tables 3 to 5 suggest that these ensemble algorithms be not sensitive to the value of k because in most cases the table entries in the same column look comparable.

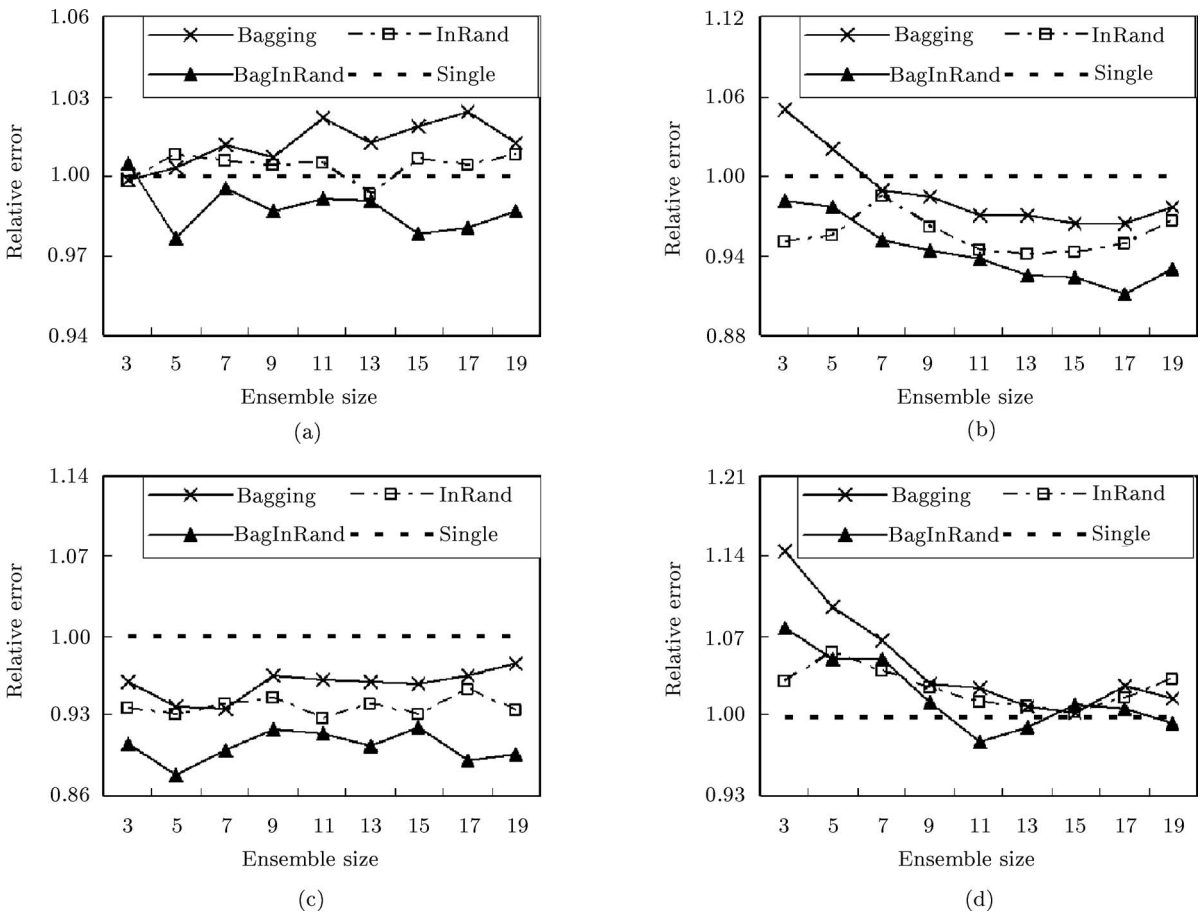

Fig.1. Impact of ensemble size for *credit-a*, *heart-c*, *sonar* and *soybean* ($k = 11$). (a) *credit-a*. (b) *heart-c*. (c) *sonar*. (d) *soybean*.

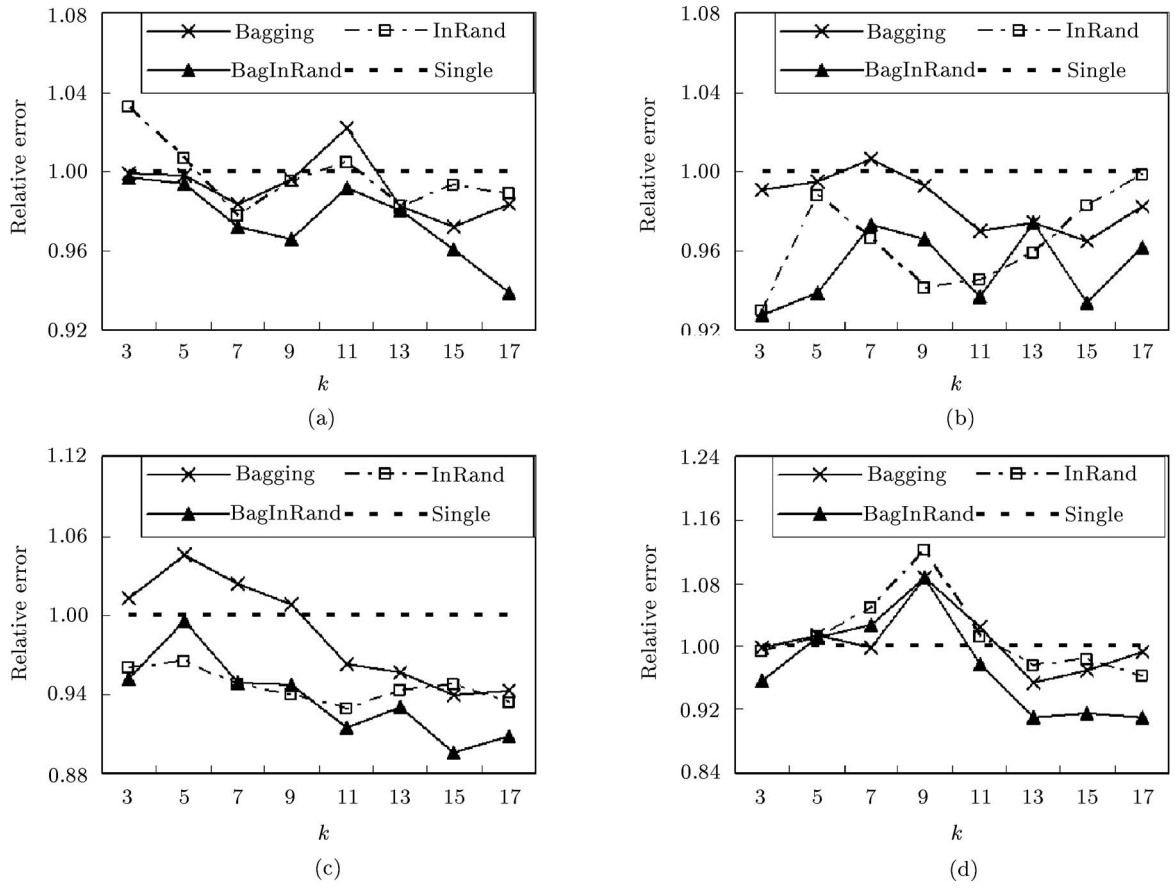

Fig.2. Impact of k value for *credit-a*, *heart-c*, *sonar* and *soybean* ($sz = 11$). (a) *credit-a*. (b) *heart-c*. (c) *sonar*. (d) *soybean*.

However, since above verdicts are made based on the observations of the overall performance of the algorithms on the twenty data sets, it is not clear whether or not they are applicable when only a concrete data set is concerned. So, the experimental results have been further analyzed. As mentioned before, it is hard to present all the detailed results and analyses, therefore here we only report the results on four typical data sets, i.e., large data set *credit-a*, median data set *heart-c*, small data set *sonar* and tiny data set *soybean*. The type of a data set is decided according to $COEF = \frac{size}{\#attributes \times \#classes}$, which considers the size of the data set relative to its dimensionality and its quantity of classes. The $COEF$ values of these four typical data sets are 21.77, 4.55, 1.73, and 0.85, respectively. The performance of Bagging, InRand and BagInRand are shown in Figs.1 and 2, where the relative errors are obtained through dividing the averaged errors of the ensembles by those of single nearest neighbor classifiers. For reference, the relative errors (1.0 according to the definition of relative error) of single classifiers are also depicted in these figures.

Fig.1 shows that relatively big ensemble size is beneficial for *heart-c* and *soybean* but not beneficial for *credit-a* and *sonar*. Therefore, although Tables 3 to 5 suggest that relatively big ensemble size be beneficial, it is only an overall tendency while for concrete data sets the influence of the ensemble size on the performance might be different. This also indicates that *selective ensemble* paradigm[17] which selects a subset of trained component learners instead of using all the component learners to comprise an ensemble can be a good choice. On the other hand, Fig.2 shows that the k value does have some influence on the performance of the ensembles, but the influence is quite complicated. In detail, the relative errors of the ensembles tend to decrease for *credit-a*, *sonar*, and *soybean* while fluctuate for *heart-c* as k increases. Therefore, although Tables 3 to 5 suggest that the overall performance of the ensemble algorithms be not sensitive to the value of k, the impact caused by k value might be different for concrete data sets.

Note that in Figs.1 and 2 when we study the impact of the ensemble size, the k value is fixed to the median of the tested values; while when we study the impact of the k value, the ensemble size is fixed to the median of the tested sizes. So, the analyses reported above are only about 1/8 (eight other ensemble sizes, seven other k values) of the analyses we have made on these four data sets. However, the other portions of our analyses (including analyses on other data sets) disclose similar facts, i.e., on concrete data sets the impact of the ensem-

ble size and the k value on these ensemble algorithms might be different.

4 Conclusion

In this paper, a new ensemble learning algorithm BagInRand is proposed, which is designed for building ensembles of nearest neighbor classifiers. This algorithm works through employing two schemes for constructing diverse component nearest neighbor classifiers, that is, perturbing the training set with bootstrap sampling and injecting randomness to distance metrics. A large scale empirical study shows that although this algorithm is simple, it can effectively improve the accuracy of nearest neighbor classifiers. This might be quite good because simple but effective algorithms might have better application potentials than complicated ones.

Dietterich[1] indicated that roughly there are four schemes for introducing diversity, that is, perturbing the training set, perturbing the input attributes, perturbing the output representation, and injecting randomness to the learning algorithm. The success of BagInRand suggests that although neither the first nor the fourth scheme is effective solely in building ensembles of nearest neighbor classifiers, their combination can be effective. We have tried to introduce into the combination the second scheme, i.e., perturbing the input attributes through assigning a random weight to each attribute, but unfortunately failed. It will be interesting to explore whether or not we can build ensembles of nearest neighbor classifiers through combining more than two of these schemes, and how if we can.

References

[1] Dietterich T G. Ensemble Learning. The Handbook of Brain Theory and Neural Networks, 2nd edition, M.A. Arbib (ed.), Cambridge, MA: MIT Press, 2002.

[2] Krogh A, Vedelsby J. Neural Network Ensembles, Cross Validation, and Active Learning. In *Advances in Neural Information Processing Systems* 7, Tesauro G, Touretzky D S, Leen T K (eds.), Cambridge, MA: MIT Press, 1995, pp.231–238.

[3] Kuncheva L J, Whitaker C J. Measures of diversity in classifier ensembles. *Machine Learning*, 2003, 51(2): 181–207.

[4] Breiman L. Bagging predictors. *Machine Learning*, 1996, 24(2): 123–140.

[5] Efron B, Tibshirani R. An Introduction to the Bootstrap. New York: Chapman & Hall, 1993.

[6] Aha D W. Lazy learning: Special issue editorial. *Artificial Intelligence Review*, 1997, 11(1–5): 7–10.

[7] Dasarathy B V. Nearest Neighbor Norms: NN Pattern Classification Techniques, Los Alamitos, CA: IEEE Computer Society Press, 1991.

[8] Kolen J F, Pollack J B. Back Propagation is Sensitive to Initial Conditions. In Advances in Neural Information Processing Systems 3, Lippmann R P, Moody J E, Touretzky D S (eds.), San Francisco, CA: Morgan Kaufmann, 1991, pp.860–867.

[9] Kwok S W, Carter C. Multiple decision trees. In *Proc. the 4th Annual Conference on Uncertainty in Artificial Intelligence*, New York, NY, 1988, pp.327–338.

[10] Dietterich T G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 2000, 40(2): 139–157.

[11] Ali K M, Pazzani M J. Error reduction through learning multiple descriptions. *Machine Learning*, 1996, 24(3): 173–202.

[12] Stanfill C, Waltz D. Toward memory-based reasoning. *Communications of the ACM*, 1986, 29(12): 1213–1228.

[13] Blake C, Keogh E, Merz C J. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html

[14] Ho T K. Nearest neighbors in random subspaces. In *Lecture Notes in Computer Science 1451*, Amin A, Dori D, Pudil P, Freeman H (eds.), Berlin: Springer, 1998, pp.640–648.

[15] Bay S D. Combine nearest neighbor classifiers through multiple feature subsets. In *Proc. the 15th International Conference on Machine Learning*, Madison, MI, 1998, pp.37–45.

[16] Alkoot F M, Kittler J. Moderating k-NN classifiers. *Pattern Analysis & Applications*, 2002, 5(3): 326–332.

[17] Zhou Z H, Wu J, Tang W. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 2002, 137(1-2): 239–263.

Zhi-Hua Zhou received the B.Sc., M.Sc., and Ph.D. degrees in computer science from Nanjing University, China, in 1996, 1998, and 2000, respectively, all with the highest honor. He joined the Department of Computer Science & Technology of Nanjing University as a lecturer in 2001, and is a professor and leader of the LAMDA group at present. His research interests are in machine learning, data mining, pattern recognition, information retrieval, neural computing, and evolutionary computing. In these areas he has published over 40 technical papers in refereed international journals or conference proceedings. He has won the Microsoft Fellowship Award (1999), the National Excellent Doctoral Dissertation Award of China (2003), and award of the National Outstanding Youth Foundation of China (2004). He is on the editorial boards of Artificial Intelligence in Medicine (Elsevier), Knowledge and Information Systems (Springer), and International Journal of Data Warehousing and Mining (Idea Group). He served as the organising chair of the 7th Chinese Workshop on Machine Learning (2000), program co-chair of the 9th Chinese Conference on Machine Learning (2004), and program committee member for numerous international conferences. He is the vice chair of the Artificial Intelligence & Pattern Recognition Society of China Computer Federation, a councilor of Chinese Association of Artificial Intelligence (CAAI), the chief secretary of CAAI Machine Learning Society, and a member of IEEE and IEEE Computer Society.

Yang Yu received the B.Sc. degree in computer science from Nanjing University, China, in 2004. He has won some awards such as China Computer World Scholarship (2004) and Scholarship for outstanding undergraduates. Now he is a member of the LAMDA group and will pursue his M.Sc. degree at the Department of Computer Science & Technology of Nanjing University since September 2005, to be supervised by Prof. Zhi-Hua Zhou. His research interests are in machine learning and evolutionary computing.