Using A Neural Network to Approximate An Ensemble of Classifiers

Xinchuan Zeng and Tony R. Martinez

Computer Science Department
Brigham Young University
Provo, Utah 84602

Abstract. Several methods (e.g., Bagging, Boosting) of constructing and combining an ensemble of classifiers have recently been shown capable of improving accuracy of a class of commonly used classifiers (e.g., decision trees, neural networks). The accuracy gain achieved, however, is at the expense of a higher requirement for storage and computation. This storage and computation overhead can decrease the utility of these methods when applied to real-world situations. In this paper, we propose a learning approach which allows a single neural network to approximate a given ensemble of classifiers. Experiments on a large number of real-world data sets show that this approach can substantially save storage and computation while still maintaining accuracy similar to that of the entire ensemble.

Keywords: approximator, bagging, boosting, ensemble of classifiers, neural networks

1. Introduction

Several methods (e.g., *Bagging, Boosting*) of constructing and combining an ensemble of classifiers have been proposed to improve the accuracy of learning algorithms. Over the last several years they have drawn increased interest and research in the fields of machine learning and neural networks.

The typical structure of this type of methods is an ensemble (group) of diverse component classifiers. A diverse ensemble is created by assigning each component classifier a different training set, which is usually derived from the original training set by resampling or other techniques. To classify an input pattern, the predictions of all component classifiers are combined, by weighted or unweighted voting, to make a final classification decision.

Differences between these approaches center mainly on how to create a diverse ensemble and how to combine the component classifiers. The most extensively studied approaches are *Bagging* by Breiman [1] and *Boosting* by Freund and Schapire [2].

Effectiveness of these methods, especially *Bagging* and *Boosting*, has been demonstrated empirically. Breiman [1] has shown that Bagging can increase accuracies of CART decision trees on several real and artificial domains. Quinlan [3], Bauer and Kohavi [4] demonstrated the capability of Bagging and Boosting to improve C4.5 decision trees, based on a large number of data sets. Also, Maclin and Opitz [5] have shown that Bagging and Boosting can improve the accuracy of neural networks.

Despite their obvious advantages, these methods have at least three weaknesses: (1) increased storage, (2) increased computation, and (3) decreased comprehensibility.

The first weakness, *increased storage*, is a direct consequence of the requirement that all component classifiers, instead of a single classifier, need to be stored after training. The total storage depends on the size of each component classifier itself and the size of the ensemble (number of classifiers in the ensemble). This concern was particularly addressed by Dietterich [6].

The second weakness is *increased computation*: to classify an input query, all component classifiers (instead of a single classifier) must be processed, and thus it requires more execution time.

The last weakness is *decreased comprehensibility*, as stressed also by Dietterich [6]. With involvement of multiple classifiers in decision-making, it is more difficult for users to perceive the underlying reasoning process leading to a decision.

The work by Margineantu and Dietterich [7] has addressed the issue of increased storage by using a pruning technique. In this approach, they applied several pruning algorithms to remove some decision trees from an ensemble constructed by boosting. The empirical results show that the storage requirement can be reduced by 60% to 80% without seriously decreasing accuracy (maintaining about 80% of the accuracy gain achieved by boosting).

Domingos [8] applied another approach to address this issue through generating a base learner by providing it with a new training set composed of a large number of examples generated and classified according to the ensemble. The empirical results using $C4.5$ *rules* as the base learner and bagging as the ensemble method show that the approach retains on average 60% of the accuracy gains obtained by bagging relative to a single run of $C4.5$ *rules*. This type of methodology of transforming (approximating) one form of hypothesis representation to another one has been previously studied by Craven and Shavlik. In their studies, they extracted rules [9] and tree-structured representations [10] from a trained neural network.

In this paper, we propose an improved approach to reduce the negative effects of the weaknesses (with emphasis on the first two) of ensemble classifiers. Our basic strategy is to approximate a given ensemble of classifiers by an alternative representation that needs much less storage, while still maintaining the same or similar accuracy as the ensemble. Specifically, we propose an approach to train a multilayer neural network to approximate the behavior of the ensemble.

We construct such an approximator by training a neural network to learn from a given ensemble. A *pseudo training set* is sampled based on the distribution of the original training set and is labeled by querying the ensemble. The neural network is then trained on this pseudo training set. One distinct feature of our approach is to use a (probability) *vector* to label the class of each item in the pseudo training set. The amplitude of the (probability) component for a class is proportional to the number of votes received from the ensemble. In contrast, previous approaches by Craven and Shavlik [9, 10] and by Domingos [8] used a *scalar* to label the class – the class that receives the maximum votes from the ensemble. We suggest that a vector can capture richer information about the decision making process of the ensemble, without losing potentially important details of the distribution of the voting.

The feasibility of this approximation approach has been tested on 16 domains using Bagging as the ensemble method. The results show that the constructed neural network has the capability of saving a considerable amount of memory (81%) and computation, while still keeping a high percentage of agreed predictions and a similar accuracy (retains 94% accuracy gain) to the given ensemble. These results compare favorably to those reported by Margineantu and Dietterich [7] and by Domingos [8]. We also compare the vector labeling against the scalar labeling for pseudo training sets. The simulation results demonstrate significant advantages of vector labeling over scalar labeling.

The rest of paper is organized as follows. Section 2 briefly reviews Bagging and Boosting, with emphasis on Bagging. Section 3 describes our approach of approximating an ensemble by a neural network. Section 4 presents experimental results on several domains. Section 5 summarizes the work and outlines the plan for future research.

2. Ensembles of Classifiers

In the following discussion we use the following conventions for the purpose of convenience and clarity. An individual classifier in an ensemble is called a *component classifier*. The form of the representation for component classifiers, such as decision tree and neural network, is called the *base representation*. The learning algorithm that outputs a component classifier based on a training set, such as C4.5, CART, and Back-propagation, is called the *base learning algorithm*. A composite classifier that bases its decision on the voting of the component classifiers is called a *voting classifier*. A method that constructs and combines an ensemble of component classifiers, such as Bagging and Boosting, is called a *ensemble method*. A classifier that approximates a voting classifier is called an *approximator*.

Although there are numerous existing methods of constructing and combining an ensemble of classifiers, we only describe here the two most popular ones – *Bagging* and *Boosting*. We will put more emphasis on the description of Bagging since it will be used in the experiments.

Bagging [1] uses a resampling technique, *bootstrap* [11], to generate multiple diverse data sets from the original training set, and then uses them as training sets to construct an ensemble of diverse component classifiers.

The *Bootstrap* method works in the following ways to generate a new training set (*bootstrap replicate*). In each iteration, it randomly (with uniform probability distribution) draws an example from the original training set and puts it into a new training set. But after each drawing, a copy of the drawn example is put back into the original training set (i.e., *with replacement*) – the original training set thus remains intact after each drawning. This process repeats until the new training set contains the same number of examples as the original training set. In a new training set, some examples in the original training set may appear multiple times while some may not appear at all.

With an ensemble of resampled training sets, Bagging then applies a base learning algorithm to construct an ensemble of component classifiers. To classify an unlabeled pattern, it uses simple (unweighted) voting to combine the predictions of the component classifiers and assigns the output of the pattern to the class receiving the maximum votes. If two or more classes have same maximum vote, then one is randomly chosen one.

Boosting [2] assigns each example an adjustable weight which is proportional to the probability of being drawn as a training example. Classifiers constructed by Boosting are order-dependent: how to construct a new classifier depends on the accuracies of previous ones. This dependence is reflected by adjusting the weights of all examples (increasing the weights of misclassified examples) after each classifier is constructed. After each round of weight adjusting, the training of a new classifier will focus more on those examples misclassified by previous classifiers. Unlike Bagging, Boosting applies a weighted voting to combine classifiers, assigning a larger voting weight to classifiers with higher accuracy.

3. Using a Neural Network as an Approximator

The objective of an approximator for an ensemble of classifiers is to have the capability of classifying future input examples in the same way the ensemble does. Ideally, we hope that an approximator can form the same hypothesis as that of the ensemble in the feature space.

We formulate the task of constructing an approximator in the following form:

Given: (1) a training set $\mathbf{S} = \{(\mathbf{x_i}, y_i)|i = 1, 2, ...N\}$ as a base for training component classifiers, where $\mathbf{x_i}$=(x_{i1}, x_{i2}, $...x_{iM}$) is an input vector (M is the number of features), and $y_i \in \{g_1, g_2, ...g_C\}$ (C is the number of classes) is the labeled class for $\mathbf{x_i}$; (2) a voting classifier η based on an ensemble of component classifiers $\{\eta_j|j = 1, 2, ...K\}$ obtained from training sets derived from $\mathbf{S}$.

Task: Construct an approximator ξ that approximates η so that $\xi(\mathbf{x_i}) = \eta(\mathbf{x_i})$ ($i = 1, 2, ...N$) with a high probability.

To complete this task, we use the following three steps: (1) sample a pseudo training set based on the distribution of the original training set, (2) label the pseudo training set by querying the given voting classifier, and (3) train a neural network using the labeled pseudo training set.

3.1 Sampling a Pseudo Training Set

A *pseudo training set*, unlike the original training set, is an artificial training set generated by sampling and labeling based on the original training set. A generated pseudo training set needs to reflect major characteristics of the original training set. The main reason for using a pseudo training set (instead of the original training set) is its unlimited size – we can sample as many examples as we need. In contrast, the size of the original training set $\mathbf{S}$ is fixed and sometimes could be very small. A large training set is often necessary for approximating a voting classifier, which typically forms a more complex hypothesis than each component classifier [6]. To learn a more complex hypothesis, more training examples are needed to probe the decision boundaries defined by a voting classifier, thus making it easier for an approximator to form a similar hypothesis.

We assume that the test set will be drawn from the same underlying distribution as that in the original training set. Therefore, we require that a sampled pseudo training set must reflect the input distribution of the original training set (thus it is also able to reflect that of the test set). To satisfy this requirement, the following method was applied to sample the pseudo training set based on the statistics of the original training set. Sampling for nominal and continuous features are handled separately.

For a *nominal* feature, the marginal distribution for each nominal value is first computed from the original training set. Sampling for this feature is then based on this marginal distribution.

For a *continuous* feature, its value is first discretized into p ($p = 10$ in our experiments) equally-sized intervals between the minimum and the maximum value, and the marginal distribution for each interval is then computed from the original training set. When sampling this feature, a random number is generated to determine which interval it falls into according to this marginal distribution. Then it interpolates its value within this interval according to the difference between the random number and the probabilities at the two ends of this interval – a sampled feature thus remains in a continuous spectrum.

3.2 Labeling a Pseudo Training Set

The *Labeling* process assigns a class label to each example in a pseudo training set sampled by the method described above. Since our goal is to let a neural network learn from a voting classifier such that it will behave similarly to the voting classifier, we can use the voting classifier as an *oracle* to answer a query and assign a class label to each example in the pseudo training set.

When a voting classifier is applied as an oracle to label an example, a returned class label may take one of two formats: a *scalar*, representing the class with the maximum votes, or a *vector*, representing the class probability vector whose components are proportional to the received votes.

Craven and Shavlik [9, 10] used a scalar to represent a returned class label from an oracle when they extracted rules and tree-structured representations from a trained neural network. This labeling format was also used by Domingos [8].

In contrast, we choose a vector as the class label for a sampled pseudo training set. The reason for this choice is that a class probability vector is able to capture more information about decisions made by a voting classifier, while a scalar only keeps the class with the maximum vote and, as a result, loses certain voting information. The performances of the two labeling formats are compared experimentally in the next section.

When a vector is used as a class label, a labeled pseudo training set can be expressed as (* represents *pseudo*):

$$\mathbf{S}^* = \{(\mathbf{x_i^*}, \mathbf{p_i^*}) | i = 1, 2, ...N^*\} \tag{1}$$

where $\mathbf{x_i^*} = (x_{i1}^*, x_{i2}^*, ...x_{iM}^*)$ is the feature vector and $\mathbf{p_i^*} = (p_{i1}^*, p_{i2}^*, ...p_{iC}^*)$ is the class probability vector. p_{ij}^* is the probability for class j for example i and is proportional to the number of votes V_{ij} received from the K component classifiers:

$$p_{ij}^* = V_{ij}/K \tag{2}$$

Obviously, the probabilities are normalized: $\sum_{j=1}^{C} p_{ij}^* = 1$, because the total number of votes received by example i is K: $\sum_{j=1}^{C} V_{ij} = K$.

3.3 Class Distribution in Sampling

An additional requirement is that a sampled pseudo training set must have the same *class* distribution as that in the original training set. To obtain the information about the class distribution in the original training set, we first applied the voting classifier to relabel the original training set, and then used the new labels (instead of the original labels) to calculate the class distribution. The reason for the relabeling is to allow a constructed approximator to perceive a class distribution in the same way as the voting classifier, so that it can have more similar behavior to the voting classifier.

Before creating a pseudo training set S^* with size N^* (number of examples in S^*), we assign a quota $N_j^* = D_j N^*$ to class j $(j = 1, 2, ...C)$ where D_j is the percentage of examples with class j in the relabeled original training set. S^* is initially empty, and the sampled examples are added into it using the following procedure until its size reaches N^*. For each sampled (unlabeled) example x^* using the sampling method described previously, it is first labeled by the voting classifier and is assigned a vector label $\mathbf{p}(x^*)$; then the class label with maximum vote $q(x^*)$ is determined from $\mathbf{p}(x^*)$: $q(x^*) = argmax_j\{p_j(x^*)|(j = 1, 2, ...C)\}$. If the number of already sampled examples with class label $q(x^*)$ in S^* is smaller than N_q^* (i.e., the quota for class q is not yet filled), then x^* is added to S^*; otherwise, x^* is not added to S^*. This process continues until all quotas N_j^* $(j = 1, 2, ...C)$ are filled up. Thus, S^* includes exactly N_j^* examples with class label j for each class j.

After sampling and labeling a pseudo training set $\mathbf{S}^*$ by the methods described above, we construct an approximator by training a neural network using the *backpropagation* learning algorithm. There is a major difference between our training approach and standard backpropagation: we use the class probability distribution (a vector) as the training target while the standard approach uses a class label (a scalar) as the training target. For this reason, we have adapted the standard procedure to a form suitable for using a vector as the training target, and details of the new procedure are discussed in the next section.

4. Experiments

We have tested the approximation approach introduced in the last section on 16 data sets drawn from the UCI machine learning repository [12]. We applied *stratified 10-fold cross-validation* [13, 14] to estimate the accuracies of classifiers in the experiments. We conduct 5 cross-validations for each data set and average the accuracy over the 5 cross-validations. The data sets chosen for testing reflect a wide range of different types of domains. The sizes of the data sets tested are in the range from small to medium (those with very large sizes were not tested due to the length of computing multiple 10-fold cross-validations).

4.1 Training Component Classifiers

Each component classifier is a single-hidden-layer neural network trained by standard *backpropagation* learning algorithm. In the experiments, each ensemble includes 10 component classifiers. Then Bagging is applied to combine them onto a voting classifier.

A well-known problem for backpropagation is that a trained neural network is prone to *over-fitting* the training data. To reduce the effect of this problem, we use a validation set to monitor training progress.

Before training, we partition the training set $\mathbf{S}$ into two subsets: sub-training set $\mathbf{S_1}$ (2/3 of $\mathbf{S}$) and sub-validation set $\mathbf{S_2}$ (1/3 of $\mathbf{S}$). $\mathbf{S_2}$ serves to monitor the performance of the trained network on $\mathbf{S_1}$ for avoiding over-fitting and selecting the number of hidden nodes. The partition of $\mathbf{S}$ is *stratified* – both $\mathbf{S_1}$ and $\mathbf{S_2}$ have approximately the same class distributions as $\mathbf{S}$ – so that monitoring the error on $\mathbf{S_2}$ can make a better esitmation to the performance of the network trained on data set $\mathbf{S_1}$. The training procedure includes the following two phases.

In the *first* phase, we use $\mathbf{S_1}$ as the training set and $\mathbf{S_2}$ as the validation set. The network has a single hidden layer and the initial number of hidden nodes is 5. All initial weights are randomly chosen from -0.5 to 0.5 and are stored for future possible retrievals. The learning rate is 0.2 and the momentum is 0.5. We used the validation set $\mathbf{S_2}$ to determine the network configuration (number of hidden nodes) in the following manner. After training the neural network with the initial number (5) of hidden nodes, we increment the number of hidden nodes by 5. After training a network with a fixed number of hidden nodes is finished, its error rate (using squared error as the error function) on the validation set will be compared with that of previously saved best configuration. If the error rate is improved (smaller), we then assume that the current number of hidden nodes (network configuration) is better that the previously saved best one, and save the current configuration as the best one.

In the *second* phase of training, the whole training set $\mathbf{S}$ ($\mathbf{S_1}$ plus $\mathbf{S_2}$) is used to train a network with the optimal number of hidden nodes determined above.

4.2 Training Approximator

The training procedure and parameter setting for a neural network approximator are the same as those for training the component neural networks but with the following two differences.

First, the target is a vector ($\mathbf{p_i^*}$) for an approximator while it is a scalar (y_i) for a component neural network. We have adapted the standard backpropagation to the following form so that a vector can be used as the training target. For training example ($\mathbf{x_i^*}, \mathbf{p_i^*}$), we set the target value of output node j to be the component p_{ij}^* of vector p_i^*, and then use the standard procedure to propagate the error ($o_{ij} - p_{ij}^*$) (where o_{ij} is the output value of the $j\underline{th}$ output node when example i is fed into the network).

Second, the error calculation on the validation set is different. For a component neural network, the output values of all output nodes are set to either 0 or 1 (with the output node with highest output value being 1 and the rest of them being 0), and then compared to the target values (also either 1 or 0). For a given pattern, the classification result is either correct or incorrect, which corresponds to an error of either 0 or 1. So the error is discrete ($0/N_2$, $1/N_2$,..., N_2/N_2) where N_2 is number of examples in the validation set.

However, for a neural network approximator, the error for example i and output node j is $(o_{ij} - p^*_{ij})$, and thus the error rate is continuous. This type of error often continuously drops slowly even though the network has already reached a minima, while a discrete error rate usually does not vary in this case.

To deal with this difference and avoid unnecessary training, we empirically set a non-zero error reduction threshold (0.001/*epoch* in our experiments) to decide if the training is in progress when monitoring the error on the validation set. Only when the average error reduction between two monitorings is larger than the threshold, does the training be considered to be in progress. (In contrast, the corresponding threshold is 0.0/*epoch* when training the component neural networks; that is, training is considered to be in progress as long as there is any amount of error decrease.)

4.3 Experimental Results

Table 1 shows the test-set accuracies and percentage of agreed predictions of different classifiers on 16 data sets drawn from the UCI data repository [12].

Table 1: Test-set accuracy and percentage of agreed predictions [1]

	Accuracy(%)				*Agreed (%)*		
Data Set	*Std*	*Appr^s*	*Appr^v*	*Bag*	*Std*	*Appr^s*	*Appr^v*
Bupa	68.6	69.1	70.5	71.2	83.2	88.1	91.2
Flag	62.2	62.5	63.5	63.4	80.9	83.6	89.1
German	73.4	74.2	75.0	75.3	85.1	86.4	88.9
Glass	62.1	62.1	64.6	66.2	82.1	83.7	89.8
Hayes	79.8	80.6	81.1	81.3	93.5	98.9	99.8
Heart(c)	82.7	83.8	84.0	84.0	93.0	95.6	96.0
Heart(h)	80.5	82.7	81.4	81.2	91.1	93.3	92.7
Heart(s)	83.3	83.7	84.4	83.9	92.1	95.2	95.9
Horse	65.3	65.6	66.2	68.5	78.6	81.3	84.6
Iris	92.9	95.9	96.0	95.7	94.3	98.5	99.2
Led7	71.3	72.6	73.0	73.0	89.9	98.1	98.8
Led17	65.6	66.5	69.0	68.6	78.4	81.5	87.5
Pima	75.8	76.3	76.9	76.5	90.9	95.1	95.4
Voting	95.5	95.9	96.0	95.7	98.3	98.9	98.9
Wave21	81.5	82.6	84.2	83.1	87.7	89.5	92.6
Zoo	93.6	94.7	94.7	94.7	97.3	98.0	98.2
Average	**77.1**	**78.1**	**78.8**	**78.9**	**88.5**	**91.6**	**93.7**

In the table, *Std* is the standard classifier that uses a single neural network trained on the original training set. *Appr* is the approximator that is a single neural network trained on a pseudo training set of size 500. *Appr^s* represents the scalar target format used by Craven and Shavlik [9, 10] and by Domingos [8], while *Appr^v* represents the vector target format proposed in this work. *Bag* is the bagging classifier based on voting from an ensemble of 10 component neural networks, trained on the data sets resampled from the original training set.

The results show that Bagging improves all of the 16 data sets. *Appr^v* approximates *Bag* better than *Std* and *Appr^s*. The average accuracy difference between *Bag* and *Appr^v* ($78.9\% - 78.8\% = 0.1\%$) is much smaller than that between *Bag* and *Std* ($78.9\% - 77.1\% = 1.8\%$) and that between *Bag* and *Appr^s* ($78.9\% - 78.1\% = 0.8\%$).

[1]Notations: heart(c) – heart (cleveland); heart(h) – heart (hungarian); heart(s) – heart (statlog).

Aside from *accuracy*, another parameter to measure the quality of an approximator is *percentage of agreed predictions*. The percentage of agreed predictions of classifier A to classifier B is defined as the percentage of test examples to which A predicts exactly the same class (whether correct or not) as B. It reflects the similarity of classifier A to classifier B with regard to classification behavior.

The last three columns in Table 1 show the percentages of agreed predictions of *Std*, *Appr*^{s} and *Appr*^{v} with respect to *Bag*. They show a similar trend as the accuracies.

The results in Table 1 show advantages of *Appr*^{v} over *Appr*^{s} in terms of both accuracy and percentage of agreed predictions. This confirms our analysis in the last section: using a vector as the target format can capture more voting information than using a scalar, and thus enable a trained approximator be more accurate in approximating a voting classifier.

One measurement for the quality of an approximator is the *relative performance* (*RP*) with respect to the ensemble [7] (or retaining rate of accuracy gain obtained by the ensemble [8]), defined as the ratio of the accuracy gain obtained by the approximator and the accuracy gain obtained by the ensemble:

$$RP = \frac{(ACC^{Appr} - ACC^{Std})}{(ACC^{Esem} - ACC^{Std})} \tag{3}$$

where ACC^{Std}, ACC^{Esem}, and ACC^{Appr} are accuracies achieved by standard, ensemble (bagging or boosting), and approximator classifiers respectively.

The *RP* value obtained by Margineantu and Dietterich [7] (tested on 10 data sets) is about 80%, and that by Domingos [8] (tested on 26 data sets) is about 60%. From Table. 1, the average *RP* value for *Appr*^{v} is 94%, which compares favorably to those obtained by Margineantu and Dietterich and by Domingos.

We also studied the hypothesis complexities of different classifiers in the experiment. When a hypothesis is represented in the framework of a single-hidden-layer neural network, its complexity can be measured by the number of hidden nodes. Table 2 compares the average number of hidden nodes for *Std*, *BagC*, *BagE* and *Appr*^{v}. *BagC* is the average number of hidden nodes for one *component* classifier, and *BagE* is the average number of hidden nodes for the whole *ensemble* (including 10 component classifiers).

Each value in the table is obtained by first averaging over 10 different folds for each 10-fold cross-validation (*BagC* needs an extra average over 10 different component classifiers for each fold), and then averaging over 5 different 10-fold cross-validations.

The results show that *Appr*^{v} has a more complex hypothesis (more hidden nodes) than *Std* and *BagC*. They are expected results because a voting classifier can form a more flexible hypothesis than any of individual component classifiers, as analyzed previously.

Another important issue is how much storage can be saved by using an approximator to replace a bagging classifier. The last column in Table 2 shows the storage ratio of an approximator to the whole ensemble (i.e., *Appr*/*BagE*). The results show a significant reduction in storage: an approximator only requires a small fraction (19% on average) of the storage for the whole ensemble, and hence is capable of saving a substantially large amount of storage space (81% on average).

The technique by Margineantu and Dietterich [7] pruned 60% to 80% of the decision trees in an ensemble. The approach proposed here can attain higher savings in storage, because it only needs to store a single network while the pruning approach needs to store multiple decision trees (potentially it may still need a large number of trees, since the number is proportional to the size of the original ensemble).

Table 2: Number of hidden nodes & Storage

Data Set	*Std*	*BagC*	*BagE*	*Appr*^{v}	*Stor*
Bupa	9.3	8.5	85	9.6	11%
Flag	7.9	9.0	90	16.4	18%
German	7.1	9.8	98	10.3	11%
Glass	8.5	8.9	89	17.2	19%
Hayes	5.6	5.8	58	16.3	28%
Heart(c)	7.2	7.1	71	13.2	19%
Heart(h)	6.3	7.0	70	12.8	18%
Heart(s)	6.8	7.0	70	12.0	17%
Horse	8.9	8.9	89	10.9	12%
Iris	5.3	5.3	53	11.3	21%
Led7	9.2	9.6	96	25.8	27%
Led17	11.1	11.0	110	21.9	20%
Pima	7.3	8.1	81	8.7	11%
Voting	6.2	5.9	59	13.1	22%
Wave21	7.3	7.5	75	13.6	18%
Zoo	8.7	7.3	73	20.8	28%
Average	**7.7**	**7.9**	**79**	**14.6**	**19%**

Domingos [8] measured the capability of memory saving using the ratio of the storage required by the approximator over the storage required by the standard classifier (smaller ratio means more memory saving). The value of the ratio ranges from 2 to 6 for the result of Domingos [8]. From Table 2, the average ratio in our experiment is 1.9, which is smaller than that reported by Domingos [8].

In the experiment, only 10 component classifiers were included in an ensemble. We would expect a larger storage reduction if there are more component classifiers in an ensemble. The reason is that only a single neural network approximator is needed regardless of the number of classifiers in an ensemble, while the storage of the pruning approach depends on the number of classifiers.

This approach can also substantially reduce the requirement for computation. To classify an example, it only needs to run a single neural network approximator in stead of running multiple component classifiers in an ensemble. The amount of reduction in computation is proportional to that for storage.

We have also studied how the size of a pseudo training set can affect the quality of a trained approximator. The results show that this approximating approach typically only needs a pseudo training set with a relatively small size to achieve a high degree of agreed predictions and thus is quite efficient The results shown in above tables are based on pseudo training sets with only 500 examples.

5. Summary

In summary, we have presented an approach to construct a neural network as an approximator for a voting classifier. We have also carried out the experiments to test this approach on 16 data sets.

The experimental results demonstrate that the proposed approach is efficient in constructing an accurate approximator of a bagging classifier. The constructed neural network is capable of saving 81% memory while still maintaining

94% of the accuracy gain by the ensemble. These results compare favorably to those using other methods [7, 8].

One important factor for the observed good performance by this approach is the adoption of the scheme using a vector as the target format for class labeling. The simulation results show that using a vector as the target format performs significantly better than that using a scalar [8, 9, 10] for the particular task of learning from an ensemble.

Although we have only tested this approach using bagging as the ensemble method and using neural networks as the component classifiers, it is straightforward to extend this approach to fit other ensemble methods and component classifiers. For ensemble methods using simple (unweighted) voting, regardless of the type of component classifier, this approach can be applied directly without modifications.

For a ensemble method using a weighted voting (e.g., boosting), this approach needs to be modified. The only required modification is to use a different method to label the class probability vector in a pseudo training set; more specifically, V_{ij} in Eq. (2) needs to take into account the voting weights of component classifiers when calculating the votes received for class j.

References

[1] L. Breiman, Bagging predictors, Machine Learning, Vol. 24, pp. 123-140, 1996.

[2] Y. Freund and R. Schapire, Experiments with a new boosting algorithm, in Proc. of the Thirteenth National Conference on Machine Learning, pp. 148-156, Morgan Kaufmann, 1996.

[3] J. R. Quinlan, Bagging, boosting, and c4.5, in Proc. of the Thirteenth National Conference on Artificial Intelligence, pp. 725-730, AAAI/MIT Press, 1996.

[4] E. Bauer, and R. Kohavi, An empirical comparison of voting classification algorithms: bagging, boosting and variants, Machine Learning, Vol 36, pp. 105-139, 1999.

[5] R. Maclin, and D. Opitz, An empirical evaluation of bagging and boosting, in Proc. of the Fourteenth National Conference on Artificial Intelligence, pp. 546-551, AAAI/MIT Press, 1997.

[6] T. G. Dietterich, Machine-learning research - four current directions, AI Magazine, Winter: pp. 97-136, 1997.

[7] D. D. Margineantu and T. G. Dietterich, Pruning adaptive boosting, in Proc. of the Fourteenth International Conference on Machine Learning, pp. 98-106, 1997.

[8] P. Dominggos, Knowledge acquisition from examples vis multiple models, in Proc. of the Fourteenth International Conference on Machine Learning, pp. 211-218, 1997.

[9] M. W. Craven and J. W. Shavlik, Learning symbolic rules using artificial neural networks, in Proc. of the 10th International Conference on Machine Learning, pp. 73-80, Amherst, MA. Kaufmann, 1993.

[10] M. W. Craven and J. W. Shavlik, Extracting tree-structured representation from trained networks, in D. S. Touretzky, M. C. Mozer and M. Hasselmo (ed.) Advances in Neural Information Processing System 8, pp. 24-30, MIT Press, 1996.

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

[12] C. J. Merz and P. M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html, 1996.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees, Wadsworth International Group, 1984.

[14] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in Proc. of the International Joint Conference on Artificial Intelligence, pp. 1137-1143, 1995.