Available online at www.sciencedirect.com
SCIENCE DIRECT
Pattern Recognition 39 (2006) 595–607
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/patcog

Classification by evolutionary ensembles

Xiao Wang, Han Wang*

School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, BLK s1-b4-12, Singapore 639798, Singapore

Received 18 June 2004; received in revised form 15 September 2005; accepted 15 September 2005

Abstract

This paper is about building an ensemble of classifiers each of which is trained based on a particular weighting over the training examples (a weighting is a set of weights associated with the examples). The task concerns search in a tremendous weighting space. In this view we propose to incorporate a genetic algorithm (GA). It performs a wide yet efficient search for appropriate weightings (chromosomes). The difference from a traditional GA is that all the weightings throughout evolution will be exploited to form the final ensemble, not just the best weighting. Our algorithm is tested on the UCI benchmark data sets and used to design a face detection system. Robust and consistently accurate classification is experienced. Comparative results with two other algorithms, i.e. AdaBoost and Bagging, are also given.

Keywords: Multiple classifier system; Genetic algorithms; Evolutionary learning; Classifier combination; AdaBoost; Bagging

1. Introduction

Suppose a feature space $\mathbb{X}$, binary label space $\mathbb{Y} = \{-1, +1\}$ and a set of training examples $S = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{X}, y_i \in \mathbb{Y}, i = 1, \ldots, N\}$. This paper is about constructing not a single classifier, but a number of classifiers $\{h_t(.)\}$ whose individual options on an input feature $\mathbf{x}$ are voted to derive a consensus decision:

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right), \quad (1)$$

where $\sum_{t=1}^{T} \alpha_t = 1$. It is expected that the subspaces of features misclassified by different $h_t(.)$ do not necessarily overlap; therefore, the *ensemble classifier* $H(.)$ try to exploit the different behavior of component classifiers $h_t(.)$ to improve the accuracy and reliability of the overall learning system. There are also hopes that if some of the component classifiers fail, the overall system can recover the error.

Numerous algorithms for ensemble classification have been developed (see Ref. [1] for a brief survey), among which those based on manipulating the training examples are receiving extensive attention. The intuitive idea is that some examples may be hard to classify while others are easy (just imagine some are close to the decision boundary and others are far away from it). One may consider assigning different weights to them when they are input into a learning algorithm (e.g. decision tree or neural network), in other words, training the classifier using weighted examples. A set of weights associated with the examples is known as a *weighting*. Since the optimal weighting is not known a priori, it is suggested that a "base" learning algorithm is called repeatedly, each time feeded with the training examples associated with a different weighting. Each time the learning algorithm is called, it generates a new classifier. Thus after many rounds, one shall obtain a number of classifiers and may combine them into an ensemble classifier. Some comparative experiments have shown that the resulting ensemble could consistently outperform a single classifier [2–4].

More formally, at time t each example $(\mathbf{x}_i, y_i)$ is assigned a weight $w_t(i)$ to form a non-negative weighting $\mathbf{w}_t = [w_t(1), \ldots, w_t(N)]$ over S (subject to $\sum_{i=1}^{N} w_t(i) = 1$). When trained with a weighted training set $\{S; \mathbf{w}_t\}$, a learning algorithm generates a particular classifier $h_t(.)$. The base learning algorithm is invoked for T times, each time with

* Corresponding author. Tel.: +65 6790 4506; fax: +65 6792 0415.
E-mail address: hw@ntu.edu.sg (H. Wang).

0031-3203/$30.00 © 2005 Published by Elsevier Ltd on behalf of Pattern Recognition Society.
doi:10.1016/j.patcog.2005.09.016

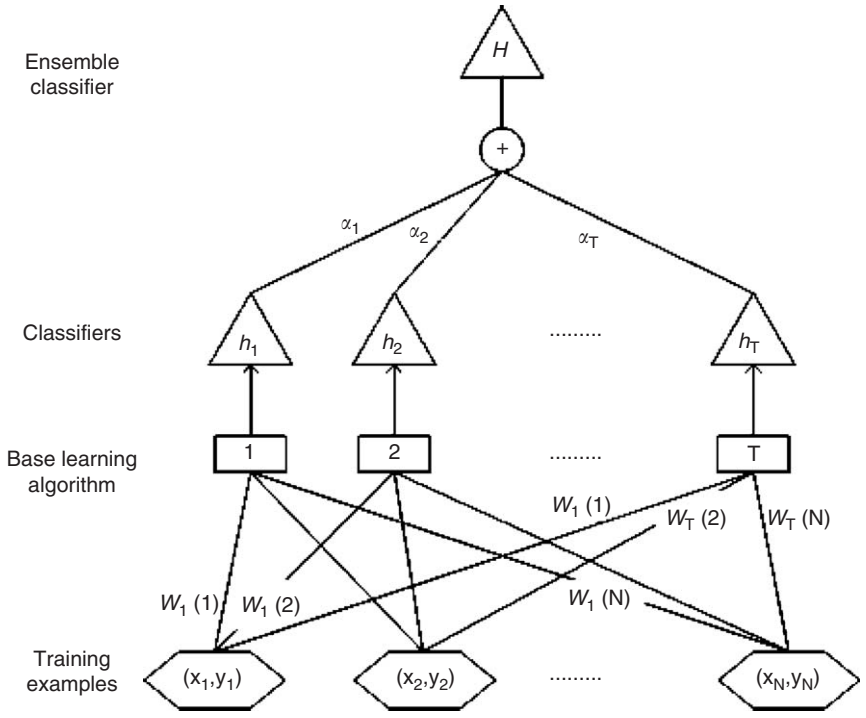

Fig. 1. Multiple classifier system based on manipulating the training examples.

different $\mathbf{w}_t$ and therefore producing different $h_t(.)$. This idea is depicted in Fig. 1. It is implemented successfully in the *Bagging* algorithm by Breiman [5]. At each time, Bagging feeds the learning algorithm with a training set that consists of N examples drawn (with replacement) from S according to a uniform distribution. The induced classifiers are then combined by unweighted voting

$$\alpha_1 = \cdots = \alpha_T = 1/T. \tag{2}$$

In each of the training sets, some examples may appear more than once while some may not appear; such a set is called a *bootstrap replicate* of S [6]. We can also think of it as assigning weight

$$w_t(i) = \text{number of occurrence of } (\mathbf{x}_i, y_i)/N \tag{3}$$

on each training example. So, in essence, Bagging is using random weights from a discrete binomial distribution.

There is another important algorithm, named *AdaBoost* [7,8] by Freund and Schapire, which sequentially adjusts the weights. AdaBoost starts from a uniform weighting

$$w_1(1) = \cdots = w_1(N) = 1/N. \tag{4}$$

At time t, one shall construct a classifier $h_t(.)$ based on $\mathbf{w}_t$ and calculate the weighted training error ε_t:

$$\varepsilon_t = \sum_{i=1}^{N} w_t(i) \cdot \mathbf{I}(y_i \neq h_t(\mathbf{x}_i)), \tag{5}$$

where $\mathbf{I}(E) = 1$ if the event E occurs or 0 otherwise. Then set

$$\beta_t = \frac{1}{2} \log \frac{1-\varepsilon_t}{\varepsilon_t}. \tag{6}$$

Now one may update the weights by the following rule:

$$w_{t+1}(i) = w_t(i) \exp\{-\beta_t y_i h_t(\mathbf{x}_i)\}/Z_t, \tag{7}$$

where Z_t is a normalization constant. A direct explanation here is that weights for those examples that have been misclassified by the previous classifier ($y_i h_t(\mathbf{x}_i) < 0$) will be increased and those that have been correctly classified ($y_i h_t(\mathbf{x}_i) > 0$) will be decreased. The above procedure is iterated for T rounds, and finally one gets the ensemble classifier using Eq. (1) with

$$\alpha_t = \beta_t \Big/ \sum_{t=1}^{T} \beta_t. \tag{8}$$

Apparently, those constituent classifiers with lower training error will have higher voting weights in the ensemble.

A careful comparison between Bagging and AdaBoost reveals that essentially they employ different ways to search for an appropriate ensemble of weightings $\{\mathbf{w}_t\}$. Bagging performs a "blind" search with randomly generated weightings. The constituent classifiers do not depend on each other and they can be constructed in parallel. Their opinions are given equal weight in the ensemble. By contrast, AdaBoost is essentially a serial procedure. One shall generate each weighting (except the first one) by making an adjustment

on the previous one, using the evaluation result of the previous classifier to decide the step size and direction. These classifiers are finally combined by weighted voting.

The above analysis inspires us to design an algorithm of ensemble classification which performs evolutionary search in the *weighting space*. A genetic algorithm (GA [9]) can be incorporated. Specifically, we treat a weighting $\mathbf{w}_t$ as a chromosome and evaluate its fitness by the classification accuracy of its corresponding classifier $h_t(.)$ on the training examples. The algorithm shall maintain a population of chromosomes at each generation. Three operators, *reproduction*, *crossover* and *mutation*, are used to evolve the population. The crossover operator will be an attracting characteristic through which large jumps are possible rather than only local search with mutations (compared with AdaBoost). More areas are thus explored in the weighting space. The resulting component classifiers are much likely to have large diversity. On the other hand, although GA uses probabilistic transition rules, not deterministic rules, it does much more than a random search (like in Bagging). The *survival of fittest* principle continuously drives the algorithm to areas with lower classification error. The component classifiers therefore tend to have high accuracy. In these two views, efficient search is thus achieved. In addition to the application of a GA in ensemble classification, another novelty of our algorithm is that all of the chromosomes throughout evolution will be exploited, not just the best one. All the induced classifiers will contribute to the ensemble. The resulting ensemble is fairly likely to have a high classification accuracy. Meanwhile, our algorithm also enjoys a simple control structure and the easiness to be parallelized.

The rest of this paper is organized as follows. A detailed description of the proposed algorithm is given in Section 2. In Section 3 are experimental results on some widely used benchmark data sets, and on a practical application—face detection. We then conclude in Section 4.

2. Algorithm description

We know that a weighting is a group of weights over the training examples. The key job in our algorithm is to find a set of weightings since each of them leads to a particular classifier. It is therefore natural and straightforward to treat a weighting as a chromosome. Formally we assume there are K chromosomes in a population and the population will be evolved for J generations. We shall generate a set of chromosomes $\{\mathbf{w}_{j,k} | j = 1, \ldots, J, k = 1, \ldots, K\}$ throughout the evolution, each of which is defined as

$$\mathbf{w}_{j,k} = (w_{j,k}(1), \ldots, w_{j,k}(N)) \tag{9}$$

(subject to $\sum_{i=1}^{N} w_{j,k}(i) = 1$ and $w_{j,k}(i) \geqslant 0$). Note that we employ real-value coding here, not a conventional bit string coding. So, in later description, we use terms "weighting" and "chromosome" interchangeably.

Every time we invoke a base learning algorithm and feed it with the weighted training set $\{S; \mathbf{w}_{j,k}\}$, we shall obtain a classifier $h_{j,k}(.)$. We compute the weighted training error of $h_{j,k}(.)$ on the N examples by

$$\varepsilon_{j,k} = \sum_{i=1}^{N} w_{j,k}(i) \cdot \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i)). \tag{10}$$

We then define the fitness of a chromosome $\mathbf{w}_{j,k}$ as

$$f_{j,k} = \frac{1}{2} \log \frac{1 - \varepsilon_{j,k}}{\varepsilon_{j,k}}. \tag{11}$$

The two equations above are inspired by Eqs. (5) and (6) in AdaBoost. We will see that a lower training error corresponds to a higher fitness value. If one views the training set S as an environment where the chromosomes compete to live, $\{f_{j,k}\}$ provide a reasonable indication of how well each chromosome adapts to it.

The algorithm is now outlined in Fig. 2. In order to make it comparable to Bagging and AdaBoost in computational cost, we always set $J \times K = T$. That is, three algorithms all have T classifiers in the final ensemble. The proposed algorithm has a simple and clear control structure. The population of chromosomes in the first generation, $\{\mathbf{w}_{1,k}\}_{k=1}^{K}$, is generated randomly. For example, referring to Eq. (9), we may generate N random numbers that are uniformly distributed in (0, 1), then normalize them to form a valid weighting. Each population $\{\mathbf{w}_{j,k}\}_{k=1}^{K}$ in the later generations ($j = 2, \ldots, J$) will be generated from its preceding population $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ by genetic operators. For each chromosome in the populations, we will train a classifier accordingly and evaluate its fitness. All the induced classifiers are finally combined to derive the ensemble classifier. In the ensemble, each component classifier $h_{j,k}(.)$ will be assigned a voting weight $\alpha_{j,k}$. The fitness $f_{j,k}$ of a classifier will determine how its opinion on a pattern is weighted with respect to the opinions of other classifiers, that is, those classifiers with high fitness evaluation will have high voting weights:

$$\alpha_{j,k} = f_{j,k} \Big/ \sum_{j=1}^{J} \sum_{k=1}^{K} f_{j,k}. \tag{12}$$

There are two important issues to be addressed: one is the mechanism to evolve the chromosomes generation by generation, and the other is how to train a classifier using weighted examples.

2.1. Reproduction, crossover and mutation

Given the chromosomes and their fitness values in generation $j - 1$, i.e. $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ and $\{f_{j-1,k}\}_{k=1}^{K}$, we repeat the following procedure:

(1) *Reproduction*: We use *roulette-wheel* selection [9]. Two chromosomes $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$ are sampled (with

Given the training set S, number of generation J, size of population K

Initialize for $j = 1$:
- Randomly generate the chromosomes $\{\mathbf{w}_{1,k}\}_{k=1}^{K}$
- Train a classifier $h_{1,k}(.)$ for each $\mathbf{w}_{1,k}$
- Evaluate a fitness value $f_{1,k}$ for each $\mathbf{w}_{1,k}$

For $j = 2...J$:
- Generate the chromosomes $\{\mathbf{w}_{j,k}\}_{k=1}^{K}$ from $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ by reproduction, crossover and mutation
- Train a classifier $h_{j,k}(.)$ for each $\mathbf{w}_{j,k}$
- Evaluate a fitness value $f_{j,k}$ for each $\mathbf{w}_{j,k}$

Output the ensemble classifier:

$$H(.) = \text{sign}\left(\sum_{j=1}^{J}\sum_{k=1}^{K}\alpha_{j,k}h_{j,k}(.)\right)$$

where $\alpha_{j,k} = f_{j,k}/\sum_{j=1}^{J}\sum_{k=1}^{K} f_{j,k}$.

Fig. 2. The algorithm of classification by evolutionary ensembles.

replacement) from $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ according to the distribution

$$\left[\frac{f_{j-1,1}}{\sum_{k=1}^{K} f_{j-1,k}}, \ldots, \frac{f_{j-1,K}}{\sum_{k=1}^{K} f_{j-1,k}}\right]. \tag{13}$$

The possibility of a chromosome to be selected is proportionate to its fitness. Chromosomes with higher fitness have more chances to copy their genes to the next generation. Note that the selected two chromosomes may be the same.

(2) *Crossover*: Only one offspring chromosome is produced. It is a linear combination of the two parent chromosomes $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$:

$$\mathbf{w}_{offspring} = \frac{f_{j-1,k1}\cdot\mathbf{w}_{j-1,k1} + f_{j-1,k2}\cdot\mathbf{w}_{j-1,k2}}{f_{j-1,k1} + f_{j-1,k2}}. \tag{14}$$

Apparently, the weight items in $\mathbf{w}_{offspring}$ will also sum to 1, like in $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$. We emphasize that in this step there is selective pressure involved; parent chromosome that has higher fitness will contribute more to the offspring chromosome. This is different from a traditional GA which normally imposes selective pressure in the *reproduction* step only [9]. Moreover, we always run this crossover operator after reproduction, while in a traditional GA it is normally run with a probability (< 1).

(3) *Mutation*: Here comes another parameter of the algorithm, probability of mutation p_m. We randomly partition the N weight items in $\mathbf{w}_{offspring}$ into $N/2$ pairs (if N is odd, then partition $N-1$ weight items into $(N-1)/2$ pairs), each of which consists of two weights. For each pair, we exchange the values of the two weights by probability p_m. This operation protects against premature convergence of the chromosomes. Usually p_m is given a very small value. We have noticed that this kind of disturbance in weightings is helpful in improving the final classification accuracy.

The above three-step procedure is repeated for K times, each time the mutated chromosome is saved as a member of the generation j. In the end we get all the chromosomes $\{\mathbf{w}_{j,k}\}_{k=1}^{K}$. Based on these new weightings, we can train a new generation of classifiers and evaluate their fitness. The iteration in Fig. 2 thus continues.

We have mentioned some differences between a traditional GA and our algorithm. They are mostly due to the consideration that the chromosomes throughout all the generations shall make contribution to our final result (through the classifiers); while in a traditional GA one only picks the best chromosome in the final generation as the final result. That also makes the *replacement* issue [10], i.e. how to replace the old chromosomes with the new generated chromosomes, unnecessary in our algorithm. To our knowledge, Yao and Liu [11] made the first attempt in neural network learning that all the chromosomes in the last generation are combined to create the final result. They experienced performance improvement. Our algorithm, in a further way, makes use of all the generations of chromosomes. This is an extension to Yao and Liu's idea for evolutionary learning.

2.2. Training with weighted examples

If one thinks of evolving the weightings as a high-level job, the low-level job concerns handling weighted training examples in a base learning algorithm. There are basically

two approaches to take a weighting into consideration. First, many learning algorithms are based on minimizing an error function, which is a sum of error items caused by each of the training examples. In this case, one can design a revised function that assigns a weight to each error item, so that heavily weighted examples are more influential. For example, one may attempt to minimize $\sum_{i=1}^{N} w_{j,k}(i) \cdot \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i))$, not the conventional $\sum_{i=1}^{N} \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i))$. However, if the learning algorithm is not readily adaptable to the inclusion of weights, one shall employ a second approach proposed in Ref. [12]. It is based on resampling S with replacement according to the weighting $\mathbf{w}_{j,k}$. This approach is to some extent similar to what is employed in Bagging. Quinlan [13] reported that both approaches are very effective. But in our algorithm, we prefer the first approach if applicable.

So far, we explained the whole algorithm in great details. It basically provides a generic approach for ensemble classification. Many modifications are possible in evolving the chromosomes [10]. For example, one may use other selection methods in the reproduction step, or even consider using other coding methods instead of the current real-value coding. The crossover operator can also be redesigned. On the other hand, the algorithm is not fixed to a particular base learner. Different kinds of learning machines, like k nearest neighbor, neural network and decision tree, can all be incorporated. The practitioners shall consider all these aspects when applying our algorithm to a particular problem.

Although we initially set a binary label space $\mathbb{Y} = \{-1, +1\}$, our algorithm applies to multi-class problems immediately. The only prerequisite is that the component classifiers can perform multi-class classification, which is fulfilled by most of the kinds of classifiers like nearest neighbor, neural network or decision tree. Referring to Eq. (10), for each potential weighting, we calculate its weighted training error by only comparing the predicted output $h_{j,k}(\mathbf{x}_i)$ with the actual label y_i. As long as $h_{j,k}()$ works, our algorithm works. There is no need for extra attention on multi-class classification. For those component classifiers that cannot perform multi-class classification, the *error correcting output coding* method [14] may be helpful. It generally reduces the multi-class problem to a number of binary problems.

The computational cost of our algorithm is comparable to that of Bagging and AdaBoost. They have the same number (T) of ensemble members and hence the same number of calls to the base learning algorithm. The overheads of evolving the weightings are negligible. Indeed, the strategy of using GA for ensemble classification can support greater computational efficiency by allowing parallelism of learning procedure which is not readily possible with AdaBoost. However, despite negligible computational overheads, our algorithm achieves appreciable and reasonably consistent capability of learning. It will be experimentally demonstrated in the next session.

3. Experimental results

We conduct experiments on some benchmark data sets. The learning capability of our algorithm is to be measured by the accuracy of an ensemble classifier, which is learned from training examples, in classifying unseen features. We compare our algorithm with the two widely used algorithms: AdaBoost and Bagging, and analysis their performance.

3.1. UCI benchmark data sets

A total of 20 data sets are selected from the UCI Machine Learning Repository [15]. These data sets have been extensively adopted for benchmark study. A summary of some of their properties is given in Table 1. We can see that these data sets are representative for a wide range of classification problems. The training set size could be large (e.g. "Letter recognition" and "Waveform") or small (e.g. "Labor"), and the dimension of feature space could be high (e.g. "Audiology" and "Sonar") or low (e.g. "Iris"). One may also notice that some of the data sets have more than two classes (e.g. "Audiology" and "Letter recognition"). In the appendix we give a brief explanation for the classification problems involved in the data sets used.

We employ a five-fold cross-validation in estimating the classification accuracy. All the training examples in a data set are partitioned randomly into five parts. Four of them are selected for training and the constructed ensemble is tested on the remaining part. This process is repeated for five times, each time with a different part for test. The test results are then combined to calculate the classification accuracy over the whole data set. This strategy makes sure that each

Table 1
Summary of the 20 UCI benchmark data sets used in the experiments

Data set	Training set size	Dimension of feature space	Number of classes
Audiology	226	69	24
Automobile	205	24	7
Breast cancer—W	699	9	2
Diabetes	768	8	2
German	1000	20	2
Glass	214	9	7
Heart	270	13	2
Hepatitis	155	19	2
Horse colic	368	21	2
Iris	150	4	3
Labor	57	16	2
Letter recognition	20 000	16	26
Segment	2310	19	7
Sonar	208	60	2
Soybean—large	683	35	19
Splice	3177	60	3
Thyroid	215	5	3
Vehicle	846	18	4
Voting	435	15	2
Waveform	5000	21	3

training example will be used for both training and test. Furthermore, to achieve a reliable estimation, on each data set we repeat the above procedure for 50 times, starting from partitioning the training examples. The 50 values of classification accuracy obtained are averaged to get the final estimation.

Three algorithms, Bagging, AdaBoost and our algorithm, are compared on each data set. As the base learning algorithm, they all use the RBF network [16]. The number of classifiers in an ensemble is set to 100. Clearly, this number is somewhat arbitrary and may not be optimal. However, since it is fixed for all algorithms, the comparison should be fair. Moreover, such a size of ensemble classifier has been widely employed in experimental studies of ensemble learning [17]. For our algorithm, we accordingly set $J = K = 10$, that means 10 generations of population each containing 10 classifiers. From a viewpoint of traditional genetic algorithms, this may seem a rather limited number of chromosomes and generations for optimization. The algorithm may not converge after such a short-term evolution. However, in the domain of machine learning, we aim on the generalization capability of constructed classifiers which is signified by the classification accuracy on new and unseen examples (we estimate this accuracy using a test set). And, a high classification accuracy on the training set does not necessarily lead to a same accuracy on the test set. A converged population of classifiers may provide optimized training accuracy, but it tends to suffer from the "over-fitting" effect and provide deteriorated generalization capability [16]. To clarify our point we modify the proposed algorithm in that only the optimized component classifiers in the last generation will be combined to form the ensemble classifier. This modified version of our algorithm is also tested on the UCI data sets and compared with the other three algorithms.

The estimated mean values of classification error rate are listed in Table 2. A lower error rate means a higher classification accuracy. Four algorithms, Bagging, AdaBoost, our algorithm and the modified version of our algorithm (denoted by "our algo.*"), are compared. The winner in each data set is given in bold style. We thus make the following observations:

(1) The classification error rate tends to be high when the training set size is small while the dimension of feature space is large (e.g. "Sonar"). In the same way, the error rate tends to be low when we have enough training examples (e.g. "Letter recognition").
(2) In all of 20 data sets, AdaBoost wins in 11 sets (according to the mean classification error rate), Bagging wins in five sets, our algorithm wins in three sets, and the modified version of our algorithm wins in only one set. It seems that AdaBoost does best and our algorithm is only comparable to Bagging which essentially performs random search. Additionally, in each of the three sets where our algorithm wins, its error rate is very close to the second best value. Because of the randomness of the estimation, we may consider that they are statistically equal.
(3) Further, we check all the sets where our algorithm loses and notice a favorable phenomenon. That is, the error rates of our algorithm are in most cases quite close to those of the best algorithm and much better than the worst algorithm. Our algorithm only ranks the worst in "Labor" while in this case the four error rates are very close (0.1847; 0.1836; 0.1852; 0.1846). This phenomenon is clearly shown in Fig. 3. We use the error rates of our algorithm as the reference (ratio 1) and calculate the ratios of Bagging/our algo., AdaBoost/our algo. and our algo.*/our algo., respectively, in each data set. A ratio less than 1 means a lower error rate and correspondingly a better algorithm. It is evident from the figure that our algorithm is not sensitive to the data set and in most cases achieves classification accuracy comparable to the best algorithm. In the mean time, Bagging and AdaBoost are not so stable and may perform poorly in some cases.
(4) The modified version of our algorithm uses only the optimized component classifiers in the last generation to construct the ensemble. It wins in the "Hepatitis" data set. However, the four error rates are actually very close (0.1702; 0.1693; 0.1693; 0.1691). Compared with Bagging and AdaBoost, the modified algorithm acts more moderately; that is, although it does not provide the best performance in most cases, it is usually not the worst.

Table 2
Estimation of mean classification error rate on 20 data sets

Data set	Error rate			
	Bagging	AdaBoost	Our algo.	Our algo.*
Audiology	0.2212	**0.1766**	0.1841	0.1955
Automobile	0.2122	0.2114	**0.2113**	0.2118
Breast cancer—W	0.0452	**0.0365**	0.0376	0.0398
Diabetes	0.2412	0.2691	**0.2405**	0.2575
German	**0.2679**	0.2763	0.2707	0.2792
Glass	0.3083	**0.2791**	0.2814	0.2898
Heart	**0.1866**	0.2085	0.1871	0.2116
Hepatitis	0.1702	0.1693	0.1693	**0.1691**
Horse colic	**0.1558**	0.1962	0.1625	0.1993
Iris	**0.0693**	0.0845	0.0696	0.0830
Labor	0.1847	**0.1836**	0.1852	0.1846
Letter recognition	0.0654	**0.0315**	0.0456	0.0435
Segment	0.0352	**0.0242**	0.0247	0.0290
Sonar	0.2991	**0.1839**	0.1907	0.2031
Soybean—large	0.1233	**0.0993**	0.1001	0.1141
Splice	0.0725	**0.0717**	0.0720	0.0718
Thyroid	0.0429	0.0426	**0.0424**	0.0440
Vehicle	0.2875	**0.2420**	0.2461	0.2468
Voting	**0.1026**	0.1187	0.1098	0.1263
Waveform	0.1524	**0.1287**	0.1295	0.1300

Bagging, AdaBoost, our algorithm and a modified version of our algorithm (denoted by "our algo.*") are compared, using RBF network as the base learning algorithm. Each of the algorithms is implemented for 50 times on every data set for a reliable estimation. The winner in each comparison is emphasized in bold value.

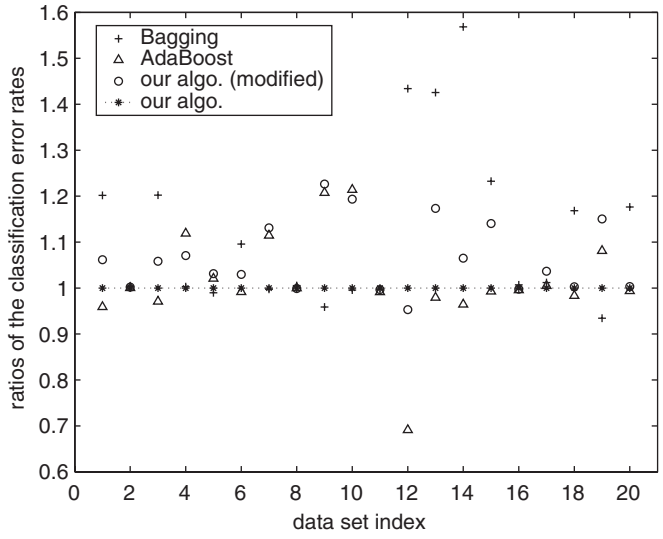

Fig. 3. The ratios of the classification error rates. They are calculated among the four algorithms in each data set. We use the error rate of our algorithm as the reference (ratio 1) and calculate the ratios of Bagging/our algo., AdaBoost/our algo and our algo.*/our algo., respectively.

Compared with our original algorithm, its performance seems more dependent on the specific data sets. We therefore draw the conclusion that the more randomized component classifiers in the early generations should not just be thrown away and they can also contribute to the ensemble as the more optimized ones in the late generations. Dietterich made the same observation in[4]. He termed the phenomenon as the *accuracy–diversity* balance within an ensemble. Although the classifiers in the early generations may not provide high training accuracy, their existence does increase the diversity and improve the overall performance.

We may use the mean ratio of error rates across domains as a measure of relative performance [3,13]. However, Webb [17] reported that this measure could have the relative difficulty of error comparison in different domains. He gave the example that two algorithms x and y achieve error rates of 0.10 and 0.20 for domain A and 0.35 and 0.20 for domain B, respectively. The ratios will be 0.50 and 1.75, giving a mean ratio of 1.125, indicating that, averagely algorithm x has error 12.5% greater than algorithm y. But, if we consider the ratio of y to x, we get $0.20/0.10=2.0$ and $0.20/0.35=0.571$, giving a mean ratio of 1.286, indicating that, algorithm y on average has error 28.6% greater than algorithm x! Clearly, it is not desirable to draw the conclusion based on the mean ratio of error rates. Webb therefore adopted the geometric mean ratio rather than the arithmetic mean ratio:

$$e^{(\sum_{i=1}^{n} \log(a_i/b_i))/n} = \sqrt[n]{\prod_{i=1}^{n} \frac{a_i}{b_i}}. \quad (15)$$

The geometric mean ratio of $\{a_i/b_i\}$ enjoys the desirable property that if it is greater than one then the geometric mean ratio of $\{b_i/a_i\}$ will be less than one. In the above example, the geometric mean ratio of x_i/y_i is 0.935 while the geometric mean ratio of y_i/x_i is 1.069, suggesting that x has the true advantage in terms of error reduction over the two domains.

We therefore calculate the geometric mean ratios of error rates for our algorithm with respect to the other three algorithms and record the result in Table 4. From the table we can see that, on average, our algorithm performs better than the other three algorithm, although the difference between our algorithm and AdaBoost is not much significant. This is in accordance with the above-mentioned observations.

For further insight, we record the variances of error rate estimation on each data set in Table 3. The data is then depicted in Fig. 4 for a clear comparison. Among the four algorithms, Bagging generally provides the least variance. This observation is consistent with the conclusion in Ref. [3]. The modified version of our algorithm seems more unstable and has high variance, possibly due to the limited population size and different areas that the algorithm converged. The situation is alleviated to some extent in our original algorithm by the inclusion of more randomized component classifiers. Our algorithm therefore acts more moderately, although there are also some exceptional cases. The geometric mean ratios of error variance are also calculated and recorded in Table 4. We can clearly see that, on average, the Bagging algorithm gives the most stable classification, followed by our algorithm and then AdaBoost, while the modified version of our algorithm is the worst. The result provides a vivid support for the usefulness of those chromosomes in the early generations.

After all, the difference among the four algorithms is generally not so remarkable. If we analysis the error rate data and variance data in combination, the four algorithms

Table 3
Variance of estimation for classification error rate in the experiment

Data set	Estimation variance			
	Bagging	AdaBoost	Our algo.	Our algo.*
Audiology	0.0035	0.0047	0.0039	0.0046
Automobile	0.0031	0.0044	0.0040	0.0048
Breast cancer—W	0.0006	0.0012	0.0010	0.0015
Diabetes	0.0033	0.0035	0.0029	0.0043
German	0.0037	0.0045	0.0035	0.0047
Glass	0.0024	0.0040	0.0033	0.0042
Heart	0.0021	0.0036	0.0032	0.0035
Hepatitis	0.0028	0.0027	0.0021	0.0033
Horse colic	0.0018	0.0030	0.0023	0.0031
Iris	0.0009	0.0005	0.0012	0.0011
Labor	0.0015	0.0036	0.0026	0.0034
Letter recognition	0.0006	0.0017	0.0009	0.0021
Segment	0.0005	0.0024	0.0018	0.0022
Sonar	0.0024	0.0035	0.0032	0.0037
Soybean—large	0.0014	0.0021	0.0028	0.0020
Splice	0.0007	0.0013	0.0011	0.0016
Thyroid	0.0005	0.0009	0.0012	0.0011
Vehicle	0.0026	0.0033	0.0030	0.0041
Voting	0.0018	0.0022	0.0028	0.0029
Waveform	0.0012	0.0025	0.0020	0.0033

The mean values are recorded in Table 2.

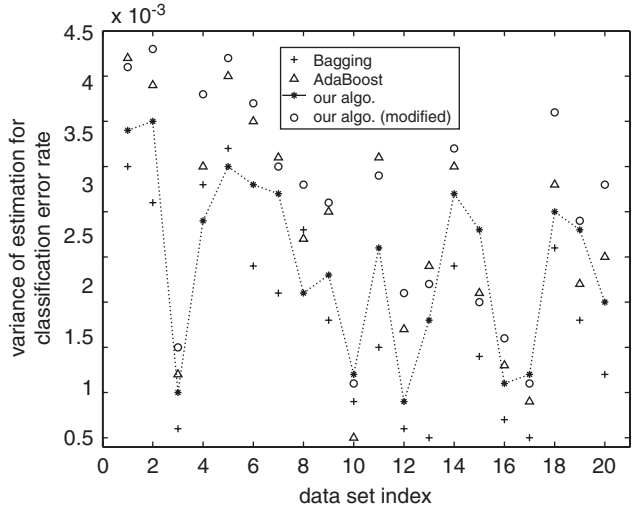

Fig. 4. The variance of estimation for classification error rate. Four algorithms are compared.

Table 4
Geometric mean ratios of error rates and error variances for out algorithm with respect to AdaBoost, Bagging and a modified version of our algorithm (our algo.*)

Geometric mean ratio	$\frac{\text{our algo.}}{\text{AdaBoost}}$	$\frac{\text{our algo.}}{\text{Bagging}}$	$\frac{\text{our algo.}}{\text{our algo.}^*}$
Error rate	0.9922	0.9025	0.9401
Error variance	0.9111	1.4416	0.7907

The mean values are recorded in Tables 2 and 3.

Table 5
Sensitivity of classification error rate with respect to the probability of mutation p_m

p_m	Audiology	Iris	Splice	Waveform
0.01	0.1855	0.0709	0.0741	0.1326
0.02	0.1829	0.0710	0.0731	0.1307
0.03	0.1847	0.0704	0.0735	0.1318
0.04	0.1836	0.0713	0.0719	0.1300
0.05	0.1841	0.0696	0.0720	0.1295
0.06	0.1873	0.0688	0.0729	0.1337

The experiment is on four data sets: Audiology, Iris, Splice and Waveform. The classification error rates are recorded.

actually provide pretty equivalent performance. Algorithms providing lower error rates generally have higher variance, for example in the "Glass" data set. It says that a specific algorithm may converge to a good local optima temporarily; however, the convergence is sensitive to the starting search point and the results across multiple runs could be much different. Nevertheless, the comparison results reveal that our algorithm performs consistently well no matter what the data set is. While it may not achieve the best classification accuracy, it always provides a satisfying approximation. This constitutes one of the attractive properties of our algorithm. It is believed to be a favorable characteristic of evolutionary algorithms which can be exploited in cases with little a priori knowledge. In this way, the intention of this paper is to present evolutionary ensemble as an interesting and promising technique for learning and classification task.

3.2. Parameter sensitivity

Note that there is no parameter for crossover in our algorithm. The parameter p_m, probability of mutation, is always set 0.05. We have experimented a range of values for $p_m \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06\}$ on four benchmark data sets (other settings are the same with the above subsection). The four data sets are selected due to their representability of different classification tasks. The results are recorded in Table 5.

In traditional genetic algorithms for numerical optimization, the probability of mutation is usually given a very small value (e.g. 0.001) and it has significant influence on the convergence performance. In our formulation of evolutionary ensemble learning, however, p_m is given a relatively large value (0.01–0.06) and the influence on classification accuracy is not so significant (Table 5). Considering the randomness of experiment, the performance in different settings for p_m is largely equal. This is partially due to the fact that the success of our algorithm does not depend solely on the convergence of evolution; the diversity of the chromosomes is also very important. For a reasonable range of p_m, the performance will not fluctuate too much. Yet, we still observe that a setting of 0.05 is generally better than 0.01 or 0.06. Too small a value will cause less diversity of component

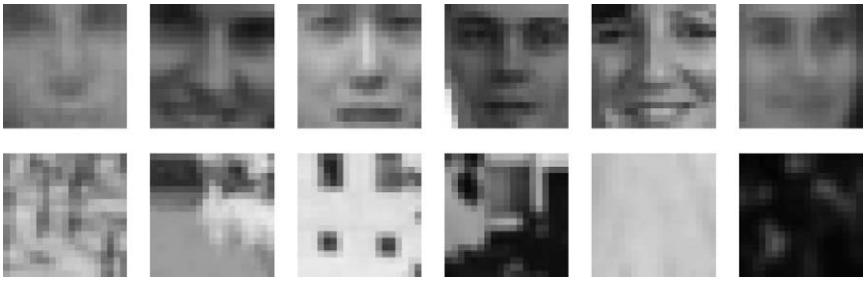

Fig. 5. Some of the training images. In the upper row are faces; non-faces are in the lower row.

classifiers in the final ensemble and too large a value, on the contrary, will deteriorate the convergence to highly accurate classifiers and eventually impair the overall performance of the ensemble.

We recognize that this setting of parameter is not a universally best choice for all practical problems. In our experiment, it is picked mainly to demonstrate the potential of evolutionary search in ensemble classification. As an application of genetic algorithm, our algorithm adopts heuristic selection of parameters. This also includes the population size and number of generations. In the present paper, the selection is based on trial-and-error. For practitioners, however, the parameters must be fine-tuned to the problem domain. Especially in those specially designed evolutionary algorithms, the settings for parameters could be much different from our recommendations. On the other hand, it is also possible to use dynamic parameters and carry out additional search for optimum values [18]. But it mostly depends on particular problem domains.

3.3. Face detection using evolutionary ensembles

Next, we verify the proposed algorithm on *face detection*, which has been regarded as a challenging problem and attracted much attention in the computer vision community (see Ref. [19] for a survey). We design a system to detect upright, frontal views of faces in gray-scale images. Such a system may find wide applications in real-life surveillance and biometric.

Usually face detection is treated as a binary classification problem. For every subregion in the image (say, 19×19 window), one extracts a feature vector and then applies a classifier which outputs a label $+1$ or -1, signifying the presence or absence of a face respectively. To detect faces larger than the window size, the image is repeatedly reduced in size by sub-sampling, and the classifier is applied at each size. The *principal component analysis* and *wavelet transform* are often used for feature extraction.

In our system, the Haar wavelet feature [20] is used. The training images including faces and non-faces are copied from the MIT CBCL Face Database [21]. Some of them are shown in Fig. 5. The proposed algorithm is used to learn an ensemble of 100 component classifiers, with decision stump as the base learning algorithm. A decision stump is a one-level decision tree which is easily trained [16]. It normally classifies a pattern based on one item of the feature vector and therefore could only give weak classification (slightly better than random guess). However, we will see that the detection results obtained by the ensemble is quite satisfying. In the same setting (ensemble of 100 decision stumps), we also implement the Bagging and AdaBoost algorithms and compare them with our algorithm.

Table 6
Face detection results on images from the test set that contains 130 images with 507 faces

Algorithm	Faces detected	False detections	Detection rate
Bagging	449	83	88.6%
AdaBoost	477	32	94.1%
Our algo.	474	41	93.5%

Bagging, AdaBoost and our algorithm are compared.

We test the algorithms on the widely-used face detection database which is created by Rowley et al. [22] (available at http://www.cs.cmu.edu/~har/faces.html). It consists of 130 images with a total of 507 frontal faces. Most images contain more than one face on a cluttered background and, so, this is a good test set to assess algorithms which detect upright frontal faces. We summarize the results by the three algorithms in Table 6.

The experimental results show that AdaBoost achieves the highest detection rate of 94.1% with 32 false detections. Our algorithm compares closely with a detection rate of 93.5% and 41 false detections. The Bagging algorithm, which randomly constructs the component classifiers, achieves clearly worse performance (detection rate 88.6% and 83 false detections). We further depict some of detection results by our algorithm in Fig. 6. The detected faces are framed in white. In each of Fig. 6 (A,C,D,I), there is only one face; while in the other subfigures there are several faces. The images in Fig. 6 (D,I) have very low quality. In most of these cases, our system detects the faces correctly. Also, we obtain missed and false detections. One face in Fig. 6 (B) and two faces in Fig. 6 (F) are missed by our system. This is probably due to the severe rotation involved. Another missed detection in Fig. 6 (H) is due to the large occlusion on face. An

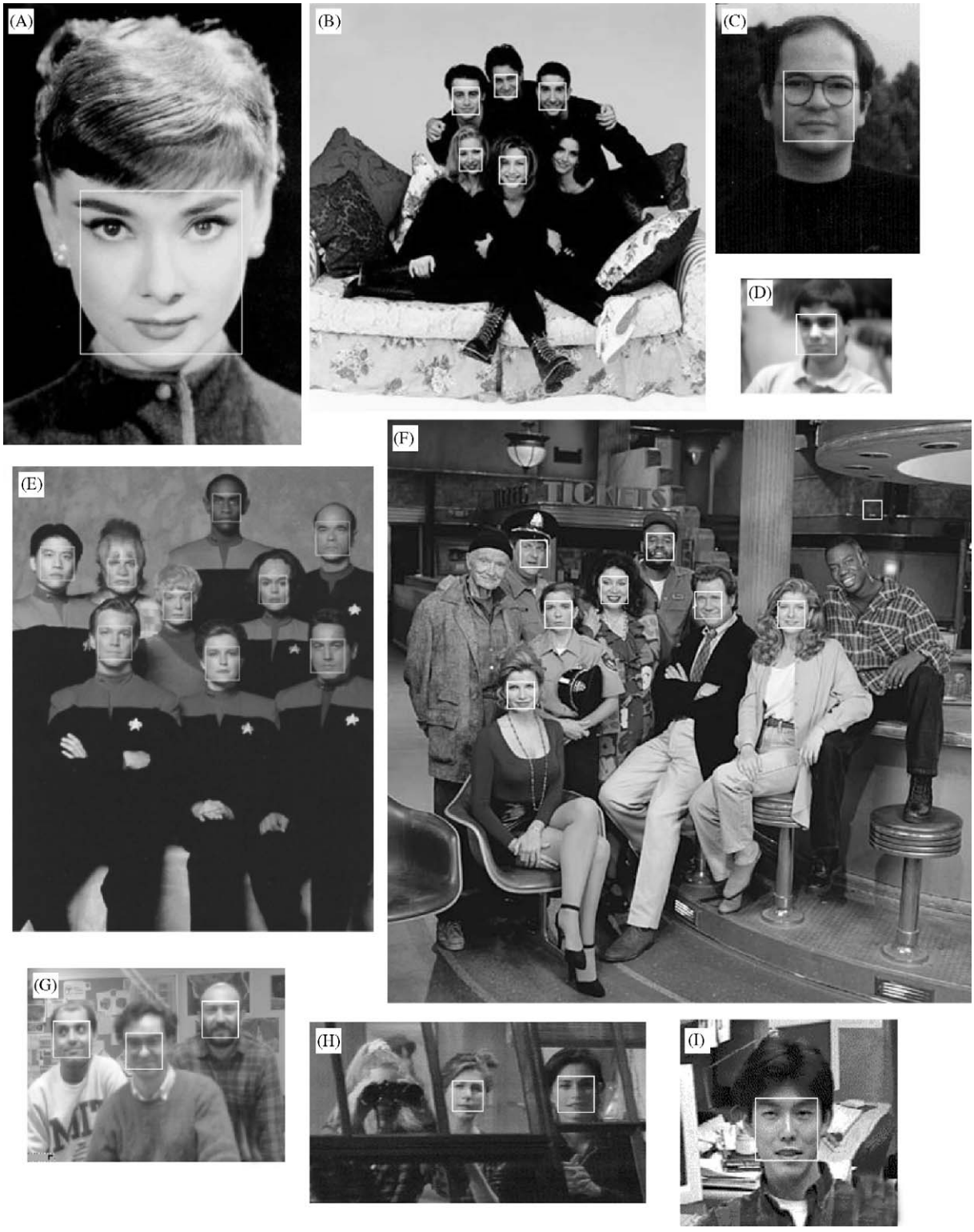

Fig. 6. Some results on test images. The detected faces are framed in white. There are missed detections in (B,F,H), and a false detection in (F). The missed detections are due to severe rotation or large occlusion. The face detection is on a subregion which is quite like a real face.

apparent false detection appears in Fig. 6 (F). The framed window in the upper-right area is rather like a face indeed. Without exploiting the contextual information, a classifier is easy to make an error here. In spite of all the classification errors mentioned above, they also suffered from many state-of-the-art face detection systems [19]. We can always

improve the classification accuracy by using more training images, adopting more salient features or incorporating special techniques to deal with the rotated faces (e.g. rotating in advance the subregions to be classified).

Again, the face detection experiment demonstrates the potential of our algorithm in learning classifier from examples.

3.4. Discussion

The necessity of ensemble classification is easily seen from the previous experiments. Generally, although we are always hoping a single optimal classifier, it is normally unpractical in real cases. On the one hand, we only have limited training examples and the feature vector may not be an integral and efficient representation of the pattern. A learning algorithm can find different classifiers that appear equally accurate with respect to the available training data. Multiple classifier systems therefore try to exploit the local different behavior of component classifiers to enhance the accuracy and reliability of the overall system, and to get a good approximation of the optimal classifier. On the other hand, the unknown classifier may be much more sophisticated than what a specific learning algorithm can get. Although many learning algorithms present universal approximation properties, these asymptotic properties usually call for NP-hard computation of optimization and with insufficient data they actually may not hold. A combination of component classifiers, however, can expand the space of representable functions, hopefully embracing and achieving a better approximation to the optimal one.

In our case of using weighted examples, we find that the weighting space directly corresponds to the classifier space. The Bagging algorithm performs a totally random search while AdaBoost does a local search through step by step mutations. Our algorithm, however, performs an efficient population-based search instead. The GA incorporated uses crossover operator to explore widely diversified areas, the component classifiers are therefore much likely to have different areas of misclassification; meanwhile, the selective pressure always drives the search for classifiers with lower classification error. In this evolutionary manner, our algorithm maintains an effective balance between the diversity and accuracy of the component classifiers. It may explain the pleasing performance of our algorithm demonstrated in the experiments.

4. Conclusion

To summarize, we propose in this paper to incorporate evolutionary search into ensemble classification which works by manipulating training examples. There are basically two novelties in our algorithm:

(a) On training with weighted examples, we employ a genetic algorithm to search for appropriate weightings. The GA implements a wide yet efficient search in the weighting space. The induced classifiers have the desirable properties of both accuracy and diversity.
(b) All of the chromosomes (weightings) throughout evolution are exploited. The reason is that they may have local different and complementary information about the feature space, therefore a majority voting among the opinions of their corresponding classifiers is expected to improve the classification accuracy and robustness of the overall ensemble.

Due to the high nonlinearity and complicated dynamics in evolutionary search, the algorithm does not easily lead to a rigorous theoretical analysis. However, the experiments that we have conducted justify the potential and usefulness of our algorithm. It is therefore a reasonable choice as a general-purpose algorithm for ensemble classification.

Acknowledgements

The authors would like to thank the anonymous reviewers. Their insightful comments and helpful suggestions greatly improved the paper. This research was sponsored by NTU and MOE for the ARC Research Grant and Scholarship.

Appendix: Descriptions of data sets

Here, we briefly describe some properties for each of the data sets that we use in the experiments. For more comprehensive discussions we refer the readers to Ref. [15].

Audiology: This data set is collected during the study of hearing, especially hearing defects and their treatment. Originally there are 200 training cases and 26 test cases. Each case is represented as a feature vector having 69 attributes. The task is to identify each case into one of the 24 classes, such as “normal ear”, “otitis media”, “possible brainstem disorder”, etc.

Automobile: This data set is donated by Jeffrey C. Schlimmer in 1987. The task here is to predict the insurance risk rating of a car using all the numeric and boolean attributes. The rating corresponds to the degree to which the auto is more risky than its price indicates. There are 205 instances in the set. Each instance includes several attributes of a car such as “length”, “width”, “height”, “wheel-base”, etc.

Breast cancer—W : This breast cancer database is obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The total 699 instances arrive periodically as Dr. Wolberg reports his clinical cases, among which about 65.5% belong to the “benign” class and 34.5% belong to the “malignant” class. The attributes in each instance include “Marginal Adhesion”, “Bare Nuclei”, “Bland Chromatin”, and so on.

Diabetes: This is a database gathered among the Pima Indians by the National Institute of Diabetes and Digestive

and Kidney Diseases. The database consists of 768 cases, eight variables and two classes. The variables are medical measurements on the patient plus age and pregnancy information. The classes are: tested positive for diabetes (268) or negative (500).

German: This data set classifies people described by a set of attributes as good or bad credit risks. There are 1000 examples among which 700 persons have good credit and 300 have bad credit. The available attributes include qualitative ones like "credit history" and numerical ones like "duration in month" and "credit amount".

Glass: This database is created in the Central Research Establishment, Home Office Forensic Science Service Aldermaston, Reading, Berkshire. Each case consists of nine chemical measurements on one of seven types of glass. There are 214 cases.

Heart: This database is collected at the University of California, San Diego Medical Center. The goal of the medical study is the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-h data. The data includes measurements like "age", "sex", "chest pain type" and "resting blood pressure". In the 270 cases collected, 150 patients survived and 120 died.

Hepatitis: There are 155 cases in this database. The attribute information includes "age", "sex", "steroid", "fatigue", etc. Two classes shall be identified: "live" and "die".

Horse colic: Originally there are 300 instances for training and 69 instances for test in this database. Several attributes about a horse are collected, like "age", "if had surgery", "rectal temperature", "pulse", etc.

Iris: This database is created and introduced by R.A. Fisher. The data set contains three classes of 50 instances each, where each class refers to a type of iris plant ("Setosa", "Versicolour" and "Virginica"). One class is linearly separable from the other two; the latter are not linearly separable from each other. The four attributes are "sepal length", "sepal width", "petal length" and "petal width".

Labor: This database includes all collective agreements reached in the business and personal services sector for locals with at least 500 members (teachers, nurses, university staff, police, etc.) in Canada in 87 and first quarter of 88. The data are used to learn the description of an acceptable and unacceptable contract. The attribute information includes "duration of agreement", "wage increase in first/second/third year of contract", "cost of living allowance", etc.

Letter recognition: This data set is constructed by David J. Slate, Odesta Corporation. Binary pixel displays of the 26 capital English letters are created and randomly distorted to produce 20 000 images. Sixteen features, consisting of statistical moments and edge counts, are extracted from each image.

Segment: The instances were drawn randomly from a database of seven outdoor images ("brickface", "sky", "foliage", "cement", "window", "pass" and "grass"). The images were hand-segmented to create a classification for every pixel. Each instance is a 3×3 region. The 19 continuous variables include "the column/row of the center pixel of the region", "measures the contrast of vertically adjacent pixels", etc.

Sonar: This data set is first used in a study of the classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time.

Soybean—large: The soybean data set consists of 683 cases, 35 variables and 19 classes. The classes are various types of soybean diseases. The variables are observations on the plants together with some climatic variables. All are nominal. Some missing values are filled in by their modal values.

Splice: This data set is about molecular biology. All examples are taken from Genbank 64.1. There are three categories: "ei" and "ie" including every split-gene for primates in Genbank 64.1, "non-splice" examples taken from sequences known not to include a splicing site. The attributes are the 60 DNA sequence elements given a position in the middle of a window.

Thyroid: This thyroid disease database is donated by Stefan Aeberhard. There are 215 cases in the record. In the data, five lab tests are used to try to predict whether a patient's thyroid to the class euthyroidism, hypothyroidism or hyperthyroidism. The five attributes include "total serum thyroxin", "T3-resin uptake test", etc.

Vehicle: The purpose is to classify a given silhouette as one of four types of vehicle ("OPEL", "SAAB", "BUS" and "VAN"), using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. The 18 attributes include "compactness", "circularity", "radius ratio", etc.

Voting: This is the 1984 United States congressional voting records database. The predicted category is the party affiliation (two categories). The attribute information includes "handicapped-infants", "adoption-of-the-budget-resolution", "religious-groups-in-schools", etc.

Waveform: There are 5000 instances in the database. Each wave is measure in 21 attributes with continuous values between 0 and 6. There are three classes of waves.

References

[1] G. Valentini, F. Masulli, Ensembles of learning machines, in: R. Tagliaferri, M. Marinaro (Eds.), Proceedings of the 13th Italian Workshop on Neural Nets, Lecture Notes in Computer Science, vol. 2468, Springer, Berlin, 2002, pp. 3–19.
[2] L. Breiman, Arcing classifiers, Ann. Statist. 26 (3) (1998) 801–849.
[3] E. Bauer, R. Kohavi, An experimental comparison of voting classification algorithms: bagging, boosting, and variants, Mach. Learn. 36 (1999) 105–139.

[4] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Mach. Learn. 40 (2000) 139–157.
[5] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.
[6] B. Efron, R. Tibshirani, An Introduction to the Bootstrap, Chapman & Hall, New York, NY, 1993.
[7] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1997) 119–139.
[8] R.E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Mach. Learn. 37 (1999) 297–336.
[9] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[10] K.F. Man, K.S. Tang, S. Kwong, Genetic Algorithms: Concepts and Designs, Springer, London, UK, 1999.
[11] X. Yao, Y. Liu, Making use of population information in evolutionary artificial neural networks, IEEE Trans. Syst. Man Cybern. B 28 (3) (1998) 417–425.
[12] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Proceedings of the 13th International Conference on Machine Learning, 1996, pp. 148–156.
[13] J.R. Quinlan, Bagging, boosting, and c4.5, in: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), 1996, pp. 725–730.
[14] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems vis error-correcting output codes, J. Artif. Intell. Res. 2 (1995) 263–286.
[15] C.L. Blake, C.J. Merz, UCI Repository of Machine Learning Databases, 1998.
[16] C.M. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1995.
[17] G.I. Webb, Multiboosting: a technique for combining boosting and wagging, Mach. Learn. 40 (2000) 159–196.
[18] N.N. Schraudolph, R.K. Belew, Dynamic parameter encoding for genetic algorithms, Mach. Learn. 9 (1992) 9–21.
[19] M.-H. Yang, D.J. Kriegman, N. Ahjua, Detecting faces in images: a survey, IEEE Trans. Pattern Anal. Mach. Intell. 24 (1) (2002) 34–58.
[20] C. Papageorgiou, T. Poggio, A trainable system for object detection, Int. J. Comput. Vis. 38 (1) (2000) 15–33.
[21] K.-K. Sung, T. Poggio, Example-based learning for view-based human face detection, IEEE Trans. Pattern Anal. Mach. Intell. 20 (1) (1998) 39–51.
[22] H. Rowley, S. Baluja, T. Kanade, Neural network-based face detection, IEEE Trans. Pattern Anal. Mach. Intell. 20 (1) (1998) 23–38.