Artif Intell Rev (2010) 33:1–39
DOI 10.1007/s10462-009-9124-7

Ensemble-based classifiers

Lior Rokach

Published online: 19 November 2009

Abstract The idea of ensemble methodology is to build a predictive model by integrating multiple models. It is well-known that ensemble methods can be used for improving prediction performance. Researchers from various disciplines such as statistics and AI considered the use of ensemble methodology. This paper, review existing ensemble techniques and can be served as a tutorial for practitioners who are interested in building ensemble based systems.

Keywords Ensemble of classifiers · Supervised learning · Classification · Boosting

1 Introduction

The purpose of supervised learning is to classify patterns (also known as instances) into a set of categories which are also referred to as *classes* or *labels*. Commonly, the classification is based on a classification models (classifiers) that are induced from an exemplary set of preclassified patterns. Alternatively, the classification utilizes knowledge that is supplied by an expert in the application domain.

In a typical supervised learning setting, a set of instances, also referred to as a *training set* is given. The labels of the instances in the training set are known and the goal is to construct a model in order to label new instances. An algorithm which constucts the model is called *inducer* and an instance of an inducer for a specific training set is called a *classifier*.

The main idea behind the ensemble methodology is to weigh several individual classifiers, and combine them in order to obtain a classifier that outperforms every one of them. In fact, human being tends to seek several opinions before making any important decision. We weigh the individual opinions, and combine them to reach our final decision (Polikar 2006).

Marie Jean Antoine Nicolas de Caritat, marquis de Condorcet (1743–1794) was a French mathematician who among others wrote in 1785 the Essay on the Application of Analysis

L. Rokach (✉)
Department of Information System Engineering, Ben-Gurion University of the Negev,
Beer-Sheva, Israel
e-mail: liorrk@bgu.ac.il

to the Probability of Majority Decisions. This work presented the well-known Condorcet's jury theorem. The theorem refers to a jury of voters who need to make a decision regarding a binary outcome (for example to convict or not a defendant). If each voter has a probability p of being correct and the probability of a majority of voters being correct is L then:

- $p > 0.5$ implies $L > p$
- Also L approaches 1, for all $p > 0.5$ as the number of voters approaches infinity.

This theorem has two major limitations: the assumption that the votes are independent; and that there are only two possible outcomes. Nevertheless, if these two preconditions are met, then a correct decision can be obtained by simply combining the votes of a large enough jury that is composed of voters whose judgments are slightly better than a random vote.

Originally, the Condorcet Jury Theorem was written to provide a theoretical basis for democracy. Nonetheless, the same principle can be applied in supervised learning. A strong learner is an inducer that is given a training set consisting of labeled data and produces a classifier which can be arbitrarily accurate. A weak learner produces a classifier which is only slightly more accurate than random classification. The formal definitions of weak and strong learners are beyond the scope of this paper. The reader is referred to Schapire (1990) for these definitions under the PAC theory.

One of the basic question that has been investigated in ensemble learning is: "can a collection of weak classifiers create a single strong one?". Applying the Condorcet Jury Theorem insinuates that this goal might be achieved. Namely, construct an ensemble that (a) consists of independent classifiers, each of which correctly classifies a pattern with a probability of $p > 0.5$; and (b) has a probability of $L > p$ to jointly classify a pattern to its correct class.

Sir Francis Galton (1822–1911) was an English philosopher and statistician that conceived the basic concept of standard deviation and correlation. While visiting a livestock fair, Galton was intrigued by a simple weight-guessing contest. The visitors were invited to guess the weight of an ox. Hundreds of people participated in this contest, but no one succeeded to guess the exact weight: 1,198 pounds. Nevertheless, surprisingly enough, Galton found out that the average of all guesses came quite close to the exact weight: 1,197 pounds. Similarly to the Condorcet jury theorem, Galton revealed the power of combining many simplistic predictions in order to obtain an accurate prediction.

James Michael Surowiecki, an American financial journalist, published in 2004 the book "The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations". Surowiecki argues, that under certain controlled conditions, the aggregation of information from several sources, results in decisions that are often superior to those that could have been made by any single individual—even experts.

Naturally, not all crowds are wise (for example, greedy investors of a stock market bubble). Surowiecki indicates that in order to become wise, the crowd should comply with the following criteria:

- **Diversity of opinion**—Each member should have private information even if it is just an eccentric interpretation of the known facts.
- **Independence**—Members' opinions are not determined by the opinions of those around them.
- **Decentralization**—Members are able to specialize and draw conclusions based on local knowledge.
- **Aggregation**—Some mechanism exists for turning private judgments into a collective decision.

The ensemble idea in supervised learning has been investigated since the late seventies. Tukey (1977) suggests combining two linear regression models. The first linear regression model is fitted to the original data and the second linear model to the residuals. Two years later, (Dasarathy and Sheela 1979) suggested to partition the input space using two or more classifiers. The main progress in the field was achieved during the Nineties. Hansen and Salamon (1990) suggested an ensemble of similarly configured neural networks to improve the predictive performance of a single one. At the same time (Schapire 1990) laid the foundations for the award winning AdaBoost (Freund and Schapire 1996) algorithm by showing that a strong classifier in the *probably approximately correct* (PAC) sense can be generated by combining "weak" classifiers (that is, simple classifiers whose classification performance is only slightly better than random classification). Ensemble methods can also be used for improving the quality and robustness of unsupervised tasks.

Ensemble methods can be also used for improving the quality and robustness of clustering algorithms (Dimitriadou et al. 2003). Nevertheless, in this paper we focus on classifier ensembles.

Given the potential usefulness of ensemble methods, it is not surprising that a vast number of methods are now available to researchers and practitioners. The aim of this paper is to provide an introductory yet extensive tutorial for the practitioners who are interested in building ensemble-based classification systems.

The rest of this paper is organized as follows: In Sect. 2 we present the ingredients of ensemble-based systems. In Sect. 3 we present the most popular methods for combining the base classifiers outputs. We discuss diversity generation approaches in Sect. 4. Section 5 presents selection methods for making the ensemble compact. Hybridization of several ensemble strategies is discussed in Sect. 6. Section 7 suggests criteria for selecting an ensemble method from the practitioner point of view. Finally, Sect. 8 concludes the paper.

2 The ensemble framework

A typical ensemble method for classification tasks contains the following building blocks:

1. Training set—A labeled dataset used for ensemble training. The training set can be described in a variety of languages. Most frequently, the instances are described as attribute-value vectors. We use the notation A to denote the set of input attributes containing n attributes: $A = \{a_1, ..., a_i, ..., a_n\}$ and y to represent the class variable or the target attribute.
2. Base Inducer—The inducer is an induction algorithm that obtains a training set and forms a classifier that represents the generalized relationship between the input attributes and the target attribute. Let I represent an inducer. We use the notation $M = I(S)$ for representing a classifier M which was induced by inducer I on a training set S.
3. Diversity Generator—This component is responsible for generating the diverse classifiers.
4. Combiner—The combiner is responsible for combining the classifications of the various classifiers.

It is useful to distinguish between dependent frameworks and independent frameworks for building ensembles. In a dependent framework the output of a classifier is used in the construction of the next classifier. Thus it is possible to take advantage of knowledge generated in previous iterations to guide the learning in the next iterations. Alternatively each classifier is built independently and their outputs are combined in some fashion.

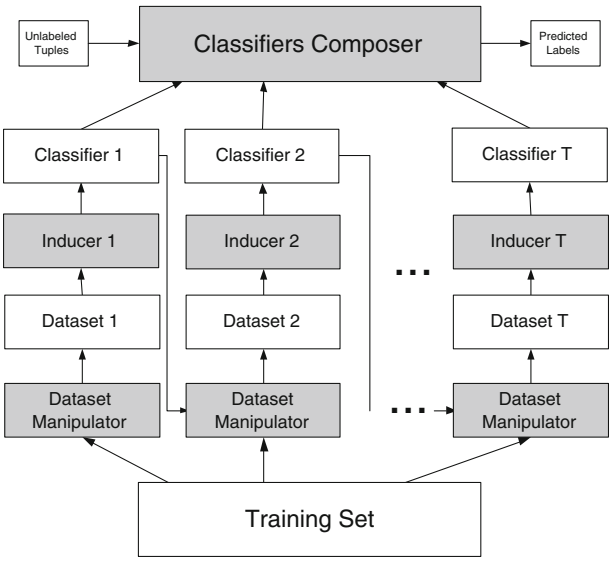

Fig. 1 Model guided instance selection diagram

2.1 Dependent methods

We distinguish between two main approaches for dependent learning, (Provost and Kolluri 1997):

- Incremental Batch Learning—In this method the classification produced in one iteration is given as "prior knowledge" to the learning algorithm in the following iteration. The learning algorithm uses the current training set together with the classification of the former classifier for building the next classifier. The classifier constructed at the last iteration is chosen as the final classifier.
- Model-guided Instance Selection—In this dependent approach, the classifiers that were constructed in previous iterations are used for manipulating the training set for the following iteration (see Fig. 1). One can embed this process within the basic learning algorithm. These methods usually ignore all data instances on which their initial classifier is correct and only learn from misclassified instances.

The most well known model-guided instance selection is boosting. Boosting (also known as arcing—Adaptive Resampling and Combining) is a general method for improving the performance of a weak learner (such as classification rules or decision trees). The method works by repeatedly running a weak learner (such as classification rules or decision trees), on various distributed training data. The classifiers produced by the weak learners are then combined into a single composite strong classifier in order to achieve a higher accuracy than the weak learner's classifiers would have had.

AdaBoost (Adaptive Boosting), which was first introduced in Freund and Schapire (1996), is a popular ensemble algorithm that improves the simple boosting algorithm via an iterative process. The main idea behind this algorithm is to give more focus to patterns that are harder to classify. The amount of focus is quantified by a weight that is assigned to every pattern in the training set. Initially, the same weight is assigned to all the patterns. In each iteration the weights of all misclassified instances are increased while the weights of correctly classified

Require: I (a weak inducer), T (the number of iterations), S (training set)
Ensure: $M_t, \alpha_t; t = 1, \ldots, T$
1: $t \leftarrow 1$
2: $D_1(i) \leftarrow 1/m; i = 1, \ldots, m$
3: **repeat**
4: Build Classifier M_t using I and distribution D_t
5: $\varepsilon_t \leftarrow \sum_{i:M_t(x_i) \neq y_i} D_t(i)$
6: **if** $\varepsilon_t > 0.5$ **then**
7: $T \leftarrow t - 1$
8: exit Loop.
9: **end if**
10: $\alpha_t \leftarrow \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
11: $D_{t+1}(i) = D_t(i) \cdot e^{-\alpha_t y_t M_t(x_i)}$
12: Normalize D_{t+1} to be a proper distribution.
13: $t++$
14: **until** $t > T$

Fig. 2 The AdaBoost algorithm

instances are decreased. As a consequence, the weak learner is forced to focus on the difficult instances of the training set by performing additional iterations and creating more classifiers. Furthermore, a weight is assigned to every individual classifier. This weight measures the *overall accuracy* of the classifier and is a function of the total weight of the correctly classified patterns. Thus, higher weights are given to more accurate classifiers. These weights are used for the classification of new patterns.

This iterative procedure provides a series of classifiers that complement one another. In particular, it has been shown that AdaBoost approximates a large margin classifier such as the SVM (Rudin et al. 2004).

The pseudo-code of the AdaBoost algorithm is described in Fig. 2. The algorithm assumes that the training set consists of m instances, labeled as -1 or $+1$. The classification of a new instance is made by voting on all classifiers $\{M_t\}$, each having a weight of α_t. Mathematically, it can be written as:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot M_t(x)\right) \tag{1}$$

Arc-x4 is a simple arcing algorithm (Breiman 1998) which aims to demonstrate that AdaBoost works because of the adaptive resampling and not because of the specific form of the weighing function. In Arc-x4 the classifiers are combined by unweighted voting and the updated $t + 1$ iteration probabilities are defined by:

$$D_{t+1}(i) = 1 + m_{t_i}^4 \tag{2}$$

where m_{t_i} is the number of misclassifications of the i-th instance by the first t classifiers.

The basic AdaBoost algorithm, described in Fig. 2, deals with binary classification. Freund and Schapire describe two versions of the AdaBoost algorithm (AdaBoost.M1, AdaBoost.M2), which are equivalent for binary classification and differ in their handling of multiclass classification problems. Figure 3 describes the pseudo-code of AdaBoost.M1. The classification of a new instance is performed according to the following equation:

$$H(x) = \underset{y \in dom(y)}{\text{argmax}} \left(\sum_{t:M_t(x)=y} \log \frac{1}{\beta_t} \right) \tag{3}$$

where β_t is defined in Fig. 3.

Require: I (a weak inducer), T (the number of iterations), S (the training set)
Ensure: $M_t, \beta_t; t = 1, \ldots, T$
1: $t \leftarrow 1$
2: $D_1(i) \leftarrow 1/m; i = 1, \ldots, m$
3: **repeat**
4: Build Classifier M_t using I and distribution D_t
5: $\varepsilon_t \leftarrow \sum_{i:M_t(x_i)\neq y_i} D_t(i)$
6: **if** $\varepsilon_t > 0.5$ **then**
7: $T \leftarrow t - 1$
8: exit Loop.
9: **end if**
10: $\beta_t \leftarrow \frac{\varepsilon_t}{1-\varepsilon_t}$
11: $D_{t+1}(i) = D_t(i) \cdot \begin{cases} \beta_t & M_t(x_i) = y_i \\ 1 & Otherwise \end{cases}$
12: Normalize D_{t+1} to be a proper distribution.
13: $t++$
14: **until** $t > T$

Fig. 3 The AdaBoost.M.1 algorithm

Algorithm AdaBoost.M2

Require: I (a base inducer), T (number of iterations), S (the original training set), μ (the sample size).
1: Initialize the weight vector $D_1(i) \leftarrow 1/m \; i = 1, \ldots, m$ and $w_y^1 = D(i)/(k-1)$ for $i = 1, \ldots, m$; $y \in Y - y_i$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Set $W^f = \Sigma_{y\neq y} w_{\iota,y}^t$
4: $q_t(i, y) = \frac{w_y^f}{W^f}$ for $y \neq y,$;
5: Set $D_t(i) = \frac{W^f}{\Sigma_{=1}^N W^f}$.
6: Call I, providing it with the distribution D_t and label weighting function q_t; get back a hypothesis Mt: $x \times Y \rightarrow [0, 1]$.
7: Calculate the pseudo-loss of M_t: $\epsilon, = \frac{1}{2}\sum_{=1}^N D, (i)(1 - h, (x, , y,) + \sum_{y\neq y} q, (i, y)h, (x, , y))$.
8: Set $\beta, = \epsilon, /(1 - \epsilon,)$.
9: Set the new weights vector to be $w_y^{r+1} = w_y^f \beta_t^{(1/2)(1+h_t(xy)-h_t(x_\ell y))}$ for $i = 1, \ldots, N, \; y \in Y - \{y, \}$.
10: **end for**

Fig. 4 The AdaBoost.M2 algorithm

AdaBoost.M2 is a second alternative extension of AdaBoost to the multi-class case. This extension requires more elaborate communication between the boosting algorithm and the weak learning algorithm. AdaBoost.M2 uses the notion of pseudo-loss which measures the goodness of the weak hypothesis. The pseudocode of AdaBoost.M2 is presented in Fig. 4. A different weight $w_{i,y}^t$ is maintained for each instance i and each label $y \in Y - \{y_i\}$. The function $q = \{1, \ldots, N\} \times Y \rightarrow [0, 1]$, called the *label weighting function*, assigns to each example i in the training set a probability distribution such that, for each i: $\sum_{y\neq y} q(i, y) = 1$. The inducer gets both a distribution D_t and a label weight function q_t. The inducer's target is to minimize the pseudo-loss ϵ_t for given distribution D and weighting function q.

Friedman et al. (1998) present a revised version of AdaBoost, called Real AdaBoost. This algorithm aims to combine the output class probabilities provided by the base classifiers

using additive logistic regression model in a forward stagewise manner. The revision reduces computation cost and may lead to better performance.

Ivoting Breiman (1999) is an improvement on boosting that is less vulnerable to noise and overfitting. Further, since it does not require weighting the base classifiers, ivoting can be used in a parallel fashion, as demonstrated in Charnes et al. (2004)

All boosting algorithms presented here assume that the weak inducers which are provided can cope with weighted instances. If this is not the case, an unweighted dataset is generated from the weighted data by a resampling technique. Namely, instances are chosen with a probability according to their weights (until the dataset becomes as large as the original training set).

AdaBoost seems to improve the performance accuracy for two main reasons:

1. It generates a final classifier whose misclassification rate can be reduced by combining many classifiers whose misclassification rate may be high.
2. It produces a combined classifier whose variance is significantly lower than the variances produced by the weak base learner.

However, AdaBoost sometimes fails to improve the performance of the base inducer. According to Quinlan (1996), the main reason for AdaBoost's failure is overfitting. The objective of boosting is to construct a composite classifier that performs well on the data by iteratively improving the classification accuracy. Nevertheless, a large number of iterations may result in an overcomplex composite classifier, which is significantly less accurate than a single classifier. One possible way to avoid overfitting is to keep the number of iterations as small as possible.

Induction algorithms have been applied with practical success in many relatively simple and small-scale problems. However, most of these algorithms require loading the entire training set to the main memory. The need to induce from large masses of data, has caused a number of previously unknown problems, which, if ignored, may turn the task of efficient pattern recognition into mission impossible. Managing and analyzing huge datasets requires special and very expensive hardware and software, which often forces us to exploit only a small part of the stored data.

Huge databases pose several challenges:

– Computing complexity: Since most induction algorithms have a computational complexity that is greater than linear in the number of attributes or tuples, the execution time needed to process such databases might become an important issue.
– Poor classification accuracy due to difficulties in finding the correct classifier. Large databases increase the size of the search space, and this in turn increases the chance that the inducer will select an over-fitted classifier that is not valid in general.
– Storage problems: In most machine learning algorithms, the entire training set should be read from the secondary storage (such as magnetic storage) into the computer's primary storage (main memory) before the induction process begins. This causes problems since the main memory's capability is much smaller than the capability of magnetic disks.

Instead of training on a very large data base, Breiman (1999) proposes taking small pieces of the data, growing a classifier on each small piece and then combining these predictors together. Because each classifier is grown on a modestly-sized training set, this method can be used on large datasets. Moreover this method provides an accuracy which is comparable to that which would have been obtained if all data could have been held in main memory. Nevertheless, the main disadvantage of this algorithm, is that in most cases it will require many iterations to truly obtain a accuracy comparable to Adaboost.

Merler et al. (2007) developed the P-AdaBoost algorithm which is a distributed version of AdaBoost. Instead of updating the "weights" associated with instance in a sequential manner, P-AdaBoost works in two phases. In the first phase, the AdaBoost algorithm runs in its sequential, standard fashion for a limited number of steps. In the second phase the classifiers are trained in parallel using weights that are estimated from the first phase. P-AdaBoost yields approximations to the standard AdaBoost models that can be easily and efficiently distributed over a network of computing nodes.

Zhang and Zhang (2008) have recently proposed a new boosting-by-resampling version of Adaboost. In the local Boosting algorithm, a local error is calculated for each training instance which is then used to update the probability that this instance is chosen for the training set of the next iteration. After each iteration in AdaBoost, a global error measure is calculated that refers to all instances. Consequently noisy instances might affect the global error measure, even if most of the instances can be classified correctly. Local boosting aims to solve this problem by inspecting each iteration locally, per instance. A local error measure is calculated for each instance of each iteration, and the instance receives a score, which will be used to measure its significance in classifying new instances. Each instance in the training set also maintains a local weight, which controls its chances of being picked in the next iteration. Instead of automatically increasing the weight of a misclassified instance (like in AdaBoost), we first compare the misclassified instance with similar instances in the training set. If these similar instances are classified correctly, the misclassified instance is likely to be a noisy one that cannot contribute to the learning procedure and thus its weight is decreased. If the instance's neighbors are also misclassified, the instance's weight is increased. As in AdaBoost, if an instance is classified correctly, its weight is decreased. Classifying a new instance is based on its similarity with each training instance.

The advantages of local boosting compared to other ensemble methods are:

1. The algorithm tackles the problem of noisy instances. It has been empirically shown that the local boosting algorithm is more robust to noise than Adaboost.
2. In respect to accuracy, LocalBoost generally outperforms Adaboost. Moreover, LocalBoost outperforms Bagging and Random Forest when the noise level is small

The disadvantages of local boosting compared to other ensemble methods are:

1. When the amount of noise is large, LocalBoost sometimes performs worse than Bagging and Random Forest.
2. Saving the data for each instance increases storage complexity; this might confine the use of this algorithm to limited training sets.

AdaBoost.M1 algorithm guaranties an exponential decrease of an upper bound of the training error rate as long as the error rates of the base classifiers are less than 50%. For multiclass classification tasks, this condition can be too restrictive for weak classifiers like decision stumps. In order to make AdaBoost.M1 suitable to weak classifiers, BoostMA algorithm modifies it by using a different function to weight the classifiers (Freund 1995). Specifically, the modified function becomes positive if the error rate is less than the error rate of default classification. As opposed to AdaBoost.M2, where the weights are increased if the error rate exceeds 50%, in BoostMA the weights are increased for instances for which the classifier performed worse than the default classification (i.e. classification of each instance as the most frequent class). Moreover in BoostMA the base classifier minimizes the confidence-rated error instead of the pseudo-loss/error-rate (as in AdaBoost.M2 or Adaboost.M1) which makes it easier to use with already existing base classifiers.

AdaBoost-r is a variant of AdaBoost which considers not only the last weak classifier, but a classifier formed by the last r selected weak classifiers (r is a parameter of the method). If

AdaBoost-r Algorithm

Require: I (a base inducer), T (number of iterations), S (the original training set), ρ (whether to perform resampling or reweighting), r (reuse level).

1: Initiate: $D_1(X_i) = \frac{1}{m}$ for all i's.
2: $T = 1$
3: **repeat**
4: **if** ρ **then**
5: $S' = resample(S, D_t)$
6: **else**
7: $S' = S$
8: **end if**
9: Train base classifier M_t with I on S' with instance weights according to D_t
10: Combine classifiers M_t, $M_{t-1}, \ldots M_{max(t-r,1)}$ to create M_t^r.
11: Calculate the error rate ε_t of the combined classifier M_t^r on S
12: **if** $\varepsilon_t \geq 0.5$ or $\varepsilon_t = 0$ **then**
13: End;
14: **end if**
15: $\alpha_t = \frac{1-\varepsilon_t}{\varepsilon_t}$
16: **for** i=1 to m **do**
17: **if** $M_t^r(X_i) \neq Y_i$ **then**
18: $D_{t+1}(X_i) = D_t(X_i)\ \alpha_t$
19: **end if**
20: **end for**
21: Renormalize D_{t+1} so it will be a distribution
22: $t \leftarrow t + 1$
23: **until** $t > T$

Fig. 5 AdaBoost-r algorithm

the weak classifiers are decision stumps, the combination of r weak classifiers is a decision tree. A primary drawback of AdaBoost-r is that it will only be useful if the classification method does not generate strong classifiers.

Figure 5 presents the pseudocode of AdaBoost-r. In line 1 we initialize a distribution on the instances so that the sum of all weights is 1 and all instances obtain the same weight. In line 3 we perform a number of iterations according to the parameter T. In lines 4–8 we define the training set S' on which the base classifier will be trained. We check whether the resampling or reweighting version of the algorithm is required. If resampling was chosen, we perform a resampling of the training set according to the distribution D_t. The resampled set S' is the same size as S. However, the instances it contains were drawn from S with repetition with the probabilities according to D_t. Otherwise (if reweighting was chosen) we simply set S' to be the entire original dataset S. In line 9 we train a base classifier M_t from the base inducer I on S' while using as instance weights the values of the distribution D_t. In line 10 lies the significant change of the algorithm compared to the normal AdaBoost. Each of the instances of the original dataset S is classified with the base classifier M_t. This most recent classification is saved in the sequence of the last R classifications. Since we are dealing with a binary class problem, the class can be represented by a single bit (0 or 1). Therefore the sequence can be stored as a binary sequence with the most recent classification being appended as the least significant bit. This is how past base classifiers are combined (through the classification sequence). The sequence can be treated as a binary number representing a leaf in the combined classifier M_t^r to which the instance belongs. Each leaf has two buckets (one for each class). When an instance is assigned to a certain leaf, its weight is added to the bucket representing the instance's real class. Afterwards, the final class of each leaf of M_t^r is decided by the heaviest bucket. The combined classifier does not need

to be explicitly saved since it is represented by the final classes of the leaves and the base classifiers $M_t,\ M_{t-1}, \ldots M_{\max(t-r,1)}$. In line 11 the error rate ε_t of M_t^r on the original dataset S is computed by summing the weight of all the instances the combined classifier has misclassified and then dividing the sum by the total weight of all the instances in S. In line 12 we check whether the error rate is over 0.5 which would indicate the newly combined classifier is even worse than random or the error rate is 0 which indicates overfitting. In case resampling was used and an error rate of 0 was obtained, it could indicate an unfortunate resampling and so it is recommended to return to the resampling section (line 8) and retry (up to a certain number of failed attempts, e.g. 10). Line 15 is executed in case the error rate was under 0.5 and therefore we define α_t to be $\frac{1-\varepsilon_t}{\varepsilon_t}$. In lines 16–20 we iterate over all of the instances in S and update their weight for the next iteration (D_{t+1}). If the combined classifier has misclassified the instance, its weight is multiplied by α_t. In line 21, after the weights have been updated, they are renormalized so that D_{t+1} will be a distribution (i.e. all weights will sum to 1). This concludes the iteration and everything is ready for the next iteration.

For classifying an instance, we traverse each of the combined classifiers, classify the instance with it and receive either -1 or 1. The class is then multiplied by $\log(\alpha_t)$, which is the weight assigned to the classifier trained at iteration t, and added to a global sum. If the sum is positive, the class "1" is returned; if it is negative, "-1" is returned; and if it is 0, the returned class is random. This can also be viewed as summing the weights of the classifiers per class and returning the class with the maximal sum. Since we do not explicitly save the combined classifier $\mathrm{M_t^r}$, we obtain its classification by classifying the instance with the relevant base classifiers and using the binary classification sequence which is given by $(\mathrm{M_t(x)}, \mathrm{M_{t-1}(x)}, \ldots \mathrm{M_{max(t-r,1)}\ (x)})$ as a leaf index into the combined classifier and using the final class of the leaf as the classification result of $\mathrm{M_t^r}$.

AdaBoost.M1 is known to have problems when the base classifiers are weak, i.e. the predictive performance of each base classifier is not much higher than that of a random guessing.

AdaBoost.M1W is a revised version of AdaBoost.M1 that aims to improve its accuracy in such cases. The required revision results in a change of only one line in the pseudo-code of AdaBoost.M1. Specifically the new weight of the base classifier is defined as:

$$\alpha_t = \ln\left(\frac{(|dom\,(y)| - 1)\,(1 - \varepsilon_t)}{\varepsilon_t}\right) \tag{4}$$

where ε_t is the error estimation which is defined in the original AdaBoost.M1 and $|dom\,(y)|$ represents the number of classes. Note the above equation generalizes AdaBoost.M1 by setting $|dom\,(y)| = 2$

2.2 Independent methods

Figure 6 illustrates the independent ensemble methodology. In this methodology the original dataset is transformed into several datasets from which several classifiers are trained. The datasets created from the original training set may be disjointed (mutually exclusive) or overlapping. A combination method is then applied in order to output the final classification. Since the method for combining the results of induced classifiers is usually independent of the induction algorithms, it can be used with different inducers at each dataset. Moreover this methodology can be easily parallelized. These independent methods aim either at improving the predictive power of classifiers or decreasing the total execution time.

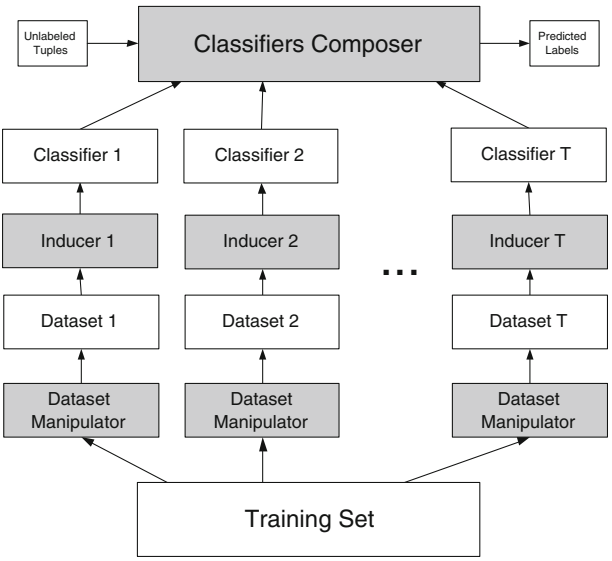

Fig. 6 Independent methods

Require: I (an inducer), T (the number of iterations), S (the training set), μ (the subsample size).
Ensure: $M_t; t = 1, \ldots, T$
1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow$ Sample μ instances from S with replacement.
4: Build classifier M_t using I on S_t
5: $t++$
6: **until** $t > T$

Fig. 7 The bagging algorithm

2.2.1 Bagging

The most well-known independent method is bagging (bootstrap aggregating). The method aims to increase accuracy by creating an improved composite classifier, I^*, by amalgamating the various outputs of learned classifiers into a single prediction.

Figure 7 presents the pseudo-code of the bagging algorithm (Breiman 1996). Each classifier is trained on a sample of instances taken with a replacement from the training set. Each sample size is equal to the size of the original training set.

Note that since sampling with replacement is used, some of the original instances of S may appear more than once in S_t and some may not be included at all.

So while the training sets in S_t may be different from each other, they are certainly not independent from a statistical point of view. To classify a new instance, each classifier returns the class prediction for the unknown instance. The composite bagged classifier, I^*, returns the class that has been predicted most often (voting method). The result is that bagging produces a combined model that often performs better than the single model built from the original single data. Breiman (1996) notes that this is true especially for unstable inducers because bagging can eliminate their instability. In this context, an inducer is considered unstable if perturbing the learning set can cause significant changes in the constructed classifier.

Require: I (an inducer), T (the number of iterations), S (the training set), d (weighting distribution).
Ensure: $M_t; t = 1, \ldots, T$
1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow S$ with random weights drawn from d.
4: Build classifier M_t using I on S_t
5: $t++$
6: **until** $t > T$

Fig. 8 The wagging algorithm

Require: IDT (a decision tree inducer), T (the number of iterations), S (the training set), μ (the subsample size). N (number of attributes used in each node)
Ensure: $M_t; t = 1, \ldots, T$
1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow$ Sample μ instances from S with replacement.
4: Build classifier M_t using $IDT(N)$ on S_t
5: $t++$
6: **until** $t > T$

Fig. 9 The random forest algorithm

Bagging, like boosting, is a technique that improves the accuracy of a classifier by generating a composite model that combines multiple classifiers all of which are derived from the same inducer. Both methods follow a voting approach, which is implemented differently, in order to combine the outputs of the different classifiers. In boosting, as opposed to bagging, each classifier is influenced by the performance of those that were built prior to its construction. Specifically, the new classifier pays more attention to classification errors that were done by the previously built classifiers where the amount of attention is determined according to their performance. In bagging, each instance is chosen with equal probability, while in boosting, instances are chosen with a probability that is proportional to their weight. Furthermore, as mentioned above, bagging requires an unstable learner as the base inducer, while in boosting inducer instability in not required, only that the error rate of every classifier be kept below 0.5.

2.2.2 Wagging

Wagging is a variant of bagging (Bauer and Kohavi 1999) in which each classifier is trained on the entire training set, but each instance is stochastically assigned a weight. Figure 8 presents the pseudo-code of the wagging algorithm.

2.2.3 Random forest

A Random Forest ensemble (also known as random subspace) (Breiman 2001) uses a large number of individual, unpruned decision trees. The individual trees are constructed using the algorithm presented in Fig. 9. The input parameter N represents the number of input variables that will be used to determine the decision at a node of the tree. This number should be much less than the number of attributes in the training set. Note that Bagging can be thought of as a special case of Random Forests obtained when N is set to the number of attributes in the original training set.

The IDT in Fig. 9 represents any top-down decision tree induction algorithm (Rokach and Maimon 2005a) with the following modification: the decision tree is not pruned and at each node, rather than choosing the best split among all attributes, the inducer randomly samples N of the attributes and choose the best split from among those variables (Rokach and Maimon 2008).

The classification of an unlabeled instance is performed using majority vote.

There are other ways to obtain random forests. For example, instead of using all the instances to determine the best split point for each feature, a sub-sample of the instances is used (Kamath and Cantu-Paz 2001). This sub-sample varies with the feature. The feature and split value that optimize the splitting criterion are chosen as the decision at that node. Since the split made at a node is likely to vary with the sample selected, this technique results in different trees which can be combined in ensembles.

Another method for randomization of the decision tree through histograms was proposed in Kamath et al. (2002). The use of histograms has long been suggested as a way of making the features discrete, while reducing the time to handle very large datasets. Typically, a histogram is created for each feature, and the bin boundaries used as potential split points. The randomization in this process is expressed by selecting the split point randomly in an interval around the best bin boundary.

Although the random forest was defined for decision trees, this approach is applicable to all types of classifiers. One important advantage of the random forest method is its ability to handle a very large number of input attributes (Skurichina and Duin 2002). Another important feature of the random forest is that it is fast.

2.2.4 Cross-validated committees

This procedure generates k classifiers by dividing the training set into k subsets and training, in turn, on all but the i-th set. Gams (1989) and Parmanto et al. (1996) employed 10-fold partitioning to construct an ensemble. Cross-validated committees can be used to scale existing induction algorithms (Domingos 1996). In this case, partitioning is applied by predetermining a maximum number of examples to which the algorithm can be applied at once. The full training set is randomly divided into approximately equal-sized partitions. Then each partition is trained separately. Each classifier induced from the partition p is tested on the instances in partition $p + 1$, in order to improve accuracy.

3 Combination methods

There are two main methods for combining the base-classifiers' outputs: weighting methods and meta-learning methods. Weighting methods are useful if the base-classifiers perform the same task and have comparable success. Meta-learning methods are best suited for cases in which certain classifiers consistently correctly classify, or consistently misclassify, certain instances.

3.1 Weighting methods

When combining classifiers with weights, a classifier's classification has a strength proportional to its assigned weight. The assigned weight can be fixed or dynamically determined for the specific instance to be classified.

3.1.1 Majority voting

In this combining scheme, a classification of an unlabeled instance is performed according to the class that obtains the highest number of votes (the most frequent vote). This method is also known as the plurality vote (PV) or the basic ensemble method (BEM). This approach has frequently been used as a combining method for comparing newly proposed methods.

Mathematically it can be written as:

$$class(x) = \underset{c_i \in dom(y)}{\arg\max} \left(\sum_k g\left(y_k(x), c_i\right) \right) \tag{5}$$

where $y_k(x)$ is the classification of the k'th classifier and $g(y, c)$ is an indicator function defined as:

$$g\left(y, c\right) = \begin{cases} 1 & y = c \\ 0 & y \neq c \end{cases} \tag{6}$$

If we are using probabilistic classifiers, the crisp classification $y_k(x)$ can be obtained as follows:

$$y_k(x) = \underset{c_i \in dom(y)}{\arg\max} \hat{P}_{M_k}(y = c_i \,|x) \tag{7}$$

where M_k represents classifier k and $\hat{P}_{M_k}(y = c \,|x)$ denotes the probability of class c given an instance x.

3.1.2 Performance weighting

The weight of each classifier can be set proportional to its accuracy performance on a validation set (Opitz and Shavlik 1996):

$$\alpha_i = \frac{(1 - E_i)}{\sum_{j=1}^{T}(1 - E_j)} \tag{8}$$

where E_i is a normalization factor which is based on the performance evaluation of classifier i on a validation set.

3.1.3 Distribution summation

The idea of the distribution summation combining method is to sum up the conditional probability vector obtained from each classifier (Clark and Boswell 1991). The selected class is chosen according to the highest value in the total vector. Mathematically, it can be written as:

$$Class(x) = \underset{c_i \in dom(y)}{\text{argmax}} \sum_k \hat{P}_{M_k}(y = c_i \,|x) \tag{9}$$

3.1.4 Bayesian combination

In the Bayesian combination method the weight associated with each classifier is the posterior probability of the classifier given the training set (Buntine 1990).

$$Class(x) = \underset{c_i \in dom(y)}{\text{argmax}} \sum_k P(M_k \,|S) \cdot \hat{P}_{M_k}(y = c_i \,|x) \tag{10}$$

where $P(M_k \,|S)$ denotes the probability that the classifier M_k is correct given the training set S. The estimation of $P(M_k \,|S)$ depends on the classifier's representation. To estimate this value for decision trees the reader is referred to Buntine (1990).

3.1.5 Dempster–Shafer

The idea of using the Dempster–Shafer theory of evidence (Buchanan and Shortliffe 1984) for combining classifiers has been suggested in Shilen (1990). This method uses the notion of basic probability assignment defined for a certain class c_i given the instance x:

$$bpa(c_i, x) = 1 - \prod_k \left(1 - \hat{P}_{M_k}(y = c_i \,|x)\right) \tag{11}$$

Consequently, the selected class is the one that maximizes the value of the belief function:

$$Bel(c_i, x) = \frac{1}{A} \cdot \frac{bpa(c_i, x)}{1 - bpa(c_i, x)} \tag{12}$$

where A is a normalization factor defined as:

$$A = \sum_{\forall c_i \in dom(y)} \frac{bpa(c_i, x)}{1 - bpa(c_i, x)} + 1 \tag{13}$$

3.1.6 Vogging

The idea of behind the vogging approach (Variance Optimized Bagging) is to optimize a linear combination of base-classifiers so as to aggressively reduce variance while attempting to preserve a prescribed accuracy (Derbeko et al. 2002). For this purpose, Derbeko et al. implemented the Markowitz Mean-Variance Portfolio Theory that is used for generating low variance portfolios of financial assets.

3.1.7 Naïve bayes

Using Bayes' rule, one can extend the Naïve Bayes idea for combining various classifiers:

$$Class(x) = \underset{\substack{c_j \in dom(y) \\ \hat{P}(y = c_j) > 0}}{\operatorname{argmax}} \hat{P}(y = c_j) \cdot \prod_{k=1} \frac{\hat{P}_{M_k}(y = c_j \,|x)}{\hat{P}(y = c_j)} \tag{14}$$

3.1.8 Entropy weighting

The idea in this combining method is to give each classifier a weight that is inversely proportional to the entropy of its classification vector.

$$Class(x) = \underset{c_i \in dom(y)}{\operatorname{argmax}} \sum_{k : c_i = \underset{c_j \in dom(y)}{\operatorname{argmax}} \hat{P}_{M_k}(y = c_j \,|x)} E(M_k, x) \tag{15}$$

where:

$$E(M_k, x) = -\sum_{c_j} \hat{P}_{M_k}(y = c_j \,|x) \log\left(\hat{P}_{M_k}(y = c_j \,|x)\right) \tag{16}$$

3.1.9 Density-based weighting

If the various classifiers were trained using datasets obtained from different regions of the instance space, it might be useful to weight the classifiers according to the probability of sampling x by classifier M_k, namely:

$$Class(x) = \underset{c_i \in dom(y)}{\operatorname{argmax}} \sum_{k:c_i = \underset{c_j \in dom(y)}{\operatorname{argmax}} \hat{P}_{M_k}(y=c_j|x)} \hat{P}_{M_k}(x) \tag{17}$$

The estimation of $\hat{P}_{M_k}(x)$ depends on the classifier representation and can not always be estimated.

3.1.10 DEA weighting method

Recently there has been attempts to use the data envelop analysis (DEA) methodology (Charnes et al. 1978) in order to assign weights to different classifiers (Sohn and Choi 2001). These researchers argue that the weights should not be specified according to a single performance measure, but should be based on several performance measures. Because there is a trade-off among the various performance measures, the DEA is employed in order to figure out the set of efficient classifiers. In addition, DEA provides inefficient classifiers with the benchmarking point.

3.1.11 Logarithmic opinion pool

According to the logarithmic opinion pool (Hansen 2000) the selection of the preferred class is performed according to:

$$Class(x) = \underset{c_j \in dom(y)}{\operatorname{argmax}} e^{\sum_k \alpha_k \cdot \log(\hat{P}_{M_k}(y=c_j|x))} \tag{18}$$

where α_k denotes the weight of the k-th classifier, such that:

$$\alpha_k \geq 0; \sum \alpha_k = 1 \tag{19}$$

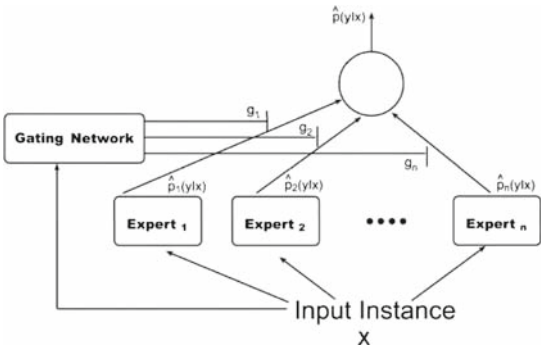

Fig. 10 Illustration of n-Expert structure

3.1.12 Gating network

Figure 10 illustrates an n-expert structure. Each expert outputs the conditional probability of the target attribute given the input instance. A gating network is responsible for combining the various experts by assigning a weight to each network. These weights are not constant but are functions of the input instance x. The gating network selects one or a few experts (classifiers) which appear to have the most appropriate class distribution for the example. In fact each expert specializes on a small portion of the input space.

An extension to the basic mixture of experts, known as hierarchical mixtures of experts (HME), has been proposed in Jordan and Jacobs (1994). This extension decomposes the space into sub-spaces, and then recursively decomposes each sub-space into sub-spaces.

Variations of the basic mixtures of experts methods have been developed to accommodate specific domain problems. A specialized modular networks called the Meta-p_i network has been used to solve the vowel-speaker problem (Hampshire and Waibel 1992; Peng et al. 1995). There have been other extensions to the ME such as nonlinear gated experts for time-series (Weigend et al. 1995); revised modular network for predicting the survival of AIDS patients (Ohno-Machado and Musen 1997); and a new approach for combining multiple experts for improving handwritten numeral recognition (Rahman and Fairhurst 1997).

3.1.13 Order statistics

Order statistics can be used to combine classifiers (Tumer and Ghosh 2000). These combiners offer the simplicity of a simple weighted combination method together with the generality of meta-combination methods (see the following section). The robustness of this method is helpful when there are significant variations among classifiers in some part of the instance space.

3.2 Meta-combination methods

Meta-learning means learning from the classifiers produced by the inducers and from the classifications of these classifiers on training data. The following sections describe the most well-known meta-combination methods.

3.2.1 Stacking

Stacking is a technique for achieving the highest generalization accuracy (Wolpert 1992). By using a meta-learner, this method tries to induce which classifiers are reliable and which are not. Stacking is usually employed to combine models built by different inducers. The idea is to create a meta-dataset containing a tuple for each tuple in the original dataset. However, instead of using the original input attributes, it uses the predicted classifications by the classifiers as the input attributes. The target attribute remains as in the original training set. A test instance is first classified by each of the base classifiers. These classifications are fed into a meta-level training set from which a meta-classifier is produced. This classifier combines the different predictions into a final one. It is recommended that the original dataset should be partitioned into two subsets. The first subset is reserved to form the meta-dataset and the second subset is used to build the base-level classifiers. Consequently the meta-classifier predications reflect the true performance of base-level learning algorithms. Stacking performance can be improved by using output probabilities for every class label from the base-level

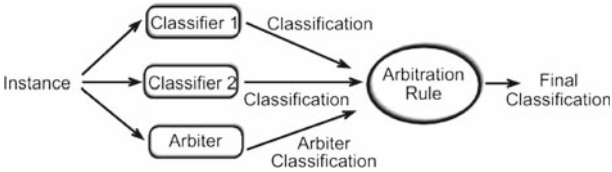

Fig. 11 A prediction from two base classifiers and a single arbiter

classifiers. In such cases, the number of input attributes in the meta-dataset is multiplied by the number of classes.

It has been shown that with stacking the ensemble performs (at best) comparably to selecting the best classifier from the ensemble by cross validation (Džeroski and Ženko 2004). In order to improve the existing stacking approach, they employed a new multi-response model tree to learn at the meta-level and empirically showed that it performs better than existing stacking approaches and better than selecting the best classifier by cross-validation.

The SCANN (Stacking, Correspondence Analysis and Nearest Neighbor) combining method (Merz 1999) uses the strategies of stacking and correspondence analysis. Correspondence analysis is a method for geometrically modeling the relationship between the rows and columns of a matrix whose entries are categorical. In this context Correspondence Analysis is used to explore the relationship between the training examples and their classification by a collection of classifiers.

A nearest neighbor method is then applied to classify unseen examples. Here, each possible class is assigned coordinates in the space derived by Correspondence Analysis. Unclassified examples are mapped into the new space, and the class label corresponding to the closest class point is assigned to the example.

3.2.2 Arbiter trees

According to Chan and Stolfo's (1993) approach, an arbiter tree is built in a bottom-up fashion. Initially, the training set is randomly partitioned into k disjoint subsets. The arbiter is induced from a pair of classifiers and recursively a new arbiter is induced from the output of two arbiters. Consequently for k classifiers, there are $\log_2(k)$ levels in the generated arbiter tree.

The creation of the arbiter is performed as follows. For each pair of classifiers, the union of their training dataset is classified by the two classifiers. A selection rule compares the classifications of the two classifiers and selects instances from the union set to form the training set for the arbiter. The arbiter is induced from this set with the same learning algorithm used in the base level. The purpose of the arbiter is to provide an alternate classification when the base classifiers present diverse classifications. This arbiter, together with an arbitration rule, decides on a final classification outcome, based upon the base predictions. Figure 11 shows how the final classification is selected based on the classification of two base classifiers and a single arbiter.

The process of forming the union of data subsets; classifying it using a pair of arbiter trees; comparing the classifications; forming a training set; training the arbiter; and picking one of the predictions, is recursively performed until the root arbiter is formed. Figure 12 illustrate an arbiter tree created for $k = 4$. $T_1 - T_4$ are the initial four training datasets from which four classifiers $M_1 - M_4$ are generated concurrently. T_{12} and T_{34} are the training sets

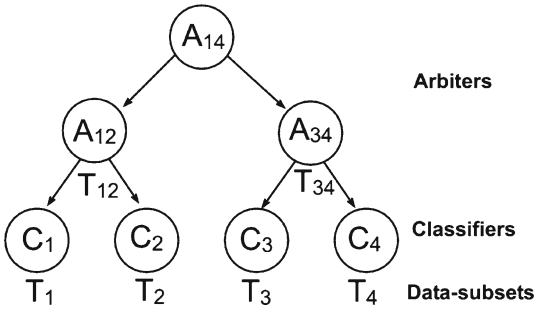

Fig. 12 Sample arbiter tree

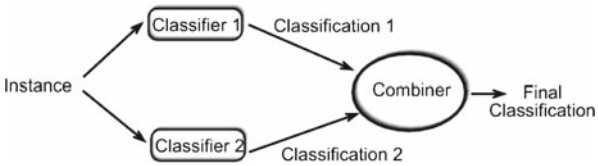

Fig. 13 A prediction from two base classifiers and a single combiner

generated by the rule selection from which arbiters are produced. A_{12} and A_{34} are the two arbiters. Similarly, T_{14} and A_{14} (root arbiter) are generated and the arbiter tree is completed.

There are several schemes for arbiter trees; each is characterized by a different selection rule. Here are three versions of selection rules:

- Only instances with classifications that disagree are chosen (group 1).
- Like group 1 defined above, plus instances where their classifications agree but are incorrect (group 2).
- Like groups 1 and 2 defined above, plus instances that have the same correct classifications (group 3).

Of the two versions of arbitration rules that have been implemented, each corresponds to the selection rule used for generating the training data at that level:

- For selection rule 1 and 2, a final classification is made by a majority vote of the classifications of the two lower levels and the arbiter's own classification, with preference given to the latter.
- For selection rule 3, if the classifications of the two lower levels are not equal, the classification made by the sub-arbiter based on the first group is chosen. In case this is not true and the classification of the sub-arbiter constructed on the third group equals those of the lower levels, then this is the chosen classification. In any other case, the classification of the sub-arbiter constructed on the second group is chosen. In fact it is possible to achieve the same accuracy level as in the single mode applied to the entire dataset but with less time and memory requirements (Chan and Stolfo 1993). More specifically it has been shown that this meta-learning strategy required only around 30% of the memory used by the single model case. This last fact, combined with the independent nature of the various learning processes, make this method robust and effective for massive amounts of data. Nevertheless, the accuracy level depends on several factors such as the distribution of the data among the subsets and the pairing scheme of learned classifiers and arbiters in each level. The decision regarding any of these issues may influence performance, but the optimal decisions are not necessarily known in advance, nor initially set by the algorithm.

3.2.3 Combiner trees

The way combiner trees are generated is very similar to arbiter trees. Both are trained bottom-up. However, a combiner, instead of an arbiter, is placed in each non-leaf node of a combiner tree (Chan and Stolfo 1997). In the combiner strategy, the classifications of the learned base classifiers form the basis of the meta-learner's training set. A composition rule determines the content of training examples from which a combiner (meta-classifier) will be generated. Figure 13 illustrates the result obtained from two base classifiers and a single combiner.

Two schemes for composition rules were proposed. The first one is the stacking scheme. The second is like stacking with the addition of the instance input attributes. It has been shown that the stacking scheme per se does not perform as well as the second scheme (Chan and Stolfo 1995). Although there is information loss due to data partitioning, combiner trees can sustain the accuracy level achieved by a single classifier. In a few cases, the single classifier's accuracy was consistently exceeded.

3.2.4 Grading

This technique uses "graded" classifications as meta-level classes (Seewald and Fürnkranz 2001). The term "graded" is used in the sense of classifications that have been marked as correct or incorrect. The method transforms the classification made by the k different classifiers into k training sets by using the instances k times and attaching them to a new binary class in each occurrence. This class indicates whether the k-th classifier yielded a correct or incorrect classification, compared to the real class of the instance.

For every classifier, one meta-classifier is trained whose aim is to indicate when each classifier tends to misclassify a given instance. At classification time, each classifier tries to classify an unlabeled instance. The final classification is obtained by combining the outputs of the classifiers that are recognized as correct by the meta-classification schemes. Grading is considered to be generalization of cross-validation selection (Schaffer 1993), which partitions the training data into k sets, builds $k - 1$ classifiers by dropping one set at a time and then uses it to find a misclassification rate. Finally, the procedure simply chooses the classifier corresponding to the subset with the smallest misclassification. Grading tries to make this decision separately for each and every instance by using only those classifiers that are predicted to classify that instance correctly. The main difference between grading and combiners (or stacking) is that the former does not change the instance attributes by replacing them with class predictions or class probabilities (or adding them to it). Instead it modifies the class values. Furthermore, in grading several sets of meta-data are created, one for each base classifier. Several meta-level classifiers are learned from those sets.

The main difference between grading and arbiters is that arbiters use information about the disagreements of classifiers for selecting a training set; grading uses disagreement with the target function to produce a new training set.

4 Diversity generation

Diversity is a crucial condition for obtaining accurate ensembles (Tumer and Ghosh 1996; Krogh and Vedelsby 1995; Kuncheva 2005; Kuncheva and Whitaker 2003). According to Hu (2001), diversified classifiers lead to uncorrelated classifications, which in turn improve classification accuracy. However, in the classification context, there is no complete and agreed

upon theory to explain why and how diversity between individual models contribute toward overall ensemble accuracy (Brown et al. 2005).

Brown et al. (2005) suggested a taxonomy of methods for creating diversity in ensembles of classifiers: varying the starting points within the hypothesis space; varying the set of hypotheses that are accessible by the ensemble members (for instance by manipulating the training set); and varying the way each member traverses the space.

In this paper we distinguish between the following approaches.

1. Manipulating the Training Sample—We vary the input that is used by the inducer for training. Each member is trained from a different training set.
2. Manipulating the Inducer—We manipulate the way in which the base inducer is used. More specifically each ensemble member is trained with an inducer that is differently manipulated.
3. Changing the target attribute representation—Each classifier in the ensemble solve a different target concept.
4. Partitioning the search space—Each member is trained on a different search subspace.
5. Hybridization—Diversity is obtained by using various base inducers or ensemble strategies.

4.1 Manipulating the inducer

There are several methods which can be used to manipulate the inducer. Below we discuss the most popular strategies to manipulate the inducer.

4.1.1 Starting point in the hypothesis space

Diversity can be obtained by starting the search in the Hypothesis Space from different starting points. For example in neural networks, we can assign different initial weights to the network's links (Kolen and Pollack 1991). Experimental study indicate that the resulting networks differed in the number of cycles in which they took to converge upon a solution, and in whether they converged at all (Sharkey et al. 1995). While it is very simple way to gain diversity, it is now generally accepted that it is not sufficient for achieving good diversity (Brown et al. 2005).

4.1.2 Hypothesis space traversal

These techniques alter the way the induction algorithm traverses the hypothesis space (Brown et al. 2005). There are two strategies for manipulating the space traversal for gaining diversity:

– In Collective-Performance-based strategy the evaluation function used in the induction of each member is extended to include a penalty term that encourages diversity. The most studied penalty method is the Negative Correlation Learning (Liu 2005; Brown and Wyatt 2003; Rosen 1996). The idea of negative correlation learning is to encourage different individual classifiers in the ensemble to represent different subspaces of the problem. While simultaneously creating the classifiers, the classifiers may interact with each other in order to specialize (for instance by using a correlation penalty term in the error function to encourage such specialization).
– The idea in Random-based strategy is to "inject randomness" into the training procedure in order to increase the independence among the ensemble's classifiers. For example in

decision trees induction algorithm instead of selecting the best feature in each stage, it selects randomly (with equal probability) an attribute from the set of the best *n* attributes.

4.1.3 Manipulating the induction algorithm's parameters

Induction algorithm can be tuned by setting the values of several parameters. For example, the well known decision tree inducer C4.5 (Quinlan 1993) has the confidence level parameter that greatly affect learning. By executing the C4.5 algorithm several times, each time with different parameters values, we obtain diverse decision trees that can constitute the ensemble.

In neural networks, diversity can be obtained by using different number of nodes (Partridge and Yates 1996; Yates and Partridge 1996). Nevertheless, these researches concludes that variation in numbers of hidden nodes is not effective method of creating diversity in neural network ensembles. Nevertheless the CNNE algorithm (Islam et al. 2003) which simultaneously determines the ensemble size along with the number of hidden nodes in individual NNs, has produced improved results. Alternatively, one can use different network topologies. For example the Addemup algorithm (Opitz and Shavlik 1996) employes genetic algorithm to choose the topologies of the base classifiers. Addemup trains with standard backpropagation, then selects groups of networks with a good error diversity according to the measurement of diversity.

4.2 Manipulating the training samples

In this method, each classifier is trained on a different variation or subset of the original dataset. This method is useful for inducers whose variance-error factor is relatively large i.e. small variations in the training set can cause a major change in the trained classifier. Most popular ensemble procedures (such as bagging and boosting) belong to this category.

4.2.1 Sampling methods

The distribution of tuples among the different classifier could be random as in the bagging algorithm or in the arbiter trees. Other methods distribute the tuples based on the class distribution such that the class distribution in each subset is approximately the same as that in the entire dataset. It has been shown that proportional distribution as used in combiner trees (Chan and Stolfo 1993) can achieve higher accuracy than random distribution.

Instead of perform sampling with replacement, we can manipulate the weights that are assigned to each example in the training sample. In this case, the base induction algorithm should be able to take these weights into consideration.

4.2.2 Partitioning

Some argue that classic ensemble techniques (such as boosting and bagging) have limitations on massive datasets, because the size of the dataset can become a bottleneck (Charnes et al. 2004). Moreover, it is suggested that partitioning the datasets into random, disjoint partitions will not only overcome the issue of exceeding memory size, but will also lead to creating an ensemble of diverse and accurate classifiers, each built from a disjoint partition but with the aggregate processing all of the data (Maimon and Rokach 2005). This can improve performance in a way that might not be possible by subsampling. In fact, empirical studies have shown that the performance of the multiple disjoint partition

approach is equivalent to the performance obtained by popular ensemble techniques such as bagging. More recently a framework for building thousands of classifiers that are trained from small subsets of data in a distributed environment was proposed (Charnes et al. 2004). It has been empirically shown that this framework is fast, accurate, and scalable.

Clustering techniques can be used to partitioning the sample. The CBCD (cluster-based concurrent decomposition) algorithm (Rokach et al. 2005) first clusters the instance space by using the K-means clustering algorithm. Then, it creates disjoint sub-samples using the clusters in such a way that each sub-sample is comprised of tuples from all clusters and hence represents the entire dataset. An inducer is applied in turn to each sub-sample. A voting mechanism is used to combine the classifierć6s classifications. Experimental study indicates that the CBCD algorithm outperforms the bagging algorithm.

4.2.3 Creation

Methods that belong to this category, add artificial instances to the training set. These instances are used to gain diversity among the base classifier. The DECORATE algorithm (Melville and Mooney 2003) is a dependent approach in which the ensemble is generated iteratively, learning a classifier at each iteration and adding it to the current ensemble. The first classifier is trained by applying the base induction algorithm on the original training set. The remaining classifiers are trained on an artificial set that combines tuples from the original training set and also on some fabricated tuples. In each iteration, the input attribute values of the fabricated tuples are generated according to the original data distribution. However, the class values of these tuples are selected so as to differ maximally from the current ensemble classifications.

4.3 Manipulating the representation of the target attribute

In methods that manipulate the target attribute, instead of inducing a single complicated classifier, several classifiers with different and usually simpler representations of the target attribute are induced. This manipulation can be based on an aggregation of the original target's values (known as *Concept Aggregation*) or more complicated functions (known as *Function Decomposition*).

Classical concept aggregation replaces the original target attribute with a function, such that the domain of the new target attribute is smaller than the original one.

Concept aggregation has been used to classify free text documents into predefined topics (Buntine 1966). This application suggests breaking the topics up into groups (co-topics) and then, instead of predicting the document's topic directly, classifying the document into one of the co-topics. Another model is then used to predict the actual topic in that co-topic.

Function decomposition was originally developed in the Fifties and Sixties for designing switching circuits. It was even used as an evaluation mechanism for checker playing programs (Samuel 1967). This approach was later improved by Biermann et al. (1982). Recently the machine learning community has adopted this approach. A manual decomposition of the problem and an expert-assisted selection of examples to construct rules for the concepts in the hierarchy was studied in Michie (1995). Compared to standard decision tree induction techniques, structured induction exhibits about the same degree of classification accuracy with the increased transparency and lower complexity of the developed models. A general-purpose function decomposition approach for machine learning was proposed in Zupan et al. (1998). According to this approach, attributes are transformed into new concepts in an iterative manner to create a hierarchy of

concepts. A different function decomposition which can be applied in data mining is the Bi-Decomposition (Long 2003). In this approach the original function is decomposed into two decomposition functions that are connected by a two-input operator called a "gate". Each of the decomposition functions depends on fewer variables than the original function. Recursive bi-decomposition represents a function as a structure of interconnected gates.

4.3.1 Error-correcting output coding

Some machine learning algorithms are designed to solve binary classification tasks, i.e. to classify an instance into only two classes. In many real problems, however, we are required to differentiate between more than two classes. A multiclass classification task is essentially more challenging than a binary classification task, since the induced classifier must classify the instances into a larger number of classes, which also increases the likelihood for misclassification. Let us consider, for example, a balanced classification problem, with a similar number of data per class, with equiprobable classes and a random classifier. If the problem is binary, the probability of obtaining a correct classification is 50%. For four classes, this probability drops to 25%.

Several machine learning algorithms, such as SVM, were originally designed to solve only binary classification tasks. There are two main approaches for applying such algorithms to multiclass tasks. The first approach, which involves extending the algorithm, has been applied to SVMs (Weston and Watkins 1999). However, extending these algorithms into a multiclass version may be either impractical or, frequently, not easy to perform (Passerini et al. 2004). For SVMs, in particular, (Hsu and Lin 2002) observed that the reformulation of this technique into multiclass versions leads to high costs in training algorithms.

The second approach is to convert the multiclass task into an ensemble of binary classification tasks, whose results are then combined. The decomposition that has been performed can be generally represented by a code-matrix $\mathbf{M}$ (Dietterich and Bakiri 1995). There are several alternatives to decomposing the multiclass problem into binary subtasks.This matrix has k rows, representing codewords ascribed to each of the k classes in the multiclass task; the columns correspond to the desired outputs of the binary classifiers induced in the decomposition.

The most simple decomposition tactic is the one-against-one (1A1) decomposition also known as the round robin classification. It consists of building $k(k-1)/2$ classifiers, each distinguishing a pair of classes i and j, where $i \neq j$. To combine the outputs produced by these classifiers, a majority voting scheme can be applied. Each 1A1 classifier provides one vote to its preferred class. The final result is the class with most of the votes.

In certain cases, only a subset of all possible pairwise classifiers should be used. In such cases it is preferable to count the "against" votes, instead of counting the "for" votes since there is a greater chance of making a mistake when counting the latter. If a certain classifier attempts to differentiate between two classes, neither of which is the true class, counting the "for" votes inevitably causes a misclassification. On the other hand, voting against one of the classes will not result in misclassification in such cases. Moreover, in the 1A1 approach, the classification of a classifier for a pair of classes (i, j) does not provide useful information when the instance does not belong to classes i or j.

Another standard methodology is called one-against-all (1AA). Given a problem with k classes, k binary classifiers are generated. Each binary classifier is responsible for differentiating a class i from the remaining classes. The classification is usually chosen according to the class with the highest probability.

Error-correcting systems, which date back to the mid-20th century, have been used to increase the reliability of communication channels. Communication channels suffer from undesired noise that distorts the received message from the original. Due to noise, an arbitrary bit in a message will be changed from its original state. To reduce the possibility of this occurring, the submitted message should be encoded in such a way that simple errors can be detected, making it possible in some cases to correct the error at the receiver end.

Each submitted symbol is encoded as a different codeword that is known in advance to both ends. When a codeword is obtained, the receiver searches for the most similar codeword using the Hamming distance. The coding is designed in such a way that the probability the receiver will misidentify the correct codeword is low.

Dietterich and Bakiri (1995) employ communication techniques to transform multiclass problems into an ensemble of binary classification tasks. The idea is to transmit the correct class of a new instance via a channel composed of the instance attributes, the training data and the learning algorithm. Due to errors that may be present in the attributes, in the training data and/or failures in the classifier learning process, the class information can be disrupted. To provide the system with the ability to recover from these transmission errors, the class is codified by an error correcting code and each of its bits is transmitted separately, that is, through separate executions of the learning algorithm.

Accordingly, a distributed output code is used to represent the k classes associated to the multiclass problem. A codeword of length l is ascribed to each class. Commonly, the size of the codewords has more bits than needed in order to uniquely represent each class. The additional bits can be used to correct eventual classification errors. For this reason, the method is named error-correcting output coding (ECOC).

The generated codes are stored on a matrix $\mathbf{M} \in \{-1, +1\}^{kxl}$. The rows of this matrix represent the codewords of each class and the columns correspond to the desired outputs of the l binary classifiers $(f_1(\mathbf{x}), \ldots, f_l(\mathbf{x}))$ induced.

A new pattern $\mathbf{x}$ can be classified by evaluating the classifications of the l classifiers, which generate a vector $\mathbf{f}(\mathbf{x})$ of length l. This vector is then measured against the rows of $\mathbf{M}$. The instance is ascribed to the class with the smallest Hamming distance.

4.4 Partitioning the search space

The idea is that each member in the ensemble explores a different part of the search space. Thus, the original instance space is divided into several sub-spaces. Each sub-space is considered independently and the total model is a (possibly soft) union of such simpler models.

When using this approach, one should decide if the subspaces will overlap. At one extreme, the original problem is decomposed into several mutually exclusive sub-problems, such that each subproblem is solved using a dedicated classifier. In such cases, the classifiers may have significant variations in their overall performance over different parts of the input space (Tumer and Ghosh 2000). At the other extreme, each classifier solves the same original task. In such cases, "If the individual classifiers are then appropriately chosen and trained properly, their performances will be (relatively) comparable in any region of the problem space. Tumer and Ghosh (2000)". However, usually the sub-spaces may have soft boundaries, namely sub-spaces are allowed to overlap.

There are two popular approaches for search space manipulations: divide and conquer approaches and feature subset-based ensemble methods.

4.4.1 Divide and conquer

In the neural-networks community, (Nowlan and Hinton 1991) examined the mixture of experts (ME) approach, which partitions the instance space into several subspaces and assigns different experts (classifiers) to the different subspaces. The subspaces, in ME, have soft boundaries (i.e., they are allowed to overlap). A gating network then combines the experts' outputs and produces a composite decision (see Section 3.1.12 for more details).

Some researchers have used clustering techniques to partition the space (Rokach et al. 2003). The basic idea is to partition the instance space into mutually exclusive subsets using K-means clustering algorithm. An analysis of the results shows that the proposed method is well suited for datasets of numeric input attributes and that its performance is influenced by the dataset size and its homogeneity.

NBTree (Kohavi 1996) is an instance space decomposition method that induces a decision tree and a Naïve Bayes hybrid classifier. Naïve Bayes, which is a classification algorithm based on Bayes' theorem and a Naïve independence assumption, is very efficient in terms of its processing time. To induce an NBTree, the instance space is recursively partitioned according to attributes values. The result of the recursive partitioning is a decision tree whose terminal nodes are Naïve Bayes classifiers. Since subjecting a terminal node to a Naïve Bayes classifier means that the hybrid classifier may classify two instances from a single hyper-rectangle region into distinct classes, the NBTree is more flexible than a pure decision tree. In order to decide when to stop the growth of the tree, NBTree compares two alternatives in terms of error estimation—partitioning into a hyper-rectangle region and inducing a single Naïve Bayes classifier. The error estimation is calculated by cross-validation, which significantly increases the overall processing time. Although NBTree applies a Naïve Bayes classifier to decision tree terminal nodes, classification algorithms other than Naïve Bayes are also applicable. However, the cross-validation estimations make the NBTree hybrid computationally expensive for more time-consuming algorithms such as neural networks.

More recently (Cohen et al. 2007) generalizes the NBTree idea and examines a decision-tree framework for space decomposition. According to this framework, the original instance-space is hierarchically partitioned into multiple subspaces and a distinct classifier (such as neural network) is assigned to each subspace. Subsequently, an unlabeled, previously-unseen instance is classified by employing the classifier that was assigned to the subspace to which the instance belongs.

The divide and conquer approach includes many other specific methods such as local linear regression, CART/MARS, adaptive subspace models, etc (Johansen and Foss 1992; Ramamurti and Ghosh 1999; Holmstrom et al. 1997).

4.4.2 Feature subset-based ensemble methods

Another less common strategy for manipulating the search space is to manipulate the input attribute set. Feature subset based ensemble methods are those that manipulate the input feature set for creating the ensemble members (Maimon and Rokach 2002). The idea is to simply give each classifier a different projection of the training set. Tumer and Oza (2003) claim that feature subset-based ensembles potentially facilitate the creation of a classifier for high dimensionality data sets without the feature selection drawbacks mentioned above. Moreover, these methods can be used to improve the classification performance due to the

reduced correlation among the classifiers. Bryll et al. (2003) also indicate that the reduced size of the dataset implies faster induction of classifiers. Feature subset avoids the class under-representation which may happen in instance subsets methods such as bagging. There are three popular strategies for creating feature subset-based ensembles: random-based, reduct-based and collective-performance-based strategy.

Random-based strategy The most straightforward techniques for creating feature subset-based ensemble are based on random selection. Ho (1998) creates a forest of decision trees. The ensemble is constructed systematically by pseudo-randomly selecting subsets of features. The training instances are projected to each subset and a decision tree is constructed using the projected training samples. The process is repeated several times to create the forest. The classifications of the individual trees are combined by averaging the conditional probability of each class at the leaves (distribution summation). Ho shows that simple random selection of feature subsets may be an effective technique because the diversity of the ensemble members compensates for their lack of accuracy.

Bay (1999) proposed using simple voting in order to combine outputs from multiple KNN (K-Nearest Neighbor) classifiers, each having access only to a random subset of the original features. Each classifier employs the same number of features. Bryll et al. (2003) introduce attribute bagging (AB) which combine random subsets of features. AB first finds an appropriate subset size by a random search in the feature subset dimensionality. It then randomly selects subsets of features, creating projections of the training set on which the classifiers are trained. A technique for building ensembles of simple Bayesian classifiers in random feature subsets was also examined (Tsymbal and Puuronen 2002) for improving medical applications.

Feature set partitioning Feature set partitioning is a particular case of feature subset-based ensembles in which the subsets are pairwise disjoint subsets (Rokach 2008). At the same time, feature set partitioning generalizes the task of feature selection which aims to provide a single representative set of features from which a classifier is constructed. Feature set partitioning, on the other hand, decomposes the original set of features into several subsets and builds a classifier for each subset. Thus, a set of classifiers is trained such that each classifier employs a different subset of the original feature set. Subsequently, an unlabelled instance is classified by combining the classifications of all classifiers.

Several researchers have shown that the partitioning methodology can be appropriate for classification tasks with a large number of features (Rokach 2006; Kusiak 2000). The search space of a feature subset-based ensemble contains the search space of feature set partitioning, and the latter contains the search space of feature selection. Mutually exclusive partitioning has some important and helpful properties:

1. There is a greater possibility of achieving reduced execution time compared to non-exclusive approaches. Since most learning algorithms have computational complexity that is greater than linear in the number of features or tuples, partitioning the problem dimensionality in a mutually exclusive manner means a decrease in computational complexity (Provost and Kolluri 1997).
2. Since mutual exclusiveness entails using smaller datasets, the classifiers obtained for each sub-problem are smaller in size. Without the mutually exclusive restriction, each classifier can be as complicated as the classifier obtained for the original problem. Smaller classifiers contribute to comprehensibility and ease in maintaining the solution.
3. According to Bay (1999), mutually exclusive partitioning may help avoid some error correlation problems that characterize feature subset based ensembles. However (Sharkey 1996) argues that mutually exclusive training sets do not necessarily result in low error correlation. This point is true when each sub-problem is representative.

4. In feature subset-based ensembles, different classifiers might generate contradictive classifications using the same features. This inconsistency in the way a certain feature can affect the final classification may increase mistrust among end-users. Accordingly, (Rokach 2006) claims that end-users can grasp mutually exclusive partitioning much easier.
5. The mutually exclusive approach encourages smaller datasets which are generally more practicable. Some data mining tools can process only limited dataset sizes (for instance, when the program requires that the entire dataset will be stored in the main memory). The mutually exclusive approach can ensure that data mining tools can be scaled fairly easily to large data sets (Chan and Stolfo 1997).

In the literature there are several works that deal with feature set partitioning. In one research, the features are grouped according to the feature type: nominal value features, numeric value features and text value features (Kusiak 2000). A similar approach was also used for developing the linear Bayes classifier (Gama 2004). The basic idea consists of aggregating the features into two subsets: the first subset containing only the nominal features and the second only the continuous features.

In another research, the feature set was decomposed according to the target class (Tumer and Ghosh 1996). For each class, the features with low correlation relating to that class were removed. This method was applied on a feature set of 25 sonar signals where the target was to identify the meaning of the sound (whale, cracking ice, etc.). Feature set partitioning has also been used for radar-based volcano recognition (Cherkauer 1996). The researcher manually decomposed a feature set of 119 into 8 subsets. Features that were based on different image processing operations were grouped together. As a consequence, for each subset, four neural networks with different sizes were built. A new combining framework for feature set partitioning has been used for text-independent speaker identification (Chen et al. 1997). Other researchers manually decomposed the features set of a certain truck backer-upper problem and reported that this strategy has important advantages (Jenkins and Yuhas 1993).

A general framework that searches for helpful feature set partitioning structures has also been proposed (Rokach and Maimon 2005b). This framework nests many algorithms, two of which are tested empirically over a set of benchmark datasets. The first algorithm performs a serial search while using a new Vapnik-Chervonenkis dimension bound for multiple oblivious trees as an evaluating scheme. The second algorithm performs a multi-search while using a wrapper evaluating scheme. This work indicates that feature set decomposition can increase the accuracy of decision trees.

4.5 Multi-inducers

In Multi-Inducer strategy, diversity is obtained by using different types of inducers (Michalski and Tecuci 1994). Each inducer contains an explicit or implicit bias (Mitchell 1980) that leads it to prefer certain generalizations over others. Ideally, this multi-inducer strategy would always perform as well as the best of its ingredients. Even more ambitiously, there is hope that this combination of paradigms might produce synergistic effects, leading to levels of accuracy that neither atomic approach by itself would be able to achieve.

Most research in this area has been concerned with combining empirical approaches with analytical methods (see for instance Towell and Shavlik 1994. Woods et al. (1997) combine four types of base inducers (decision trees, neural networks, k-nearest neighbor, and quadratic Bayes). They then estimate local accuracy in the feature space to choose the appropriate classifier for a given new unlabled instance. Wang et al. (2000) examined the usefulness of

adding decision trees to an ensemble of neural networks. The researchers concluded that adding a few decision trees (but not too many) usually improved the performance. Langdon et al. (2002) proposed using Genetic Programming to find an appropriate rule for combining decision trees with neural networks.

Brodley (1995) proposed the model class selection (MCS) system. MCS fits different classifiers to different subspaces of the instance space, by employing one of three classification methods (a decision-tree, a discriminant function or an instance-based method). In order to select the classification method, MCS uses the characteristics of the underlined training-set, and a collection of expert rules. Brodley's expert-rules were based on empirical comparisons of the methods' performance (i.e., on prior knowledge).

5 Ensemble selection

An important aspect of ensemble methods is to determine how many base classifiers and which classifiers should be included in the final ensemble. Several algorithms, such as bagging, predetermine the ensemble size, by using a controlling parameter such as number of iterations, that can be set by the user. Other ensemble algorithms try to determine the best ensemble size while training. When new members are added to the ensemble, we check if the performance of the ensemble has improved. If it is not, the procedure stops and no new base classifier are trained. Usually these algorithms also have a controlling parameter which bounds the number of base classifiers in the ensemble. An algorithm that decides when a sufficient number of classification trees have been created was recently proposed (Robert et al. 2007). The algorithm uses the out-of-bag error estimate, and is shown to result in an accurate ensemble for those methods that incorporate bagging into the construction of the ensemble. Specifically, the algorithm works by first smoothing the out-of-bag error graph with a sliding window in order to reduce the variance. After the smoothing has been completed, the algorithm takes a larger window on the smoothed data points and determines the maximum accuracy within that window. It continues to process windows until the maximum accuracy within a particular window no longer increases. At this point, the stopping criterion has been reached and the algorithm returns the ensemble with the maximum raw accuracy from within that window.

It is sometimes useful to let the ensemble extend unlimitedly and then prune the ensemble in order to get more effective and compact ensembles (Rokach 2009). Liu et al. (2004) conducted an empirical study in order to understand how the ensemble accuracy and diversity are affected by the ensemble size. They showed that a small ensemble can be constructed from a larger one while maintaining the accuracy and diversity of the full ensemble. Ensemble selection is important for two reasons: efficiency and predictive performance (Tsoumakas et al. 2008). A large ensemble incurs a computational cost that is higher than that of a smaller one. Moreover, empirical examinations conducted by Margineantu and Dietterich (1997) indicate that pruned ensembles may be more accurate than the original ensemble. However, they also proved the constructing an ensemble whose performance is equivalent to that of a boosted ensemble is NP-hard. This is even hard to approximate (Tamon and Xiang 2000), and the selection may sacrifice the generalization ability of the final ensemble. After (Zhou et al. 2002) proved the "many-could-be-better-than-all" theorem, it has become wellknown that it is possible to get a small yet strong ensemble. This led to the introduction of many new ensemble selection methods. Roughly speaking, the two most popular approaches for selecting an ensemble subset employ Ranking-based and Search Based Methods (see Tsoumakas et al. 2008 for additional approaches).

5.1 Ranking-based approach

The idea behind this approach is to rank once the individual members according to a certain criterion and than choose the top classifiers whose rank is above a given threshold. For example, (Prodromidis et al. 1999) suggest ranking classifiers according to their classification performance on a separate validation set and their ability to correctly classify specific classes. Similarly, (Caruana et al. 2004) presented a forward stepwise selection procedure that selects the most relevant classifiers (maximizing the ensemble's performance) from thousands of classifiers. The algorithm FS-PP-EROS generates a selective ensemble of rough subspaces (Hu et al. 2007). The algorithm performs an accuracy-guided forward search to select the most relevant members. The experimental results show that FS-PP-EROS outperforms bagging and random subspace methods in terms of accuracy and size of ensemble systems. In attribute bagging (Bryll et al. 2003), classification accuracy of randomly selected m-attribute subsets is evaluated by using the wrapper approach. The ensemble voting considers only the classifiers constructed on the highest ranking subsets. Margineantu and Dietterich (1997) present an agreement-based ensemble selection. In this approach, the Kappa statistics is calculated for all pairs of classifiers. Pairs of classifiers are selected in ascending order of their agreement level until a desired ensemble size is reached.

5.2 Search-based methods

Instead of separately ranking the members, one can perform a heuristic search in the space of all possible ensemble subsets while evaluating the collective merit of a candidate subset. The GASEN algorithm was developed for selecting the most appropriate classifiers in a given ensemble (Zhou et al. 2002). Each classifier is assigned with a weight (initially random). Genetic algorithms are employed to evolve these weights so that they can characterize to some extent the fitness of the classifiers to join the ensemble. Finally, it removes from the ensemble the classifiers whose weight is less than a predefined threshold value. A revised version of the GASEN algorithm called GASEN-b was suggested by Zhou and Tang (2003). In this algorithm, instead of assigning a weight to each classifier, a bit is assigned to each classifier indicating whether it will be included in the final ensemble. Zhou and Tang conducted an experiment using C4.5 decision trees as the inducer. It was showed in this case that ensembles generated by their selective ensemble algorithm, may be not only be smaller in size but also have stronger generalization than ensembles generated by non-selective algorithms.

Rokach et al. (2006), Arbel and Rokach (2006) suggest first to rank the classifiers according to their ROC performance. Then, they suggest evaluating the performance of the ensemble subset by using the top ranked members. Members are gradually added to the ensemble until several successive additions do not improve the performance. Prodromidis and Stolfo (2001) introduce a backwards-correlation-based selection algorithm. The main idea is to remove the members that are the least correlated to a metaclassifier which is trained based on the classifiersć6 outputs. In each iteration, one member is removed and the new reduced meta-classifier is recomputed (with the remaining members). The meta-classifier in this case is used to evaluate the collective merit of the ensemble.

Windeatt and Ardeshir (2001) compared several subset evaluation methods that were applied to Boosting and Bagging. Specifically, the following pruning methods were compared: Minimum Error Pruning (MEP), Error-based Pruning (EBP), Reduced-Error Pruning (REP), Critical Value Pruning (CVP) and Cost-Complexity Pruning (CCP). The results indicate that on average ERP outperforms the other methods. Zhang et al. (2006) formulated the

ensemble selection problem as a quadratic integer programming problem in order to look for a subset of classifiers that has the optimal accuracy-diversity trade-off. Using a semi-definite programming (SDP) technique, they efficiently approximate the optimal solution, despite the fact that the quadratic problem is NP-hard.

6 Multistrategy ensemble learning

Multistrategy ensemble learning combines several ensemble strategies. It has been shown that this hybrid approach increases the diversity of ensemble members. For example, Non-Linear Boosting Projection (NLBP) combines boosting and subspace methods as follows (Garcia-Pddrajas et al. 2007):

1. All the classifiers receive as input all the training instances for learning. Since all are equally weighted, we are not placing more emphasis on misclassified instances as in boosting.
2. Each classifier uses a different nonlinear projection of the original data onto a space of the same dimension.
3. The nonlinear projection is based on the projection made by the hidden neuron layer of a multilayer perceptron neural network.
4. Following the basic principles of boosting, each nonlinear projection is constructed in order to make it easier to classify difficult instances.

Figure 14 presents the pseudocode of the NLBP algorithm. In lines 1–2 we convert the original data set (S) to a standard data set for multilayer perceptron learning (S'). It includes a transformation from nominal attributes to binary attributes (one binary attribute for each nominal value) and a normalization of all numeric (and binary) attributes to the $[-1, 1]$ range. In line 3 we construct the first classifier M_0, using the base inducer (I) and the converted data set S'. In lines 4–14 we construct the remaining $T - 1$ classifiers.

Each iteration includes the following steps. In lines 5–9 we try to classify each instance in S' with the previous classifier. S'' stores all the instances that we didn't classify correctly. In lines 10–11 we train a multilayer perceptron neural network using S''. The hidden layer

NLBP - Building the ensemble

Require: I (a base inducer), T (number of iterations), S (the original training set).
1: $S* =$ A nominal to binary transfomation of S
2: $S' =$ A nomalization of $S*$
3: $M_1 = I(S')$
4: **for** $t = 2$ to T **do**
5: $S'' = \emptyset$
6: **for** each $x_j \in S'$ **do**
7: **if** $M_{t-1}(x_j) \neq y_j$ **then**
8: $S'' = S'' \bigcup \{Xj\}$
9: **end if**
10: **end for**
11: Train network H using S'' and get the projection $P(X)$ implemented by hidden layer of H
12: $ProjectionArray[t] = P$
13: $S''' = P(S')$
14: $M_t = I(S''')$
15: **end for**

Fig. 14 NLBP—building the ensemble

of the network consists of the same number of hidden neurons as input attributes in S''. After training the network we retrieve the input weights of the hidden neurons, and store them in a data structure with the iteration index (ProjectionArray). In line 12 we project all the instances of S' with the projection P to get S'''. In line 13 we construct the current classifier using the converted and projected data set (S''') and the base inducer (I). In lines 15–16 we convert the instance (nominal to binary and normalization of attributes) as we did to the training data set. The final classification is a simple majority vote.

The advantages of NLBP compared to other ensemble methods are:

1. Experiments comparing NLBP to popular ensemble methods (Bagging, AdaBoost, LogitBoost, Arc-x4) using different base classifiers (C4.5, ANN, SVM) show very good results for NLBP.
2. Analysis of bagging, boosting, and NLBP by the authors of the paper suggests that: *"Bagging provides diversity, but to a lesser degree than boosting. On the other hand, boosting's improvement of diversity has the side-effect of deteriorating accuracy. NLBP behavior is midway between these two methods. It is able to improve diversity, but to a lesser degree than boosting, without damaging accuracy as much as boosting. This behavior suggests that the performance of NLBP in noisy problems can be better than the performance of boosting methods.*

The drawbacks of NLBP compared to other ensembles methods are:

1. (a) The necessity of training an additional neuronal network for each iteration increases the computational complexity of the algorithm compared to other approaches.
   (b) Using a neuronal network for projection may increase the dimensionality of the problem. Every nominal attribute is transformed to a set of binary ones.
   (c) The classifiers that are constructed by this approach use a different set of attributes than the original ones. Since these new attribute lose the meaning of the original ones, it is difficult to understand the meaning of the constructed models in terms of the original domain.

7 Choosing the best ensemble method for a problem in hand

There are several criteria that can help practitioners selecting the most suitable ensemble method for their specific needs. Nevertheless, the difficulty in choosing the ensemble methods results from the fact that this is a MCDM (Multiple Criteria Decision Making) problem. There are trade off relationships among the criteria and some criteria can not be measured in commensurate units. Thus, in order to systematically chose the right method, the practitioner is encouraged to implement one of the MCDM solving technique such as AHP (Analytic Hierarchy Process) (Saaty 1996).

The main selection criteria include:

- Accuracy—The classification accuracy provided by the ensemble.
- Computational Cost—The complexity cost for constructing the ensemble and in real time application also the time required for classifying an unseen instance.
- Scalability—The ability of the ensemble method to scale to large data sets.
- Flexibility—The ability to use any inducer (inducer-independent), or any combiner (Combiner-independent), in order to provide a solution to variety of classification tasks (for example it is should not be limited to a binary classification task).

AdaCost

Require: I (a base inducer), T (number of iterations), $S = \{(x_1, c_1, y_1), \ldots, (x_m, c_m, y_m)\}$. $x_i \in \mathcal{X}, c_i \in R^+, y_i \in \{-1, +1\}$

1: Initialize $D_1(i)$(T: such as $D_1(i) = c_i / \sum_j^m c_j$).
2: **repeat**
3: Train weak inducer using distribution D_t.
4: Compute weak classifier h_t: $\mathcal{X} \rightarrow R$.
5: Choose $\alpha_t \in R$ and $\beta(i) \in R^+$.
6: Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i) \beta(i))}{Z_t}$
7: $t \leftarrow t + 1$
8: **until** $t > T$

Fig. 15 AdaCost algorithm

- Usability—Providing a set of controlling parameters that are comprehensive and can be easily tuned.
- Interpretability—The user ability to understand the ensemble results.
- Software Availability—How many off-the-shelf software packages support this ensemble method. High Availability implies that the practitioner can move from one software to another, without the need to replace his ensemble method.

8 Ensemble methods for advanced classification tasks

8.1 Cost-sensitive classification

AdaBoost does not differentiate between the various classes. Thus, a misclassification in the majority class is treated equally as a misclassification of the minority class. However, in certain scenarios it is more desirable to augment the weight of misclassification errors of the minority class. For example, in direct marketing scenarios, firms are interested in estimating customer interest in their offer. However, positive response rates are usually low. For example, a mail marketing response rate of 2

$$D^{t+1}(i) = \frac{D^t(i) \sqrt{\frac{\sum_{[i, M_t(x_i) \neq y_i]} \delta \cdot W_i}{\sum_{[i, M_t(x_i) = y_i]} \delta \cdot W_i}}}{Z_t} \tag{20}$$

For unsuccessful classification, the distribution update is revised to:

$$D^{t+1}(i) = \frac{D^t(i) \Big/ \sqrt{\frac{\sum_{[i, M_t(x_i) \neq y_i]} \delta \cdot W_i}{\sum_{[i, M_t(x_i) = y_i]} \delta \cdot W_i}}}{Z_t} \tag{21}$$

where Z_t is a normalization factor.

Fan et al. (1999) presented AdaCost. The purpose of AdaCost is to improve AdaBoostć6s fixed and variable misclassification costs. It introduces a cost-adjustment function which is integrated into the weight updating rule. In addition to assigning high initial weights to costly instances, the weight updating rule takes cost into account and increases the weights of costly misclassification. Figure 15 presents the pseudocode of AdaCost. where $\beta(i) = \beta(\text{sign}(y_i h_t(x_i)), c_i)$ is a cost-adjustment function. Z_t is a normalization factor chosen so that D_{t+1} will be a distribution. The final classification is:

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

8.2 Ensemble for learning concept drift

Concept drift is an online learning task in which concepts change or drift over time. More specifically, concept drift occurs when the class distribution changes over time.

Concept drift exists in many applications that involve models of human behavior, such as recommender systems. Kolter and Maloof (2007) suggested an algorithm that tries to solve this problem by presenting an ensemble method for concept drift that dynamically creates and removes weighted experts according to a change in their performance. The suggested algorithm known as dynamic weighted majority (DWM) is an extension of the weighted majority algorithm (MWA) but it adds and removes base learners in response to global and local performance. As a result, DWM is better able to respond in non-stationary environments than other algorithms, especially those that rely on an ensemble of unweighted learners (such as SEA). The main disadvantage of DWM is its poor performance in terms of running time, compared to the AdaBoost algorithm.

8.3 Reject driven classification

Reject driven classification (Frelicot and Mascarilla 2001) is a method of classification that allows a tradeoff between misclassification and ambiguity (assigning more than one class to an instance). Specifically, the algorithm introduces a method for combining reject driven classifiers using belief theory methods. The algorithm adjusts the results of the reject driven classifiers by using the Dempster-Shafer theory. For each classifier, a basic probability assignment (BPA) is calculated to classify unseen instances.

The main strength of this algorithm is its ability to control the tradeoff between ambiguity and rejection. We can decide (with the proper threshold) if we prefer to classify an unseen instance to a single class and might be wrong or give an ambiguity classification. A major drawback of the algorithm is its inability to handle datasets with many classes since the BPA calculation needs to calculate the probability for any pair of classes.

9 Conclusions

Ensemble methodology imitates our second nature to seek several opinions before making a crucial decision. The core principle is to weigh several individual pattern classifiers, and combine them in order to reach a classification that is better than the one obtained by each of them separately. Researchers from various disciplines such as pattern recognition, statistics, and machine learning have explored the use of ensemble methods since the late seventies. Given the growing interest in the field, it is not surprising that researchers and practitioners have a wide variety of methods at their disposal. An ensemble is largely characterized by the diversity generation mechanism and the choice of its combination procedure. We hope that this tutorial will encourage the readers to explore ensemble methods for themselves.

References

Arbel R, Rokach L (2006) Classifier evaluation under limited resources. Pattern Recognit Lett 27(14):1619–1631

Banfield RE, Hall LO, Bowyer KW, Kegelmeyer WP (2007) A comparison of decision tree ensemble creation techniques. IEEE Trans Pattern Anal Mach Intell 29(1):173–180

Bauer E, Kohavi R (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 35:1–38

Bay S (1999) Nearest neighbor classification from multiple feature subsets. Intell Data Anal 3(3):191–209

Biermann AW, Faireld J, Beres T (1982) Signature table systems and learning. IEEE Trans Syst Man Cybern 12(5):635–648

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (1998) Arcing classifiers. Ann Stat 26(3):801–849

Breiman L (1999) Pasting small votes for classification in large databases and on-line. Mach Learn 36(2): 85–103

Breiman L (2001) Random forests. Mach Learn 45:5–32

Brodley CE (1995) Recursive automatic bias selection for classifier construction. Mach Learn 20:63–94

Bryll R, Gutierrez-Osuna R, Quek F (2003) Bagging: improving accuracy of classifier ensembles by using random feature subsets. Pattern Recognit 36:1291–1302

Brown G, Wyatt JL (2003) Negative correlation learning and the ambiguity family of ensemble methods. Proceedings of 4th international workshop, Mult Classifier Syst 2003, Guilford, UK, June 11–13, 2003, Lecture Notes in Computer Science, vol 2709, pp 266–275

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. Inf Fusion 6(1):5–20

Buchanan BG, Shortliffe EH (1984) Rule based expert systems. Addison-Wesley, Reading 272–292

Buntine W (1990) A theory of learning classification rules. Doctoral Dissertation. School of Computing Science, University of Technology. Sydney. Australia

Buntine W (1996) Graphical models for discovering knowledge. In: Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI/MIT Press, Cambridge pp 59–82

Caruana R, Niculescu-Mizil A, Crew G, Ksikes A (2004) Ensemble selection from libraries of models, twenty-first international conference on Machine learning, July 04–08, 2004, Banff, Alberta, Canada

Chan PK, Stolfo SJ (1993) Toward parallel and distributed learning by meta-learning. In: AAAI Workshop in Knowledge Discovery in Databases, pp 227–240

Chan PK, Stolfo SJ (1995) A comparative evaluation of voting and meta-learning on partitioned data, Proceeding of 12th international conference On machine learning ICML-95

Chan PK, Stolfo SJ (1997) On the accuracy of meta-learning for scalable data mining. J Intell Inf Syst 8:5–28

Charnes A, Cooper WW, Rhodes E (1978) Measuring the efficiency of decision making units. Eur J Oper Res 2(6):429–444

Chawla NV, Hall LO, Bowyer KW, Kegelmeyer WP (2004) Learning ensembles from bites: a scalable and accurate approach. J Mach Learn Res Arch 5:421–451

Chen K, Wang L, Chi H (1997) Methods of combining multiple classifiers with different features and their applications to text-independent speaker identification. Intern J Pattern Recognit Artif Intell 11(3): 417–445

Cherkauer KJ (1996) Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In: Working notes, integrating multiple learned models for improving and scaling machine learning algorithms workshop, thirteenth national conference on artificial intelligence. AAAI Press, Portland, OR

Clark P, Boswell R (1991) Rule induction with CN2: some recent improvements. In: Proceedings of the European working session on learning, Pitman, pp 151–163

Cohen S, Rokach L, Maimon O (2007) Decision tree instance space decomposition with grouped gain-ratio. Inf Sci 177:3592–3612

Dasarathy BV, Sheela BV (1979) Composite classifier system design: concepts and methodology. Proc IEEE 67(5):708–713

Derbeko P, El-Yaniv R, Meir R (2002) Variance optimized bagging. Eur Conf Mach Learn

Džeroski S, Ženko B (2004) Is combining classifiers with stacking better than selecting the best one? Mach Learn 54(3):255–273

Dietterich TG, Bakiri G (1995) Solving multiclass learning problems via error-correcting output codes. J Artif Intell Res 2:263–286

Dimitriadou E, Weingessel A, Hornik K (2003) A cluster ensembles framework, Design and application of hybrid intelligent systems. IOS Press, Amsterdam
Domingos P (1996) Using partitioning to speed up specific-to-general rule induction. In: Proceedings of the AAAI-96 workshop on integrating multiple learned models. AAAI Press, Cambridge, pp 29–34
Frelicot C, Mascarilla L (2001) Reject strategies driver combination of pattern classifiers
Freund S (1995) Boosting a weak learning algorithm by majority. Inf Comput 121(2):256–285
Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Machine learning: proceedings of the thirteenth international conference, pp 325–332
Friedman J, Hastie T, Tibshirani R (1998) Additive logistic regression: a statistical view of boosting
Gama J (2004) A linear-bayes classifier. In: Monard C (ed) Advances on artificial intelligence—SBIA2000. LNAI 1952, Springer, Berlin, pp 269–279
Gams M (1989) New measurements highlight the importance of redundant knowledge. In: European working session on learning, Montpeiller, France, Pitman
Garcia-Pddrajas N, Garcia-Osorio C, Fyfe C (2007) Nonlinear boosting projections for ensemble construction. J Mach Learn Res 8:1–33
Hampshire JB, Waibel A (1992) The meta-Pi network—building distributed knowledge representations for robust multisource pattern-recognition. Pattern Anal Mach Intell 14(7):751–769
Hansen LK, Salamon P (1990) Neural network ensembles. IEEE Trans Pattern Anal Mach Intell 12(10): 993–1001
Hansen J (2000) Combining predictors. Meta machine learning methods and bias, variance & ambiguity decompositions. PhD Dissertation. Aurhus University
Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844
Holmstrom L, Koistinen P, Laaksonen J, Oja E (1997) Neural and statistical classifiers—taxonomy and a case study. IEEE Trans Neural Netw 8:5–17
Hsu CW, Lin CJ (2002) A comparison of methods for multi-class support vector machines. IEEE Trans Neural Netw 13(2):415–425
Hu Q, Yu D, Xie Z, Li X (2007) EROS: ensemble rough subspaces. Pattern Recognit 40:3728–3739
Hu X (2001) Using rough sets theory and database operations to construct a good ensemble of classifiers for data mining applications. ICDM01. pp 233–240
Islam MM, Yao X, Murase K (2003) A constructive algorithm for training cooperative neural network ensembles. IEEE Trans Neural Netw 14(4):820–834
Jenkins R, Yuhas BPA (1993) Simplified neural network solution through problem decomposition: The case of Truck backer-upper. IEEE Trans Neural Netw 4(4):718–722
Johansen TA, Foss BA (1992) A narmax model representation for adaptive control based on local model—Modeling. Identif Control 13(1):25–39
Jordan MI, Jacobs RA (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Comput 6: 181–214
Kamath C, Cantu-Paz E (2001) Creating ensembles of decision trees through sampling, Proceedings, 33rd symposium on the interface of computing science and statistics, Costa Mesa, CA, June 2001
Kamath C, Cantú-Paz E, Littau D (2002) Approximate splitting for ensembles of trees using histograms. In: Second SIAM international conference on data mining (SDM-2002)
Kohavi R (1996) Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In: Proceedings of the second international conference on knowledge discovery and data mining, pp 114–119
Kolen JF, Pollack JB (1991) Back propagation is sesitive to initial conditions. In: Advances in neural information processing systems, vol 3. Morgan Kaufmann, San Francisco, pp 860–867
Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation and active learning. Adv Neural Inf Process Syst 7:231–238
Kuncheva L (2005) Combining pattern classifiers. Wiley Press, New York
Kuncheva L, Whitaker C (2003) Measures of diversity in classifier ensembles and their relationship with ensemble accuracy. Mach Learn 51(2):181–207
Kuncheva LI (2005) Diversity in multiple classifier systems (Editorial). Inf Fusion 6(1):3–4
Kusiak A (2000) Decomposition in data mining: an industrial case study. IEEE Trans Electron Packaging Manuf 23(4):345–353
Langdon WB, Barrett SJ, Buxton BF, (2002) Combining decision trees and neural networks for drug discovery. In: Genetic programming, proceedings of the 5th European conference, EuroGP 2002, Kinsale, Ireland, pp 60–70
Liu Y (2005) Generate different neural networks by negative correlation learning. ICNC (1): 149–156
Liu H, Mandvikar A, Mody J (2004) An empirical study of building compact ensembles. WAIM 622–627

Long C (2003) Bi-decomposition of function sets using multi-valued logic, Eng Doc Dissertation, Technischen Universitat Bergakademie Freiberg
Maimon O, Rokach L (2005) Decomposition methodology for knowledge discovery and data mining: theory and applications. World Scientific, Singapore
Maimon O, Rokach L (2002) Improving supervised learning by feature decomposition. In: Proceedings of foundations of information and knowledge systems, Salzan Castle, Germany, pp 178–196
Margineantu D, Dietterich T (1997) Pruning adaptive boosting. In: Proceedings of fourteenth international conference machine learning, pp 211–218
Melville P, Mooney RJ (2003) Constructing diverse classifier ensembles using artificial training examples. IJCAI 505–512
Merler S, Caprile B, Furlanello C (2007) Parallelizing AdaBoost by weights dynamics. Comput Stat Data Anal 51:2487–2498
Merz CJ (1999) Using correspondence analysis to combine classifier. Mach Learn 36(1–2):33–58
Michalski RS, Tecuci G (1994) Machine learning, a multistrategy approach. Morgan Kaufmann, San Francisco
Michie D (1995) Problem decomposition and the learning of skills, In: Proceedings of the European conference on machine learning, Springer, Berlin, pp 17–31
Mitchell T (1980) The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, Department of Computer Science, New Brunswick
Nowlan SJ, Hinton GE (1991) Evaluation of adaptive mixtures of competing experts. In: Lippmann RP, Moody JE, Touretzky DS (eds) Advances in neural information processing systems, vol 3. Morgan Kaufmann Publishers, San Francisco, pp 774–780
Ohno-Machado L, Musen MA (1997) Modular neural networks for medical prognosis: quantifying the benefits of combining neural networks for survival prediction. Connect Sci 9(1):71–86
Opitz D (1999) Feature selection for ensembles. In: Proceedings of 16th National Conference on Artificial Intelligence, AAAI. pp 379–384
Opitz D, Shavlik J (1996) Generating accurate and diverse members of a neuralá1network ensemble. In: Touretzky DS, Mozer MC, Hasselmo ME (eds) Advances in neural information processing systems, vol 8. The MIT Press, Cambridge, pp 535–541
Parmanto B, Munro PW, Doyle HR (1996) Improving committee diagnosis with resampling techinques. In: Touretzky DS, Mozer MC, Hesselmo ME (eds) Advances in neural information processing systems, vol 8. MIT Press, Cambridge, pp 882–888
Partridge D, Yates WB (1996) Engineering multiversion neural-net systems. Neural Comput 8(4):869–893
Passerini A, Pontil M, Frasconi P (2004) New results on error correcting output codes of kernel machines. IEEE Trans Neural Netw 15:45–54
Peng F, Jacobs RA, Tanner MA (1995) Bayesian inference in mixtures-of-experts and hierarchical mixtures-of-experts models with an application to speech recognition. J Am Stat Assoc 91(435):953–960
Polikar R (2006) Ensemble based systems in decision making. IEEE Circuits Syst Mag 6(3):21–45
Prodromidis AL, Stolfo SJ, Chan PK (1999) Effective and efficient pruning of metaclassifiers in a distributed Data Mining system. Technical report CUCS-017-99, Columbia University
Prodromidis AL, Stolfo SJ (2001) Cost complexity-based pruning of ensemble classifiers. Knowl Inf Syst 3(4):449–469
Provost FJ, Kolluri V (1997) A survey of methods for scaling up inductive learning algorithms. In: Proceeding of 3rd international conference on knowledge discovery and data mining
Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, Los Altos
Quinlan JR (1996) Bagging, Boosting, and C4.5. In: Proceedings of the thirteenth national conference on artificial intelligence, pp 725–730
Rahman AFR, Fairhurst MC (1997) A new hybrid approach in combining multiple experts to recognize handwritten numerals. Pattern Recognit Lett 18:781–790
Ramamurti V, Ghosh J (1999) Structurally adaptive modular networks for non-stationary environments. IEEE Trans Neural Netw 10(1):152–160
Rokach L (2006) Decomposition methodology for classification tasks—A meta decomposer framework. Pattern Anal Appl 9:257–271
Rokach L (2008) Genetic algorithm-based feature set partitioning for classification problems. Pattern Recognit 41(5):1676–1700
Rokach L (2009) Collective-agreement-based pruning of ensembles. Comput Stat Data Anal 53(4):1015–1026
Rokach L, Arbel R, Maimon O (2006) Selective voting—getting more for less in sensor fusion. Intern J Pattern Recognit Artif Intell 20(3):329–350
Rokach L, Maimon O (2005a) Top down induction of decision trees classifiers: A survey. IEEE SMC Trans Part C 35(4):476–487

Rokach L, Maimon O (2005b) Feature Set decomposition for decision trees. J Intell Data Anal 9(2):131–158
Rokach L, Maimon O (2008) Data mining with decision trees: theory and applications. World Scientific Publishing, Singapore
Rokach L, Maimon O, Arad O (2005) Improving supervised learning by sample decomposition. Int J Comput Intell Appl 5(1):37–54
Rokach L, Maimon O, Lavi I (2003) Space decomposition in data mining: A clustering approach. In: Proceedings of the 14th international symposium on methodologies for intelligent systems, Maebashi, Japan, Lecture Notes in Computer Science, Springer, Berlin, pp 24–31
Rudin C, Daubechies I, Schapire RE (2004) The dynamics of Adaboost: cyclic behavior and convergence of margins. J Mach Learn Res 5:1557–1595
Rosen BE (1996) Ensemble learning using decorrelated neural networks. Connect Sci 8(3):373–384
Samuel A (1967) Some studies in machine learning using the game of checkers II: Recent progress. IBM J Res Develop 11:601–617
Saaty X (1996) The analytic hierarchy process: A 1993 overview. Cent Eur J Oper Res Econ 2(2):119–137
Schaffer C (1993) Selecting a classification method by cross-validation. Mach Learn 13(1):135–143
Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
Seewald AK, Fürnkranz J (2001) Grading classifiers, Austrian research institute for Artificial intelligence
Sharkey A (1996) On combining artificial neural nets. Connect Sci 8:299–313
Sharkey N, Neary J, Sharkey A (1995) Searching weight space for backpropagation solution types, current trends in connectionism: Proceedings of the 1995 Swedish conference on connectionism, pp 103–120
Shilen S (1990) Multiple binary tree classifiers. Pattern Recognit 23(7):757–763
Skurichina M, Duin RPW (2002) Bagging, boosting and the random subspace method for linear classifiers. Pattern Anal Appl 5(2):121–135
Sohn SY, Choi H (2001) Ensemble based on data envelopment analysis. ECML Meta Learning workshop
Tamon C, Xiang J (2000) On the boosting pruning problem. In: Proceedings of the 11th European conference on machine learning, pp 404–412
Towell G, Shavlik J (1994) Knowledge-based artificial neural networks. Artif Intell 70:119–165
Tsoumakas G, Partalas I, Vlahavas I (2008) A taxonomy and short review of ensemble selection. In: ECAI 2008, workshop on supervised and unsupervised ensemble methods and their applications
Tsymbal A, Puuronen S (2002) Ensemble feature selection with the simple bayesian classification in medical diagnostics, In: Proceedings of 15th IEEE symposium on Computer-Based Medical Systems CBMS2002, IEEE CS Press, Maribor, Slovenia, pp 225–230
Tukey JW (1977) Exploratory data analysis. Addison-Wesley, Reading
Tumer K, Ghosh J (1996) Error correlation and error reduction in ensemble classifiers. Connection science, special issue on combining artificial neural networks: ensemble approaches. 8(3–4):385–404
Tumer K, Ghosh J (2000) Robust Order Statistics based Ensembles for Distributed Data Mining. In: Kargupta H, Chan P (eds) Advances in distributed and parallel knowledge discovery. AAAI/MIT Press, Cambridge pp 185–210
Tumer K, Oza CN (2003) Input decimated ensembles. Pattern Anal Appl 6:65–77
Wang W, Jones P, Partridge D (2000) Diversity between neural networks and decision trees for building multiple classifier systems, In: Proceeding of international workshop on multiple classifier systems (LNCS 1857), Springer, Calgiari, pp 240–249
Weigend AS, Mangeas M, Srivastava AN (1995) Nonlinear gated experts for time-series—discovering regimes and avoiding overfitting. Int J Neural Syst 6(5):373–399
Weston J, Watkins C (1999) Support vector machines for multi-class pattern recognition. In: Verleysen M (ed) Proceedings of the 7th European symposium on artificial neural networks (ESANN-99), Bruges, Belgium, pp 219–224
Windeatt T, Ardeshir G (2001) An Empirical comparison of pruning methods for ensemble classifiers, IDA2001, LNCS 2189, pp 208–217
Woods K, Kegelmeyer W, Bowyer K (1997) Combination of multiple classifiers using local accuracy estimates. IEEE Trans Pattern Anal Mach Intell 19:405–410
Wolpert DH (1992) Stacked generalization. Neural Networks, vol 5. Pergamon Press, Oxford, pp 241–259
Yanim S, Kamel MS, Wong AKC, Wang Y (2007) Cost-sensitive boosting for classification of imbalanced data. Pattern Recogn 40(12):3358–3378
Yates W, Partridge D (1996) Use of methodological diversity to improve neural network generalization. Neural Comput Appl 4(2):114–128
Zhang Y, Burer S, Street WN (2006) Ensemble pruning via semi-definite programming. J Mach Learn Res 7:1315–1338
Zhang CX, Zhang JS (2008) A local boosting algorithm for solving classification problems. Comput Stat Data Anal 52(4):1928–1941

Zhou ZH, Tang W (2003) Selective ensemble of decision trees. In: Wang G, Liu Q, Yao Y, Skowron A (eds) Rough sets, fuzzy sets, data mining, and granular computing, 9th international conference, RSFDGrC, Chongqing, China, Proceedings. Lecture Notes in Computer Science 2639, pp 476–483

Zhou ZH, Wu J, Tang W (2002) Ensembling neural networks: many could be better than all. Artif Intell 137:239–263

Zupan B, Bohanec M, Demsar J, Bratko I (1998) Feature transformation by function decomposition. IEEE Intell Syst Appl 13:38–43