Chapter 45

ENSEMBLE METHODS FOR CLASSIFIERS

Lior Rokach
Department of Industrial Engineering
Tel-Aviv University
liorr@eng.tau.ac.il

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. In this chapter we provide an overview of ensemble methods in classification tasks. We present all important types of ensemble methods including boosting and bagging. Combining methods and modeling issues such as ensemble diversity and ensemble size are discussed.

Keywords: Ensemble, Boosting, AdaBoost, Windowing, Bagging, Grading, Arbiter Tree, Combiner Tree

1. Introduction

The main idea of ensemble methodology is to combine a set of models, each of which solves the same original task, in order to obtain a better composite global model, with more accurate and reliable estimates or decisions than can be obtained from using a single model. The idea of building a predictive model by integrating multiple models has been under investigation for a long time. Bühlmann and Yu (2003) pointed out that the history of ensemble methods starts as early as 1977 with Tukeys Twicing, an ensemble of two linear regression models. Ensemble methods can be also used for improving the quality and robustness of clustering algorithms (Dimitriadou *et al*., 2003). Nevertheless, in this chapter we focus on classifier ensembles.

In the past few years, experimental studies conducted by the machine-learning community show that combining the outputs of multiple classifiers reduces the generalization error (Domingos, 1996; Quinlan, 1996; Bauer and Kohavi, 1999; Opitz and Maclin, 1999). Ensemble methods are very effective, mainly due to the phenomenon that various types of classifiers have differ-

ent "inductive biases" (Geman *et al*., 1995; Mitchell, 1997). Indeed, ensemble methods can effectively make use of such diversity to reduce the variance-error (Tumer and Ghosh, 1999; Ali and Pazzani, 1996) without increasing the bias-error. In certain situations, an ensemble can also reduce bias-error, as shown by the theory of large margin classifiers (Bartlett and Shawe-Taylor, 1998).

The ensemble methodology is applicable in many fields such as: finance (Leigh *et al*., 2002), bioinformatics (Tan *et al*., 2003), healthcare (Mangiameli *et al*., 2004), manufacturing (Maimon and Rokach, 2004), geography (Bruzzone *et al*., 2004) etc.

Given the potential usefulness of ensemble methods, it is not surprising that a vast number of methods is now available to researchers and practitioners. This chapter aims to organize all significant methods developed in this field into a coherent and unified catalog. There are several factors that differentiate between the various ensembles methods. The main factors are:

1. Inter-classifiers relationship — How does each classifier affect the other classifiers? The ensemble methods can be divided into two main types: sequential and concurrent.

2. Combining method — The strategy of combining the classifiers generated by an induction algorithm. The simplest combiner determines the output solely from the outputs of the individual inducers. Ali and Pazzani (1996) have compared several combination methods: uniform voting, Bayesian combination, distribution summation and likelihood combination. Moreover, theoretical analysis has been developed for estimating the classification improvement (Tumer and Ghosh, 1999). Along with simple combiners there are other more sophisticated methods, such as stacking (Wolpert, 1992) and arbitration (Chan and Stolfo, 1995).

3. Diversity generator — In order to make the ensemble efficient, there should be some sort of diversity between the classifiers. Diversity may be obtained through different presentations of the input data, as in bagging, variations in learner design, or by adding a penalty to the outputs to encourage diversity.

4. Ensemble size — The number of classifiers in the ensemble.

The following sections discuss and describe each one of these factors.

2. Sequential Methodology

In sequential approaches for learning ensembles, there is an interaction between the learning runs. Thus it is possible to take advantage of knowledge generated in previous iterations to guide the learning in the next iterations. We

distinguish between two main approaches for sequential learning, as described in the following sections (Provost and Kolluri, 1997).

2.1 Model-guided Instance Selection

In this sequential approach, the classifiers that were constructed in previous iterations are used for manipulating the training set for the following iteration. One can embed this process within the basic learning algorithm. These methods, which are also known as constructive or conservative methods, usually ignore all data instances on which their initial classifier is correct and only learn from misclassified instances.

The following sections describe several methods which embed the sample selection at each run of the learning algorithm.

2.1.1 Uncertainty Sampling. This method is useful in scenarios where unlabeled data is plentiful and the labeling process is expensive. We can define uncertainty sampling as an iterative process of manual labeling of examples, classifier fitting from those examples, and the use of the classifier to select new examples whose class membership is unclear (Lewis and Gale, 1994). A teacher or an expert is asked to label unlabeled instances whose class membership is uncertain. The pseudo-code is described in Figure 45.1.

Input: I (a method for building the classifier), b (the selected bulk size), U (a set on unlabled instances), E (an Expert capable to label instances)
Output: C

1: $X_{new} \leftarrow Random\ set\ of\ size\ b\ selected\ from\ U$
2: $Y_{new} \leftarrow E(X_{new})$
3: $S \leftarrow (X_{new}, Y_{new})$
4: $C \leftarrow I(S)$
5: $U \leftarrow U - X_{new}$
6: **while** E is willing to label instances **do**
7: $X_{new} \leftarrow$ Select a subset of U of size b such that C is least certain of its classification.
8: $Y_{new} \leftarrow E(X_{new})$
9: $S \leftarrow S \cup (X_{new}, Y_{new})$
10: $C \leftarrow I(S)$
11: $U \leftarrow U - X_{new}$
12: **end while**

Figure 45.1. Pseudo-Code for Uncertainty Sampling.

It has been shown that using uncertainty sampling method in text categorization tasks can reduce by a factor of up to 500 the amount of data that had to be labeled to obtain a given accuracy level (Lewis and Gale, 1994).

Simple uncertainty sampling requires the construction of many classifiers. The necessity of a cheap classifier now emerges. The cheap classifier selects instances "in the loop" and then uses those instances for training another, more expensive inducer. The *Heterogeneous Uncertainty Sampling* method achieves a given error rate by using a cheaper kind of classifier (both to build and run) which leads to reducted computational cost and run time (Lewis and Catlett, 1994).

Unfortunately, an uncertainty sampling tends to create a training set that contains a disproportionately large number of instances from rare classes. In order to balance this effect, a modified version of a C4.5 decision tree was developed (Lewis and Catlett, 1994). This algorithm accepts a parameter called loss ratio (LR). The parameter specifies the relative cost of two types of errors: false positives (where negative instance is classified positive) and false negatives (where positive instance is classified negative). Choosing a loss ratio greater than 1 indicates that false positives errors are more costly than the false negative. Therefore, setting the LR above 1 will counterbalance the overrepresentation of positive instances. Choosing the exact value of LR requires sensitivity analysis of the effect of the specific value on the accuracy of the classifier produced.

The original C4.5 determines the class value in the leaves by checking whether the split decreases the error rate. The final class value is determined by majority vote. In a modified C4.5, the leaf's class is determined by comparison with a probability threshold of LR/(LR+1) (or its appropriate reciprocal). Lewis and Catlett (1994) show that their method leads to significantly higher accuracy than in the case of using random samples ten times larger.

2.1.2 Boosting. Boosting (also known as arcing — Adaptive Resampling and Combining) is a general method for improving the performance of any learning algorithm. 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.

Schapire introduced the first boosting algorithm in 1990. In 1995 Freund and Schapire introduced the AdaBoost algorithm. The main idea of this algorithm is to assign a weight in each example in the training set. In the beginning, all weights are equal, but in every round, the weights of all misclassified instances are increased while the weights of correctly classified instances are decreased. As a consequence, the weak learner is forced to focus on the diffi-

cult instances of the training set. This procedure provides a series of classifiers that complement one another.

The pseudo-code of the AdaBoost algorithm is described in Figure 45.2. 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 $\{C_t\}$, each having a weight of α_t. Mathematically, it can be written as:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t \cdot C_t(x))$$

Input: I (a weak inducer), T (the number of iterations), S (training set)
Output: $C_t, \alpha_t; t = 1, \ldots, T$

1: $t \leftarrow 1$
2: $D_1(i) \leftarrow 1/m; i = 1, \ldots, m$
3: **repeat**
4: Build Classifier C_t using I and distribution D_t
5: $\varepsilon_t \leftarrow \sum_{i:C_t(x_i) \neq y_i} D_t(i)$
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: $D_{t+1}(i) = D_t(i) \cdot e^{-\alpha_t y_t C_t(x_i)}$
12: Normalize D_{t+1} to be a proper distribution.
13: $t++$
14: **until** $t > T$

Figure 45.2. The AdaBoost Algorithm.

The basic AdaBoost algorithm, described in Figure 45.2, deals with binary classification. Freund and Schapire (1996) 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 45.3 describes the pseudo-code of AdaBoost.M1. The classification of a new instance is performed according to the following equation:

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

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

Input: I (a weak inducer), T (the number of iterations), S (the training set)
Output: $C_t, \beta_t; t = 1, \ldots, T$

1: $t \leftarrow 1$
2: $D_1(i) \leftarrow 1/m; i = 1, \ldots, m$
3: **repeat**
4: Build Classifier C_t using I and distribution D_t
5: $\varepsilon_t \leftarrow \sum_{i:C_t(x_i) \neq y_i} D_t(i)$
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: $D_{t+1}(i) = D_t(i) \cdot \begin{cases} \beta_t & C_t(x_i) = y_i \\ 1 & Otherwise \end{cases}$
12: Normalize D_{t+1} to be a proper distribution.
13: $t++$
14: **until** $t > T$

Figure 45.3. The AdaBoost.M.1 Algorithm.

unweighted dataset is generated from the weighted data by a resampling technique. Namely, instances are chosen with probability according to their weights (until the dataset becomes as large as the original training set).

Boosting seems to improve performances for two main reasons:

1. It generates a final classifier whose error on the training set is small by combining many hypotheses whose error may be large.
2. It produces a combined classifier whose variance is significantly lower than those produced by the weak learner.

On the other hand, boosting sometimes leads to deterioration in generalization performance. According to Quinlan (1996) the main reason for boosting's failure is overfitting. The objective of boosting is to construct a composite classifier that performs well on the data, but a large number of iterations may create a very complex composite classifier, that is significantly less accurate than a single classifier. A possible way to avoid overfitting is by keeping the number of iterations as small as possible.

Another important drawback of boosting is that it is difficult to understand. The resulted ensemble is considered to be less comprehensible since the user is required to capture several classifiers instead of a single classifier. Despite the above drawbacks, Breiman (1996) refers to the boosting idea as the most significant development in classifier design of the nineties.

2.1.3 Windowing. Windowing is a general method aiming to improve the efficiency of inducers by reducing the complexity of the problem. It was initially proposed as a supplement to the ID3 decision tree in order to address complex classification tasks that might have exceeded the memory capacity of computers. Windowing is performed by using a sub-sampling procedure. The method may be summarized as follows: a random subset of the training instances is selected (a window). The subset is used for training a classifier, which is tested on the remaining training data. If the accuracy of the induced classifier is insufficient, the misclassified test instances are removed from the test set and added to the training set of the next iteration. Quinlan (1993) mentions two different ways of forming a window: in the first, the current window is extended up to some specified limit. In the second, several "key" instances in the current window are identified and the rest are replaced. Thus the size of the window stays constant. The process continues until sufficient accuracy is obtained, and the classifier constructed at the last iteration is chosen as the final classifier. Figure 45.4 presents the pseudo-code of the windowing procedure.

Input: I (an inducer), S (the training set), r (the initial window size), t (the maximum allowed windows size increase for sequential iterations).
Output: C

```
1:  Window ← Select randomly r instances from S.
2:  Test ← S-Window
3:  repeat
4:     C ← I(Window)
5:     Inc ← 0
6:     for all (x_i, y_i) ∈ Test do
7:        if C(x_i) ≠ y_i then
8:           Test ← Test − (x_i, y_i)
9:           Window = Window ∪ (x_i, y_i)
10:          Inc++
11:       end if
12:       if Inc = t then
13:          exit Loop
14:       end if
15:    end for
16: until Inc = 0
```

Figure 45.4. The Windowing Procedure.

The windowing method has been examined also for separate-and-conquer rule induction algorithms (Furnkranz, 1997). This research has shown that for this type of algorithm, significant improvement in efficiency is possible

in noise-free domains. Contrary to the basic windowing algorithm, this one removes all instances that have been classified by consistent rules from this window, in addition to adding all instances that have been misclassified. Removal of instances from the window keeps its size small and thus decreases induction time.

In conclusion, both windowing and uncertainty sampling build a sequence of classifiers only for obtaining an ultimate sample. The difference between them lies in the fact that in windowing the instances are labeled in advance, while in uncertainty, this is not so. Therefore, new training instances are chosen differently. Boosting also builds a sequence of classifiers, but combines them in order to gain knowledge from them all. Windowing and uncertainty sampling do not combine the classifiers, but use the best classifier.

2.2 Incremental Batch Learning

In this method the classifier produced in one iteration is given as "prior knowledge" to the learning algorithm in the following iteration (along with the subsample of that iteration). The learning algorithm uses the current subsample to evaluate the former classifier, and uses the former one for building the next classifier. The classifier constructed at the last iteration is chosen as the final classifier.

3. Concurrent Methodology

In the concurrent ensemble methodology, the original dataset is partitioned into several subsets from which multiple classifiers are induced concurrently. The subsets created from the original training set may be disjoint (mutually exclusive) or overlapping. A combining procedure is then applied in order to produce a single classification for a given instance. 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 subset. These concurrent methods aim either at improving the predictive power of classifiers or decreasing the total execution time. The following sections describe several algorithms that implement this methodology.

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

Figure 45.5 presents the pseudo-code of the bagging algorithm (Breiman, 1996). Each classifier is trained on a sample of instances taken with replacement from the training set. Usually each sample size is equal to the size of the original training set.

Input: I (an inducer), T (the number of iterations), S (the training set), N (the subsample size).
Output: $C_t; t = 1, \ldots, T$

1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow$ Sample N instances from S with replacment.
4: Build classifier C_t using I on S_t
5: $t++$
6: **until** $t > T$

Figure 45.5. The Bagging Algorithm.

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 the training sets S_t are different from each other, but are certainly not independent. 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. However, the bagging method is rather hard to analyze and it is not easy to understand by intuition what are the factors and reasons for the improved decisions.

Bagging, like boosting, is a technique for improving the accuracy of a classifier by producing different classifiers and combining multiple models. They both use a kind of voting for classification in order to combine the outputs of the different classifiers of the same type. In boosting, unlike bagging, each classifier is influenced by the performance of those built before, so the new classifier tries to pay more attention to errors that were made in the previous ones and to their performances. In bagging, each instance is chosen with equal probability, while in boosting, instances are chosen with probability proportional to their weight. Furthermore, according to Quinlan (1996), as mentioned above, bagging requires that the learning system should not be stable, where boosting does not preclude the use of unstable learning systems, provided that their error rate can be kept below 0.5.

3.0.2 Cross-validated Committees. This procedure creates k classifiers by partitioning the training set into k-equal-sized sets and in turn, training on all but the i-th set. This method, first used by Gams (1989), employed 10-fold partitioning. Parmanto *et al*. (1996) have also used this idea for creating an

ensemble of neural networks. Domingos (1996) has used cross-validated committees to speed up his own rule induction algorithm RISE, whose complexity is $O(n^2)$, making it unsuitable for processing large databases. 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. RISE is then run on each partition separately. Each set of rules grown from the examples in partition p is tested on the examples in partition $p+1$, in order to reduce overfitting and improve accuracy.

4. Combining Classifiers

The way of combining the classifiers may be divided into two main groups: simple multiple classifier combinations and meta-combiners. The simple combining methods are best suited for problems where the individual classifiers perform the same task and have comparable success. However, such combiners are more vulnerable to outliers and to unevenly performing classifiers. On the other hand, the meta-combiners are theoretically more powerful but are susceptible to all the problems associated with the added learning (such as over-fitting, long training time).

4.1 Simple Combining Methods

4.1.1 Uniform Voting. In this combining schema, each classifier has the same weight. A classification of an unlabeled instance is performed according to the class that obtains the highest number of votes. Mathematically it can be written as:

$$Class(x) = \underset{c_i \in dom(y)}{\text{argmax}} \sum_{\forall k c_i = \underset{c_j \in dom(y)}{\text{argmax}} \hat{P}_{M_k}(y=c_j|x)} 1$$

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

4.1.2 Distribution Summation. This combining method was presented by Clark and Boswell (1991). The idea is to sum up the conditional probability vector obtained from each classifier. The selected class is chosen according to the highest value in the total vector. Mathematically, it can be written as:

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

4.1.3 Bayesian Combination. This combining method was investigated by Buntine (1990). The idea is that the weight associated with each

classifier is the posterior probability of the classifier given the training set.

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

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. Buntine (1990) demonstrates how to estimate this value for decision trees.

4.1.4 Dempster–Shafer. The idea of using the Dempster–Shafer theory of evidence (Buchanan and Shortliffe, 1984) for combining models has been suggested by Shilen (1990; 1992). This method uses the notion of basic probability assignment defined for a certain class c_i given the instance x:

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

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

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

where A is a normalization factor defined as:

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

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

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

4.1.6 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.

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

where:

$$Ent(M_k, x) = -\sum_{c_j \in dom(y)} \hat{P}_{M_k}(y = c_j \,|x\,) \log\left(\hat{P}_{M_k}(y = c_j \,|x\,)\right)$$

4.1.7 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:

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

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

4.1.8 DEA Weighting Method. Recently there has been attempt to use the DEA (Data Envelop Analysis) methodology (Charnes *et al*., 1978) in order to assign weight to different classifiers (Sohn and Choi, 2001). They argue that the weights should not be specified based on a single performance measure, but 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.

4.1.9 Logarithmic Opinion Pool . According to the logarithmic opinion pool (Hansen, 2000) the selection of the preferred class is performed according to:

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

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

$$\alpha_k \geq 0; \sum \alpha_k = 1$$

4.1.10 Order Statistics. Order statistics can be used to combine classifiers (Tumer and Ghosh, 2000). These combiners have the simplicity of a simple weighted combining method with the generality of meta-combining 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.

4.2 Meta-combining 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-combining methods.

4.2.1 Stacking . Stacking is a technique whose purpose is to achieve the highest generalization accuracy. 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 classification of the classifiers as the input attributes. The target attribute remains as in the original training set.

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 will 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 performances could be improved by using output probabilities for every class label from the base-level classifiers. In such cases, the number of input attributes in the meta-dataset is multiplied by the number of classes.

Džeroski and Ženko (2004) have evaluated several algorithms for constructing ensembles of classifiers with stacking and show that the ensemble performs (at best) comparably to selecting the best classifier from the ensemble by cross validation. In order to improve the existing stacking approach, they propose to employ 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.

4.2.2 Arbiter Trees. This approach builds an arbiter tree in a bottom-up fashion (Chan and Stolfo, 1993). 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 45.6 shows how the final classification is selected based on the classification of two base classifiers and a single arbiter.

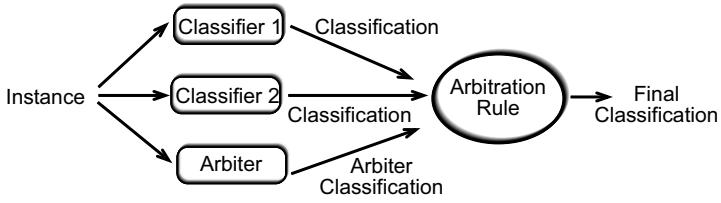

Figure 45.6. A Prediction from Two Base Classifiers and a Single Arbiter.

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 45.7 illustrate an arbiter tree created for $k = 4$. $T_1 - T_4$ are the initial four training datasets from which four classifiers $C_1 - C_4$ are generated concurrently. T_{12} and T_{34} are the training sets 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.

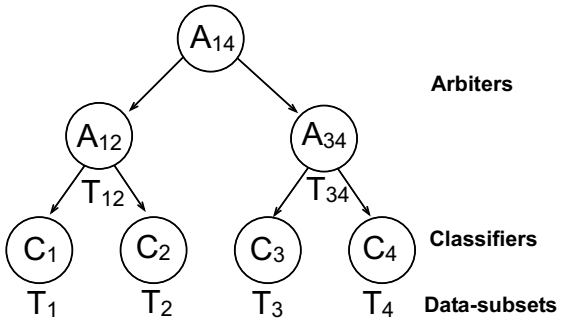

Figure 45.7. Sample Arbiter Tree.

Several schemes for arbiter trees were examined and differentiated from each other by the selection rule used. Here are three versions of rule selection:

- Only instances with classifications that disagree are chosen (group 1).
- Like group 1 defined above, plus instances that 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).

Two versions of arbitration rules have been implemented; each one 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. Chan and Stolfo (1993) achieved the same accuracy level as in the single mode applied to the entire dataset but with less time and memory requirements. 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 in any of these issues may influence performance, but the optimal decisions are not necessarily known in advance, nor initially set by the algorithm.

4.2.3 Combiner Trees. The way combiner trees are generated is very similar to arbiter trees. A combiner tree is 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. In classifying an instance, the base classifiers first generate their classifications and based on the composition rule, a new instance is generated. The aim of this strategy is to combine the classifications from the base classifiers by learning the relationship between these classifications and the correct classification. Figure 45.8 illustrates the result obtained from two base classifiers and a single combiner.

Two schemes of composition rule were proposed. The first one is the stacking schema. The second is like stacking with the addition of the instance input attributes. Chan and Stolfo (1995) showed that the stacking schema per se does not perform as well as the second schema. Although there is information loss due to data partitioning, combiner trees can sustain the accuracy level

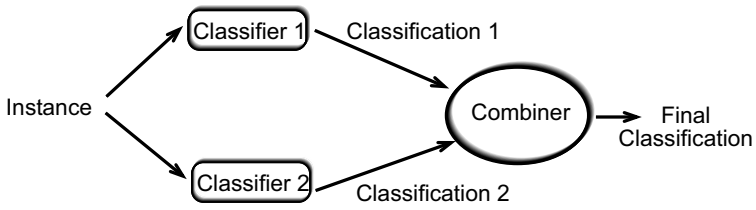

Figure 45.8. A Prediction from Two Base Classifiers and a Single Combiner.

achieved by a single classifier. In a few cases, the single classifier's accuracy was consistently exceeded.

4.2.4 Grading. This technique uses "graded" classifications as meta-level classes (Seewald and Furnkranz, 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 each base classifier, one meta-classifier is learned whose task is to classify when the base classifier will misclassify. At classification time, each base classifier classifies the unlabeled instance. The final classification is derived from the classifications of those base classifiers that are classified to be correct by the meta-classification schemes. In case several base classifiers with different classification results are classified as correct, voting, or a combination considering the confidence estimates of the base classifiers, is performed. Grading may be considered as a generalization of cross-validation selection (Schaffer, 1993), which divides the training data into k subsets, builds $k - 1$ classifiers by dropping one subset at a time and then using 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) are 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, while grading uses disagreement with the target function to produce a new training set.

5. Ensemble Diversity

In an ensemble, the combination of the output of several classifiers is only useful if they disagree about some inputs (Tumer and Ghosh, 1996). According to Hu (2001) diversified classifiers lead to uncorrelated errors, which in turn improve classification accuracy.

5.1 Manipulating the Inducer

A simple method for gaining diversity is to manipulate the inducer used for creating the classifiers. Ali and Pazzani (1996) propose to change the rule learning HYDRA algorithm in the following way: Instead of selecting the best literal in each stage (using, for instance, information gain measure), the literal is selected randomly such that its probability of being selected is proportional to its measure value. Dietterich (2000a) has implemented a similar idea for C4.5 decision trees. Instead of selecting the best attribute in each stage, it selects randomly (with equal probability) an attribute from the set of the best 20 attributes. The simplest way to manipulate the back-propagation inducer is to assign different initial weights to the network (Kolen and Pollack, 1991). MCMC (Markov Chain Monte Carlo) methods can also be used for introducing randomness in the induction process (Neal, 1993).

5.2 Manipulating the Training Set

Most ensemble methods construct the set of classifiers by manipulating the training instances. Dietterich (2000b) distinguishes between three main methods for manipulating the dataset.

5.2.1 Manipulating the Tuples. In this method, each classifier is trained on a different subset of the original dataset. This method is useful for inducers whose variance-error factor is relatively large (such as decision trees and neural networks), namely, small changes in the training set may cause a major change in the obtained classifier. This category contains procedures such as bagging, boosting and cross-validated committees.

The distribution of tuples among the different subsets 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. Proportional distribution was used in combiner trees (Chan and Stolfo, 1993). It has been

shown that proportional distribution can achieve higher accuracy than random distribution.

Recently Christensen et al. (2004) suggest a novel framework for construction of an ensemble in which each instance contributes to the committee formation with a fixed weight, while contributing with different individual weights to the derivation of the different constituent models. This approach encourages model diversity whilst not biasing the ensemble inadvertently towards any particular instance.

5.2.2 Manipulating the Input Feature Set. Another less common strategy for manipulating the training set is to manipulate the input attribute set. The idea is to simply give each classifier a different projection of the training set.

5.3 Measuring the Diversity

For regression problems *variance* is usually used to measure diversity (Krogh and Vedelsby, 1995). In such cases it can be easily shown that the ensemble error can be reduced by increasing ensemble diversity while maintaining the average error of a single model.

In classification problems, a more complicated measure is required to evaluate the diversity. Kuncheva and Whitaker (2003) compared several measures of diversity and concluded that most of them are correlated. Furthermore, it is usually assumed that increasing diversity may decrease ensemble error (Zenobi and Cunningham, 2001).

6. Ensemble Size

6.1 Selecting the Ensemble Size

An important aspect of ensemble methods is to define how many component classifiers should be used. This number is usually defined according to the following issues:

- Desired accuracy — Hansen (1990) argues that ensembles containing ten classifiers is sufficient for reducing the error rate. Nevertheless, there is empirical evidence indicating that in the case of AdaBoost using decision trees, error reduction is observed in even relatively large ensembles containing 25 classifiers (Opitz and Maclin, 1999). In the disjoint partitioning approaches, there may be a tradeoff between the number of subsets and the final accuracy. The size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective classifier. Chan and Stolfo (1993) varied

the number of subsets in the arbiter trees from 2 to 64 and examined the effect of the predetermined number of subsets on the accuracy level.

- User preferences — Increasing the number of classifiers usually increase computational complexity and decreases the comprehensibility. For that reason, users may set their preferences by predefining the ensemble size limit.

- Number of processors available — In concurrent approaches, the number of processors available for parallel learning could be put as an upper bound on the number of classifiers that are treated in paralleled process.

Caruana et al. (2004) presented a method for constructing ensembles from libraries of thousands of models. They suggest using forward stepwise selection in order to select the models that maximize the ensemble's performance. Ensemble selection allows ensembles to be optimized to performance metrics such as accuracy, cross entropy, mean precision, or ROC Area.

6.2 Pruning Ensembles

As in decision trees induction, it is sometime useful to let the ensemble grow freely and then prune the ensemble in order to get more effective and more compact ensembles. Empirical examinations indicate that pruned ensembles may obtain a similar accuracy performance as the original ensemble (Margineantu and Dietterich, 1997).

The efficiency of pruning methods when meta-combining methods are used have been examined in (Prodromidis *et al.*, 2000). In such cases the pruning methods can be divided into two groups: pre-training pruning methods and post-training pruning methods. Pre-training pruning is performed before combining the classifiers. Classifiers that seem to be attractive are included in the meta-classifier. On the other hand, post-training pruning methods, remove classifiers based on their effect on the meta-classifier. Three methods for pre-training pruning (based on an individual classification performance on a separate validation set, diversity metrics, the ability of classifiers to classify correctly specific classes) and two methods for post-training pruning (based on decision tree pruning and the correlation of the base classifier to the unpruned meta-classifier) have been examined in (Prodromidis *et al.*, 2000). As in (Margineantu and Dietterich, 1997), it has been shown that by using pruning, one can obtain similar or better accuracy performance, while compacting the ensemble.

The GASEN algorithm was developed for selecting the most appropriate classifiers in a given ensemble (Zhou *et al.*, 2002). In the initialization phase, GASEN assigns a random weight to each of the classifiers. Consequently, it

uses genetic algorithms to evolve those weights so that they can characterize to some extent the fitness of the classifiers in joining the ensemble. Finally, it removes from the ensemble those classifiers whose weight is less than a predefined threshold value. Recently a revised version of the GASEN algorithm called GASEN-b has been suggested (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 used in the final ensemble. They show that the obtained ensemble is not only smaller in size, but in some cases has better generalization performance.

Liu et al. (2004) conducted an empirical study of the relationship of ensemble sizes with ensemble accuracy and diversity, respectively. They show that it is feasible to keep a small ensemble while maintaining accuracy and diversity similar to those of a full ensemble. They proposed an algorithm called $LVFd$ that selects diverse classifiers to form a compact ensemble.

7. Cluster Ensemble

This chapter focused mainly on ensembles of classifiers. However ensemble methodology can be used for other Data Mining tasks such as regression and clustering.

The cluster ensemble problem refers to the problem of combining multiple partitionings of a set of instances into a single consolidated clustering. Usually this problem is formalized as a combinatorial optimization problem in terms of shared mutual information.

Dimitriadou et al. (2003) have used ensemble methodology for improving the quality and robustness of clustering algorithms. In fact they employ the same ensemble idea that has been used for many years in classification and regression tasks. More specifically they suggested various aggregation strategies and studied a greedy forward aggregation.

Hu and Yoo (2004) have used ensemble for clustering gene expression data. In this research the clustering results of individual clustering algorithms are converted into a distance matrix. These distance matrices are combined and a weighted graph is constructed according to the combined matrix. Then a graph partitioning approach is used to cluster the graph to generate the final clusters.

Strehl and Ghosh (2003) propose three techniques for obtaining high-quality cluster combiners. The first combiner induces a similarity measure from the partitionings and then reclusters the objects. The second combiner is based on hypergraph partitioning. The third one collapses groups of clusters into meta-clusters, which then compete for each object to determine the combined clustering. Moreover, it is possible to use supra-combiners that evaluate all three approaches against the objective function and pick the best solution for a given situation.

References

Ali K. M., Pazzani M. J., Error Reduction through Learning Multiple Descriptions, Machine Learning, 24: 3, 173-202, 1996.

Bartlett P. and Shawe-Taylor J., Generalization Performance of Support Vector Machines and Other Pattern Classifiers, In "Advances in Kernel Methods, Support Vector Learning", Bernhard Scholkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA, 1998.

Bauer, E. and Kohavi, R., "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants". Machine Learning, 35: 1-38, 1999.

Breiman L., Bagging predictors, Machine Learning, 24(2):123-140, 1996.

Bruzzone L., Cossu R., Vernazza G., Detection of land-cover transitions by combining multidate classifiers, Pattern Recognition Letters, 25(13): 1491–1500, 2004.

Buchanan, B.G. and Shortliffe, E.H., Rule Based Expert Systems, 272-292, Addison-Wesley, 1984.

Buhlmann, P. and Yu, B., Boosting with L_2 loss: Regression and classification, Journal of the American Statistical Association, 98, 324338. 2003.

Buntine, W., A Theory of Learning Classification Rules. Doctoral dissertation. School of Computing Science, University of Technology. Sydney. Australia, 1990.

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

Chan P. K. and Stolfo, S. J., Toward parallel and distributed learning by meta-learning, In AAAI Workshop in Knowledge Discovery in Databases, pp. 227-240, 1993.

Chan P.K. and Stolfo, S.J., A Comparative Evaluation of Voting and Meta-learning on Partitioned Data, Proc. 12th Intl. Conf. On Machine Learning ICML-95, 1995.

Chan P.K. and Stolfo S.J, On the Accuracy of Meta-learning for Scalable Data Mining, J. Intelligent Information Systems, 8:5-28, 1997.

Charnes, A., Cooper, W. W., and Rhodes, E., Measuring the efficiency of decision making units, European Journal of Operational Research, 2(6):429-444, 1978.

Christensen S. W. , Sinclair I., Reed P. A. S., Designing committees of models through deliberate weighting of data points, The Journal of Machine Learning Research, 4(1):39–66, 2004.

Clark, P. and Boswell, R., "Rule induction with CN2: Some recent improvements." In Proceedings of the European Working Session on Learning, pp. 151-163, Pitman, 1991.

Džeroski S., Ženko B., Is Combining Classifiers with Stacking Better than Selecting the Best One?, Machine Learning, 54(3): 255–273, 2004.

Dietterich, T. G., An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. 40(2):139-157, 2000.

Dietterich T., Ensemble methods in machine learning. In J. Kittler and F. Roll, editors, First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science, pages 1-15. Springer-Verlag, 2000

Dimitriadou E., Weingessel A., Hornik K., A cluster ensembles framework, Design and application of hybrid intelligent systems, IOS Press, Amsterdam, The Netherlands, 2003.

Domingos, P., Using Partitioning to Speed Up Specific-to-General Rule Induction. In Proceedings of the AAAI-96 Workshop on Integrating Multiple Learned Models, pp. 29-34, AAAI Press, 1996.

Freund Y. and Schapire R. E., Experiments with a new boosting algorithm. In Machine Learning: Proceedings of the Thirteenth International Conference, pages 325-332, 1996.

Fürnkranz, J., More efficient windowing, In Proceeding of The 14th national Conference on Artificial Intelegence (AAAI-97), pp. 509-514, Providence, RI. AAAI Press, 1997.

Gams, M., New Measurements Highlight the Importance of Redundant Knowledge. In European Working Session on Learning, Montpeiller, France, Pitman, 1989.

Geman S., Bienenstock, E., and Doursat, R., Neural networks and the bias variance dilemma. Neural Computation, 4:1-58, 1995.

Hansen J., Combining Predictors. Meta Machine Learning Methods and Bias Variance & Ambiguity Decompositions. PhD dissertation. Aurhus University. 2000.

Hansen, L. K., and Salamon, P., Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10), 993–1001, 1990.

Hu, X., Using Rough Sets Theory and Database Operations to Construct a Good Ensemble of Classifiers for Data Mining Applications. ICDM01. pp. 233-240, 2001.

Hu X., Yoo I., Cluster ensemble and its applications in gene expression analysis, Proceedings of the second conference on Asia-Pacific bioinformatics, pp. 297–302, Dunedin, New Zealand, 2004.

Kolen, J. F., and Pollack, J. B., Back propagation is sesitive to initial conditions. In Advances in Neural Information Processing Systems, Vol. 3, pp. 860-867 San Francisco, CA. Morgan Kaufmann, 1991.

Krogh, A., and Vedelsby, J., Neural network ensembles, cross validation and active learning. In Advances in Neural Information Processing Systems 7, pp. 231-238 1995.

Kuncheva, L., & Whitaker, C., Measures of diversity in classifier ensembles and their relationship with ensemble accuracy. Machine Learning, pp. 181–207, 2003.

Leigh W., Purvis R., Ragusa J. M., Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural networks, and genetic algorithm: a case study in romantic decision support, Decision Support Systems 32(4): 361–377, 2002.

Lewis D., and Catlett J., Heterogeneous uncertainty sampling for supervised learning. In Machine Learning: Proceedings of the Eleventh Annual Conference, pp. 148-156 , New Brunswick, New Jersey, Morgan Kaufmann, 1994.

Lewis, D., and Gale, W., Training text classifiers by uncertainty sampling, In seventeenth annual international ACM SIGIR conference on research and development in information retrieval, pp. 3-12, 1994.

Liu H., Mandvikar A., Mody J., An Empirical Study of Building Compact Ensembles. WAIM 2004: pp. 622-627.

Maimon O. Rokach L., Ensemble of Decision Trees for Mining Manufacturing Data Sets, Machine Engineering, vol. 4 No1-2, 2004.

Mangiameli P., West D., Rampal R., Model selection for medical diagnosis decision support systems, Decision Support Systems, 36(3): 247–259, 2004.

Margineantu D. and Dietterich T., Pruning adaptive boosting. In Proc. Fourteenth Intl. Conf. Machine Learning, pages 211–218, 1997.

Mitchell, T., Machine Learning, McGraw-Hill, 1997.

Neal R., Probabilistic inference using Markov Chain Monte Carlo methods. Tech. Rep. CRG-TR-93-1, Department of Computer Science, University of Toronto, Toronto, CA, 1993.

Opitz, D. and Maclin, R., Popular Ensemble Methods: An Empirical Study, Journal of Artificial Research, 11: 169-198, 1999.

Parmanto, B., Munro, P. W., and Doyle, H. R., Improving committee diagnosis with resampling techinques. In Touretzky, D. S., Mozer, M. C., and Hesselmo, M. E. (Eds). Advances in Neural Information Processing Systems, Vol. 8, pp. 882-888 Cambridge, MA. MIT Press, 1996.

Prodromidis, A. L., Stolfo, S. J. and Chan, P. K., Effective and efficient pruning of metaclassifiers in a distributed Data Mining system. Technical report CUCS-017-99, Columbia Univ., 1999.

Provost, F.J. and Kolluri, V., A Survey of Methods for Scaling Up Inductive Learning Algorithms, Proc. 3rd International Conference on Knowledge Discovery and Data Mining, 1997.

Quinlan, J. R., C4.5: Programs for Machine Learning, Morgan Kaufmann, Los Altos, 1993.

Quinlan, J. R., Bagging, Boosting, and C4.5. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pages 725-730, 1996.

Schaffer, C., Selecting a classification method by cross-validation. Machine Learning 13(1):135-143, 1993.

Seewald, A.K. and Fürnkranz, J., Grading classifiers, Austrian research institute for Artificial intelligence, 2001.

Sharkey, A., On combining artificial neural nets, Connection Science, Vol. 8, pp.299-313, 1996.

Shilen, S., Multiple binary tree classifiers. Pattern Recognition 23(7): 757-763, 1990.

Shilen, S., Nonparametric classification using matched binary decision trees. Pattern Recognition Letters 13: 83-87, 1992.

Sohn S. Y., Choi, H., Ensemble based on Data Envelopment Analysis, ECML Meta Learning workshop, Sep. 4, 2001.

Strehl A., Ghosh J. (2003), Cluster ensembles - a knowledge reuse framework for combining multiple partitions, The Journal of Machine Learning Research, 3: 583-617, 2003.

Tan A. C., Gilbert D., Deville Y., Multi-class Protein Fold Classification using a New Ensemble Machine Learning Approach. Genome Informatics, 14:206–217, 2003.

Tukey J.W., Exploratory data analysis, Addison-Wesley, Reading, Mass, 1977.

Tumer, K. and Ghosh J., Error Correlation and Error Reduction in Ensemble Classifiers, Connection Science, Special issue on combining artificial neural networks: ensemble approaches, 8 (3-4): 385-404, 1996.

Tumer, K., and Ghosh J., Linear and Order Statistics Combiners for Pattern Classification, in Combining Articial Neural Nets, A. Sharkey (Ed.), pp. 127-162, Springer-Verlag, 1999.

Tumer, K., and Ghosh J., Robust Order Statistics based Ensembles for Distributed Data Mining. In Kargupta, H. and Chan P., eds, Advances in Distributed and Parallel Knowledge Discovery , pp. 185-210, AAAI/MIT Press, 2000.

Wolpert, D.H., Stacked Generalization, Neural Networks, Vol. 5, pp. 241-259, Pergamon Press, 1992.

Zenobi, G., and Cunningham, P. Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error. In Proceedings of the European Conference on Machine Learning, 2001.

Zhou, Z. H., and Tang, W., Selective Ensemble of Decision Trees, in Guoyin Wang, Qing Liu, Yiyu Yao, Andrzej Skowron (Eds.): Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, 9^{th} International Conference, RSFDGrC, Chongqing, China, Proceedings. Lecture Notes in Computer Science 2639, pp.476-483, 2003.

Zhou, Z. H., Wu J., Tang W., Ensembling neural networks: many could be better than all. Artificial Intelligence 137: 239-263, 2002.