

Classification and Regression Trees, Bagging, and Boosting

Clifton D. Sutton

1. Introduction

Tree-structured classification and regression are alternative approaches to classification and regression that are not based on assumptions of normality and user-specified model statements, as are some older methods such as discriminant analysis and ordinary least squares (OLS) regression. Yet, unlike the case for some other nonparametric methods for classification and regression, such as kernel-based methods and nearest neighbors methods, the resulting tree-structured predictors can be relatively simple functions of the input variables which are easy to use.

Bagging and boosting are general techniques for improving prediction rules. Both are examples of what [Breiman \(1998\)](#) refers to as *perturb and combine* (P&C) methods, for which a classification or regression method is applied to various perturbations of the original data set, and the results are combined to obtain a single classifier or regression model. Bagging and boosting can be applied to tree-based methods to increase the accuracy of the resulting predictions, although it should be emphasized that they can be used with methods other than tree-based methods, such as neural networks.

This chapter will cover tree-based classification and regression, as well as bagging and boosting. This introductory section provides some general information, and briefly describes the origin and development of each of the methods. Subsequent sections describe how the methods work, and provide further details.

1.1. Classification and regression trees

Tree-structured classification and regression are nonparametric computationally intensive methods that have greatly increased in popularity during the past dozen years. They can be applied to data sets having both a large number of cases and a large number of variables, and they are extremely resistant to outliers (see [Steinberg and Colla, 1995, p. 24](#)).

Classification and regression trees can be good choices for analysts who want fairly accurate results quickly, but may not have the time and skill required to obtain

1 them using traditional methods. If more conventional methods are called for, trees 1
2 can still be helpful if there are a lot of variables, as they can be used to identify 2
3 important variables and interactions. Classification and regression trees have become 3
4 widely used among members of the data mining community, but they can also be 4
5 used for relatively simple tasks, such as the imputation of missing values (see [Harrell,](#) 5
6 [2001](#)). 6

7 Regression trees originated in the 1960s with the development of AID (Automatic 7
8 Interaction Detection) by [Morgan and Sonquist \(1963\)](#). Then in the 1970s, [Morgan and](#) 8
9 [Messenger \(1973\)](#) created THAID (Theta AID) to produce classification trees. AID 9
10 and THAID were developed at the Institute for Social Research at the University of 10
11 Michigan. 11

12 In the 1980s, statisticians [Breiman et al. \(1984\)](#) developed CART (Classification 12
13 And Regression Trees), which is a sophisticated program for fitting trees to data. Since 13
14 the original version, CART has been improved and given new features, and it is now 14
15 produced, sold, and documented by Salford Systems. Statisticians have also developed 15
16 other tree-based methods; for example, the QUEST (Quick Unbiased Efficient Statistical 16
17 Tree) method of [Loh and Shih \(1997\)](#). 17

18 Classification and regression trees can now be produced using many different soft- 18
19 ware packages, some of which are relatively expensive and are marketed as being 19
20 commercial data mining tools. Some software, such as S-Plus, use algorithms that are 20
21 very similar to those underlying the CART program. In addition to creating trees using 21
22 `tree` (see [Clark and Pregibon, 1992](#)), users of S-Plus can also use `rpart` (see 22
23 [Therneau and Atkinson, 1997](#)), which can be used for Poisson regression and survival 23
24 analysis in addition to being used for the creation of ordinary classification and regres- 24
25 sion trees. [Martinez and Martinez \(2002\)](#) provide MATLAB code for creating trees, 25
26 which are similar to those that can be created by CART, although their routines do not 26
27 handle nominal predictors having three or more categories or splits involving more than 27
28 one variable. The MATLAB Statistics Toolbox also has functions for creating trees. 28
29 Other software, including some ensembles aimed at data miners, create trees using vari- 29
30 ants of CHAID (CHI-squared Automatic Interaction Detector), which is a descendant of 30
31 THAID developed by [Kass \(1980\)](#). SPSS sells products that allow users to create trees 31
32 using more than one major method. 32

33 The machine learning community has produced a large number of programs to create 33
34 decision trees for classification and, to a lesser extent, regression. Very notable among 34
35 these is Quinlan's extremely popular C4.5 (see [Quinlan, 1993](#)), which is a descendant 35
36 of his earlier program, ID3 (see [Quinlan, 1979](#)). A newer product of Quinlan is his 36
37 system See5/C5.0. [Quinlan \(1986\)](#) provides further information about the development 37
38 of tree-structured classifiers by the machine learning community. 38

39 Because there are so many methods available for tree-structured classification and 39
40 regression, this chapter will focus on one of them, CART, and only briefly indicate how 40
41 some of the others differ from CART. For a fuller comparison of tree-structured clas- 41
42 sifiers, the reader is referred to [Ripley \(1996, Chapter 7\)](#). [Gentle \(2002\)](#) gives a shorter 42
43 overview of classification and regression trees, and includes some more recent refer- 43
44 ences. It can also be noted that the internet is a source for a large number of descrip- 44
45 tions of various methods for tree-structured classification and regression. 45

1 1.2. Bagging and boosting

2 Bootstrap aggregation, or bagging, is a technique proposed by Breiman (1996a) that
3 can be used with many classification methods and regression methods to reduce the
4 variance associated with prediction, and thereby improve the prediction process. It is
5 a relatively simple idea: many bootstrap samples are drawn from the available data,
6 some prediction method is applied to each bootstrap sample, and then the results are
7 combined, by averaging for regression and simple voting for classification, to obtain the
8 overall prediction, with the variance being reduced due to the averaging.
9

10 Boosting, like bagging, is a committee-based approach that can be used to improve
11 the accuracy of classification or regression methods. Unlike bagging, which uses a simple
12 averaging of results to obtain an overall prediction, boosting uses a weighted average
13 of results obtained from applying a prediction method to various samples. Also, with
14 boosting, the samples used at each step are not all drawn in the same way from the
15 same population, but rather the incorrectly predicted cases from a given step are given
16 increased weight during the next step. Thus boosting is an iterative procedure, incorporating
17 weights, as opposed to being based on a simple averaging of predictions, as is the
18 case with bagging. In addition, boosting is often applied to *weak learners* (e.g., a simple
19 classifier such as a two node decision tree), whereas this is not the case with bagging.
20

21 Schapire (1990) developed the predecessor to later boosting algorithms developed by
22 him and others. His original method pertained to two-class classifiers, and combined the
23 results of three classifiers, created from different learning samples, by simple majority
24 voting. Freund (1995) extended Schapire's original method by combining the results
25 of a larger number of weak learners. Then Freund and Schapire (1996) developed the
26 *AdaBoost algorithm*, which quickly became very popular. Breiman (1998) generalized
27 the overall strategy of boosting, and considered Freund and Schapire's algorithm as a
28 special case of the class of *arcing* algorithms, with the term *arcing* being suggested
29 by *adaptive resampling and combining*. But in the interest of brevity, and due to the
30 popularity of Freund and Schapire's algorithm, this chapter will focus on AdaBoost
31 and only briefly refer to related approaches. (See the bibliographic notes at the end of
32 Chapter 10 of Hastie et al. (2001) for additional information about the development of
33 boosting, and for a simple description of Schapire's original method.)

34 Some (see, for example, Hastie et al., 2001, p. 299) have indicated that boosting is
35 one of the most powerful machine/statistical learning ideas to have been introduced during
36 the 1990s, and it has been suggested (see, for example, Breiman (1998) or Breiman's
37 statement in Olshen (2001, p. 194)) that the application of boosting to classification trees
38 results in classifiers which generally are competitive with any other classifier. A particularly
39 nice thing about boosting and bagging is that they can be used very successfully with
40 simple "off-the-shelf" classifiers (as opposed to needing to carefully tune and tweak the
41 classifiers). This fact serves to somewhat offset the criticism that with both bagging
42 and boosting the improved performance comes at the cost of increased computation time.
43 It can also be noted that Breiman (1998) indicates that applying boosting to CART to
44 create a classifier can actually be much quicker than fitting a neural net classifier.
45 However, one potential drawback to perturb and combine methods is that the final prediction
46 rule can be appreciably more complex than what can be obtained using

1 a method that does not combine predictors. Of course, it is often the case that simplicity 1
2 has to be sacrificed to obtain increased accuracy. 2

3 2. Using CART to create a classification tree 3

4
5
6
7 In the general classification problem, it is known that each case in a sample belongs 7
8 to one of a finite number of possible classes, and given a set of measurements for a 8
9 case, it is desired to correctly predict to which class the case belongs. A classifier is a 9
10 rule that assigns a predicted class membership based on a set of related measurements, 10
11 x_1, x_2, \dots, x_{K-1} , and x_K . Taking the measurement space \mathcal{X} to be the set of all possible 11
12 values of (x_1, \dots, x_K) , and letting $\mathcal{C} = \{c_1, c_2, \dots, c_J\}$ be the set of possible classes, 12
13 a classifier is just a function with domain \mathcal{X} and range \mathcal{C} , and it corresponds to a partition 13
14 of \mathcal{X} into disjoint sets, B_1, B_2, \dots, B_J , such that the predicted class is j if $\mathbf{x} \in B_j$, 14
15 where $\mathbf{x} = (x_1, \dots, x_K)$. 15

16 It is normally desirable to use past experience as a basis for making new predictions, 16
17 and so classifiers are usually constructed from a *learning sample* consisting of cases 17
18 for which the correct class membership is known in addition to the associated values of 18
19 (x_1, \dots, x_K) . Thus statistical classification is similar to regression, only the response 19
20 variable is nominal. Various methods for classification differ in how they use the data 20
21 (the learning sample) to partition \mathcal{X} into the sets B_1, B_2, \dots, B_J . 21
22

23 2.1. Classification trees 23

24
25 Tree-structured classifiers are constructed by making repetitive splits of \mathcal{X} and the 25
26 subsequently created subsets of \mathcal{X} , so that a hierarchical structure is formed. For ex- 26
27 ample, \mathcal{X} could first be divided into $\{\mathbf{x} \mid x_3 \leq 53.5\}$ and $\{\mathbf{x} \mid x_3 > 53.5\}$. Then the 27
28 first of these sets could be further divided into $A_1 = \{\mathbf{x} \mid x_3 \leq 53.5, x_1 \leq 29.5\}$ 28
29 and $A_2 = \{\mathbf{x} \mid x_3 \leq 53.5, x_1 > 29.5\}$, and the other set could be split into 29
30 $A_3 = \{\mathbf{x} \mid x_3 > 53.5, x_1 \leq 74.5\}$ and $A_4 = \{\mathbf{x} \mid x_3 > 53.5, x_1 > 74.5\}$. If there 30
31 are just two classes ($J = 2$), it could be that cases having unknown class for which \mathbf{x} 31
32 belongs to A_1 or A_3 should be classified as c_1 (predicted to be of the class c_1), and cases 32
33 for which \mathbf{x} belongs to A_2 or A_4 should be classified as c_2 . (Making use of the notation 33
34 established above, we would have $B_1 = A_1 \cup A_3$ and $B_2 = A_2 \cup A_4$.) Figure 1 shows 34
35 the partitioning of \mathcal{X} and Figure 2 shows the corresponding representation as a tree. 35

36 While it can be hard to draw the partitioning of \mathcal{X} when more than two predictor 36
37 variables are used, one can easily create a tree representation of the classifier, which 37
38 is easy to use no matter how many variables are used and how many sets make up 38
39 the partition. Although it is generally harder to understand how the predictor variables 39
40 relate to the various classes when the tree-structured classifier is rather complicated, the 40
41 classifier can easily be used, without needing a computer, to classify a new observation 41
42 based on the input values, whereas this is not usually the case for nearest neighbors 42
43 classifiers and kernel-based classifiers. 43

44 It should be noted that when \mathcal{X} is divided into two subsets, these subsets do not 44
45 both have to be subsequently divided using the same variable. For example, one subset 45

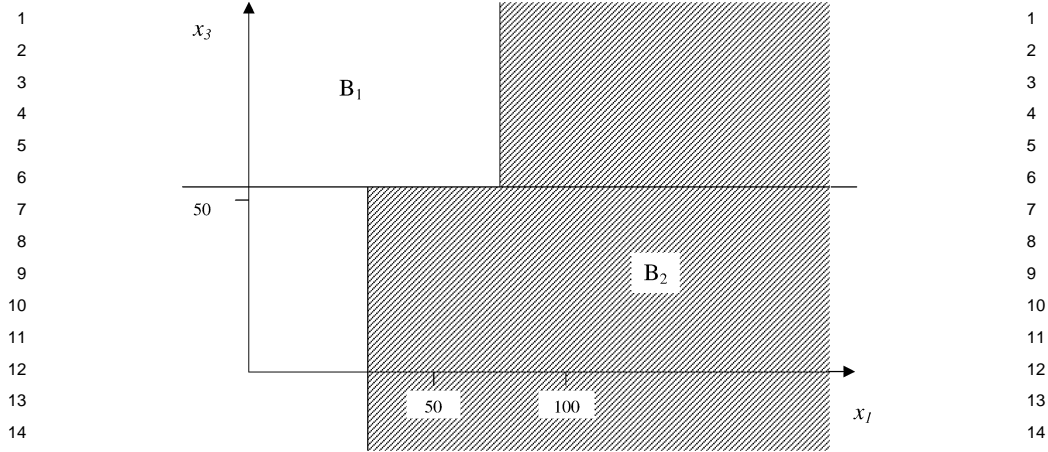


Fig. 1. A partition of \mathcal{X} formed by orthogonal splits.

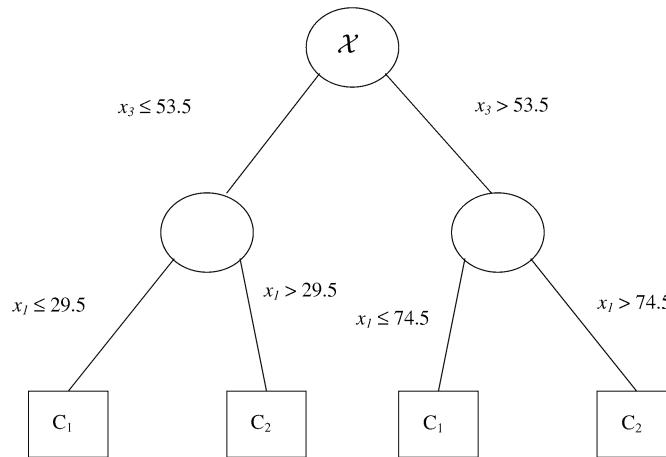


Fig. 2. Binary tree representation of the classifier corresponding to the partition shown in Figure 1.

could be split using x_1 and the other subset could be split using x_4 . This allows us to model a nonhomogeneous response: the classification rules for different regions of \mathcal{X} can use different variables, or be different functions of the same variables. Furthermore, a classification tree does not have to be symmetric in its pattern of nodes. That is, when \mathcal{X} is split, it may be that one of the subsets produced is not split, while the other subset is split, with each of the new subsets produced being further divided.

2.2. Overview of how CART creates a tree

What needs to be determined is how to best split subsets of \mathcal{X} (starting with \mathcal{X} itself), to produce a tree-structured classifier. CART uses recursive binary partitioning to create

1 a binary tree. (Some other methods for creating classification trees allow for more than 1
2 two branches to descend from a node; that is, a subset of \mathcal{X} can be split into more than 2
3 two subsets in a single step. It should be noted that CART is not being restrictive in only 3
4 allowing binary trees, since the partition of \mathcal{X} created by any other type of tree structure 4
5 can also be created using a binary tree.) So, with CART, the issues are: 5

- 6 (1) how to make each split (identifying which variable or variables should be used to 6
7 create the split, and determining the precise rule for the split), 7
- 8 (2) how to determine when a node of the tree is a terminal node (corresponding to a 8
9 subset of \mathcal{X} which is not further split), and 9
- 10 (3) how to assign a predicted class to each terminal node. 10

11
12 The assignment of predicted classes to the terminal nodes is relatively simple, as is 12
13 determining how to make the splits, whereas determining the right-sized tree is not so 13
14 straightforward. In order to explain some of these details, it seems best to start with 14
15 the easy parts first, and then proceed to the tougher issue; and in fact, one needs to first 15
16 understand how the simpler parts are done since the right-sized tree is selected from a set 16
17 of candidate tree-structured classifiers, and to obtain the set of candidates, the splitting 17
18 and class assignment issues have to be handled. 18

19 20 2.3. Determining the predicted class for a terminal node 20

21
22 If the learning sample can be viewed as being a random sample from the same pop- 22
23 ulation or distribution from which future cases to be classified will come, and if all 23
24 types of misclassifications are considered to be equally bad (for example, in the case 24
25 of $J = 2$, classifying a c_1 case as a c_2 is given the same penalty as classifying a c_2 25
26 as a c_1), the class assignment rule for the terminal nodes is the simple *plurality rule*: 26
27 the class assigned to a terminal node is the class having the largest number of members 27
28 of the learning sample corresponding to the node. (If two or more classes are tied for 28
29 having the largest number of cases in the learning sample with \mathbf{x} belonging to the set 29
30 corresponding to the node, the predicted class can be arbitrarily selected from among 30
31 these classes.) If the learning sample can be viewed as being a random sample from the 31
32 same population or distribution from which future cases to be classified will come, but 32
33 all types of misclassifications are not considered to be equally bad (that is, classifying a 33
34 c_1 case as a c_2 may be considered to be twice as bad, or ten times as bad, as classifying 34
35 a c_2 as a c_1), the different misclassification penalties, which are referred to as *misclas-* 35
36 *sification costs*, should be taken into account, and the assigned class for a terminal node 36
37 should be the one which minimizes the total misclassification cost for all cases of the 37
38 learning sample corresponding to the node. (Note that this rule reduces to the plural- 38
39 ity rule if all types of misclassification have the same cost.) Likewise, if all types of 39
40 misclassifications are to be penalized the same, but the learning sample has class mem- 40
41 bership proportions that are different from those that are expected for future cases to be 41
42 classified with the classifier, weights should be used when assigning predicted classes 42
43 to the nodes, so that the predictions will hopefully minimize the number of misclassifi- 43
44 cations for future cases. Finally, if all types of misclassifications are not to be penalized 44
45 the same *and* the learning sample has class membership proportions that are different 45

1 from those that are expected for cases to be classified with the classifier, two sets of
2 weights are used to assign the predicted classes. One set of weights is used to account
3 for different misclassification costs, and the other set of weights is used to adjust for
4 the class proportions of future observations being different from the class proportions
5 of the learning sample.

6 It should be noted that, in some cases, weights are also used in other aspects of
7 creating a tree-structured classifier, and not just for the assignment of the predicted
8 class. But in what follows, the simple case of the learning sample being viewed as a
9 random sample from the same population or distribution from which future cases to be
10 classified will come, and all types of misclassifications being considered equally bad,
11 will be dealt with. In such a setting, weights are not needed. To learn what adjustments
12 should be made for other cases, the interested reader can find some pertinent information
13 in Breiman et al. (1984), Hastie et al. (2001), Ripley (1996).

14 2.4. Selection of splits to create a partition 15

16 In determining how to divide subsets of \mathcal{X} to create two children nodes from a parent
17 node, the general goal is to make the distributions of the class memberships of the cases
18 of the learning sample corresponding to the two descendant nodes different, in such a
19 way as to make, with respect to the response variable, the data corresponding to each of
20 the children nodes purer than the data corresponding to the parent node. For example,
21 in a four class setting, a good first split may nearly separate the c_1 and c_3 cases from the
22 c_2 and c_4 cases. In such a case, uncertainty with regard to class membership of cases
23 having \mathbf{x} belonging to a specific subset of \mathcal{X} is reduced (and further splits may serve to
24 better divide the classes).

25 There are several different types of splits that can be considered at each step. For a
26 predictor variable, x_k , which is numerical or ordinal (coded using successive integers),
27 a subset of \mathcal{X} can be divided with a plane orthogonal to the x_k axis, such that one of the
28 newly created subsets has $x_k \leq s_k$, and the other has $x_k > s_k$. Letting

$$30 \quad y_{k(1)} < y_{k(2)} < \dots < y_{k(M)} \quad 31$$

32 be the ordered distinct values of x_k observed in the portion of the learning sample be-
33 longing to the subset of \mathcal{X} to be divided, the values

$$34 \quad (y_{k(m)} + y_{k(m+1)})/2 \quad (m = 1, 2, \dots, M - 1) \quad 35$$

36 can be considered for the split value s_k . So even if there are many different continuous or
37 ordinal predictors, there is only a finite (albeit perhaps rather large) number of possible
38 splits of this form to consider. For a predictor variable, x_k , which is nominal, having
39 class labels belonging to the finite set D_k , a subset of \mathcal{X} can be divided such that one
40 of the newly created subsets has $x_k \in S_k$, and the other has $x_k \notin S_k$, where S_k is a
41 nonempty proper subset of D_k . If D_k contains d members, there are $2^{d-1} - 1$ splits of
42 this form to be considered.

43 Splits involving more than one variable can also be considered. Two or more continu-
44 ous or ordinal variables can be involved in a *linear combination split*, with which a sub-
45 set of \mathcal{X} is divided with a hyperplane which is not perpendicular to one of the axes. For

1 example, one of the created subsets can have points for which $2.7x_3 - 11.9x_7 \leq 54.8$, 1
2 with the other created subset having points for which $2.7x_3 - 11.9x_7 > 54.8$. Simi- 2
3 larly, two or more nominal variables can be involved in a *Boolean split*. For example, 3
4 two nominal variables, gender and race, can be used to create a split with cases corre- 4
5 sponding to white males belonging to one subset, and cases corresponding to males who 5
6 are not white and females belonging to the other subset. (It should be noted that while 6
7 [Breiman et al. \(1984\)](#) describe such Boolean splits, they do not appear to be included in 7
8 the Salford Systems implementation.) 8

9 In many instances, one may not wish to allow linear combination and Boolean splits, 9
10 since they can make the resulting tree-structured classifier more difficult to interpret, 10
11 and can increase the computing time since the number of candidate splits is greatly 11
12 increased. Also, if just single variable splits are used, the resulting tree is invariant 12
13 with respect to monotone transformations of the variables, and so one does not have to 13
14 consider whether say dose, or the log of dose, should be used as a predictor. But if linear 14
15 combination splits are allowed, transforming the variables can make a difference in the 15
16 resulting tree, and for the sake of simplicity, one might want to only consider single 16
17 variable splits. However, it should be noted that sometimes a single linear combination 17
18 split can be better than many single variable splits, and it may be preferable to have a 18
19 classifier with fewer terminal nodes, having somewhat complicated boundaries for the 19
20 sets comprising the partition, than to have one created using only single variable splits 20
21 and having a large number of terminal nodes. 21

22 At each stage in the recursive partitioning, all of the allowable ways of splitting a 22
23 subset of \mathcal{X} are considered, and the one which leads to the greatest increase in node 23
24 purity is chosen. This can be accomplished using an *impurity function*, which is a func- 24
25 tion of the proportions of the learning sample belonging to the possible classes of the 25
26 response variable. These proportions will be denoted by $p_1, p_2, \dots, p_{J-1}, p_J$. 26

27 The impurity function should be such that it is maximized whenever a subset of \mathcal{X} 27
28 corresponding to a node in the tree contains an equal number of each of the possible 28
29 classes. (If there are the same number of c_1 cases as there are c_2 cases and c_3 cases 29
30 and so on, then we are not able to sensibly associate that node with a particular class, 30
31 and in fact, we are not able to sensibly favor any class over any other class, giving 31
32 us that uncertainty is maximized.) The impurity function should assume its minimum 32
33 value for a node that is completely pure, having all cases from the learning sample 33
34 corresponding to the node belonging to the same class. Two such functions that can 34
35 serve as the impurity function are the *Gini index of diversity*, 35

$$36 \quad g(p_1, \dots, p_J) = 2 \sum_{j=1}^{J-1} \sum_{j'=j+1}^J p_j p_{j'} = 1 - \sum_{j=1}^J p_j^2, \quad 36$$

37 and the *entropy function*, 37

$$38 \quad h(p_1, \dots, p_J) = - \sum_{j=1}^J p_j \log p_j, \quad 38$$

39 provided that $0 \log 0$ is taken to be $\lim_{p \downarrow 0} p \log p = 0$. 39

1 To assess the quality of a potential split, one can compute the value of the impurity 1
2 function using the cases in the learning sample corresponding to the parent node (the 2
3 node to be split), and subtract from this the weighted average of the impurity for the 3
4 two children nodes, with the weights proportional to the number of cases of the learning 4
5 sample corresponding to each of the two children nodes, to get the decrease in overall 5
6 impurity that would result from the split. To select the way to split a subset of \mathcal{X} in 6
7 the tree growing process, all allowable ways of splitting can be considered, and the one 7
8 which will result in the greatest decrease in node impurity (greatest increase in node 8
9 purity) can be chosen. 9
10

11 2.5. Estimating the misclassification rate and selecting the right-sized tree 11

12
13 The trickiest part of creating a good tree-structured classifier is determining how com- 13
14 plex the tree should be. If nodes continue to be created until no two distinct values of \mathbf{x} 14
15 for the cases in the learning sample belong to the same node, the tree may be overfitting 15
16 the learning sample and not be a good classifier of future cases. On the other hand, if 16
17 a tree has only a few terminal nodes, then it may be that it is not making enough use 17
18 of information in the learning sample, and classification accuracy for future cases will 18
19 suffer. Initially, in the tree-growing process, the predictive accuracy typically improves 19
20 as more nodes are created and the partition gets finer. But it is usually the case that 20
21 at some point the misclassification rate for future cases will start to get worse as the tree 21
22 becomes more complex. 22

23 In order to compare the prediction accuracy of various tree-structured classifiers, 23
24 there needs to be a way to estimate a given tree's misclassification rate for future ob- 24
25 servations, which is sometimes referred to as the *generalization error*. What does not 25
26 work well is to use the *resubstitution estimate* of the misclassification rate (also known 26
27 as the *training error*), which is obtained by using the tree to classify the members of 27
28 the learning sample (that were used to create the tree), and observing the proportion 28
29 that are misclassified. If no two members of the learning sample have the same value 29
30 of \mathbf{x} , then a tree having a resubstitution misclassification rate of zero can be obtained by 30
31 continuing to make splits until each case in the learning sample is by itself in a terminal 31
32 node (since the class associated with a terminal node will be that of the learning sample 32
33 case corresponding to the node, and when the learning sample is then classified using 33
34 the tree, each case in the learning sample will drop down to the terminal node that it 34
35 created in the tree-growing process, and will have its class match the predicted class 35
36 for the node). Thus the resubstitution estimate can be a very poor estimate of the tree's 36
37 misclassification rate for future observations, since it can decrease as more nodes are 37
38 created, even if the selection of splits is just responding to "noise" in the data, and not 38
39 real structure. This phenomenon is similar to R^2 increasing as more terms are added to 39
40 a multiple regression model, with the possibility of R^2 equaling one if enough terms 40
41 are added, even though more complex regression models can be much worse predictors 41
42 than simpler ones involving fewer variables and terms. 42

43 A better estimate of a tree's misclassification rate can be obtained using an independ- 43
44 ent *test sample*, which is a collection of cases coming from the same population or 44
45 distribution as the learning sample. Like the learning sample, for the test sample the 45

1 true class of each case is known in addition to the values for the predictor variables. 1
2 The *test sample estimate* of the misclassification rate is just the proportion of the cases 2
3 in the test sample that are misclassified when predicted classes are obtained using the 3
4 tree created from the learning sample. If the cases to be classified in the future will also 4
5 come from the same distribution that produced the test sample cases, the test sample 5
6 estimation procedure is unbiased. 6
7 Since the test sample and the learning sample are both composed of cases for which 7
8 the true class is known in addition to the values for the predictor variables, choosing to 8
9 make the test sample larger will result in a smaller learning sample. Often it is thought 9
10 that about one third of the available cases should be set aside to serve as a test sample, 10
11 and the rest of the cases should be used as the learning sample. But sometimes a smaller 11
12 fraction, such as one tenth, is used instead. 12
13 If one has enough suitable data, using an independent test sample is the best thing 13
14 to do. Otherwise, obtaining a *cross-validation estimate* of the misclassification rate is 14
15 preferable. For a V -fold cross-validation, one uses all of the available data for the learn- 15
16 ing sample, and divides these cases into V parts of approximately the same size. This 16
17 is usually done randomly, but one may use stratification to help make the V cross- 17
18 validation groups more similar to one another. V is typically taken to be 5 or 10. In a 18
19 lot of cases, little is to be gained by using a larger value for V , and the larger V is, the 19
20 greater is the amount of time required to create the classifier. In some situations, the 20
21 quality of the estimate is reduced by making V too large. 21
22 To obtain a cross-validation estimate of the misclassification rate, each of the V 22
23 groups is in turn set aside to serve temporarily as an independent test sample and a tree 23
24 is grown, according to certain criteria, using the other $V - 1$ groups. In all, V trees 24
25 are grown in this way, and for each tree the set aside portion of the data is used to 25
26 obtain a test sample estimate of the tree's misclassification rate. Then the V test sample 26
27 estimates are averaged to obtain the estimate of the misclassification rate for the tree 27
28 grown from the entire learning sample using the same criteria. (If $V = 10$, the hope is 28
29 that the average of the misclassification rates of the trees created using 90% of the data 29
30 will not be too different from the misclassification rate of the tree created using all of the 30
31 data.) Further details pertaining to how cross-validation is used to select the right-sized 31
32 tree and estimate its misclassification rate are given below. 32
33 Whether one is going to use cross-validation or an independent test sample to esti- 33
34 mate misclassification rates, it still needs to be specified how to grow the best tree, or 34
35 how to create a set of candidate trees from which the best one can be selected based 35
36 on their estimated misclassification rates. It does not work very well to use some sort 36
37 of a *stop splitting rule* to determine that a node should be declared a terminal node and 37
38 the corresponding subset of \mathcal{X} not split any further, because it can be the case that the 38
39 best split possible at a certain stage may decrease impurity by only a small amount, but 39
40 if that split is made, each of the subsets of \mathcal{X} corresponding to both of the descendant 40
41 nodes can be split in ways to produce an appreciable decrease in impurity. Because of 41
42 this phenomenon, what works better is to first grow a very large tree, splitting subsets in 42
43 the current partition of \mathcal{X} even if a split does not lead to an appreciable decrease in im- 43
44 purity. For example, splits can be made until 5 or fewer members of the learning sample 44
45 correspond to each terminal node (or all of the cases corresponding to a node belong 45

1 to the same class). Then a sequence of smaller trees can be created by *pruning* the ini- 1
2 tial large tree, where in the pruning process, splits that were made are removed and a 2
3 tree having a fewer number of nodes is produced. The accuracies of the members of 3
4 this sequence of subtrees—really a finite sequence of nested subtrees, since the first tree 4
5 produced by pruning is a subtree of the original tree, and a second pruning step creates a 5
6 subtree of the first subtree, and so on—are then compared using good estimates of their 6
7 misclassification rates (either based on a test sample or obtained by cross-validation), 7
8 and the best performing tree in the sequence is chosen as the classifier. 8

9 A specific way to create a useful sequence of different-sized trees is to use *minimum* 9
10 *cost-complexity pruning*. In this process, a nested sequence of subtrees of the initial 10
11 large tree is created by *weakest-link cutting*. With weakest-link cutting (pruning), all 11
12 of the nodes that arise from a specific nonterminal node are pruned off (leaving that 12
13 specific node as a terminal node), and the specific node selected is the one for which the 13
14 corresponding pruned nodes provide the smallest *per node* decrease in the resubstitution 14
15 misclassification rate. If two or more choices for a cut in the pruning process would 15
16 produce the same per node decrease in the resubstitution misclassification rate, then 16
17 pruning off the largest number of nodes is favored. In some cases (minimal pruning 17
18 cases), just two children terminal nodes are pruned from a parent node, making it a 18
19 terminal node. But in other cases, a larger group of descendant nodes are pruned all 19
20 at once from an internal node of the tree. For example, at a given stage in the pruning 20
21 process, if the increase in the estimated misclassification rate caused by pruning four 21
22 nodes is no more than twice the increase in the estimated misclassification rate caused 22
23 by pruning two nodes, pruning four nodes will be favored over pruning two nodes. 23

24 Letting $R(T)$ be the resubstitution estimate of the misclassification rate of a tree, T , 24
25 and $|T|$ be the number of terminal nodes of the tree, for each $\alpha \geq 0$ the *cost-complexity* 25
26 *measure*, $R_\alpha(T)$, for a tree, T , is given by 26

$$27 \quad R_\alpha(T) = R(T) + \alpha|T|. \quad 27$$

28
29 Here, $|T|$ is a measure of tree complexity, and $R(T)$ is related to misclassification cost 29
30 (even though it is a biased estimate of the cost). α is the contribution to the measure 30
31 for each terminal node. To minimize this measure, for small values of α , trees having a 31
32 large number of nodes, and a low resubstitution estimate of misclassification rate, will 32
33 be favored. For large enough values of α , a one node tree (with \mathcal{X} not split at all, and 33
34 all cases to be classified given the same predicted class), will minimize the measure. 34

35 Since the resubstitution estimate of misclassification rate is generally overoptimistic 35
36 and becomes unrealistically low as more nodes are added to a tree, it is hoped that 36
37 there is some value of α that properly penalizes the overfitting of a tree which is too 37
38 complex, so that the tree which minimizes $R_\alpha(T)$, for the proper value of α , will be 38
39 a tree of about the right complexity (to minimize the misclassification rate of future 39
40 cases). Even though the proper value of α is unknown, utilization of the weakest-link 40
41 cutting procedure described above guarantees that for each value for α (greater than or 41
42 equal to 0), a subtree of the original tree that minimizes $R_\alpha(T)$ will be a member of the 42
43 finite nested sequence of subtrees produced. 43

44 The sequence of subtrees produced by the pruning serves as the set of candidates for 44
45 the classifier, and to obtain the classifier, all that remains to be done is to select the one 45

1 which will hopefully have the smallest misclassification rate for future predictions. The 1
2 selection is based on estimated misclassification rates, obtained using a test sample or 2
3 by cross-validation. 3

4 If an independent test sample is available, it is used to estimate the error rates of the 4
5 various trees in the nested sequence of subtrees, and the tree with the smallest estimated 5
6 misclassification rate can be selected to be used as the tree-structured classifier. A pop- 6
7 ular alternative is to recognize that since all of the error rates are not accurately known, 7
8 but only estimated, it could be that a simpler tree with only a slightly higher estimated 8
9 error rate is really just as good or better than the tree having the smallest estimated error 9
10 rate, and the least complex tree having an estimated error rate within one standard error 10
11 of the estimated error rate of the tree having the smallest estimated error rate can be 11
12 chosen, taking simplicity into account (and maybe not actually harming the prediction 12
13 accuracy). This is often referred to as the *1 SE rule*. 13

14 If cross-validation is being used instead of a test sample, then things are a bit more 14
15 complicated. First the entire collection of available cases are used to create a large tree, 15
16 which is then pruned to create a sequence of nested subtrees. (This sequence of trees, 16
17 from which the classifier will ultimately be selected, contains the subtree which mini- 17
18 mizes $R_\alpha(T)$ for every nonnegative α .) The first of the V portions of data is set aside to 18
19 serve as a test sample for a sequence of trees created from the other $V - 1$ portions of 19
20 data. These $V - 1$ portions of data are collectively used to grow a large tree, and then 20
21 the same pruning process is applied to create a sequence of subtrees. Each subtree in 21
22 the sequence is the optimal subtree in the sequence, according to the $R_\alpha(T)$ criterion, 22
23 for some range of values for α . Similar sequences of subtrees are created and the cor- 23
24 responding values of α for which each tree is optimal are determined, by setting aside, 24
25 one at a time, each of the other $V - 1$ portions of the data, resulting in V sequences 25
26 in all. Then, for various values of α , cross-validation estimates of the misclassification 26
27 rates of the corresponding trees created using all of the data are determined as follows: 27
28 for a given value of α the misclassification rate of the corresponding subtree in each of 28
29 the V sequences is estimated using the associated set aside portion as a test sample, and 29
30 the V estimates are averaged to arrive at a single error rate estimate corresponding to 30
31 that value of α , and this estimate serves as the estimate of the true misclassification rate 31
32 for the tree created using all of the data and pruned using this value of α . Finally, these 32
33 cross-validation estimates of the error rates for the trees in the original sequence of sub- 33
34 trees are compared, and the subtree having the smallest estimated misclassification rate 34
35 is selected to be the classifier. (Note that in this final stage of the tree selection process, 35
36 the role of α is to create, for each value of α considered, matchings of the subtrees from 36
37 the cross-validation sequences to the subtrees of the original sequence. The trees in each 37
38 matched set can be viewed as having been created using the same prescription: a large 38
39 tree is grown, weakest-link pruning is used to create a sequence of subtrees, and the 39
40 subtree which minimizes $R_\alpha(T)$ is selected.) 40
41

42 2.6. Alternative approaches 42

43 As indicated previously, several other programs that create classification trees are ap- 43
44 parently very similar to CART, but some other programs have key differences. Some 44
45

1 can handle only two classes for the response variable, and some only handle categori- 1
2 cal predictors. Programs can differ in how splits are selected, how missing values are 2
3 dealt with, and how the right-sized tree is determined. If pruning is used, the pruning 3
4 procedures can differ. 4

5 Some programs do not restrict splits to be binary splits. Some allow for *soft splits*, 5
6 which in a sense divide cases that are very close to a split point, having such cases 6
7 represented in both of the children nodes at a reduced weight. (See Ripley (1996) 7
8 for additional explanation and some references.) Some programs use resubstitution 8
9 estimates of misclassification rates to select splits, or use the result of a chi-squared 9
10 test, instead of using an impurity measure. (The chi-squared approach selects the split 10
11 that produces the most significant result when the null hypothesis of homogeneity is 11
12 tested against the general alternative using a chi-squared test on the two-way table of 12
13 counts which results from cross-tabulating the members of the learning sample in the 13
14 subset to be split by the class of their response and by which of the children node 14
15 created by the split they correspond to.) Some use the result of a chi-squared test 15
16 to determine that a subset of \mathcal{X} should not be further split, with the splitting ceas- 16
17 ing if the most significant split is not significant enough. That is, a stopping rule is 17
18 utilized to determine the complexity of the tree and control overfitting, as opposed 18
19 to first growing a complex tree, and then using pruning to create a set of candidate 19
20 trees, and estimated misclassification rates to select the classifier from the set of candi- 20
21 dates. 21

22 Ciampi et al. (1987), Quinlan (1987, 1993), and Gelfandi et al. (1991) consider alter- 22
23 native methods of pruning. Crawford (1989) examines using bootstrapping to estimate 23
24 the misclassification rates needed for the selection of the right-sized tree. Buntine (1992) 24
25 describes a Bayesian approach to tree construction. 25

26 In all, there are a large number of methods for the creation of classification trees, 26
27 and it is safe to state than none of them will work best in every situation. It may be 27
28 prudent to favor methods, and particular implementations of methods, which have been 28
29 thoroughly tested, and which allow the user to make some adjustments in order to tune 29
30 the method to yield improved predictions. 30
31
32

33 **3. Using CART to create a regression tree** 33 34

35 CART creates a regression tree from data having a numerical response variable in much 35
36 the same way as it creates a classification tree from data having a nominal response, but 36
37 there are some differences. Instead of a predicted class being assigned to each terminal 37
38 node, a numerical value is assigned to represent the predicted value for cases having 38
39 values of \mathbf{x} corresponding to the node. The sample mean or sample median of the re- 39
40 sponse values of the members of the learning sample corresponding to the node may 40
41 be used for this purpose. Also, in the tree-growing process, the split selected at each 41
42 stage is the one that leads to the greatest reduction in the sum of the squared differences 42
43 between the response values for the learning sample cases corresponding to a partic- 43
44 ular node and their sample mean, or the greatest reduction in the sum of the absolute 44
45 differences between the response values for the learning sample cases corresponding to 45

1 a particular node and their sample median. For example, using the squared differences 1
2 criterion, one seeks the plane which divides a subset of \mathcal{X} into the sets A and B for 2
3 which 3

$$\sum_{i: \mathbf{x}_i \in A} (y_i - \bar{y}_A)^2 + \sum_{i: \mathbf{x}_i \in B} (y_i - \bar{y}_B)^2$$

4 4
5 5
6 6
7 7
8 is minimized, where \bar{y}_A is the sample mean of the response values for cases in the learn- 8
9 ing sample corresponding to A , and \bar{y}_B is the sample mean of the response values for 9
10 cases in the learning sample corresponding to B . So split selection is based on making 10
11 the observed response values collectively close to their corresponding predicted values. 11
12 As with classification trees, for the sake of simplicity, one may wish to restrict consid- 12
13 eration to splits involving only one variable. 13

14 The sum of squared or absolute differences is also used to prune the trees. This 14
15 differs from the classification setting, where minimizing the impurity as measured by 15
16 the Gini index might be used to grow the tree, and the resubstitution estimate of the 16
17 misclassification rate is used to prune the tree. For regression trees, the error-complexity 17
18 measure used is 18

$$R_\alpha(T) = R(T) + \alpha|T|,$$

19 19
20 20
21 21
22 where $R(T)$ is either the sum of squared differences or the sum of absolute differences 22
23 of the response values in the learning sample and the predicted values corresponding 23
24 to the fitted tree, and once again $|T|$ is the number of terminal nodes and α is the con- 24
25 tribution to the measure for each terminal node. So, as is the case with classification, 25
26 a biased resubstitution measure is used in the pruning process, and the $\alpha|T|$ term serves 26
27 to penalize the overfitting of trees which partition \mathcal{X} too finely. To select the tree to be 27
28 used as the regression tree, an independent test sample or cross-validation is used to se- 28
29 lect the right-sized tree based on the mean squared prediction error or the mean absolute 29
30 prediction error. 30

31 31
32 Some do not like the discontinuities in the prediction surface which results from 32
33 having a single prediction value for each subset in the partition, and may prefer an al- 33
34 ternative regression method such as MARS (Multivariate Adaptive Regression Splines 34
35 (see [Friedman, 1991](#))) or even OLS regression. However, an advantage that regression 35
36 trees have over traditional linear regression models is the ability to handle a nonho- 36
37 mogeneous response. With traditional regression modeling, one seeks a linear function 37
38 of the inputs and transformations of the inputs to serve as the response surface for the 38
39 entire measurement space, \mathcal{X} . But with a regression tree, some variables can heavily 39
40 influence the predicted response on some subsets of \mathcal{X} , and not be a factor at all on 40
41 other subsets of \mathcal{X} . Also, regression trees can easily make adjustments for various inter- 41
42 actions, whereas discovering the correct interaction terms can be a difficult process in 42
43 traditional regression modeling. On the other hand, a disadvantage with regression trees 43
44 is that additive structure is hard to detect and capture. (See [Hastie et al. \(2001, p. 274\)](#) 44
45 for additional remarks concerning this issue.) 45

4. Other issues pertaining to CART

4.1. Interpretation

It is often stated that trees are easy to interpret, and with regard to seeing how the input variables in a smallish tree are related to the predictions, this is true. But the instability of trees, meaning that sometimes very small changes in the learning sample values can lead to significant changes in the variables used for the splits, can prevent one from reaching firm conclusions about issues such as overall variable importance by merely examining the tree which has been created.

As is the case with multiple regression, if two variables are highly correlated and one is put into the model at an early stage, there may be little necessity for using the other variable at all. But the omission of a variable in the final fitted prediction rule should not be taken as evidence that the variable is not strongly related to the response.

Similarly, correlations among the predictor variables can make it hard to identify important interactions (even though trees are wonderful for making adjustments for such interactions). For example, consider a regression tree initially split using the variable x_3 . If x_1 and x_2 are two highly correlated predictor variables, x_1 may be used for splits in the left portion of a tree, with x_2 not appearing, and x_2 may be used for splits in the right portion of a tree, with x_1 not appearing. A cursory inspection of the tree may suggest the presence of interactions, while a much more careful analysis may allow one to detect that the tree is nearly equivalent to a tree involving only x_1 and x_3 , and also nearly equivalent to a tree involving only x_2 and x_3 , with both of these alternative trees suggesting an additive structure with no, or at most extremely mild, interactions.

4.2. Nonoptimality

It should be noted that the classification and regression trees produced by CART or any other method of tree-structured classification or regression are not guaranteed to be optimal. With CART, at each stage in the tree growing process, the split selected is the one which will immediately reduce the impurity (for classification) or variation (for regression) the most. That is, CART grows trees using a *greedy algorithm*. It could be that some other split would better set things up for further splitting to be effective. However, a tree-growing program that “looks ahead” would require much more time to create a tree.

CART also makes other sacrifices of optimality for gains in computational efficiency. For example, when working with a test sample, after the large initial tree is grown, more use could be made of the test sample in identifying the best subtree to serve as a classifier. But the minimal cost-complexity pruning procedure, which makes use of inferior resubstitution estimates of misclassification rates to determine a good sequence of subtrees to compare using test sample estimates, is a lot quicker than an exhaustive comparison of all possible subtrees using test sample estimates.

4.3. Missing values

Sometimes it is desired to classify a case when one or more of the predictor values is missing. CART handles such cases using *surrogate splits*. A surrogate split is based

1 on a variable other than the one used for the primary split (which uses the variable
2 that leads to the greatest decrease in node impurity). The surrogate split need not be
3 the second best split based on the impurity criterion, but rather it is a split that mimics
4 as closely as possible the primary split; that is, one which maximizes the frequency
5 of cases in the learning sample being separated in the same way that they are by the
6 primary split.

9 5. Bagging

11 Bagging (from bootstrap aggregation) is a technique proposed by Breiman (1996a,
12 1996b). It can be used to improve both the stability and predictive power of classifica-
13 tion and regression trees, but its use is not restricted to improving tree-based predictions.
14 It is a general technique that can be applied in a wide variety of settings to improve pre-
15 dictions.

17 5.1. Motivation for the method

18 In order to gain an understanding of why bagging works, and to determine in what
19 situations one can expect appreciable improvement from bagging, it may be helpful
20 to consider the problem of predicting the value of a numerical response variable, $Y_{\mathbf{x}}$,
21 that will result from, or occur with, a given set of inputs, \mathbf{x} . Suppose that $\phi(\mathbf{x})$ is the
22 prediction that results from using a particular method, such as CART, or OLS regression
23 with a prescribed method for model selection (e.g., using Mallows' C_p to select a model
24 from the class of all linear models that can be created having only first- and second-
25 order terms constructed from the input variables). Letting μ_ϕ denote $E(\phi(\mathbf{x}))$, where
26 the expectation is with respect to the distribution underlying the learning sample (since,
27 viewed as a random variable, $\phi(\mathbf{x})$ is a function of the learning sample, which can be
28 viewed as a high-dimensional random variable) and not \mathbf{x} (which is considered to be
29 fixed), we have that

$$\begin{aligned} & E([Y_{\mathbf{x}} - \phi(\mathbf{x})]^2) \\ &= E([(Y_{\mathbf{x}} - \mu_\phi) + (\mu_\phi - \phi(\mathbf{x}))]^2) \\ &= E([Y_{\mathbf{x}} - \mu_\phi]^2) + 2E(Y_{\mathbf{x}} - \mu_\phi)E(\mu_\phi - \phi(\mathbf{x})) + E([\mu_\phi - \phi(\mathbf{x})]^2) \\ &= E([Y_{\mathbf{x}} - \mu_\phi]^2) + E([\mu_\phi - \phi(\mathbf{x})]^2) \\ &= E([Y_{\mathbf{x}} - \mu_\phi]^2) + \text{Var}(\phi(\mathbf{x})) \\ &\geq E([Y_{\mathbf{x}} - \mu_\phi]^2). \end{aligned}$$

40 (Above, the independence of the future response, $Y_{\mathbf{x}}$, and the predictor based on the
41 learning sample, $\phi(\mathbf{x})$, is used.) Since in nontrivial situations, the variance of the pre-
42 predictor $\phi(\mathbf{x})$ is positive (since typically not all random samples that could be the learning
43 sample yield the sample value for the prediction), so that the inequality above is strict,
44 this result gives us that if $\mu_\phi = E(\phi(\mathbf{x}))$ could be used as a predictor, it would have a
45 smaller mean squared prediction error than does $\phi(\mathbf{x})$.

1 Of course, in typical applications, μ_ϕ cannot serve as the predictor, since the infor- 1
2 mation needed to obtain the value of $E(\phi(\mathbf{x}))$ is not known. To obtain what is sometimes 2
3 referred to as the *true* bagging estimate of $E(\phi(\mathbf{x}))$, the expectation is based on the 3
4 empirical distribution corresponding to the learning sample. In principle, it is possible to 4
5 obtain this value, but in practice it is typically too difficult to sensibly obtain, and so the 5
6 bagged prediction of $Y_{\mathbf{x}}$ is taken to be 6

$$\frac{1}{B} \sum_{b=1}^B \phi_b^*(\mathbf{x}),$$

7
8
9
10
11 where $\phi_b^*(\mathbf{x})$ is the prediction obtained when the base regression method (e.g., CART) is 11
12 applied to the b th bootstrap sample drawn (with replacement) from the original learning 12
13 sample. That is, to use bagging to obtain a prediction of $Y_{\mathbf{x}}$ in a regression setting, one 13
14 chooses a regression method (which is referred to as the *base method*), and applies the 14
15 method to B bootstrap samples drawn from the learning sample. The B predicted values 15
16 obtained are then averaged to produce the final prediction. 16

17 In the classification setting, B bootstrap samples are drawn from the learning sample, 17
18 and a specified classification method (e.g., CART) is applied to each bootstrap sample 18
19 to obtain a predicted class for a given input, \mathbf{x} . The final prediction—the one that results 19
20 from bagging the specified base method—is the class that occurs most frequently in the 20
21 B predictions. 21

22 An alternative scheme is to bag class probability estimates for each class and then let 22
23 the predicted class be the one with the largest average estimated probability. For exam- 23
24 ple, with classification trees one has a predicted class corresponding to each terminal 24
25 node, but there is also an estimate of the probability that a case having \mathbf{x} corresponding 25
26 to a specific terminal node belongs to a particular class. There is such an estimate for 26
27 each class, and to predict the class for \mathbf{x} , these estimated probabilities from the B trees 27
28 can be averaged, and the class corresponding to the largest average estimated proba- 28
29 bility chosen. This can yield a different result than what is obtained by simple voting. 29
30 Neither of the two methods works better in all cases. [Hastie et al. \(2001\)](#) suggest that 30
31 averaging the probabilities tends to be better for small B , but also includes an example 31
32 having the voting method doing slightly better with a large value of B . 32

33 Some have recommended using 25 or 50 for B , and in a lot of cases going beyond 33
34 25 bootstrap samples will lead to little additional improvement. However, [Figure 8.10](#) 34
35 of [Hastie et al. \(2001\)](#) shows that an appreciable amount of additional improvement 35
36 can occur if B is increased from 50 to 100 (and that the misclassification rate remained 36
37 nearly constant for all choices of B greater than or equal to 100), and so taking B to be 37
38 100 may be beneficial in some cases. Although making B large means that creating the 38
39 classifier will take longer, some of the increased time requirement is offset by the fact 39
40 that with bagging one does not have to use cross-validation to select the right amount of 40
41 complexity or regularization. When bagging, one can use the original learning sample as 41
42 a test set. (A test set is supposed to come from the same population the learning sample 42
43 comes from, and in bagging, the learning samples for the B predictors are randomly 43
44 drawn from the original learning sample of available cases. A test set can easily be 44
45 drawn as well, in the same way, although [Breiman \(1996a\)](#) suggests that one may use 45

1 the original learning sample as a test set (since if a huge test set is randomly selected 1
2 from the original learning sample of size N , each of the original cases should occur in 2
3 the huge test set with an observed sample proportion of roughly N^{-1} , and so testing 3
4 with a huge randomly drawn test set should be nearly equivalent to testing with the 4
5 original learning sample.) Alternatively, one could use *out-of-bag* estimates, letting the 5
6 cases not selected for a particular bootstrap sample serve as independent test sample 6
7 cases to use in the creation of the classifier from the bootstrap sample. 7
8

9 5.2. When and how bagging works 9

10 Bagging works best when the base regression or classification procedure that is being 10
11 bagged is not very stable. That is, when small changes in the learning sample can often 11
12 result in appreciable differences in the predictions obtained using a specified method, 12
13 bagging can result in an appreciable reduction in average prediction error. (See [Breiman](#) 13
14 [\(1996b\)](#) for additional information about instability.) 14

15 That bagging will work well when applied to a regression method which is rather 15
16 unstable for the situation at hand is suggested by the result shown in the preceding 16
17 subsection. It can be seen that the difference of the mean squared prediction error for 17
18 predicting $Y_{\mathbf{x}}$ with a specified method, 18
19

$$20 \quad E([Y_{\mathbf{x}} - \phi(\mathbf{x})]^2), \quad 20$$

21 and the mean squared prediction error for predicting $Y_{\mathbf{x}}$ with the mean value of the 21
22 predictor, $\mu_{\phi} = E(\phi(\mathbf{x}))$, 22
23

$$24 \quad E([Y_{\mathbf{x}} - \mu_{\phi}]^2), \quad 24$$

25 is equal to the variance of the predictor, $\text{Var}(\phi(\mathbf{x}))$. The larger this variance is relative 25
26 to the mean squared prediction error of the predictor, the greater the percentage reduction 26
27 of the mean squared prediction error will be if the predictor is replaced by its mean. In 27
28 situations for which the predictor has a relatively large variance, bagging can apprecia- 28
29 bly reduce the mean squared prediction error, provided that the learning sample is large 29
30 enough for the bootstrap estimate of μ_{ϕ} to be sufficiently good. 30
31

32 When predicting $Y_{\mathbf{x}}$ using OLS regression in a case for which the proper form of the 32
33 model is *known* and all of the variables are included in the learning sample, the error 33
34 term distribution is approximately normal and the assumptions associated with least 34
35 squares are closely met, and the sample size is not too small, there is little to be gained 35
36 from bagging because the prediction method is pretty stable. But in cases for which the 36
37 correct form of the model is complicated and not known, there are a lot of predictor 37
38 variables, including some that are just noise not related to the response variable, and the 38
39 sample size is not really large, bagging may help a lot, if a generally unstable regression 39
40 method such as CART is used. 40

41 Like CART, other specific methods of tree-structured regression, along with MARS 41
42 and neural networks, are generally unstable and should benefit, perhaps rather greatly, 42
43 from bagging. In general, when the model is fit to the data adaptively, as is done in 43
44 the construction of trees, or when using OLS regression with some sort of stepwise 44
45 procedure for variable selection, bagging tends to be effective. Methods that are much 45

1 more stable, like nearest neighbors regression and ridge regression, are not expected to 1
2 greatly benefit from bagging, and in fact they may suffer a degradation in performance. 2
3 Although the inequality in the preceding subsection indicates that μ_ϕ will not be a worse 3
4 predictor than $\phi(\mathbf{x})$, since in practice the bagged estimate of μ_ϕ has to be used instead 4
5 of the true value, it can be that this lack of correspondence results in bagging actually 5
6 hurting the prediction process in a case for which the regression method is rather stable 6
7 and the variance of $\phi(\mathbf{x})$ is small (and so there is not a lot to be gained even if μ_ϕ could 7
8 be used). However, in regression settings for which bagging is harmful, the degradation 8
9 of performance tends to be slight. 9

10 In order to better understand why, and in what situations, bagging works for classifi- 10
11 cation, it may be helpful to understand the concepts of bias and variance for classifiers. 11
12 A classifier ϕ , viewed as a function of the learning sample, is said to be *unbiased* at \mathbf{x} if 12
13 when applied to a randomly drawn learning sample from the parent distribution of the 13
14 actual learning sample, the class which is predicted for \mathbf{x} with the greatest probability 14
15 is the one which actually occurs with the greatest probability with \mathbf{x} ; that is, the class 15
16 predicted by the Bayes classifier. (The Bayes classifier is an ideal classifier that always 16
17 predicts the class which is most likely to occur with a given set of inputs, \mathbf{x} . The Bayes 17
18 classifier will not always make the correct prediction, but it is the classifier with the 18
19 smallest possible misclassification rate. For example, suppose that for a given \mathbf{x} , *class 1* 19
20 occurs with probability 0.4, *class 2* occurs with probability 0.35, and *class 3* occurs with 20
21 probability 0.25. The Bayes classifier makes a prediction of *class 1* for this \mathbf{x} . The prob- 21
22 ability of a misclassification for this \mathbf{x} is 0.6, but any prediction other than *class 1* will 22
23 result in a greater probability of misclassification. It should be noted that in practice it is 23
24 rare to have the information necessary to determine the Bayes classifier, in which case 24
25 the best one can hope for is to be able to create a classifier that will perform similarly to 25
26 the Bayes classifier.) Letting \mathcal{U}_ϕ be the subset of the measurement space on which ϕ is 26
27 unbiased, and \mathcal{B}_ϕ be the subset of the measurement space on which it is not unbiased, if 27
28 we could apply ϕ to a large set of independently drawn replications from the parent dis- 28
29 tribution of the learning sample to create an ensemble of classifiers, and let them vote, 29
30 then with high probability this aggregated classifier would produce the same predicted 30
31 class that the Bayes classifier will (due to a law of large numbers effect), *provided that* 31
32 *the \mathbf{x} being classified belongs to \mathcal{U}_ϕ .* 32

33 The *variance* of a classifier ϕ is defined to be the difference of the probability that 33
34 \mathbf{X} (a random set of inputs from the distribution corresponding to the learning sample) 34
35 belongs to \mathcal{U}_ϕ and is classified correctly by the Bayes classifier, and the probability 35
36 that \mathbf{X} belongs to \mathcal{U}_ϕ and is classified correctly by ϕ . (To elaborate, here we consider 36
37 a random vector, \mathbf{X} , and its associated class, and state that \mathbf{X} is classified correctly by 37
38 ϕ if $\phi(\mathbf{X})$ is the same as the class associated with \mathbf{X} , where $\phi(\mathbf{X})$ is a function of \mathbf{X} 38
39 *and* the learning sample, which is considered to be random. The situation is similar for 39
40 \mathbf{X} being classified correctly by the Bayes classifier, except that the Bayes classifier is 40
41 not a function of the learning sample.) Since an aggregated classifier will behave very 41
42 similarly to the Bayes classifier on the subset of the measurement space on which the 42
43 base classifier is unbiased (provided that the ensemble of base classifiers from which 43
44 the aggregated classifier is constructed is suitably large), the aggregated classifier can 44
45 have a variance very close to zero, even if the base classifier does not. The key is that the 45

1 voting greatly increases the probability that the best prediction for \mathbf{x} (the prediction of 1
2 the Bayes classifier, which is not necessarily the correct prediction for a particular case) 2
3 *will be made* provided that \mathbf{x} belongs to the part of the measurement space for which 3
4 the base classifier will make the best prediction with a greater probability than it makes 4
5 any other prediction. (Note that the probabilities for predicting the various classes with 5
6 the base classifier are due to the random selection of the learning sample as opposed 6
7 to being due to some sort of random selection given a fixed learning sample.) Even if 7
8 the base classifier only slightly favors the best prediction, the voting process ensures 8
9 that the best prediction is made with high probability. When bagging is applied using 9
10 a particular base classifier and a given learning sample, the hope is that the learning 10
11 sample is large enough so that bootstrap samples drawn from it are not too different 11
12 from random replications from the parent distribution of the learning sample (which is 12
13 assumed to correspond to the distribution of cases to be classified in the future), and that 13
14 the base classifier is unbiased over a large portion of the measurement space (or more 14
15 specifically, a subset of the measurement space having a large probability). 15
16 The *bias* of a classifier ϕ is defined to be the difference in the probability that \mathbf{X} 16
17 belongs to \mathcal{B}_ϕ and is classified correctly by the Bayes classifier, and the probability 17
18 that \mathbf{X} belongs to \mathcal{B}_ϕ and is classified correctly by ϕ . Thus the bias of ϕ pertains to 18
19 the difference in the performance of ϕ and that of the Bayes classifier when \mathbf{X} takes a 19
20 value in the part of the measurement space on which ϕ is more likely to predict a class 20
21 other than the class corresponding to the best possible prediction. (Note that the sum 21
22 of the variance of ϕ and the bias of ϕ equals the difference in the misclassification rate 22
23 of ϕ and the misclassification rate of the Bayes classifier. Breiman (1998, Section 2) 23
24 gives more information about bias and variance in the classification setting, suggests 24
25 that these terms are not ideal due to a lack of correspondence with bias and variance in 25
26 more typical settings, and notes that others have given alternative definitions.) Just as 26
27 the process of replication and voting makes a bagged classifier, with high probability, 27
28 give the same predictions as the Bayes classifier on the subset of the measurement space 28
29 on which the base classifier is unbiased, it makes a bagged classifier, with high proba- 29
30 bility, give predictions *different* from those given by the Bayes classifier on the subset 30
31 of the measurement space on which the base classifier is biased. But if this subset of 31
32 the measurement space is rather small, as measured by its probability, the amount by which 32
33 the bias can increase due to aggregation is bounded from above by a small number. *So a* 33
34 *key to success with bagging is to apply it to a classification method that has a small bias* 34
35 *with the situation at hand (which typically corresponds to a relatively large variance).* 35
36 It does not matter if the classifier has a large variance, perhaps due to instability, since 36
37 the variance can be greatly reduced by bagging. (It should be noted that this simple ex- 37
38 planation of why bagging a classifier having low bias works ignores effects due to the 38
39 fact that taking bootstrap samples from the original learning sample is not the same as 39
40 having independent samples from the parent distribution of the learning sample. But if 40
41 the original learning sample is not too small, such effects can be relatively small, and 41
42 so the preceding explanation hopefully serves well to get across the main idea. One can 42
43 also see Breiman (1996a, Section 4.2), which contains an explanation of why bagging 43
44 works in the classification setting.) 44
45

1 Generally, bagging a good classifier tends to improve it, while bagging a bad classi- 1
2 fier can make it worse. (For a simple example of the latter phenomenon, see [Hastie et](#) 2
3 [al. 2001, Section 8.7.1.](#)) But bagging a nearest neighbors classifier, which is relatively 3
4 stable, and may be rather good in some settings, can lead to no appreciable change, for 4
5 better or for worse, in performance. Bagging the often unstable classifier CART, which 5
6 is typically decent, but not always great, can often make it close to being ideal (meaning 6
7 that its misclassification rate is close to the Bayes rate, which is a lower bound which 7
8 is rarely achieved when creating classifiers from a random sample). This is due to the 8
9 fact that if a carefully created classification tree is not too small, it will typically have 9
10 a relatively small bias, but perhaps a large variance, and bagging can greatly decrease 10
11 the variance without increasing the bias very much. Results in [Breiman \(1996a\)](#) show 11
12 that bagged trees outperformed nearest neighbors classifiers, which were not improved 12
13 by bagging. Bagging can also improve the performances of neural networks, which are 13
14 generally unstable, but like CART, tend to have relatively low bias. 14

15 [Breiman \(1996a\)](#) gives some indications of how well bagging can improve predic- 15
16 tions, and of the variability in performance when different data sets are considered. With 16
17 the classification examples examined, bagging reduced CART's misclassification rates 17
18 by 6% to 77%. When the amount of improvement due to bagging is small, it could be 18
19 that there is little room for improvement. That is, it could be that both the unbagged 19
20 and the bagged classifier have misclassification rates close to the Bayes rate, and that 20
21 bagging actually did a good job of achieving the limited amount of improvement which 21
22 was possible. When compared with 22 other classification methods used in the Statlog 22
23 Project (see [Michie et al., 1994](#)) on four publicly available data sets from the Stat- 23
24 log Project, [Breiman \(1996a\)](#) shows that bagged trees did the best overall (although it 24
25 should be noted that boosted classifiers were not considered). With the regression ex- 25
26 amples considered by [Breiman \(1996a\)](#), bagging reduced CART's mean squared error 26
27 by 21% to 46%. 27
28
29

30 6. Boosting 30

31
32 Boosting is a method of combining classifiers, which are iteratively created from 32
33 weighted versions of the learning sample, with the weights adaptively adjusted at each 33
34 step to give increased weight to the cases which were misclassified on the previous step. 34
35 The final predictions are obtained by weighting the results of the iteratively produced 35
36 predictors. 36

37 Boosting was originally developed for classification, and is typically applied to *weak* 37
38 *learners*. For a two-class classifier, a weak learner is a classifier that may only be slightly 38
39 better than random guessing. Since random guessing has an error rate of 0.5, a weak 39
40 classifier just has to predict correctly, on average, slightly more than 50% of the time. An 40
41 example of a weak classifier is a *stump*, which is a two node tree. (In some settings, even 41
42 such a simple classifier can have a fairly small error rate (if the classes can be nearly 42
43 separated with a single split). But in other settings, a stump can have an error rate of 43
44 almost 0.5, and so stumps are generally referred to as weak learners.) However, boosting 44
45 is not limited to being used with weak learners. It can be used with classifiers which are 45

1 fairly accurate, such as carefully grown and pruned trees, serving as the *base learner*. 1
2 [Hastie et al. \(2001\)](#) claim that using trees with between four and eight terminal nodes 2
3 works well in most cases, and that performance is fairly insensitive to the choice, from 3
4 this range, which is made. [Friedman et al. \(2000\)](#) give an example in which boosting 4
5 stumps does appreciably worse than just using a single large tree, but boosting eight 5
6 node trees produced an error rate that is less than 25 percent of the error rate of the 6
7 single large tree, showing that the choice of what is used as the base classifier can make a 7
8 large difference, and that the popular choice of boosting stumps can be far from optimal. 8
9 However, they also give another example for which stumps are superior to larger trees. 9
10 As a general strategy, one might choose stumps if it is suspected that effects are additive, 10
11 and choose larger trees if one anticipates interactions for which adjustments should be 11
12 made. 12

14 6.1. AdaBoost 14

15 AdaBoost is a boosting algorithm developed by [Freund and Schapire \(1996\)](#) to be used 15
16 with classifiers. There are two versions of AdaBoost. 16

17 *AdaBoost.M1* first calls for a classifier to be created using the learning sample (that 17
18 is, the base learner is fit to the learning sample), with every case being given the same 18
19 weight. If the learning sample consists of N cases, the initial weights are all $1/N$. The 19
20 weights used for each subsequent step depend upon the weighted resubstitution error 20
21 rate of the classifier created in the immediately preceding step, with the cases being 21
22 misclassified on a given step being given greater weight on the next step. 22

23 Specifically, if $I_{m,n}$ is equal to 1 if the n th case is misclassified on the m th step, and 23
24 equal to 0 otherwise, and $w_{m,n}$ is the weight of the n th case for the m th step, where the 24
25 weights are positive and sum to 1, for the weighted resubstitution error rate for the m th 25
26 step, e_m , we have 26

$$27 e_m = \sum_{n=1}^N w_{m,n} I_{m,n}. \quad 27$$

28 The weights for the $(m + 1)$ th step are obtained from the weights for the m th step by 28
29 multiplying the weights for the correctly classified cases by $e_m/(1 - e_m)$, which should 29
30 be less than 1, and then multiplying the entire set of values by the number greater than 30
31 1 which makes the N values sum to 1. This procedure downweights the cases which 31
32 were correctly classified, which is equivalent to giving increased weight to the cases 32
33 which were misclassified. For the second step, there will only be two values used for the 33
34 weights, but as the iterations continue, the cases which are often correctly classified can 34
35 have weights much smaller than the cases which are the hardest to classify correctly. 35
36 If the classification method of the weak learner does not allow weighted cases, then 36
37 from the second step onwards, a random sample of the original learning sample cases 37
38 is drawn to serve as the learning sample for that step, with independent selections made 38
39 using the weights as the selection probabilities. 39

40 The fact that the cases misclassified on a given step are given increased weight on the 40
41 next step typically leads to a different fit of the base learner, which correctly classifies 41
42 some of the misclassified cases from the previous step, contributing to a low correlation 42
43 44
45

1 of predictions from one step and the next. [Amit et al. \(1999\)](#) provide some details which 1
2 indicate that the AdaBoost algorithm attempts to produce low correlation between the 2
3 predictions of the fitted base classifiers, and some believe that this phenomenon is an 3
4 important factor in the success of AdaBoost (see Breiman's discussion in [Friedman et 4
5 al. \(2000\)](#) and [Breiman \(2001, p. 20\)](#)). 5

6 Assuming that the weighted error rate for each step is no greater than 0.5, the boost- 6
7 ing procedure creates M classifiers using the iteratively adjusted weights. Values such 7
8 as 10 and 100 have been used for M . But in many cases it will be better to use a larger 8
9 value, such as 200, since performance often continues to improve as M approaches 200 9
10 or 400, and it is rare that performance will suffer if M is made to be too large. The M 10
11 classifiers are then combined to obtain a single classifier to use. If the error rate exceeds 11
12 0.5, which should not occur with only two classes, but could occur if there are more than 12
13 2 classes, the iterations are terminated, and only the classifiers having error rates less 13
14 than 0.5 are combined. In either case, the classifiers are combined by using a weighted 14
15 voting to assign the predicted class for any input \mathbf{x} , with the classifier created on the 15
16 m th step being given weight $\log((1 - e_m)/e_m)$. This gives the greatest weights to the 16
17 classifiers having the lowest weighted resubstitution error rates. Increasing the number 17
18 of iterations for AdaBoost.M1 drives the training error rate of the composite classifier 18
19 to zero. The error rate for an independent test sample need not approach zero, but in 19
20 many cases it can get very close to the smallest possible error rate. 20

21 Breiman's discussion in [Friedman et al. \(2000\)](#) makes the point that after a large 21
22 number of iterations, if one examines the weighted proportion of times that each of the 22
23 learning sample cases has been misclassified, the values tend to be all about the same. 23
24 So instead of stating that AdaBoost concentrates on the hard to classify cases, as some 24
25 have done, it is more accurate to state that AdaBoost places (nearly) equal importance 25
26 on correctly classifying each of the cases in the learning sample. [Hastie et al. \(2001\)](#) 26
27 show that AdaBoost.M1 is equivalent to forward stagewise additive modeling using an 27
28 exponential loss function. 28

29 *AdaBoost.M2* is equivalent to AdaBoost.M1 if there are just two classes, but appears 29
30 to be better than AdaBoost.M1 (see results in [Freund and Schapire, 1996](#)) if there are 30
31 more than two classes, because for AdaBoost.M1 to work well the weak learners should 31
32 have weighted resubstitution error rates less than 0.5, and this may be rather difficult to 32
33 achieve if there are many classes. The weak learners used with AdaBoost.M2 assign to 33
34 each case a set of *plausibility values* for the possible classes. They also make use of a 34
35 loss function that can assign different penalties to different types of misclassifications. 35
36 The loss function values used are updated for each iteration to place more emphasis on 36
37 avoiding the same types of misclassifications that occurred most frequently in the pre- 37
38 vious step. After the iterations have been completed, for each \mathbf{x} of interest the sequence 38
39 of classifiers is used to produce a weighted average of the plausibilities for each class, 39
40 and the class corresponding to the largest of these values is taken to be the prediction 40
41 for \mathbf{x} . See [Freund and Schapire \(1996, 1997\)](#) for details pertaining to AdaBoost.M2. 41

42 6.2. Some related methods 42

43
44 [Breiman \(1998\)](#) presents an alternative method of creating a classifier based on combin- 44
45 ing classifiers that are constructed using adaptively resampled versions of the learning 45

1 sample. He calls his method *arc-x4* and he refers to AdaBoost as *arc-fs* (in honor of 1
2 Freund and Schapire), since he considers both *arc-x4* and *arc-fs* to be arcing algorithms 2
3 (and so Breiman considered boosting to be a special case of arcing). When the per- 3
4 formances of these two methods were compared in various situations, Breiman (1998) 4
5 found that there were no large differences, with one working a little better in some sit- 5
6 uations, and the other working a little better in other situations. He concludes that the 6
7 *adaptive resampling is the key to success*, and not the particular form of either algo- 7
8 rithm, since although both use adaptive resampling, with misclassified cases receiving 8
9 larger weights for the next step, they are quite different in other ways (for example, they 9
10 differ in how the weights are established at each step, and they differ in how the voting 10
11 is done after the iterations have been completed, with *arc-x4* using unweighted voting). 11

12 Schapire and Singer (1998) improve upon AdaBoost.M2, with their *AdaBoost.MH* 12
13 algorithm, and Friedman et al. (2000) propose a similar method, called *Real AdaBoost*. 13
14 Applied to the two class setting, Real AdaBoost fits an additive model for the logit trans- 14
15 formation of the probability that the response associated with \mathbf{x} is a particular one of the 15
16 classes, and it does this in a stagewise manner using a loss function which is appropri- 16
17 ate for fitting logistic regression models. Friedman et al. (2000) also propose *Gentle* 17
18 *AdaBoost* and *LogitBoost* algorithms, which in some cases, but not consistently, give 18
19 improved performance when compared to the other AdaBoost algorithms. Bühlmann 19
20 and Yu, in their discussion of Friedman et al. (2000), suggest boosting bagged stumps 20
21 or larger trees, and they give some results pertaining to this method, which they refer to 21
22 as *bag-boosting*. 22

23 The boosting approach can be extended to the regression setting. Friedman (2001) 23
24 proposes *TreeBoost* methods for function approximation, which create additive models, 24
25 having regression trees as components, in a stagewise manner using a *gradient boosting* 25
26 technique. Hastie et al. (2001) refer to this approach as *MART* (Multiple Additive Re- 26
27 gression Trees), and they find that incorporating a shrinkage factor to encourage slow 27
28 learning can improve accuracy. 28

30 6.3. When and how boosting works 30

31
32 When used with a rather complex base classification method that is unstable, such as 32
33 carefully grown and pruned trees, boosting, and arcing in general, like bagging, in many 33
34 cases can dramatically lower the error rate of the classification method. Stated simply, 34
35 this is due *in part* to the fact that unstable methods have relatively large variances, and 35
36 boosting decreases the variance without increasing the bias. Breiman (1998) shows that 36
37 when applied to linear discriminant analysis (LDA), which is a fairly stable method of 37
38 classification, neither bagging nor boosting has any appreciable effect. In some cases 38
39 (see, for example, results reported by Freund and Schapire (1996) and Quinlan (1996), 39
40 where C4.5 was compared to boosted C4.5), boosting *can* make a classifier appreciably 40
41 worse, but this does not seem to happen in a large percentage of settings which have 41
42 been investigated, and when it does happen, perhaps a small learning sample size is a 42
43 contributing factor. Several studies have indicated that boosting can be outperformed 43
44 when the classes have significant overlap. Nevertheless, it is often the case that some 44
45 form of boosted classifier can do nearly as good or better than any other classifier. 45

1 As to how boosting typically achieves such a low misclassification rate, the opinions 1
2 are many and varied. Friedman et al. (2000) point out that when used with a rather 2
3 simple base classifier which is stable but perhaps highly biased, such as trees lim- 3
4 ited to a rather small number of terminal nodes, which is a common way of applying 4
5 boosting, the success of boosting is due much more to bias reduction (see Schapire 5
6 et al., 1998) than it is to variance reduction, and this makes boosting fundamentally 6
7 different than bagging, despite the fact that both methods combine classifiers based 7
8 on various perturbations of the learning sample. Schapire et al. (1998) offer the opin- 8
9 ion that AdaBoost works because of its ability to produce generally high margins. But 9
10 Breiman (1999) claims that their explanation is incomplete, and provides some expla- 10
11 nation of his own. Some have attempted to explain boosting from a Bayesian point 11
12 of view. Friedman et al. (2000) give support to their opinion that viewing boosting 12
13 procedures as stagewise algorithms for fitting additive models greatly helps to explain 13
14 their performance, and make a connection to maximum likelihood. In the discussion of 14
15 Friedman et al. (2000), various authors offer additional explanation as to why boosting 15
16 works. In particular, Breiman finds fault with the explanation provided by Friedman 16
17 et al. (2000), since it does not explain the strong empirical evidence that boosting is 17
18 generally resistant to overfitting. Friedman et al. (2000) provide one example of boost- 18
19 ing leading to an overfit predictor. Also, Rätsch et al. (2001) conclude that overfitting 19
20 can occur when there is a lot of noise, and they provide some pertinent references.) 20
21 In his discussion, Breiman indicates that the key to the success of boosting is that 21
22 it produces classifiers of reasonable strength which have low correlation. Breiman 22
23 provided additional insight with his 2002 Wald Lecture on Machine Learning (see 23
24 <http://stat-www.berkeley.edu/users/breiman/wald2002-1.pdf>).

25 While the preceding paragraph indicates that it is not fully understood why boosting 25
26 works as well as it does, there is no question about the fact that it can work very well. In 26
27 various comparisons with bagging, boosting generally, but not always, does better, and 27
28 leading researchers in the field of machine learning tend to favor boosting. It should be 28
29 noted that some of the comparisons of boosting to bagging are a bit misleading because 29
30 they use stumps as the base classifier for both boosting and bagging, and while stumps, 30
31 being weak learners, can work very well with boosting, they can be too stable and have 31
32 too great a bias to work really well with bagging. Trees larger than stumps tend to work 32
33 better with bagging, and so a fairer comparison would be to compare bagging fairly 33
34 large trees, boosting both stumps and slightly larger trees, and using an assortment of 34
35 good classifiers without applying boosting or bagging. Boosting and bagging classifiers 35
36 other than trees can also be considered. From the results of some such studies, it can be 36
37 seen that often there is not a lot of difference in the performances of bagged large trees 37
38 and boosted large trees, although it appears that boosting did better more often than not. 38

39 Of course, the main interest should be in an overall winner, and not the comparison 39
40 of boosting with bagging. However, various studies indicate that no one method works 40
41 best in all situations. This suggests that it may be wise to be familiar with a large variety 41
42 of methods, and to try to develop an understanding about the types of situations in which 42
43 each one works well. Also, one should be aware that there are other types of methods 43
44 for classification and regression, such as kernel-based methods and support vector ma- 44
45 chines, that have not been covered in this chapter. Indeed, one other class of methods, 45

1 of the P&C variety, *random forests* with random feature selection, have been shown by 1
2 [Breiman \(2001\)](#) to compare very favorably with AdaBoost. Furthermore, despite the fact 2
3 that random forests use ensembles of classifiers created from independent identically 3
4 distributed learning samples, as opposed to the adaptively produced learning samples 4
5 used in boosting, [Breiman \(2001\)](#) claims that AdaBoost is similar to a random forest. 5
6 So it seems premature to declare that boosting is the clear-cut best of the new methods 6
7 pertaining to classification developed during the past 15 years, while at the same time it 7
8 is safe to declare that boosting can be used to create classifiers that are very often among 8
9 the best. 9

12 References 12

- 13
14 Amit, Y., Blanchard, G., Wilder, K. (1999). Multiple randomized classifiers: MRCL. Technical Report. De- 14
15 partment of Statistics, University of Chicago. 15
16 Breiman, L. (1996a). Bagging predictors. *Machine Learning* **24**, 123–140. 16
17 Breiman, L. (1996b). Heuristics of instability and stabilization in model selection. *Ann. Statist.* **24**, 2350– 17
18 2383. 18
19 Breiman, L. (1998). Arcing classifiers (with discussion). *Ann. Statist.* **26**, 801–849. 19
20 Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Comput.* **11**, 1493–1517. 20
21 Breiman, L. (2001). Random forests. *Machine Learning* **45**, 5–32. 21
22 Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). *Classification and Regression Trees*. 22
23 Wadsworth, Pacific Grove, CA. 23
24 Buntine, W.L. (1992). Learning classification trees. *Statist. Comput.* **2**, 63–73. 24
25 Ciampi, A., Chang, C.-H., Hogg, S., McKinney, S. (1987). Recursive partitioning: a versatile method for 25
26 exploratory data analysis in biostatistics. In: MacNeil, I.B., Umphrey, G.J. (Eds.), *Biostatistics*. Reidel, 26
27 Dordrecht. 27
28 Clark, L.A., Pregibon, D. (1992). Tree based models. In: Chambers, J.M., Hastie, T.J. (Eds.), *Statistical Mod-* 28
29 *els in S*. Wadsworth and Brooks/Cole, Pacific Grove, CA. 29
30 Crawford, S.L. (1989). Extensions to the CART algorithm. *Int. J. Man-Machine Stud.* **31**, 197–217. 30
31 Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Inform. Comput.* **121**, 256–285. 31
32 Freund, Y., Schapire, R. (1996). Experiments with a new boosting algorithm. In: Saitta, L. (Ed.), *Machine* 32
33 *Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann, San Francisco, 33
34 CA. 34
35 Freund, Y., Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to 35
36 boosting. *J. Comput. System Sci.* **55**, 119–139. 36
37 Friedman, J. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Statist.* **19**, 1–141. 37
38 Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *Ann. Statist.* **29**, 1189– 38
39 1232. 39
40 Friedman, J., Hastie, T., Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with 40
41 discussion). *Ann. Statist.* **28**, 337–407. 41
42 Gelfandi, S.B., Ravishanker, C.S., Delp, E.J. (1991). An iterative growing and pruning algorithm for classifi- 42
43 cation tree design. *IEEE Trans. Pattern Anal. Machine Intelligence* **13**, 163–174. 43
44 Gentle, J.E. (2002). *Elements of Computational Statistics*. Springer-Verlag, New York. 44
45 Harrell, F.E. (2001). *Regression Modeling Strategies: with Applications to Linear Models, Logistic Regres-* 45
46 *sion, and Survival Analysis*. Springer-Verlag, New York. 46
47 Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference,* 47
48 *and Prediction*. Springer-Verlag, New York. 48
49 Kass, G.V. (1980). An exploratory technique for investigating large quantities of categorical data. *Appl. Sta-* 49
50 *tist.* **29**, 119–127. 50
51 Loh, W.-Y., Shih, Y.-S. (1997). Split selection methods for classification trees. *Statist. Sin.* **7**, 815–840. 51

- 1 Martinez, W.L., Martinez, A.R. (2002). *Computational Statistics Handbook with MATLAB*. Chapman and 1
2 Hall/CRC, Boca Raton, FL. 2
- 3 Michie, D., Spiegelhalter, D.J., Taylor, C.C. (1994). *Machine Learning, Neural and Statistical Classification*. 3
4 Horwood, London. 4
- 5 Morgan, J.N., Messenger, R.C. (1973). THAID: a sequential search program for the analysis of nominal scale 5
6 dependent variables. Institute for Social Research, University of Michigan, Ann Arbor, MI. 6
- 7 Morgan, J.N., Sonquist, J.A. (1963). Problems in the analysis of survey data, and a proposal. *J. Amer. Statist.* 7
8 *Assoc.* **58**, 415–434. 8
- 9 Olshen, R. (2001). A conversation with Leo Breiman. *Statist. Sci.* **16**, 184–198. 9
- 10 Quinlan, J.R. (1979). Discovering rules by induction from large classes of examples. In: Michie, D. (Ed.), 10
11 *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh. 11
- 12 Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning* **1**, 81–106. 12
- 13 Quinlan, J.R. (1987). Simplifying decision trees. *Int. J. Man–Machine Stud.* **27**, 221–234. 13
- 14 Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA. 14
- 15 Quinlan, J.R. (1996). Bagging, boosting, and C4.5. In: *Proceedings of AAAI '96 National Conference on* 15
16 *Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA. 16
- 17 Rättsch, G., Onoda, T., Müller, K.R. (2001). Soft margins for AdaBoost. *Machine Learning* **42**, 287–320. 17
- 18 Ripley, B.D. (1996). *Pattern Recognition and Neural Networks*. Cambridge Univ. Press, Cambridge. 18
- 19 Schapire, R. (1990). The strength of weak learnability. *Machine Learning* **5**, 197–227. 19
- 20 Schapire, R., Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. In: *Pro-* 20
21 *ceedings of the Eleventh Annual Conference on Computational Learning Theory*. ACM Press. 21
- 22 Schapire, R., Freund, Y., Bartlett, P., Lee, W. (1998). Boosting the margin: a new explanation for the effec- 22
23 tiveness of voting methods. *Ann. Statist.* **26**, 1651–1686. 23
- 24 Steinberg, D., Colla, P. (1995). *CART: Tree-Structured Nonparametric Data Analysis*. Salford Systems, San 24
25 Diego, CA. 25
- 26 Therneau, T.M., Atkinson, E.J. (1997). An introduction to recursive partitioning using the rpart routine. Tech- 26
27 nical Report. Section of Statistics, Mayo Clinic. 27
- 28 28
- 29 29
- 30 30
- 31 31
- 32 32
- 33 33
- 34 34
- 35 35
- 36 36
- 37 37
- 38 38
- 39 39
- 40 40
- 41 41
- 42 42
- 43 43
- 44 44
- 45 45