Topics in Tree-Based Methods

Alex Lauf Goldstein

University of Pennsylvania, alex.lauf.goldstein@gmail.com

Follow this and additional works at: http://repository.upenn.edu/edissertations

Part of the Statistics and Probability Commons

Recommended Citation

Goldstein, Alex Lauf, "Topics in Tree-Based Methods" (2014). Publicly Accessible Penn Dissertations. 1288.
http://repository.upenn.edu/edissertations/1288

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/edissertations/1288
For more information, please contact libraryrepository@pobox.upenn.edu.
Abstract
This work introduces methods and associated software for enhancing the interpretability of fitted models, with emphasis on classification and regression trees. We begin in Chapter 1 by describing novel techniques for growing classification and regression trees designed to induce visually interpretable trees. This is achieved by penalizing splits that extend the subset of features used in a particular branch of the tree. After a brief motivation, we summarize existing methods and introduce new ones, providing illustrative examples throughout. Using a number of real classification and regression datasets, we find that these procedures can offer more interpretable fits than the CART methodology with very modest increases in out-of-sample loss.

These techniques are implemented in the R package itree, described in Chapter 2. In addition to the procedures introduced in Chapter 1, itree implements a method for visualizing the out-of-sample risk as well as the usual classification and regression tree methodologies. Chapter 2 presents illustrative examples and demonstrates itree's usage for aspects of the software that are novel or unique to itree.

Whereas Chapters 1 and 2 relate to tree-based methods, Chapter 3 describes Individual Conditional Expectation (ICE) plots, a methodology for visualizing the model estimated by any supervised learning algorithm. Classical partial dependence plots (PDPs) help visualize the average partial relationship between the predicted response and one or more features. In the presence of substantial interaction effects, the partial response relationship can be heterogeneous. Thus, an average curve, such as the PDP, can obfuscate the complexity of the modeled relationship. Accordingly, ICE plots refine the partial dependence plot by graphing the functional relationship between the predicted response and the feature for individual observations. ICE plots highlight the variation in the fitted values across the range of a covariate, suggesting where and to what extent heterogeneities might exist. In addition to providing a plotting suite for exploratory analysis, we include a visual test for additive structure in the data generating model. The procedures outlined in Chapter 3 are available in the R package ICEbox.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Statistics

First Advisor
Andreas Buja

Keywords
Classification and Regression Trees, Decision Trees, Exploratory Data Analysis, Machine Learning, Model Visualization

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/1288
TOPICS IN TREE-BASED METHODS

Alex L. Goldstein

A DISSERTATION

in

Statistics

For the Graduate Group in
Managerial Science and Applied Economics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2014

Supervisor of Dissertation

Signature ________________________
Andreas Buja
The Liem Sioe Liong/First Pacific Company Professor of Statistics

Graduate Group Chairperson

Signature ________________________
Eric Bradlow
Professor of Marketing, Statistics, and Education

Dissertation Committee
Richard A. Berk, Professor of Criminology and Statistics
Ed George, Universal Furniture Professor of Statistics
Abba Krieger, Robert Steinberg Professor of Statistics and Operations Research
TOPICS IN TREE-BASED METHODS

COPYRIGHT © 2014

Alex L. Goldstein
To my late grandparents, Selma and Irving Lauf – my accomplishments rest on your shoulders.
The set of people who have supported me during the process of earning a Ph.D. is remarkable both for its size and the extraordinary quality of its members. Acknowledging them all here in a manner in keeping with what they deserve results in a document inconsistent with the University’s notion of an “acknowledgment page.” I have chosen to address this puzzle by expressing my immense gratitude to everyone individually.
ABSTRACT

TOPICS IN TREE-BASED METHODS

Alex L. Goldstein

Andreas Buja

This work introduces methods and associated software for enhancing the interpretability of fitted models, with emphasis on classification and regression trees. We begin in Chapter 1 by describing novel techniques for growing classification and regression trees designed to induce visually interpretable trees. This is achieved by penalizing splits that extend the subset of features used in a particular branch of the tree. After a brief motivation, we summarize existing methods and introduce new ones, providing illustrative examples throughout. Using a number of real classification and regression datasets, we find that these procedures can offer more interpretable fits than the CART methodology with very modest increases in out-of-sample loss.

These techniques are implemented in the R package itree, described in Chapter 2. In addition to the procedures introduced in Chapter 1, itree implements a method for visualizing the out-of-sample risk as well as the usual classification and regression tree methodologies. Chapter 2 presents illustrative examples and demonstrates itree’s usage for aspects of the software that are novel or unique to itree.

Whereas Chapters 1 and 2 relate to tree-based methods, Chapter 3 describes Individual Conditional Expectation (ICE) plots, a methodology for visualizing the model estimated by any supervised learning algorithm. Classical partial dependence plots (PDPs) help visualize the average partial relationship between the predicted response and one or more features. In the presence of substantial interaction effects, the partial response relationship can be heterogeneous. Thus, an average curve, such
as the PDP, can obfuscate the complexity of the modeled relationship. Accordingly, ICE plots refine the partial dependence plot by graphing the functional relationship between the predicted response and the feature for individual observations. ICE plots highlight the variation in the fitted values across the range of a covariate, suggesting where and to what extent heterogeneities might exist. In addition to providing a plotting suite for exploratory analysis, we include a visual test for additive structure in the data generating model. The procedures outlined in Chapter 3 are available in the R package ICEbox.
### 3 Statistical Learning Model Visualization with Individual Conditional Expectation Plots

3.1 Introduction .................................................. 53
3.2 Background .................................................... 54
   3.2.1 Survey of Black Box Visualization ........................................... 54
   3.2.2 Friedman's PDP .................................................. 55
3.3 The ICE Toolbox ................................................ 58
   3.3.1 The ICE Procedure ................................................. 58
   3.3.2 The Centered ICE Plot ............................................. 60
   3.3.3 The Derivative ICE Plot ........................................... 61
   3.3.4 Visualizing a Second Feature ...................................... 63
3.4 Simulations ...................................................... 64
   3.4.1 Additivity Assessment ............................................. 65
   3.4.2 Finding interactions and regions of interactions ..................... 66
   3.4.3 Extrapolation Detection ........................................... 68
3.5 Real Data ....................................................... 69
   3.5.1 Depression Clinical Trial ......................................... 71
   3.5.2 White Wine .................................................... 73
   3.5.3 Diabetes Classification in Pima Indians ............................. 74
3.6 A Visual Test for Additivity ..................................... 76
   3.6.1 Procedure ..................................................... 76
   3.6.2 Examples ...................................................... 78
3.7 Discussion ....................................................... 79

### 4 Conclusion ..................................................... 82

### A Appendices ..................................................... 84

A.1 Chapter 1 Supplement:
   - Penalized Split Criteria for Interpretable Trees ...................... 84
     - A.1.1 Gain Function Scaling ........................................ 84
     - A.1.2 One-Sided Split Criteria .................................... 86
     - A.1.3 Algorithms .................................................. 88
     - A.1.4 Out-of-Bag Performance Statistics ........................... 89

A.2 Chapter 2 Supplement:
   - Software for Interpretable Classification and Regression Trees .... 94

A.3 Chapter 3 Supplement:
   - Statistical Learning Model Visualization with Individual Conditional Expectation Plots .......................... 95
     - A.3.1 Algorithms .................................................. 95

### Bibliography .................................................. 96
List of Tables

1.1 Summary of Notation ........................................... 4
2.1 Summary of Notation ........................................... 28
A.1 Scaling of Impurity Functions ................................. 86
A.2 Definitions of One-Sided Impurity Functions .......... 87
A.3 OOB performance of penalization: CART on regression datasets . . . 89
A.4 OOB performance of penalization: One-Sided Purity on regression datasets . . . 90
A.5 OOB performance of penalization: High-Means on regression datasets 90
A.6 OOB performance of penalization: Low-Means on regression datasets 91
A.7 OOB performance of penalization: CART on classification datasets . 92
A.8 OOB performance of penalization: One-Sided Purity on classification datasets . . . 92
A.9 OOB performance of penalization: One-Sided Extremes on classification datasets . . . 93
1.1 CART fit to the Boston Housing data
1.2 CART fit to the Boston Housing data with and without the New Variable Penalty
1.3 CART fit to the Boston Housing data with the New Variable Penalty
1.4 One-sided High-Means fit to the Boston Housing data
1.5 One-Sided Purity fit to the Pima data with and without the New Variable Penalty
1.6 CART fit to the Boston Housing data with the EMA-Style penalty
1.7 High-Means fit to the Boston Housing data with the EMA penalty
1.8 One-Sided Extremes fit to the Pima data with and without the EMA penalty
1.9 One-Sided Purity fit to the Pima data with and without the EMA penalty
1.10 CART fit to the Red Wine data with and without the EMA Penalty

2.1 CART and One-Sided Purity applied to the Boston Housing Data
2.2 CART and One-Sided Purity applied to the Pima Indians data
2.3 CART and One-Sided Purity applied to the Autism data
2.4 Close-up: One-Sided Purity applied to the Autism data
2.5 CART and One-Sided Purity applied to the Digit Recognition data
2.6 CART and One-Sided Extremes applied to the Boston Housing data
2.7 CART and One-Sided Extremes applied to the Digit Recognition data
2.8 Parsimony Example
2.9 CART applied to the Boston Housing data with the New Variable Penalty
2.10 One-Sided Extremes applied to the Digit Recognition data with the New Variable Penalty
2.11 One-Sided Purity applied to the Pima Indians data with and without the EMA penalty
2.12 Boston Housing: Local Risk Estimates for CART
2.13 Pima Indians Data: Local Risk Estimate for CART
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Scatterplot and PDP of $X_2$ versus $Y$ for a sample from the process described in Equation 3.3</td>
<td>58</td>
</tr>
<tr>
<td>3.2</td>
<td>SGB ICE plot for $X_2$ from the data generating process described by Equation 3.3</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>RF ICE plot for the predictor $age$ in the Boston Housing data</td>
<td>60</td>
</tr>
<tr>
<td>3.4</td>
<td>c-ICE plot for $age$ with $x^*$ set to the minimum value of $age$</td>
<td>62</td>
</tr>
<tr>
<td>3.5</td>
<td>d-ICE plot for $age$ in the BHD</td>
<td>63</td>
</tr>
<tr>
<td>3.6</td>
<td>The c-ICE plot for $age$ of Figure 3.4, colored by $rm$</td>
<td>64</td>
</tr>
<tr>
<td>3.7</td>
<td>ICE and d-ICE plots for $S = X_1$ for a GAM</td>
<td>66</td>
</tr>
<tr>
<td>3.8</td>
<td>ICE plots for an SGB fit to the simple interaction model of Equation 3.6</td>
<td>68</td>
</tr>
<tr>
<td>3.9</td>
<td>ICE plots for $S = x_1$ of a RF model fit to Equation 3.7</td>
<td>70</td>
</tr>
<tr>
<td>3.10</td>
<td>ICE plots of a BART model for the effect of treatment on depression score</td>
<td>72</td>
</tr>
<tr>
<td>3.11</td>
<td>ICE plots of a NN model for wine ratings versus $pH$</td>
<td>73</td>
</tr>
<tr>
<td>3.12</td>
<td>ICE plots for a RF for the variable $skin$ colored $age$</td>
<td>75</td>
</tr>
<tr>
<td>3.13</td>
<td>Additivity lineup test for $treatment$ in the depression data of Section 3.5.1</td>
<td>79</td>
</tr>
<tr>
<td>3.14</td>
<td>Additivity lineup test for $pH$ in the white wine dataset of Section 3.5.2</td>
<td>80</td>
</tr>
</tbody>
</table>
Penalized Split Criteria for Interpretable Trees

Abstract

This chapter describes techniques for growing classification and regression trees designed to induce visually interpretable trees. This is achieved by penalizing splits that extend the subset of features used in a particular branch of the tree. After a brief motivation, we summarize existing methods and introduce new ones, providing illustrative examples throughout. Using a number of real classification and regression datasets, we find that these procedures can offer more interpretable fits than the CART methodology with very modest increases in out-of-sample loss.

1.1 Introduction

We assume familiarity with the techniques introduced in Breiman et al. (1984) for fitting binary trees to data. For brevity we refer to these techniques both collectively and individually by the acronym CART. Its authors state that CART is designed to “produce an accurate classifier or to uncover the predictive structure” of a problem.

*This chapter is joint work with Andreas Buja.
In comparison with the former task, the degree to which a model “uncovers structure”
eludes quantification. We offer no help on this front, but adopt [Breiman et al. (1984)]’s
preference for “simple characterizations of the conditions that determine when an
object is [in] one class rather than another” as our guiding principle, which we call
**interpretability**.

What is meant by a “simple characterization”? For classification trees, the question
of whether we predict $y$ is in one class or another is determined by the terminal node
to which its associated $\mathbf{x}$ vector belongs. Hence the conditions leading to $y$’s predicted
class are exactly the sequence of splitting rules that lead to its terminal node. As such,
the tree that offers the simpler sequence of splits also offers the simpler explanation of
$y$’s predicted class. In this sense, splitting procedures that encourage simple sequences
of split rules can result in particularly interpretable trees. Such procedures are the
focus of this chapter.

In Section 1.2 we review the fundamentals of CART, paying special attention to
gain and impurity, the critical functions for tree-growing. Further, we make the no-
ton of “simple sequences of splits” more precise. In Section 1.3 we present novel tree
growing techniques for the usual classification and regression settings where interpret-
ability is desirable. Section 1.4 reviews the out-of-sample performance of these
methods. The evidence suggests that in many cases the methods described in Section
1.3 yield interpretable trees with little sacrifice in generalization error. Section 1.5
concludes.
1.2 Fundamentals of Classification and Regression Trees

1.2.1 Splits and Splitting Criteria

Where possible we follow the terminology and notation of Breiman (1996b), as outlined below. Readers will recall that given a learning sample \( L \) of \( N \) pairs \( z_i = (y_i, x_i) \) from an arbitrary distribution in which \( E(y|x) = f(x) \), the algorithms described in Breiman et al. (1984) output a binary tree \( \hat{f}(x) \) that aims to approximate \( f \) or threshold \( f(x) \) when \( y \) is binary. Here \( \hat{f} \) is called a classification or regression tree depending on whether \( y \) is categorical or continuous, respectively.

For any \( x \), \( \hat{f}(x) \) is given by the mean (for regression) or the most common (in classification) \( y_i \) value over all \( i \in L \) that are in the same terminal node as \( x \), denoted by \( t(x) \). In either case, all observations in a given node share the same fitted value, which we denote \( j(t) \) herein.

Each non-terminal node in the tree is defined by a splitting rule \( s \). Each splitting rule comprises a pair \((x, t)\) consisting of a variable \( x \) and a split location \( t \). The rule \( s = (x_1, 0) \), for instance, divides the \( n_t \) observations in \( t \) into two subsets, depending on whether each \( x \) vector has a positive first coordinate. In this example \( x_1 \) is termed the split variable and 0 the split point. The growing phase consists of selecting the best \( s \) at \( t \) and then sending \( t \)'s observations to the appropriate child nodes, where the recursion begins anew. Though the details of both growing and pruning certainly influence interpretability, our focus here is on tree-growing methods. Defining procedures for choosing splits that lead to interpretable trees is the subject of Section 1.3.

CART determines the “best \( s \)” by the goodness of split criterion or gain function \( \theta(t,s) \) which quantifies the benefit of splitting node \( t \) as per rule \( s \). Each node splits
at

\[ s^* = \arg \max_{s \in S} \theta(t, s), \tag{1.1} \]

meaning we choose the split that maximizes the split criterion, where \( S \) is the set of all possible splits including no split. For CART, \( \theta \) is of the form

\[ \theta(t, s) = \phi(t) - \left[ \frac{n_{tL}}{n_t} \phi(t_L) + \frac{n_{tR}}{n_t} \phi(t_R) \right], \tag{1.2} \]

where \( t_L \) and \( t_R \) are the left and right child nodes defined by \( s \), and \( \phi \) is the loss or so-called impurity function. By multiplying \( \phi_L \) and \( \phi_R \) by the proportion of observations in the left and right child nodes, \( \theta(t, s) \) measures the average improvement in impurity from splitting \( t \) as per rule \( s \). For convenience, Table 1.1 summarizes our notational conventions.

### Table 1.1: Summary of Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{L} )</td>
<td>Training sample of ( N ) ((y_i, \mathbf{x}_i)) pairs</td>
</tr>
<tr>
<td>( \hat{f} )</td>
<td>Recursive partitioning tree grown using the training sample</td>
</tr>
<tr>
<td>( \mathbf{x} )</td>
<td>An arbitrary point in predictor space</td>
</tr>
<tr>
<td>( \hat{f}(\mathbf{x}) )</td>
<td>Tree ( \hat{f}'s ) fitted value at ( \mathbf{x} )</td>
</tr>
<tr>
<td>( t(\mathbf{x}) )</td>
<td>The terminal node to which ( \mathbf{x} ) belongs</td>
</tr>
<tr>
<td>( j(t) )</td>
<td>The fitted value associated with node ( t )</td>
</tr>
<tr>
<td>( t_L, t_R )</td>
<td>Node ( t )'s left and right child nodes if ( t ) is non-terminal</td>
</tr>
<tr>
<td>( n_t )</td>
<td>Number of training observations in node ( t )</td>
</tr>
<tr>
<td>( s )</td>
<td>Splitting rule consisting of a (split variable, split point) pair</td>
</tr>
<tr>
<td>( s_x )</td>
<td>Split variable associated with splitting rule ( s )</td>
</tr>
<tr>
<td>( \theta )</td>
<td>Goodness of split criterion / gain function</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Impurity function (see (1.1) above for the relation between ( \theta ) and ( \phi ))</td>
</tr>
<tr>
<td>( \hat{p}_{k,t} )</td>
<td>Proportion of ( y_i )'s in node ( t ) that are of class ( k ) (for categorical ( y ))</td>
</tr>
<tr>
<td>( \Theta(\hat{f}) )</td>
<td>Loss function (MSE or misclassification rate), for use later</td>
</tr>
</tbody>
</table>
1.2.2 CART Impurity Functions

In a regression setting we typically seek to minimize absolute or squared deviations between fitted and observed values. Though Breiman et al. (1984) presents regression trees based on both criteria, it is commonplace to use squared-error loss and so we set

$$\phi_R(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - j(t))^2.$$  \hspace{1cm} (1.3)

Recall that in regression we set $j(t)$ to the sample mean of the in-node $y$ values, and so readers will quickly identify (1.3) as $t$’s (biased) sample variance, $\hat{\sigma}^2(t)$. Further, as the sample mean minimizes squared error loss, we see that $j(t)$ minimizes empirical within-node impurity.

Though intuitively appealing, when growing trees for classification we do not take $\phi$ to be the weighted average misclassification error (Breiman et al., 1984). The reason is that the misclassification rate is insensitive to certain distinctions in desirability of splits. As a heuristic example, consider the following proposed splits for classifying $y \in \{A, B\}$ in a 100 observation node with $n_A = 70$ and $n_B = 30$.

<table>
<thead>
<tr>
<th>Split</th>
<th>Left Node Distribution</th>
<th>Right Node Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$n_A = 45, n_B = 0$</td>
<td>$n_A = 25, n_B = 30$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$n_A = 60, n_B = 15$</td>
<td>$n_A = 10, n_B = 15$</td>
</tr>
</tbody>
</table>

Here $s_1$ and $s_2$ both have misclassification error of 0.25, even though $s_1$ yields a node without errors. Clearly $s_1$’s left node has zero impurity on $L$ and requires no further splits, making $s_1$ preferable. The difficulty lies in the fact that the misclassification rate is piecewise linear in the sample proportion $p_A$, whereas the example illustrates that the impurity function should decrease more rapidly as $p_A$ approaches 0 or 1. See Buja and Lee (2001) or Buja et al. (2005) for a more complete discussion of impurity functions for classification trees.

Instead, it is common to use either the Gini criterion or Cross-entropy criterion.
For the multiclass problem with \( y \in \mathcal{K} = \{1, 2, \ldots, K\} \) the Gini criterion is written

\[
\phi_G(t) = \sum_{k \in \mathcal{K}} \hat{p}_{k,t}(1 - \hat{p}_{k,t})
\]

where \( \hat{p}_{k,t} \) is the proportion of \( y_i \)'s in node \( t \) that are of class \( k \). Cross-entropy is defined

\[
\phi_{CE}(t) = \sum_{k \in \mathcal{K}} \hat{p}_{k,t} \log(\hat{p}_{k,t}).
\]

It is easy to verify that both functions satisfy the requirement above. Breiman (1996b) notes that empirically, Gini tends to yield splits resulting in purer nodes, especially when \( K > 2 \). In addition, if in-node sample proportions are interpreted as class probability estimates, Gini corresponds to squared-error loss (see Breiman et al. (1984) or Hastie et al. (2009)). In their informative description of the R package \texttt{rpart}, a popular implementation of CART, Therneau and Atkinson (1997) comment that from a practical perspective there is usually little difference between the methods, especially when \( K = 2 \). Like \texttt{rpart}, many software packages implement both criteria but default to Gini. For brevity we do likewise; when referring to the conventional method of growing classification trees we assume Gini impurity as defined in (1.4).

### 1.2.3 Interpretability of Trees

The interpretability of a particular tree is a function of its splitting rules. As an example, consider the regression tree in Figure 1.1. This tree, \( \hat{f} \), is the result of applying the CART procedure to the Boston Housing data, where the goal is to fit median housing prices in census tracts using a variety of features about homes’ average physical characteristics and locations. As our focus is on the growing phase rather than pruning, unless noted otherwise all trees herein cease splitting once the current node contains 5% of all observations.
Figure 1.1: CART fit to the Boston Housing data. Terminal nodes are restricted to contain no fewer than 5% of all observations. In-sample $R^2 = 0.8$.

Now because $\hat{f}$ is a binary tree we can find $\hat{f}(x)$ simply by applying a series of rules. We write $\mathcal{B}_t$ to denote the sequence of split variables leading to node $t$. Let $t$ be the left-most terminal node. Then the sequence of splits leading to $t$ is therefore $(\text{rm}, 6.941), (\text{lstat}, 14.4), (\text{crim}, 6.992), (\text{lstat}, 19.85)$ corresponding to $\mathcal{B}_t = \{\text{rm}, \text{lstat}, \text{crim}, \text{lstat}\}$. As noted previously, the fitted value of an observation for which $x \in t$ is explained by simply enumerating this sequence of rules:

“If the \text{rm} is less than 6.94, \text{lstat} is greater than 14.4, \text{crim} is greater than 6.99, and \text{lstat} is greater than 19.85, then the fitted value is 10.56.”

The node is at depth four and so the explanation is an intersection of four rules. Now clearly the more features used to reach a given terminal node $t$, the more difficult it is to summarize the partition of $\mathcal{X}$ $t$ describes. Note, however, that in this case the explanation can be simplified by condensing the two statements about $\text{lstat}$ into the single rule “\text{lstat} greater than 19.85.” Similarly, the right-most terminal node can be described with the single rule “if \text{rm} exceeds 7.437, then $\hat{f}(x)$ is 45.1,” despite the fact that it is at depth two.
More generally, because terminal nodes represent contiguous regions of $\mathcal{X}$, depth $d$ terminal nodes whose branches split on fewer than $d$ separate predictors can be interpreted as the intersection of fewer than $d$ rules. Put differently, sequential splits on the same variable are easily explained because they predict $y$ using a single dimension of $\mathcal{X}$. In the most extreme case, therefore, a node whose branch uses only a single variable corresponds to a contiguous region in $\mathcal{X}$ defined by a single dimension. This yields a single-rule explanation of the fitted value, regardless of the depth at which the node appears. Additionally, if these sequential splits uncover a monotonic relationship between the split points and fitted values, the explanation becomes easier still. In this sense, Breiman’s concept of “simple characterizations” of $\mathcal{X}$ can be understood in part by the extent to which a tree’s branches tend to reuse split variables.

1.3 Penalized Split Criteria for Interpretable Trees

1.3.1 Penalized Split Criteria

As we have seen, branches comprising small subsets of predictors are more interpretable than those containing new predictors at each split point. With this in mind, the criterion presented in this section encourages interpretable trees by penalizing splits that extend the set of features used in a given branch. Under this criterion the chosen split $s^*$ is not necessarily the one that most reduces impurity, which obviously worsens the extent to which the tree fits the data. Nevertheless, it is encouraging that the presence of a single split which minimizes impurity does not imply the absence of other suitable split options, even if minimizing impurity is the sole objective. Readers familiar with the literature will recall that the chosen split $s^*$ can be quite unstable, and that in reality many different splits may result in similar values of the gain function. In [Breiman et al. (1984)] the authors describe this phenomenon as
At any given node, there may be a number of splits on different variables, all of which give almost the same decrease in impurity. Since the data are noisy, the choice between competing splits is almost random.

As pointed out by many authors, the variability of CART splits is a drawback from a bias-variance perspective (see Breiman (1996a) and Hastie et al. (2009)). Here we focus on interpretability, and in the following sections we show how the presence of multiple splits with similar \( \phi \) values can actually be advantageous for growing interpretable trees.

The central idea is that if choosing a particular split rule from a set of competing rules with similar \( \phi \)'s is “almost random” as Breiman et al. (1984) asserts, then selecting the most interpretable one from the set rather than that which strictly maximizes the gain function should yield a tree that both fits the data and is easy to explain. To that end, given a non-negative penalty function \( \gamma \) for splitting \( t \) as per rule \( s \), we split according to

\[
  s^* = \arg \max_{s \in S} \{ \theta(t, s) - \gamma_k(t, s, \mathcal{B}_t) \},
\]

(1.6)

where the \( k \) refers to a penalization constant to be discussed shortly. As before, \( \mathcal{B}_t \) is the ordered list of split variables used in the branch of the tree leading to \( t \). The algorithm is still recursive but is now path dependent. Particular definitions of \( \gamma \) are the subject of Sections 1.3.2 and 1.3.3. Note that while penalizing the split criterion as in (1.6) is related to variable costs insofar as both methodologies can reduce the subset of variables a tree uses, variable costs must be specified by the user a priori. In contrast, the methodologies described herein are completely automatic.

The constant \( k \) is a tuning parameter that controls the tradeoff between the gain function and the penalty: high \( k \) values will correspond to a strong preference for interpretable splits, potentially at the gain function’s expense. Naturally, choosing
splits with less than the maximal gain can result in reduced fit in terms of $R^2$ or the misclassification rate. Nevertheless, as we shall see in the subsequent sections, in many cases the reduction is not drastic and could well be worth the improvement in interpretability. Of course the nature of the tradeoff varies with the dataset, and so it is advisable to run the algorithm for a variety of $k$ values. If we do not wish to use the tree for out-of-sample prediction this could very well be the end of the story – we simply choose the tree that yields the best combination of fit and interpretability for the problem at hand.

If a more systematic approach is desired, a natural procedure is to select the highest $k$ that results in a global fit no worse than that of the unpenalized tree’s by some predefined fraction. We define this formally as follows. Recalling that $\mathcal{L}$ denotes our learning sample of $N (y_i, x_i)$ pairs, we write $\Theta[f, \mathcal{L}]$ to denote tree $\hat{f}$’s loss evaluated on $\mathcal{L}$. At this point we only consider in-sample metrics (Section 1.4 discusses penalization’s out-of-sample performance), and so $\mathcal{L}$ serves as $\hat{f}$’s training data as well. In regression, for example, we take

$$\Theta[\hat{f}, \mathcal{L}] = \sum_{i \in \mathcal{L}} (y_i - \hat{f}(x_i))^2,$$

the usual squared-error loss. For convenience, in plots and tables we re-express this quantity as $R^2$ in order to remove the scale of $y$. In classification we let $\Theta$ be the misclassification rate (MR). Writing $\hat{f}_k$ indicate a tree grown with a particular tuning parameter, we choose the parameter $k^*$ as per

$$k^* = \max_k \left\{ k : \Theta[\hat{f}_k, \mathcal{L}] \leq (1 + c) \Theta[\hat{f}_0, \mathcal{L}] \right\},$$

where $c > 0$. That is, we choose the largest $k$ that still results in a tree whose loss is no worse than that of the unpenalized tree’s by $100c\%$. Unless noted otherwise, all
k’s for the penalized trees displayed in Sections 1.3.2 and 1.3.3 are chosen according to this procedure with \( c = 0.10 \).

Note that in regression \( \theta(t, s) \) is the decrease in mean squared error, which is dependent on the scale of the response variable. Penalizing MSE directly means the choice of \( k \) in (1.6) is dependent on the level of \( y \) in a given problem. To make \( k \) values comparable across datasets, in the sections below we re-express \( \theta \) to measure the proportional improvement in impurity gained by splitting \( t \) as per rule \( s \). The details of the scaling vary with the impurity function and are deferred to Appendix A.1.1, but in each case we ensure that \( \theta \leq 1 \) for all \( s \in S \), we prefer splits with larger \( \theta \), and we are indifferent between splitting and not splitting when \( \theta = 0 \). Herein we assume scaled gain functions, letting us restrict \( k \) to the interval \([0, 1]\).

### 1.3.2 New Variable Penalty

The first of our new methods is targeted at limiting the number of predictors used to reach a tree’s terminal nodes. As we have described, the more variables used to reach \( t \) the more complex the explanation of \( t \)’s subset of \( X \), and so in cases where many splits offer nearly the same \( \phi \) it may be preferable to choose a split on a variable already used in \( B_t \).

Letting \( s_x \in \{1, \ldots, p\} \) denote rule \( s \)’s split variable, the new variable penalty is written

\[
\gamma_k(t, s, B_t) = k \mathbf{1}(s_x \notin B_t). \tag{1.9}
\]

Hence if \( s \) introduces a new variable into the branch the penalty is \( k \). If \( s \) uses a previously used variable, there is no penalty. Thus splits that introduce new variables must improve \( \theta \) by at least \( k \) in order to be selected, whereas splits on old variables can be selected so long as the improvement is greater than 0. Whatever the split criterion, amongst many splits with similar \( \theta \)’s, using (1.9) gives preference to splits that do
not introduce new variables into the branch. This penalty (and more generally any penalized criterion written in the form of Equation 1.6) can be made compatible with any suitably scaled split criterion. In the following we demonstrate the performance of the penalty (1.9) on the previously used datasets for a selection of split criteria. In Figure 1.2 we compare trees grown to the Boston Housing data using (1.3), the conventional CART regression criterion, with and without the new variable penalty. First we note that despite the penalization, the $R^2$ values are comparable. The trees are equivalent up to the third level of splits, where the conventionally grown tree (Figure 1.2a) introduces $\text{crim}$ into the leftmost branch. All told, the unpenalized tree uses as many as five variables in reaching a terminal node, whereas the penalized tree (Figure 1.2b) never uses more than three. This makes a considerable difference when one attempts to explain the fit at a particular node. For instance, the region described by the bottom-left node of the penalized tree (for which $j(t) = 20.63$) might be described by saying “if $\text{rm}$ is between 5.85 and 6.54 and $\text{lstat}$ is between 9.66 and 14.4, the fitted value is 20.63.” Constructing an analogous description of the bottom-left node of the unpenalized tree is substantially more tedious.

Figure 1.2: CART applied to the Boston Housing data.

(a) Unpenalized

In-sample $R^2 = 0.8$

(b) New Variable Penalty ($k^* = 0.27$)

In-sample $R^2 = 0.79$
As per (1.8), 0.27 is the maximal value for $k$ that achieves a mean-squared error no more than 1.10 times that of the traditionally grown tree. Of course depending on how the analyst values fit versus interpretability, he can use higher values for $k$ resulting in even fewer variables used and a commensurate increase in in-sample MSE (decrease in $R^2$). For instance, Figure 1.3 uses $k = .4$, and largely describes the monotonic relationship between average home size and median prices.

Figure 1.3: CART fit to the Boston Housing data with the New Variable Penalty ($k^* = 0.4$). In-sample $R^2 = 0.67$.

The penalization framework applies to split criteria besides the usual CART methodology. As an example, we consider the one-sided high means criterion described in [Buja and Lee (2001)](Buja and Lee (2001)). Unpenalized, this method chooses the split $s$ that isolates the single child node with the highest mean:

$$s_{hm}^* = \arg \max_{s \in S} \left\{ \max_s \{ \bar{y}_L, \bar{y}_R \} \right\}. \quad (1.10)$$

An overview of the one-sided procedures introduced in [Buja and Lee (2001)](Buja and Lee (2001)) is contained in Appendix A.1.2. Applying this procedure to the Boston Housing data yields the left tree in Figure 1.4 with the penalized version appearing on the right. The $R^2$ values are comparable, but the penalized tree is considerably simpler as it involves only three predictors instead of six. The trees use only $\text{rm}$ and $\text{lstat}$ until the unpenalized tree splits on $\text{dis}$ at depth six. Further down, the unpenalized tree splits
on \textit{nox} and \textit{tax}, whereas the penalized tree uses only \textit{crim} and \textit{lstat}, leaving the monotonic relationships undisturbed.

Figure 1.4: One-sided High-Means fit to the Boston Housing data.

(a) Unpenalized

In-sample $R^2 = 0.79$

(b) New Variable Penalty ($k^* = 0.08$)

In-sample $R^2 = 0.78$

Turning to classification, Figure 1.5 combines the new variable penalty with \textit{Buja and Lee} (2001)'s one-sided purity criterion, which splits so as to isolate the single child node with minimum Gini (minimum classification impurity). Comparing the penalized tree in Figure 1.5b with its unpenalized counterpart in Figure 1.5a, we see that we can achieve less than 10% increase in the in-sample misclassification rate while reducing the total number of predictors used from seven to two. Here applying the new variable penalty allows us to uncover high-purity regions of $\mathcal{X}$ that are also relatively simple to interpret.

1.3.3 EMA-Style Penalty

Let us consider more closely the four leaf nodes at depth 6 in the penalized tree in Figure 1.2b (the leftmost of these leaf nodes has $j(t) = 20.63$). Using the new variable penalty allows us to see that the fits here depend on both \textit{rm} and \textit{lstat}. 
This represents an improvement over the corresponding branches in the unpenalized tree in Figure 1.2a that eventually split on dis, age and nox. Nevertheless, the fact that the predictors are interleaved makes constructing a more precise explanation difficult. Longer sequences of splits on the same variable would enable us to interpret the fits as monotonic relationships in rm and/or lstat, but here that is not possible. This should be no surprise – while (1.9) expresses our preference for using fewer variables, it is indifferent to the ordering of variables in a given branch.

Our second method targets both preferences. Here we penalize not only new variables, but also favor variables used recently in the branch. We achieve this by employing an exponential moving average-style (EMA) penalty, defined as:

$$
\gamma_k(t, s, \mathcal{A}_t) = \sum_{j=0}^{d-1} 1(s_x \neq s_j)k(1 - k)^{(d-1)-j} \quad \text{for } d > 0,
$$

and otherwise 0. As before, $k \in [0, 1]$ is the user-specified penalty constant and $s_x$ is the variable corresponding to the proposed split $s$. We let $j \in \{0, 1, \ldots\}$ index the
depth of $\mathcal{B}_t$’s nodes, and so $s_j$ is the split variable in $\mathcal{B}_t$ at depth $j$. The branches we last discussed from Figure 1.2 have $s_0=\text{rm}$, $s_1=\text{lstat}$ and $s_2=\text{rm}$, for instance. Here $d$ is the depth of the branch not including the proposed split, or equivalently, the number of nodes in $\mathcal{B}_t$. Hence when considering candidates for the second split in a branch we have $d = 1$. Obviously when considering the root split there should be no penalty (nor does (1.11) make sense), and so we set $\gamma = 0$.

Setting aside the notational details, we see that (1.11) is an exponential moving average of indicator functions. The $j$-th indicator is 1 if $s$’s split variable is different from the variable used at depth $j$. If $s$ splits on the same variable, as we prefer, the indicator is 0. Further, as $j \to 0$ we know $k(1-k)(d−1−j)$ decreases, and so the weights attenuate as we move up $\mathcal{B}_t$ towards the root. This conforms to our preferences: splitting a node on a different predictor from its parent is a graver offense than splitting on a different predictor from the root. Correspondingly, the former infraction contributes more to $\gamma$ than the latter. Lastly we note that setting $k = 0$ recovers the unpenalized version of the splitting criterion.

Figure 1.6 displays a regression tree grown using the CART procedure but with the EMA-style penalty. The unpenalized version of this tree appears in Figure 1.2a. Immediately we see that the new penalty eliminates the previously observed tendency for consecutive nodes to switch between splitting on $\text{rm}$ and $\text{lstat}$. The benefit is that the fit is easily explained primarily in terms of two monotonic relationships: for areas with very large homes ($\text{rm} > 6.94$) prices are monotonically increasing in home size, and for the remaining areas prices are decreasing in $\text{lstat}$. A very similar story emerges from using the EMA penalty with the high-means criterion, as displayed in Figure 1.7. In fact, some of the nodes in these trees characterize the exact same partition of $X$. 

16
Figure 1.6: CART fit to the Boston Housing data with the EMA-Style penalty \((k^* = .15)\). In-sample \(R^2 = 0.77\). In comparison with the unpenalized version in Figure 1.2, this tree uses only two predictors.

Figure 1.7: High-Means fit to the Boston Housing data with the EMA penalty \((k^*=.01)\). In-sample \(R^2 = 0.78\).

In Figure 1.8, we apply the EMA penalty to the Pima Indians data and Buja and Lee (2001)’s one-sided extremes criteria. This procedure chooses the split that results in the single child node with the highest sample proportion of a specified class. Here we search for regions of \(X\) associated with high incidence of diabetes. From previous examples we know that this dataset can withstand very high penalties before the misclassification rate breaks down. Hence in this example we set \(c\) to 0 and choose the penalization parameter whose associated tree’s misclassification rate is no higher than the unpenalized version’s. Notice that the trees have the same shape and misclassification rates, but the right tree uses only a single predictor. The unpenalized
tree, in comparison, never uses the same variable more than twice consecutively and employs seven predictors in all. Figure 1.9 displays the one-sided purity tree with and without the EMA penalty when $c = 0.10$.

Figure 1.8: One-Sided Extremes fit to the Pima Indians data with and without the EMA penalty. Figure 1.8b uses the EMA penalty with the highest penalty parameter such that the penalized tree’s misclassification is no higher than that of the unpenalized tree. Note the penalized tree uses only plasma, whereas the unpenalized tree uses 7 predictors.

(a) Unpenalized

In-sample MR = 0.25

(b) EMA Penalty ($k^* = 0.70$)

In-sample MR = 0.25
1.4 Out of Sample Performance

We have seen that one-sided split criteria and penalization often yield more interpretable trees than the traditional CART methodology with only modest sacrifices in in-sample loss, $\Theta$. Until now we have computed loss over our learning sample $\mathcal{L}$, but naturally it is important to understand how these techniques fare on new data, $z^{new} = (y^{new}, x^{new})$, as well. To that end, in this section we study the impact of the various techniques for growing $\hat{f}$ on the risk, defined by

$$R = \int_{z^{new}} \int_{\mathcal{L}} \Theta[\hat{f}_L, (y^{new}, x^{new})] \, dP(\mathcal{L}) \, dP(z^{new}). \quad (1.12)$$

We write $\hat{f}_L$ to emphasize that the fitted tree is a function of the training sample $\mathcal{L}$. In general our results suggest that applying an interpretability penalty to a given splitting criterion has very little impact on out-of-sample loss in comparison with the unpenalized criterion. This holds for both classification and regression problems over
a variety of splitting methods. In the remainder of this section we discuss these results in greater detail.

As we have neither true distribution functions nor an elegant form for the fitting procedure \( \mathcal{L} \to \hat{f}_L \) at our disposal, we study (1.12) using the “out-of-bag” generalization error estimate discussed in Breiman (1997). For each dataset we take \( B \) bootstrap samples \( L_1, \ldots, L_B \) from \( \mathcal{L} \). Let the bootstrap samples be indexed by \( b \in \{1, \ldots, B\} \). Observations in \( \mathcal{L} \) not in \( L_b \) are set aside as holdout data, \( H_b \). Using a tree fitting procedure \( F \) we fit a tree to each sample. Then for each tree we evaluate loss \( \Theta \) on its holdout data \( H_b \), yielding an estimate of generalization error \( \hat{\Theta}_b \). The procedure is given completely by Algorithm 1 in Appendix A.1.3. We then approximate \( R \) with the mean of the \( \hat{\Theta} \) values:

\[
R_{OOB} = \frac{1}{B} \sum_{b=1}^{B} \hat{\Theta}_b. \tag{1.13}
\]

In Algorithm 1 \( F \) represents the fitting procedure. Here we use both CART and the one-sided splitting criteria introduced in Buja and Lee (2001). One-sided splitting criteria are written

\[
\theta_{OS}(t, s) = \phi(t) - \min \{ \phi(t_L), \phi(t_R) \}, \tag{1.14}
\]

with \( s^* \in S \) still chosen by maximizing the gain function as in (1.1). In replacing (1.2)’s weighted sum over child nodes with minimization, (1.14) favors splits with low \( \phi \) on the left at the expense of high \( \phi \) on the right and vice versa, regardless of relative node size. Appendix A.1.2 describes how the high means and one-sided purity methodologies seen previously fit into this framework in addition to summarizing the remainder of the procedures described by Buja and Lee (2001).

Turning to the interpretability penalties, the reader will recall that we set the penalization constant \( k \) using (1.8). Roughly speaking this procedure aims to return
the most interpretable tree that still achieves a certain fraction of the unpenalized method’s performance on the data at hand. When we write “penalization method” or “penalization procedure” we mean a particular penalization function coupled with our rule for choosing \( k \). To study the penalties’ out-of-sample performance, we compute the out-of-bag error estimate as before but apply \((1.8)\) to each bootstrap learning sample. By this we mean that the \( F \) from Algorithm 1’s line \( T_b \leftarrow F(L_b) \) includes the search over possible \( k \) values. Hence \( \hat{\Theta} \) remains a metric of out-of-sample performance.

Starting with Table A.3 in Appendix A.1.4 we display the estimated loss obtained from applying each splitting criterion and penalty method combination (including no penalty) to our datasets. We set \( B = 100 \) and \( k = (0.01, 0.02, \ldots, 0.99) \). For the penalized methods, the column entitled “Average \( k^\star \)” reports the mean \( k \) value selected across the \( B \) bootstrap samples. Low average \( k^\star \) values suggest that on average, the splits chosen by the non-penalized methods have relatively few competitors in terms of reducing loss. The two wine datasets are examples of this – apparently in predicting wine quality, swapping the “best” split for a more interpretable one coincides with a substantial increase in MSE. In contrast, high \( k^\star \), such as those found on the “ankara dataset”, suggest that many predictors yield similar performance.

Generally, the results suggest that our method for choosing \( k^\star \) results in penalized trees whose risk remains quite close to that of the unpenalized methods. For example, Table A.3 shows that on our ten benchmark regression tasks, penalized CART’s estimated risk is always less than 10% higher than CART’s. In fact over all \( 2 \times 4 \times 10 = 80 \) possible penalty/criterion/dataset combinations in Tables A.3–A.6 only one has an increase in MSE above 10%. The evidence from classification is similar – in just one case does applying a penalty increase a splitting criterion’s holdout misclassification rate by more than 10%. In many cases misclassification rate decreases.

Moreover, the gains in interpretability can be substantial amounting to a “free lunch” of sorts. As an example consider the red wine dataset, where we wish to
predict each wine’s (ostensibly) human-labelled quality score using predictors that measure various aspects of the wine’s chemical composition. Figure 1.10 displays the unpenalized CART tree on the left and the EMA penalized tree on the right. We select $k^* = .07$ by our usual method with $c = 0.10$. The unpenalized tree uses as many six predictors in a branch, whereas the penalized tree uses only alcohol and sulphates in the entire tree. Moreover, the right tree’s fit is easily described as an increasing relationship between alcohol and quality for low values of alcohol and an increasing relationship between sulphates and quality for higher alcohol values. The EMA penalty’s out-of-bag risk estimate is only 1.8% higher than that of CART’s (see Table A.3), suggesting we can replace the CART fit with a far more interpretable tree that we can expect to perform essentially just as well on new data.

Figure 1.10: CART fit to the Red Wine data.

(a) Unpenalized

In-sample $R^2 = 0.36$

(b) EMA Penalty ($k^* = .07$)

In-sample $R^2 = 0.3$

1.5 Conclusion

This chapter describes penalization methods for growing classification and regression trees targeted at settings where interpreting the resultant tree is particularly important. These penalties directly encourage interpretability by controlling the size of the
subset of variables used in each branch. By requiring that less interpretable candidate splits decrease the parent node’s impurity more than others, penalization allows us to favor interpretability when many splits offer similar improvements. Interestingly, it is the tendency for many splits to offer very similar decreases in impurity – one of CART’s perceived disadvantages – that makes this possible.

Using real datasets we show that the penalty functions can indeed result in trees that are substantially easier to explain than their unpenalized counterparts. This observation holds for a variety of splitting criteria and across both classification and regression problems. Further, our study suggests that tuning a penalization parameter to maintain in-sample loss no more than a fraction $c$ of that of the unpenalized procedure’s results in generalization error that is almost always within $100c\%$ of the unpenalized method’s. That is, in nearly all cases the penalization techniques return a more interpretable fit for very little increase in out-of-sample loss, yielding a “free lunch” of sorts. This raises a number of interesting questions, such as why this might be the case, what $\mathcal{X}$ designs it is true for, or if further gains can be made by explicitly tuning penalty parameters to minimize holdout loss.

Acknowledgements

We thank Ed George and Abba Krieger for their insightful comments on this project. We also thank Richard Berk for his thorough reading and substantive edits on an earlier draft.
Software for Interpretable Classification and Regression Trees

Abstract

This chapter describes itree, an R package for fitting for classification and regression trees. Besides the familiar CART methodologies, the package implements splitting criteria and risk estimation techniques that aim to enhance a tree’s visual interpretability. For the procedures unique to itree we give a methodological overview, present illustrative examples, and demonstrate itree’s usage.

2.1 Introduction

Recursive partitioning trees are a popular supervised learning technique. Indeed, the rich variety of R packages implementing ideas that descend from Breiman et al. (1984) or Quinlan (1986) attests to the community’s continued interest in these methods. In this chapter we describe itree, an addition to R’s tree-fitting landscape that implements a variety of methods useful for growing interpretable and/or parsimonious trees. Much as the techniques described here represent extensions and modifications of the
usual CART methodology, the software extends and modifies \texttt{rpart}, the excellent package for classification and regression trees.

From a methodology perspective, the procedures unique to \texttt{itree} are naturally organized into three groups. The first set of techniques was first introduced in Buja and Lee (2001) and concern splitting criteria that induce imbalanced trees. Buja and Lee (2001) refers to them collectively as \textit{one-sided} procedures. The second group is based on the work of Goldstein and Buja (2013), which discusses ways to favor splits that restrict the subset of predictors used in a given branch of a tree. Goldstein and Buja (2013) achieves this by introducing penalties into the splitting criteria. As we will see, \texttt{itree} implements these penalties as to work with any valid splitting rule whether CART, one-sided, or even user-defined. The third concerns a technique first illustrated in Breiman (1997) for using the out-of-bag observations created by bagging to assess a tree’s local out-of-sample performance. Here we extend the ideas found in Breiman’s paper both by generalizing the methodology to classification and allowing the user to plot the metric alongside a tree’s fitted values. The result is a visual diagnostic for assessing a method’s out-of-sample performance over different regions of the feature space.

The chapter is organized as follows. In Section 2.2 we provide a brief CART overview and introduce some notational conventions. Sections 2.3, 2.4 and 2.5 address the one-sided criteria, penalization, and local risk procedures in turn. In each section we demonstrate the package’s use with code snippets alongside the methodology discussion. Section 2.6 contains concluding remarks.

### 2.2 Overview of Classification and Regression Trees

Where possible we follow the notation used in Breiman (1996b). For convenience the conventions introduced in this section are summarized in Table 2.1. Readers will
recall that given a learning sample $\mathcal{L}$ of $N$ pairs $(y_i, x_i)$ from an arbitrary distribution, the algorithms described in Breiman et al. (1984) output a recursively grown binary tree $\hat{f}(x)$. Here $\hat{f}$ is called a classification or regression tree and the loss function $\Theta$ is taken to be the misclassification rate or mean-squared-error (MSE, herein) depending on whether $y$ is categorical or continuous, respectively.

For any $x$, $\hat{f}(x)$ is given by the mean (for regression) or most common (in classification) $y_i$ value over all $i \in \mathcal{L}$ that are in the same terminal node as $x$, denoted $t(x)$. In either case, all observations in a given node $t$ share a single fitted value, which we denote $j(t)$ herein.

To each non-terminal node, a splitting rule $s$ is applied, defined by a pair $(x, t)$ consisting of a variable $x$ and a split location $t$ on that variable. The rule $s = (x_1, 0)$, for instance, divides the $n_t$ observations in $t$ into two subsets, depending on whether each $x$ vector has a positive first coordinate. We call $s_x = x_1$ the split variable and 0 the split point. The growing phase consists of selecting the best split for the observations in $t$ and then sending them to the appropriate child nodes, where the recursion begins anew.\footnote{To lessen overfitting, it is common practice to set a minimum node size and/or prune the tree by collapsing child nodes.} Proposing new meanings of “best split” is the subject of sections 2.3 and 2.4.

CART determines the “best $s$” by the goodness of split criterion or gain function $\theta(t, s)$ which quantifies the benefit of splitting node $t$ as per rule $s$. Each node splits at

$$s^* = \arg \max_{s \in S} \theta(t, s),$$

meaning we choose the split that maximizes the split criterion, where $S$ is the set of all possible splits including no split. $\theta$ is of the form

$$\theta(t, s) = \phi(t) - \frac{n_{t_L}}{n_t} \phi(t_L) + \frac{n_{t_R}}{n_t} \phi(t_R),$$
where $t_L$ and $t_R$ are the left and right child nodes defined by $s$, and $\phi$ is the loss or so-called impurity function. By multiplying $\phi_L$ and $\phi_R$ by the proportion of observations in the left and right child nodes, $\theta(t, s)$ measures the average improvement in impurity from splitting $t$ as per rule $s$.

As mentioned above, in regression settings we wish to minimize MSE and so we set

$$\phi_R(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - j(t))^2$$  \hspace{1cm} (2.3)

Recall that in regression we set $j(t)$ to the sample mean of the in-node $y$ values, and so readers will quickly identify (2.3) as $t$'s (biased) sample variance, $\hat{\sigma}^2(t)$. This is the impurity function \texttt{rpart} uses when one sets \texttt{method="anova"}.

For reasons that are well documented, when growing trees for classification or class probability estimation we do not take $\phi$ to be the weighted average misclassification error (see Breiman et al. (1984) or Buja and Lee (2001), for instance). Rather, we use either the Gini or Cross-entropy criterion. For the $K$-class problem with $y \in \{1, 2, \ldots, K\}$ the Gini criterion is written

$$\phi_G(t) = \sum_{k \in K} \hat{p}_{k,t}(1 - \hat{p}_{k,t})$$  \hspace{1cm} (2.4)

where $\hat{p}_{k,t}$ is the proportion of $y_i$'s in node $t$'s that are of class $k$. Cross-entropy is defined by

$$\phi_{CE}(t) = \sum_{k \in K} \hat{p}_{k,t} \log(\hat{p}_{k,t}).$$  \hspace{1cm} (2.5)

Breiman (1996b) notes that empirically, Gini tends to yield splits resulting in purer nodes, especially when $K > 2$. In their informative description of the \texttt{R} package \texttt{rpart}, Therneau and Atkinson (1997) comment that from a practical perspective there is usually little difference between the methods, especially when $K = 2$. The \texttt{rpart} package implements both criteria, but defaults to Gini. Both \texttt{itree} and this chapter
do likewise; when referring to the conventional method of growing classification trees we assume Gini impurity as defined in (2.4).

Table 2.1: Summary of Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}$</td>
<td>Learning sample of $N (y_i, x_i)$ pairs</td>
</tr>
<tr>
<td>$\hat{f}$</td>
<td>Recursive partitioning tree grown using training data</td>
</tr>
<tr>
<td>$\hat{f}(x)$</td>
<td>Tree $\hat{f}$’s fitted value at $x$</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>Loss function (MSE or misclassification rate)</td>
</tr>
<tr>
<td>$t(x)$</td>
<td>The terminal node to which observation $x$ belongs</td>
</tr>
<tr>
<td>$j(t)$</td>
<td>The fitted value associated with node $t$</td>
</tr>
<tr>
<td>$t_L, t_R$</td>
<td>Node $t$’s left and right child nodes if $t$ is non-terminal</td>
</tr>
<tr>
<td>$n_t$</td>
<td>Number of training observations in node $t$</td>
</tr>
<tr>
<td>$s$</td>
<td>Splitting rule consisting of a (split variable, split point) pair</td>
</tr>
<tr>
<td>$s_x$</td>
<td>Split variable associated with splitting rule $s$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Goodness of split criterion / gain function</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Impurity function (see above for the relationship between $\theta$ and $\phi$)</td>
</tr>
<tr>
<td>$\hat{p}_{k,t}$</td>
<td>Proportion of $y_i$’s in node $t$ that are of class $k$</td>
</tr>
</tbody>
</table>

2.3 One-Sided Gain Functions

Clearly (2.2) enforces a measure of balance between the impurity in right and left child nodes - candidate splits with low impurity on the left at the expense of high impurity on the right will likely be passed over for splits with more even performance. Put differently, (2.2) represents a compromise between $\phi_L$ and $\phi_R$. Insofar as we care about $\hat{f}$’s performance over the entire $X$ space, as is typically the case, balance is a virtue.

An alternative approach to growing binary trees is described in Buja and Lee (2001). Instead of balancing a statistic $\phi$ over the left and right nodes, these methods split at the rule yielding the lowest value of $\phi$ in either child node. These one-sided
splitting criteria can be written

\[ \theta_{OS}(t, s) = \phi(t) - \min \{\phi(t_L), \phi(t_R)\}, \]  

(2.6)

with \( s^* \in S \) still chosen by maximizing the gain function as in \((2.1)\). In replacing \((2.2)'s weighted sum over child nodes with minimization, \((2.6)\) favors splits with low \( \phi \) on the left at the expense of high \( \phi \) on the right, regardless of relative node size. It is because they ignore the \( \phi \) value in one of the child nodes that Buja and Lee (2001) refer to these methods as “one-sided.”

### 2.3.1 One-Sided Purity

Recursively splitting as per \((2.6)\) amounts to a greedy search for partitions of \( X \) associated with low \( \phi \). If \( \phi \) is a loss metric, for instance, the procedure finds partitions of \( X \) in which \( f(x) \) is particularly well approximated by a binary tree. More simply, this corresponds to finding \( x \) vectors whose associated \( y \)'s are close together. To that end, Buja and Lee (2001) proposes setting \( \phi \) to be the conventional CART impurity functions. For regression this gives us

\[ \phi_{osp,R}(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - \hat{j}(t))^2 = \hat{\sigma}^2(t), \]  

(2.7)

and combined with \((2.6)\) this yields

\[ \theta_{osp,R}(t) = \hat{\sigma}^2(t) - \min \{\hat{\sigma}^2(t_L), \hat{\sigma}^2(t_R)\}. \]  

(2.8)

Recalling that we split at the rule that maximizes \( \theta \), it is clear that \( s^* \) is the split that finds the single child node with the lowest average squared-error loss.

In \texttt{itree} we fit a one-sided purity tree by setting the \texttt{method} argument appropri-
ately. As an example, we use \texttt{itree} to fit both the conventional CART tree and the one-sided purity tree to the well-known Boston Housing data, where the goal is to fit median home prices in a census tract using a variety of predictors regarding the homes’ average physical characteristics and locations. The code is as follows.

\begin{verbatim}
> library(itree)
> bh.cart <- itree(medv~.,data=bh,minbucket=25,minsplit=25,cp=0)
> bh.purity <- itree(medv~.,data=bh,method="purity",
+                 minbucket=25,minsplit=25,cp=0)
\end{verbatim}

Note that in the first line \texttt{itree} figures out from \texttt{medv} that this is a regression problem and chooses \texttt{method="anova"} for the CART tree implicitly. More generally, readers familiar with \texttt{rpart} will note that the syntax and arguments are exactly the same, save the fact that \texttt{itree} accepts \texttt{"purity"} as a valid method. In fact, in cases where one enters a valid \texttt{rpart} command, \texttt{itree} gives the same results. Thus many scripts written for \texttt{rpart} can be modified to use \texttt{itree} by swapping occurrences of \texttt{"rpart"} for \texttt{"itree"} and adjusting the arguments. The call creating \texttt{bh.cart} would run identically in \texttt{rpart} and give the same tree, for example.

The methods outlined in this chapter focus on tree-growing rather than pruning, and so our convention is to cease splitting once \( n_t \) reaches 5\% of \( n \). This is controlled using the \texttt{minsplit} and \texttt{minbucket} arguments, which function identically to \texttt{rpart}'s. The Boston Housing data has 506 observations, hence we set \texttt{minbucket=minsplit=25} above. To be clear, the 5\% convention is not the default functioning of \texttt{itree}, hence leaving these arguments out of the commands above would result in trees with smaller terminal nodes. As an aside, we remark that the \texttt{cp} argument (identical to \texttt{rpart}'s) also allows for premature stopping of the tree-growing phase. See Therneau and Atkinson (1997) for details. Its functioning is considerably more complex than the node size arguments, and thus we deactivate it using \texttt{cp=0} for this article's examples.
Figure 2.1 displays the result of plotting the returned objects using the usual `plot(...)` command. Each subfigure's caption shows the applicable model's in-sample $R^2$. Note that plotting follows `rpart`, and so the call `plot(bh.cart)` plots `bh.cart`'s skeleton and `text(bh.cart)` then labels the nodes by printing splits and fitted values where appropriate.

Figure 2.1: CART and One-Sided Purity applied to the Boston Housing Data. Terminal nodes are restricted to contain no fewer than 5% of all observations.

(a) CART

In-sample $R^2 = 0.8$

(b) One-Sided Purity

In-sample $R^2 = 0.79$

For one-sided trees, branches highlighted in blue correspond to the node generating minimum $\phi$. This is done using the `highlight.color` argument as shown below.

```r
> plot(bh.purity, highlight.color="BLUE")
```

Setting `highlight.color="BLACK"` or `highlight.color="RED"` turns off highlighting and highlights the branch in red, respectively.

Turning to the trees themselves, we see that despite their similar $R^2$ values, these fits suggest very different explanations of what makes homes particularly expensive or cheap. Immediately we see that the root splits, known to be the most stable, are different. Whereas the conventionally grown tree in Figure 2.1a splits first on `rm`, the
average number of rooms, \( r_m \) only enters the one-sided tree in Figure 2.1b at depth 6. The one-sided tree’s root split uses \( p_t \), the parent-teacher-ratio, which does not enter the CART tree at all. Obviously the variables used to greedily minimize CART impurity and those used to find regions of purity are quite different.

More generally, splitting as per (2.6) yields greater variation in the depth of terminal nodes, giving the one-sided tree an unbalanced look. The reason for this is clear – should we find a bucket with high purity, it is both likely to be small and unlikely to be split again. If our goal is to understand the subsets of \( X \) with high purity in \( y \) this is ideal – fewer splits yield simpler, more intuitive explanations.

In classification the analogous function one-sided purity function is

\[
\phi_{osp,C}(t) = \sum_{k \in K} \hat{p}_{k,t}(1 - \hat{p}_{k,t}).
\]  

Buja and Lee (2001) only considers the two-class case in which (2.9) simplifies to \( \hat{p}_0 \hat{p}_1 \). As itree extends Buja and Lee (2001) to the multi-class problem, we leave the function as written in (2.9) to make the generalization to the multi-class case obvious – we just compute the Gini criterion at each node as before. Here again, substituting (2.9) into the one-sided split criterion shows that \( s^* \) is the split which identifies the child node with minimum Gini impurity.

As an example, Figure 2.2 displays a one-sided purity tree along with the usual CART tree on the Pima Indians dataset. In this problem \( y \in \{\text{pos}, \text{neg}\} \) depending on whether the individual has diabetes and \( x \in \mathbb{R}^8 \). Once again, we stop splitting once \( n_t \) is 5% of the overall \( n \). Note that for illustrative purposes we set method="class_purity", in which case the software does not need to make an educated guess as to whether the user intends to fit a classification or regression tree. In this case the diabetes variable is of class \texttt{factor}, and so simply passing

\footnote{Readers should note that this is one possible generalization to the multi-class purity problem; certainly others exist.}
method="purity" would give the same result.

```r
> pima.cart <- itree(diabetes~., pima, minsplit=38, minbucket=38, cp=0)
> pima.purity <- itree(diabetes~., pima, method="class_purity",
+ minsplit=38, minbucket=38, cp=0)
```

Figure 2.2: CART and One-Sided Purity applied to the Pima Indians data. (MR = Misclassification Rate)

(a) CART

In-sample MR=0.20

(b) One-Sided Purity

In-sample MR=0.21

As with the regression trees we see similar performance in terms of the misclassification rate but sharply different trees. Some readers may point out that the one-sided purity tree’s performance is aided by the fact that it has more terminal nodes. While this is true, recall that both trees were grown using the same minsplit and minbucket parameters. Hence the larger number of terminal nodes in the purity tree is evidence that a finer recursive partition of \( X \) is possible with the one-sided purity criterion than with CART. For example, whereas the CART tree never splits on the same variable in succession, the one-sided tree splits repeatedly on plasma. The repeated splits on plasma levels uncover a monotonic relationship that CART misses.
Turning to the multi-class case, we demonstrate the one-sided purity criterion using data regarding autism diagnoses. Here the response variable is a physician’s diagnosis of a proband’s location on the autism spectrum. Specifically we have $y \in \{\text{autism, pdd, aspergers}\}$. The predictors comprise phenotype variables and a categorical variable $\text{site}$ corresponding to the hospital where the proband was observed.

```r
> aut.cart <- itree(diagnosis~.,aut,minsplit=138,minbucket=138,cp=0)
> aut.purity <- itree(diagnosis~.,aut,method="purity",
+     minsplit=138,minbucket=138,cp=0)
```

Figure 2.3 shows the CART and one-sided purity trees. Since about 70% of cases have $y = \text{autism}$, the 25% misclassification rate is only modestly better than we could achieve by simply guessing the most common class. What is more interesting is the depth and shape of the one-sided tree in comparison with the CART tree. Following the blue branches, we see that one-sided purity sequentially splits off low-impurity
buckets from the main group, whereas CART’s focus on reducing overall impurity leads it to cease splitting relatively quickly.

In Figure 2.4 we restrict the purity tree’s depth by setting maxdepth=3, giving us a magnified view of the top splits from the one-sided purity tree in Figure 2.3b. As in rpart we print counts in the leaf nodes by passing use.n=T to the text function. Here the power of one-sided splitting criteria is readily apparent – the purity tree immediately identifies sites in which all probands are diagnosed as having autism. Whether these hospitals are subject to a different distribution of probands or use drastically different diagnostic standards is unclear, but the tree shows plainly that the joint distribution of \((y, x)\) varies significantly across sites. Taking the root split as an example, by ignoring the relative impurity in the right child node, one-sided purity enables us to find a perfectly pure bucket on the left; in contrast, CART discards this split in its search for balance.

In Figure 2.5 we demonstrate one-sided purity on a classification problem with 10 classes. In this problem we are given an image of a handwritten digit and wish to predict which digit the writer intended. Specifically, we have \(y \in \{0, 1, \ldots, 9\}\) with \(x \in \mathcal{R}^{63}\) corresponding to 63 features taken from a bitmap image of the character. Though the increase in misclassification is more substantial, the one-sided purity tree
immediately finds a bucket with low impurity – the rightmost split has $\phi_{osp,C}(t) = .33$
in comparison with $\phi_{osp,C} = .90$ at the root. In contrast, two of three CART tree leaf
nodes at depth two have impurity greater than 0.4.

```r
> digit.cart <- itree(factor(Digit)~.,digit.rec, 
+ msplit=192,minbucket=192,cp=0)
> digit.purity <- itree(factor(Digit)~.,digit.rec,method="purity", 
+ msplit=192,minbucket=192,cp=0)
```

Figure 2.5: CART and One-Sided Purity applied to the Digit Recognition data.

2.3.2 One-Sided Extremes

In a severe departure from traditional methodologies, Buja and Lee (2001)’s one-sided
extremes abandons goodness of fit altogether and simply finds $x$ vectors whose associ-
ated $y$’s are particularly high, low, or predominately a specified class. In comparison
with CART, not only do we replace 2.2 with the one-sided criterion, but also we strip
$\phi$ of its connection to loss or impurity. Here $\phi$ is not representative of the fit’s quality,
but rather of the fitted values themselves.
When \( y \) is continuous a prominent characteristic of fitted values is their level; that is, whether a child node’s mean is particularly high or low in comparison with its parent’s. In this vein Buja and Lee (2001) defines the high means criterion

\[
\phi_{ose,hm}(t) = -j(t).
\] (2.10)

Recalling that \( j(t) = \bar{y}_t \), we can substitute this into the one-sided gain function 2.6 as follows

\[
\theta_{hm}(t, s) = -j(t) - \min\{-j(L), -j(R)\}
\]

\[
= \max\{\bar{y}_L, \bar{y}_R\} - \bar{y}_t.
\]

Hence \( s^* \) returns the split that identifies the child node with the highest mean. Buja and Lee (2001) defines the related low means criterion in the obvious way: \( \phi_{ose,lm}(t) = j(t) \). In this case \( s^* \) returns the split with the lowest child node mean.

In itree we grow one-sided extremes trees by setting method="extremes". For regression problems there remains the question of whether we want to look for high or low means, which we specify by setting parms=1 or parms=-1 respectively. High means is the default. As an example Figure 2.6 shows the low-means tree alongside the CART fit for the Boston Housing data.

\[
> bh.low.mean <- itree(medv~., bh, method="extremes", parms=-1,
+ minbucket=25, msplit=25, cp=0)
\]

Here again, we get very different trees. Whereas CART splits immediately on \( rm \), the low means tree splits on \( crim \). The message is that it simply does not matter how large the home is if the location is problematic. The low means tree also uncovers two monotone associations: first in \( lstat \), the neighborhood’s proportion of lower status people, and then in \( rm \). First we split off a series of higher \( lstat \) regions to find \( x \).
vectors associated with lower home values. Once $\text{lstat} < 10.14$ it seems that house size is of primary importance. Arguably most people who have bought or sold a home will find the interpretation offered by the right tree more in conformance with their intuition than that of the left tree.

As categorical $y$ values lack a natural ordering, to find extremes in a classification setting we must specify a class of interest $k'$. Buja and Lee (2001) then chooses $s^*$ to be the split yielding the single child node with the highest $\hat{p}_{k'}$. The statistic of interest at each node is $\hat{p}_{k'}$ and so in the presence of 2.2’s minimization we set

$$\phi_{\text{usc},C} = 1 - \hat{p}_{k'}$$

(2.11)
interest as the number of classes expands.

The digit recognition problem illustrates this point well. Imagine we are particularly interested in understanding which $x$ vectors are associated with a certain handwritten digit, say “2”. For this purpose the CART or one-sided purity trees from Figure 2.5 are of limited use, as leaf nodes with $j(t) = “2”$ do not appear in the trees until depth four and five, respectively. This is not a surprise insofar as these procedures’ impurity functions depend on Gini, which weighs all classes equally.

This situation suggests fitting a one-sided extremes classification tree with $k’ = “2”$, which we can do as follows.

```r
> digit.extremes <- itree(factor(Digit)~.,digit.rec,
+                      minsplit=192,minbucket=192,cp=0,method="extremes",
+                      parms=list(classOfInterest="2"))
```

The resulting tree is displayed in Figure 2.7. We see that it immediately isolates nodes with $j(t) = “2”$ at depths one and two. Note that for classification, the `parms` argument must be given as a list.

Figure 2.7: CART and One-Sided Extremes applied to the Digit Recognition data.

(a) CART

In-sample MR=0.30

(b) One-Sided Extremes ($k’ = “2”$)

In-sample MR=0.69
2.4 Penalized Split Criteria for Interpretable Trees

We turn now to `itree`'s functions for restricting the subset of variables used in tree construction. As explained in Goldstein and Buja (2013), these methods work by penalizing splits that complicate a tree's structure. Goldstein and Buja (2013) contains a more detailed explication of these procedures and their performance that is beyond the scope of this chapter. Rather, we give a brief overview of the motivation and mechanics behind these techniques and present some illustrative examples using `itree`.

Readers familiar with CART will recall that the chosen split \( s^* \) can be quite unstable, and that in reality many different splits may result in similar values of the gain function. Breiman et al. (1984) describes this variability in a tree’s structure as follows.

\[
\text{At any given node, there may be a number of splits on different variables, all of which give almost the same decrease in impurity. Since the data are noisy, the choice between competing splits is almost random.}
\]

In essence, Goldstein and Buja (2013)'s procedures helps choose parsimonious splits in cases where many similarly performing split options are available. From the perspective of Goldstein and Buja (2013), parsimony is a function of how many variables are used to reach a given node. As an illustration, consider the toy dataset below.

```r
> set.seed(271)
> x1 <- (1:100)/100
> x2 <- x1+.05*(runif(100) -.5)
> Y <- (x1>.25) + (x1>.5) +(x1>.5)*(x2>.75) +rnorm(100,sd=.2)
```

Here \( y = f(x_1, x_2) + \epsilon \) where \( f \) is a binary tree. Further, \( x_2 \) is a corrupted version of \( x_1 \) where the corruption is quite low – the correlation between these variables is
greater than 0.99. In Figure 2.8 we plot two CART trees fit to the data; the left uses both $x_1$ and $x_2$ whereas the right uses only $x_1$.

Figure 2.8: Parsimony Example.

(a) Tree 1
In-sample $R^2=0.98$

(b) Tree 2
In-sample $R^2=0.97$

Imagine now that we are given this data without explicit variable definitions and must choose between the trees presented in Figure 2.8. Which is preferable? In-sample they have nearly identical fitted values and so $R^2$ is of little help. Further, knowing $\hat{\rho}(x_1, x_2) > 0.99$, we would expect them to give essentially the same predictions on future data. Clearly, the salient difference is that Tree 1’s rightmost branch includes two variables whereas the analogous branch of Tree 1 continues to split on $x_1$. That is, Tree 1 hypothesizes an interaction effect whereas Tree 2 hypothesizes a monotone association. The latter not only provides a simpler mathematical representation of the data, but also a simpler interpretation of the fitted values.

CART’s greedy optimization yields the less desirable Tree 1 despite the fact that the two are effectively the same. Ideally we would like to trade parsimony against in-sample loss in a more sensible manner. Goldstein and Buja (2013) addresses this with the penalized split criterion, defined as follows. Given a non-negative penalty
function $\gamma$ for splitting $t$ as per rule $s$, we split according to

$$s^* = \arg \max_{s \in S} \{ \theta(t, s) - \gamma_k(t, s, \mathcal{B}_t) \}. \quad (2.12)$$

Here $\mathcal{B}_t$ is the ordered list of split variables used in the branch of the tree leading to $t$. The rightmost child node in Figure 2.8's right hand tree has $\mathcal{B}_t = \{x_1, x_2\}$, for instance. Its parent has $\mathcal{B}_t = \{x_1\}$. We write $\gamma_k$ to indicate that $\gamma$ takes a user-specified tuning parameter, $k$.

Goldstein and Buja (2013) uses two penalty functions, both of which are implemented in itree. The first simply penalizes the entry of a new variable into a branch. Write $s_x \in \{x_1, \ldots, x_p\}$ to denote rule $s$'s split variable. Then the new variable penalty is given by

$$\gamma_k(t, s, \mathcal{B}_t, k) = k 1(s_x \in \mathcal{B}_t). \quad (2.13)$$

Hence if $s$ introduces a new variable into the branch the penalty is $k$. If $s$ reuses a previously used variable, there is no penalty. Higher values of $k$ correspond to larger penalties for introducing new variables. Selecting an appropriate value for the tuning parameter $k$ is obviously of importance. In the interest of brevity we omit a detailed discussion, but note that in what follows we choose the maximum $k$ such that our loss is within 10% of the unpenalized in-sample loss. We refer interested readers to Goldstein and Buja (2013), which suggests that penalized trees grown using such rules perform quite similarly to their unpenalized counterparts out-of-sample.

In itree we grow trees with the new variable penalty via the argument penalty="newvar" with $k$ passed using interp_param1=k. We take the Boston Housing data as an example.

```r
> bh.cart.nvp <- itree(medv~., data=bh, minbucket=25, minsplit=25, cp=0,
+ penalty="newvar", interp_param1=.1)
> digit.ext.nvp <- itree(factor(Digit)~., digit.rec, method="extremes",
+ penalty="newvar", interp_param1=.1)
```
Figure 2.9: CART applied to the Boston Housing data with the New Variable Penalty. In-sample $R^2=0.8$.

The trees in Figures 2.9 and 2.10 use five and two variables respectively in comparison with their unpenalized counterparts which use six and five. The new variable penalty does not directly limit the total number of variables but rather the number of variables in a particular branch, and so this is achieved as a side effect. More directly, the penalized CART tree uses more than three variables in a branch compared with as many as five for the unpenalized version. Similarly the extremes tree uses two variables in its single branch versus up to six without the penalty. In the case of the CART tree, this comes at no price in terms of reduced $R^2$.

Note that in the function call we specify method and penalty separately. Much as equation (2.12) uses $\theta$ and $\gamma$ to generalize the methodology to arbitrary impurity and penalty functions, itree implements impurity and penalty procedures independently in C (as opposed to once for each criterion-penalty pair) with aggregation left until the end. This allows the user to mix and match impurity and penalty procedures in any way he sees fit.

The second criterion penalizes variables in accordance with how recently they were...
used in a particular branch. The idea is that sequences of splits on the same variable suggest a monotonic relationship, which is certainly a less complex hypothesis than a series of interaction terms. Goldstein and Buja (2013) implements this idea using an exponential moving average-style penalty, defined as

\[ \gamma_k(t, s, B_t) = \sum_{j=0}^{d-1} 1(s_d \neq s_j)k(1 - k)^{(d-1)-j} \text{ for } d > 0, \quad (2.14) \]

and otherwise 0. Here \( d \) is the depth of the branch not including the proposed split, or equivalently, the number of nodes in \( B_t \). Say we are considering splitting a node at depth 3. Then we have \( d = 3 \). If the variable under consideration is different from that used to split the parent node \((j = 2)\), the penalty increases by \( k(1 - k)^0 = k \). If it is different from the grandparent node \((j = 1)\), the penalty increases by \( k(1 - k)^1 \), and so on. Recalling that \( 0 \leq k \leq 1 \), we see that the penalty for the proposed split variable \( s_d \) being different from its ancestor \( s_j \) decreases as the nodes become further apart.

In \texttt{itree} we fit a tree using this penalty by setting \texttt{penalty="ema"}. As before we specify \( k \) using \texttt{interp_param1}. 

44
> pima.purity.ema <- itree(diabetes~., pima, minsplit=38, minbucket=38, cp=0, + method="purity", penalty="ema", interp_param1=.03)

Figure 2.11 plots the resulting tree alongside the unpenalized version.

Figure 2.11: One-Sided Purity applied to the Pima Indians data with and without the EMA penalty.

(a) Unpenalized
In-sample MR=0.21
(b) EMA Penalty
In-sample MR=0.23

2.5 Local Risk Estimation

2.5.1 A Bootstrap Procedure for Local Risk Estimation

Until now we have assessed performance by averaging loss over all observations in our trading data $\mathcal{L}$. Naturally it is of interest to determine how a given procedure fares on new data, $z^{new} = (y^{new}, x^{new})$, as well. Moreover, in most datasets it is reasonable to expect this performance to depend on where $x^{new}$ is located in $\mathcal{X}$. To that end, this section discusses tools for estimating and visualizing local risk estimates.

We begin by making this notion precise. For a given subset $X \in \mathcal{X}$, we define a
procedure’s X-conditional local risk to be

\[
R_X = \mathbb{E} \left[ \Theta(\hat{f}_L, (y^{\text{new}}, x^{\text{new}})) | x^{\text{new}} \in X \right]
\]

(2.15)

\[
= \int \int \Theta(\hat{f}_L, (y^{\text{new}}, x^{\text{new}})) \mathbf{1}(x^{\text{new}} \subset X) \mathbb{P}(dL)\mathbb{P}(dz^{\text{new}}) = \int \mathbf{1}(x^{\text{new}} \subset X)\mathbb{P}(dz^{\text{new}})
\]

We write \( \hat{f}_L \) to emphasize that the fitted tree is a function of the training sample \( L \). Note that while the outer integral is over a particular subset, the inner integral in (2.15) is unconstrained with respect to training sets \( L \). Thus \( R_X \) reflects the performance of a procedure fit to all \( X \) but evaluated “locally” with respect to \( X \).

One can see that \( R_X \) measures the risk of the procedure rather than that of an individual tree by observing that \( \hat{f} \) varies with \( L \). Now clearly the equation makes sense for any evaluation region \( X \subset X \), but given an actual tree, ostensibly our interest in \( R_X \) peaks when \( X \) corresponds to one of our tree’s terminal nodes. In such cases, knowing \( R_X \) means we can use the tree to display not only how \( y \) varies with \( x \) but also how the method’s risk varies with \( x \).

We do not know \( R_X \), and so we employ the bootstrap and Breiman (1997)’s out-of-bag device to generate estimates. Breiman (1997) only considers the regression case where \( \Theta \) is MSE. In the following we generalize Breiman (1997)’s procedure to arbitrary splitting methods and loss functions.

We begin by using the bootstrap’s “out-of-bag” observations to form holdout estimates of \( \Theta \), for each observation \( i \in \{1 \ldots N\} \). For example, \( \hat{\Theta}_1 \) is the average loss incurred in predicting \( y_1 \) with \( \hat{y}_{1,b} \) where \( b \) varies over all trees not using the first observation as a training example. Then for any leaf node \( t \) belonging to a tree \( T \) we estimate \( R_t \) using
That is, we estimate a node’s risk using the average out-of-bag risk estimates for observations appearing in the node. The details of this procedure are given by Algorithm 2.

### 2.5.2 Implementation in itree

The `itree` package implements Algorithm 2 and extends `rpart`’s plotting capabilities to allow the user to display a node’s risk estimate alongside its fitted value. In this case the tree serves as both a fit for $y$ as well as a visual diagnostic of the procedure’s performance in various partitions of $\mathcal{X}$. Note that using `itree` we can generate and plot these $R_{oob,t}$ values for trees grown using any splitting criterion and penalty combination we wish across both regression and classification tasks.

As an example, we consider local risk estimates for the typical CART methodology applied to the Boston Housing data.

```r
> bh.cart <- itree(medv~.,bh,minsplit=25,minbucket=25,cp=0)
> theta_hats <- getOOBLoss(model_tree.obj=bh.cart,data=bh,nboot=100)
> lre <- estNodeRisk(tree.obj=bh.cart,
+ est_observation_loss=theta_hats$avgOOBloss)
```

In the first line we fit a tree to the whole training sample, as we have seen before. Next, we use `getOOBLoss` to compute the average out-of-bag loss for each observation. That is, $\hat{\Theta}_i = \theta_hats[i]$. Then for each leaf node in the original tree `estNodeRisk` estimates local risk by the mean in-node $\hat{\Theta}_i$ found in the second line. In Figure 2.12 we plot the original tree displaying both the fitted value and local risk estimate at each node.
Figure 2.12 shows that out-of-sample mean squared error varies considerably across partitions of $\mathcal{X}$. For example, the observations in the rightmost node (with predicted value of 45.1) have an average out-of-sample MSE of about 51, whereas the leftmost node’s observations (predicted value of 10.56) value is only 14.2. Observing the splits leading to the rightmost node, the tree suggests that areas with high $\text{rm}$ values (those with large homes) have large MSEs in addition to high average prices. Further investigation reveals that 11 of the 16 observations with truncated response values (in this dataset the response is censored at 50) are in the rightmost node. Possibly CART’s poor performance in this region has more to do with the high incidence of inaccurately recorded data points rather than CART’s inability to learn the appropriate structure.
For classification problems the process and syntax is exactly the same, but instead we compute and report the average misclassification rate at each node (that is, \( \Theta \) is the misclassification rate). Figure 2.13 shows the commands and plot when the procedure is applied to a CART tree for the Pima Indians dataset.

Figure 2.13: Pima Indians Data: Local Risk Estimate for CART

```r
> pima.cart <- itree(diabetes~., pima, minsplit=38, minbucket=38, cp=0)
> theta_hats <- getOOBLoss(pima.cart, pima, 100)
> lre <- estNodeRisk(pima.cart, theta_hats$avgOOBloss)
> plot(pima.cart, do_node_re= TRUE, uniform=TRUE)
> text(pima.cart, est_node_risk = lre, use.n=TRUE)
```

Figure 2.13 shows that the estimated out-of-sample misclassification rate ranges from approximately 0.10 to 0.56, and so here, too, we see substantial variability in risk over different regions of \( \mathcal{X} \). Note that the second line in each terminal node displays the in-sample count of \texttt{neg/pos} cases, from which we can compute in-sample misclassification rates. Comparing this with the risk estimate, we see that the left-most node’s 0.085 in-sample misclassification rate \( \frac{23}{23+248} \approx 0.085 \) is close
to the honest out-of-bag estimate of 0.10. In contrast, the bottom most leaf node with misclassification rate equal to $11/(11 + 27) \approx 0.29$ has an estimated out-of-bag misclassification rate of 0.558, far higher than the counts imply. Using the local risk estimates, we know that the first node’s low in-sample misclassification rate is much less a product of chance capitalization than the second node’s.

Users should note that the functions described in this section are entirely new to \texttt{itree} and thus not documented in sources pertaining to \texttt{rpart}. As such, we provide some comments and usage notes before proceeding to the concluding remarks. The function \texttt{getOOBLoss} generates \texttt{mboot} bootstrap samples and runs \texttt{itree()} on each using the formula, method, and penalty found in the \texttt{model_tree.obj} argument. It returns a matrix of out-of-sample predicted values as well as the vector of losses, \texttt{avgOOBloss}. In \texttt{getOOBLoss} it is possible to override \texttt{model_tree.obj}’s \texttt{minsplit} and \texttt{minbucket} arguments as well as define one’s own function for generating bootstrap or cross-validation samples. The current versions of \texttt{getOOBLoss} and \texttt{estNodeRisk} do not support cost-weighted classification problems or unequally weighted observations. We expect subsequent version of \texttt{itree} will incorporate this functionality.

In the \texttt{plot} command, setting \texttt{do_node_re} to \texttt{TRUE} tells the \texttt{plot.itree} function to allow appropriate space for printing local risk estimates. This argument defaults to \texttt{FALSE}, which corresponds to the usual spacing. In the corresponding \texttt{text} command, we pass information about $\hat{\Theta}$ by setting the \texttt{est_node_risk} argument. This tells the graphics device what numbers to print where. Similarly this argument’s default is \texttt{NULL}, in which case nothing is printed.

More generally, we remark that using the same tree growing methodology $F$ to both fit $T$ and compute the $\hat{\Theta}_i$’s is no way a requirement. Similarly, it is not strictly necessary that $T$ and $\Theta_i$ use the same training datasets; this simply results in a display where $j(t)$ and $R_{oob,t}$ are estimated using different observations. Despite this flexibility, it is certainly more intuitive to hold both $F$ and $\mathcal{L}$ fixed as we have done
here. As we have seen, in this case the final plot serves as both a fit for $y$ as well as a visual diagnostic of the procedure's performance in various partitions of $\mathcal{X}$.

### 2.6 Conclusion

This chapter demonstrates `itree`, an R package that implements various ideas for growing classification and regression trees. We focus on the three aspects of `itree` that differentiate it from other similar R packages: one-sided splitting criteria, penalized splitting criteria and local risk estimation.

Finally, we wish to emphasize that this software is based on the code of `rpart`. As such, `itree` inherits much of what `rpart` does with little or no modifications. This includes snipping and pruning of trees, prediction, and cross validation of the cp parameter amongst other features. When doing cross-validation with penalties, each run uses the same penalization function and $k$ value as passed to the `itree` call. Note however that `cp` is not defined for one-sided methods, and so here `itree$cptable=NULL` and calling the cross-validation routine returns an error. Otherwise `itree`'s behaviour is essentially the same as `rpart`'s. Hence resources such as Therneau and Atkinson (1997) and Therneau et al. (2012) remain useful references.

### Acknowledgements

We thank Richard Berk, Ed George and Abba Krieger for their helpful comments and suggestions. We also wish to thank Terry Therneau for his suggestions regarding the details of releasing the package.
Abstract

This article presents Individual Conditional Expectation (ICE) plots, a tool for visualizing the model estimated by any supervised learning algorithm. Classical partial dependence plots (PDPs) help visualize the average partial relationship between the predicted response and one or more features. In the presence of substantial interaction effects, the partial response relationship can be heterogeneous. Thus, an average curve, such as the PDP, can obfuscate the complexity of the modeled relationship. Accordingly, ICE plots refine the partial dependence plot by graphing the functional relationship between the predicted response and the feature for individual observations. Specifically, ICE plots highlight the variation in the fitted values across the range of a covariate, suggesting where and to what extent heterogeneities might exist.

In addition to providing a plotting suite for exploratory analysis, we include a visual test for additive structure in the data generating model. Through simulated examples and real data sets, we demonstrate how ICE plots can shed light on estimated models.

*This chapter is joint work with Adam Kapelner, Justin Bleich and Emil Pitkin.
in ways PDPs cannot. Procedures outlined are available in the R package ICEbox.

3.1 Introduction

The goal of this article is to present Individual Conditional Expectation (ICE) plots, a toolbox for visualizing models produced by “black box” algorithms. These algorithms use training data \( \{x_i, y_i\}_{i=1}^{N} \) (where \( x_i = (x_{i,1}, \ldots, x_{i,p}) \) is a vector of predictors and \( y_i \) is the response) to construct a model \( \hat{f} \) that maps the features \( x \) to fitted values \( \hat{f}(x) \). Though these algorithms can produce fitted values that enjoy low generalization error, it is often difficult to understand how the resultant \( \hat{f} \) uses \( x \) to generate predictions. The ICE toolbox helps visualize this mapping.

ICE plots extend Friedman (2001)’s Partial Dependence Plot (PDP), which highlights the average partial relationship between a set of predictors and the predicted response. ICE plots disaggregate this average by displaying the estimated functional relationship for each observation. Plotting a curve for each observation helps identify interactions in \( \hat{f} \) as well as extrapolations in predictor space.

The paper proceeds as follows. Section 3.2 gives background on visualization in machine learning and introduces PDPs more formally. Section 3.3 describes the procedure for generating ICE plots and its associated plots. In Section 3.4 simulated data examples illustrate that ICE plots can be used to identify features of \( \hat{f} \) that are not visible in PDPs, or where the PDPs may even be misleading. Each example is chosen to illustrate a particular principle. Section 3.5 provides examples of ICE plots on real data. In Section 3.6 we shift the focus from the fitted \( \hat{f} \) to a data generating process \( f \) and use ICE plots as part of a visual test for additivity in \( f \). Section 3.7 concludes.
3.2 Background

3.2.1 Survey of Black Box Visualization

There is an extensive literature that attests to the superiority of black box machine learning algorithms in minimizing predictive error, both from a theoretical and an applied perspective. Breiman (2001b), summarizing, states “accuracy generally requires more complex prediction methods ...[and] simple and interpretable functions do not make the most accurate predictors.” Problematically, black box models offer little in the way of interpretability, unless the data is of very low dimension. When we are willing to compromise interpretability for improved predictive accuracy, any window into black box’s internals can be beneficial.

Authors have devised a variety of algorithm-specific techniques targeted at improving the interpretability of a particular statistical learning procedure’s output. Rao and Potts (1997) offers a technique for visualizing the decision boundary produced by bagging decision trees. Although applicable to high dimensional settings, their work primarily focuses on the low dimensional case of two covariates. Tzeng (2005) develops visualization of the layers of neural networks to understand dependencies between the inputs and model outputs and yields insight into classification uncertainty. Jakulin et al. (2005) improves the interpretability of support vector machines by using a device called “nomograms” which provide graphical representation of the contribution of variables to the model fit. Pre-specified interaction effects of interest can be displayed in the nomograms as well. Breiman (2001a) uses randomization of out-of-bag observations to compute a variable importance metric for Random Forests (RF). Those variables for which predictive performance degrades the most vis-a-vis the original model are considered the strongest contributors to forecasting accuracy. This method is also applicable to stochastic gradient boosting (Friedman, 2002).
et al. (2000) plots neural network predictions in a scatterplot for each variable by sampling points from covariate space. Amongst the existing literature, this work is the most similar to ICE, but was only applied to neural networks and does not have a readily available implementation.

Other visualization proposals are model agnostic and can be applied to a host of supervised learning procedures. For instance, Strumbelj and Kononenko (2011) consider a game-theoretic approach to assess the contributions of different features to predictions that relies on an efficient approximation of the Shapley value. Jiang and Owen (2002) use quasi-regression estimation of black box functions. Here, the function is expanded into an orthonormal basis of coefficients which are approximated via Monte Carlo simulation. These estimated coefficients can then be used to determine which covariates influence the function and whether any interactions exist.

3.2.2 Friedman’s PDP

Another particularly useful model agnostic tool is Friedman (2001)’s PDP, which this paper extends. The PDP plots the change in the average predicted value as specified feature(s) vary over their marginal distribution. Many supervised learning models applied across a number of disciplines have been better understood thanks to PDPs. Green and Kern (2010) use PDPs to understand the relationship between predictors and the conditional average treatment effect for a voter mobilization experiment, with the predictions being made by Bayesian Additive Regression Trees (BART, Chipman et al., 2010). Berk and Bleich (2013) demonstrate the advantage of using RF and the associated PDPs to accurately model predictor-response relationships under asymmetric classification costs that often arise in criminal justice settings. In the ecological literature, Elith et al. (2008), who rely on stochastic gradient boosting, use PDPs to understand how different environmental factors influence the distribution of
a particular freshwater eel.

To formally define the PDP, let $S \subset \{1, ..., p\}$ and let $C$ be the complement set of $S$. Here $S$ and $C$ index subsets of predictors; for example, if $S = \{1, 2, 3\}$, then $x_S$ refers to a $3 \times 1$ vector containing the values of the first three coordinates of $x$. Then the partial dependence function of $f$ on $x_S$ is given by

$$f_S = \mathbb{E}_{x_C} [f(x_S, x_C)] = \int f(x_S, x_C) dP(x_C) \quad (3.1)$$

where $dP(x_C)$ is the marginal distribution of $x_C$. Each subset of predictors $S$ has its own partial dependence function $f_S$, which gives the average value of $f$ when $x_S$ is fixed and $x_C$ varies over its marginal distribution $dP(x_C)$. As neither the true $f$ nor $dP(x_C)$ are known, we estimate Equation $3.1$ by computing

$$\hat{f}_S = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(x_S, x_{Ci}) \quad (3.2)$$

where $\{x_{C1}, ..., x_{CN}\}$ represent the different values of $x_C$ that are observed in the training data. Note that the approximation here is twofold: we estimate the true model with $\hat{f}$, the output of a statistical learning algorithm, and we estimate the integral over $x_C$ by averaging over the $N x_C$ values observed in the training set.

This is a visualization tool in the following sense: if $\hat{f}_S$ is evaluated at the $x_S$ observed in the data, a set of $N$ ordered pairs will result: $\{(x_{S\ell}, \hat{f}_{S\ell})\}_{\ell=1}^{N}$, where $\hat{f}_{S\ell}$ refers to the estimated partial dependence function evaluated at the $\ell$th coordinate of $x_S$, denoted $x_{S\ell}$. Then for one or two dimensional $x_S$, Friedman (2001) proposes plotting the $N x_{S\ell}$'s versus their associated $\hat{f}_{S\ell}$'s, conventionally joined by lines. The resulting graphic, which is called a partial dependence plot, displays the average value of $\hat{f}$ as a function of $x_S$. For the remainder of the paper we consider a single predictor.
of interest at a time ($|S| = 1$) and write $x_S$ without boldface accordingly.

As an extended example, consider the following data generating process with a simple interaction:

$$Y = 0.2X_1 - 5X_2 + 10X_21_{X_3 \geq 0} + \mathcal{E},$$

$$\mathcal{E} \overset{iid}{\sim} \mathcal{N}(0, 1), \quad X_1, X_2, X_3 \overset{iid}{\sim} U(-1, 1).$$

We generate 1,000 observations from this model and fit a stochastic gradient boosting model (SGB) via the R package gbm [Ridgeway 2013] where the number of trees is chosen via cross-validation and the interaction depth is set to 3. We now consider the association between predicted $Y$ values and $X_2$ ($S = \{2\}$). In Figure 3.1a we plot $X_2$ versus $Y$ in our sample. Figure 3.1b displays the fitted model’s partial dependence plot for predictor $X_2$. The PDP suggests that on average, $X_2$ is not meaningfully associated with the predicted $Y$. In light of Figure 3.1a, this conclusion is plainly wrong. Clearly $X_2$ is associated with $Y$; it is simply that the averaging inherent in the PDP shields this discovery from view.

In fact, the original work introducing PDPs argues that the PDP can be a useful summary for the chosen subset of variables if their dependence on the remaining features is not too strong. When the dependence is strong, however – that is, when interactions are present – the PDP can be misleading. Nor is the PDP particularly effective at revealing extrapolations in $X$-space. ICE plots are intended to address these issues.
Figure 3.1: Scatterplot and PDP of $X_2$ versus $Y$ for a sample of size 1000 from the process described in Equation 3.3. In this example $\hat{f}$ is fit using SGB. The PDP incorrectly suggests that there is no meaningful relationship between $X_2$ and the predicted $Y$.

### 3.3 The ICE Toolbox

#### 3.3.1 The ICE Procedure

Visually, ICE plots disaggregate the output of classical PDPs. Rather than plot the target covariates’ *average* partial effect on the predicted response, we instead plot the $N$ estimated conditional expectation curves: each reflects the predicted response as a function of covariate $x_S$, conditional on an observed $x_C$.

Consider the observations $\{(x_{Si}, x_{Ci})\}_{i=1}^{N}$, and the estimated response function $\hat{f}$. For each of the $N$ observed and fixed values of $x_C$, a curve $\hat{f}_{S}^{(i)}$ is plotted against the observed values of $x_S$. Therefore, at each x-coordinate, $x_S$ is fixed and the $x_C$ varies across $N$ observations. Each curve defines the conditional relationship between $x_S$ and $\hat{f}$ at fixed values of $x_C$. Thus, the ICE algorithm gives the user insight into the several variants of conditional relationships estimated by the black box.
The ICE algorithm is given in Algorithm 3 in Appendix A.3.1. Note that the PDP curve is the average of the $N$ ICE curves and can thus be viewed as a form of post-processing. Although in this paper we focus on the case where $|S| = 1$, the pseudocode is general. All plots in this paper are produced using the R package ICEbox (Goldstein et al., 2013), available on CRAN.

Returning to the simulated data described by Equation 3.3, Figure 3.2 shows the ICE plot for the SGB when $S = \{2\}$. In contrast to the PDP in Figure 3.1b, the ICE plot makes it clear that the fitted values are related to $X_2$. Specifically, the SGB’s predicted values are approximately linearly increasing or decreasing in $X_2$ depending upon which region of $\mathcal{X}$ an observation is in.

![Figure 3.2: SGB ICE plot for $X_2$ from 1000 realizations of the data generating process described by Equation 3.3. We see that the SGB’s fitted values are either approximately linearly increasing or decreasing in $X_2$.](image)

Now consider the well known Boston Housing Data (BHD). The goal in this dataset is to predict a census tract’s median home price using features of the census tract itself. It is important to note that the median home prices for the tracts are truncated at 50, and hence one may observe potential ceiling effects when analyzing the data. We use Random Forests (RF) implemented in R (Liaw and Wiener, 2002) to
fit $\hat{f}$. The ICE plot in Figure 3.3 examines the association between the average age of homes in a census tract and the corresponding median home value for that tract ($S = \text{age}$). The PDP is largely flat, perhaps displaying a slight decrease in predicted median home price as age increases. The ICE plot shows those observations for which increasing age is actually associated with higher predicted values, thereby describing how individual behavior departs from the average behavior.

Figure 3.3: RF ICE plot for BHD for predictor age. The highlighted thick line is the PDP. For each curve, the location of its observed age is marked by a point. For some observations, higher age is associated with a higher predicted values. The upper set of tick marks on the horizontal axis indicate the observed deciles of age.

### 3.3.2 The Centered ICE Plot

When the curves have a wide range of intercepts and are consequently “stacked” on each other, heterogeneity in the model can be difficult to discern. In Figure 3.3, for example, the variation in effects between curves and cumulative effects are veiled. In such cases the “centered ICE” plot (the “c-ICE”), which removes level effects, is useful.

c-ICE works as follows. Choose a location $x^*$ in the range of $x_S$ and join or “pinch” all prediction lines at that point. We have found that choosing $x^*$ as the minimum or
the maximum observed value results in the most interpretable plots. For each curve \( \hat{f}^{(i)} \) in the ICE plot, the corresponding c-ICE curve is given by

\[
\hat{f}_{\text{cent}}^{(i)} = \hat{f}^{(i)} - \mathbf{1}\hat{f}(x^*, \mathbf{x}_{C_i}),
\]

where the unadorned \( \hat{f} \) denotes the fitted model and \( \mathbf{1} \) is a vector of 1’s of the appropriate dimension. By subtracting \( \hat{f}(x^*, \mathbf{x}_{C_i}) \) from \( \hat{f}^{(i)} \), we ensure that all c-ICE curves are zero at \( x^* \). Hence the point \((x^*, \hat{f}(x^*, \mathbf{x}_{C_i}))\) acts as a “base case” for the \( i \)-th curve. If \( x^* \) is the minimum value of \( x_S \), for example, this ensures that all curves originate at 0, thus removing the differences in level due to the different \( \mathbf{x}_{C_i} \)’s. At the maximum \( x_S \) value, each centered curve’s level reflects the cumulative effect of \( x_S \) on \( \hat{f} \) relative to the base case. The result is a plot that better isolates the combined effect of \( x_S \) on \( \hat{f} \), holding \( \mathbf{x}_C \) fixed.

Figure 3.4 shows a c-ICE plot for the predictor \( \text{age} \) of the BHD for the same RF model as examined previously. From the c-ICE plot we can now see clearly that the cumulative effect of \( \text{age} \) on predicted median value increases for some cases, and decreases for others. Such divergences of the centered curves suggest the existence of interactions between \( x_S \) and \( \mathbf{x}_C \) in the model. Also, the magnitude of the effect, as a fraction of the range of \( y \), can be seen in the vertical axis displayed on the right of the graph.

3.3.3 The Derivative ICE Plot

To further explore the presence of interaction effects, we develop plots of the partial derivative of \( \hat{f} \) with respect to \( x_S \). To illustrate, consider the scenario in which \( x_S \) does not interact with the other predictors in the fitted model. This implies \( \hat{f} \) can be written as
Figure 3.4: c-ICE plot for age with \( x^* \) set to the minimum value of age. The right vertical axis displays changes in \( \hat{f} \) over the baseline as a fraction of \( y \)'s observed range. In this example, interactions between age and other predictors create cumulative differences in fitted values of up to about 14% of the range of \( y \).

\[
\hat{f}(x) = \hat{f}(xS, xC) = g(xS) + h(xC), \quad \text{so that} \quad \frac{\partial \hat{f}(x)}{\partial xS} = g'(xS),
\]

meaning the relationship between \( xS \) and \( \hat{f} \) does not depend on \( xC \). Thus the ICE plot for \( xS \) would display a set of \( N \) curves that share a single common shape but differ by level shifts according to the values of \( xC \).

As it can be difficult to visually assess derivatives from ICE plots, it is useful to plot an estimate of the partial derivative directly. The details of this procedure are given in Algorithm 4 in Appendix A.3.1. We call this a “derivative ICE” plot, or “d-ICE.” When no interactions are present in the fitted model, all curves in the d-ICE plot are equivalent, and the plot shows a single line. When interactions do exist, the derivative lines will be heterogeneous.

As an example, consider the d-ICE plot for the RF model in Figure 3.5. The plot
suggests that when age is below approximately 60, \( g' \approx 0 \) for all observed values of \( x_C \). In contrast, when age is above 60 there are observations for which \( g' > 0 \) and others for which \( g' < 0 \), suggesting an interaction between age and the other predictors. Also, the standard deviation of the partial derivatives at each point, plotted in the lower panel, serves as a useful summary to highlight regions of heterogeneity in the estimated derivatives (i.e., potential evidence of interactions in the fitted model).

![Figure 3.5: d-ICE plot for age in the BHD. The left vertical axis' scale gives the partial derivative of the fitted model. Below the d-ICE plot we plot the standard deviation of the derivative estimates at each value of age. The scale for this standard deviation plot is on the bottom of the right vertical axis.](image)

### 3.3.4 Visualizing a Second Feature

Color allows ICE, c-ICE and d-ICE plots to convey information regarding a second predictor of interest \( x_k \). Specifically, one can assess how the second predictor influences the relationship between \( x_S \) and \( \hat{f} \). If \( x_k \) is categorical, we assign colors to its levels and plot each prediction line \( \hat{f}^{(i)} \) in the color of \( x_{ik} \)'s level. If \( x_k \) is continuous, we vary the color shade from light (low \( x_k \)) to dark (high \( x_k \)).
We replot the c-ICE from Figure 3.4 with lines colored by a newly constructed predictor, \( x = 1(\text{rm} > \text{median(\text{rm})}) \). Lines are colored red if the average number of rooms in a census tract is greater than the median number of rooms across all census tracts and are colored blue otherwise. Figure 3.6 suggests that for census tracts with a larger number of average rooms, predicted median home price value is positively associated with \text{age} and for census tracts with a lesser number of average rooms, the association is negative.

Figure 3.6: The c-ICE plot for \text{age} of Figure 3.4 in the BHD. Red lines correspond to observations with \text{rm} greater than the median \text{rm} and blue lines correspond to those with fewer.

### 3.4 Simulations

Each of the following examples is designed to emphasize a particular model characteristic that the ICE toolbox can detect. The examples are purposely stylized to more clearly demonstrate given scenarios with minimal interference from issues that one typically encounters in actual data, such as noise and model misspecification.
3.4.1 Additivity Assessment

We begin by showing that ICE plots can be used as a diagnostic in evaluating the extent to which a fitted model \( \hat{f} \) fits an additive model.

Consider again the prediction task in which \( \hat{f}(x) = g(x_S) + h(x_C) \). For arbitrary vectors \( x_{C_i} \) and \( x_{C_j} \), \( \hat{f}(x_S, x_{C_i}) - \hat{f}(x_S, x_{C_j}) = h(x_{C_i}) - h(x_{C_j}) \) for all values of \( x_S \). The term \( h(x_{C_i}) - h(x_{C_j}) \) represents the shift in level due to the difference between \( x_{C_i} \) and \( x_{C_j} \) and is independent of the value of \( x_S \). Thus the ICE plot for \( x_S \) will display a set of \( N \) curves that share a common shape but differ by level shifts according to the unique values of \( x_C \).

As an illustration, consider the following additive data generating model

\[
Y = X_1^2 + X_2 + \epsilon, \quad X_1, X_2 \sim \text{iid } \mathcal{U}(-1, 1), \quad \epsilon \sim \mathcal{N}(0, 1).
\]

We simulate 1000 independent \( (X_i, Y_i) \) pairs according to the above and fit a generalized additive model (\( \text{GAM, Hastie and Tibshirani 1986} \)) via the \( \text{R} \) package \text{gam} (\text{Hastie, 2013}). As we have specified it, the \( \text{GAM} \) assumes

\[
f(X) = f_1(X_1) + f_2(X_2) + f_3(X_1 X_2)
\]

where \( f_1, f_2 \) and \( f_3 \) are unknown functions estimated internally by the procedure using smoothing splines. Because \( f_3 \) appears in the model specification but not in the data generating process, any interaction effects that \( \text{GAM} \) fits are spurious.\(^2\) Here, ICE plots inform us of the degree to which interactions were fit. Were there no interaction in \( \hat{f} \) between \( X_1 \) and \( X_2 \), the ICE plots for \( X_1 \) would display a set of curves equivalent

\(^2\)If we were to eliminate \( f_3 \) from the \( \text{GAM} \) then we would know a priori that \( \hat{f} \) would not display interaction effects.
in shape but differing in level.

Figure 3.7a displays the ICE plots for $X_1$ and indicates that this is indeed the case: all curves display a similar parabolic relationship between $\hat{f}$ and $X_1$, shifted by a constant, and independent of the value of $X_2$. Accordingly, the associated d-ICE plot in Figure 3.7b displays little variation between curves. The ICE suite makes it apparent that $f_3$ (correctly) contributes relatively little to the GAM model fit. Note that additive structure cannot be observed from the PDP alone in this example (or any other).

![ICE and d-ICE plots](image)

Figure 3.7: ICE and d-ICE plots for $S = X_1$ when $\hat{f}$ is a GAM with possible interaction effects between $X_1$ and $X_2$. So as to keep the plot uncluttered we plot only a fraction of all 1000 curves. In the ICE plots the dots indicate the actual location of $X_1$ for each curve.

### 3.4.2 Finding interactions and regions of interactions

As noted in [Friedman (2001)](#), the PDP is most instructive when there are no interactions between $x_S$ and the other features. In the presence of interaction effects, the averaging procedure in the PDP can obscure any heterogeneity in $\hat{f}$. Let us return
to the simple interaction model

\[ Y = 0.2X_1 - 5X_2 + 10X_2 \mathbb{I}_{X_3 \geq 0} + \mathcal{E}, \]  

(3.6)

\[ \mathcal{E} \sim \mathcal{N}(0, 1), \quad X_1, X_2, X_3 \sim \mathcal{U}(-1, 1) \]

to examine the relationship between SGB’s \( \hat{f} \) and \( X_3 \). Figure 3.8a displays an ICE plot for \( X_3 \). Similar to the PDP we saw in Section 3.1, the plot suggests that averaged over \( X_1 \) and \( X_2 \), \( \hat{f} \) is not associated with \( X_3 \). By following the non-parallel ICE curves, however, it is clear that \( X_3 \) modulates the fitted value through interactions with \( X_1 \) and \( X_2 \).

Where in the range of \( X_3 \) do these interactions occur? The d-ICE plot of Figure 3.8b shows that interactions are in a neighborhood around \( X_3 \approx 0 \). This is expected; in the model given by Equation 3.6, being above or below \( X_3 = 0 \) changes the response level. The plot suggests that the fitted model’s interactions are concentrated in \( X_3 \in [-0.025, 0.025] \) which we call the “region of interaction”.

Generally, regions of interaction are identified by noting where the derivative lines are variable. In our example, the lines have highly variable derivatives (both positive and negative) in \([-0.025, 0.025]\). The more heterogeneity in these derivative lines, the larger the effect of the interaction between \( x_S \) and \( x_C \) on the model fit. Regions of interaction can be seen most easily by plotting the standard deviation of the derivative lines at each \( x_S \) value. In this example, the standard deviation function is plotted in the bottom pane of Figure 3.8b and demonstrates that fitted interactions peak at \( X_3 \approx 0 \).
3.4.3 Extrapolation Detection

As the number of predictors $p$ increases, the sample vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are increasingly sparse in the feature space $\mathcal{X}$. A consequence of this curse of dimensionality is that for many $\mathbf{x} \in \mathcal{X}$, $\hat{f}(\mathbf{x})$ represents an extrapolation rather than an interpolation (see Hastie et al., 2009 for a more complete discussion).

Extrapolation may be of particular concern when using a black-box algorithm to forecast $\mathbf{x}_{\text{new}}$. Not only may $\hat{f}(\mathbf{x}_{\text{new}})$ be an extrapolation of the $(\mathbf{x}, y)$ relationship observed in the training data, but the black-box nature of $\hat{f}$ precludes us from gaining any insight into what the extrapolation might look like. Fortunately, ICE plots can cast light into these extrapolations.

Recall that each curve in the ICE plot includes the fitted value $\hat{f}(x_{si}, \mathbf{x}_{ci})$ where $x_{si}$ is actually observed in the training data for the $i$th observation. The other points on this curve represent extrapolations in $\mathcal{X}$. Marking each curve in the ICE plot at the observed point helps us assess the presence and nature of $\hat{f}$’s hypothesized extrapolations in $\mathcal{X}$.

Consider the following model:
\[ Y = 10X_1^2 + 1_{X_2 \geq 0} + \mathcal{E}, \quad (3.7) \]

\[ \mathcal{E} \overset{iid}{\sim} \mathcal{N}(0, .1^2), \quad \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \begin{cases} U(-1, 0), \ U(-1, 0) & \text{w.p. } \frac{1}{3} \\ U(0, 1), \ U(-1, 0) & \text{w.p. } \frac{1}{3} \\ U(-1, 0), \ U(0, 1) & \text{w.p. } \frac{1}{3}. \end{cases} \]

Notice \( \mathbb{P}(X_1 > 0, X_2 > 0) = 0 \), leaving the quadrant \([0, 1] \times [0, 1]\) empty. We simulate 1000 observations and fit a RF model to the data. The ICE plot for \( x_1 \) is displayed in Figure 3.9a with the points corresponding to the 1000 observed \((x_1, x_2)\) values marked by dots. We highlight observations with \( x_2 < 0 \) in red and those with \( x_2 \geq 0 \) in blue. The two subsets are plotted separately in Figures 3.9b and 3.9c.

The absence on the blue curves of points where both \( x_1, x_2 > 0 \) confirms that \( \mathbb{P}(X_1 > 0, X_2 > 0) = 0 \). From Figure 3.9c we see that in this region of \( \mathcal{X} \), \( \hat{f} \) increases roughly in proportion with \( x_1^2 \) even though no data exists. Ostensibly the RF model has extrapolated the polynomial relationship from the observed \( \mathcal{X} \)-space to where both \( x_1 > 0 \) and \( x_2 > 0 \).

Whether it is desirable for \( \hat{f} \) to display such behavior in unknown regions of \( \mathcal{X} \) depends on the character of the extrapolations in conjunction with the application at hand. Moreover, different algorithms will likely give different extrapolations. Examining the ICE plots can reveal the nature of these extrapolations and guide the user to a suitable choice.

### 3.5 Real Data

We now demonstrate the ICE toolbox on three real data examples. We emphasize features of \( \hat{f} \) that might otherwise have been overlooked.
(a) All observations  (b) Observations with $x_2 < 0$  (c) Observations with $x_2 \geq 0$

Figure 3.9: ICE plots for $S = x_1$ of a RF model fit to Equation 3.7. The left plot shows the ICE plot for the entire dataset where $x_2 < 0$ is colored red and $x_2 \geq 0$ in blue. The middle plot shows only the red curves and the right only the blue. Recall that there is no training data in the quadrant $[0, 1] \times [0, 1]$, and so Figure 3.9c contains no points for observed values when $x_1 > 0$ (when both $x_1$ and $x_2$ are positive). Nevertheless, from Figure 3.9c’s ICE curves it is apparent that the fitted values are increasing in $x_1$ for values above 0. Here, the ICE plot elucidates the existence and nature of the RF’s extrapolation outside the observed $X$-space.
3.5.1 Depression Clinical Trial

The first dataset comes from a depression clinical trial [DeRubeis et al., 2014]. The response variable is the Hamilton Depression Rating Scale (a common composite score of symptoms of depression where lower scores correspond to being less depressed) after 15 weeks of treatment. The treatments are placebo, cognitive therapy (a type of one-on-one counseling), and paroxetine (an anti-depressant medication). The study also collected 37 covariates which are demographic (e.g. age, gender, income) or related to the medical history of the subject (e.g. prior medications and whether the subject was previously treated). For this illustration, we drop the placebo subjects to focus on the 156 subjects who received either of the two active treatments.

The goal of the analysis in [DeRubeis et al., 2014] is to understand how different subjects respond to different treatments, conditional on their personal covariates. The difference between the two active treatments, assuming the classic linear (and additive) model for treatment, was found to be statistically insignificant. If the clinician believes that the treatment effect is heterogeneous and the relationship between the covariates and response is complex, then flexible nonparametric models could be an attractive exploratory tool.

Using the ICE toolbox, one can visualize the impact of the treatment variable on an \( \hat{f} \) given by a black-box algorithm. Note that extrapolations in the treatment indicator (i.e. predicting at 0 for an observed 1 or vice versa) correspond to counterfactuals in a clinical setting, allowing the researcher to see how the same patient might have responded to a different treatment.

We first modeled the response as a function of the 37 covariates as well as treatment to obtain the best fit of the functional relationship using the black-box algorithm BART (implemented by Kapelner and Bleich, 2014) and obtained an in-sample \( R^2 \approx 0.40 \).

Figure 3.10a displays an ICE plot of the binary treatment variable, with cognitive
therapy coded as “0” and paroxetine coded as “1”, colored by marital status (blue if married and red if unmarried). The plot shows a flat PDP, demonstrating no relationship between the predicted response and treatment when averaging over the effects of other covariates. However, the crossing of ICE curves indicates the presence of interactions in \( \hat{f} \), which is confirmed by the c-ICE plot in Figure 3.10b. After centering, it becomes clear that the flat PDP obscures a complex relationship: the model predicts between -3 and +3 points on the Hamilton scale, which is a highly clinically significant range (and almost 20% of the observed response’s range). Further, we can see that BART fits an interaction between treatment and marital status: married subjects are generally predicted to do better on cognitive therapy and unmarried subjects are predicted to do better with paroxetine.

![Figure 3.10: ICE plots of a BART model for the effect of treatment on depression score after 15 weeks. Married subjects are colored in blue and unmarried subjects are colored in red.](a) ICE  (b) c-ICE)
3.5.2 White Wine

The second data set concerns 5,000 white wines produced in the *vinto verde* region of Portugal obtained from the UCI repository ([Bache and Lichman 2013](#)). The response variable is a wine quality metric, taken to be the median preference score of three blind tasters on a scale of 1-10, treated as continuous. The 11 covariates are physicochemical metrics that are commonly collected for wine quality control such as citric acid content, sulphates, etc. The model is fit with a neural network (NN) using the R package **nnet** ([Venables and Ripley, 2002](#)). We fit a NN with 3 hidden units and a small parameter value for weight decay³ and achieved an in-sample $R^2$ of approximately 0.37.

![ICE plots for white wine](image)

(a) c-ICE for NN  (b) d-ICE for NN

Figure 3.11: ICE plots of a NN model for wine ratings versus pH of white wine colored by whether the alcohol content is high (blue) or low (red). To prevent cluttering, only a fraction of the 5,000 observations are plotted.

We find the covariate **pH** to be the most illustrative. The c-ICE plot is displayed in Figure 3.11a. Wines with high alcohol content are colored blue and wines with low

³Note that NN models are highly sensitive to the number of hidden units and weight decay parameter. We therefore offer the following results as merely representative of the type of plots which NN models can generate.
alcohol content are colored red. Note that the PDP shows a linear trend, indicating that on average, higher pH is associated with higher fitted preference scores. While this is the general trend for wines with higher alcohol content, the ICE plots reveal that interaction effects are present in \( \hat{f} \). For many white wines with low alcohol content, the illustration suggests a nonlinear and cumulatively negative association. For these wines, the predicted preference score is actually negatively associated with pH for low values of pH and then begins to increase — a severe departure from what the PDP suggests. However, the area of increase contains no data points, signifying that the increase is merely an extrapolation likely driven by the positive trend of the high alcohol wines. Overall, the ICE plots indicate that for more alcoholic wines, the predicted score is increasing in pH while the opposite is true for wines with low alcohol content. Also, the difference in cumulative effect is meaningful; when varied from the minimum to maximum values of pH, white wine scores vary by roughly 40% of the range of the response variable.

Examining the derivative plot of Figure 3.11b confirms the observations made above. The NN model suggests interactions exist for lower values of pH in particular. Wines with high alcohol content have mostly positive derivatives while those with low alcohol content have mostly negative derivatives. As pH increases, the standard deviation of the derivatives decreases, suggesting that interactions are less prevalent at higher levels of pH.

### 3.5.3 Diabetes Classification in Pima Indians

The last dataset consists of 332 Pima Indians (Smith and Everhart, 1988) obtained from the R library MASS. Of the 332 subjects, 109 were diagnosed with diabetes, the binary response variable which was fit using seven predictors (with body metrics such as blood pressure, glucose concentration, etc.). We model the data using a RF and
achieve an out-of-bag misclassification rate of 22%.

Figure 3.12: ICE plots for a RF model for estimated centered logit of the probability of contracting diabetes versus skin colored by subject age.

Once again, ICE plots offer the practitioner a more comprehensive view of the output of the black box. For example, the covariate skin thickness about the triceps is plotted as a c-ICE in Figure 3.12a. The PDP clearly shows an increase in the predicted centered log odds of contracting diabetes. This is expected given that skin is a proxy for obesity, a major risk factor for diabetes. However, the ICE plot illustrates a more elaborate model fit. Many subjects with high skin have a flat risk of diabetes according to \( \hat{f} \); others with comparable thickness exhibit a much larger centered log-odds increase. Figure 3.12b shows that the RF model fits interactions across the range of skin with the largest heterogeneity in effect occurring when skin is slightly above 30. This can be seen in the standard deviation of the derivative in the bottom pane of Figure 3.12b.

The curves at the top of the figure mainly correspond to younger people. The estimated effect of high thickness is seen to be an extrapolation.
3.6 A Visual Test for Additivity

Thus far we have used the ICE toolbox to explore the output of black box models. We have explored whether \( \hat{f} \) has additive structure or if interactions exist, and also examined \( \hat{f} \)'s extrapolations in \( \mathcal{X} \)-space. To better visualize interactions, we plotted individual curves in colors according to the value of a second predictor \( x_k \). We have not asked whether these findings are reflective of phenomena in any underlying model.

When heterogeneity in ICE plots is observed, the researcher can adopt two mindsets. When one considers \( \hat{f} \) to be the fitted model used for subsequent predictions, the heterogeneity is of interest because it determines future fitted values. This is the mindset we have considered thus far. Separately, it might be interesting to ascertain whether interactions between \( x_S \) and \( x_C \) exist in the data generating model, denoted \( f \). This question exists for other discoveries made using ICE plots, but we focus here on interactions.

The problem of assessing the statistical validity of discoveries made by examining plots is addressed in Buja et al. (2009) and Wickham et al. (2010). The central idea in these papers is to insert the observed plot randomly into a lineup of null plots generated from data sampled under a null distribution. If the single real plot is correctly identified amongst 19 null plots, for example, then “the discovery can be assigned a \( p \)-value of 0.05” (Buja et al., 2009). A benefit of this approach is that the procedure is valid despite the fact that we have not specified the form of the alternative distribution — the simple instruction “find the plot that appears different” is sufficient.

3.6.1 Procedure

We adapt this framework to the specific problem of using ICE plots to evaluate additivity in a statistically rigorous manner. For the exposition in this section, suppose
that the response \( y \) is continuous, the covariates \( x \) are fixed, and \( y = f(x) + \mathcal{E} \). Further assume \( \mathbb{E} [\mathcal{E}] = 0 \) and

\[
f(x) = g(x_S) + h(x_C),
\]  

(3.8)

meaning the true \( x \)-conditional expectation of \( y \) is additive in functions of \( x_S \) and \( x_C \). Let \( F \) be the distribution of \( \hat{f} \) when Equation 3.8 holds and \( f \) is additive. We wish to test \( H_0: \hat{f} \sim F \) versus \( H_a: H_0 \) is false.

Recall that ICE plots displaying non-parallel curves suggest that \( \hat{f} \) is not additive in functions of \( x_S \) and \( x_C \). Thus if we can correctly identify a plot displaying such features amongst \( K - 1 \) null plots generated under \( F \), the discovery is valid at \( \alpha = 1/K \).

We sample from \( F \) by using backfitting (Breiman and Friedman, 1985) to generate \( g^* \) and \( h^* \), estimates of \( g \) and \( h \), and then bootstrapping the residuals. Both \( g^* \) and \( h^* \) can be obtained via any supervised learning procedures. The general procedure for \( |S| = 1 \) proceeds as follows.

1. Using backfitting, obtain \( g^* \) and \( h^* \). Then compute a vector of fitted values \( \hat{y}^* = g^*(x_S) + h^*(x_C) \) and a vector of residuals \( r^* := y - \hat{y}^* \).

2. Let \( r_b \) be a random resampling of \( r^* \). If heteroscedasticity is of concern, one can keep \( r^* \)'s absolute values fixed and let \( r_b \) be a permutation of \( r^* \)'s signs. Define \( y_b := \hat{y}^* + r_b \). Note that \( \mathbb{E} [y_b \mid x] \) is additive in \( g^*(x_S) \) and \( h^*(x_C) \).

3. Fit \( y_b \) to \( X \) using the same learning algorithm that generated the original ICE (c-ICE or d-ICE) plot to produce \( \hat{f}_b \). This yields a potentially non-additive approximation to null data generated using an additive model.

4. Display an ICE (or c-ICE or d-ICE) plot for \( \hat{f}_b \). Deviations from additivity observed
in this plot must be due to sources other than interactions between \( x_S \) and \( x_C \) in the underlying data.

5 Repeat steps (2) - (4) \( K - 1 \) times, then randomly insert the true plot amongst these \( K - 1 \) null plots.

6 If the viewer can correctly identify the true plot amongst all \( K \) plots, the discovery is valid for level \( \alpha = 1/K \). Note that the discovery is conditional on the procedures for generating \( g^* \) and \( h^* \).

### 3.6.2 Examples

An application of this visual test where \( g \) is taken to be the “supersmoother” \cite{Friedman1984} and \( h \) is a BART model is illustrated using the depression data of Section 3.5.1. We sample \( r_b \) by permuting signs. The data analyst might be curious if the ICE plot is consistent with the treatment being additive in the model. We employ the additivity lineup test in Figure 3.13 using 20 images. We reject the null hypothesis of additivity of the treatment effect at \( \alpha = 1/20 = 0.05 \) since the true plot (row 2, column 2) is clearly identifiable. This procedure can be a useful test in clinical settings when the treatment effect is commonly considered linear and additive and can alert the practitioner that interactions should be investigated.

Another application of this visual test where \( g \) is taken to be the supersmoother and \( h \) is a NN model is illustrated using the wine data of Section 3.5.2. Here again we sample \( r_b \) by permuting signs. The data analyst may want to know if the fitted model is suggestive of interactions between pH and the remaining features in the underlying model. We employ the additivity lineup test in Figure 3.14, again using 20 images.

Looking closely one sees that the first and third plots in the last row have the largest range of cumulative effects and exhibit more curvature in individual curves than most of the other plots, making them the most extreme violations of the null.
Figure 3.13: Additivity lineup test for the predictor \textit{treatment} in the depression clinical trial dataset of Section 3.5.1.

Readers that singled out the first plot in the last row would have a valid discovery at $\alpha = 0.05$, but clearly the evidence of non-additivity is much weaker here than in the previous example. Whereas Figure 3.13 suggests the real plot is identifiable amongst more than 20 images, it would be easy to confuse Figure 3.14’s true plot with the one in row 4, column 3. Hence there is only modest evidence that pH’s impact on $\hat{f}$ is different from what a NN might generate if there were no interactions between pH and the other predictors.

### 3.7 Discussion

We developed a suite of tools for visualizing the fitted values generated by an arbitrary supervised learning procedure. Our work extends the classical partial dependence plot (PDP), which has rightfully become a very popular visualization tool for black-box machine learning output. The partial functional relationship, however, often varies conditionally on the values of the other variables. The PDP offers the average of these relationships and thus individual conditional relationships are consequently masked,
unseen by the researcher. These individual conditional relationships can now be visualized, giving researchers additional insight into how a given black box learning algorithm makes use of covariates to generate predictions.

The ICE plot, our primary innovation, plots an entire distribution of individual conditional expectation functions for a variable $x_S$. Through simulations and real data examples, we illustrated much of what can be learned about the estimated model $\hat{f}$ with the help of ICE. For instance, when the remaining features $x_C$ do not influence the association between $x_S$ and $\hat{f}$, all ICE curves lie on top of one another. When $\hat{f}$ is additive in functions of $x_C$ and $x_S$, the curves lie parallel to each other. And when the partial effect of $x_S$ on $\hat{f}$ is influenced by $x_C$, the curves will differ from each other in shape. Additionally, by marking each curve at the $x_S$ value observed in the training data, one can better understand $\hat{f}$’s extrapolations. Sometimes these properties are more easily distinguished in the complementary “centered ICE” (c-ICE) and “derivative ICE” (d-ICE) plots. In sum, the suite of ICE plots provides a tool for visualizing an arbitrary fitted model’s map between predictors and predicted values.

The ICE suite has a number of possible uses that were not explored in this work.
While we illustrate ICE plots using the same data as was used to fit \( \hat{f} \), out-of-sample ICE plots could also be valuable. For instance, ICE plots generated from random vectors in \( \mathbb{R}^p \) can be used to explore other parts of \( \mathcal{X} \) space, an idea advocated by Plate et al. (2000). Further, for a single out-of-sample observation, plotting an ICE curve for each predictor can illustrate the sensitivity of the fitted value to changes in each predictor for this particular observation, which is the goal of the “contribution plots” of Strumbelj and Kononenko (2011). Additionally, investigating ICE plots from \( \hat{f} \)’s produced by multiple statistical learning algorithms can help the researcher compare models. Exploring other functionality offered by the ICEbox package, such as the ability to cluster ICE curves, is similarly left for subsequent research.

The tools summarized thus far pertain to exploratory analysis. Many times the ICE toolbox provides evidence of interactions, but how does this evidence compare to what these plots would have looked like if no interactions existed? Section 3.6 proposed a testing methodology. By generating additive models from a null distribution and introducing the actual ICE plot into the lineup, interaction effects can be distinguished from noise, providing a test at a known level of significance. Future work will extend the testing methodology to other null hypotheses of interest.

Acknowledgements

We thank Richard Berk for insightful comments on multiple drafts and suggesting color overloading. We thank Andreas Buja for helping conceive the testing methodology. We thank Abba Krieger for his helpful suggestions. We also wish to thank Zachary Cohen for the depression data of Section 3.5.1 and helpful comments.
This thesis describes methods for enhancing the interpretability of fitted models, with emphasis on classification and regression trees. A major emphasis of this work is to not only develop useful methodologies, but also make these techniques accessible to practitioners. As such, we provide fully documented software implementations of these procedures via open source R packages.

Chapter 1 introduces novel techniques for growing classification and regression trees designed to induce visually interpretable trees. The software implementation of these methods is available in the R package itree, described in Chapter 2. The central idea is to capitalize on the realization that “different models, all of them equally good, may give different pictures of the relation between the predictor and response variables” (Breiman, 2001b) by selecting amongst such models one that is readily interpretable. We achieve this by using penalties to control the subset of variables and sequence of variables used in each branch of a tree.

Using real datasets, we find that our procedures return trees that are more interpretable than their unpenalized counterparts but suffer very little increase in out-of-sample loss, yielding a “free lunch” of sorts. This is despite the fact that our parameters are not tuned to out-of-sample loss. The question of whether these parameters can be explicitly selected to minimize holdout loss is beyond the scope of
this work, but presents an interesting direction for future research.

Whereas Chapters 1 and 2 relate to tree-based methods, Chapter 3 introduces Individual Conditional Expectation plots, a suite of tools for visualizing the model estimated by any supervised learning algorithm. The procedures outlined in Chapter 3 are available in the R package ICEbox. This work is a refinement of partial dependence plots, which help visualize the average partial relationship between the predicted response and one or more features. We show that the averaging inherent in partial dependence plots can obscure the complexity of the modeled relationship. Accordingly, our plots disaggregate the partial dependence plot by graphing the estimated conditional expectation of the predicted response and the feature for individual observations.

By applying these plots to simulated examples and actual datasets, we demonstrate how these tools can highlight the variation in fitted values across the range of a covariate. The degree and nature of the variation in this relationship across observations is reflective of a variety of characteristics of the fitted model. In particular, we show how these plots can be used to diagnose features of the fitted model such as additivity, interaction effects and extrapolation in \( \mathcal{X} \)-space. In addition to providing a plotting suite for exploratory analysis, we include a visual test for additive structure in the data generating model.
A.1 Chapter 1 Supplement: Penalized Split Criteria for Interpretable Trees

A.1.1 Gain Function Scaling

As mentioned in section 1.3, splitting criteria must be adjusted to fit into the penalization framework. Here we give the details of how this is done for each criterion on both classification and regression.

As a motivating example, consider the CART regression tree algorithm. Recall that this algorithm uses the gain function (1.2) and impurity function (1.3), resulting in a search for the split yielding the minimal per-observation mean-squared error. If we wish to induce a more interpretable fit by using one of the penalties we must specify the constant $k$. However, the mean-squared error’s magnitude varies directly with the level the $y_i$’s, and so penalizing the gain function directly would require us to calibrate $k$ to $y$. We avoid this by scaling $\theta$ by the parent node’s impurity, as follows:

$$\theta_{scaled}(t, s) = \frac{\theta(t, s)}{\phi(t)} = \frac{\phi(t) - \left[\frac{n_t}{n_L} \phi(t_L) + \frac{n_t}{n_R} \phi(t_R)\right]}{\phi(t)}, \quad (A.1)$$
where $\phi(t)$, $\phi(t_L)$ and $\phi(t_R)$ are the parent node MSE, left daughter MSE and right daughter MSE respectively. Now because the parent node’s MSE is constant across all candidate splits we have that

$$\arg \max_{s \in S} \{\theta_{scaled}(t, s)\} = \arg \max_{s \in S} \{\theta(t, s)\},$$

meaning that the optimal split $s^*$ is invariant to the scaling. Moreover, $\theta_{scaled}$ can be thought of as the fractional improvement in the impurity function, freeing its magnitude from any direct dependence on the level of $y$. Note that if the numerator of (A.1) is negative it is best not to split, and hence for any feasible $s$ we have $\theta_{scaled}(t, s) \in [0, 1]$. Thus we can safely apply a penalty function to $\theta_{scaled}$ using $k \in (0, 1)$. The end result is that splits yielding non-zero penalties (that is, those that are less interpretable) require larger fractional improvements in impurity than those that do not.

There are two features of (A.1) that make this possible.

- The ordering of $\theta(t, s)$ is equivalent to that of $\theta_{scaled}(t, s)$ for all $s \in S$.
- The fact that $0 \leq \theta_{scaled}(t, s) \leq 1$ for all feasible $s$.

The first ensures that the scaling does not change the optimal split $s^*$ and allows us to recover the unpenalized criterion by setting $k = 0$. The second ensures that we can restrict $k \in [0, 1]$. In most situations the unscaled gain function is bounded above by the parent node’s impurity function, and so scaling by parent-node impurity suffices. This is the case for CART.

For the one-sided methods such as [Buja and Lee (2001)], The only cases in which this does not work is the high and low means one-sided extremes criteria. Table A.1 summarizes the specifics.
Table A.1: Scaling of Impurity Functions

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Impurity Function</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART, (1.2)</td>
<td>regression</td>
<td>( \phi_R(t) )</td>
</tr>
<tr>
<td></td>
<td>classification</td>
<td>( \phi_G(t) )</td>
</tr>
<tr>
<td>One-Sided, (1.14)</td>
<td>purity, regression</td>
<td>( \phi_{osp,R}(t) )</td>
</tr>
<tr>
<td></td>
<td>purity, classification</td>
<td>( \phi_{osp,C}(t) )</td>
</tr>
<tr>
<td></td>
<td>extremes, regression; high means</td>
<td>( \max { { i \in t[y_i] } - \bar{y}_t } )</td>
</tr>
<tr>
<td></td>
<td>low means</td>
<td>( \bar{y}_t - \min { { i \in t[y_i] } } )</td>
</tr>
<tr>
<td></td>
<td>extremes, classification</td>
<td>( \phi_{ose,C}(t) )</td>
</tr>
</tbody>
</table>

A.1.2 One-Sided Split Criteria

The one-sided splitting procedures introduced in Buja and Lee (2001) fit into the framework described in Section 1.2. Whereas all CART techniques use the split criteria defined by Equation 1.2, all one-sided methods use

\[
\theta_{OS}(t, s) = \phi(t) - \min \{ \phi(t_L), \phi(t_R) \} .
\] (A.2)

Because they ignore the \( \phi \) value in one of the child nodes, Buja and Lee (2001) call methods that follow (1.14) one-sided. Buja and Lee (2001) uses two classes of impurity functions \( \phi \), resulting in two types of one-sided methods: one-sided impurity and one-sided extremes. In regression, the former seeks the single child node with lowest MSE whereas the latter seeks the child node with the highest (or lowest) average \( y \) value. In classification, one-sided purity uses Gini impurity and one-sided extremes seeks nodes with high sample proportions of a particular class. The impurity functions \( \phi \) are defined formally in Table A.2.

Note that the one-sided purity methods using the same impurity function as CART. In regression we use \( \phi_R \), the within-node sample variance, and in classification we use \( \phi_G \), Gini impurity. In contrast, the one-sided extremes procedures use impurity functions that quantify some aspect of the \( y \) values themselves as opposed to their variability. The high (low) means technique finds the single bucket with
Table A.2: Definitions of One-Sided Impurity Functions

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Problem</th>
<th>Impurity Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided purity</td>
<td>Regression</td>
<td>( \phi_{osp,R} = \frac{1}{n_t} \sum_{i \in t} (y_i - \bar{y}_t)^2 )</td>
</tr>
<tr>
<td>One-sided purity</td>
<td>Classification</td>
<td>( \phi_{osp,C} = \sum_{k \in K} \hat{p}<em>{k,t} (1 - \hat{p}</em>{k,t}) )</td>
</tr>
<tr>
<td>One-sided extremes, high means</td>
<td>Regression</td>
<td>( \phi_{ose,hm} = \bar{y}_t )</td>
</tr>
<tr>
<td>One-sided extremes, low means</td>
<td>Regression</td>
<td>( \phi_{ose,lm} = -\bar{y}_t )</td>
</tr>
<tr>
<td>One-sided extremes</td>
<td>Classification</td>
<td>( \phi_{ose,C} = \hat{p}_{k} )</td>
</tr>
</tbody>
</table>

highest (lowest) sample mean, for example. Note that to use one-sided extremes in a classification setting, the user needs to specify the *class of interest*, denoted \( k' \). If we are classifying handwritten digits, for instance, setting \( k' = \text{“2”} \) means we choose \( s^* \) as to isolate the child node with the highest proportion of observations with \( y = \text{“2”} \). It is apparent that in general this will be a different split than if we set \( k' = \text{“3”} \).
A.1.3 Algorithms

Algorithm 1 Procedure for Generating Out-of-Bag Error Estimates: Given $L$, a learning sample of $N (y_i, x_i)$ pairs; $B$, the number of bootstrap replications; $F$, a tree-fitting procedure inclusive of the method for choosing $k^*$; $\Theta[\hat{f}, (y, x)]$, a function specifying the loss from estimating $y$ with $\hat{f}(x)$; return $\hat{\Theta}$, a $B \times 1$ vector of the estimated risk for each bootstrap replicate.

1: function OOBEE($L$, $B$, $F$, $\Theta[\hat{f}, (y, x)]$)
2:     #initialize:
3:     $l \leftarrow 0_{B \times 1}$
4:     $nl \leftarrow 0_{B \times 1}$
5:     $\hat{\Theta} \leftarrow 0_{B \times 1}$
6:     
7:     for $b \leftarrow 1 \ldots B$ do
8:         #bootstrap sampling:
9:         $L_b \leftarrow N$ samples with replacement from $L$
10:        $H_b \leftarrow L \setminus L_b$
11:        
12:        #fit a tree to the bootstrap learning sample:
13:        $T_b \leftarrow F(L_b)$
14:        
15:        #evaluate the tree on holdout data:
16:        for $i \leftarrow 1 \ldots N$ do
17:            if $(y_i, x_i) \in H_b$ then
18:                $\lambda \leftarrow \Theta[T_b, (y_i, x_i)]$
19:                $l[b] \leftarrow l[b] + \lambda$
20:                $nl[b] \leftarrow nl[b] + 1$
21:            end if
22:        end for
23:        
24:        #normalize:
25:        for $i \leftarrow 1 \ldots B$ do
26:            $\hat{\Theta}[i] \leftarrow l[i]/nl[i]$
27:        end for
28:        
29:        return $\hat{\Theta}$
30: end function
A.1.4 Out-of-Bag Performance Statistics

Table A.3: This table displays the OOB performance of the applying the penalization methods to CART on 10 regression datasets. The two leftmost columns show the average out-of-bag $R^2$ and MSE over 100 bootstrap runs using unpenalized CART. The $R^2$ column is not directly relevant but gives a sense of the difficulty of each problem. The columns titled “MSE Increase%” show the average percentage increase in out-of-bag MSE incurred from applying the penalties with $c = .10$. The “Average $k^*$” columns show the mean value over the 100 runs of the penalization constant when it is chosen as per (1.8) with $c = .10$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized</th>
<th>New Variable Penalty</th>
<th>EMA Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OOB $R^2$</td>
<td>OOB MSE</td>
<td>MSE Increase%</td>
</tr>
<tr>
<td>boston</td>
<td>0.73</td>
<td>22.88</td>
<td>1.5</td>
</tr>
<tr>
<td>abalone</td>
<td>0.46</td>
<td>5.67</td>
<td>5.0</td>
</tr>
<tr>
<td>wine.red</td>
<td>0.29</td>
<td>0.47</td>
<td>1.8</td>
</tr>
<tr>
<td>wine.white</td>
<td>0.26</td>
<td>0.57</td>
<td>4.3</td>
</tr>
<tr>
<td>ozone</td>
<td>0.63</td>
<td>23.87</td>
<td>0.3</td>
</tr>
<tr>
<td>pole</td>
<td>0.78</td>
<td>403.37</td>
<td>3.0</td>
</tr>
<tr>
<td>triazine</td>
<td>0.06</td>
<td>0.02</td>
<td>-1.7</td>
</tr>
<tr>
<td>ankara</td>
<td>0.96</td>
<td>10.16</td>
<td>3.3</td>
</tr>
<tr>
<td>baseball</td>
<td>0.57</td>
<td>703210.90</td>
<td>1.3</td>
</tr>
<tr>
<td>compactiv</td>
<td>0.77</td>
<td>78.34</td>
<td>4.7</td>
</tr>
</tbody>
</table>
Table A.4: OOB performance of penalization methods on regression datasets when using the One-Sided Purity split criterion. The simulation settings and meaning of the columns follows those in Table A.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized OOB MSE</th>
<th>New Variable Penalty MSE Increase%</th>
<th>Average $k^*$</th>
<th>EMA Penalty MSE Increase%</th>
<th>Average $k^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>boston</td>
<td>28.68</td>
<td>6.8</td>
<td>0.12</td>
<td>3.5</td>
<td>0.24</td>
</tr>
<tr>
<td>abalone</td>
<td>6.07</td>
<td>3.3</td>
<td>0.47</td>
<td>2.4</td>
<td>0.71</td>
</tr>
<tr>
<td>wine.red</td>
<td>0.50</td>
<td>3.7</td>
<td>0.16</td>
<td>2.2</td>
<td>0.31</td>
</tr>
<tr>
<td>wine.white</td>
<td>0.63</td>
<td>5.4</td>
<td>0.11</td>
<td>4.5</td>
<td>0.34</td>
</tr>
<tr>
<td>ozone</td>
<td>25.03</td>
<td>2.3</td>
<td>0.06</td>
<td>1.7</td>
<td>0.29</td>
</tr>
<tr>
<td>pole</td>
<td>431.82</td>
<td>-0.6</td>
<td>0.07</td>
<td>-6.4</td>
<td>0.26</td>
</tr>
<tr>
<td>triazine</td>
<td>0.02</td>
<td>-0.6</td>
<td>0.16</td>
<td>0.2</td>
<td>0.44</td>
</tr>
<tr>
<td>ankara</td>
<td>11.01</td>
<td>4.1</td>
<td>0.03</td>
<td>2.6</td>
<td>0.25</td>
</tr>
<tr>
<td>baseball</td>
<td>1009994.68</td>
<td>-2.3</td>
<td>0.10</td>
<td>-2.2</td>
<td>0.28</td>
</tr>
<tr>
<td>compactiv</td>
<td>157.60</td>
<td>2.4</td>
<td>0.13</td>
<td>-6.7</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table A.5: OOB performance of penalization methods on regression datasets when using the High-Means split criterion. The simulation settings and meaning of the columns follows those in Table A.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized OOB MSE</th>
<th>New Variable Penalty MSE Increase%</th>
<th>Average $k^*$</th>
<th>EMA Penalty MSE Increase%</th>
<th>Average $k^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>boston</td>
<td>22.19</td>
<td>5.9</td>
<td>0.03</td>
<td>4.5</td>
<td>0.11</td>
</tr>
<tr>
<td>abalone</td>
<td>6.10</td>
<td>1.1</td>
<td>0.99</td>
<td>1.1</td>
<td>0.99</td>
</tr>
<tr>
<td>wine.red</td>
<td>0.49</td>
<td>2.4</td>
<td>0.53</td>
<td>2.3</td>
<td>0.60</td>
</tr>
<tr>
<td>wine.white</td>
<td>0.59</td>
<td>6.4</td>
<td>0.85</td>
<td>6.5</td>
<td>0.87</td>
</tr>
<tr>
<td>ozone</td>
<td>24.68</td>
<td>2.4</td>
<td>0.20</td>
<td>3.7</td>
<td>0.32</td>
</tr>
<tr>
<td>pole</td>
<td>623.68</td>
<td>-0.5</td>
<td>0.16</td>
<td>0.7</td>
<td>0.26</td>
</tr>
<tr>
<td>triazine</td>
<td>0.02</td>
<td>0.3</td>
<td>0.06</td>
<td>-3.0</td>
<td>0.25</td>
</tr>
<tr>
<td>ankara</td>
<td>9.74</td>
<td>4.5</td>
<td>0.01</td>
<td>5.2</td>
<td>0.10</td>
</tr>
<tr>
<td>baseball</td>
<td>862508.60</td>
<td>6.8</td>
<td>0.07</td>
<td>4.5</td>
<td>0.17</td>
</tr>
<tr>
<td>compactiv</td>
<td>203.87</td>
<td>0.7</td>
<td>0.04</td>
<td>-1.4</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Table A.6: OOB performance of penalization methods on regression datasets when using the Low-Means split criterion. The simulation settings and meaning of the columns follows those in Table A.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized OOB MSE</th>
<th>New Variable Penalty Average Increase</th>
<th>EMA Penalty Average Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>boston</td>
<td>23.44</td>
<td>11.3 0.02</td>
<td>8.2 0.15</td>
</tr>
<tr>
<td>abalone</td>
<td>6.00</td>
<td>4.5 0.39</td>
<td>3.1 0.53</td>
</tr>
<tr>
<td>wine.red</td>
<td>0.48</td>
<td>5.9 0.02</td>
<td>2.8 0.13</td>
</tr>
<tr>
<td>wine.white</td>
<td>0.60</td>
<td>6.4 0.02</td>
<td>2.6 0.11</td>
</tr>
<tr>
<td>ozone</td>
<td>21.70</td>
<td>5.0 0.02</td>
<td>2.4 0.16</td>
</tr>
<tr>
<td>pole</td>
<td>303.06</td>
<td>4.6 0.05</td>
<td>6.1 0.14</td>
</tr>
<tr>
<td>triazine</td>
<td>0.02</td>
<td>2.7 0.08</td>
<td>-0.0 0.25</td>
</tr>
<tr>
<td>ankara</td>
<td>10.36</td>
<td>4.5 0.01</td>
<td>2.7 0.08</td>
</tr>
<tr>
<td>baseball</td>
<td>834320.83</td>
<td>1.7 0.03</td>
<td>0.1 0.16</td>
</tr>
<tr>
<td>compactiv</td>
<td>81.19</td>
<td>5.6 0.40</td>
<td>3.4 0.42</td>
</tr>
</tbody>
</table>
Table A.7: OOB performance of penalization methods on classification datasets when using CART. The columns titled “OOB MR” report the average out-of-bag misclassification rate over 100 bootstrap runs. The “Average $k^*$” columns show the mean value of the penalization constant over the 100 runs when $k^*$ is chosen as per (1.8) with $c = .10$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized OOB MR</th>
<th>New Variable Penalty OOB MR</th>
<th>Average $k^*$</th>
<th>EMA Penalty OOB MR</th>
<th>Average $k^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pima</td>
<td>0.26</td>
<td>0.27</td>
<td>0.05</td>
<td>0.26</td>
<td>0.02</td>
</tr>
<tr>
<td>breast.cancer</td>
<td>0.07</td>
<td>0.07</td>
<td>0.42</td>
<td>0.07</td>
<td>0.40</td>
</tr>
<tr>
<td>bands</td>
<td>0.33</td>
<td>0.33</td>
<td>0.98</td>
<td>0.33</td>
<td>0.98</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.13</td>
<td>0.12</td>
<td>0.14</td>
<td>0.13</td>
<td>0.08</td>
</tr>
<tr>
<td>cardio</td>
<td>0.14</td>
<td>0.14</td>
<td>0.03</td>
<td>0.14</td>
<td>0.03</td>
</tr>
<tr>
<td>parkinsons</td>
<td>0.16</td>
<td>0.16</td>
<td>0.18</td>
<td>0.16</td>
<td>0.14</td>
</tr>
<tr>
<td>glass</td>
<td>0.36</td>
<td>0.37</td>
<td>0.01</td>
<td>0.37</td>
<td>0.01</td>
</tr>
<tr>
<td>iris</td>
<td>0.06</td>
<td>0.06</td>
<td>0.47</td>
<td>0.06</td>
<td>0.46</td>
</tr>
<tr>
<td>digit.rec</td>
<td>0.31</td>
<td>0.34</td>
<td>0.01</td>
<td>0.55</td>
<td>0.01</td>
</tr>
<tr>
<td>waveform1</td>
<td>0.28</td>
<td>0.29</td>
<td>0.03</td>
<td>0.29</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table A.8: OOB performance of penalization methods on classification datasets when using One-Sided Purity. The simulation settings and meaning of the columns follows those in Table A.7.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Unpenalized OOB MR</th>
<th>New Variable Penalty OOB MR</th>
<th>Average $k^*$</th>
<th>EMA Penalty OOB MR</th>
<th>Average $k^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pima</td>
<td>0.26</td>
<td>0.27</td>
<td>0.07</td>
<td>0.26</td>
<td>0.04</td>
</tr>
<tr>
<td>breast.cancer</td>
<td>0.06</td>
<td>0.06</td>
<td>0.38</td>
<td>0.06</td>
<td>0.32</td>
</tr>
<tr>
<td>bands</td>
<td>0.42</td>
<td>0.42</td>
<td>0.98</td>
<td>0.42</td>
<td>0.98</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.18</td>
<td>0.18</td>
<td>0.13</td>
<td>0.18</td>
<td>0.08</td>
</tr>
<tr>
<td>cardio</td>
<td>0.14</td>
<td>0.14</td>
<td>0.03</td>
<td>0.14</td>
<td>0.03</td>
</tr>
<tr>
<td>parkinsons</td>
<td>0.17</td>
<td>0.17</td>
<td>0.32</td>
<td>0.17</td>
<td>0.19</td>
</tr>
<tr>
<td>glass</td>
<td>0.43</td>
<td>0.43</td>
<td>0.31</td>
<td>0.44</td>
<td>0.12</td>
</tr>
<tr>
<td>iris</td>
<td>0.10</td>
<td>0.11</td>
<td>0.48</td>
<td>0.11</td>
<td>0.47</td>
</tr>
<tr>
<td>digit.rec</td>
<td>0.40</td>
<td>0.42</td>
<td>0.12</td>
<td>0.42</td>
<td>0.04</td>
</tr>
<tr>
<td>waveform1</td>
<td>0.25</td>
<td>0.27</td>
<td>0.12</td>
<td>0.26</td>
<td>0.04</td>
</tr>
</tbody>
</table>
Table A.9: Performance of penalization methods on classification datasets when using One-Sided Extremes. We arbitrarily assign each observed class in the dataset an index 1, ..., K. The columns under “class1” correspond to setting the class of interest to be the first class, and likewise for the second and third. Hence for binary classification problems the third group of columns is blank. When there are more than two classes we report results when the class of interest is set to be the third class in our random ordering. Under each class of interest, the three columns refer to the unpenalized, new variable penalty and EMA procedures, respectively. The first row in a class-method pair reports the mean out-of-bag misclassification rates (100 runs) and the second reports the average $k^*$ value when $c = 0.10$. For the Pima Indians data, for example, when the class of interest is class1 the average misclassification rate is 0.27 and the average $k^*$ is 0.66.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>class1</th>
<th>class2</th>
<th>class3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U</td>
<td>NV</td>
<td>EMA</td>
</tr>
<tr>
<td>pima</td>
<td>0.27</td>
<td>0.27</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>0.66</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>breast.cancer</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0.40</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>bands</td>
<td>0.32</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>0.97</td>
<td>0.97</td>
<td></td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>cardio</td>
<td>0.16</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>0.36</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>parkinsons</td>
<td>0.18</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>0.71</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>glass</td>
<td>0.51</td>
<td>0.49</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>0.86</td>
<td>0.80</td>
<td></td>
</tr>
<tr>
<td>iris</td>
<td>0.37</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>0.94</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>digit.rec</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>waveform1</td>
<td>0.40</td>
<td>0.43</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.08</td>
<td></td>
</tr>
</tbody>
</table>
A.2 Chapter 2 Supplement: Software for Interpretable Classification and Regression Trees

Algorithm 2 Local Risk Estimation Procedure: Given \( \mathcal{L} \), a learning sample of \( N \) \((y_i, x_i)\) pairs; \( B \), the number of bootstrap replications; \( F \), a tree-fitting procedure; \( \Theta[\hat{f}, (y, x)] \), a function specifying the loss from estimating \( y \) with \( \hat{f}(x) \); return \( \hat{\Theta} \), an \( N \times 1 \) vector of the estimated risk for each observation in \( \mathcal{L} \).

1: function LRE(\( \mathcal{L}, B, F, \Theta[\hat{f}, (y, x)] \))
2:    #initialize:
3:    \( l \leftarrow 0_{N \times 1} \)
4:    \( nl \leftarrow 0_{N \times 1} \)
5:    \( \hat{\Theta} \leftarrow 0_{N \times 1} \)
6:
7:    for \( b \leftarrow 1 \ldots B \) do
8:        #bootstrap sampling:
9:        \( L_b \leftarrow N \) samples with replacement from \( \mathcal{L} \)
10:       \( H_b \leftarrow \mathcal{L} \setminus L_b \)
11:    
12:        #fit a tree to the bootstrap learning sample:
13:       \( T_b \leftarrow F(L_b) \)
14:    
15:        #evaluate the tree on holdout data:
16:       for \( i \leftarrow 1 \ldots N \) do
17:           if \( (y_i, x_i) \in H_b \) then
18:              \( \lambda \leftarrow \Theta[T_b, (y_i, x_i)] \)
19:              \( l[i] \leftarrow l[i] + \lambda \)
20:              \( nl[i] \leftarrow nl[i] + 1 \)
21:           end if
22:       end for
23:    end for
24:    
25:    #normalize:
26:    for \( i \leftarrow 1 \ldots N \) do
27:       \( \hat{\Theta}[i] \leftarrow l[i]/nl[i] \)
28:    end for
29:    
30:    return \( \hat{\Theta} \)
31: end function
A.3 Chapter 3 Supplement: Statistical Learning Model Visualization with Individual Conditional Expectation Plots

A.3.1 Algorithms

Algorithm 3 ICE algorithm: Given $X$, the $N \times p$ feature matrix, $\hat{f}$, the fitted model, $S \subset \{1, \ldots, p\}$, the subset of predictors for which to compute partial dependence, return $\hat{f}^{(1)}_S, \ldots, \hat{f}^{(N)}_S$, the estimated partial dependence curves for constant values of $x_C$.

```plaintext
1: function ICE($X, \hat{f}, S$)
2:     for $i \leftarrow 1 \ldots N$ do
3:         $f^{(i)}_S \leftarrow 0_{N \times 1}$
4:         $x_C \leftarrow X[i, C]$ \COMMENT{fix $x_C$ at the $i$th observation’s $C$ columns}
5:             for $\ell \leftarrow 1 \ldots N$ do
6:                 $x_S \leftarrow X[\ell, S]$ \COMMENT{vary $x_S$}
7:                     $\hat{f}^{(i)}_S \leftarrow \hat{f}([x_S, x_C])$ \COMMENT{the $i$th curve’s $\ell$th coordinate}
6:     end for
9: end for
10: return $[\hat{f}^{(1)}_S, \ldots, \hat{f}^{(N)}_S]$
11: end function
```

Algorithm 4 d-ICE algorithm: Given $X$, the $N \times p$ feature matrix; $\hat{f}^{(1)}_S, \ldots, \hat{f}^{(N)}_S$, the estimated partial dependence functions for subset $S$ in the ICE plot; $D$, a function that computes the numerical derivative; returns $d\hat{f}^{(1)}_S, \ldots, d\hat{f}^{(N)}_S$, the derivatives of the estimated partial dependence. In our implementation $D$ first smooths the ICE plot using the “supersmoothers” and subsequently estimates the derivative from the smoothed ICE plot.

```plaintext
1: function d-ICE($X, \hat{f}^{(1)}_S, \ldots, \hat{f}^{(N)}_S, D$)
2:     for $i \leftarrow 1 \ldots N$ do
3:         $d\hat{f}^{(i)}_S \leftarrow 0_{N \times 1}$
4:         $x_C \leftarrow X[i, C]$ \COMMENT{row of the $i$th observation, columns corresponding to $C$}
5:             for $\ell \leftarrow 1 \ldots N$ do
6:                 $x_S \leftarrow X[\ell, S]$ \COMMENT{vary $x_S$}
7:                     $d\hat{f}^{(i)}_S \leftarrow D(\hat{f}([x_S, x_C])$ \COMMENT{numerical partial derivative at $\hat{f}^{(i)}([x_S, x_C]$ w.r.t. $x_S$}
6:     end for
9: end for
10: return $[d\hat{f}^{(1)}_S, \ldots, d\hat{f}^{(N)}_S]$
11: end function
```


