# Recursive Partitioning and Tree-Based Methods

# Introduction

An algorithm known as **recursive partitioning** is the key to the nonparametric statistical method of **classification and regression trees (CART)**.

Recursive partitioning is the step-by-step process by which a **decision tree** is constructed by either splitting or not splitting each node on the tree into two daughter nodes.

An attractive feature of the CART methodology is that because the algorithm asks a sequence of hierarchical Boolean questions.

**Example**

Is $X_i \leq \theta_j$?

- $\theta_j$ is a threshold value.

It is relatively simple to understand and interpret the results.

As we described in previous chapters, classification and regression are both supervised learning techniques, but they differ in the way their output variables are defined.

For binary classification problems, the output variable, $Y$, is binary-valued, whereas for regression problems, $Y$ is a continuous variable.

Such a formulation is particularly useful when assessing how well a classification or regression methodology does in predicting $Y$ from a given set of input variables $X_1, X_2, \ldots, X_r$.

In the CART methodology, the input space, $\mathcal{R}^r$, is partitioned into a number of nonoverlapping rectangular ($r = 2$) or cuboid ($r > 2$) regions, each of which is viewed as homogeneous for the purpose of predicting $Y$.

Each region, which has sides parallel to the axes of input space, is assigned a class (in a classification problem) or a constant value (in a regression problem).

Such a partition corresponds to a classification or regression tree (as appropriate).

Tree-based methods have been used extensively in a wide variety of fields.

They have been found especially useful in biomedical and genetic research, marketing, political science, speech recognition, and other applied sciences.

# Classification Trees

A classification tree is the result of asking an ordered sequence of questions, and the type of question asked at each step in the sequence depends upon the answers to the previous questions of the sequence.

The sequence terminates in a prediction of the class.

The unique starting point of a classification tree is called the **root node** and consists of the entire learning set $\mathcal{L}$ at the top of the tree.

A **node** is a subset of the set of variables, and it can be a terminal or nonterminal node. A **nonterminal (or parent) node** is a node that splits into two daughter nodes (a **binary split**).

Such a binary split is determined by a Boolean condition on the value of a single variable, where the condition is either satisfied (yes) or not satisfied (no) by the observed value of that variable.

All observations in $\mathcal{L}$ that have reached a particular (parent) node,

- satisfy the condition for that variable drop down to one of the two daughter nodes;
- the remaining observations at that (parent) node that do not satisfy the condition drop down to the other daughter node.

A node that does not split is called a **terminal node** and is assigned a class label. Each observation in $\mathcal{L}$ falls into one of the terminal nodes.

When an observation of unknown class is dropped down the tree and ends up at a terminal node, it is assigned the class corresponding to the class label attached to that node. There may be more than one terminal node with the same class label.

A single-split tree with only two terminal nodes is called a **stump**. The set of all terminal nodes is called a **partition** of the data.

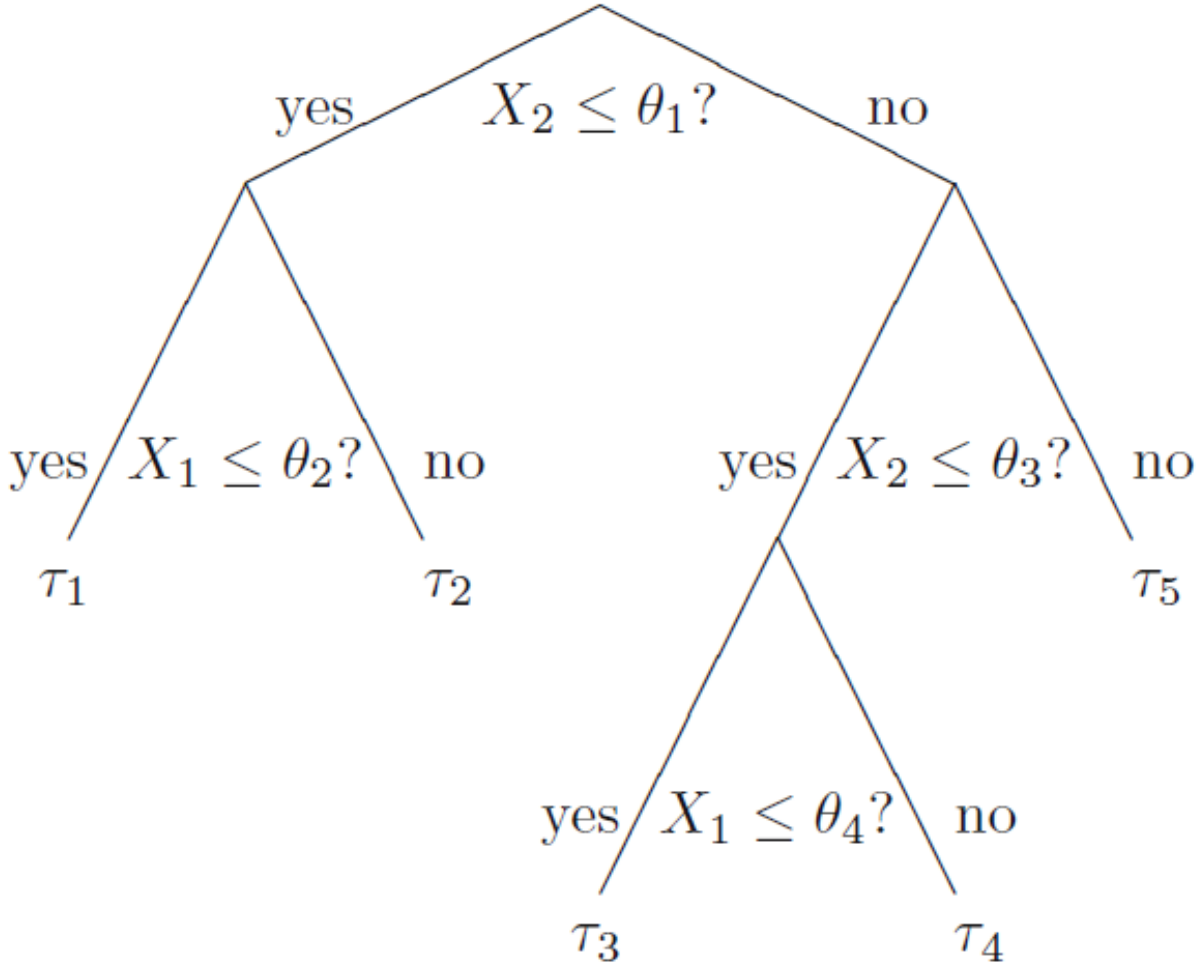


FIGURE: Example of recursive partitioning with two input variables $X_1$ and $X_2$. It shows a decision tree with five terminal nodes, $\tau_1 - \tau_5$, and four splits.

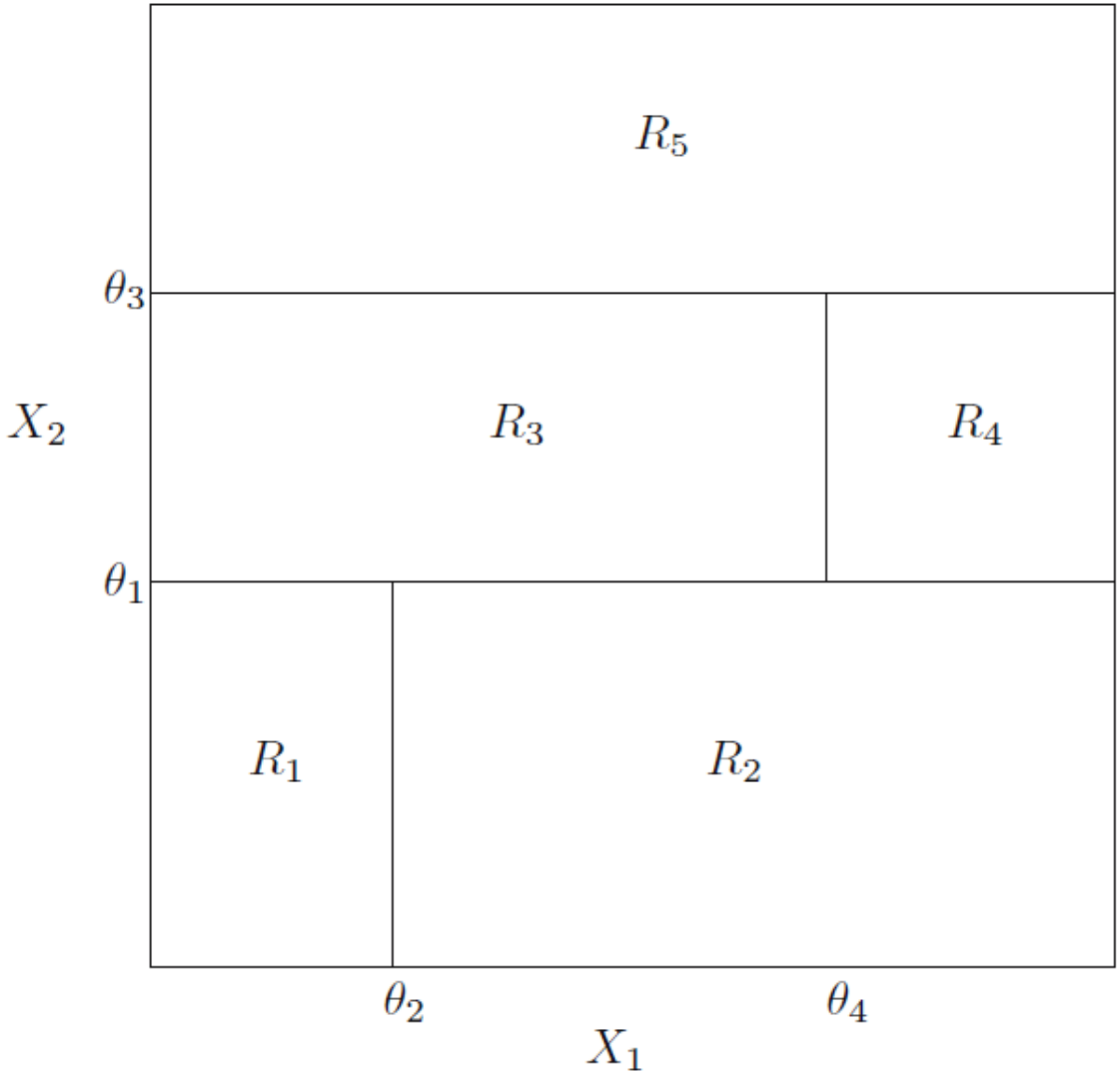


FIGURE: Example of recursive partitioning with two input variables $X_1$ and $X_2$. It shows the partitioning of $\mathcal{R}^2$ into five regions, $R_1 - R_5$, corresponding to the five terminal nodes.

# Tree-Growing Procedure

In order to grow a classification tree, we need to answer four basic questions:

1. How do we choose the Boolean conditions for splitting at each node?
2. Which criterion should we use to split a parent node into its two daughter nodes?
3. How do we decide when a node become a terminal node (i.e., stop splitting)?
4. How do we assign a class to a terminal node?

# Splitting Strategies

At each node, the tree-growing algorithm has to decide on which variable it is best to split.

We need to consider every possible split over all variables present at that node, then enumerate all possible splits, evaluate each one, and decide which is best in some sense.

For a description of splitting rules, we need to make a distinction between ordinal (or continuous) and nominal (or categorical) variables.

# Ordinal or Continuous Variable

For a continuous or ordinal variable, the number of possible splits at a given node is one fewer than the number of its distinctly observed values.

# Nominal or Categorical Variable

Suppose that a particular categorical variable is defined by $M$ distinct categories, $l_1, \ldots, l_M$. The set $S$ of possible splits at that node for that variable is the set of all subsets of $\{l_1, \ldots, l_M\}$.

Denote by $\tau_L$ and $\tau_R$ the left daughter-node and right daughter-node, respectively, emanating from a (parent) node $\tau$.

In general, there are $2^{M-1} - 1$ distinct splits in $S$ for an $M$-categorical variable.

# Total Number of Possible Splits

In the Cleveland heart-disease example, we now add the number of possible splits from categorical variables (19) to the total number of possible splits from continuous variables (372) to get 391 possible splits over all 13 variables at the root node.

In other words, there are 391 possible splits of the root node into two daughter nodes.

**Question**

Which split is best?

# Node Impurity Functions

To choose the best split over all variables, we first need to choose the best split for a given variable.

Accordingly, we define a measure of goodness of a split.

Let $\Pi_1, \ldots, \Pi_K$ be the $K \geq 2$ classes.

For node $\tau$, we define the node impurity function $i(\tau)$ as

$$i(\tau) = \phi\big(p(1|\tau), \ldots, p(K|\tau)\big).$$

- $p(k|\tau)$ is an estimate of $P(\mathbf{X} \in \Pi_k|\tau)$, the conditional probability that an observation $\mathbf{X}$ is in $\Pi_k$ given that it falls into node $\tau$.

We require $\phi$ to be a symmetric function, defined on the set of all $K$-tuples of probabilities $(p_1, \ldots, p_K)$ with unit sum, minimized at the points $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, 0, \ldots, 0, 1)$ and maximized at the point $(1/K, \ldots, 1/K)$.

In the two-class case ($K = 2$), these conditions reduce to a symmetric $\phi(p)$ maximized at the point $p = 1/2$ with $\phi(0) = \phi(1) = 0$.

# Entropy Function

One such function $\phi$ is the **entropy function**,

$$i(\tau) = -\sum_{k=1}^{K} p(k|\tau) \ln p(k|\tau).$$

When there are two classes, the entropy function reduces to

$$i(\tau) = -p \ln p - (1-p) \ln(1-p).$$

- $p = p(1|\tau)$.

# Gini Diversity Index

Several other $\phi$-functions have also been suggested, including the **Gini diversity index**,

$$i(\tau) = \sum_{k \neq k'} p(k|\tau)p(k'|\tau) = 1 - \sum_{k} \{p(k|\tau)\}^2.$$

In the two-class case, the Gini index reduces to

$$i(\tau) = 2p(1-p).$$

This function can be motivated by considering which quadratic polynomial satisfies the above conditions for the two-class case.

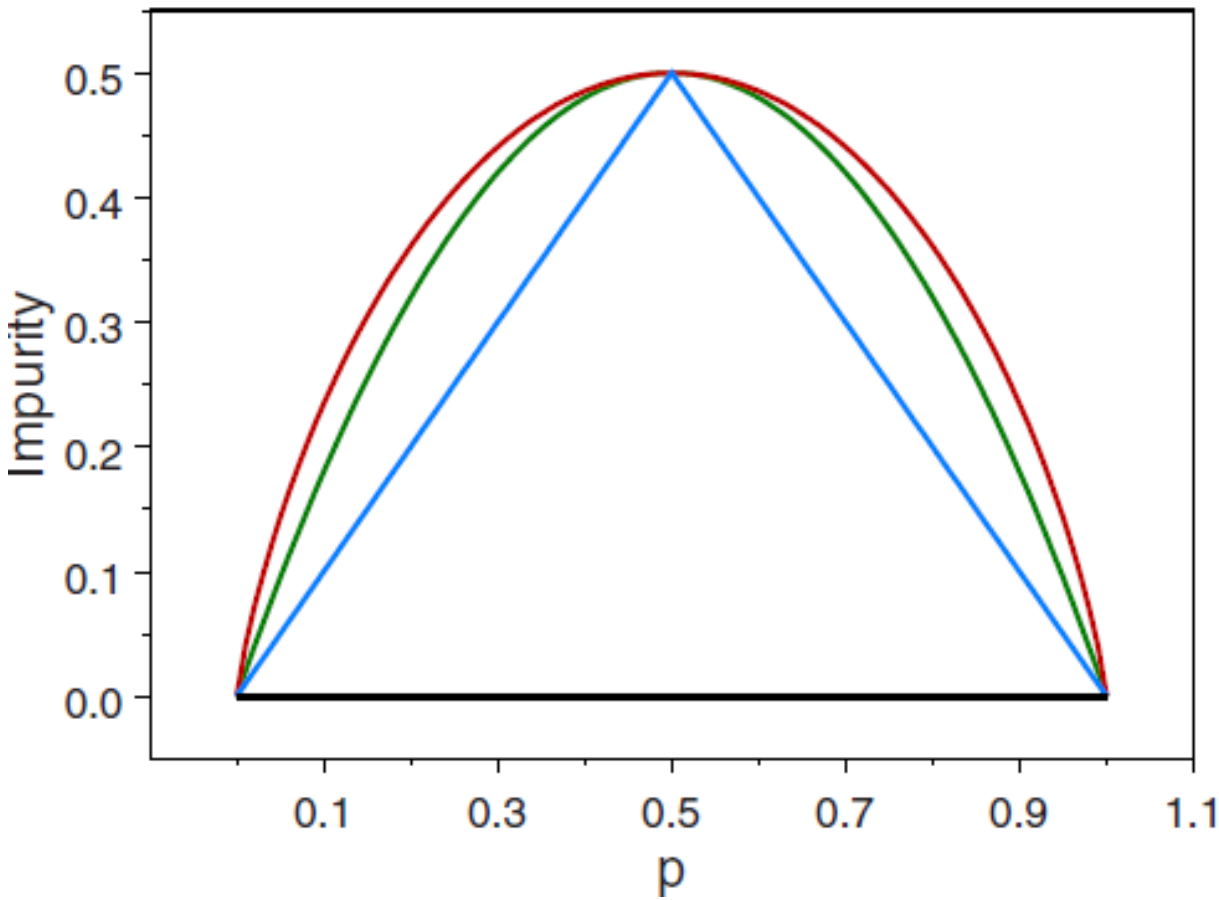


FIGURE: Node impurity functions for the two-class case. The entropy function (rescaled) is the red curve, the Gini index is the green curve, and the resubstitution estimate of the misclassification rate is the blue curve.

# Choosing the Best Split for a Variable

Suppose, at node $\tau$ , we apply split $s$ so that

- a proportion $p_L$ of the observations drops down to the left daughter-node $\tau_L$;
- the remaining proportion $p_R$ drops down to the right daughter-node $\tau_R$.

For example, suppose we have a data set in which the response variable $Y$ has two possible values, 0 and 1.

Suppose that one of the possible splits of the input variable $X_j$ is

$$X_j \leq c \quad \text{v.s.} \quad X_j > c.$$

- $c$ is some value of $X_j$.

We can obtain the following table.

TABLE: Two-by-two table for a split on the variable $X_j$, where the response variable has value 1 or 0.

| | 1 | 0 | Row Total |
|---|---|---|---|
| $X_j \leq c$ | $n_{11}$ | $n_{12}$ | $n_{1+}$ |
| $X_j > c$ | $n_{21}$ | $n_{22}$ | $n_{2+}$ |
| Column Total | $n_{+1}$ | $n_{+2}$ | $n_{++}$ |

Consider, first, the parent node $\tau$. We use the entropy function (9.3) as our impurity measure.

Estimate $p_L$ by $n_{+1}/n_{++}$ and $p_R$ by $n_{+2}/n_{++}$, and then the estimated impurity function is

$$i(\tau) = -\left(\frac{n_{+1}}{n_{++}}\right) \ln\left(\frac{n_{+1}}{n_{++}}\right) - \left(\frac{n_{+2}}{n_{++}}\right) \ln\left(\frac{n_{+2}}{n_{++}}\right).$$

**Note**

$i(\tau)$ is completely independent of the type of proposed split.

Now, for the daughter nodes, $\tau_L$ and $\tau_R$.

For $X_j \leq c$, we estimate $p_L$ by $n_{11}/n_{1+}$ and $p_R$ by $n_{12}/n_{1+}$, and for $X_j > c$, we estimate $p_L$ by $n_{21}/n_{2+}$ and $p_R$ by $n_{22}/n_{2+}$.

We then compute

$$i(\tau_L) = -\left(\frac{n_{11}}{n_{1+}}\right)\ln\left(\frac{n_{11}}{n_{1+}}\right) - \left(\frac{n_{12}}{n_{1+}}\right)\ln\left(\frac{n_{12}}{n_{1+}}\right),$$

$$i(\tau_R) = -\left(\frac{n_{21}}{n_{2+}}\right)\ln\left(\frac{n_{21}}{n_{2+}}\right) - \left(\frac{n_{22}}{n_{2+}}\right)\ln\left(\frac{n_{22}}{n_{2+}}\right).$$

The **goodness of split** $s$ at node $\tau$ is given by the reduction in impurity gained by splitting the parent node $\tau$ into its daughter nodes, $\tau_L$ and $\tau_R$,

$$\Delta i(s, \tau) = i(\tau) - p_L i(\tau_L) - p_R i(\tau_R).$$

The best split for the single variable $X_j$ is the one that has the largest value of $\Delta i(s, \tau)$ over all $s \in S_j$, the set of possible distinct splits for $X_j$.

# Recursive Partitioning

In order to grow a tree, we start with the root node, which consists of the learning set $\mathcal{L}$.

Using the goodness-of-split criterion for a single variable, the tree algorithm finds the best split at the root node for each of the variables, $X_1$ to $X_r$.

The best split $s$ at the root node is then defined as the one that has the largest value of $\Delta i(s, \tau)$ over all $r$ single-variable best splits at that node.

When those splits are completed, we continue to split each of the subsequent nodes. This sequential splitting process of building a tree layer-by-layer is called **recursive partitioning**.

If every parent node splits into two daughter nodes, the result is called a **binary tree**. If the binary tree is grown until none of the nodes can be split any further, we say the tree is **saturated**.

It is very easy in a high-dimensional classification problem to let the tree get overwhelmingly large, especially if the tree is allowed to grow until saturation.

One way to counter this type of situation is to restrict the growth of the tree. This was the philosophy of early tree-growers.

For example, we can declare a node to be **terminal** if it fails to be larger than a certain critical size; if $n(\tau) \leq n_{\text{min}}$, where $n(\tau)$ is the number of observations in node $\tau$ and $n_{\text{min}}$ is some previously declared minimum size of a node.

Because a terminal node cannot be split into daughter nodes, it acts as a brake on tree growth; the larger the value of $n_{\text{min}}$, the more severe the brake.

Another early action was to stop a node from splitting if the largest goodness-of-split value at that node is smaller than a certain predetermined limit.

These stopping rules, however, do not turn out to be such good ideas.

A better approach is to let the tree grow to saturation and then prune it back.

# Plurality Rule

How do we associate a class with a terminal node?

Suppose at terminal node $\tau$ there are $n(\tau)$ observations, of which $n_k(\tau)$ are from class $\Pi_k$, $k = 1, 2, \ldots, K$.

Then, the class which corresponds to the largest of the $\{n_k(\tau)\}$ is assigned to $\tau$.

This is called the **plurality rule**.

This rule can be derived from the Bayes's rule classifier, where we assign the node $\tau$ to class $\Pi_i$ if $p(i|\tau) = \max_k p(k|\tau)$.

If we estimate the prior probability $\Pi_k$ by $n_k(\tau)/n(\tau)$, $k = 1, 2, \ldots, K$, then this boils down to the plurality rule.

# Estimating the Misclassification Rate

The resubstitution estimate of the misclassification rate $R(\tau)$ of an observation in node $\tau$ is

$$r(\tau) = 1 - \max_k p(k|\tau).$$

For the two-class case, it reduces to

$$r(\tau) = 1 - \max(p, 1-p) = \min(p, 1-p).$$

If $p < 1/2$, the resubstitution estimate increases linearly in $p$, and if $p > 1/2$, it decreases linearly in $p$.

Because of its poor properties (e.g., nondifferentiability), the resubstitution estimate is not used much in practice.

Let $T$ be the tree classifier and let $\tilde{T} = \{\tau_1, \tau_2, \ldots, \tau_L\}$ denote the set of all terminal nodes of $T$.

We can now estimate the (unknown) true misclassification rate for a given tree $T$, as follows,

$$R(T) = \sum_{\tau \in \tilde{T}} R(\tau) P(\tau) = \sum_{l=1}^{L} R(\tau_l) P(\tau_l).$$

- $P(\tau)$ is the probability that an observation falls into node $\tau$.

If we estimate $P(\tau_l)$ by the proportion $p(\tau_l)$ of all observations that fall into node $\tau_l$, then, the resubstitution estimate of $R(T)$ is

$$R^{re}(T) = \sum_{l=1}^{L} r(\tau_l)p(\tau_l) = \sum_{l=1}^{L} R^{re}(\tau_l).$$

- $R^{re}(\tau_l) = r(\tau_l)p(\tau_l)$.

The resubstitution estimate $R^{re}(T)$, however, leaves much to be desired as an estimate of $R(T)$.

- Bigger trees (i.e., more splitting) have smaller values of $R^{re}(T)$; that is, $R^{re}(T') \leq R^{re}(T)$, where $T'$ is formed by splitting a terminal node of $T$.
- Using only the re-substitution estimate tends to generate trees that are too big for the given data.
- The re-substitution estimate $R^{re}(T)$ is a much-too-optimistic estimate of $R(T)$. More realistic estimates of $R(T)$ are given below.

# Pruning the Tree

The philosophy of growing trees is to grow the tree large and then prune off branches (from the bottom up) until the tree is the right size.

A pruned tree is a subtree of the original large tree. How to prune a tree, then, is the crucial part of the process.

Because there are many different ways to prune a large tree, we decide which is the best of those subtrees by using an estimate of $R(T)$.

# Pruning Algorithm

1. Grow a large tree, say, $T_{\max}$, where we keep splitting until the nodes each contain fewer than $n_{\min}$ observations;

2. Compute an estimate of $R(\tau)$ at each node $\tau \in T_{\max}$;

3. Prune $T_{\max}$ upwards toward its root node so that at each stage of pruning, the estimate of $R(T)$ is minimized.

Instead of using the re-substitution measure $R^{re}(T)$ as our estimate of $R(T)$, we modify it for tree pruning by adopting a regularization approach.

Let $\alpha \geq 0$ be a **complexity parameter**. For any node $\tau \in T$, set

$$R_\alpha(\tau) = R^{re}(\tau) + \alpha.$$

We define a **cost-complexity pruning measure** for a tree as follows:

$$R_\alpha(T) = \sum_{l=1}^{L} R_\alpha(\tau_l) = R^{re}(T) + \alpha|\tilde{T}|.$$

- $|\tilde{T}| = L$ is the number of terminal nodes in the subtree $T$ of $T_{\max}$.

**Note**

Think of $\alpha|\tilde{T}|$ as a penalty term for tree size, so that $R_\alpha(T)$ penalizes $R^{re}(T)$ for generating too large a tree.

For each $\alpha$, we then choose that subtree $T(\alpha)$ of $T_{\text{max}}$ that minimizes $R_\alpha(T)$:

$$R_\alpha\big(T(\alpha)\big) = \min_T R_\alpha(T).$$

If $T(\alpha)$ satisfies the above formula, then it is called a **minimizing subtree** (or an **optimally pruned subtree**) of $T_{\text{max}}$.

For any $\alpha$, there may be more than one minimizing subtree of $T_{\text{max}}$.

The value of $\alpha$ determines the tree size.

When $\alpha$ is very small, the penalty term will be small, and so the size of the minimizing subtree $T(\alpha)$, which will essentially be determined by $R^{re}\big(T(\alpha)\big)$, will be large.

For example, suppose we set $\alpha = 0$ and grow the tree $T_{\text{max}}$ so large that

- each terminal node contains only a single observation;
- each terminal node takes on the class of its solitary observation, every observation is classified correctly, and $R^{re}(T_{\text{max}}) = 0$.

So, $T_{\text{max}}$ minimizes $R_0(T)$.

As we increase $\alpha$, the minimizing subtrees $T(\alpha)$ will have fewer and fewer terminal nodes.

When $\alpha$ is very large, we will have pruned the entire tree $T_{\text{max}}$, leaving only the root node.

Note that although $\alpha$ is defined on the interval $[0, \infty)$, the number of subtrees of $T$ is finite.

Suppose that, for $\alpha = \alpha_1$, the minimizing subtree is $T_1 = T(\alpha_1)$.

As we increase the value of $\alpha$,

- $T_1$ continues to be the minimizing subtree until a certain point, say, $\alpha = \alpha_2$, is reached, and
- a new subtree, $T_2 = T(\alpha_2)$, becomes the minimizing subtree.

As we increase $\alpha$ further, the subtree $T_2$ continues to be the minimizing subtree until a value of $\alpha$ is reached, $\alpha = \alpha_3$, say, when a new subtree $T_3 = T(\alpha_3)$ becomes the minimizing subtree.

This argument is repeated a finite number of times to produce a sequence of minimizing subtrees $T_1, T_2, T_3, \ldots$

How do we get from $T_{\max}$ to $T_1$?

Suppose the node $\tau$ in the tree $T_{\max}$ has daughter nodes $\tau_L$ and $\tau_R$, both of which are terminal nodes. Then,

$$R^{re}(\tau) \geq R^{re}(\tau_L) + R^{re}(\tau_R).$$

If equality occurs in the above formula at node $\tau$, then prune the terminal nodes $\tau_L$ and $\tau_R$ from the tree.

Continue this pruning strategy until no further pruning of this type is possible. The resulting tree is $T_1$.

Next, we find $T_2$. Let

- $\tau$ be any nonterminal node of $T_1$,
- $T_\tau$ be the subtree whose root node is $\tau$,
- $\tilde{T}_\tau = \{\tau'_1, \tau'_2, \ldots, \tau'_{L_\tau}\}$ be the set of terminal nodes of $T_\tau$,
- $$R^{re}(T_\tau) = \sum_{\tau' \in \tilde{T}_\tau} R^{re}(\tau') = \sum_{l'=1}^{L_\tau} R^{re}(\tau'_{l'}).$$

Then, $R^{re}(\tau) > R^{re}(T_\tau)$.

Now, set

$$R_\alpha(T_\tau) = R^{re}(T_\tau) + \alpha|\tilde{T}_\tau|.$$

As long as $R_\alpha(\tau) > R_\alpha(T_\tau)$, the subtree $T_\tau$ has a smaller cost-complexity than its root node $\tau$, and, therefore, it pays to retain $T_\tau$.

We subsequently solving $\alpha$ yields

$$\alpha < \frac{R^{re}(\tau) - R^{re}(T_\tau)}{|\tilde{T}_\tau| - 1}.$$

So the right-hand side of the above formula, which is positive, computes the reduction in $R^{re}$ (due to going from a single node to the subtree with that node as root) relative to the increase in the number of terminal nodes.

For $\tau \in T_1$, define

$$g_1(\tau) = \frac{R^{re}(\tau) - R^{re}(T_{1,\tau})}{|\tilde{T}_{1,\tau}| - 1}, \quad \tau \notin \tilde{T}(\alpha_1).$$

- $T_{1,\tau}$ is the same as $T_\tau$.

Then, $g_1(\tau)$ can be regarded as a critical value for $\alpha$: as long as $g_1(\tau) > \alpha_1$, we do not prune the nonterminal nodes $\tau \in T_1$.

We define the **weakest-link node** $\tilde{\tau}_1$ as the node in $T_1$ that satisfies

$$g_1(\tilde{\tau}_1) = \min_{\tau \in T_1} g_1(\tau).$$

As $\alpha$ increases, $\tilde{\tau}_1$ is the first node for which $R_\alpha(\tau) = R_\alpha(T_\tau)$, so that $\tilde{\tau}_1$ is preferred to $T_{\tilde{\tau}_1}$.

Set $\alpha_2 = g_1(\tilde{\tau}_1)$ and define the subtree $T_2 = T(\alpha_2)$ of $T_1$ by pruning away the subtree $T_{\tilde{\tau}_1}$ (so that $\tilde{\tau}_1$ becomes a terminal node) from $T_1$.

The same procedure applies to obtain $T_3$.

And so on for a finite number of steps.

As we noted above, there may be several minimizing subtrees for each $\alpha$.

How do we choose between them?

For a given value of $\alpha$, we call $T(\alpha)$ the smallest minimizing subtree if it is a minimizing subtree and satisfies the following condition:

$$\text{if} \quad R_\alpha(T) = R_\alpha\big(T(\alpha)\big), \quad \text{then} \quad T \succ T(\alpha).$$

$T \succ T(\alpha)$ means that $T(\alpha)$ is a subtree of $T$ and, hence, has fewer terminal nodes than $T$.

This condition says that, in the event of any ties, $T(\alpha)$ is taken to be the smallest tree out of all those trees that minimize $R(\alpha)$.

The above construction gives us a finite increasing sequence of complexity parameters,

$$0 = \alpha_0 < \alpha_1 < \alpha_2 < \cdots < \alpha_M,$$

which corresponds to a finite sequence of nested subtrees of $T_{\text{max}}$,

$$T_{\text{max}} = T_0 \succ T_1 \succ T_2 \succ \cdots \succ T_M.$$

- $T_k = T(\alpha_k)$ is the unique smallest minimizing subtree for $\alpha \in [\alpha_k, \alpha_{k+1})$, and $T_M$ is the root-node subtree.

# Pruning Process Summary

1. Start with $T_1$ and increase $\alpha$ until $\alpha = \alpha_2$ determines the weakest-link node $\tilde{\tau}_1$.

2. Prune the subtree $T_{\tilde{\tau}_1}$ with that node as root. This gives us $T_2$.

3. Repeat this procedure by finding $\alpha = \alpha_3$ and the weakest-link node $\tilde{\tau}_2$ in $T_2$ and prune the subtree $T_{\tilde{\tau}_2}$ with that node as root. This gives us $T_3$.

4. This pruning process is repeated until we arrive at $T_M$.

# Choosing the Best Pruned Subtree

Thus far, we have constructed a finite sequence of decreasing-size subtrees $T_1, T_2, T_3, \ldots, T_M$ by pruning more and more nodes from $T_{\text{max}}$.

## Questions

- When do we stop pruning?
- Which subtree of the sequence do we choose as the best pruned subtree?

Choice of the best subtree depends upon having a good estimate of the misclassification rate $R(T_k)$ corresponding to the subtree $T_k$.

Breiman et al. (1984) offered two estimation methods: use an independent test sample or use cross-validation.

- When the data set is very large, use of an independent test set is straightforward and computationally efficient, and is, generally, the preferred estimation method.

- For smaller data sets, cross-validation is preferred.

# Independent Test Set and Cross-Validation

See the textbook.

# The One-SE Rule

To overcome possible instability in selecting the best-pruned subtree, Breiman et al. (1984, Section 3.4.3) propose an alternative rule.

Let $\hat{R}(T_*) = \min_k R(T_k)$ denote the estimated misclassification rate, calculated from either a test set or cross-validation.

Then we choose the smallest tree $T_{**}$ that satisfies the **1-SE rule**, namely,

$$\hat{R}(T_{**}) \leq \hat{R}(T_*) + \hat{\text{SE}}(\hat{R}(T_*)).$$

This rule appears to produce a better subtree than using $T_*$ because it responds to the variability (through the standard error) of the cross-validation estimates.

# Regression Trees

Suppose the data are given by $\mathcal{D} = \left\{(\mathbf{X}_i, Y_i), i = 1, 2, \ldots, n\right\}$.

- $Y_i$ are measurements made on a continuous response variable $Y$,
- $\mathbf{X}_i$ are measurements on an input $r$-vector $\mathbf{X}$.

We assume that $Y$ is related to $\mathbf{X}$ as in multiple regression, and we wish to use a tree-based method to predict $Y$ from $\mathbf{X}$.

Regression trees are constructed similarly to classification trees, and the method is generally referred to as recursive-partitioning regression.

In a classification tree, the class of a terminal node is defined as that class that commands a plurality (a majority in the two-class case) of all the observations in that node, where ties are decided at random.

In a regression tree, the output variable is set to have the constant value $Y(\tau)$ at terminal $\tau$.

Hence, the tree can be represented as an $r$-dimensional histogram estimate of the regression surface, where $r$ is the number of input variables, $X_1, X_2, \ldots, X_r$.

# The Terminal-Node Value

How do we find $Y(\tau)$?

Recall that the resubstitution estimate of prediction error is

$$R^{re}(\hat{\mu}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

- $\hat{Y}_i = \hat{\mu}(\mathbf{X}_i)$ is the estimated value of the predictor at $\mathbf{X}_i$.

For $\hat{Y}_i$ to be constant at each node, the predictor has to have the form

$$\hat{\mu}(\mathbf{X}) = \sum_{\tau \in \tilde{T}} Y(\tau) \mathsf{I}[\mathbf{X} \in \tau] = \sum_{l=1}^{L} Y(\tau_l) \mathsf{I}[\mathbf{X} \in \tau_l].$$

For $\mathbf{X}_i \in \tau_l$, $R^{re}(\hat{\mu})$ is minimized by taking $\hat{Y}_i = \bar{Y}(\tau_l)$ as the constant value $Y(\tau_l)$ where

$$\bar{Y}(\tau_l) = \frac{1}{n(\tau_l)} \sum_{\mathbf{X}_i \in \tau_l} Y_i, \quad \text{with} \quad n(\tau_l) = \sum_{i=1}^{n} \mathsf{I}[\mathbf{X}_i \in \tau_l].$$

Changing notation slightly to reflect the tree structure, the resubstitution estimate is

$$R^{re}(T) = \frac{1}{n} \sum_{l=1}^{L} \sum_{\mathbf{X}_i \in \tau_l} \left(Y_i - \bar{Y}(\tau_l)\right)^2 = \sum_{l=1}^{L} R^{re}(\tau_l),$$

$$R^{re}(\tau_l) = \frac{1}{n} \sum_{\mathbf{X}_i \in \tau_l} \left(Y_i - \bar{Y}(\tau_l)\right)^2 = p(\tau) s^2(\tau_l).$$

- $s^2(\tau_l)$ is the (biased) sample variance of all the $Y_i$ values in node $\tau_l$,
- $p(\tau_l) = n(\tau_l)/n$ is the proportion of observations in node $\tau_l$.

# Splitting Strategy

How do we determine the type of split at any given node of the tree?

We take as our splitting strategy at node $\tau \in T$ the split that provides the biggest reduction in the value of $R^{re}(T)$.

The reduction in $R^{re}(\tau)$ due to a split into $\tau_L$ and $\tau_R$ is given by

$$\Delta R^{re}(\tau) = R^{re}(\tau) - R^{re}(\tau_L) - R^{re}(\tau_R).$$

The best split at $\tau$ is the one that maximizes $\Delta R^{re}(\tau)$.

The result of employing such a splitting strategy is that the best split will divide up observations according to whether $Y$ has a small or large value.

In general, where splits occur, we see either $\bar{y}(\tau_L) < \bar{y}(\tau) < \bar{y}(\tau_R)$ or its reverse with $\bar{y}(\tau_L)$ and $\bar{y}(\tau_R)$ interchanged.

We note that finding $\tau_L$ and $\tau_R$ to maximize $\Delta R^{re}(\tau)$ is equivalent to minimizing $R^{re}(\tau_L) + R^{re}(\tau_R)$.

This boils down to finding $\tau_L$ and $\tau_R$ to solve

$$\min_{\tau_L, \tau_R} = \left\{ p(\tau_L) s^2(\tau_L) + p(\tau_R) s^2(\tau_R) \right\}.$$

- $p(\tau_L)$ and $p(\tau_R)$ are the proportions of observations in $\tau$ that split to $\tau_L$ and $\tau_R$, respectively.

# Pruning the Tree

The method for pruning a regression tree incorporates the same ideas as is used to prune a classification tree.

As before, we first grow a large tree, $T_{\text{max}}$, by splitting nodes repeatedly until each node contains fewer than a given number of observations.

That is, until $n(\tau) \geq n_{\text{min}}$ for each $\tau \in \tilde{T}$, where we typically set $n_{\text{min}} = 5$.

Next, we set up an **error-complexity measure**,

$$R_\alpha(T) = R^{re}(T) + \alpha|\tilde{T}|.$$

- $\alpha > 0$ is a complexity parameter.

Use $R_\alpha(T)$ as the criterion for deciding when and how to split, just as we did in pruning classification trees.

The result is a sequence of subtrees,

$$T_{\text{max}} = T_0 \succ T_1 \succ T_2 \succ T_3 \succ \cdots \succ T_M,$$

and an associated sequence of complexity parameters,

$$0 = \alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots < \alpha_M,$$

such that for $\alpha \in [\alpha_k, \alpha_{k+1})$, $T_k$ is the smallest minimizing subtree of $T_{\text{max}}$.

# Selecting the Best Pruned Subtree

See the textbook.

# Trees Versus Linear Models

Regression and classification trees have a very different flavor from the more classical approaches for regression and classification.

In particular, linear regression assumes a model of the form

$$f(\mathbf{X}) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j,$$

whereas regression trees assume a model of the form

$$f(\mathbf{X}) = \sum_{m=1}^{M} c_m \mathrm{I}[\mathbf{X} \in R_m].$$

- $R_1, \ldots, R_M$ represent a partition of feature space.

Which model is better?

It depends on the problem at hand.

If the relationship between the features and the response is well approximated by a linear model, then an approach such as linear regression will likely work well, and will outperform a method such as a regression tree that does not exploit this linear structure.

If instead there is a highly non-linear and complex relationship between the features and the response, then decision trees may outperform classical approaches.

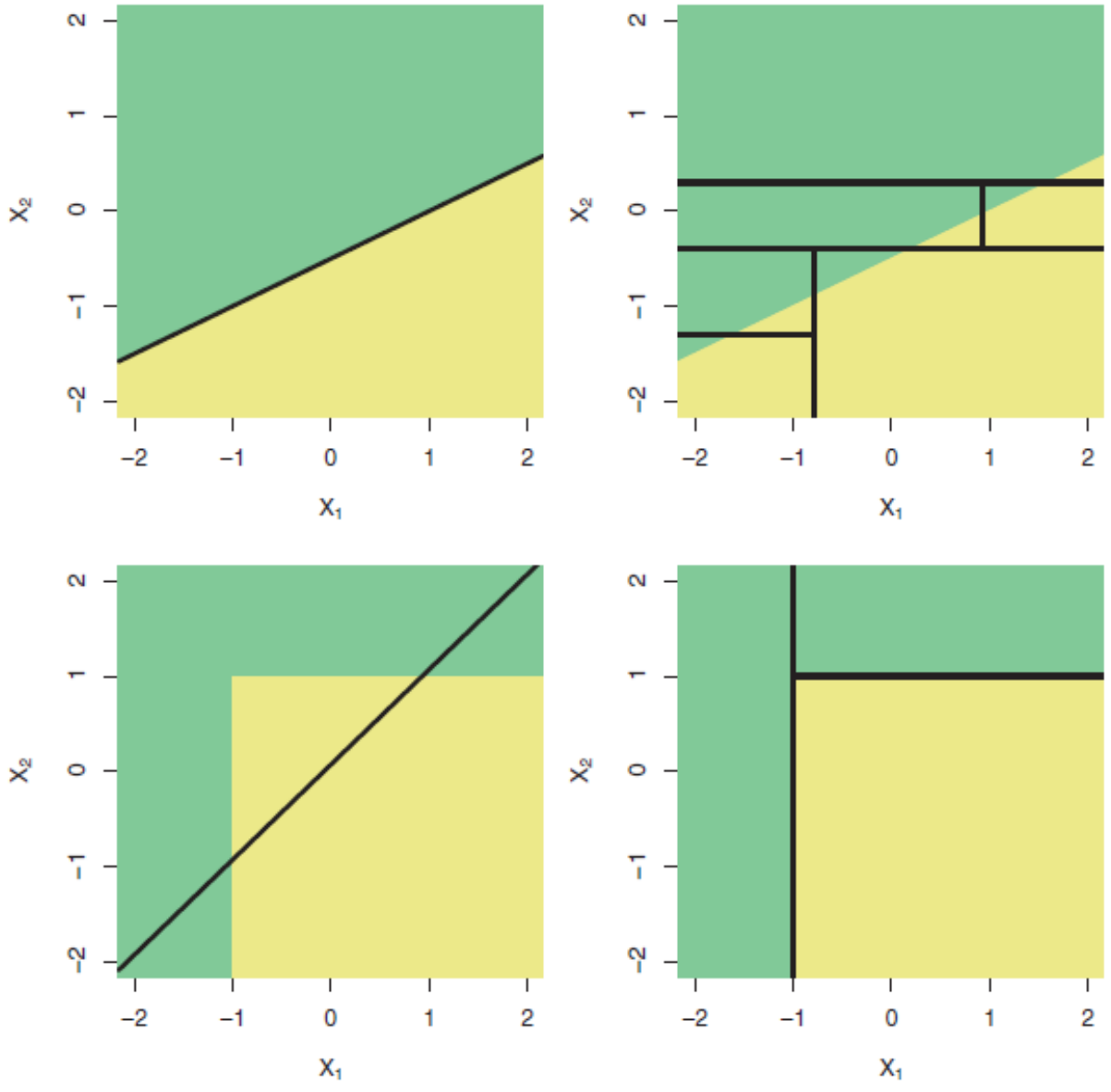


FIGURE: An illustrative example.

- Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).

- Bottom Row: Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).

The relative performances of tree-based and classical approaches can be assessed by estimating the test error, using either cross-validation or the validation set approach.

Of course, other considerations beyond simply test error may come into play in selecting a statistical learning method.

For instance, in certain settings, prediction using a tree may be preferred for the sake of interpretability and visualization.

# Multivariate Responses

Some work has been carried out on constructing classification trees for multivariate responses, especially where each response is binary.

In such cases, the measure of within-node homogeneity at node $\tau$ for a single binary variable is generalized to a scalar-valued function of a matrix argument.

Examples include $-\log|\mathbf{V}_\tau|$, where $\mathbf{V}_\tau$ is the within-node sample covariance matrix of the s binary responses at node $\tau$, and a node-based quadratic form in $\mathbf{V}$, the covariance matrix derived from the root node.

The cost-complexity of tree $T$ is then defined as $R_\alpha(T)$ in (9.19), where $R^{re}(T)$ is a within-node homogeneity measure summed over all terminal nodes.

When dealing with multivariate responses, it is clear from an applied point of view that the amount of data available for tree construction has to be very large.

# Survival Trees

See the textbook.

# MARS

Recursive partitioning used in constructing regression trees has been generalized to a flexible class of nonparametric regression models called **multivariate adaptive regression splines (MARS)**.

In the MARS approach, $Y$ is related to $\mathbf{X}$ via the model $Y = \mu(\mathbf{X}) + \epsilon$, where the error term $\epsilon$ has mean zero.

The regression function, $\mu(\mathbf{X})$, is taken to be a weighted sum of $L$ basis functions,

$$\mu(\mathbf{X}) = \beta_0 + \sum_{l=1}^{L} \beta_l B_l(\mathbf{X}).$$

The $l$th basis function,

$$B_l(\mathbf{X}) = \prod_{m=1}^{M_l} \phi_{lm}(X_{q(l,m)}),$$

is the product of $M_l$ univariate spline functions $\{\phi_{lm}(X)\}$, where $M_l$ is a finite number and $q(l, m)$ is an index depending upon the $l$th basis function and the $m$th spline function.

Thus, for each $l$, $B_l(\mathbf{X})$ can consist of a single spline function or a product of two or more spline functions, and no input variable can appear more than once in the product.

These spline functions (for $l$ odd) are often taken to be linear of the form,

$$\phi_{lm}(X) = (X - t_{lm})_+, \quad \phi_{l+1,m}(X) = (t_{lm} - X)_+.$$

- $t_{lm}$ is a **knot** of $\phi_{lm}(X)$ occurring at one of the observed values of $X_{q(l,m)}$, $m = 1, 2, \ldots, M_l$, $l = 1, 2, \ldots, L$.

If $B_l(\mathbf{X}) = \mathrm{I}[\mathbf{X} \in \tau_l]$ and $\beta_l = Y(\tau_l)$, then the regression function is equivalent to the regression-tree predictor.

Thus, whereas regression trees fit a constant at each terminal node, MARS fits more complicated piecewise linear basis functions.

Basis function are first introduced into the model in a forwards-stepwise manner.

The process starts by entering the intercept $\beta_0$ (i.e., $B_0(\mathbf{X}) = 1$) into the model, and then at each step adding one pair of terms (i.e., choosing an input variable and a knot) by minimizing an error sum of squares criterion,

$$ESS(L) = \sum_{i=1}^{n} (y_i - \mu_L(\mathbf{x}_i))^2.$$

Suppose the forwards-stepwise procedure terminates at $M$ terms.

This model is then pruned back by using a backwards-stepwise procedure to prevent possibly overfitting the data.

At each step in the backwards-stepwise procedure, we remove one term from the model. This yields $M$ different nested models.

To choose between these $M$ models, MARS uses a version of generalized cross-validation (GCV),

$$GCV(m) = \frac{n^{-1}\sum_{i=1}^{n}(y_i - \hat{\mu}_m(\mathbf{x}_i))^2}{\left(1 - \frac{C(m)}{n}\right)^2}, \quad m = 1, 2, \ldots, M.$$

- $\hat{\mu}_m(\mathbf{x})$ is the fitted value of $\mu(\mathbf{x})$ based upon $m$ terms,
- the numerator is the apparent error rate (or resubstitution error rate),
- $C(m)$ is a complexity cost function that represents the effective number of parameters in the model.

The best choice of model has $m^* = \arg\min_m GCV(m)$ terms.

# Missing Data

In some classification and regression problems, there may be missing values in the test set.

Fortunately, there are a number of ways of dealing with missing data when using tree-based methods.

One obvious way is to drop a future observation with a missing data value (or values) down the tree constructed using only complete-data observations and see how far it goes.

If the variable with the missing value is not involved in the construction of the tree, then the observation will drop to its appropriate terminal node, and we can then classify the observation or predict its $Y$ value.

If, on the other hand, the observation cannot drop any further than a particular internal node $\tau$ (because the next split at $\tau$ involves the variable with the missing value), we can either stop the observation at $\tau$ or force all the observations with a missing value for that variable to drop down to the same daughter node.

A method of **surrogate splits** has been proposed to deal with missing data.

The idea of a surrogate split at a given node $\tau$ is that we use a variable that best predicts the desired split as a substitute variable on which to split at node $\tau$.

If the best-splitting variable for a future observation at $\tau$ has a missing value at that split, we use a surrogate split at $\tau$ to force that observation further down the tree, assuming, of course, that the variable defining the surrogate split has complete data.

If the missing data occur for a nominal input variable with $L$ levels, then we could introduce an additional level of missing or NA so that the variable now has $L + 1$ levels.

# Advantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression.

- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.

# Advantages of Trees

- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).

- Trees can easily handle qualitative predictors without the need to create dummy variables.

# Disadvantages of Trees

Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

However, by aggregating many decision trees, using methods like **bagging**, **random forests**, and **boosting**, the predictive performance of trees can be substantially improved.