Machine Learning I

Tree-based methods

Souhaib Ben Taieb

University of Mons

Table of contents

Table of contents

Tree-based methods

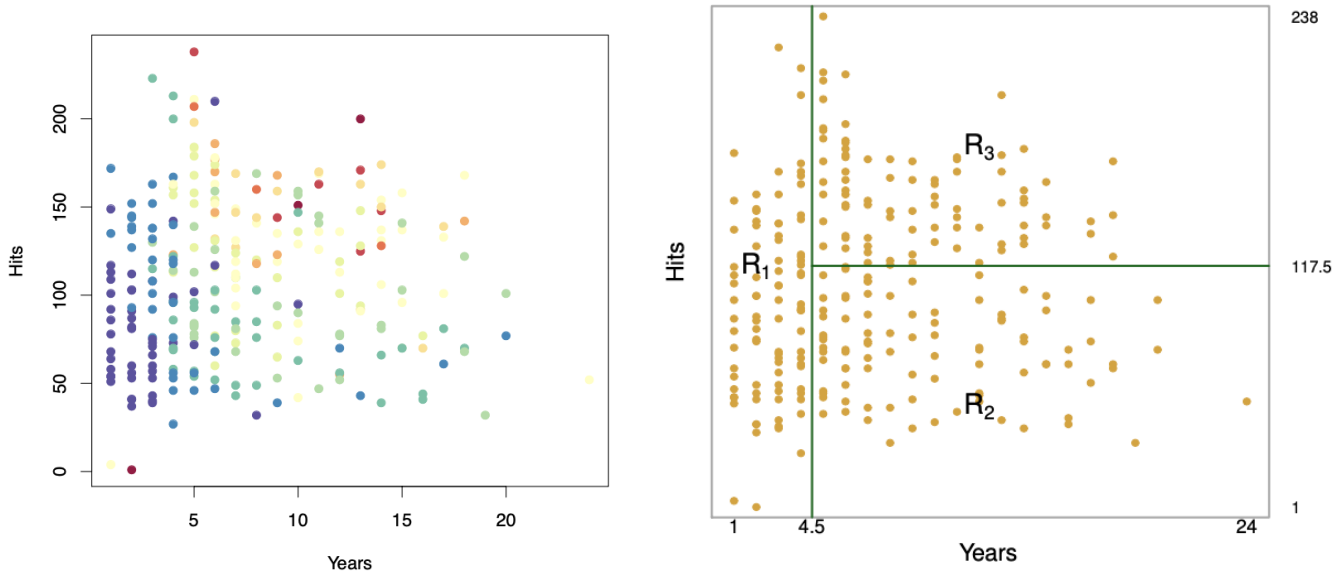
- Tree-based methods involve **stratifying** or **segmenting** the input space into a number of simple **regions**.
- Since we can represent the set of splitting rules to segment the input space in a **tree**, these types of methods are known as **decision-tree** methods.
- Tree-based methods can be used for both **regression** and **classification** problems.
- Tree-based methods are simple and useful for **intepretation**.
- A single tree is often not competitive with the best supervised learning methods. However, we can obtain significant improvement in prediction accuracy by **combining** a large number of trees (see bagging and random forests).

Table of contents

Table of contents

Example - Baseball salary data

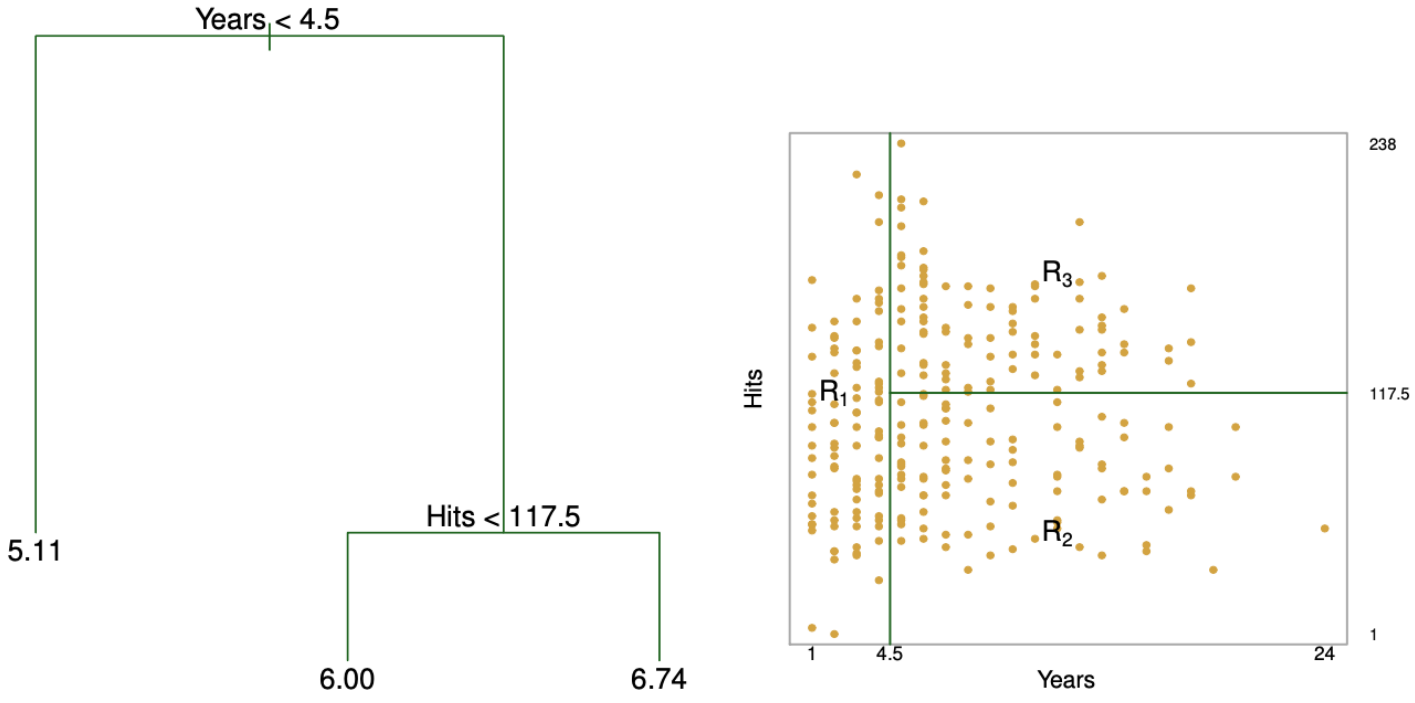
Predict log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of Ihits.



We split the input space into three regions:

- $R_1 = \{(\text{Years}, \text{Hits}) | \text{Years} < 4.5\}$
- $R_2 = \{(\text{Years}, \text{Hits}) | \text{Years} \geq 4.5, \text{Hits} < 117.5\}$
- $R_3 = \{(\text{Years}, \text{Hits}) | \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$

Example - Baseball salary data

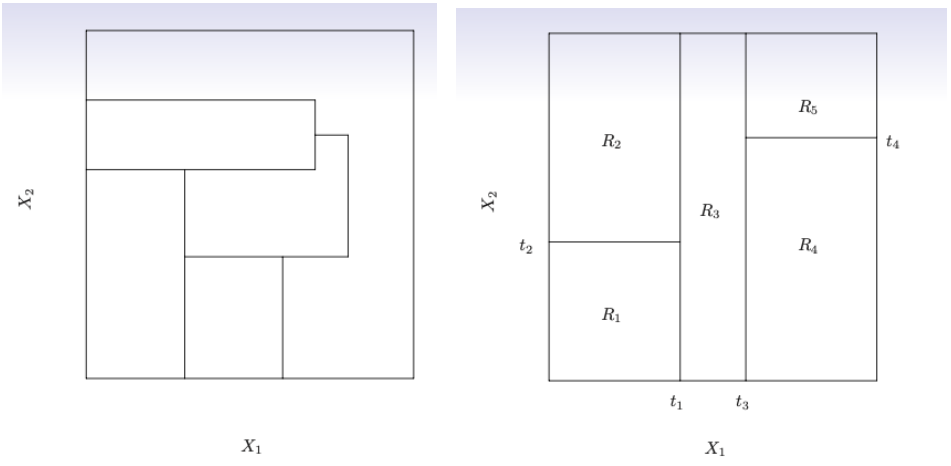


- R_1, R_2 and R_3 are known as **terminal nodes/leaves**.
- Years < 4.5 and Hits < 117.5 are **internal nodes**
- The number in each leaf is the **predicted salary** for all observations that fall there.
- How do we interpret the tree?

Table of contents

Details of the tree-building process

- We divide the input space into J **distinct** and **non-overlapping** regions, $R_1, r_2, \ldots, R_J$.
- While the regions could have any shape, **rectangles** (or boxes) are used for simplicity and for ease of interpretation.



- For every observation that falls into the region R_j, we make the **same prediction**: In regression, it is the mean of the output values for the observations in R_j. Why?

Details of the tree-building process

In other words, given a partition $R_1, R_2, \ldots, R_J$, and the associated constants $c_1, c_2, \ldots, c_J$, the hypothesis can be written as

$$h(x) = \sum_{j=1}^{J} c_j \; I(x \in R_j), .$$

i.e. $h(x) = c_j$ in each region R_j.

In regression, we want to minimize the RSS. In other words, given a partition $R_1, R_2, \ldots, R_J$, we want to compute:

$$\underset{c_1, c_2, \ldots, c_J, c_j \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - h(x_i))^2 := \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - c_j)^2.$$

Details of the tree-building process

It suffices to solve it for each partition R_j independently, which gives

$$\hat{y}_{R_j} = \text{argmin}_{c \in \mathbb{R}} \sum_{i \in R_j} (y_i - c)^2 = \frac{1}{|R_j|} \sum_{i \in R_j} y_i, \quad j = 1, 2, \ldots, J.$$

In other words, given a partition $R_1, R_2, \ldots, R_J$, the RSS is given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2.$$

How do we construct the regions $R_1, ..., R_J$?

Details of the tree-building process

- It is intractable to consider every possible partition of the feature space into J rectangles.
- **Recursive binary splitting** is a **top-down** and **greedy** approach.
- **Top-down**: starts at the top of the tree and successively split the input space. Each split add two new branches to the tree.
- **Greedy**: At each step, we choose the best split without looking ahead.

Details of the tree-building process

- We start with a **single region** (the entire input space), and we consider all the possible cutpoints for all the input variables.
- We select the input variable X_j and the cutpoint s such that splitting the input space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. We now have **two regions**.
- Next, we repeat the process for each of the two regions, selecting the best variable and best cutpoint to split one of these two regions. We now have **three regions**.
- We repeat the process again to split one of these three regions further, so as to minimize the RSS.
- We continue until a **stopping criterion** is reached; for example, we continue until no region contains more than five observations.

Details of previous figure

Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

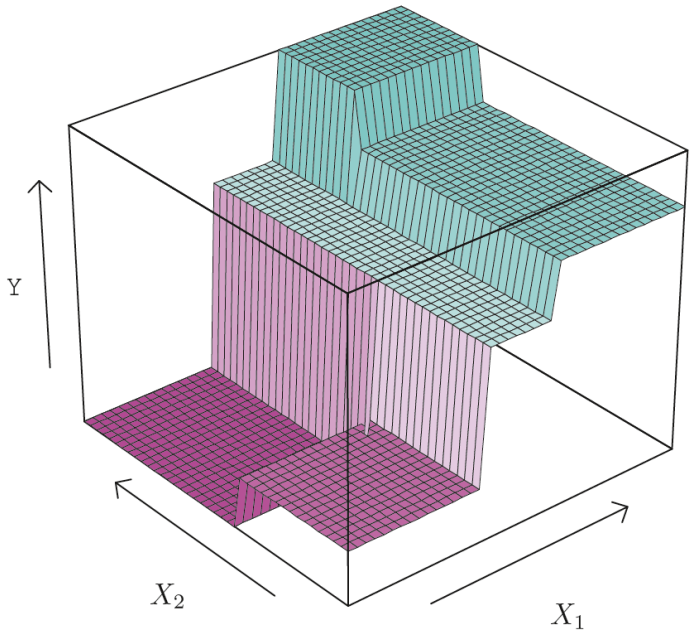
Top Right: The output of recursive binary splitting on a two-dimensional example.

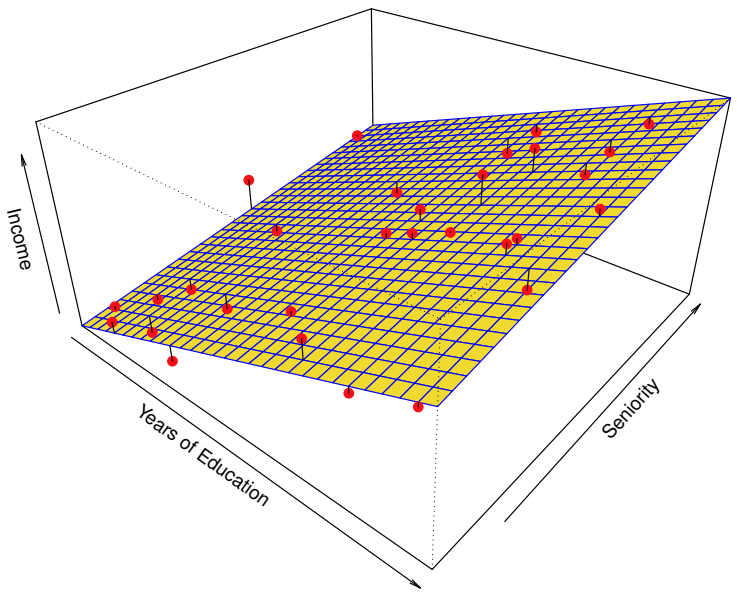
Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Regression trees versus linear models

$$h(X) = \sum_{j=1}^{J} c_j \; I(X \in R_j) \qquad h(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$





A constant model would be $h(X) = c$ where $c \in \mathbb{R}$.

Table of contents

Pruning a tree

- Recursive binary splitting is likely to **overfit** the data, leading to poor out-of-sample performance. Why?
- A smaller tree with fewer splits (i.e. fewer regions) might lead to **lower variance** at the cost of a little bias.
- We could split only if the decrease in RSS is **substantial**. However, a seemingly worthless split early on might be followed by a very good split.
- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a **subtree**. We can do this with the **weakest link pruning** algorithm.

Weakest link pruning

1. Starting with with the initial full tree T_0, replace a subtree with a leaf node to obtain a new tree T_1. Select subtree to prune by minimizing

$$\frac{\text{RSS}(T_1) - \text{RSS}(T_0)}{|T_1| - |T_0|}$$

2. Iterate this pruning to obtain a sequence $T_0, T_1, T_2, \ldots, T_R$ where T_R is the tree with a single leaf node.
3. Select the optimal tree T_j by cross validation

Cost complexity pruning

For a given subtree $T \subset T_0$, consider the following objective function

$$\sum_{j=1}^{|T|} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|,$$

where

- α is a nonnegative tuning parameter
- $|T|$ indicates the number of terminal nodes of the tree T

It can be shown that the solution for each α is among $T_0, T_1, T_2, \ldots, T_R$ from weakest link pruning.

- α controls the trade-off between the tree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation
- Then, using the full dataset, we compute the subtree corresponding to $\hat{\alpha}$

Summary: tree algorithm

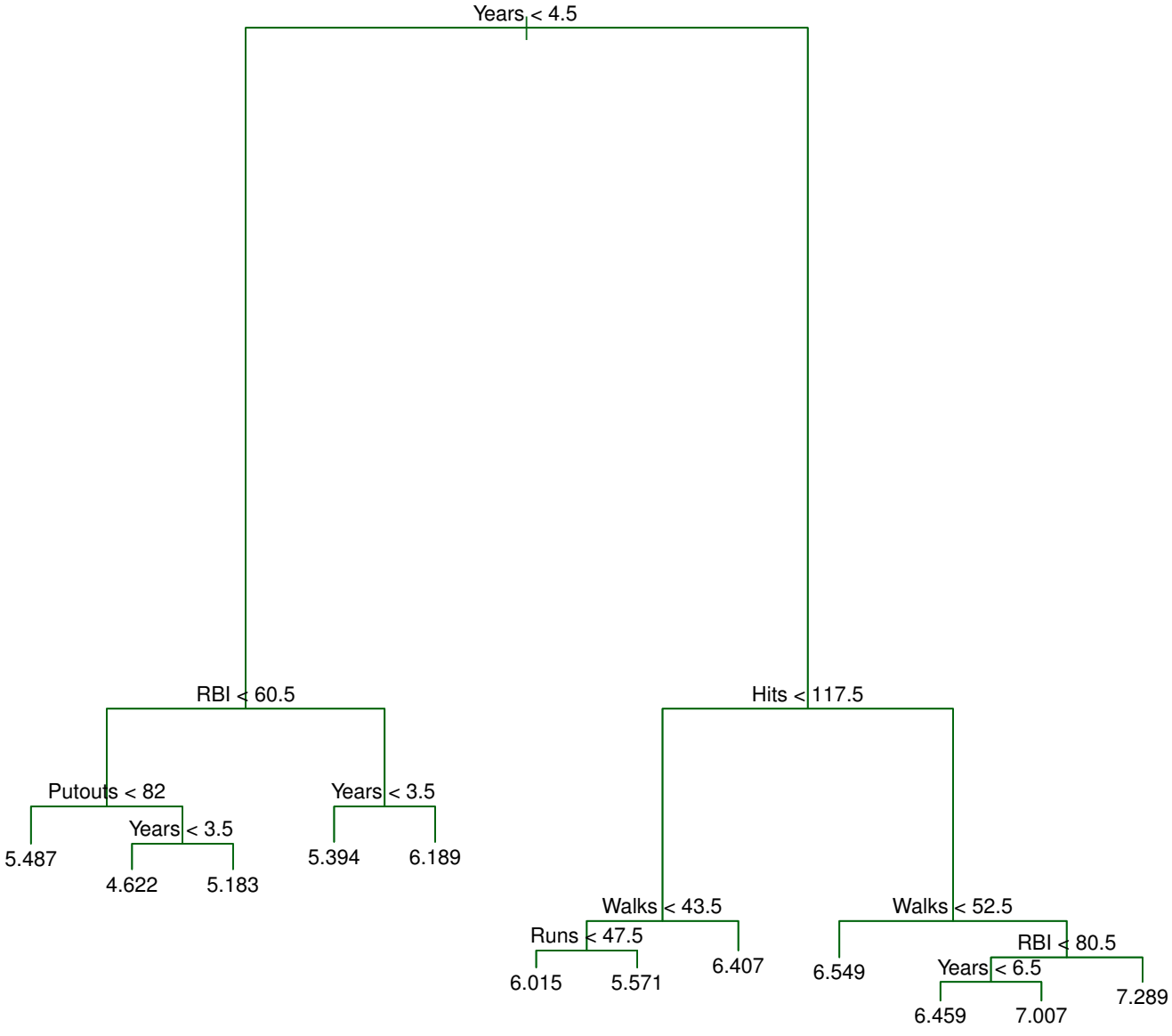
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



Baseball example continued

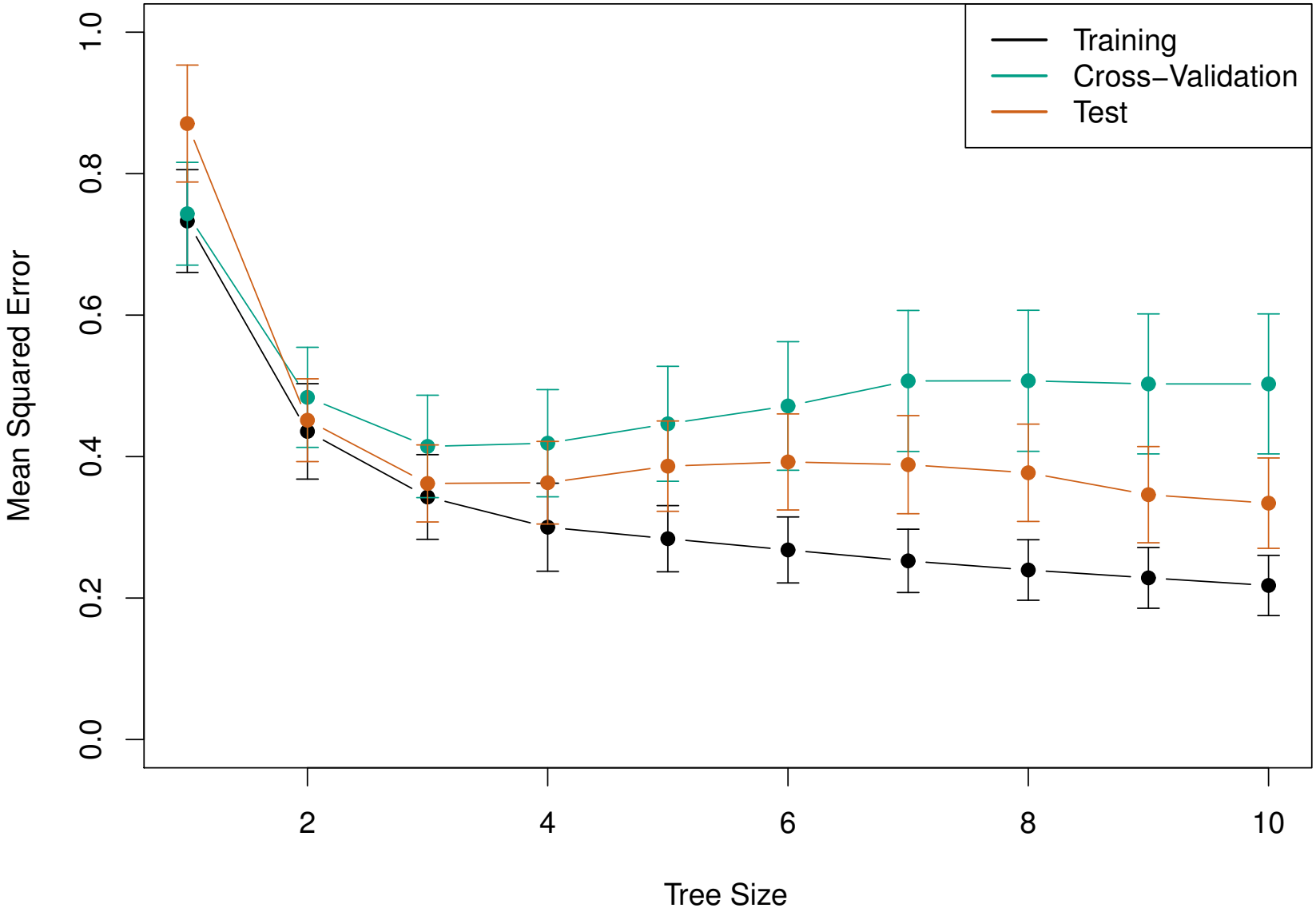


Table of contents

Table of contents

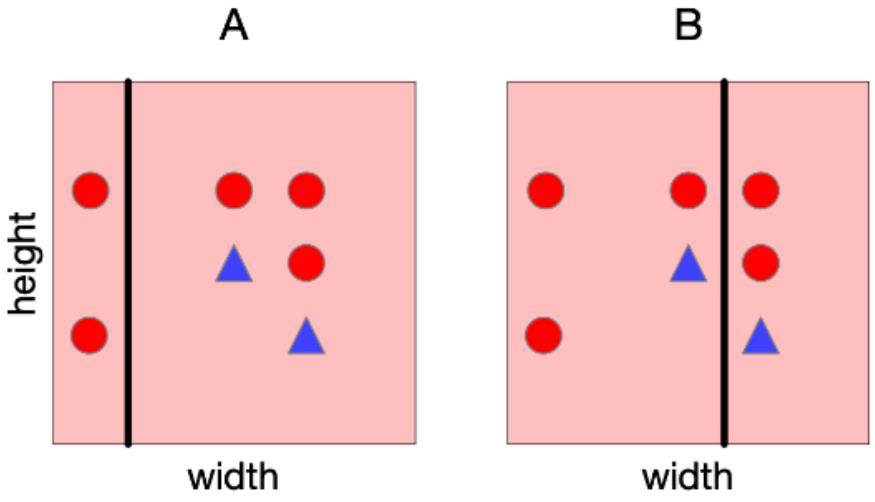
Classification trees

- In each region, we predict the **most commonly occurring class**.
- As in the regression setting, we use **recursive binary splitting** to grow a classification tree.
- A natural alternative to RSS is the **misclassification rate**, i.e. we compute the fraction of observations that do not belong to the most common class in each region.
- Let $\hat{p}_{mk}$ represent the proportion of training observations in the m-th region that are from the k-th class. The **misclassification rate** is given by

$$1 - \max_k(\hat{p}_{mk}).$$

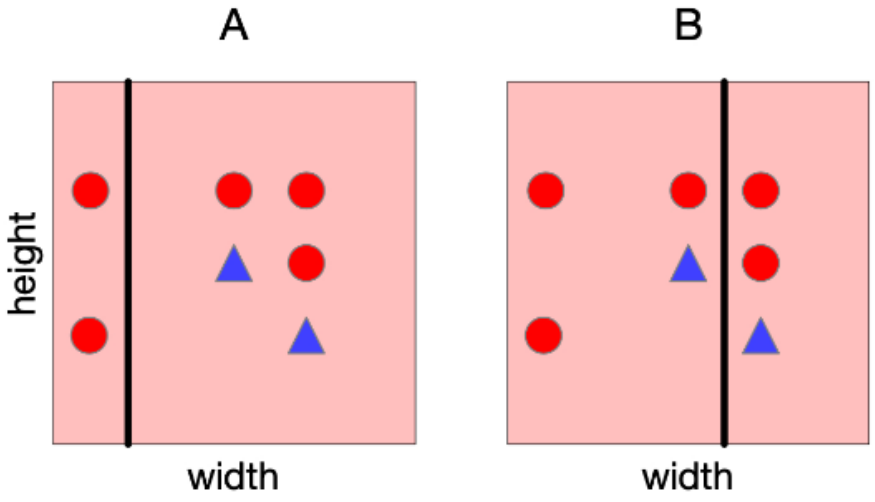
Does it work well?

Classification error for tree-growing



A and B have the same misclassification rate, so which is the best split?

Classification error for tree-growing



A and B have the same misclassification rate, so which is the best split?

- A feels like a better split, because the left-hand region is **very certain** about whether the response is red.
- Classification error is not **sufficiently sensitive** for **tree-growing,** and in practice other measures are preferable.

How to choose a good split?

- How can we quantify uncertainty in prediction for a given leaf node/region?
  - If all examples in leaf have same class: **good, low uncertainty**
  - If each class has same amount of examples in leaf: **bad, high uncertainty**
- At a given leaf node m, the main idea will be to compute an empirical PMF, i.e. $\hat{p}_{mk}$ for each class k and use a probabilistic notion of uncertainty to decide splits.

Gini index and Deviance

- The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is *cross-entropy*, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

- It turns out that the Gini index and the cross-entropy are very similar numerically.

Gini index, deviance (cross-entropy) and misclassification error

$K = 2$ (Binary case)

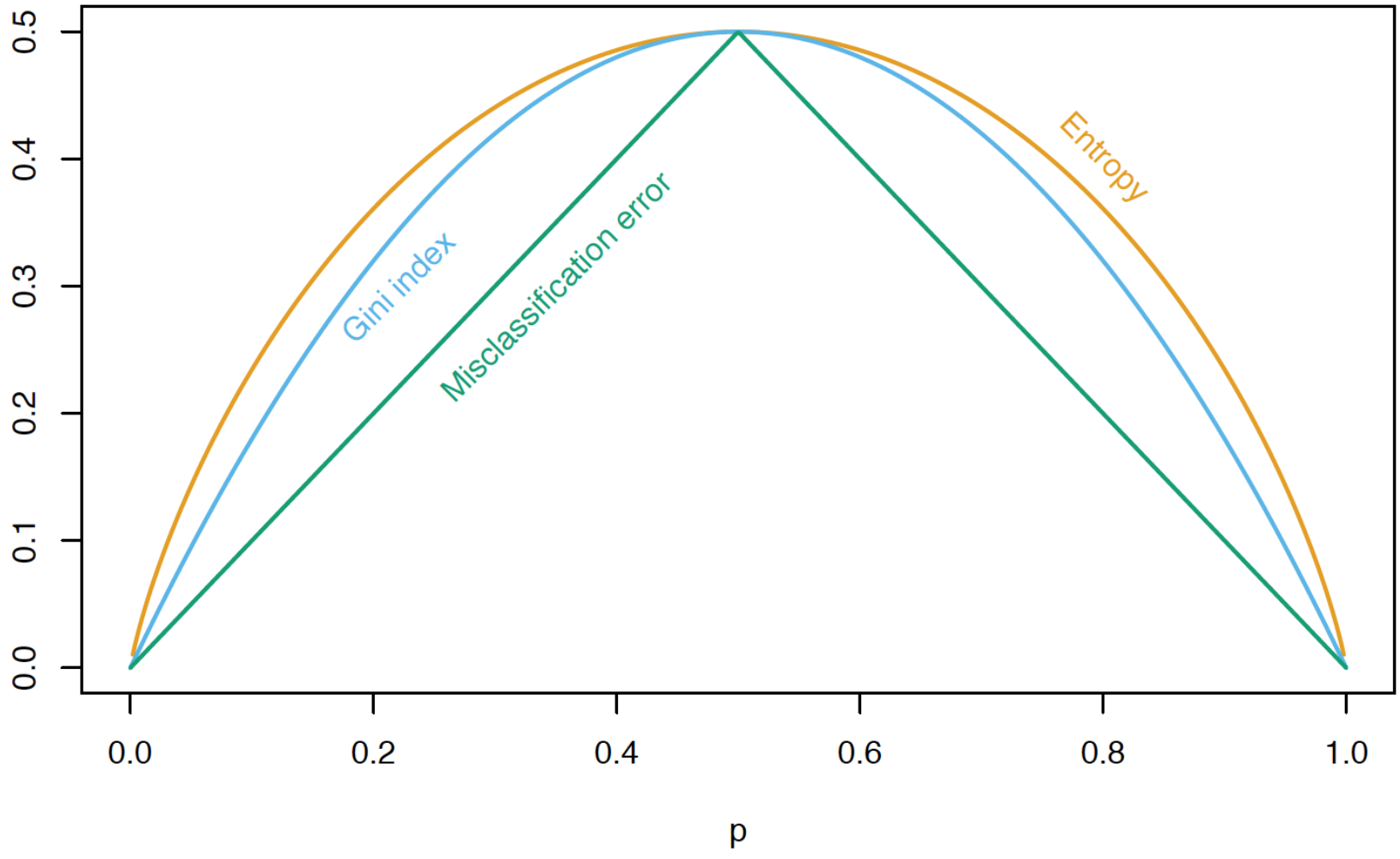


Table of contents

Quantifying uncertainty with the entropy

- The **entropy** of a discrete random variable Y is a number that quantifies the uncertainty inherent in its possible outcomes. It is given by

$$H(Y) = -\sum_{y \in \mathcal{Y}} p(y) \log_2 p(y).$$

- Entropy can also be seen as the expected information content of a random draw from the probability distribution
- **High entropy**
  - Uniform-like distribution over many outcomes
  - Values sampled from it are less predictable
- **Low entropy**
  - Distribution is concentrated on only a few outcomes
  - Values sampled from it are more predictable

Example

- The entropy of a loaded coin with probability p of heads is given by

$$-p \log_2 p - (1-p) \log_2 (1-p)$$

- With $p = 1/9$, we have

$$-\frac{1}{9} \log_2 \frac{1}{9} - \frac{8}{9} \log_2 \frac{8}{9} \approx 0.5$$

- With $p = 4/9$, we have

$$-\frac{4}{9} \log_2 \frac{4}{9} - \frac{5}{9} \log_2 \frac{5}{9} \approx 0.99$$

- In the extreme case $p = 0$ or $p = 1$, we were certain of the outcome before observing. So, we gained no certainty by observing it, i.e., entropy is 0.

Conditional entropy

The **conditional entropy** of Y given $X = x$ is given by

$$H(Y|X = x) = -\sum_{y \in \mathcal{Y}} p(y|x) \log_2 p(y|x).$$

It gives the reduction in our uncertainty about Y after observing X.

The **expected conditional entropy** is defined as

$$H(Y|X) = \mathbb{E}_x[H(Y|x)] = \sum_{x \in \mathcal{X}} p(x)\ H(Y|X = x) \tag{1}$$

$$= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log_2 p(y|x). \tag{2}$$

Useful properties:

- $H(Y|X) \leq H(Y)$
- $H(Y|Y) = 0$
- if Y and X are independent, $H(Y|X) = H(Y)$.

Conditional entropy - Example

Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

- What is the entropy of cloudiness Y , given that **it is raining**?
  - $H(Y|X = \text{Raining}) \approx 0.24$
- What is the entropy of cloudiness, given the knowledge of **whether or not it is raining**?
  - $H(Y|X) \approx 0.75$

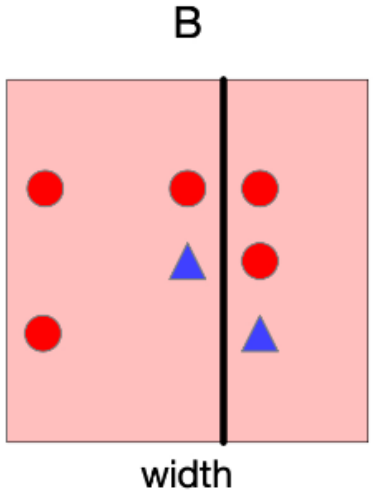
Information gain

- The **information gain** in Y due to X, or the mutual information of Y and X is given by

$$\mathrm{IG}(Y|X) = H(Y) - H(Y|X),$$

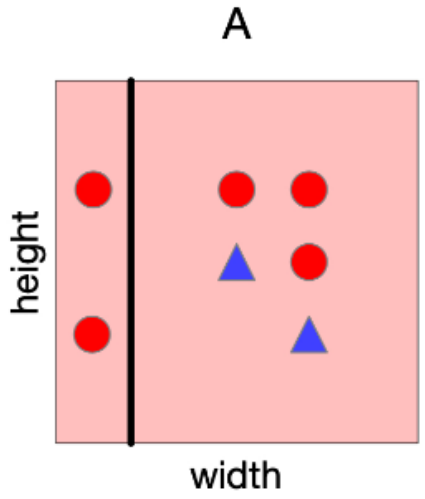
  i.e. my uncertainty in Y minus my expected uncertainty that would remain in Y given X.
- If X is completely **uninformative** about Y : $\mathrm{IG}(Y|X) = 0$.
- If X is completely **informative** about Y : $IG(Y|X) = H(Y)$.
- Information gain measures the informativeness of a variable, which is exactly what we desire in a **decision tree split**!
- The **information gain of a split**: how much information (over the training set) about the class label Y is gained by knowing which side of a split you're on.

Information gain - Example



- Root entropy: $H(Y) = -\frac{2}{7}\log_2\frac{2}{7} - \frac{5}{7}\log_2\frac{5}{7} \approx 0.86$
- Leaf conditional entropy (assuming we split at width $= 2$) :
  - $H(Y|\text{width} < 2) \approx 0.81$
  - $H(Y|\text{width} > 2) \approx 0.92$
- IG(split) $\approx 0.86 - (\frac{4}{7} \times 0.81 + \frac{3}{7} \times 0.92) \approx 0.006$

Information gain - Example



- Root entropy: $H(Y) = -\frac{2}{7}\log_2\frac{2}{7} - \frac{5}{7}\log_2\frac{5}{7} \approx 0.86$
- Leaf conditional entropy (assuming we split at width $= 0.5$):
  - $H(Y|\text{width} < 0.5) \approx 0$
  - $H(Y|\text{width} > 0.5) \approx 0.97$
- IG(split) $\approx 0.86 - (\frac{2}{7} \times 0 + \frac{5}{7} \times 0.97) \approx 0.17$.

$\rightarrow$ A split is better than B split!

Gini index, deviance (cross-entropy) and misclassification error

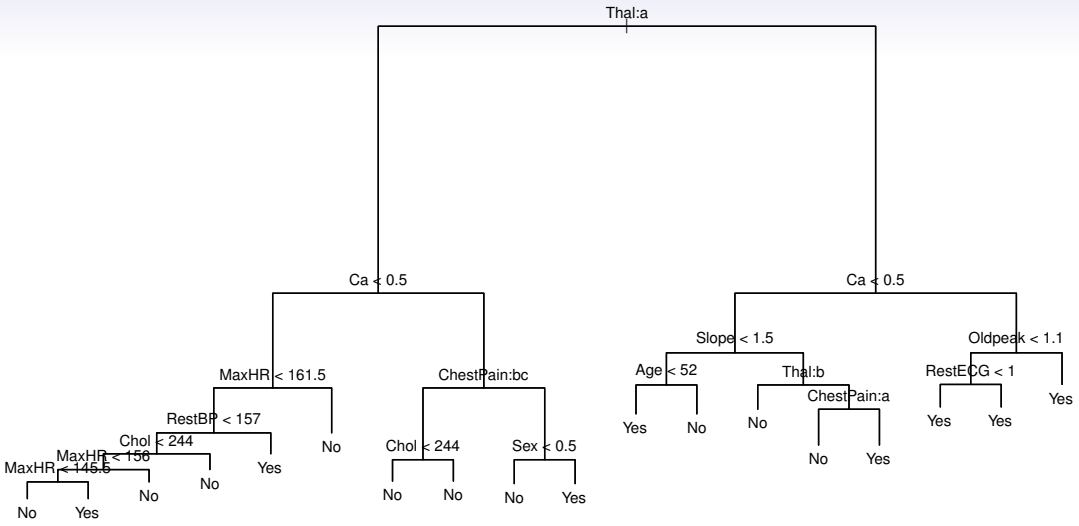
- When building a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split.
- Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

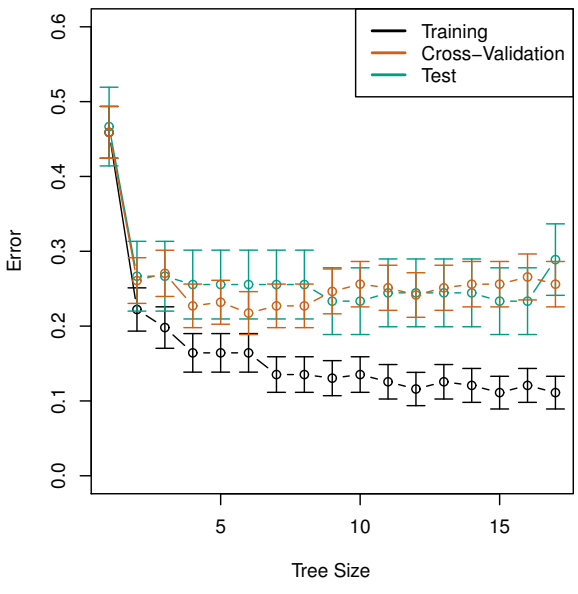
Note on categorical predictors

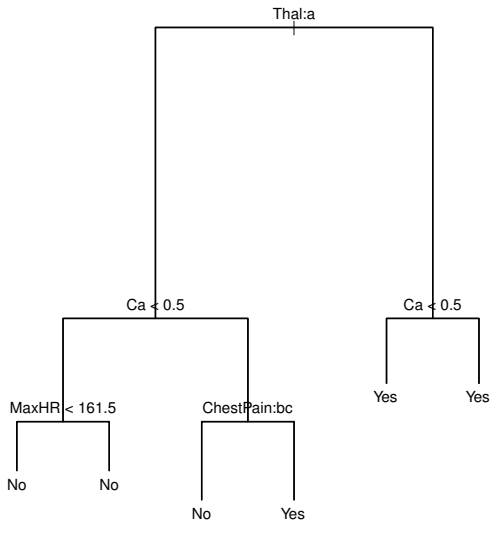
- So far, we have assumed that the predictors take on continuous values. However, decision trees can be constructed even in the presence of **qualitative/categorical predictors** using **one-hot encoding**.
- A split on one of these variables amounts to assigning some of the qualitative values to one branch and assigning the remaining to the other branch.

Example: heart data

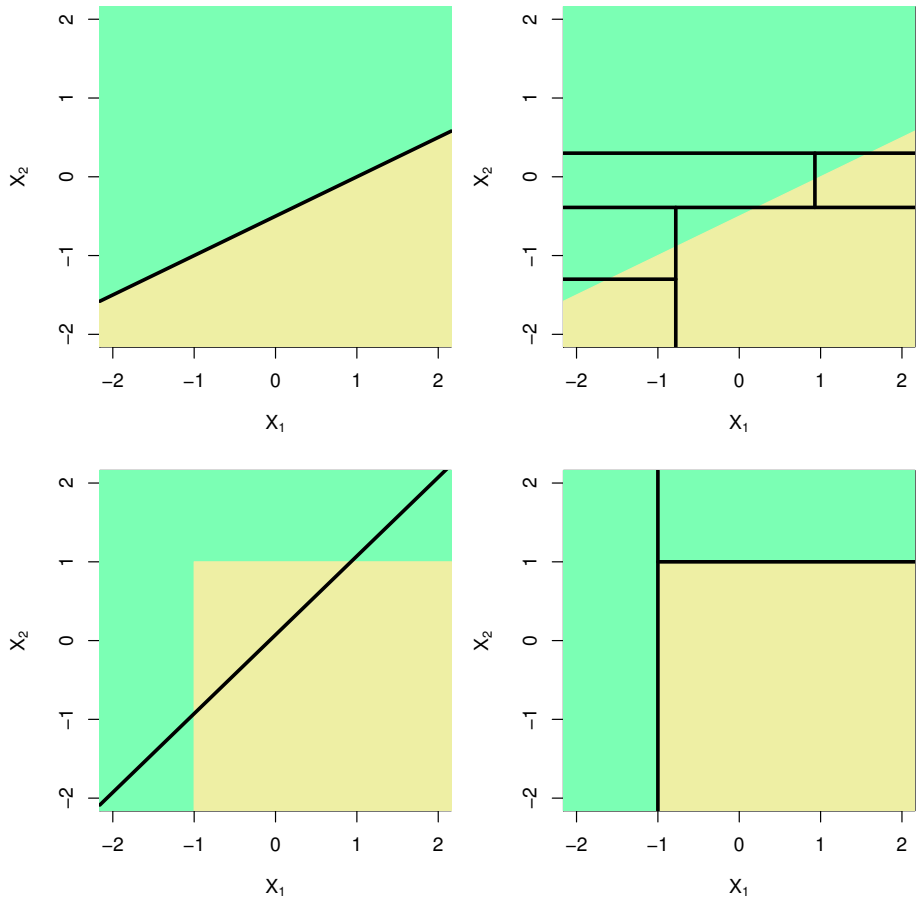
- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
RestBP < 157
Chol < 244
Sex < 0.5
Age < 52
Thal:b
RestECG < 1
MaxHR < 156
MaxHR < 145.5
ChestPain:a
No
Yes


Training
Cross−Validation
Test
Error
Tree Size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Yes
No

Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

Advantages and Disadvantages 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.
- ▲ 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.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Table of contents

Bagging

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $h^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

$$h_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} h^{*b}(x).$$

- This is called *Bagging*

Bagging classification trees

- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Bagging with trees

- The trees are grown deep, and are **not pruned**.
- Each individual tree has **high variance, but low bias**. Why?
- Averaging these B trees **reduces the variance**. Why?

Bagging with trees - Bias and variance tradeoff

$$h_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} h^{*b}(x)$$

$$\text{MSE} = \text{NOISE} + \text{BIAS}^2 + \text{VARIANCE}$$

- BIAS: If trees are sufficiently deep, they have very small bias. Why?
- $\text{VARIANCE} = \text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} h^{*b}(x)\right) = \ ?$

Variance of sum of i.d. and correlated variables

- Given a set of B independent random variables $Z_1, \ldots, Z_B$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by $\frac{\sigma^2}{B}$.
- If the variables are simply i.d. (identically distributed, but not necessarily independent) with positive pairwise correlation ρ ($\rho > 0$), then

$$\text{Var}(\bar{Z}) = \rho\sigma^2 + \sigma^2 \frac{1-\rho}{B}$$

Variance of sum of i.d. and correlated variables

$$
\begin{aligned}
\mathrm{Var}(\bar{Z}) &= \mathrm{Var}\left(\frac{\sum_{b=1}^{B} Z_b}{B}\right) \\
&= \frac{\sum_{i=1}^{B}\sum_{j=1}^{B}\mathrm{Cov}(Z_i, Z_j)}{B^2} \\
&= \frac{\sum_{b=1}^{B}\mathrm{Var}(Z_b) + 2\sum_{i<j}\mathrm{Cov}(Z_i, Z_j)}{B^2} \\
&= \frac{B\sigma^2 + B(B-1)\rho\sigma^2}{B^2} \\
&= \rho\sigma^2 + \sigma^2\frac{1-\rho}{B},
\end{aligned}
$$

since $\mathrm{Cov}(Z_i, Z_j) = \rho\sigma^2$.

Variance of sum of i.d. and correlated variables

$$\mathrm{Var}(\bar{Z}) = \rho\sigma^2 + \sigma^2\frac{1-\rho}{B}$$

- $\rho\sigma^2$: decreases if ρ decreases
- $\sigma^2\frac{1-\rho}{B}$: decreases if B increases (irrespective of ρ)

Table of contents

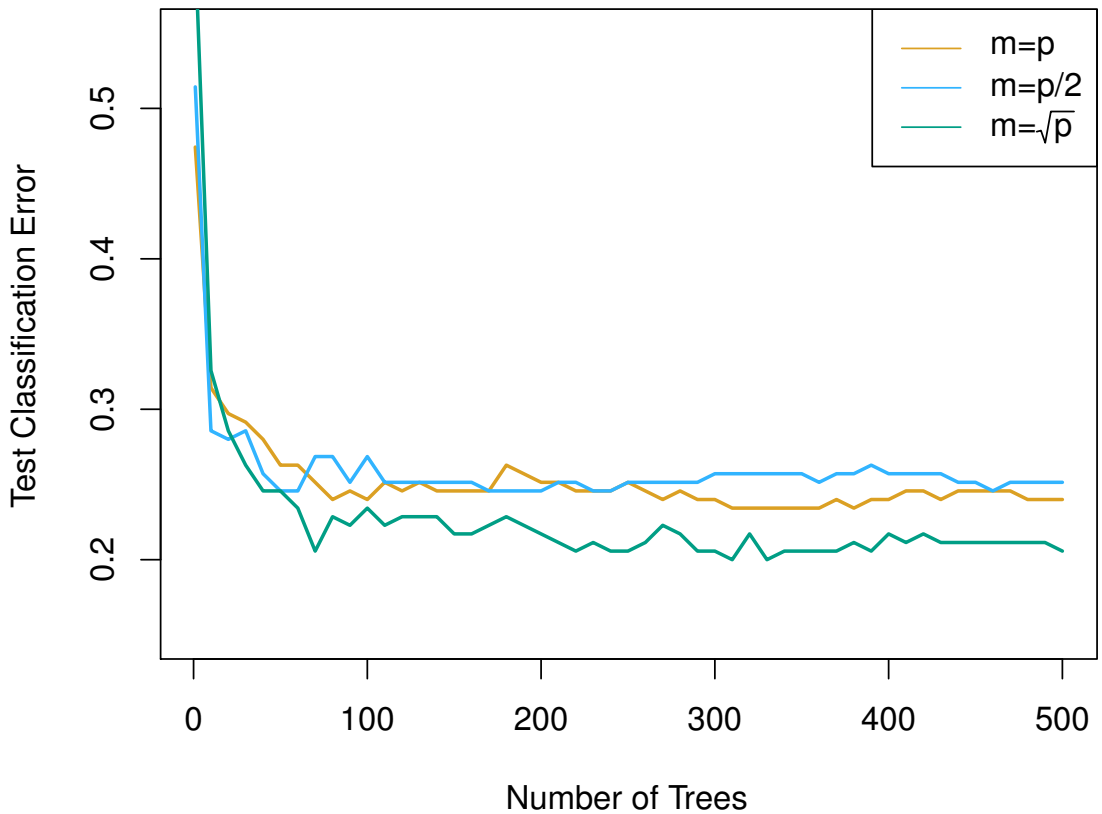
Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random selection of m predictors* is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Results: gene expression data



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Table of contents

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

Boosting algorithm for regression trees

1. Set $\mathrm{h}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - 2.1 Fit a tree h^b with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\mathsf{h}(x) \leftarrow \mathsf{h}(x) + \lambda\, \mathsf{h}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda\, \mathsf{h}^b(x_i).$$

3. Output the boosted model.

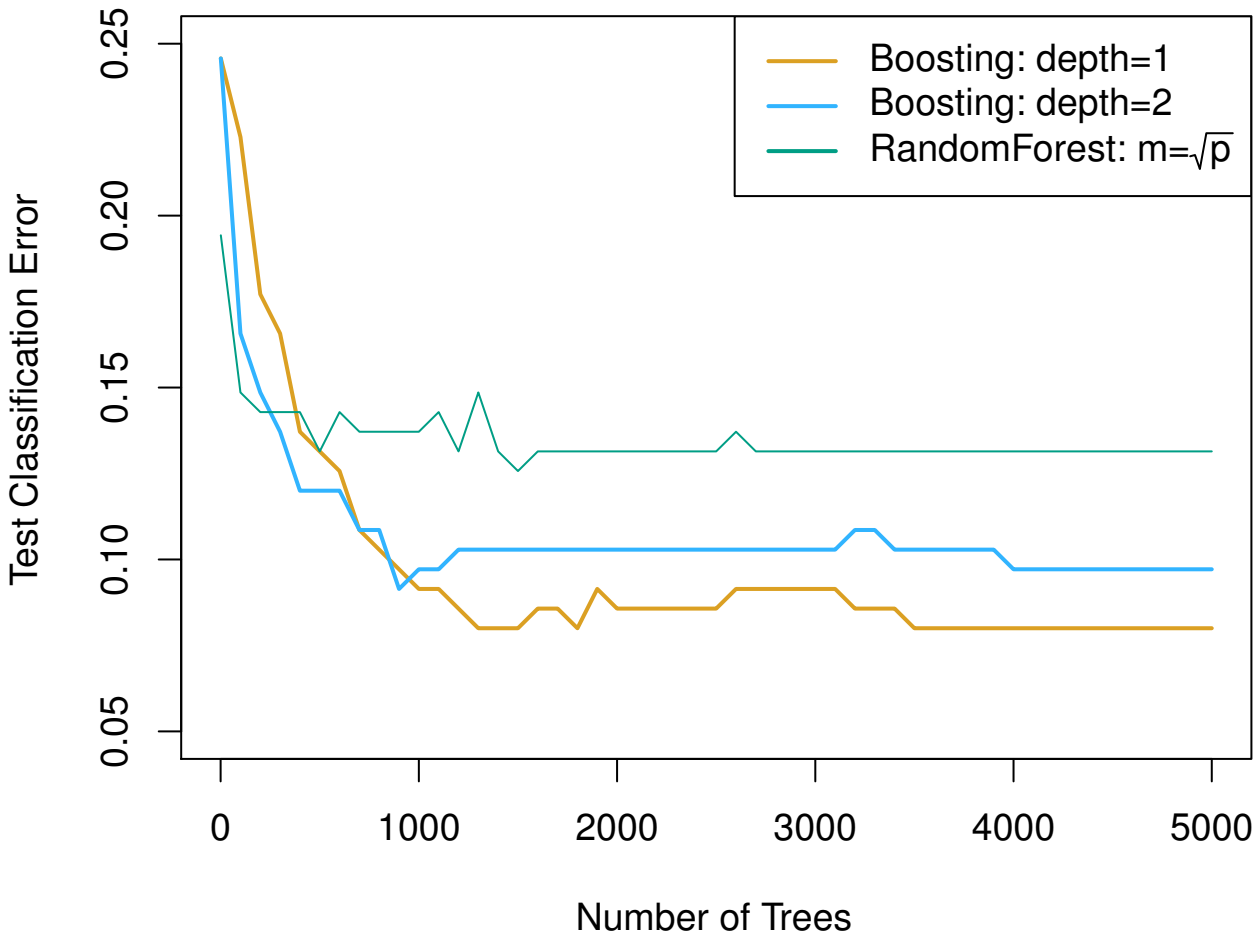
What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve h in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here.

Gene expression data continued



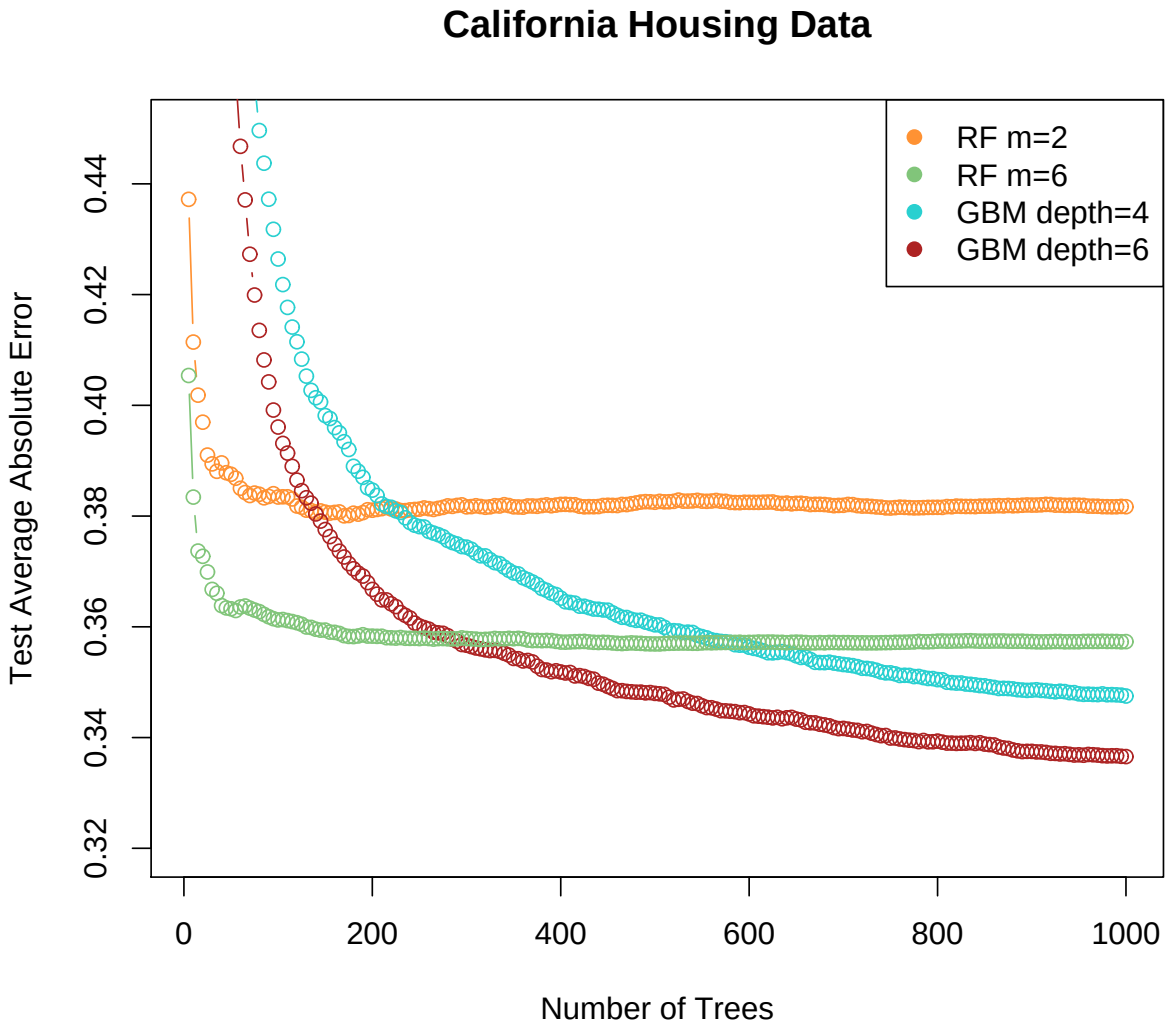
Details of previous figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict *cancer* versus *normal*.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Tuning parameters for boosting

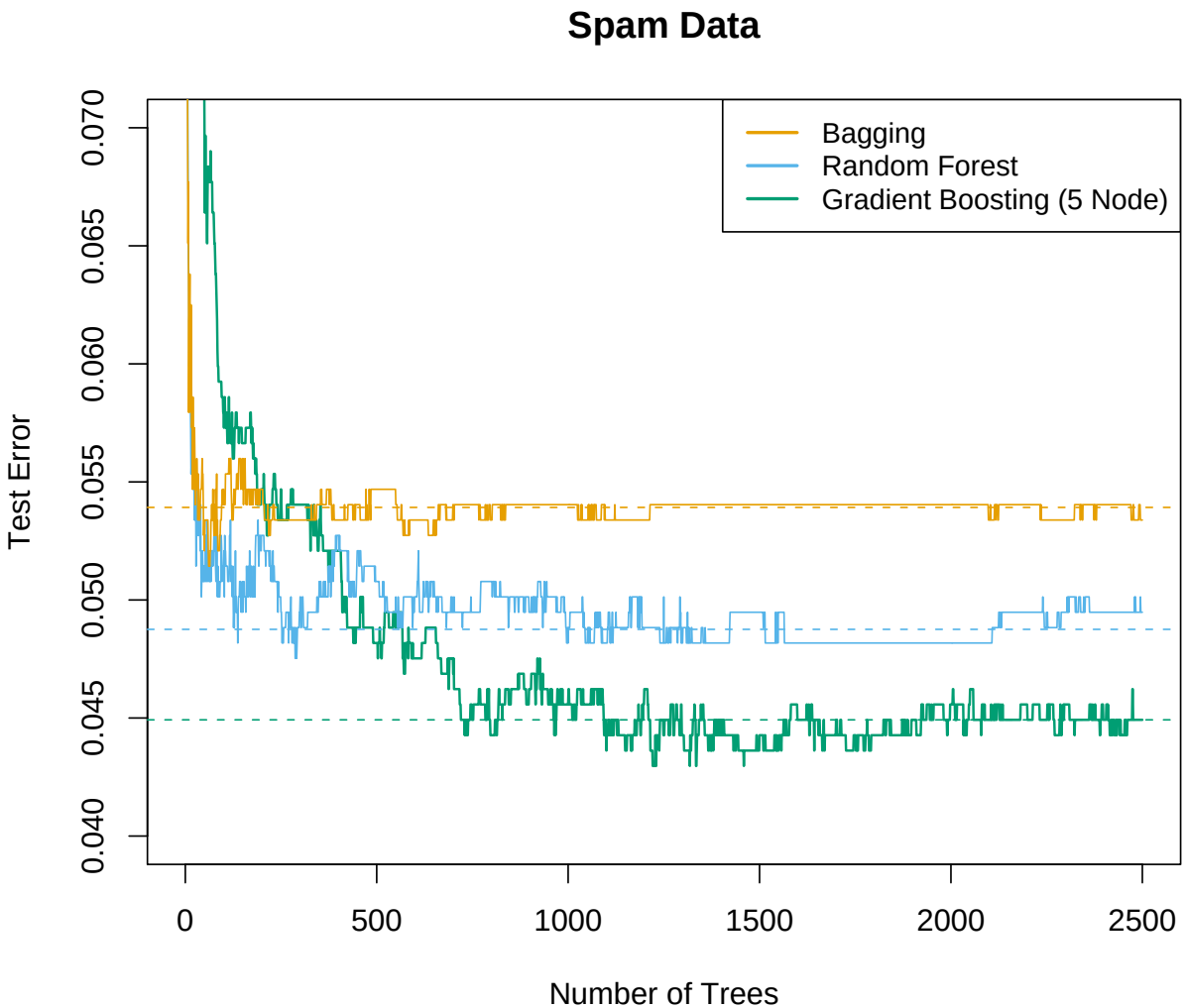
1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The *number of splits* d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally d is the *interaction depth*, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Another regression example



from *Elements of Statistical Learning, chapter 15.*

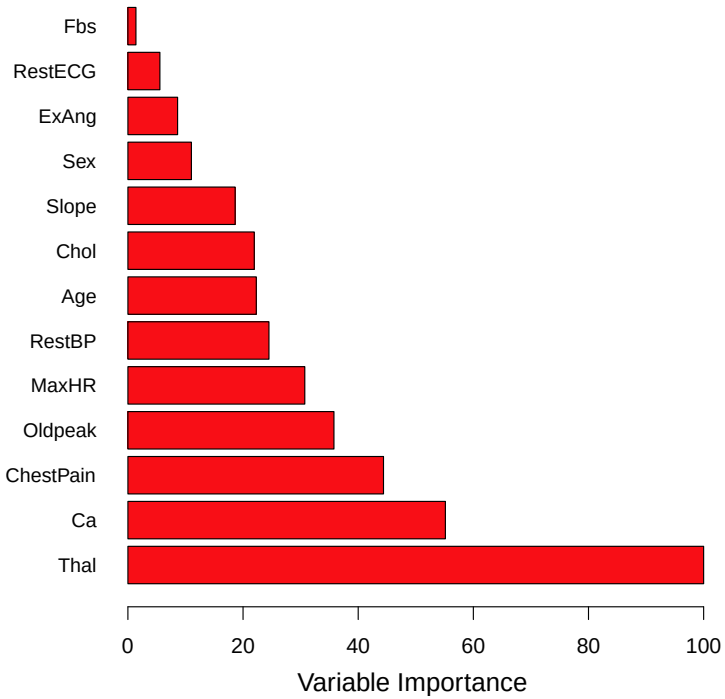
Another classification example



from *Elements of Statistical Learning, chapter 15.*

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.