Machine Learning I

Tree-based methods

Souhaib Ben Taieb

University of Mons

Table of contents

Outline

Tree-based methods

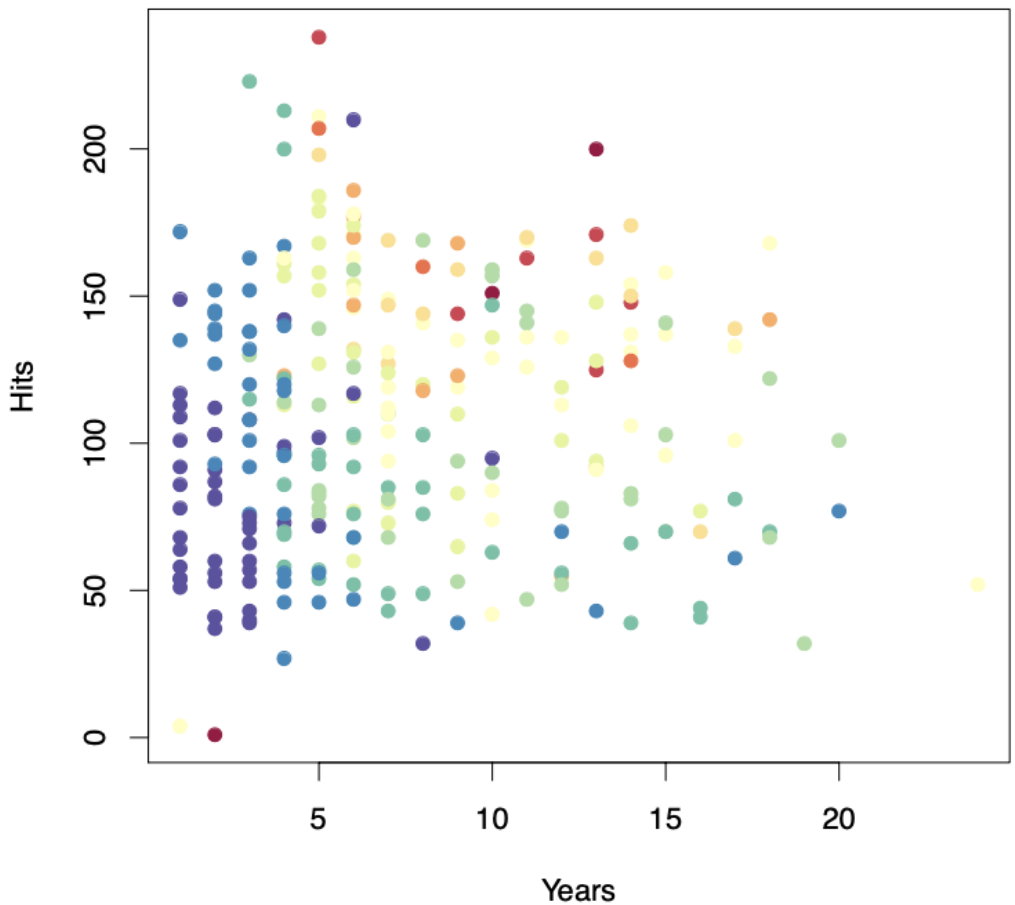
- Tree-based methods involve **stratifying** or **segmenting** the input space into a number of simple **regions**.
- Since we can represent the set of splitting rules to segment the input space in a **tree**, these types of methods are known as **decision-tree** methods.
- Tree-based methods can be 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).

Outline

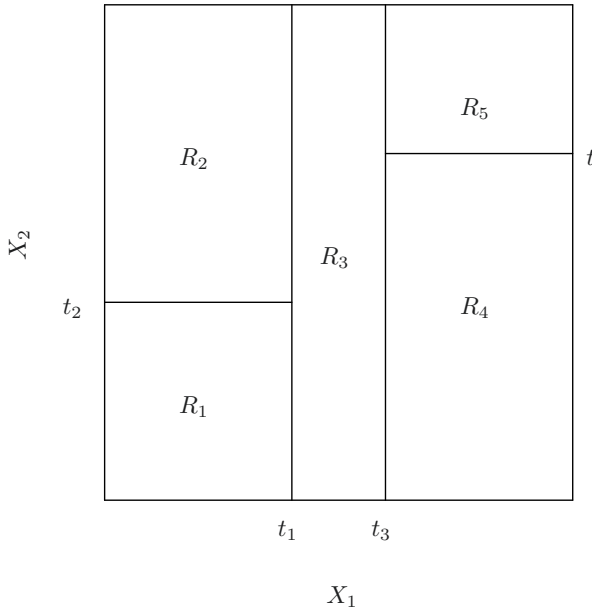
Outline

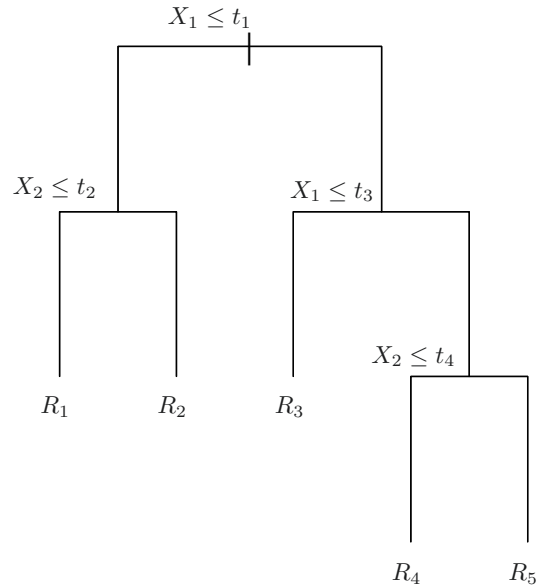
Example - Baseball salary data

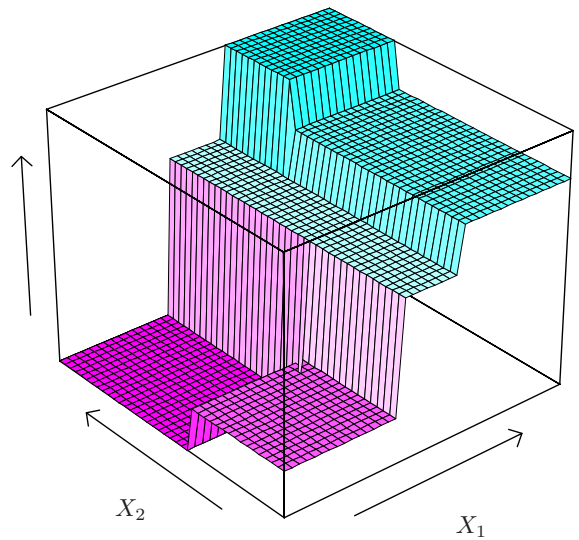
Predict log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits.



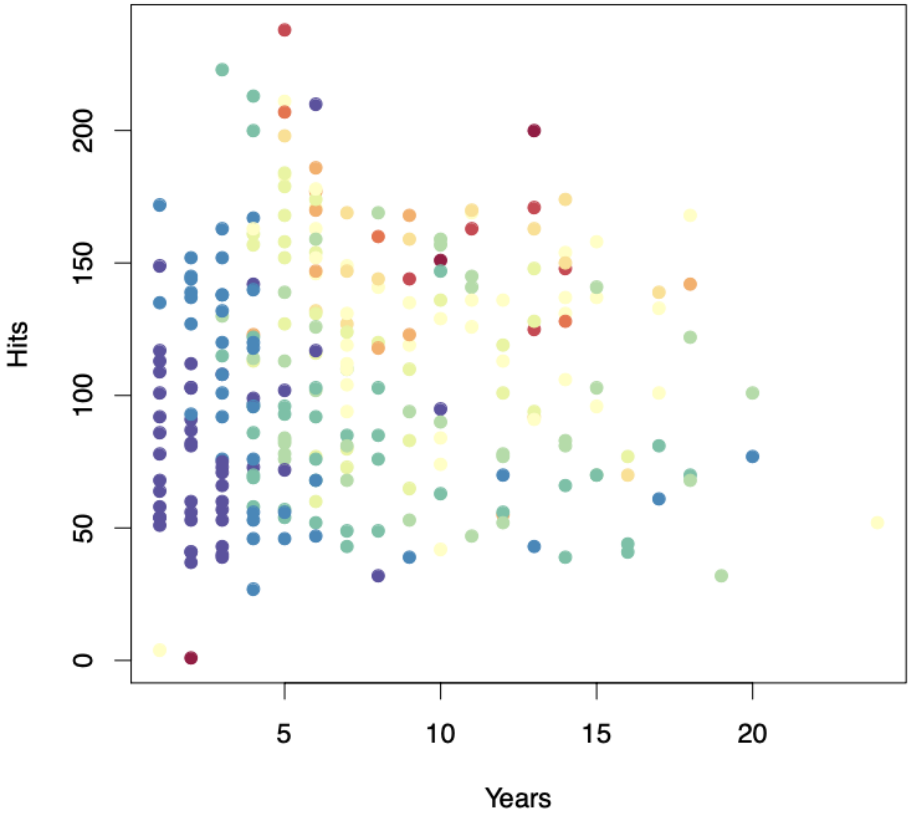
Regression trees

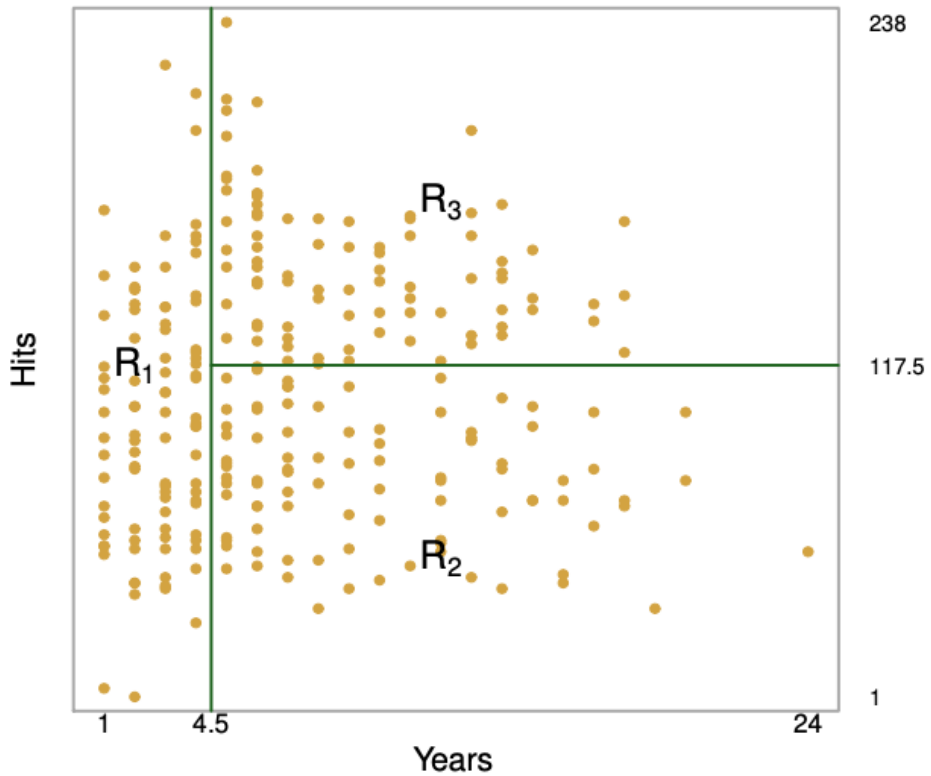






Example - Baseball salary data

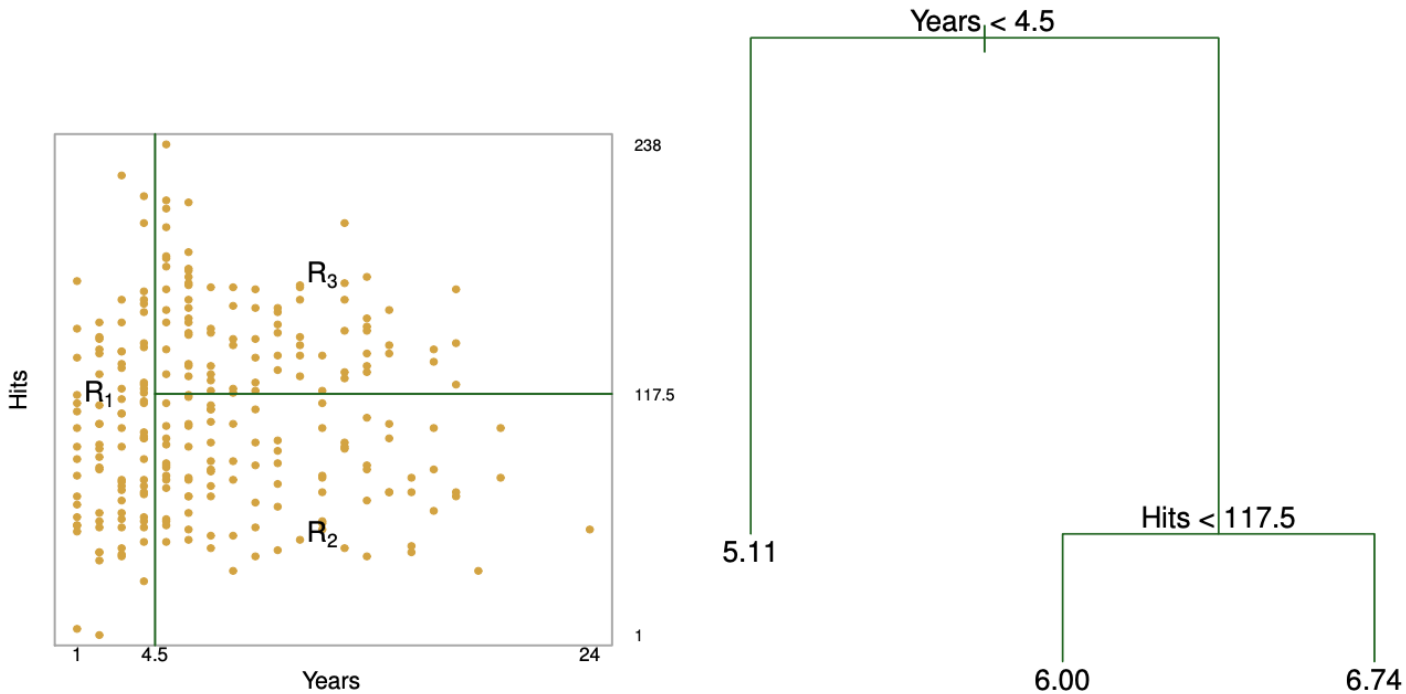




We split the input space into three regions:

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

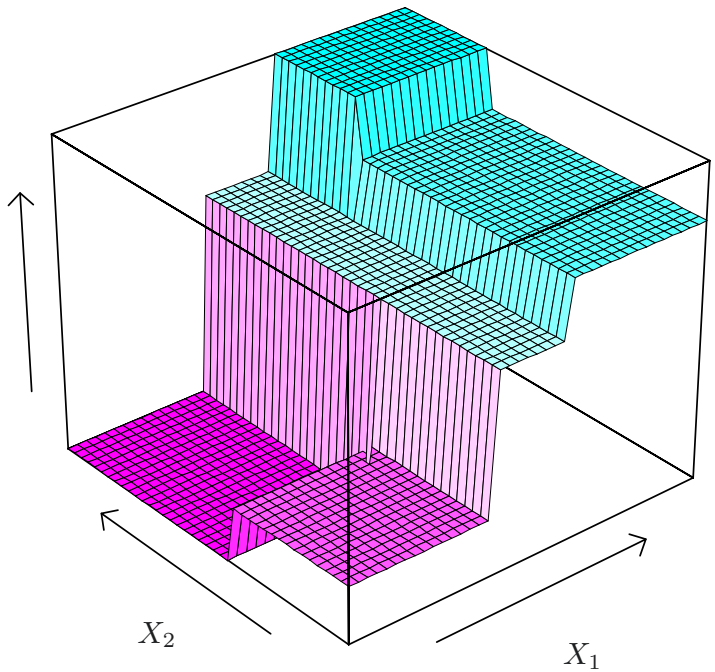
Example - Baseball salary data

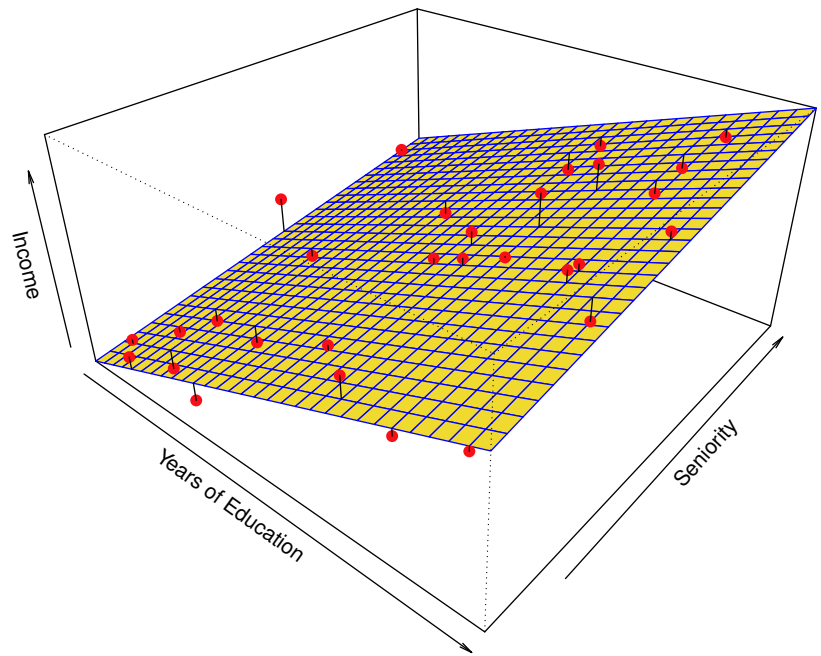


- R_1, R_2 and R_3 are known as **terminal nodes/leaves**.
- Years $<$ 4.5 and Hits $<$ 117.5 are **internal nodes**
- The number in each leaf is the **predicted salary** for all observations that fall there.

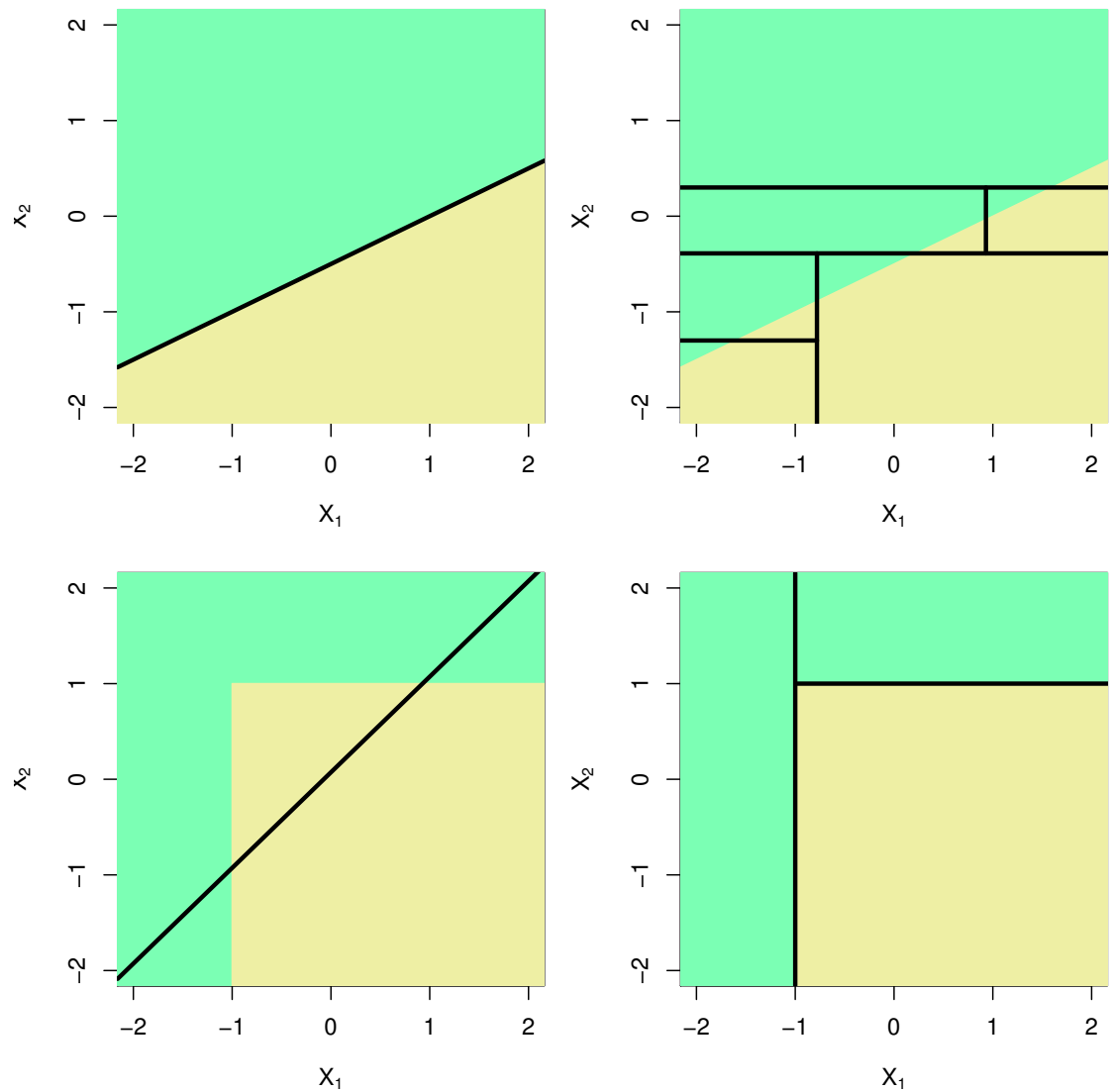
Regression trees vs linear regression

$$h(x) = \sum_{j=1}^{J} c_j \, \mathbb{I}(x \in R_j) \qquad h(x) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$





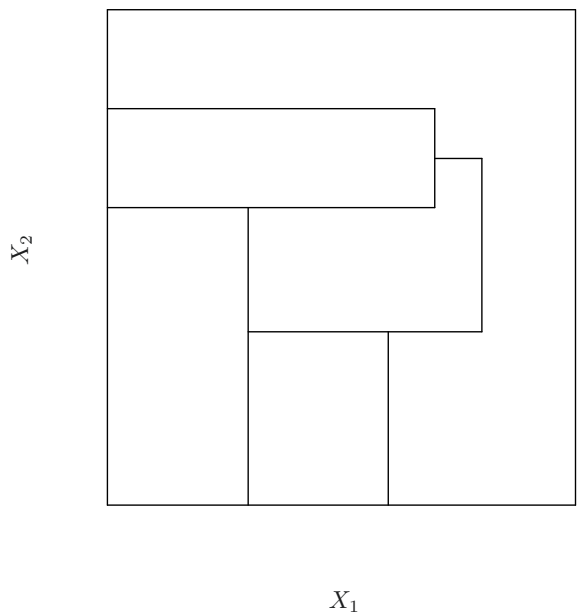
Regression trees vs linear regression

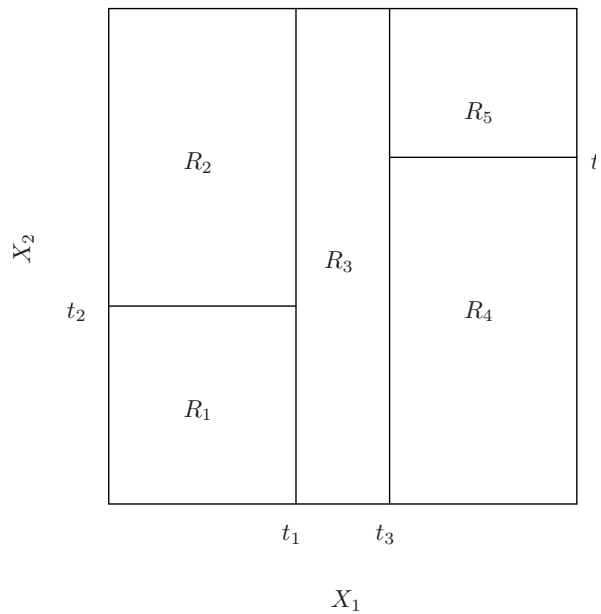


Outline

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.





Tree-building process

For every observation that falls into the region R_j, we make the **same prediction**. 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

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

Let T be the regression tree associated to the split into the regions $R_1, R_2, \ldots, R_J$. The constants $c_1, c_2, \ldots, c_J$, can be estimated by minimizing the RSS, i.e. $\sum_{i=1}^{n} (y_i - h(x_i))^2 = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - c_j)^2$.

It suffices to minimize it for each partition R_j independently:

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

The RSS of tree T is given by

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

Tree-building process

For every observation that falls into the region R_j, we make the **same prediction**. 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

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

Let T be the regression tree associated to the split into the regions $R_1, R_2, \ldots, R_J$. The constants $c_1, c_2, \ldots, c_J$, can be estimated by minimizing the RSS, i.e. $\sum_{i=1}^{n}(y_i - h(x_i))^2 = \sum_{j=1}^{J}\sum_{i \in R_j}(y_i - c_j)^2$.

It suffices to minimize it for each partition R_j independently:

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

The RSS of tree T is given by

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

Tree-building process

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

- 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**.

Recursive binary splitting

- 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.

Recursive binary splitting

- 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.

Recursive binary splitting

- 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.

Recursive binary splitting

- 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.

Recursive binary splitting

- 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.

Outline

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, denoted T_0, and then **prune** it back in order to obtain a **subtree**. We can do this with the **weakest link pruning** algorithm.

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, denoted T_0, and then **prune** it back in order to obtain a **subtree**. We can do this with the **weakest link pruning** algorithm.

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, denoted T_0, and then **prune** it back in order to obtain a **subtree**. We can do this with the **weakest link pruning** algorithm.

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, denoted T_0, and then **prune** it back in order to obtain a **subtree**. We can do this with the **weakest link pruning** algorithm.

Pruning a tree

1. Starting 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_r $(r = 0, 1, 2, \ldots, R)$ by cross validation

Estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees.

Pruning a tree

1. Starting 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_r $(r = 0, 1, 2, \ldots, R)$ by cross validation

Estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees.

Cost complexity pruning (weakest link pruning)

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

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

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

- 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 $\hat{\alpha}$

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

Building a regression tree - summary

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. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

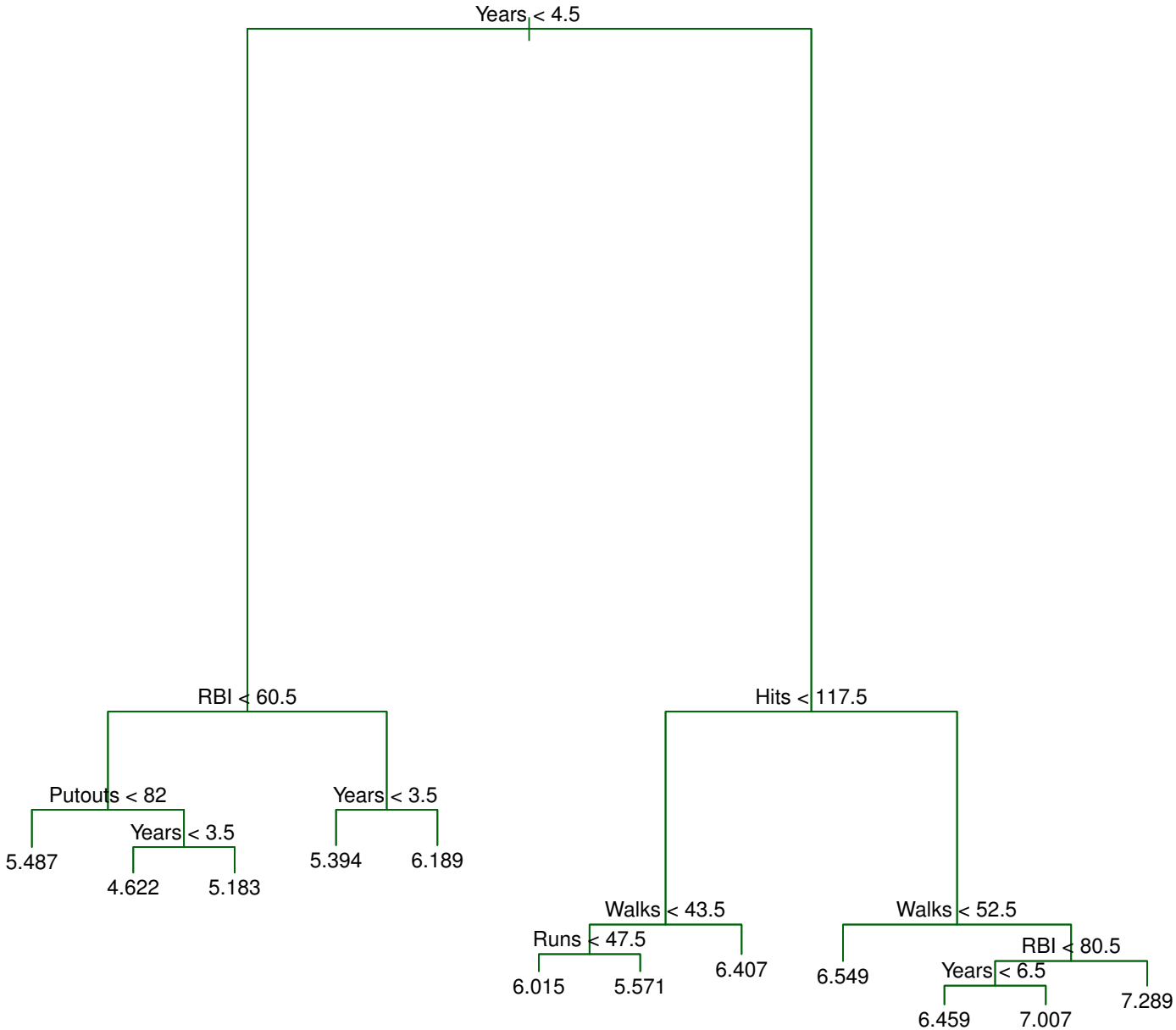
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

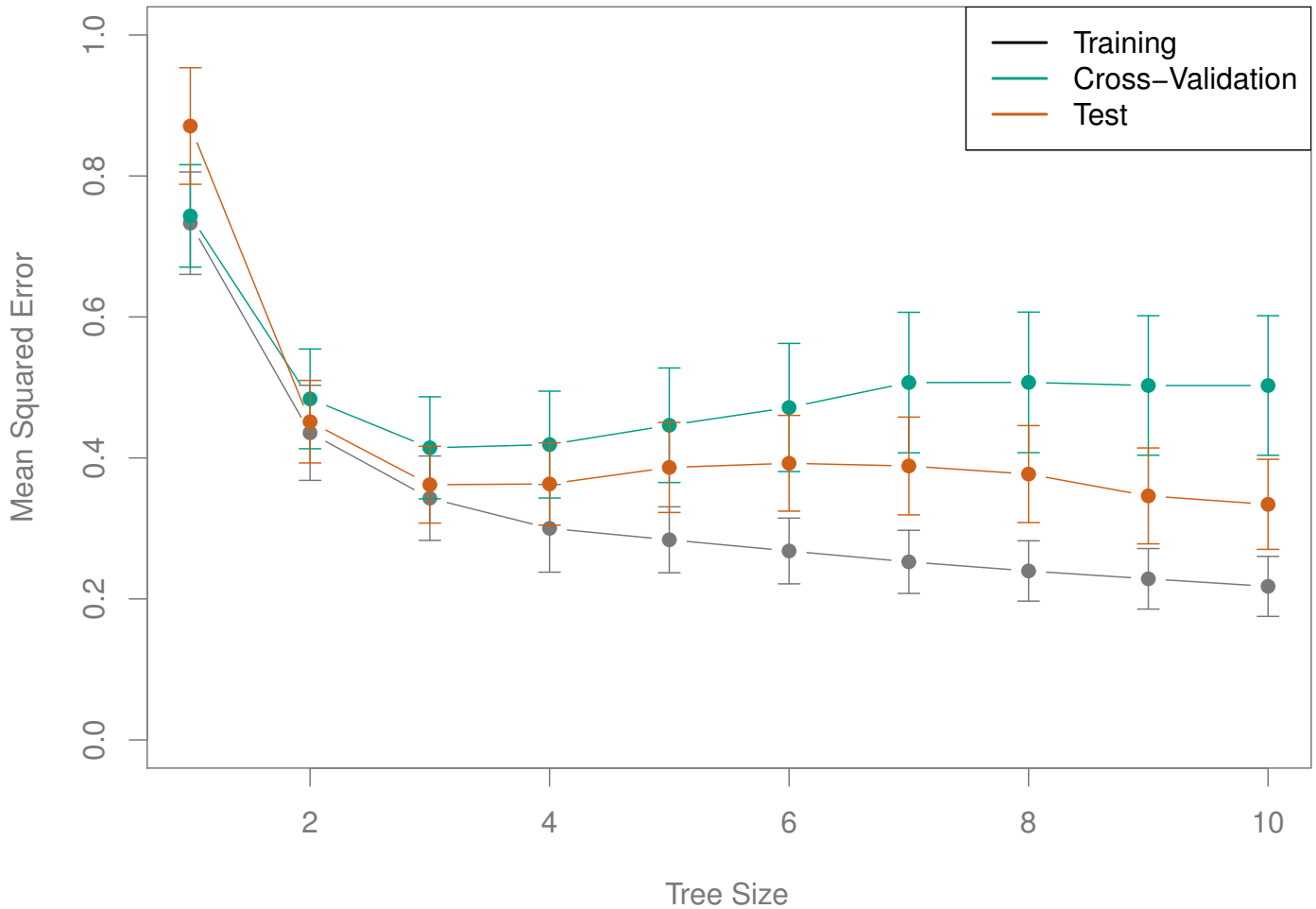
   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

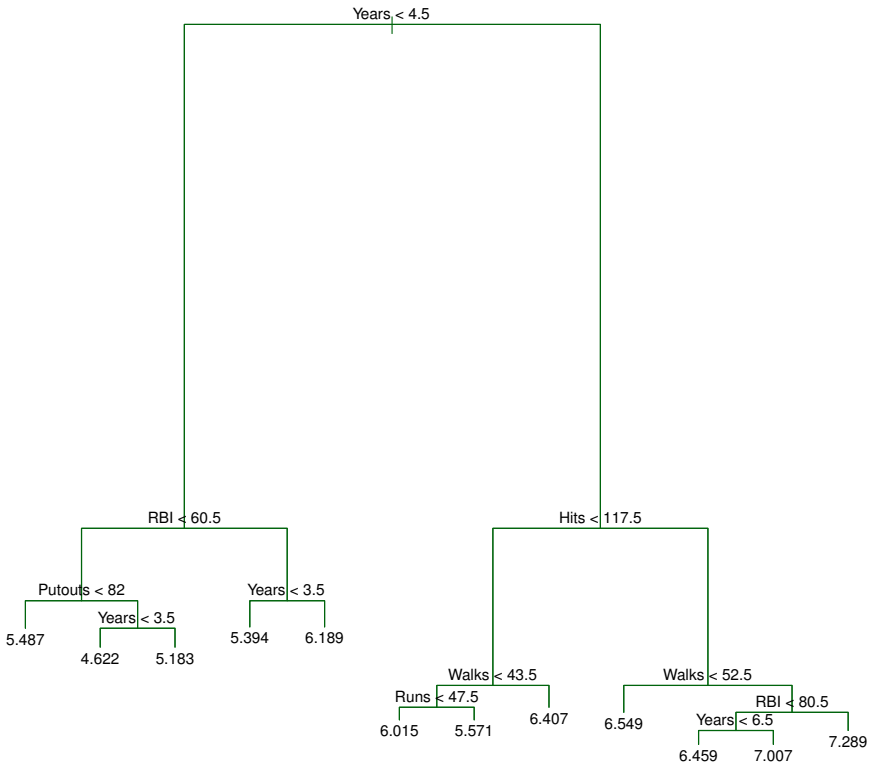
Example


Years < 4.5
RBI < 60.5
Hits < 117.5
Putouts < 82
Years < 3.5
Years < 3.5
Walks < 43.5
Walks < 52.5
Runs < 47.5
RBI < 80.5
Years < 6.5
5.487
4.622
5.183
5.394
6.189
6.015
5.571
6.407
6.549
6.459
7.007
7.289

Example


Training
Cross−Validation
Test
Mean Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Tree Size

Example


Years < 4.5
RBI < 60.5
Hits < 117.5
Putouts < 82
Years < 3.5
Years < 3.5
Walks < 43.5
Walks < 52.5
Runs < 47.5
RBI < 80.5
Years < 6.5
5.487
4.622
5.183
5.394
6.189
6.015
5.571
6.407
6.549
6.459
7.007
7.289


Years < 4.5
Hits < 117.5
5.11
6.00
6.74

Outline

Outline

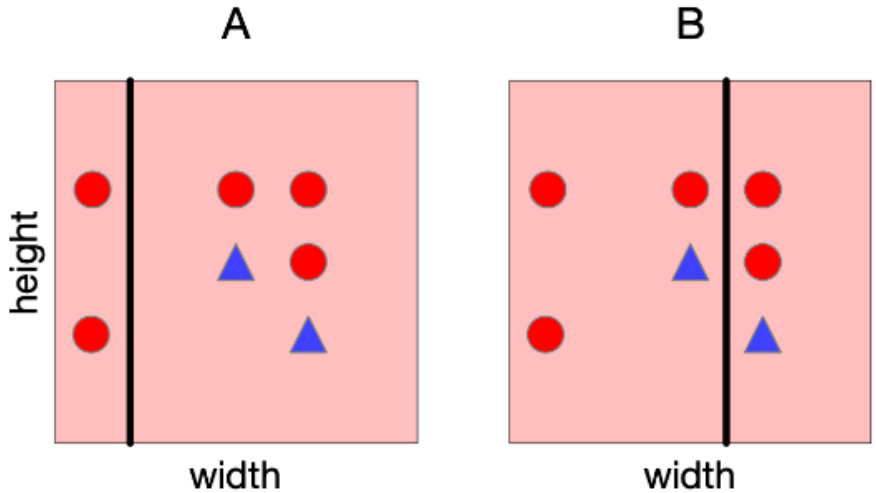
Classification trees

- As in the regression setting, we use **recursive binary splitting** to grow a classification tree.
- In each region, we predict the **most commonly occurring class**.
- 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

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

Let us examine using the misclassification rate for tree building.

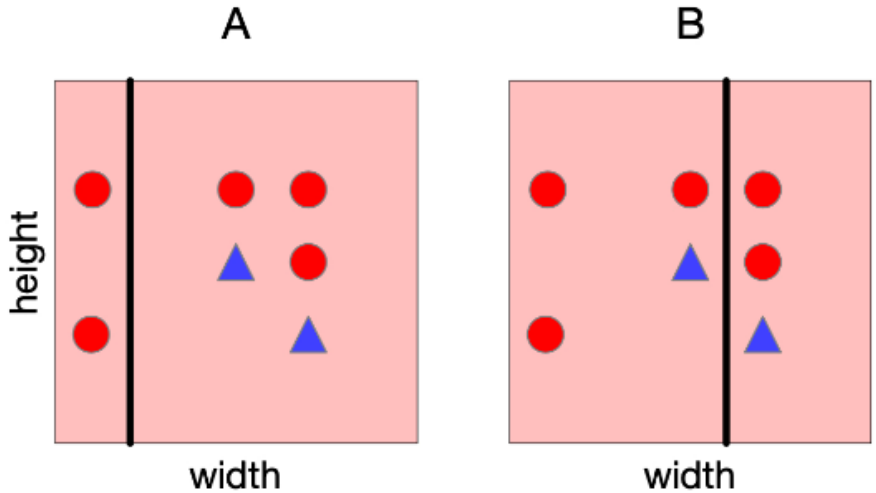
Classification error for tree-growing



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.

Classification error for tree-growing



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

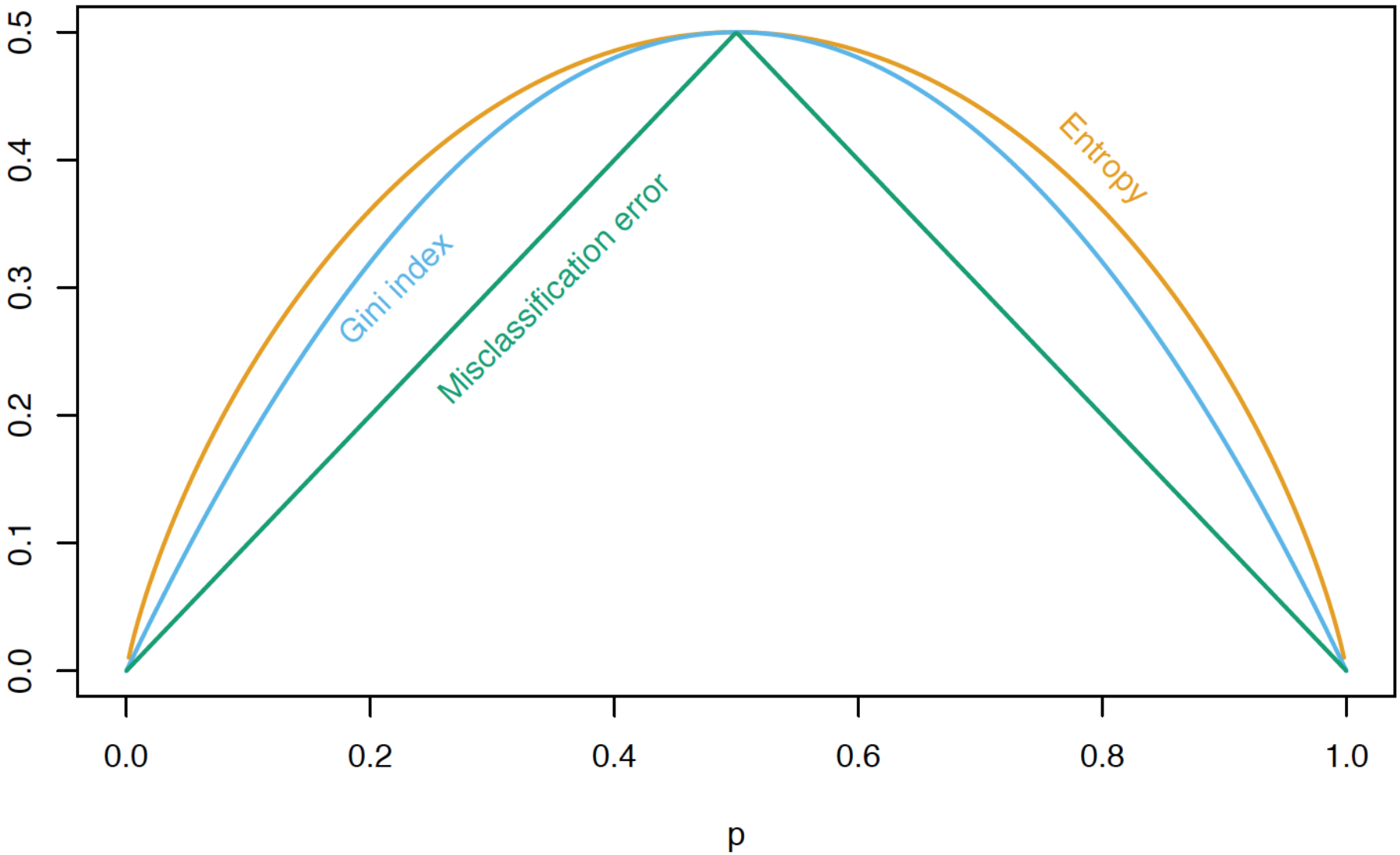
- A feels like a better split, because the left-hand region is **very certain** about whether the response is red.
- Classification error is not **sufficiently sensitive** for **tree-growing,** and in practice other measures are preferable.

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. $\hat{\boldsymbol{p}}_m = (\hat{p}_{m1}, \ldots, \hat{p}_{mK})^T$ and use a **probabilistic notion of uncertainty** to decide splits.

Gini index, entropy and misclassification error

$K = 2$ (Binary case)



Gini index and Entropy

The **Gini index** is defined as

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

where $\hat{\boldsymbol{p}}_m = (\hat{p}_{m1}, \ldots, \hat{p}_{mK})^T$. The Gini index is a measure of **total variance across the K classes.** A small value indicates that a node contains predominantly observations from a single class. It is referred to as a measure of **node purity**.

An alternative to the Gini index is **entropy**, given by

$$H(\hat{\boldsymbol{p}}_m) = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}),$$

which will take on a value near zero if the p_{mk}'s are all near zero or near one.

Outline

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**?
  - $H(Y|X) \approx 0.75$

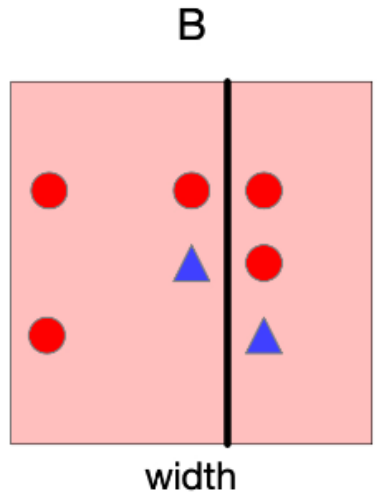
Information gain

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

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

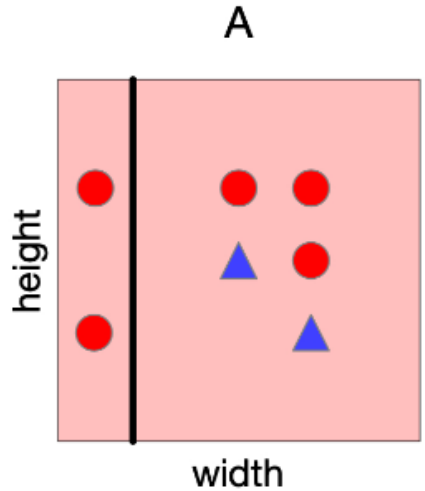
  i.e. my uncertainty in Y minus my expected uncertainty that would remain in Y given X.
- If X is completely **uninformative** about Y : $\mathrm{IG}(Y|X) = 0$.
- If X is completely **informative** about Y : $IG(Y|X) = H(Y)$.
- Information gain measures the informativeness of a variable, which is exactly 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) $\approx 0.86 - (\frac{4}{7} \times 0.81 + \frac{3}{7} \times 0.92) \approx \mathbf{0.006}$

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) $\approx 0.86 - (\frac{2}{7} \times 0 + \frac{5}{7} \times 0.97) \approx \mathbf{0.17}$.

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

Gini index, 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.

Advantages and disadvantages of trees

- Trees can be displayed **graphically** and easily **interpreted**. They are even easier to explain than linear regression.
- Some people believe that decision trees more closely mirror **human decision-making** than do the other regression and classification methods.
- Trees are very flexible (**small bias**) and can easily handle **qualitative predictors**.
- Trees can be very **non-robust**: a small change in the data can cause a large change in the final estimated tree (**high variance**).

However, by **aggregating** many decision trees, the predictive performance of trees can be substantially improved.

Outline

Bagging

- Bagging (Bootstrap aggregation) is a *general-purpose* procedure for **reducing the variance** of a learning method.
- It is particularly useful and frequently used in the context of decision trees.
- Bagging is based on the idea that the variance of the mean of B **independent** random variables $Z_1, Z_2, \ldots, Z_B$, each with variance σ^2 is given by σ^2/B. In fact, we have

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

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

Bagging

- However, we can **bootstrap**, by taking repeated samples from our training set
- Specifically, we generate B different bootstrapped training sets. We then train our model on the bth bootstrapped training set in order to get $h^{*b}(x)$, the prediction at a point x.
- With **regression** models, we then **average** all the predictions:

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

- With **classification** models, we can take a **majority vote**, i.e. we predict the most commonly occurring class among the B predictions.

Bagging with trees

- We train B **deep** trees (**not pruned**) on B different bootstrapped training sets. Hence, each individual tree has **low bias** but **high variance**.
- Averaging these B low-bias/high-variance trees will lead to a **low-bias** model with **reduced variance**.
- The number of trees B is **not a critical hyperparameter** with bagging; using a very large value of B will not lead to overfitting. In practice we use a value of B sufficiently large that the error has settled down.

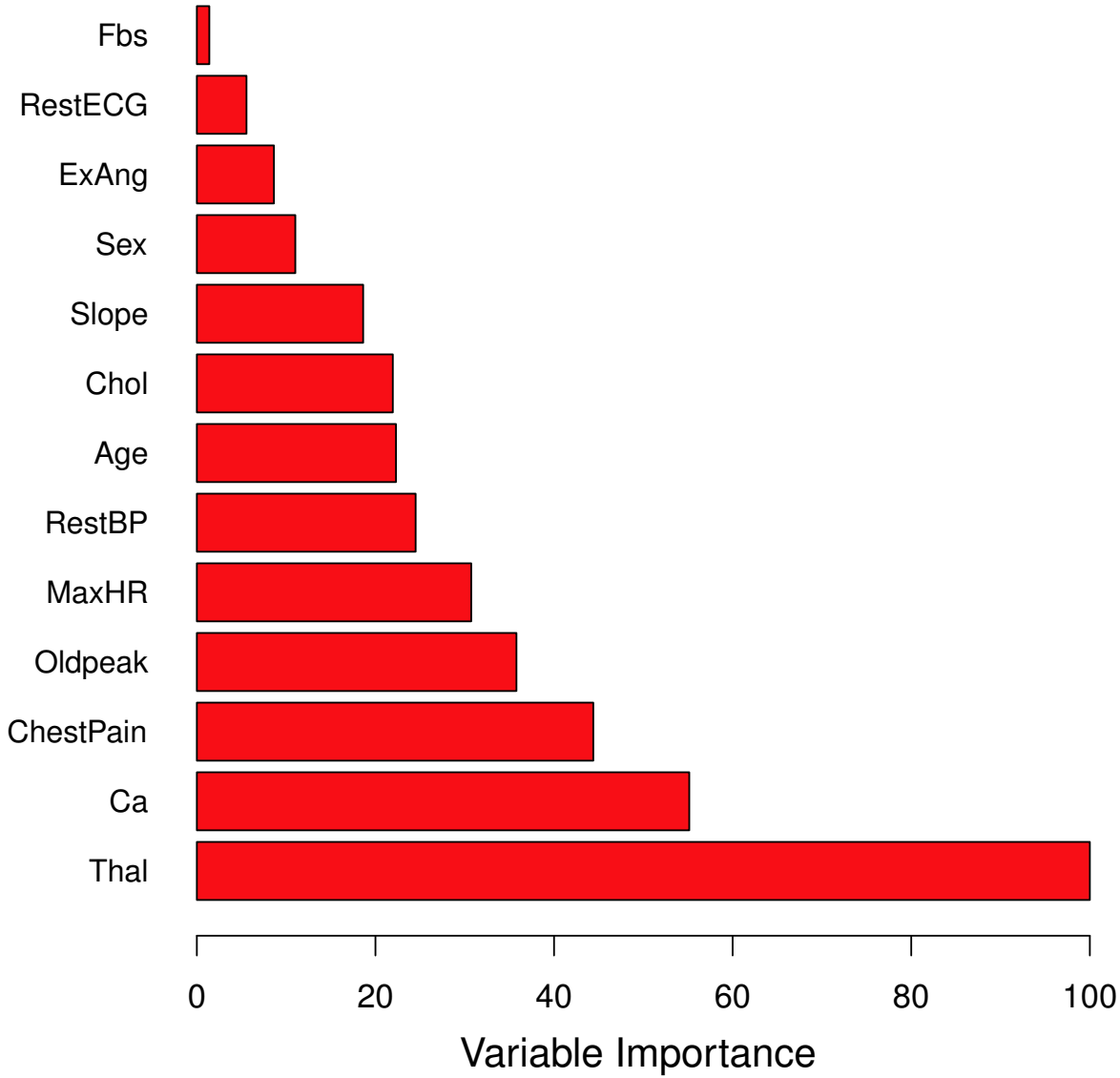
Out-of-bag Error Estimation

- There is a very straightforward way to **estimate the test error** of a bagged model, without the need to perform *(cross-)validation*
- Since we are using the Bootstrap, on average, each bagged tree makes use of around **two-thirds** of the training data.
- The remaining **one-third** of the training data not used to fit a given bagged tree are referred to as the **out-of-bag (OOB)** data points.
- 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**.

Variable importance measures

- Recall that one of the advantages of decision trees is that they can be easily **interpreted**.
- However, it is more difficult to interpret a bagged model with a large number of trees, and it is no longer clear **which variables are most important**. Thus, bagging improves **prediction accuracy** at the expense of **interpretability**.
- We can obtain an overall summary of the importance of each predictor using the **RSS** (for bagging regression trees) or the **Gini index** (for bagging classification trees).
- We can record the **total amount that the RSS/Gini is decreased** due to splits over a given predictor, averaged over all B trees

Variable importance measures



Outline

Variance of sum of i.d. and correlated variables

- Given a set of B **i.i.d.** random variables $Z_1, \ldots, Z_B$, each with variance σ^2, we have

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

- If the variables are simply **i.d.** (identically distributed, but not necessarily independent) with **positive pairwise correlation** ρ ($\rho > 0$), we have

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \rho\sigma^2 + \sigma^2\frac{(1-\rho)}{B}$$

  - $\rho\sigma^2$: decreases if ρ decreases
  - $\sigma^2\frac{(1-\rho)}{B}$: decreases if B increases (irrespective of ρ)
    $\rightarrow$ The Random Forests model will exploit this observation.

Variance of sum of i.d. and correlated variables

- Given a set of B **i.i.d.** random variables $Z_1, \ldots, Z_B$, each with variance σ^2, we have

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \frac{B\sigma^2}{B^2} = \frac{\sigma^2}{B}$$

- If the variables are simply **i.d.** (identically distributed, but not necessarily independent) with **positive pairwise correlation** ρ ($\rho > 0$), we have

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \rho\sigma^2 + \sigma^2\frac{(1-\rho)}{B}$$

  - $\rho\sigma^2$: decreases if ρ decreases
  - $\sigma^2\frac{(1-\rho)}{B}$: decreases if B increases (irrespective of ρ)
    $\rightarrow$ The Random Forests model will exploit this observation.

Variance of sum of i.d. and correlated variables

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

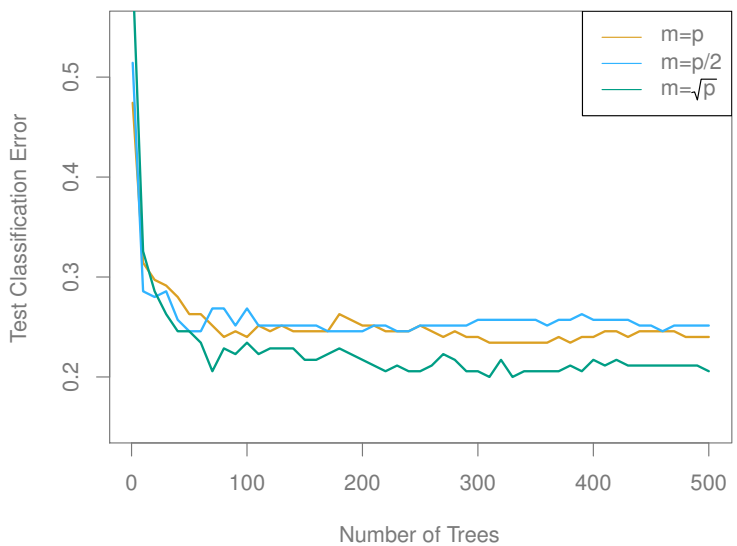
since $\operatorname{Cov}(Z_i, Z_j) = \rho\sigma^2$.

Random Forests

- As in bagging, we build a number of decision trees on **bootstrapped training sets**
- Random Forests add a small tweak that **decorrelates the trees**, which **reduces the variance** when we average the trees.
- When building these decision trees, each time a split in a tree is considered, a **random selectio**n of m (often $m = \sqrt{p}$) 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.
- Of course, if a random forest is built using $m = p$, then this amounts simply to **bagging**

Example

- High-dimensional biological data set consisting of expression measurements of $p = 4,718$ genes measured on tissue samples from $n = 349$ patients
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer
- The goal is to use random forests to predict cancer type based on the $p = 500$ genes that have the largest variance in the training set
- The error rate of a single tree is 45.7%, and the null rate is 75.4%.



Outline

Boosting

- Like bagging, boosting is a *general approach* that can be applied to many learning methods for regression or classification.
- In bagging, each tree is built on a bootstrap data set, **independent** of the other trees. Then, these trees are **combined** in order to create a single predictive model.
- In boosting the trees are grown **sequentially**: each tree is grown using information from **previously grown trees**. Boosting does not involve bootstrap sampling; instead each tree is fit on a **modified version** of the original data set.
- In the following, we will focus on boosting with regression trees, also called **gradient boosting models** (GBM) for regression.

What is the idea behind Boosting?

- 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 current 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** (**high bias**), with just a few terminal nodes, determined by a hyperparameter. These are called **weak learners**.
- By fitting small trees to the residuals, we **slowly** improve the model fit in areas where it does not perform well. A shrinkage hyperparameter can slow the process down even further, allowing more and different shaped trees to fit the residuals.
- If bagging can be interpreted as a **variance reduction** procedure, boosting is a **bias reduction** procedure.

Boosting for Regression Trees

Boosting for Regression Trees

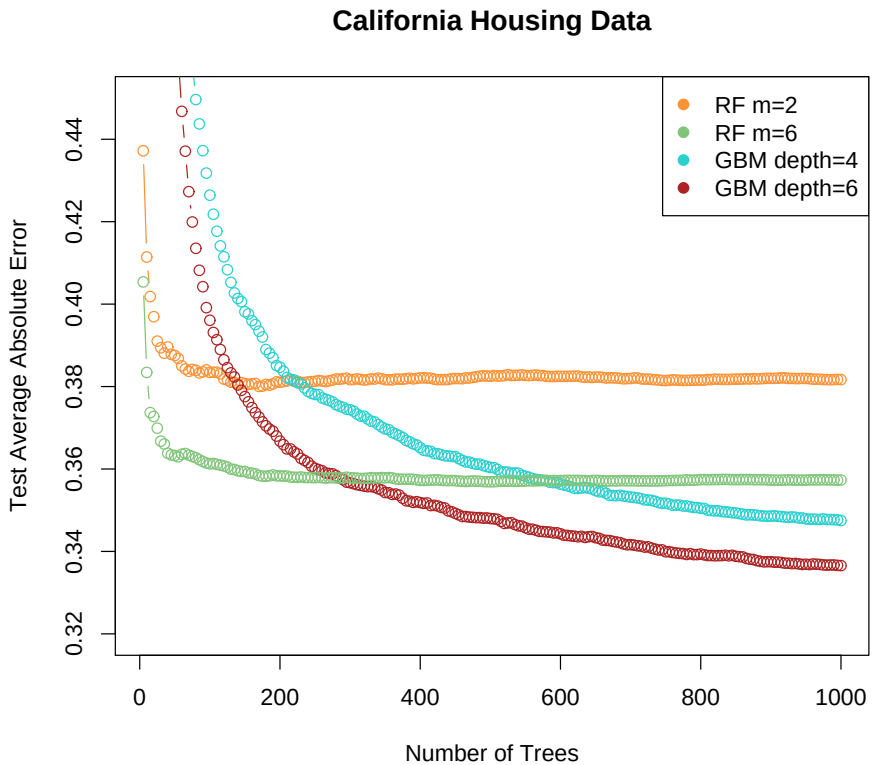
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

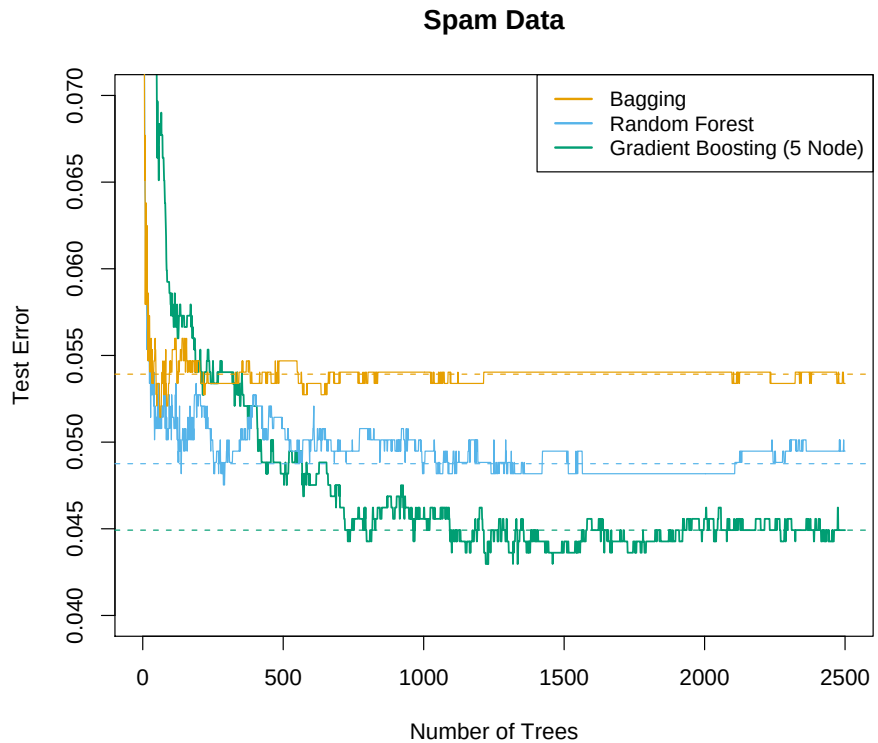
Three tuning hyperparameters: **the number of trees** B, **the shrinkage parameter** λ, and **the number of splits** d.

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 controls the complexity of the boosted ensemble. If $d = 1$, 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

Example


California Housing Data
RF m=2
RF m=6
GBM depth=4
GBM depth=6
Test Average Absolute Error
0.32
0.34
0.36
0.38
0.40
0.42
0.44
0
200
400
600
800
1000
Number of Trees


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees

Summary

- Decision trees are **flexible** and **interpretable** models for regression and classification.
- However due to their **high variance** they are often not competitive with other methods in terms of predictive 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.
- Random forests and boosting are among the **state-of-the-art methods** for supervised learning (with tabular data). However, they are difficult to **interpret**.