# Decision Trees

Tal Linzen
Slides based on Lecture 10 from David Rosenberg's course materials
(https://github.com/davidrosenberg/mlcourse)

CDS, NYU

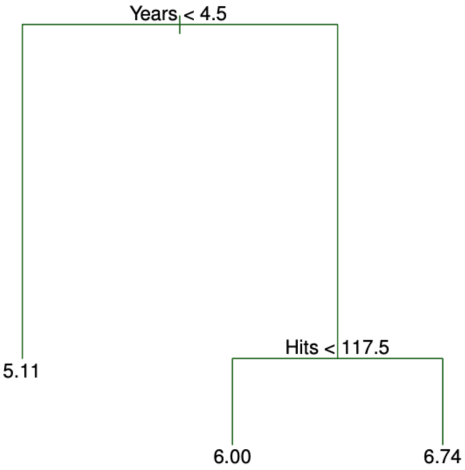
April 5, 2022

# Today's lecture

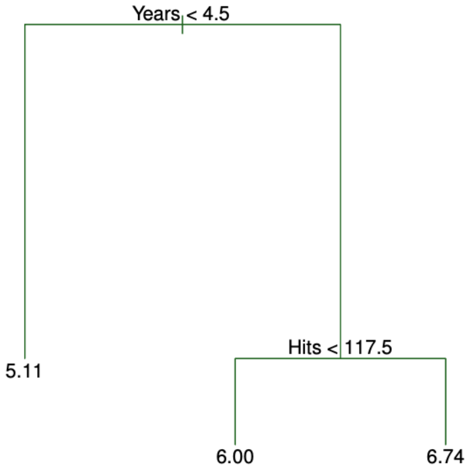
- Our first inherently non-linear classifier: decision trees.
- Ensemble methods: bagging and boosting.

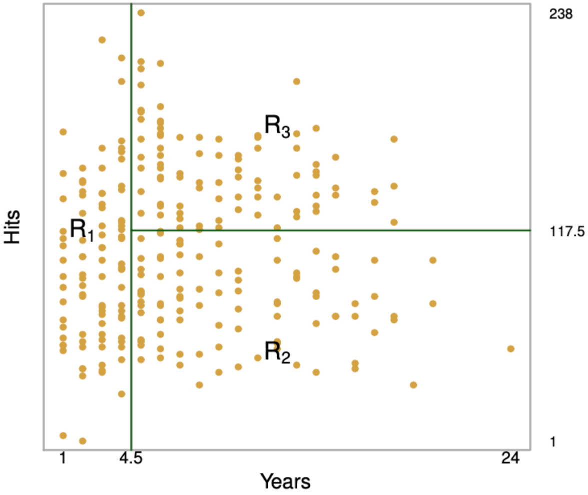
# Decision Trees

# Regression trees: Predicting basketball players' salaries

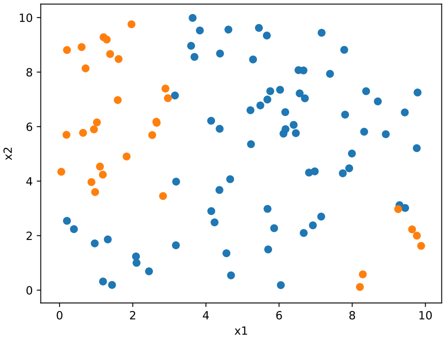


# Regression trees: Predicting basketball players' salaries



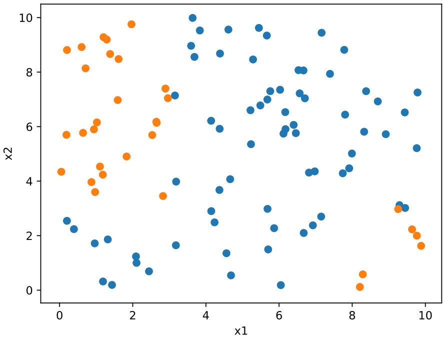



# Classification trees



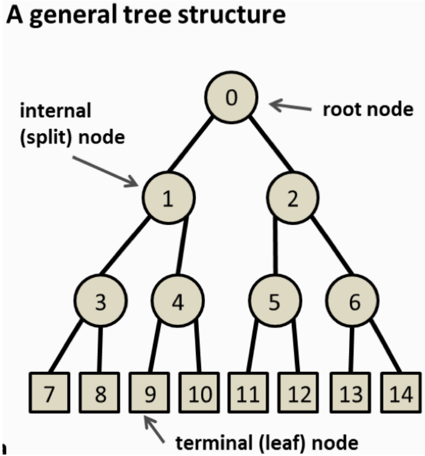

- Can we classify these points using a linear classifier?

# Classification trees




- Can we classify these points using a linear classifier?
- Partition the data into axis-aligned regions recursively (on the board)

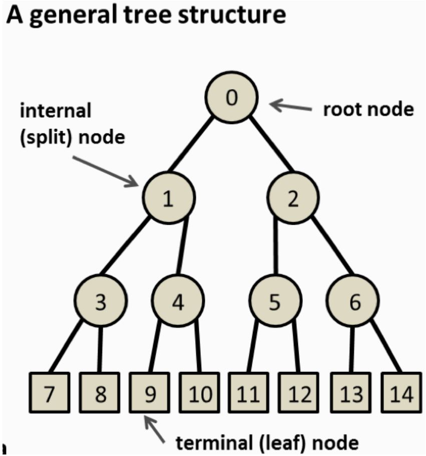
# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

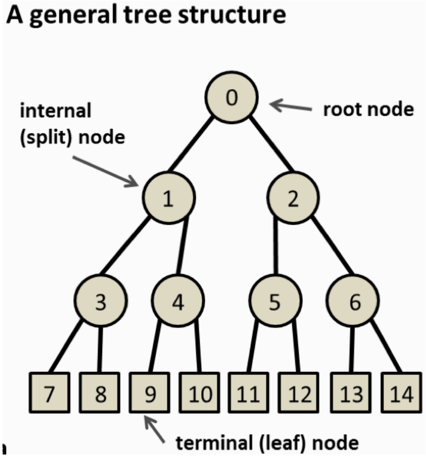
# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

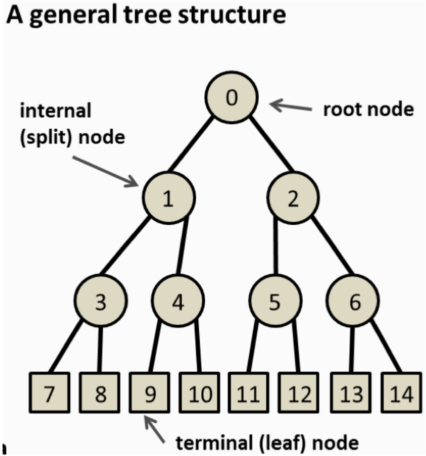
# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points
- The data splits created by each node involve only a *single* feature

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

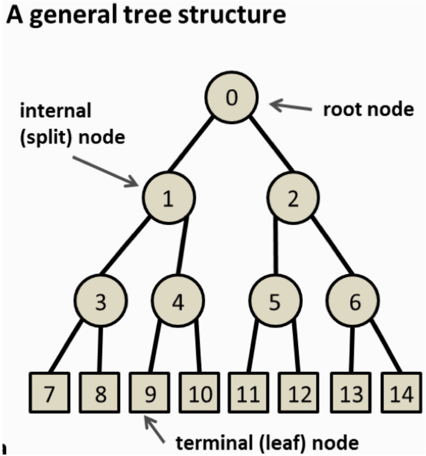
# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points
- The data splits created by each node involve only a *single* feature
- For continuous variables, the splits are always of the form $x_i \leqslant t$

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

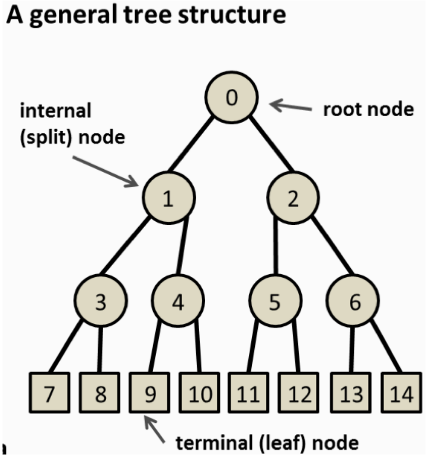
# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points
- The data splits created by each node involve only a *single* feature
- For continuous variables, the splits are always of the form $x_i \leqslant t$
- For discrete variables, we partition values into two sets (not covered today)

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# Decision trees setup




- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points
- The data splits created by each node involve only a *single* feature
- For continuous variables, the splits are always of the form $x_i \leqslant t$
- For discrete variables, we partition values into two sets (not covered today)
- Predictions are made in terminal nodes

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

**Solution** Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

**Solution** Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

- We only split regions defined by previous non-terminal nodes

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

**Solution** Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

- We only split regions defined by previous non-terminal nodes

**Prediction** Our prediction is the mean value of a terminal node: $\hat{y}_{R_m} = \text{mean}(y_i \mid x_i \in R_m)$

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

**Solution** Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

- We only split regions defined by previous non-terminal nodes

**Prediction** Our prediction is the mean value of a terminal node: $\hat{y}_{R_m} = \text{mean}(y_i \mid x_i \in R_m)$

- A greedy algorithm is the one that make the best **local** decisions, without lookahead to evaluate their downstream consequences

# Constructing the tree

**Goal** Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

**Problem** Finding the optimal binary tree is computationally intractable.

**Solution** Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

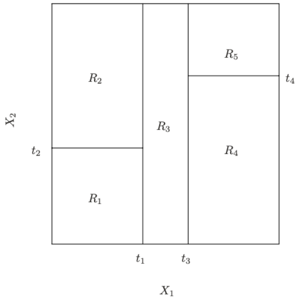
- We only split regions defined by previous non-terminal nodes

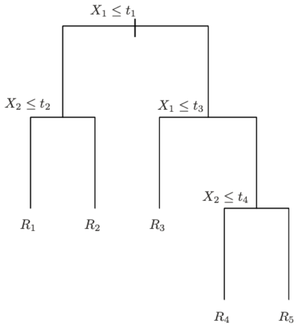
**Prediction** Our prediction is the mean value of a terminal node: $\hat{y}_{R_m} = \text{mean}(y_i \mid x_i \in R_m)$

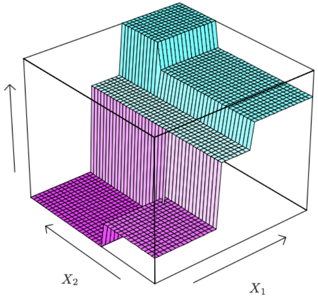
- A greedy algorithm is the one that make the best **local** decisions, without lookahead to evaluate their downstream consequences

- This procedure is not very likely to result in the globally optimal tree

# Prediction in a Regression Tree



# Finding the Best Split Point

- We enumerate all features and all possible split points for each feature. There are infinitely many split points, but...

# Finding the Best Split Point

- We enumerate all features and all possible split points for each feature. There are infinitely many split points, but...
- Suppose we are now considering splitting on the $j$-th feature $x_j$, and let $x_{j(1)}, \ldots, x_{j(n)}$ be the sorted values of the $j$-th feature.

# Finding the Best Split Point

- We enumerate all features and all possible split points for each feature. There are infinitely many split points, but...
- Suppose we are now considering splitting on the $j$-th feature $x_j$, and let $x_{j(1)}, \ldots, x_{j(n)}$ be the sorted values of the $j$-th feature.
- We only need to consider split points between two adjacent values, and any split point in the interval $(x_{j(r)}, x_{(j(r+1))})$ will result in the same loss

# Finding the Best Split Point

- We enumerate all features and all possible split points for each feature. There are infinitely many split points, but...
- Suppose we are now considering splitting on the $j$-th feature $x_j$, and let $x_{j(1)}, \ldots, x_{j(n)}$ be the sorted values of the $j$-th feature.
- We only need to consider split points between two adjacent values, and any split point in the interval $(x_{j(r)}, x_{(j(r+1)})$ will result in the same loss
- It is common to split half way between two adjacent values:

$$s_j \in \left\{ \frac{1}{2} \left( x_{j(r)} + x_{j(r+1)} \right) \mid r = 1, \ldots, n-1 \right\}. \qquad n-1 \text{ splits} \tag{1}$$

# Decision Trees and Overfitting

- What will happen if we keep splitting the data into more and more regions?

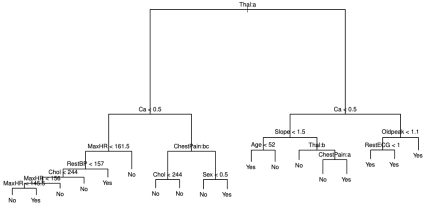
# Decision Trees and Overfitting

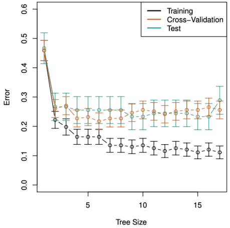
- What will happen if we keep splitting the data into more and more regions?
  - Every data point will be in its own region—overfitting.

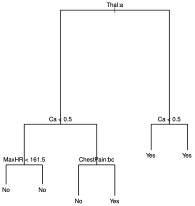
# Decision Trees and Overfitting

- What will happen if we keep splitting the data into more and more regions?
  - Every data point will be in its own region—overfitting.
- When should we stop splitting? (Controlling the complexity of the hypothesis space)
  - Limit total number of nodes.
  - Limit number of terminal nodes.
  - Limit tree depth.
  - Require minimum number of data points in a terminal node.
  - **Backward pruning** (the approach used in **CART**; Breiman et al 1984):
    1. Build a really big tree (e.g. until all regions have $\leqslant 5$ points).
    2. *Prune* the tree back greedily, potentially all the way to the root, until validation performance starts decreasing.

# Pruning: Example



# What Makes a Good Split for Classification?

Our plan is to predict the **majority label** in each region.

Which of the following splits is better?

Split 1 $R_1 : 8+/2-$ $\qquad$ $R_2 : 2+/8-$

Split 2 $R_1 : 6+/4-$ $\qquad$ $R_2 : 4+/6-$

# What Makes a Good Split for Classification?

Our plan is to predict the **majority label** in each region.

Which of the following splits is better?

Split 1 $R_1 : 8+/2-$ $\quad$ $R_2 : 2+/8-$

Split 2 $R_1 : 6+/4-$ $\quad$ $R_2 : 4+/6-$

How about here?

Split 1 $R_1 : 8+/2-$ $\quad$ $R_2 : 2+/8-$

Split 2 $R_1 : 6+/4-$ $\quad$ $R_2 : 0+/10-$

# What Makes a Good Split for Classification?

Our plan is to predict the **majority label** in each region.

Which of the following splits is better?

Split 1 $R_1 : 8+/2-$ $R_2 : 2+/8-$

Split 2 $R_1 : 6+/4-$ $R_2 : 4+/6-$

How about here?

Split 1 $R_1 : 8+/2-$ $R_2 : 2+/8-$

Split 2 $R_1 : 6+/4-$ $R_2 : 0+/10-$

Intuition: we want to produce *pure* nodes, i.e. nodes where most instances have the same class.

# Misclassification error in a node

- Let's consider the multiclass classification case: $\mathcal{Y} = \{1, 2, \ldots, K\}$.
- Let node $m$ represent region $R_m$, with $N_m$ observations
- We denote the proportion of observations in $R_m$ with class $k$ by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\{i : x_i \in R_m\}} 1(y_i = k).$$

- We predict the majority class in node $m$:

$$k(m) = \arg\max_k \hat{p}_{mk}$$

# Node Impurity Measures

- Three measures of **node impurity** for leaf node *m*:

# Node Impurity Measures

- Three measures of **node impurity** for leaf node $m$:
  Misclassification error
  $$1 - \hat{p}_{mk(m)}.$$

# Node Impurity Measures

- Three measures of **node impurity** for leaf node $m$:
  Misclassification error
  $$1-\hat{p}_{mk(m)}.$$
  The Gini index encourages $\hat{p}_{mk}$ to be close to 0 or 1
  $$\sum_{k=1}^{K}\hat{p}_{mk}(1-\hat{p}_{mk})$$

# Node Impurity Measures

- Three measures of **node impurity** for leaf node $m$:

  Misclassification error

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

The Gini index encourages $\hat{p}_{mk}$ to be close to 0 or 1

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

Entropy / Information gain

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

# Node Impurity Measures

- Three measures of **node impurity** for leaf node $m$:

  Misclassification error

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

  The Gini index encourages $\hat{p}_{mk}$ to be close to 0 or 1

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

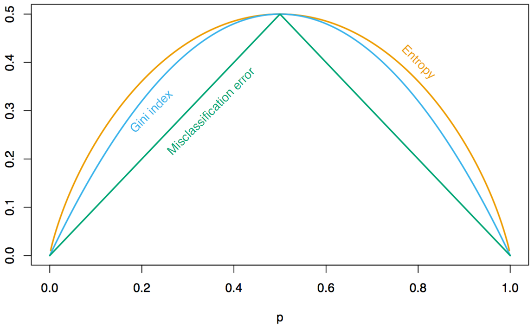
  Entropy / Information gain

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

- The Gini index and entropy are numerically similar to each other, and both work better in practice than the misclassification error.

# Impurity Measures for Binary Classification

($p$ is the relative frequency of class 1)




HTF Figure 9.3

# Quantifying the Impurity of a Split

Scoring a potential split that produces the nodes $R_L$ and $R_R$:

- Suppose we have $N_L$ points in $R_L$ and $N_R$ points in $R_R$.
- Let $Q(R_L)$ and $Q(R_R)$ be the node impurity measures for each node.
- We aim to find a split that minimizes the *weighted average of node impurities*:

$$\frac{N_L Q(R_L) + N_R Q(R_R)}{N_L + N_R}$$

# Discussion: Interpretability of Decision Trees

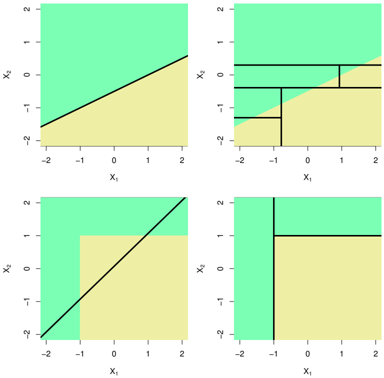

- Trees are easier to visualize and explain than other classifiers (even linear regression)

# Discussion: Interpretability of Decision Trees


- Trees are easier to visualize and explain than other classifiers (even linear regression)
- Small trees are interpretable – large trees, maybe not so much

# Discussion: Trees vs. Linear Models

Trees may have to work hard to capture linear decision boundaries, but can easily capture certain nonlinear ones:



# Discussion: Review

Decision trees are:

- Non-linear: the decision boundary that results from splitting may end up being quite complicated
- Non-metric: they do not rely on the geometry of the space (inner products or distances)
- Non-parametric: they make no assumptions about the distribution of the data

# Discussion: Review

Decision trees are:

- Non-linear: the decision boundary that results from splitting may end up being quite complicated
- Non-metric: they do not rely on the geometry of the space (inner products or distances)
- Non-parametric: they make no assumptions about the distribution of the data

Additional pros:

- Interpretable and simple to understand

# Discussion: Review

Decision trees are:

- Non-linear: the decision boundary that results from splitting may end up being quite complicated
- Non-metric: they do not rely on the geometry of the space (inner products or distances)
- Non-parametric: they make no assumptions about the distribution of the data

Additional pros:

- Interpretable and simple to understand

Cons:

- Struggle to capture linear decision boundaries
- They have high variance and tend to overfit: they are sensitive to small changes in the training data (The ensemble techniques we discuss next can mitigate these issues)

# Bagging and Random Forests

# Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$

# Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$
- A **statistic** $s = s(\mathcal{D})$ is any function of the data:

# Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$
- A **statistic** $s = s(\mathcal{D})$ is any function of the data:
  - E.g., sample mean, sample variance, histogram, empirical data distribution

# Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$
- A **statistic** $s = s(\mathcal{D})$ is any function of the data:
  - E.g., sample mean, sample variance, histogram, empirical data distribution
- A statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ is a **point estimator** of $\theta$ if $\hat{\theta} \approx \theta$

# Recap: Bias and Variance of an Estimator

- Statistics are random, so they have probability distributions.
- The distribution of a statistic is called a **sampling distribution**.
- The standard deviation of the sampling distribution is called the **standard error**.

# Recap: Bias and Variance of an Estimator

- Statistics are random, so they have probability distributions.
- The distribution of a statistic is called a **sampling distribution**.
- The standard deviation of the sampling distribution is called the **standard error**.
- Some parameters of the sampling distribution we might be interested in:

Bias $\text{Bias}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}\right] - \theta.$

Variance $\text{Var}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}^2\right] - \mathbb{E}^2\left[\hat{\theta}\right].$

# Recap: Bias and Variance of an Estimator

- Statistics are random, so they have probability distributions.
- The distribution of a statistic is called a **sampling distribution**.
- The standard deviation of the sampling distribution is called the **standard error**.
- Some parameters of the sampling distribution we might be interested in:

  Bias $\text{Bias}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}\right] - \theta.$

  Variance $\text{Var}(\hat{\theta}) \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}^2\right] - \mathbb{E}^2\left[\hat{\theta}\right].$

- Why does variance matter if an estimator is unbiased?

# Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance $\sigma^2$: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate $\theta$.

# Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance $\sigma^2$: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate $\theta$.
- Its standard error is $\sqrt{\text{Var}(\hat{\theta})} = \sigma$

# Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance $\sigma^2$: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate $\theta$.
- Its standard error is $\sqrt{\text{Var}(\hat{\theta})} = \sigma$
- Consider a new estimator that takes the average of i.i.d. $\hat{\theta}_1, \ldots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathcal{D}^i)$.

## Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance $\sigma^2$: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\mathrm{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate $\theta$.
- Its standard error is $\sqrt{\mathrm{Var}(\hat{\theta})} = \sigma$
- Consider a new estimator that takes the average of i.i.d. $\hat{\theta}_1, \ldots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathcal{D}^i)$.

## Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance $\sigma^2$: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate $\theta$.
- Its standard error is $\sqrt{\text{Var}(\hat{\theta})} = \sigma$
- Consider a new estimator that takes the average of i.i.d. $\hat{\theta}_1, \ldots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathcal{D}^i)$.
- The average has the same expected value but smaller standard error (recall that $Var(cX) = c^2 Var(X)$, and that the $\hat{\theta}_i$-s are uncorrelated):

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \theta \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n} \tag{2}$$

# Averaging Independent Prediction Functions

- Suppose we have $B$ independent training sets, all drawn from the same distribution ($\mathcal{D} \sim p(\cdot \mid \theta)$).

# Averaging Independent Prediction Functions

- Suppose we have $B$ independent training sets, all drawn from the same distribution ($\mathcal{D} \sim p(\cdot \mid \theta)$).
- Our learning algorithm gives us $B$ prediction functions: $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$

# Averaging Independent Prediction Functions

- Suppose we have $B$ independent training sets, all drawn from the same distribution $(\mathcal{D} \sim p(\cdot \mid \theta))$.
- Our learning algorithm gives us $B$ prediction functions: $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$
- We will define the average prediction function as:

$$\hat{f}_{\text{avg}} \stackrel{\text{def}}{=} \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b \tag{3}$$

# Averaging Reduces Variance of Predictions

- The average prediction for $x_0$ is

$$\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_0).$$

- $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, but
- $\hat{f}_{\text{avg}}(x_0)$ has smaller variance:

$$\text{Var}(\hat{f}_{\text{avg}}(x_0)) = \frac{1}{B}\text{Var}\left(\hat{f}_1(x_0)\right)$$

# Averaging Reduces Variance of Predictions

- The average prediction for $x_0$ is

$$\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_0).$$

- $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, but
- $\hat{f}_{\text{avg}}(x_0)$ has smaller variance:

$$\mathrm{Var}(\hat{f}_{\text{avg}}(x_0)) = \frac{1}{B}\mathrm{Var}\left(\hat{f}_1(x_0)\right)$$

- Problem: in practice we don't have $B$ independent training sets!

# The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size $n$ drawn *with replacement* from $\mathcal{D}_n$

# The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size $n$ drawn *with replacement* from $\mathcal{D}_n$
- Some elements of $\mathcal{D}_n$ will show up multiple times, and some won't show up at all

# The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size $n$ drawn *with replacement* from $\mathcal{D}_n$
- Some elements of $\mathcal{D}_n$ will show up multiple times, and some won't show up at all
- Each $x_i$ has a probability of $(1 - 1/n)^n$ of not being included in a given bootstrap sample

# The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size $n$ drawn *with replacement* from $\mathcal{D}_n$
- Some elements of $\mathcal{D}_n$ will show up multiple times, and some won't show up at all
- Each $x_i$ has a probability of $(1 - 1/n)^n$ of not being included in a given bootstrap sample
- For large $n$,

$$\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \approx .368. \tag{4}$$

- So we expect ~63.2% of elements of $\mathcal{D}_n$ will show up at least once.

# The Bootstrap Method

**Definition**

A **bootstrap method** simulates $B$ independent samples from $P$ by taking $B$ bootstrap samples from the sample $\mathcal{D}_n$.

- Given original data $\mathcal{D}_n$, compute $B$ bootstrap samples $D_n^1, \ldots, D_n^B$.
- For each bootstrap sample, compute some function

$$\phi(D_n^1), \ldots, \phi(D_n^B)$$

- Use these values as though $D_n^1, \ldots, D_n^B$ were i.i.d. samples from $P$.
- This often ends up being very close to what we'd get with independent samples from $P$!

# Independent Samples vs. Bootstrap Samples

- Point estimator $\hat{\alpha} = \hat{\alpha}(\mathcal{D}_{100})$ for samples of size 100, for a synthetic case where the data generating distribution is known
- Histograms of $\hat{\alpha}$ based on
  - 1000 independent samples of size 100 (left), vs.
  - 1000 bootstrap samples of size 100 (right)

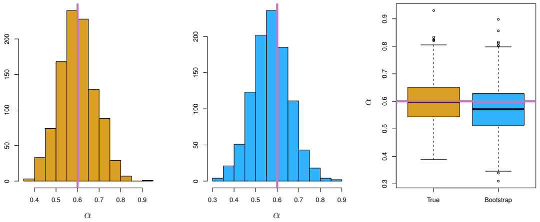



Figure 5.10 from *ISLR* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

# Ensemble Methods

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them
- Parallel ensemble (e.g., bagging): models are built independently

# Ensemble Methods

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them
- Parallel ensemble (e.g., bagging): models are built independently
- Sequential ensemble (e.g., boosting): models are built sequentially

# Ensemble Methods

**Key ideas:**

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them
- Parallel ensemble (e.g., bagging): models are built independently
- Sequential ensemble (e.g., boosting): models are built sequentially
    - We try to find new learners that do well where previous learners fall short

# Bagging: Bootstrap Aggregation

- We draw $B$ bootstrap samples $D^1, \ldots, D^B$ from original data $\mathcal{D}$

# Bagging: Bootstrap Aggregation

- We draw $B$ bootstrap samples $D^1, \ldots, D^B$ from original data $\mathcal{D}$
- Let $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_B$ be the prediction functions resulting from training on $D^1, \ldots, D^B$, respectively

# Bagging: Bootstrap Aggregation

- We draw $B$ bootstrap samples $D^1, \ldots, D^B$ from original data $\mathcal{D}$
- Let $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_B$ be the prediction functions resulting from training on $D^1, \ldots, D^B$, respectively
- The **bagged prediction function** is a *combination* of these:

$$\hat{f}_{\text{avg}}(x) = \text{Combine}\left(\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)\right)$$

# Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees

# Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead

# Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead
- Increasing the number of trees we use in bagging does not lead to overfitting

# Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead
- Increasing the number of trees we use in bagging does not lead to overfitting
- Is there a downside, compared to having a single decision tree?

# Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead
- Increasing the number of trees we use in bagging does not lead to overfitting
- Is there a downside, compared to having a single decision tree?
- Yes: if we have many trees, the bagged predictor is much less interpretable

# Aside: Out-of-Bag Error Estimation

- Recall that each bagged predictor was trained on about 63% of the data.
- The remaining 37% are called **out-of-bag (OOB)** observations.

# Aside: Out-of-Bag Error Estimation

- Recall that each bagged predictor was trained on about 63% of the data.
- The remaining 37% are called **out-of-bag (OOB)** observations.
- For $i$th training point, let

$$S_i = \left\{ b \mid D^b \text{ does not contain } i\text{th point} \right\}$$

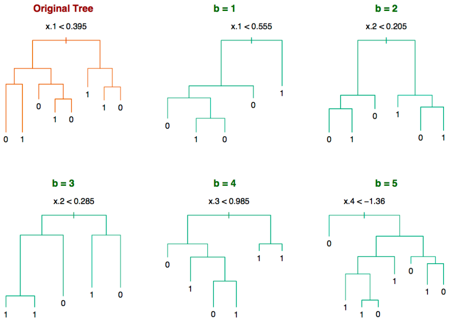
- The **OOB prediction** on $x_i$ is

$$\hat{f}_{\mathrm{OOB}}(x_i) = \frac{1}{|S_i|} \sum_{b \in S_i} \hat{f}_b(x_i)$$

- The OOB error is a good estimate of the test error
- Similar to cross validation error: both are computed on the training set

# Applying Bagging to Classification Trees

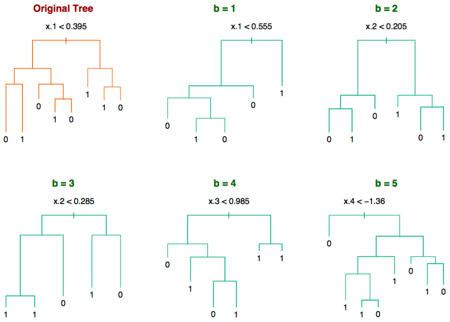
- Input space $\mathcal{X} = \mathbb{R}^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.




From HTF Figure 8.9

# Applying Bagging to Classification Trees

- Input space $\mathcal{X} = \mathbf{R}^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.

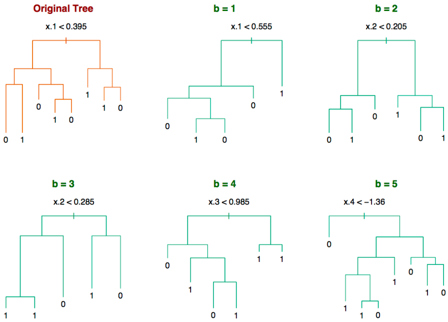



- Each bootstrap tree is quite different: different splitting variable at the root!

From HTF Figure 8.9

# Applying Bagging to Classification Trees

- Input space $\mathcal{X} = \mathbb{R}^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.

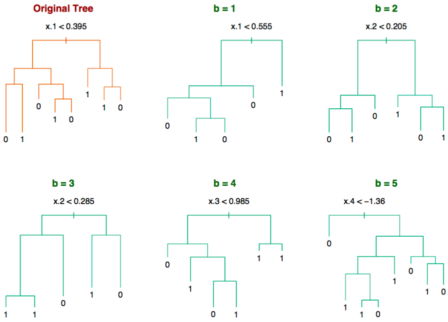



- Each bootstrap tree is quite different: different splitting variable at the root!
- **High variance**: small perturbations of the training data lead to a high degree of model variability

From HTF Figure 8.9

# Applying Bagging to Classification Trees

- Input space $\mathcal{X} = R^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.




- Each bootstrap tree is quite different: different splitting variable at the root!
- **High variance**: small perturbations of the training data lead to a high degree of model variability
- Bagging helps most when the base learners are relatively unbiased but have high variance (exactly the case for decision trees)

From HTF Figure 8.9

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large $n$, the covariance term dominates, limiting the benefits of averaging

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large $n$, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large $n$, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are
  - independent samples from the training set, but

## Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\text{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large $n$, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are
  - independent samples from the training set, but
  - not independent samples from $P_{\mathcal{X}\times\mathcal{Y}}$

# Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\mathrm{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large $n$, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are
  - independent samples from the training set, but
  - not independent samples from $P_{\mathcal{X}\times\mathcal{Y}}$
- Can we reduce the dependence between $\hat{f}_i$'s?

# Random Forests

**Key idea**

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before

# Random Forests

**Key idea**

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size $m$
  - This prevents a situation where all trees are dominated by the same small number of strong features (and are therefore too similar to each other)

# Random Forests

**Key idea**

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size $m$
  - This prevents a situation where all trees are dominated by the same small number of strong features (and are therefore too similar to each other)
- We typically choose $m \approx \sqrt{p}$, where $p$ is the number of features (or we can choose $m$ using cross validation)

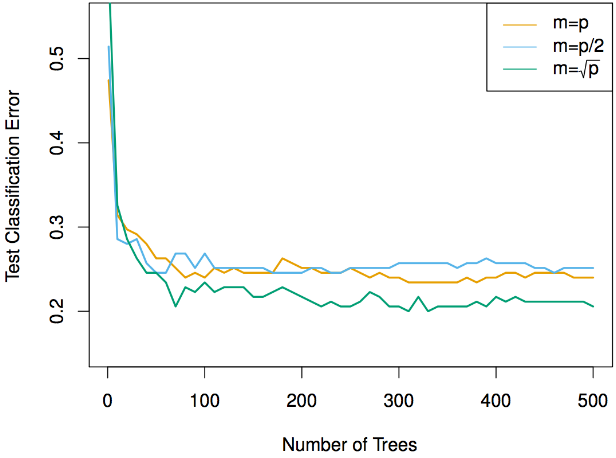
# Random Forests

**Key idea**

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size $m$
  - This prevents a situation where all trees are dominated by the same small number of strong features (and are therefore too similar to each other)
- We typically choose $m \approx \sqrt{p}$, where $p$ is the number of features (or we can choose $m$ using cross validation)
- If $m = p$, this is just bagging

# Random Forests: Effect of $m$




From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

# Review

- The usual approach is to build very deep trees—low bias but high variance

# Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate

# Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate
- Use bootstrap to simulate many data samples from one dataset
  - $\implies$ Bagged decision trees

# Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate
- Use bootstrap to simulate many data samples from one dataset
  - $\Longrightarrow$ Bagged decision trees
- But bootstrap samples (and the induced models) are correlated

# Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate
- Use bootstrap to simulate many data samples from one dataset
  - $\implies$ Bagged decision trees
- But bootstrap samples (and the induced models) are correlated
- Ensembling works better when we combine a diverse set of prediction functions
  - $\implies$ Random forests: select a random subset of features for each decision tree

# Boosting

# Boosting: Overview

Bagging Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel (on different datasets obtained through sampling).

# Boosting: Overview

**Bagging** Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel (on different datasets obtained through sampling).

**Boosting** Reduce the error rate of a high bias estimator by ensembling many estimators trained in sequence (without bootstrapping).

# Boosting: Overview

**Bagging** Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel (on different datasets obtained through sampling).

**Boosting** Reduce the error rate of a high bias estimator by ensembling many estimators trained in sequence (without bootstrapping).

- Like bagging, boosting is a general method that is particularly popular with decision trees.

# Boosting: Overview

**Bagging** Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel (on different datasets obtained through sampling).

**Boosting** Reduce the error rate of a high bias estimator by ensembling many estimators trained in sequence (without bootstrapping).

- Like bagging, boosting is a general method that is particularly popular with decision trees.
- Main intuition: instead of fitting the data very closely using a large decision tree, train gradually, using a sequence of simpler trees

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:
  - Each weak learner focuses on different training examples (*reweighted data*)

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:
  - Each weak learner focuses on different training examples (*reweighted data*)
  - Weak learners make different contributions to the final prediction (*reweighted classifier*)

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:
  - Each weak learner focuses on different training examples (*reweighted data*)
  - Weak learners make different contributions to the final prediction (*reweighted classifier*)
- A set of smaller, simpler trees may improve interpretability

# Boosting: Overview

- A **weak/base learner** is a classifier that does slightly better than chance.
- Weak learners are like rules of thumb:
  - "Inheritance" $\implies$ spam
  - From a friend $\implies$ not spam
- **Key idea**:
  - Each weak learner focuses on different training examples (*reweighted data*)
  - Weak learners make different contributions to the final prediction (*reweighted classifier*)
- A set of smaller, simpler trees may improve interpretability
- We'll focus on a specific implementation, AdaBoost (Freund & Schapire, 1997)

# AdaBoost: Setting

- Binary classification: $\mathcal{Y} = \{-1, 1\}$

# AdaBoost: Setting

- Binary classification: $\mathcal{Y} = \{-1, 1\}$
- Base hypothesis space $\mathcal{H} = \{h : \mathcal{X} \to \{-1, 1\}\}$.

# AdaBoost: Setting

- Binary classification: $\mathcal{Y} = \{-1, 1\}$
- Base hypothesis space $\mathcal{H} = \{h : \mathcal{X} \to \{-1, 1\}\}$.
- Typical base hypothesis spaces:
  - **Decision stumps** (tree with a single split)
  - Trees with few terminal nodes
  - Linear decision functions

# Weighted Training Set

Each base learner is trained on weighted data.

- Training set $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$.
- Weights $(w_1, \ldots, w_n)$ associated with each example.

# Weighted Training Set

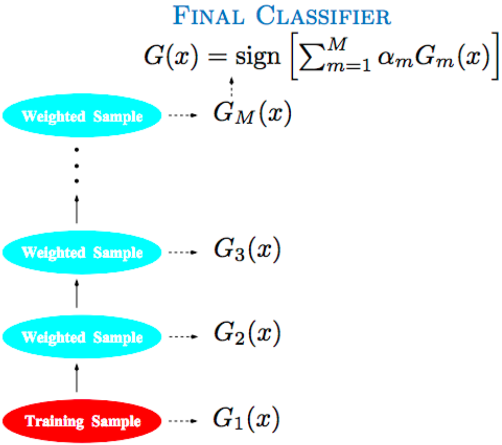
Each base learner is trained on weighted data.

- Training set $\mathcal{D} = ((x_1, y_1), \dots, (x_n, y_n))$.
- Weights $(w_1, \dots, w_n)$ associated with each example.
- **Weighted empirical risk**:

$$\hat{R}_n^w(f) \stackrel{\text{def}}{=} \frac{1}{W} \sum_{i=1}^{n} w_i \ell(f(x_i), y_i) \quad \text{where } W = \sum_{i=1}^{n} w_i$$

- Examples with larger weights affect the loss more.

# AdaBoost: Schematic




From ESL Figure 10.1

# AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$

# AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$
- Repeat for $m = 1, \ldots, M$ (where $M$ is the number of classifiers we plan to train):

# AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$
- Repeat for $m = 1, \ldots, M$ (where $M$ is the number of classifiers we plan to train):
  - Train base classifier $G_m(x)$ on the weighted training data; this classifier may not fit the data well

# AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$
- Repeat for $m = 1, \ldots, M$ (where $M$ is the number of classifiers we plan to train):
  - Train base classifier $G_m(x)$ on the weighted training data; this classifier may not fit the data well
  - Increase the weight of the points misclassified by $G_m(x)$ (this is the key idea of boosting!)

# AdaBoost: Sketch of the Algorithm

- Start with equal weights for all training points: $w_1 = \cdots = w_n = 1$
- Repeat for $m = 1, \ldots, M$ (where $M$ is the number of classifiers we plan to train):
  - Train base classifier $G_m(x)$ on the weighted training data; this classifier may not fit the data well
  - Increase the weight of the points misclassified by $G_m(x)$ (this is the key idea of boosting!)
- Our final prediction is $G(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

# AdaBoost: Classifier Weights

- Our final prediction is $G(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
- We would like $\alpha_m$ to be:
  - Nonnegative
  - Larger when $G_m$ fits its weighted training data well
- The **weighted 0-1 error** of $G_m(x)$ is

$$\text{err}_m = \frac{1}{W}\sum_{i=1}^{n} w_i 1(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^{n} w_i.$$

- $\text{err}_m \in [0,1]$

# AdaBoost: Classifier Weights

- The weight of classifier $G_m(x)$ is $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$

# AdaBoost: Classifier Weights

- The weight of classifier $G_m(x)$ is $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$


- Higher weighted error $\implies$ lower weight

# Adaboost: Example Reweighting

- We train $G_m$ to minimize weighted error; the resulting error rate is $\text{err}_m$
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of $G_m$ in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

# Adaboost: Example Reweighting

- We train $G_m$ to minimize weighted error; the resulting error rate is $\text{err}_m$
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of $G_m$ in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

- Suppose $w_i$ is the weight of example $x_i$ before training:

# Adaboost: Example Reweighting

- We train $G_m$ to minimize weighted error; the resulting error rate is $\text{err}_m$
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of $G_m$ in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

- Suppose $w_i$ is the weight of example $x_i$ before training:
  - If $G_m$ classifies $x_i$ correctly, keep $w_i$ as is

# Adaboost: Example Reweighting

- We train $G_m$ to minimize weighted error; the resulting error rate is $\text{err}_m$
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of $G_m$ in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

- Suppose $w_i$ is the weight of example $x_i$ before training:
    - If $G_m$ classifies $x_i$ correctly, keep $w_i$ as is
    - Otherwise, increase $w_i$:

$$\begin{aligned} w_i &\leftarrow w_i e^{\alpha_m} \\ &= w_i \left(\frac{1-\text{err}_m}{\text{err}_m}\right) \end{aligned}$$

# Adaboost: Example Reweighting

- We train $G_m$ to minimize weighted error; the resulting error rate is $\text{err}_m$
- Then $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of $G_m$ in the final ensemble

We want the next base learner to focus more on examples misclassified by the previous learner.

- Suppose $w_i$ is the weight of example $x_i$ before training:
    - If $G_m$ classifies $x_i$ correctly, keep $w_i$ as is
    - Otherwise, increase $w_i$:

$$\begin{aligned} w_i &\leftarrow w_i e^{\alpha_m} \\ &= w_i \left(\frac{1-\text{err}_m}{\text{err}_m}\right) \end{aligned}$$

    - If $G_m$ is a strong classifier overall, then its $\alpha_m$ will be large; this means that if $x_i$ is misclassified, $w_i$ will increase to a greater extent

# AdaBoost with Decision Stumps

- After 1 round:

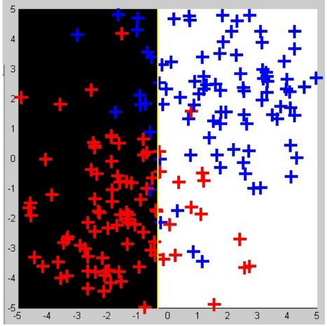



Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

# AdaBoost with Decision Stumps

- After 3 rounds:

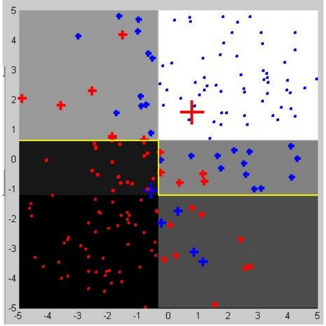



Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

# AdaBoost with Decision Stumps

- After 120 rounds:

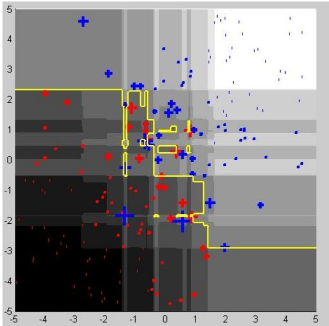
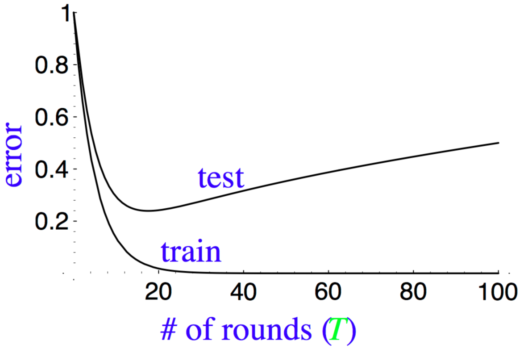



Figure: Size of plus sign represents weight of example. Blackness represents preference for red class; whiteness represents preference for blue class.

KPM Figure 16.10

# Does AdaBoost overfit?

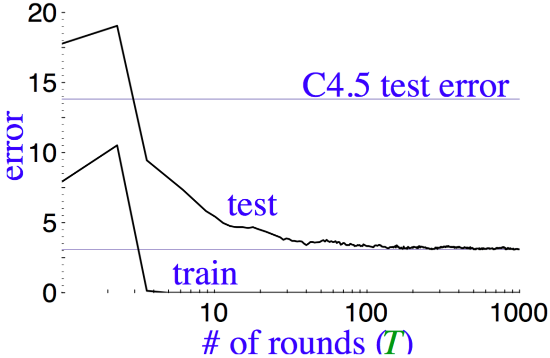
- Does a large number of rounds of boosting lead to overfitting?
- If we were overfitting, the learning curves would look like:




From Rob Schapire's NIPS 2007 Boosting tutorial.

# Learning Curves for AdaBoost

- AdaBoost is usually quite resistant to overfitting
- The test error continues to decrease even after the training error drops to zero!




From Rob Schapire's NIPS 2007 Boosting tutorial.

# Summary

- Boosting is used to reduce bias from shallow decision trees

# Summary

- Boosting is used to reduce bias from shallow decision trees
- AdaBoost is a very powerful off-the-self classifier!

# Summary

- Boosting is used to reduce bias from shallow decision trees
- AdaBoost is a very powerful off-the-self classifier!
- Next week

# Summary

- Boosting is used to reduce bias from shallow decision trees
- AdaBoost is a very powerful off-the-self classifier!
- Next week
  - What is the objective function of AdaBoost?

# Summary

- Boosting is used to reduce bias from shallow decision trees
- AdaBoost is a very powerful off-the-self classifier!
- Next week
  - What is the objective function of AdaBoost?
  - Generalizations to other loss functions