Lecture 9: Decision Trees and Random Forests

STATS 202: Statistical Learning and Data Science

Linh Tran

tranlm@stanford.edu

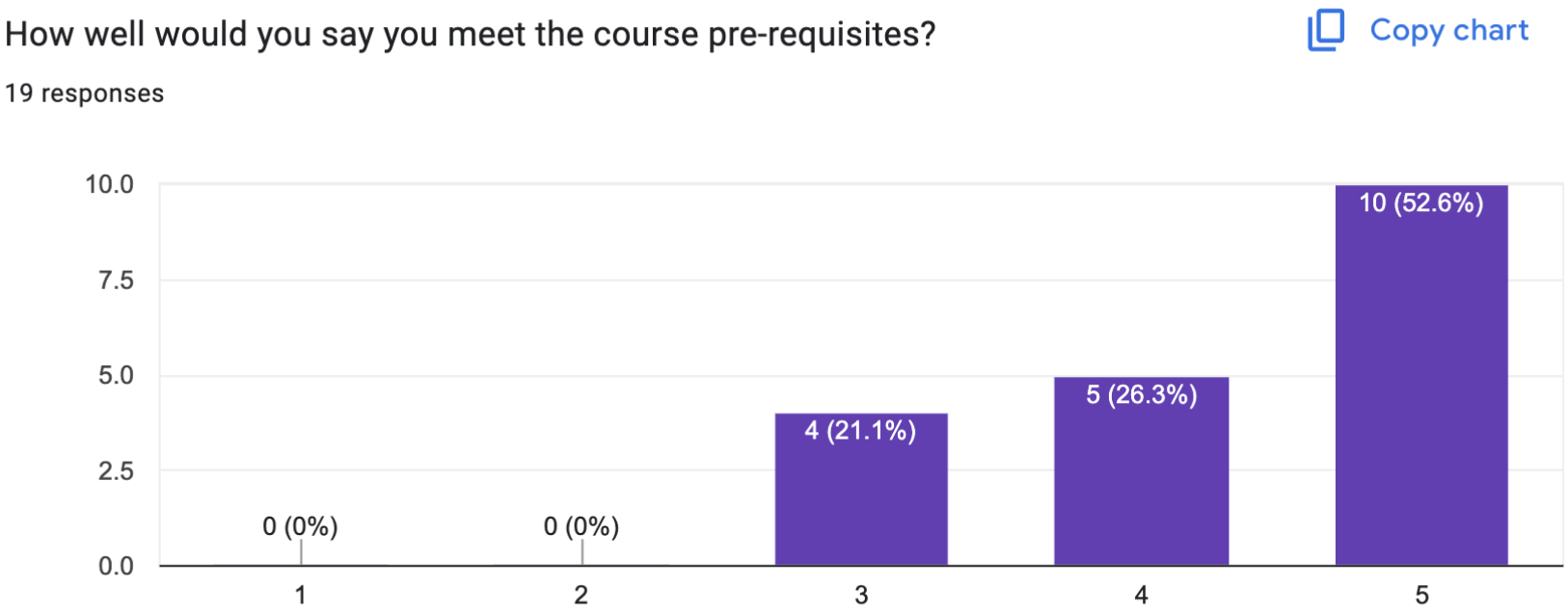
Department of Statistics
Stanford University

July 23, 2025

Announcements

- Midterms are still being graded.
- Homework 3 due next Monday
- Final predictions due in 2.5 weeks
- Survey results are out (84% completion rate)

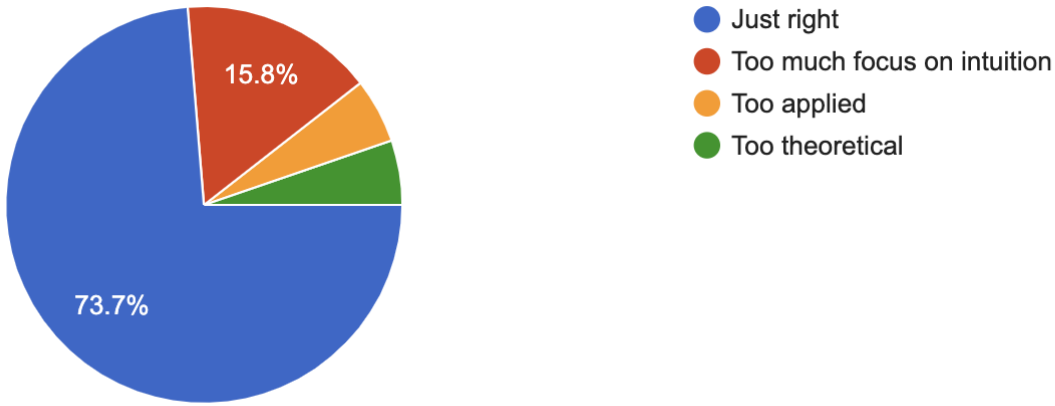
Survey
Course pre-requisites



How do you feel about the combination of course material overall (including lectures, homework, and exams)?

Copy chart

19 responses

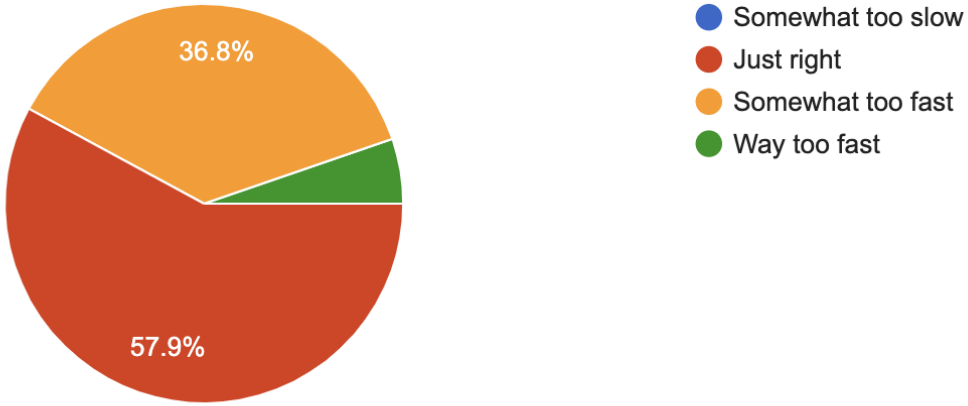


Survey
Course pace

How do you feel about the course pace?

19 responses

Copy chart

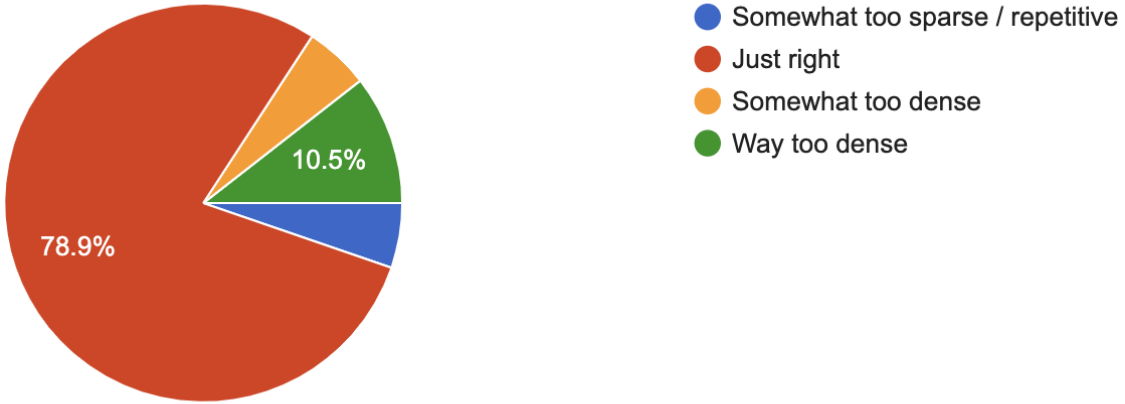


Survey
Course density

How do you feel about the material density?

19 responses

Copy chart

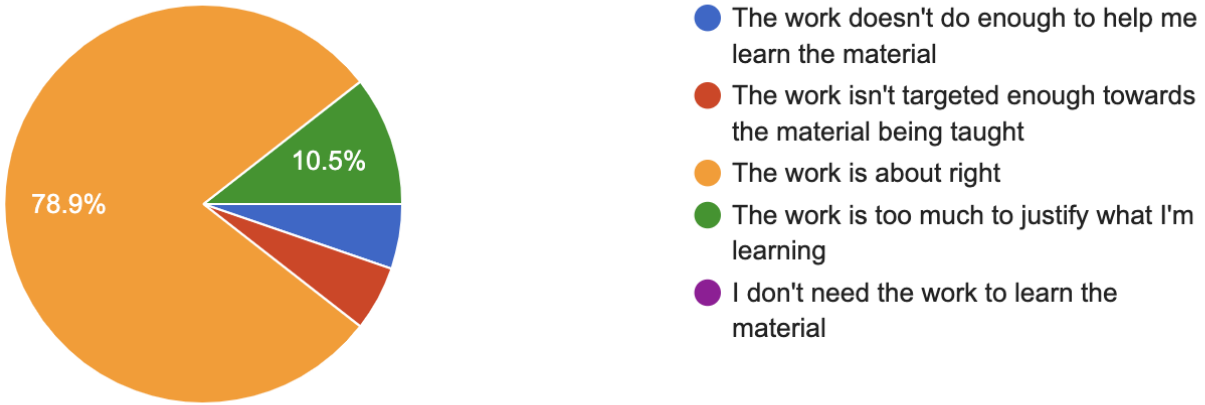


Survey
Course workload

How do you feel about the course workload?

19 responses

Copy chart



Survey
Course feedback (paraphrased)

- Well structured / informative
- Excellent
- Nice practical / industry perspectives
- Staff is great / helpful
- Can't access Google calendar

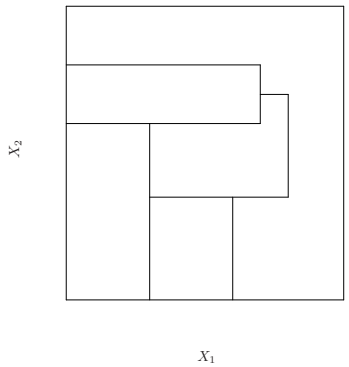
Survey
Course feedback (paraphrased)

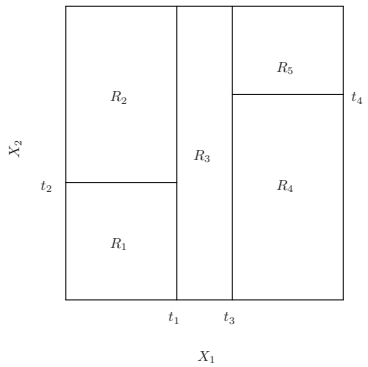
- Do more:
  - TAs endorse some learning posts
  - Keep whiteboarding
  - Call on students to answer questions
  - More math
  - More hands on examples
  - Try offering hybrid office hours
- Do less:
  - Don't deduct hw for page mis-alignment
  - Theoretical problems / proofs
  - Don't go so fast

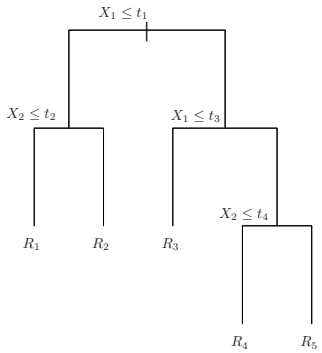
Outline

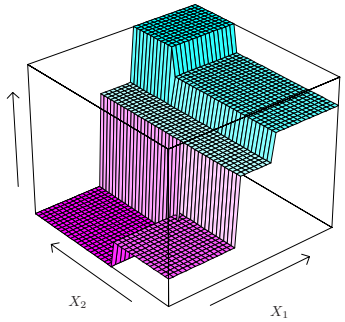
- Decision trees
  - Regression trees
  - Classification trees
  - Advantages / disadvantages
  - Misc details
- Bagging
- Random Forests

Decision trees, 10,000 foot view



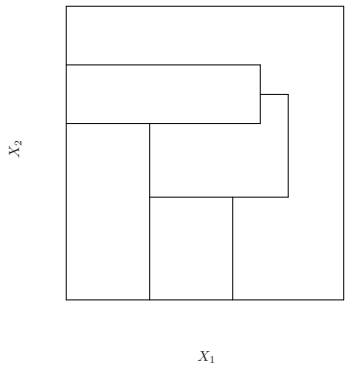


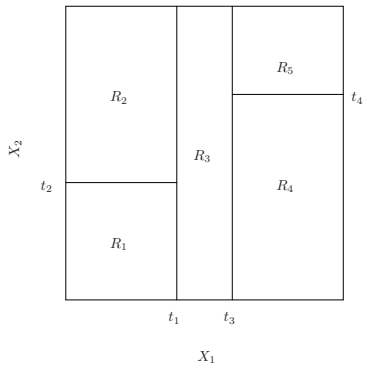


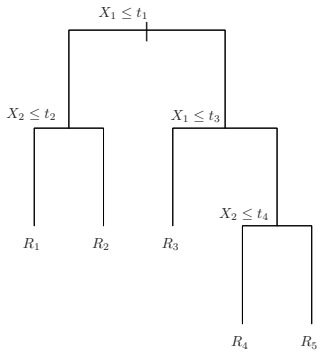


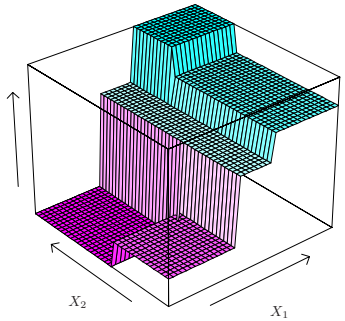
1. Find a partition of the space of predictors.
2. Predict a constant in each set of the partition.
3. The partition is defined by splitting the range of one predictor at a time.

Decision trees, 10,000 foot view









1. Find a partition of the space of predictors.
2. Predict a constant in each set of the partition.
3. The partition is defined by splitting the range of one predictor at a time.

   $\rightarrow$ Not all partitions are possible.

Our estimated function

Given our split regions R_m, we can come up with a prediction via

$$\hat{f}_n(x) = \sum_{m=1}^{|\mathcal{R}|} c_m \mathbb{I}(x \in R_m) \tag{1}$$

where $R_1, ..., R_m \in \mathcal{R}$ and

$$\hat{c}_m = \frac{1}{\sum_i^n \mathbb{I}(x_i \in R_m)} \sum_{i=1}^{n} y_i \mathbb{I}(x_i \in R_m) \tag{2}$$

Example: Predicting a baseball player's salary

1986 Baseball Dataset (n=322)

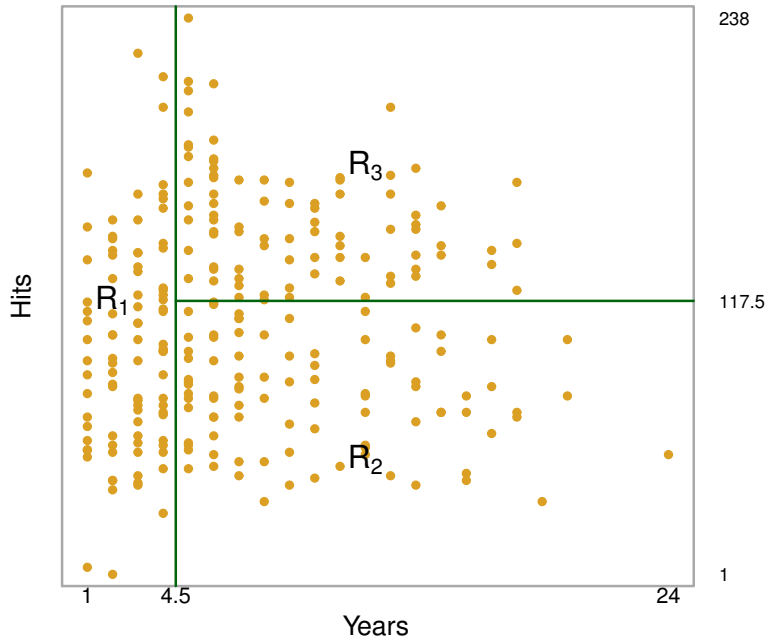
Features:

- Years in league
- RBI
- Putouts
- Walks
- Runs
- ...

Label:

- Salary ($K)

Example: Predicting a baseball player's salary



The prediction for a point in R_i is the average of the training points in R_i.

How is a decision tree built?

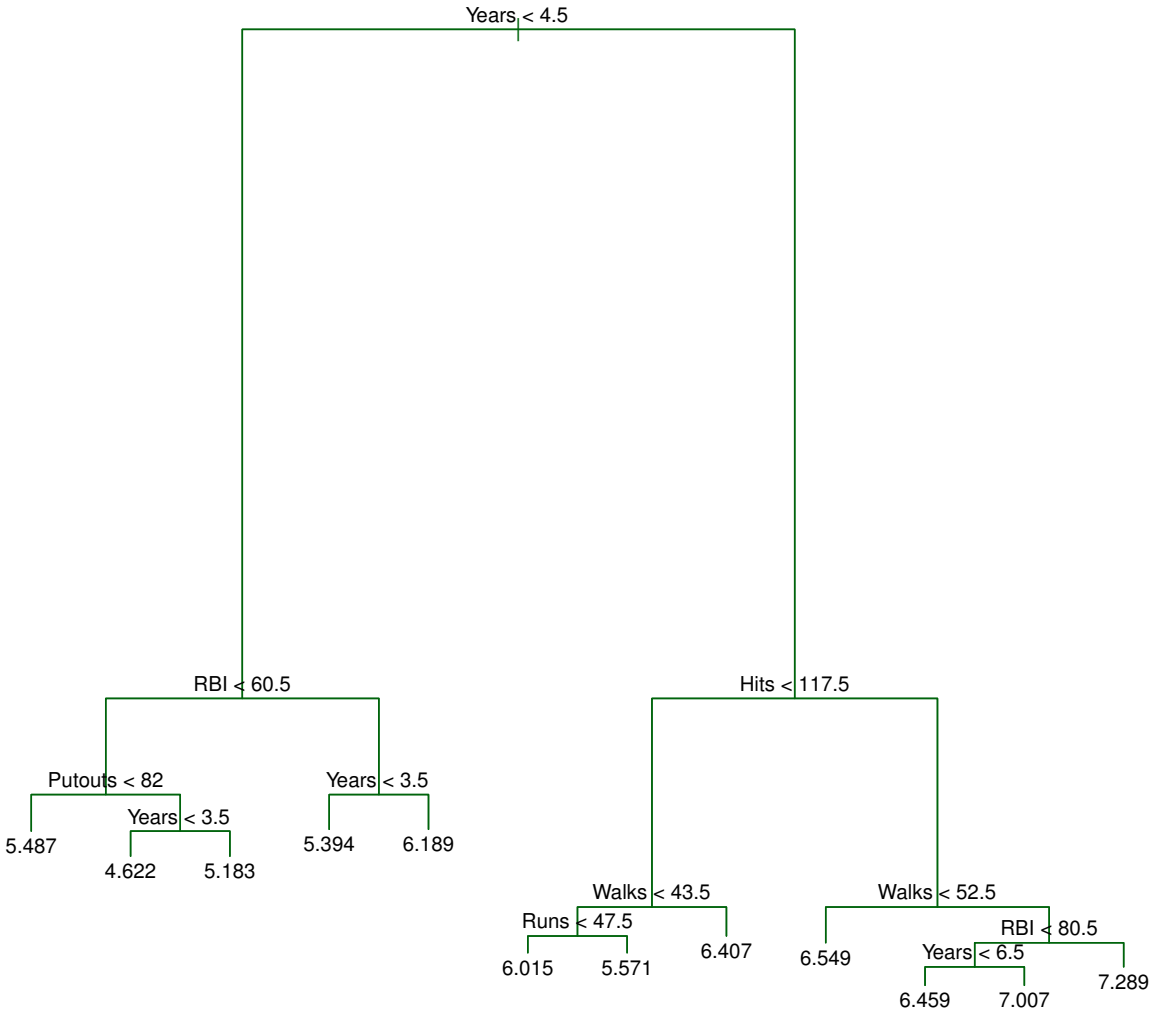
Using a *greedy* approach:

- Start with a single region R_1, and iterate:
  - Select a region R_k, a predictor X_j, and a splitting point s, such that splitting R_k with the criterion $X_j < s$ produces the largest decrease in RSS:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2$$

  - Redefine the regions with this additional split.
- Terminate when there are 5 observations or fewer in each region.
- This grows the tree from the root towards the leaves.

How is a decision tree built?



How do we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.

How do we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.

  $\rightarrow$ There are too many possibilities, so we would still over fit.

How do we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.

  $\rightarrow$ There are too many possibilities, so we would still over fit.

- **Idea 2:** Stop growing the tree when the RSS doesn't drop by more than a threshold with any new cut.

How do we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.

  $\rightarrow$ There are too many possibilities, so we would still over fit.

- **Idea 2:** Stop growing the tree when the RSS doesn't drop by more than a threshold with any new cut.

  $\rightarrow$ In our greedy algorithm, it is possible to find good cuts after bad ones.

How do we control overfitting?

Solution:

Prune a large tree from the leaves to the root.

- **Weakest link pruning:**
  - Starting with T_0, substitute a subtree with a leaf to obtain T_1, by minimizing:
$$\frac{RSS(T_1) - RSS(T_0)}{|T_0| - |T_1|}.$$
  - Iterate this pruning to obtain a sequence $T_0, T_1, T_2, \ldots, T_m$ where T_m is the null tree.
  - Select the optimal tree T_i by cross validation.

How do we control overfitting?

... or an equivalent procedure

- **Cost complexity pruning:**
  - Solve the problem:

$$\text{minimize} \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T|.$$

  - When $\alpha = \infty$, we select the null tree.
  - When $\alpha = 0$, we select the full tree.
  - The solution for each α is among $T_1, T_2, \ldots, T_m$ from weakest link pruning.
  - Choose the optimal α (the optimal T_i) by cross validation.

Cross validation

1. Construct a sequence of trees $T_0, \ldots, T_m$ for a range of values of α.
2. Split the training points into 10 folds.
3. For $k = 1, \ldots, 10$,
   - For each tree T_i, use every fold except the kth to estimate the averages in each region.
   - For each tree T_i, calculate the RSS in the test fold.
4. For each tree T_i, average the 10 test errors, and select the value of α that minimizes the error.

Cross validation

1. Construct a sequence of trees $T_0, \ldots, T_m$ for a range of values of α.
2. Split the training points into 10 folds.
3. For $k = 1, \ldots, 10$,
   - For each tree T_i, use every fold except the kth to estimate the averages in each region.
   - For each tree T_i, calculate the RSS in the test fold.
4. For each tree T_i, average the 10 test errors, and select the value of α that minimizes the error.

THIS IS THE WRONG WAY TO DO CROSS VALIDATION!

Cross validation, the right way

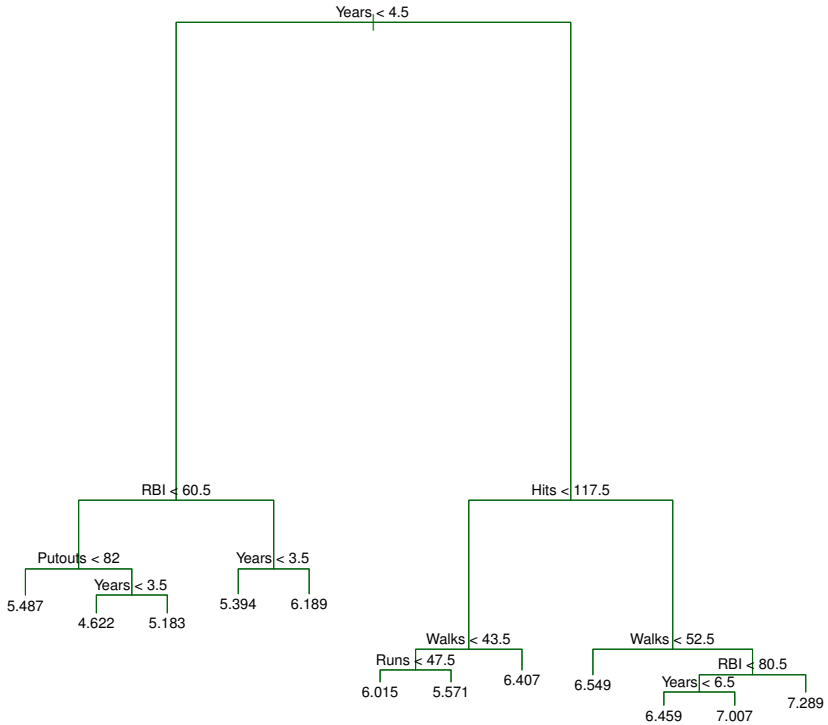
1. Split the training points into 10 folds.
2. For $k = 1, \ldots, 10$, using every fold except the kth:
   - Construct a sequence of trees $T_1, \ldots, T_m$ for a range of values of α, and find the prediction for each region in each one.
   - For each tree T_i, calculate the RSS on the test set.
3. For each tree T_i, average the 10 test errors, and select the value of α that minimizes the error.

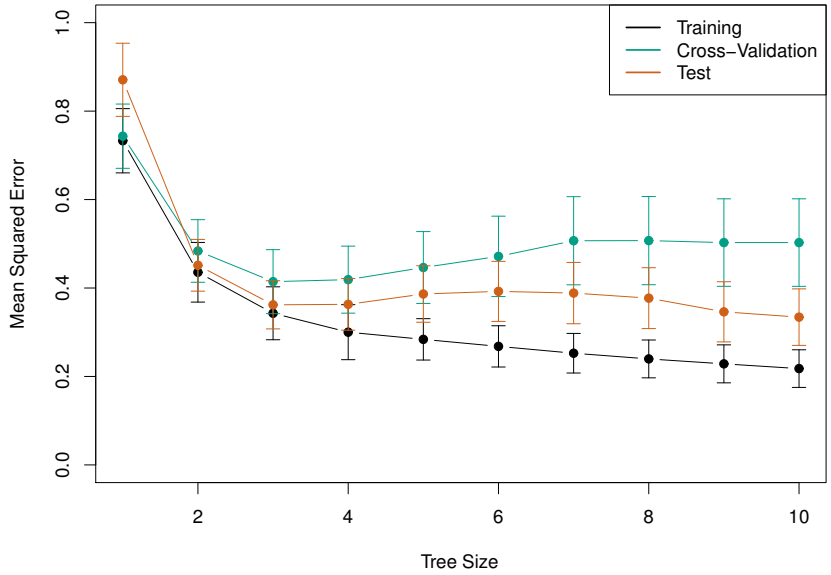
Cross validation, the right way

1. Split the training points into 10 folds.
2. For $k = 1, \ldots, 10$, using every fold except the kth:
   - Construct a sequence of trees $T_1, \ldots, T_m$ for a range of values of α, and find the prediction for each region in each one.
   - For each tree T_i, calculate the RSS on the test set.
3. For each tree T_i, average the 10 test errors, and select the value of α that minimizes the error.

Note: We are doing all fitting, including the construction of the trees, using only the training data.

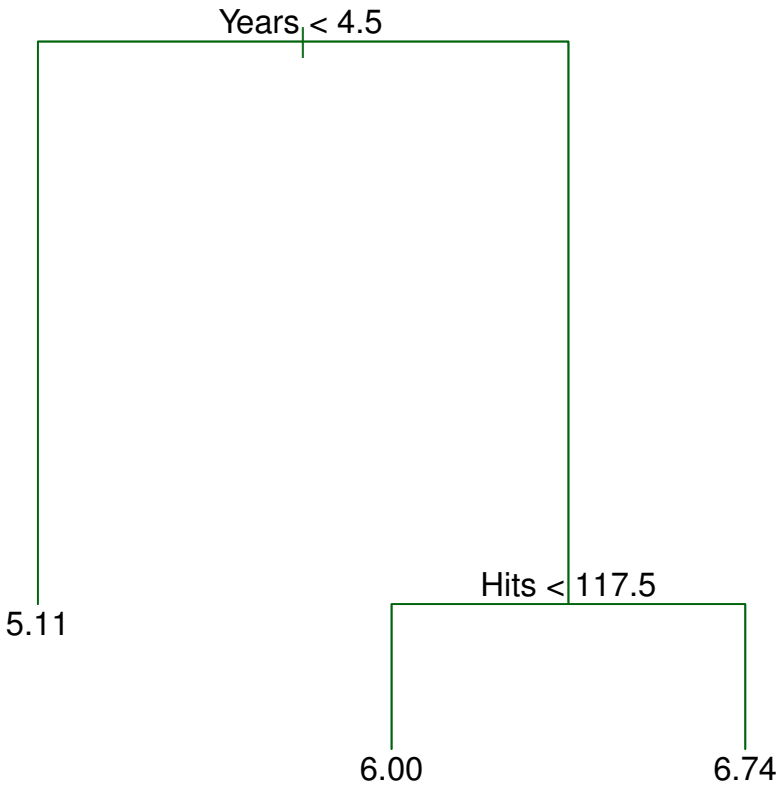
Example. Predicting baseball salaries

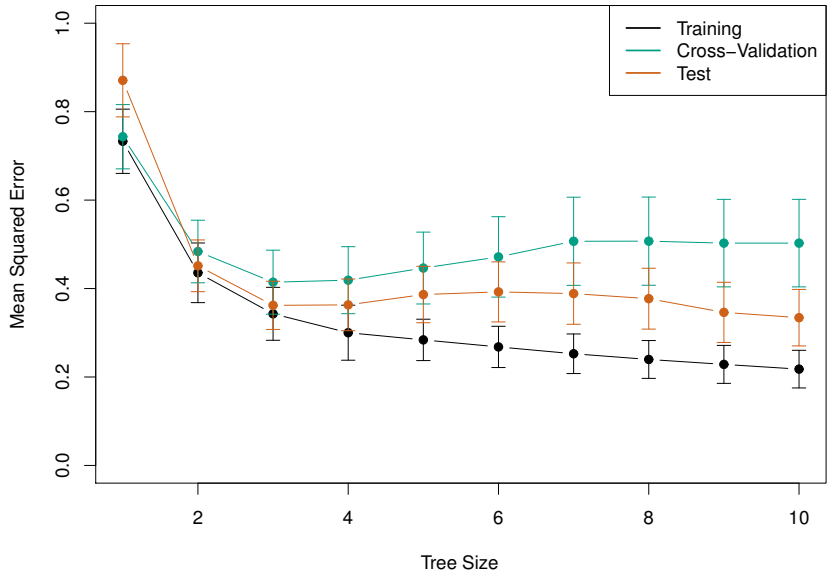




Unpruned tree (size=12)

Example. Predicting baseball salaries





Short tree (size=3)

Classification trees

Work much like regression trees.

- We predict the response by **majority vote**, i.e. pick the most common class in every region.
- Instead of trying to minimize the RSS:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2$$

we minimize a classification loss function.

- Multiple losses to choose from

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

- The Gini index:

$$\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

where $\hat{p}_{m,k}$ is the proportion of class k within R_m, and q_m is the proportion of samples in R_m.

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

- The Gini index:

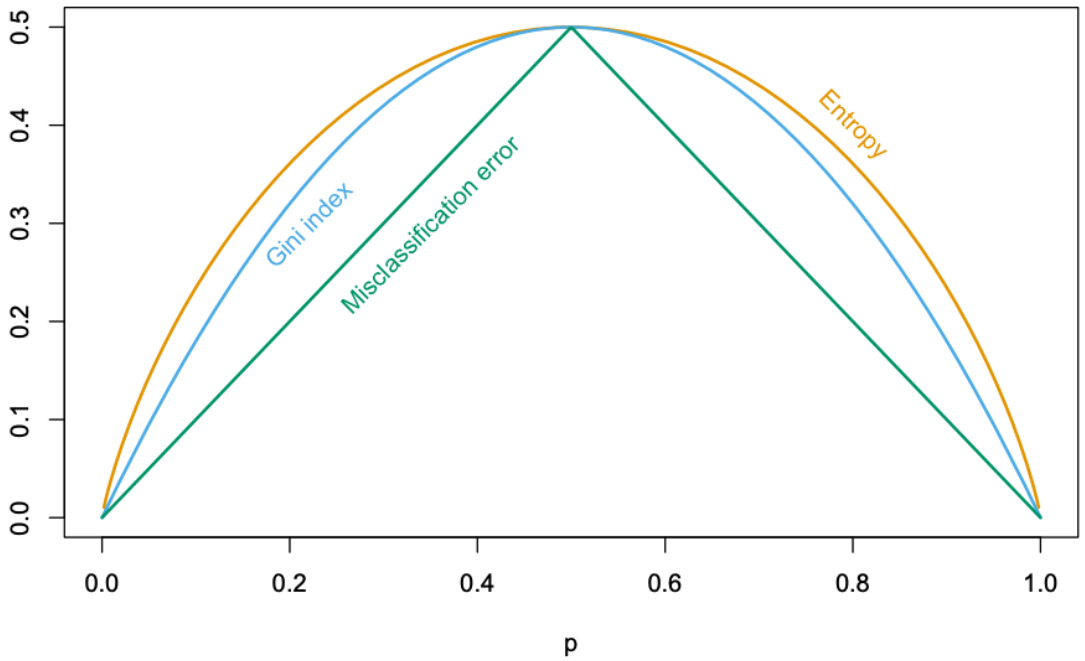
$$\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  where $\hat{p}_{m,k}$ is the proportion of class k within R_m, and q_m is the proportion of samples in R_m.

- The entropy:

$$-\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}).$$

Classification losses



Losses for 2-class classication, as a function of the proportion p. Entropy has been scaled to pass through (0.5, 0.5).

Remarks:

- **Motivation for the Gini index:**

  Expected misclassification rate under random sampling.

$$\begin{aligned}
\sum_{k=1}^{K} \hat{p}_k(1-\hat{p}_k) &= \sum_{k=1}^{K}(\hat{p}_k - \hat{p}_k^2) \\
&= \sum_{k=1}^{K} \hat{p}_k - \sum_{k=1}^{K} \hat{p}_k^2 \\
&= 1 - \sum_{k=1}^{K} \hat{p}_k^2 \\
&= \sum_{j=1}^{K} \sum_{k \neq j} p_j p_k
\end{aligned}$$

Classification losses

Remarks:

- The Gini index and entropy are better measures of the purity of a region, i.e. they are low when the region is mostly one category.
- It is typical to use the Gini index or entropy for growing the tree, while using the misclassification rate when pruning the tree.

Example. Heart dataset.

Heart disease dataset (n=303)

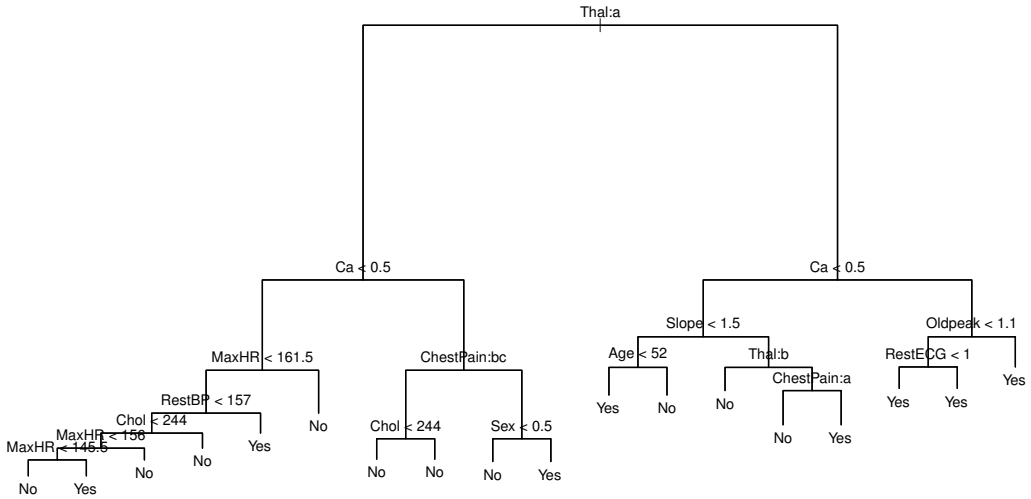
Features:

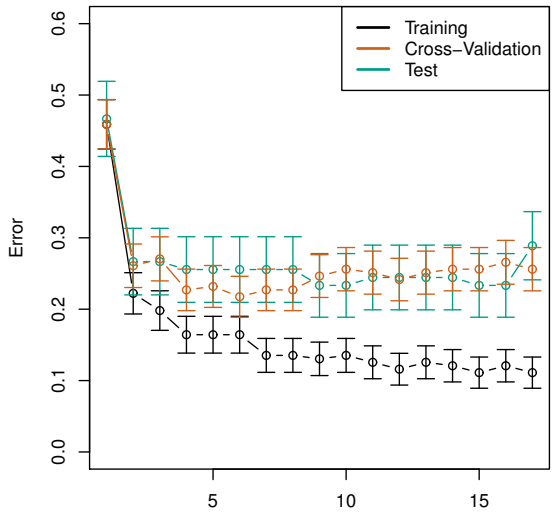
- Age
- Cholesterol level
- Resting BP
- Sex
- Chest Pain
- ...

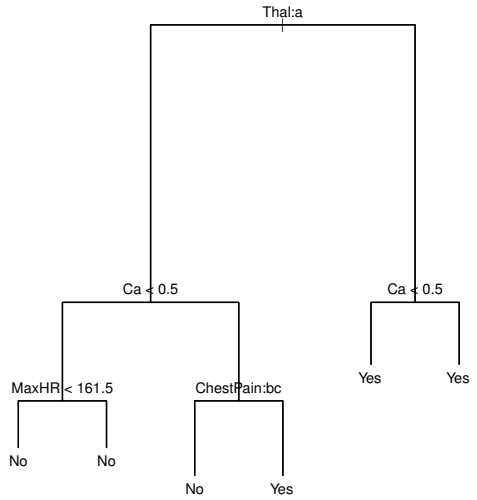
Label:

- Heart disease

Example. Heart dataset.

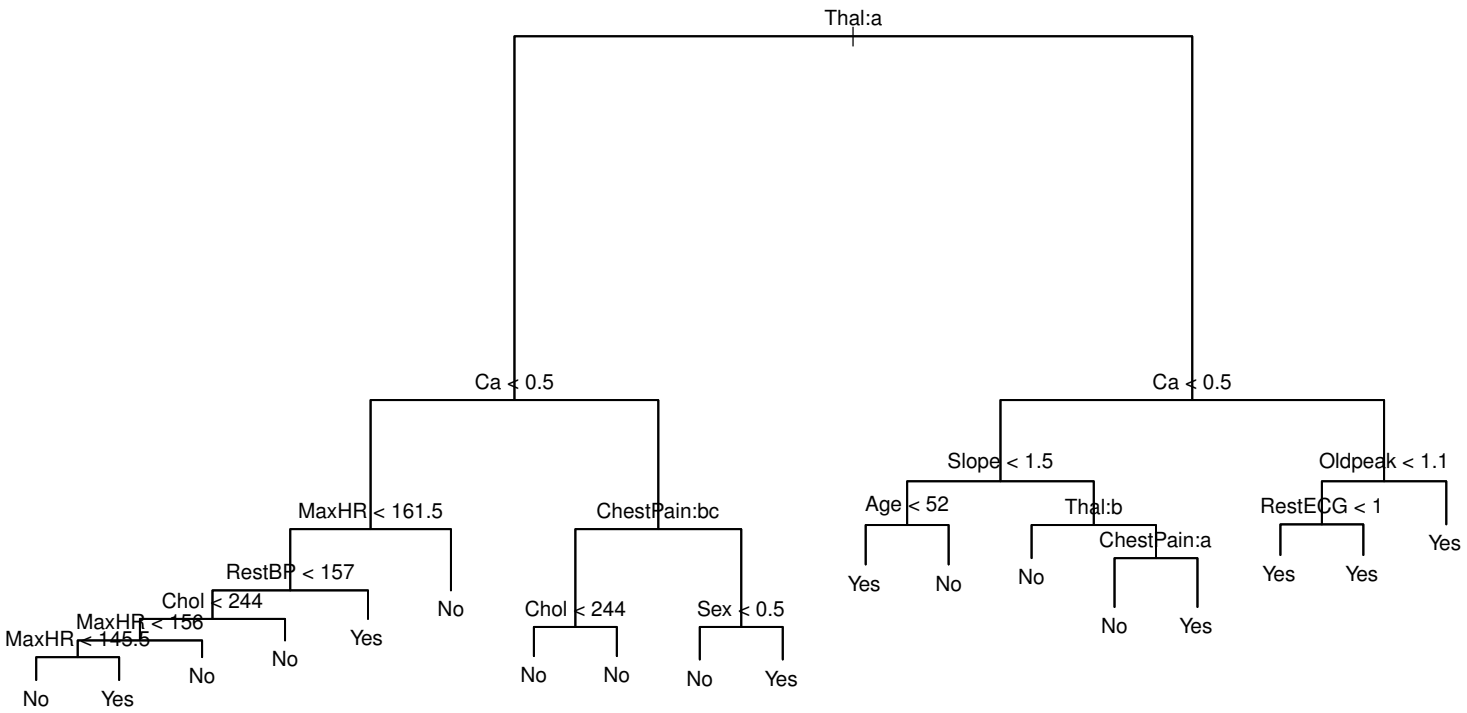






CART Issues: categorical data

Question: How do we deal with categorical predictors?



Categorical predictors

- If there are only 2 categories, then the split is obvious. We don't have to choose the splitting point s, as for a numerical variable.
- If there are more than 2 categories:
  - Order the categories according to the average of the response:

$$\texttt{ChestPain} : \texttt{a} > \texttt{ChestPain} : \texttt{c} > \texttt{ChestPain} : \texttt{b}$$

  - Treat as a numerical variable with this ordering, and choose a splitting point s.
- This is the optimal way of partitioning.

Missing data

Problem: If a sample is missing variable X_j, and a tree contains a split according to $X_j > s$, then we may not be able to assign the sample to a region.

Missing data

Problem: If a sample is missing variable X_j, and a tree contains a split according to $X_j > s$, then we may not be able to assign the sample to a region.

Solution:

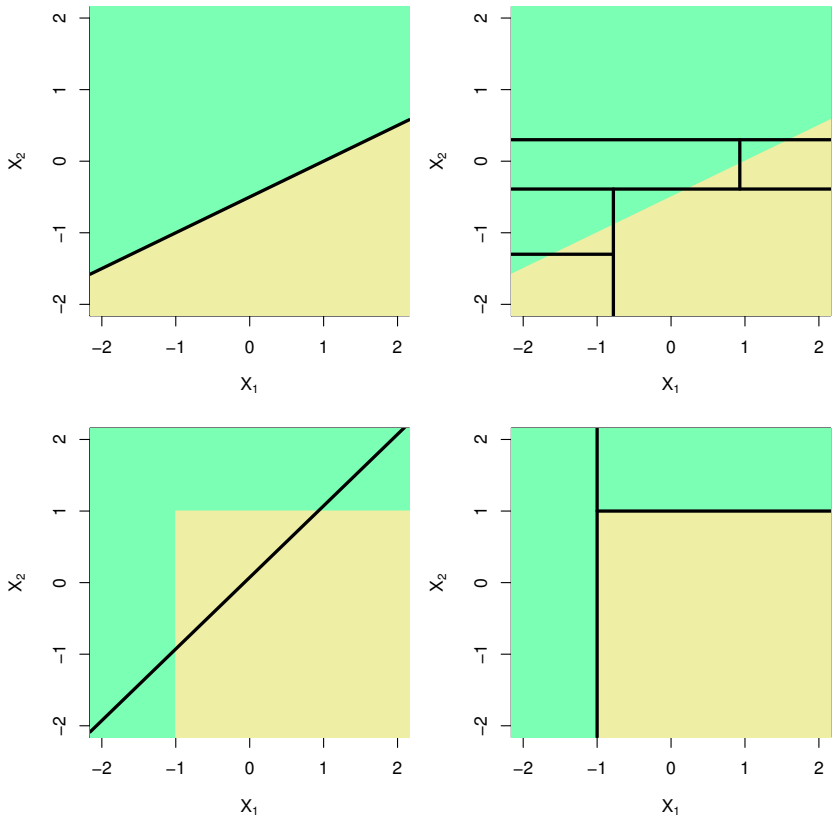
- When choosing a new split with variable X_j (growing the tree):
  - Only consider the samples which have the variable X_j.
  - In addition to choosing the best split, choose a second best split using a different variable, and a third best, ...
- To propagate a sample down the tree, if it is missing a variable to make a decision, try the second best decision, or the third best, etc...

Decision tree advantages

- Very easy to interpret!
- Closer to human decision-making.
- Can capture complex interactions between variables.
- Easy to visualize graphically.
- Easily handle qualitative predictors and missing data.

Decision tree disadvantages

- Doesn't capture simple (e.g. linear) relationships well.
- Less accurate than other ML methods.
- Can have high variance.



Using linear combinations

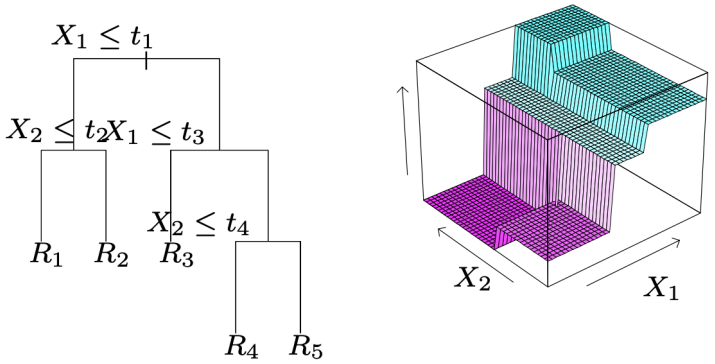
Recall: Our decision boundaries are fixed, i.e. $\mathbf{X}_j \leq s$ **An idea**: Make it linear, i.e. $\sum_j a_j \mathbf{X}_j \leq x$

Some software packages implement these (e.g. oblique random forests)

- More computationally expensive, and might not provide enough value

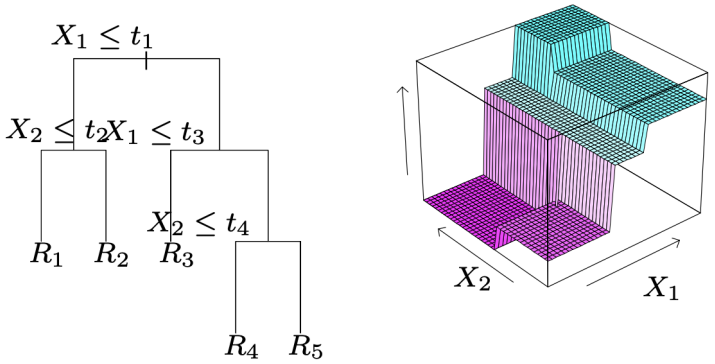
Generalizing CART

Recall: Splitting our regions gives us a corresponding tree



Generalizing CART

Recall: Splitting our regions gives us a corresponding tree



Questions:

- Do we need constant predictions in each leaf node?
- Do we need "hard" splits at each tree node?

Hieracrhical Mixtures of Experts

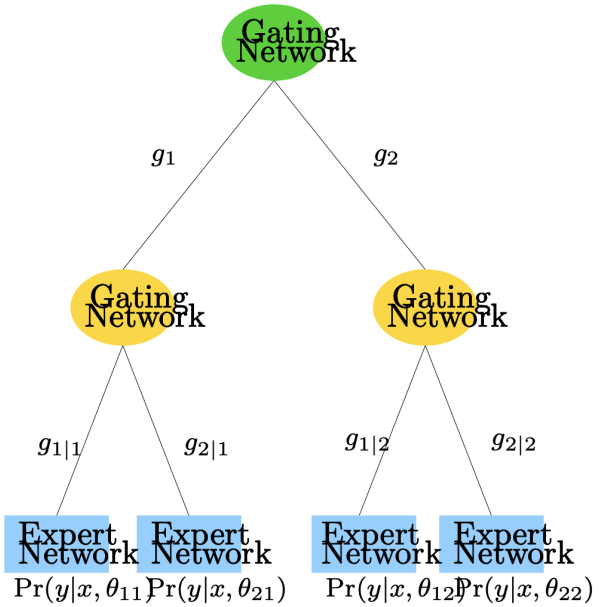


FIGURE 9.13. *A two-level hierarchical mixture of experts (HME) model.*

Define each gate with a logistic model, i.e.

$$g_j(\mathbf{X}, \gamma_j) = \frac{\exp(\gamma_j^\top \mathbf{X})}{\sum_{k=1}^{K} \exp(\gamma_k^\top \mathbf{X})} \tag{3}$$

Hieracrhical Mixtures of Experts

Define each leaf node with a corresponding model, e.g.

Regression

$$Y = \beta_R^\top \mathbf{X} + \epsilon : \epsilon \sim \mathcal{N}(0, \sigma_R^2) \tag{4}$$

Classification

$$P(Y = 1|\mathbf{X})_R = \frac{1}{1 + \exp(-\beta_R^T \mathbf{X})} \tag{5}$$

n.b. Parameters are estimated via MLE using EM

Bagging

Recall:

- Bagging = Bootstrap Aggregating
- We replicate our dataset by sampling with replacement:
  - Original dataset: $\texttt{x} = \texttt{c}(\texttt{x1}, \texttt{x2}, \ldots, \texttt{x100})$
  - Bootstrap samples:
    $\texttt{boot1} = \texttt{sample}(\texttt{x}, 100, \texttt{replace} = \texttt{True})$, ...,
    $\texttt{bootB} = \texttt{sample}(\texttt{x}, 100, \texttt{replace} = \texttt{True})$.
- We average the predictions of a model fit to many Bootstrap samples:

$$\hat{f}_n^{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^b(x).$$

When does Bagging make sense?

When a regression method or a classifier has a tendency to overfit, Bagging reduces the variance of the prediction.

- When n is large, the empirical distribution is similar to the true distribution of the samples.
- Bootstrap samples are like independent realizations of the data.
- Bagging amounts to averaging the fits from many independent datasets, which would reduce the variance by a factor $1/B$, i.e. $\frac{1}{B}\sigma^2$.

Bagging decision trees

- **Disadvantage:** Every time we fit a decision tree to a bootstrap sample, we get a different tree T^b.

  $\rightarrow$ Loss of interpretability

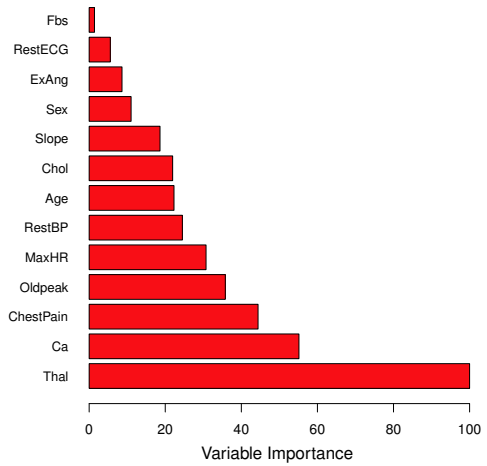
Bagging decision trees

- **Disadvantage:** Every time we fit a decision tree to a bootstrap sample, we get a different tree T^b.

  $\rightarrow$ Loss of interpretability

Variable importance:

- For each predictor, add up the total by which the RSS (or Gini index) decreases every time we use the predictor in T^b.
- Average this total over each Boostrap estimate $T^1, \ldots, T^B$.



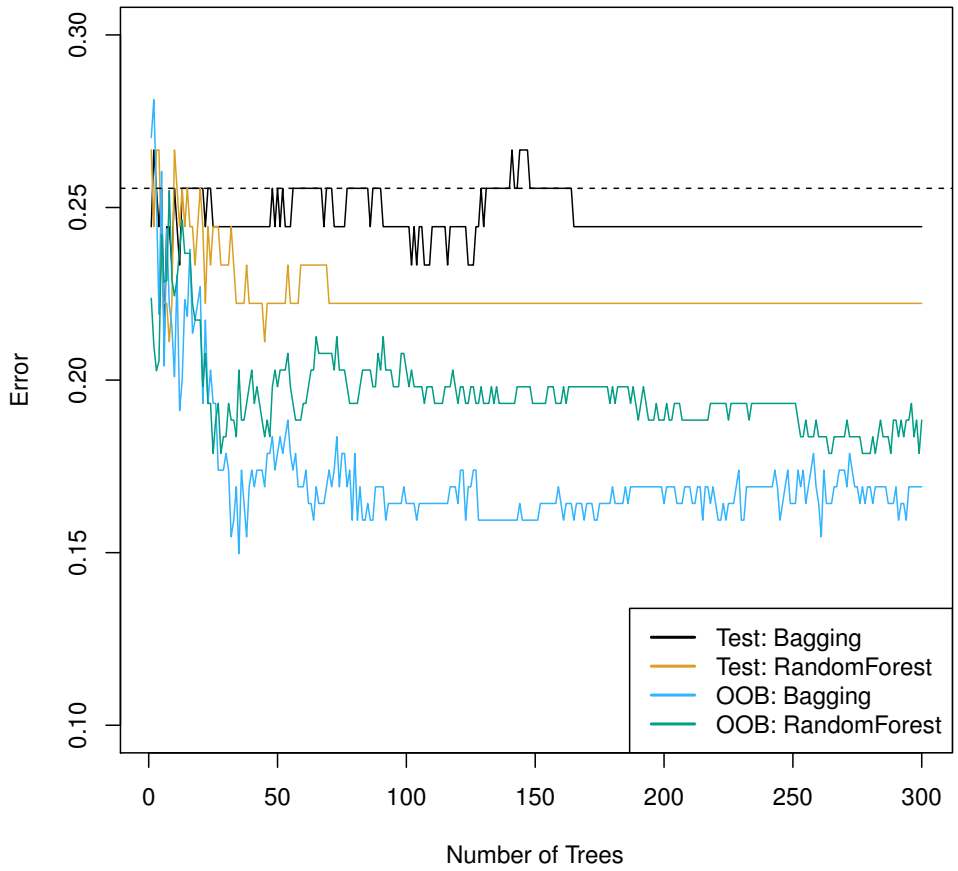
Out-of-bag (OOB) error

To estimate the test error of a bagging estimate, we *could* use cross-validation.

Or instead, could just use observations that weren't sampled

- Each time we draw a Bootstrap sample, we only use 63% of the observations.
- **Idea:** use the rest of the observations as a test set.
- **OOB error:**
  - For each sample x_i, find the prediction $\hat{y}_i^b$ for all bootstrap samples b which do not contain x_i. There should be around $0.37B$ of them. Average these predictions to obtain $\hat{y}_i^{\text{oob}}$.
  - Compute the error $(y_i - \hat{y}_i^{\text{oob}})^2$.
  - Average the errors over all observations $i = 1, \ldots, n$.

Out-of-bag (OOB) error



The test error decreases as we increase B
(dashed line is the error for a plain decision tree).

In general, bagging has a problem:

$\rightarrow$ The trees produced by different Bootstrap samples can be very similar.

Specifically: The variance from bagging is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Lowering ρ can lower our variance

Random Forests:

- We fit a decision tree to different Bootstrap samples.
- When growing the tree, we select a random sample of $m < p$ predictors to consider in each step.
- This will lead to very different (or "uncorrelated") trees from each sample.
- Finally, average the prediction of each tree.

Random Forests algorithm

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

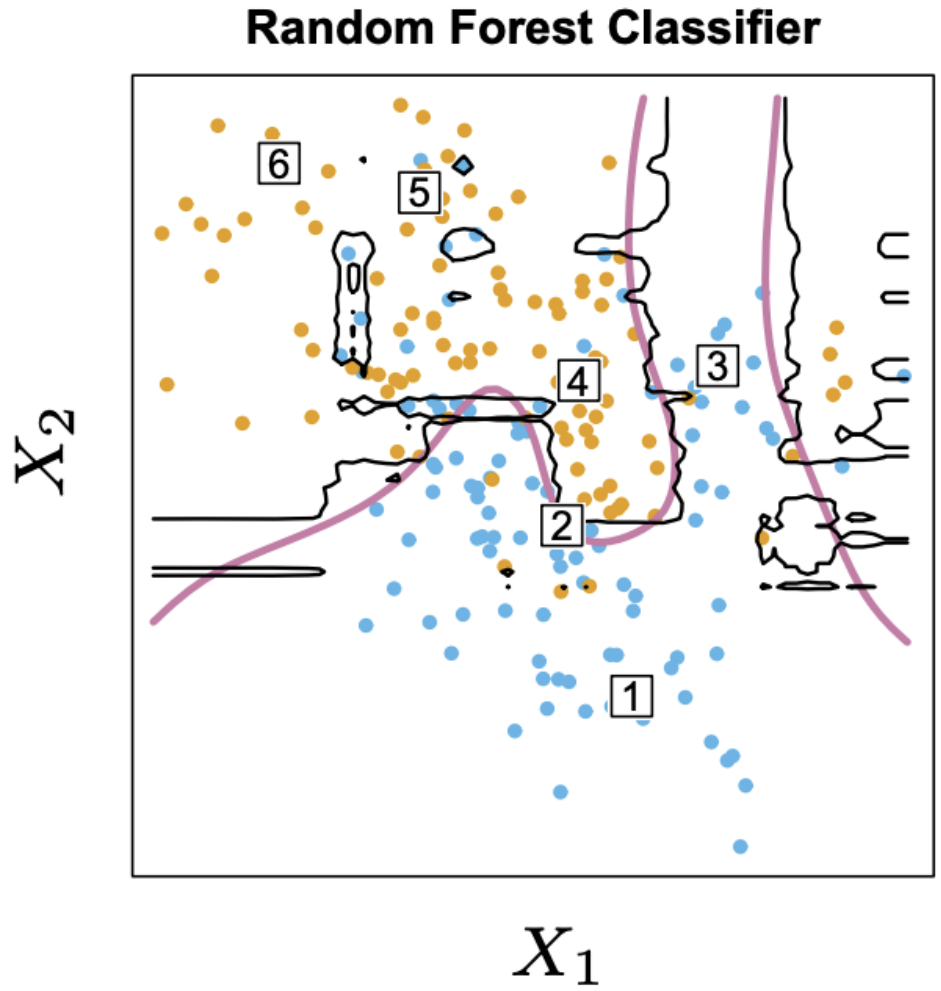
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

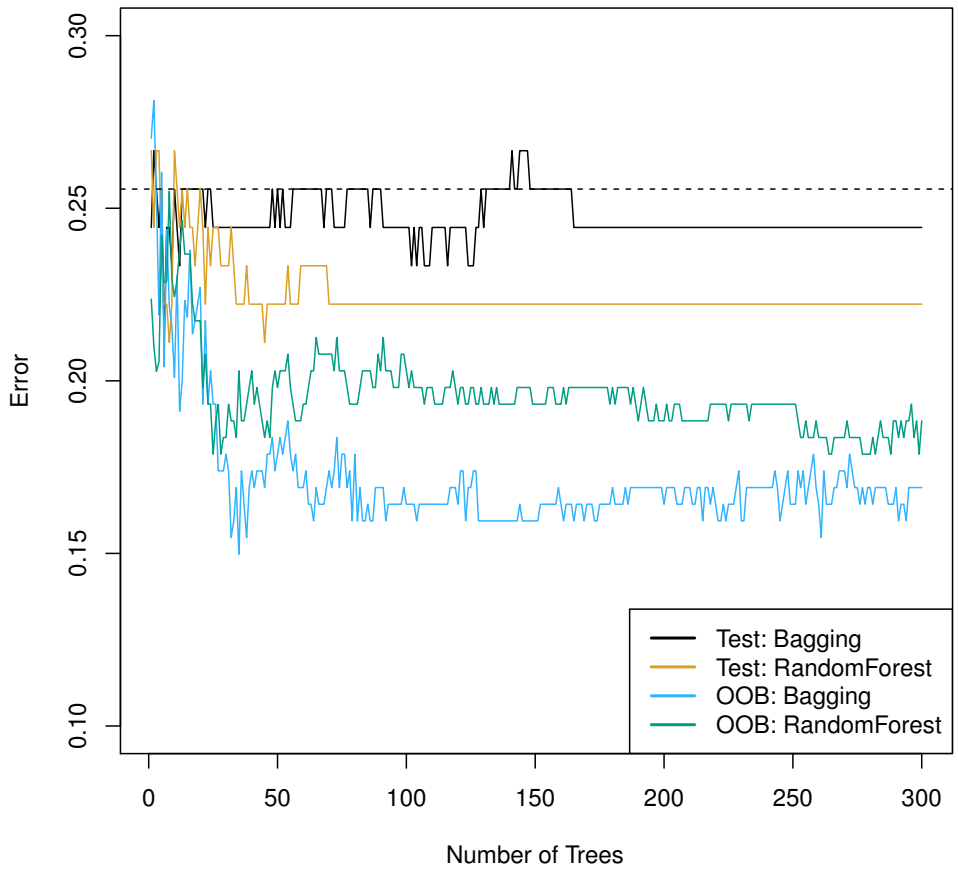
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random Forests example

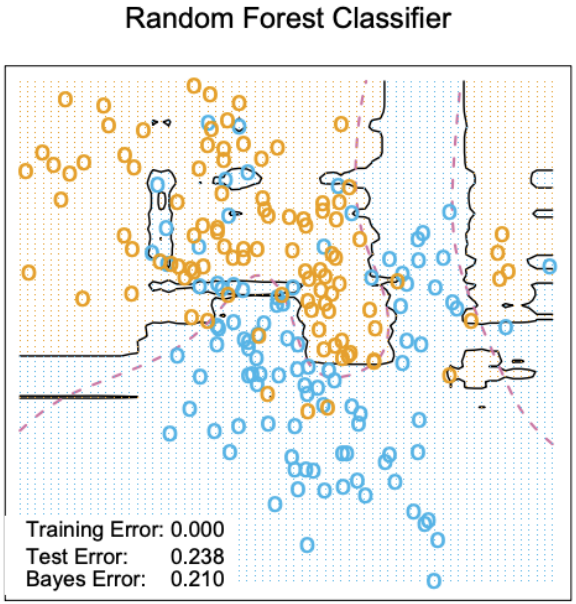
Applied to simulated data with known ground truth

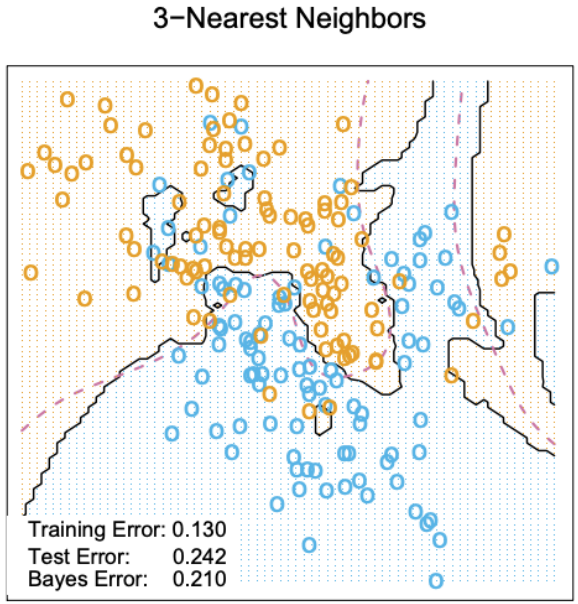


Random Forests vs. Bagging



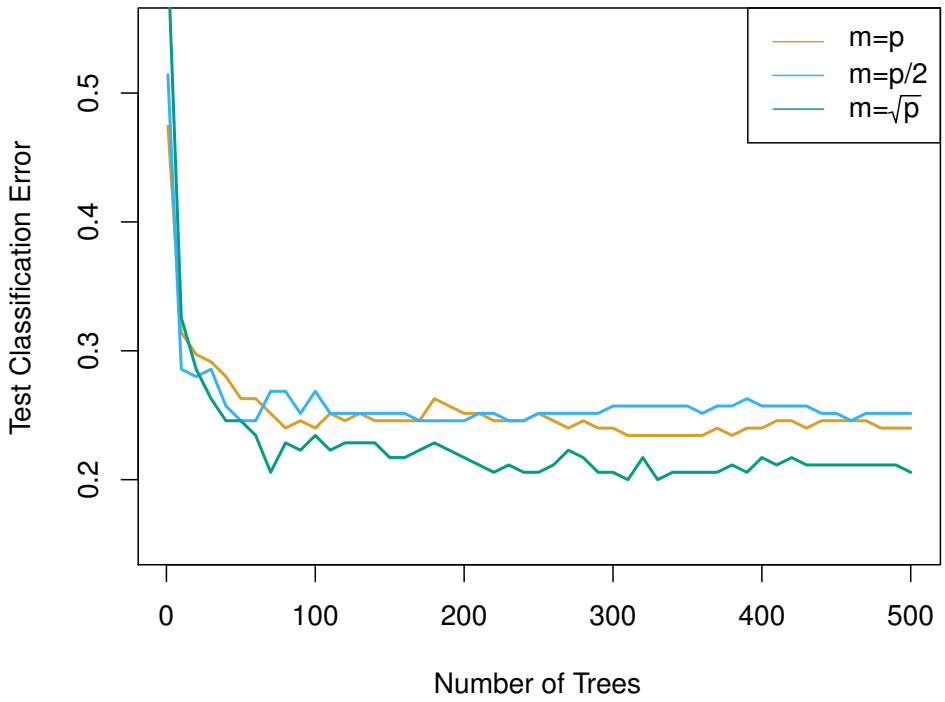
Random Forests vs. KNN





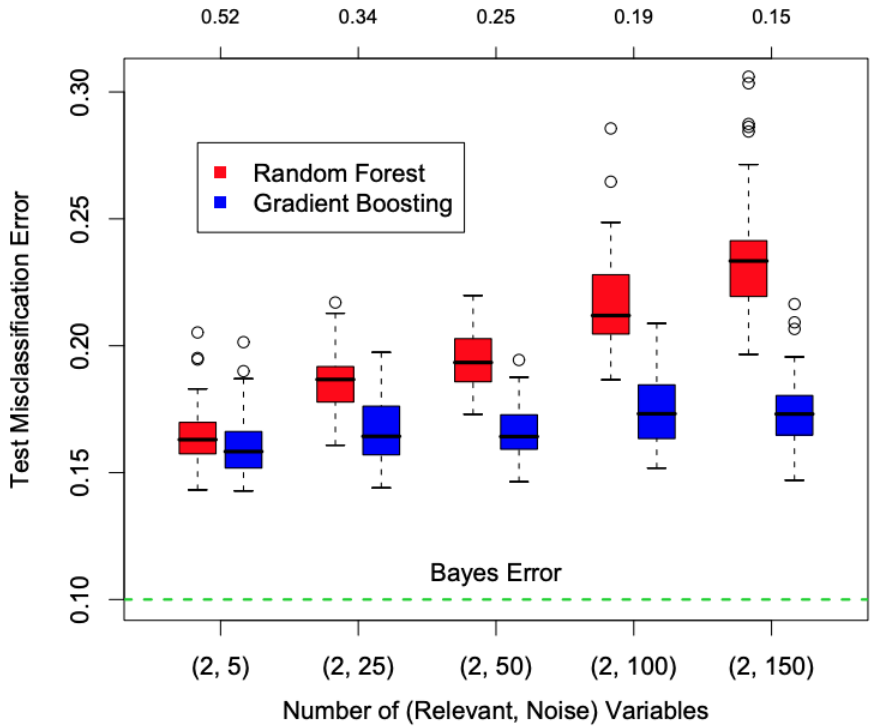
Random forest can be thought of as weighted voting of the closest points.

Random Forests, choosing m



The optimal m is usually around $\sqrt{p}$,
but this can be used as a tuning parameter.

Overfitting with Random Forests



Yes, we can overfit using Random Forests!

References

[1] ISL. Chapter 8

[2] ESL. Chapter 9.2,15