Introduction to Statistical Learning and Machine Learning

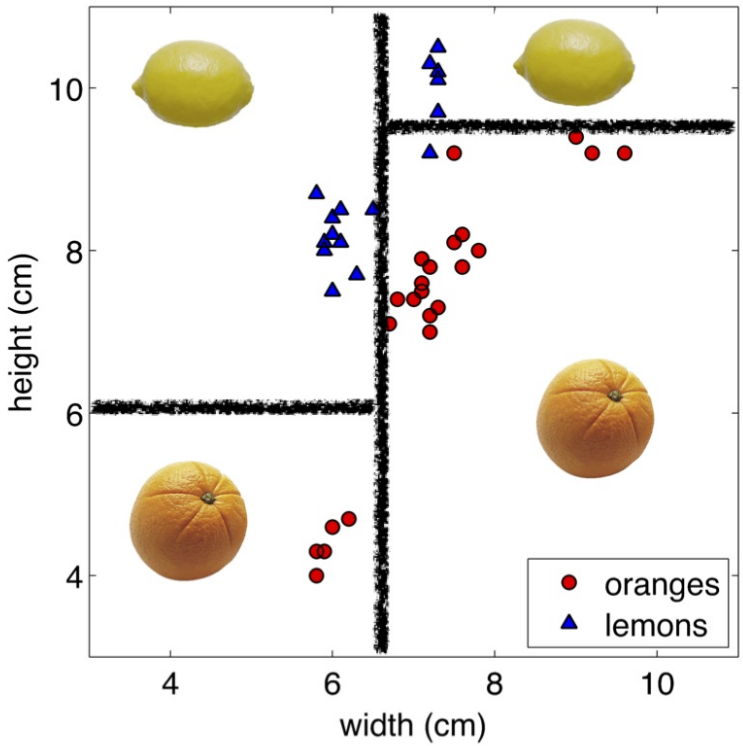
Chap 12 – Introduction to Tree-based methods

Yanwei Fu

School of Data Science, Fudan University

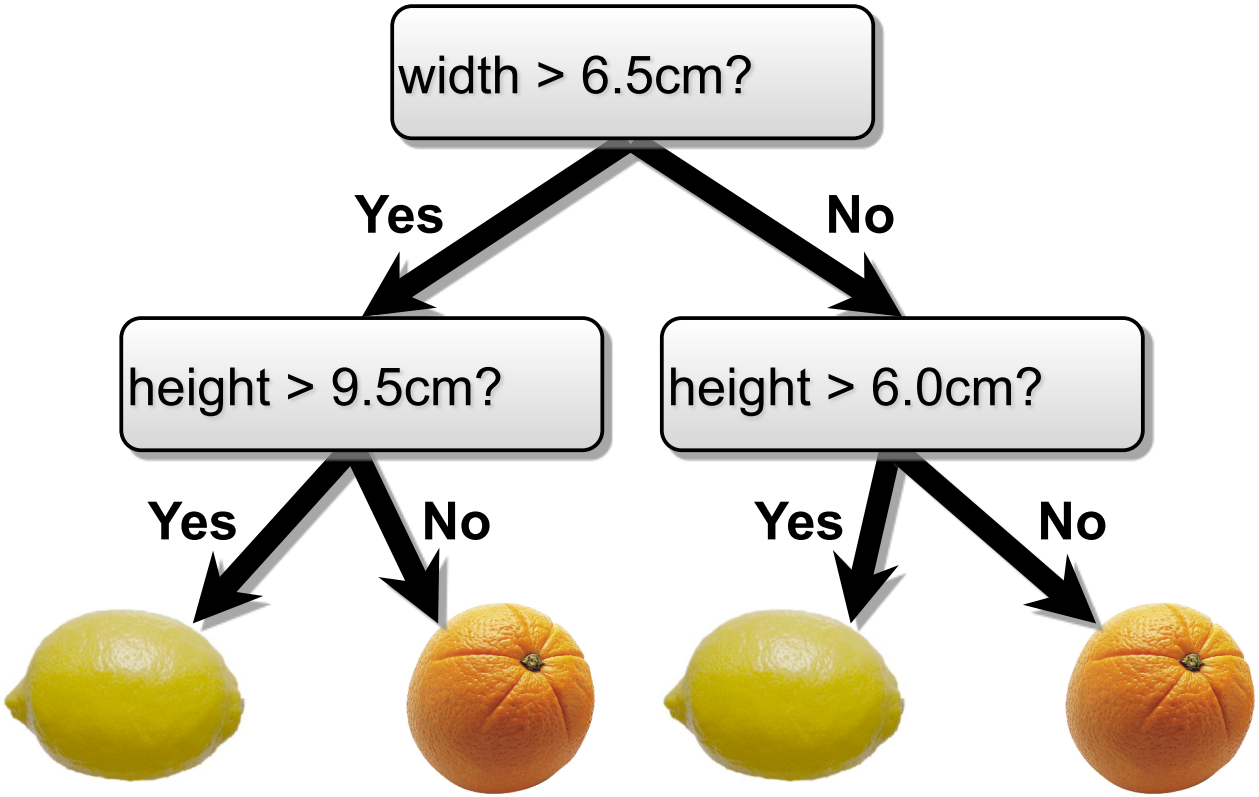
Another Classification Idea

- We could view the decision boundary as being the composition of several simple boundaries.

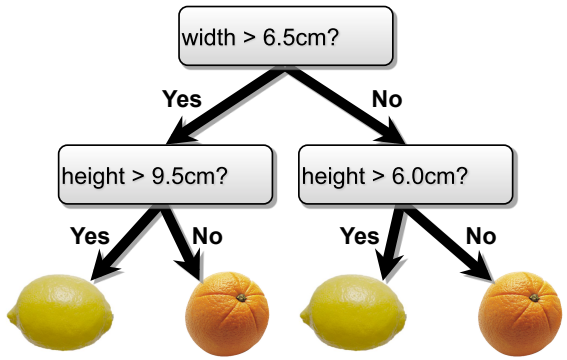


- Decision trees can be applied to both regression and classification problems.
- Pros and Cons:
  - Tree-based methods are simple and useful for interpretation.
  - However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
  - Hence the method of bagging, random forests, and boosting grow multiple trees which are then combined to yield a single consensus prediction.
  - Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

Decision Tree: Example



Decision Tree: Example



- Internal nodes test attributes
- Branching is determined by attribute value
- Leaf nodes are outputs (class assignments)
- In general, a decision tree can represent any binary function

Decision Tree: Algorithm

- Choose an attribute on which to descend at each level.
- Condition on earlier (higher) choices.
- Generally, restrict only one dimension at a time.
- Declare an output value when you get to the bottom
- In the orange/lemon example, we only split each dimension once, but that is not required.
- How do you construct a useful decision tree?
- We use information theory to guide us

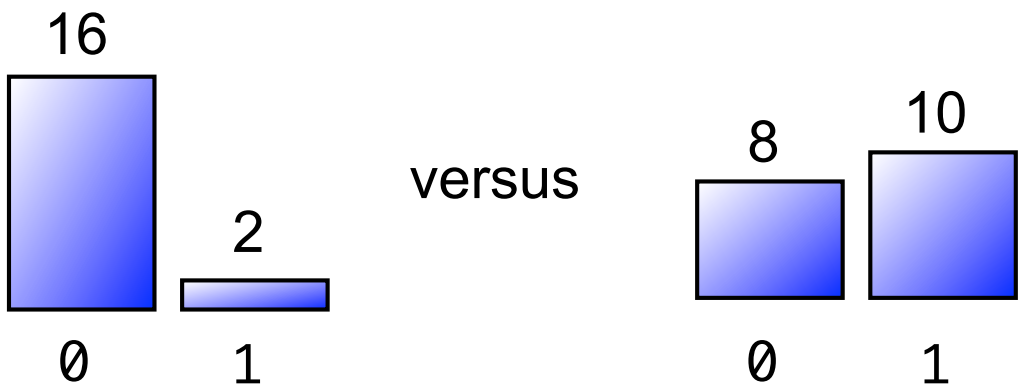
Two Binary Sequences

Sequence 1:

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ... ?

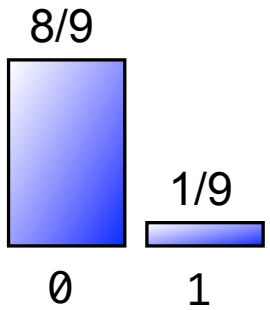
Sequence 2:

0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?

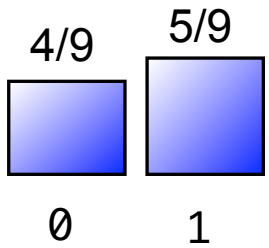


Quantifying Uncertainty: Shannon Entropy

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$



$$\frac{8}{9} \log_2 \frac{8}{9} - \frac{1}{9} \log_2 \frac{1}{9} \approx \frac{1}{2}$$



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

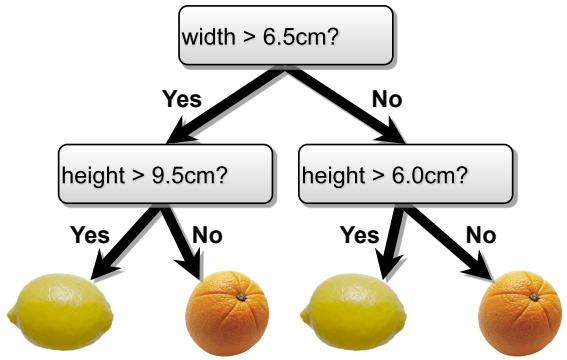
- How surprised are we by a new value in the sequence?
- How much information does it convey?

Quantifying Uncertainty: Shannon Entropy

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

- Shannon Entropy is an extremely powerful concept.
- It tells you how much you can compress your data!

Decision Tree: Algorithm



- Choose an attribute on which to descend at each level.
- Condition on earlier (higher) choices
- Generally, restrict only one dimension at a time.
- How do you construct a useful decision tree?
- We use information theory to guide us

Entropy of a Joint Distribution

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

$$
\begin{aligned}
H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x,y) \\
&= -\frac{24}{100}\log_2\frac{24}{100} - \frac{1}{100}\log_2\frac{1}{100} - \frac{25}{100}\log_2\frac{25}{100} - \frac{50}{100}\log_2\frac{50}{100} \\
&\approx 1.56\text{bits}
\end{aligned}
$$

Specific Conditional Entropy

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

- What is the entropy of cloudiness, given that it is raining?

$$\begin{aligned} H(X|Y=y) &= \sum_{x\in X} p(x|y)\log_2 p(x|y) \\ &= -\frac{24}{25}\log_2\frac{24}{25} - \frac{1}{25}\log_2\frac{1}{25} \\ &\approx 0.24\text{bits} \end{aligned}$$

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

- The expected conditional entropy:

$$\begin{aligned} H(X|Y) &= \sum_{y \in Y} p(y) H(X|Y=y) \\ &= -\sum_{y \in Y} \sum_{x \in X} p(x,y) \log_2 p(x|y) \end{aligned}$$

Specific Conditional Entropy

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

- What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

$$\begin{aligned} H(X|Y) &= \sum_{y \in Y} p(y) H(X|Y=y) \\ &= \frac{1}{4} H(\text{clouds}|\text{is raining})) + \frac{3}{4} H(\text{clouds}|\text{not raining}) \\ &\approx 0.75 \text{ bits} \end{aligned}$$

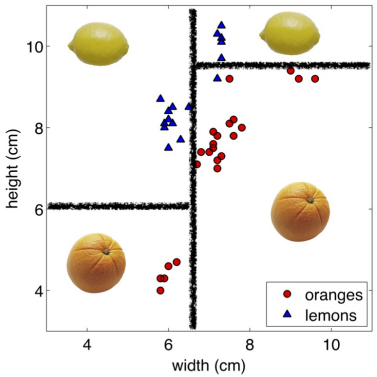
	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

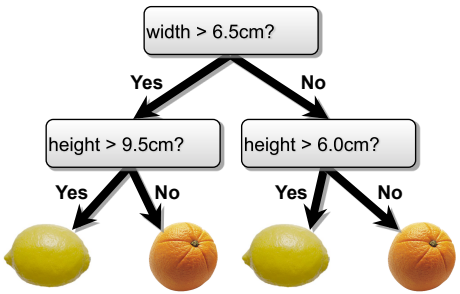
- How much information about cloudiness do we get by discovering whether it is raining?

$$\begin{aligned} IG(X|Y) &= H(X) - H(X|Y) \\ &\approx 0.25 \text{ bits} \end{aligned}$$

- Also called information gain in X due to Y
- For decision trees, X is the class/label and Y is an attribute

Constructing Decision Trees





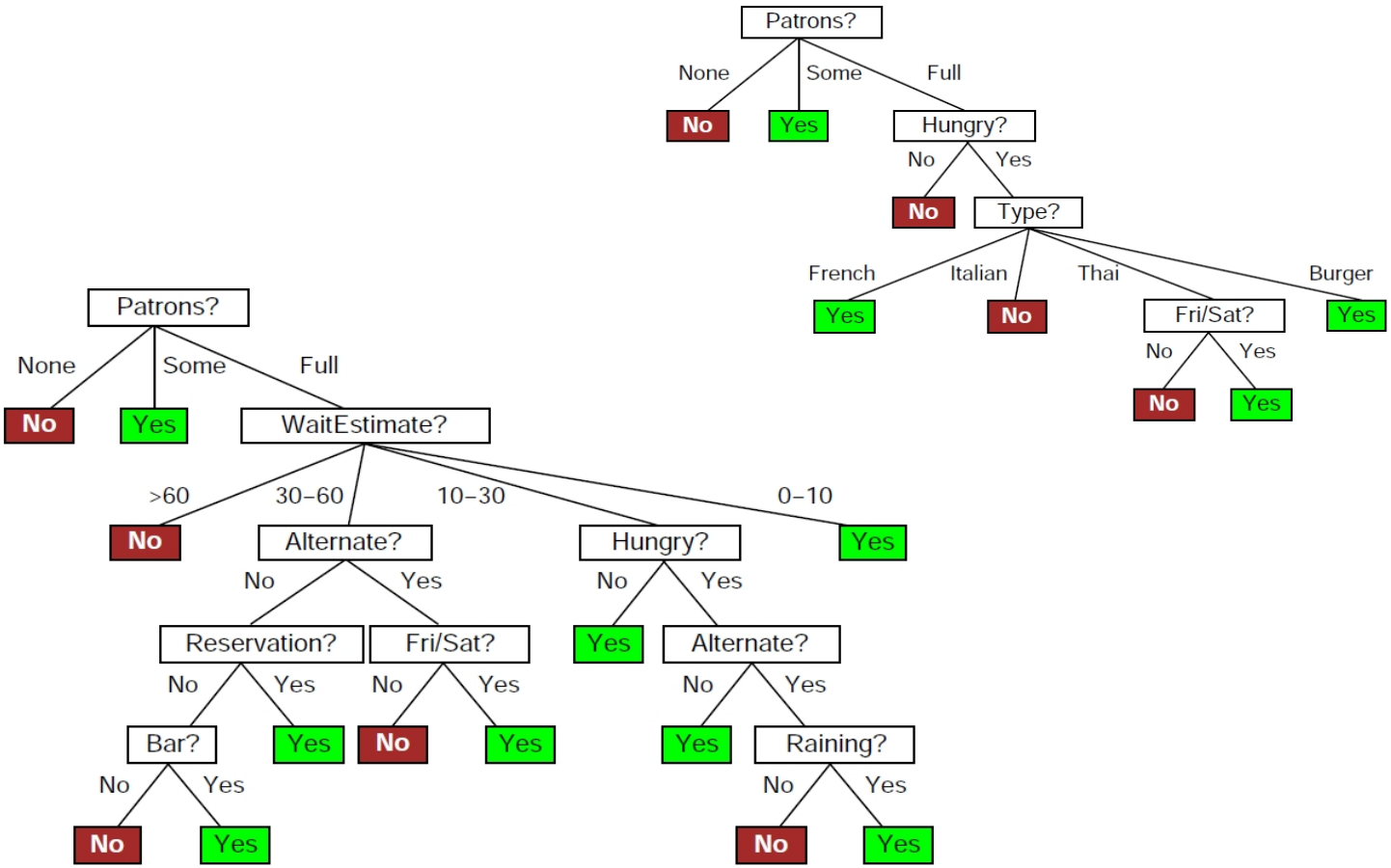
- I made the fruit data partitioning just by eyeballing it.
- We can use the mutual information to automate the process.
- At each level, one must choose:
  1. Which variable to split.
  2. Possibly where to split it.
- Choose them based on how much information we would gain from the decision!

Decision Tree Algorithm

- Simple, greedy, recursive approach, builds up tree node-by-node

1. pick an attribute to split at a non-terminal node
2. split examples into groups based on attribute value
3. for each group:
   - if no examples – return majority from parent
   - else if all examples in same class – return class
   - else loop to step 1

Which Tree is Better?



What Makes a Good Tree?

- Not too small: need to handle important but possibly subtle distinctions in data
- Not too big:
  - Computational efficiency (avoid redundant, spurious attributes)
  - Avoid over-fitting training examples
- Occam's Razor: find the simplest hypothesis (smallest tree) that fits the observations
- Inductive bias: small trees with informative nodes near the root

- Problems:
  - You have exponentially less data at lower levels.
  - Too big of a tree can overfit the data.
  - Greedy algorithms don't necessarily yield the global optimum.
- In practice, one often regularizes the construction process to try to get small but highly-informative trees.
- Decision trees can also be used for regression on real-valued outputs, but it requires a different formalism.

Comparison to k-NN

K-Nearest Neighbors

- Decision boundaries: piece-wise
- Test complexity:
  non-parametric, few parameters besides (all?) training examples

Decision Trees

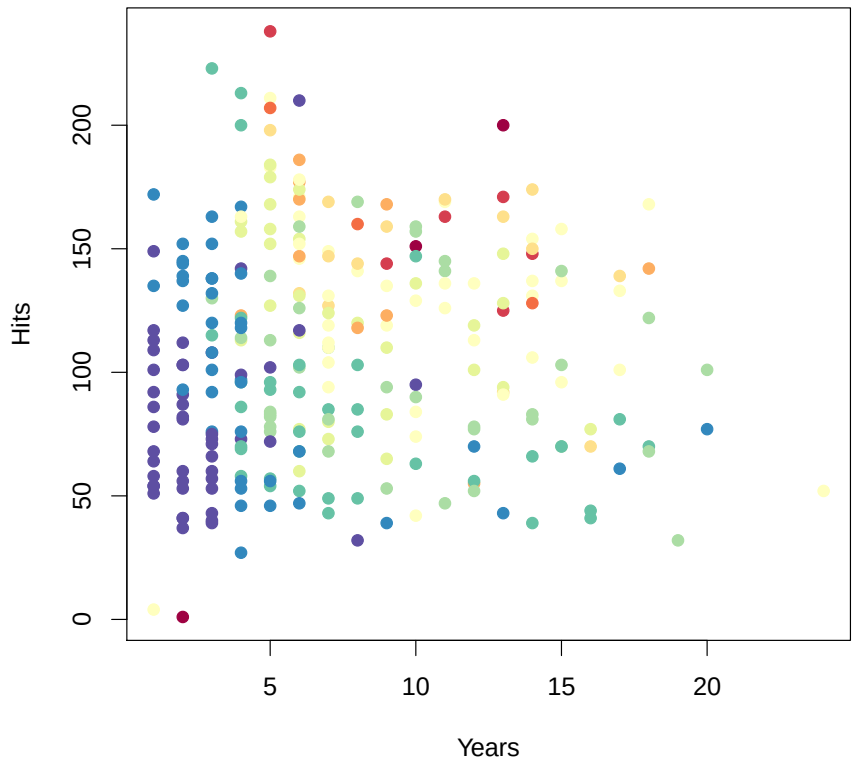
- Decision boundaries:
  axis-aligned, tree structured
- Test complexity: attributes and splits

Applications of Decision Trees

- Can express any Boolean function, but most useful when function depends critically on few attributes
- Bad on: parity, majority functions; also not well-suited to continuous attributes
- Practical Applications:
  - Flight simulator: 20 state variables; 90K examples based on expert pilot's actions; auto-pilot tree
  - Yahoo Ranking Challenge
  - Random Forests

Baseball salary data: how would you stratify it?

Salary is color-coded from low (blue, green) to high (yellow,red)

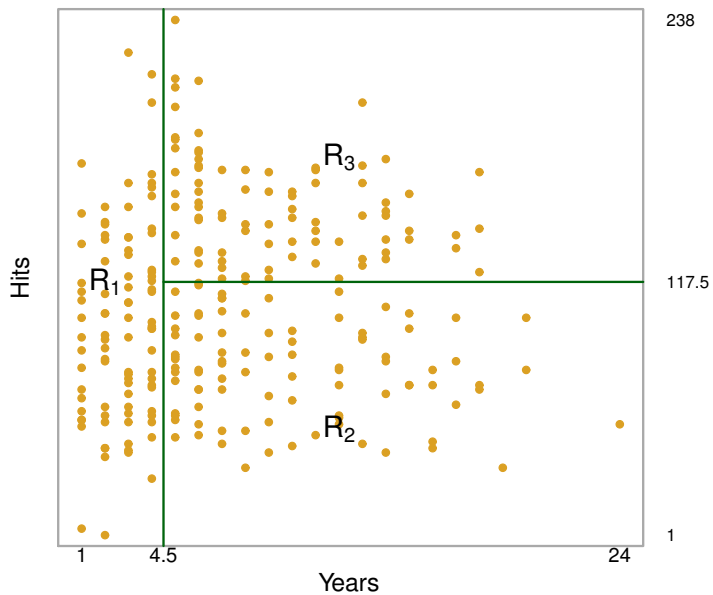


Baseball salary data: how would you stratify it?

Baseball salary data: how would you stratify it?

- For the Hitters data, a regression tree for predicting the 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 that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to `Years`<4.5, and the right-hand branch corresponds to `Years`>=4.5.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{\mathrm{X} \mid \texttt{Years} < 4.5\}$, $R_2 = \{\mathrm{X} \mid \texttt{Years} >= 4.5, \texttt{Hits} < 117.5\}$, and $R_3 = \{\mathrm{X} \mid \texttt{Years} >= 4.5, \texttt{Hits} >= 117.5\}$.



Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text `Years`<4.5 and `Hits`<117.5.

Interpretation of Results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

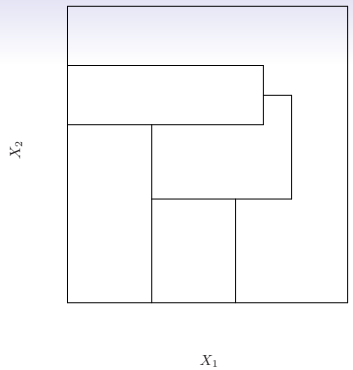
Details of the tree-building process

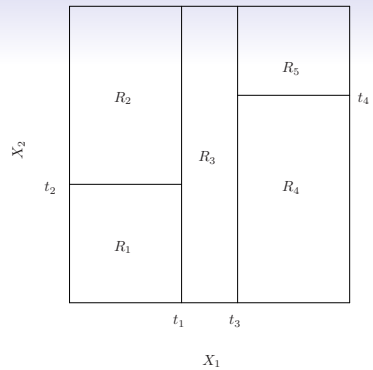
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

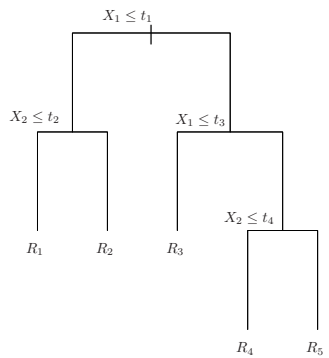
Details of the tree-building process

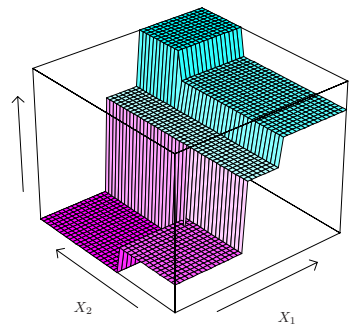
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_2
X_1


$X_1 \le t_1$
$X_2 \le t_2$
$X_1 \le t_3$
$X_2 \le t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

Details of previous gure

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

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

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

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

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted:* a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

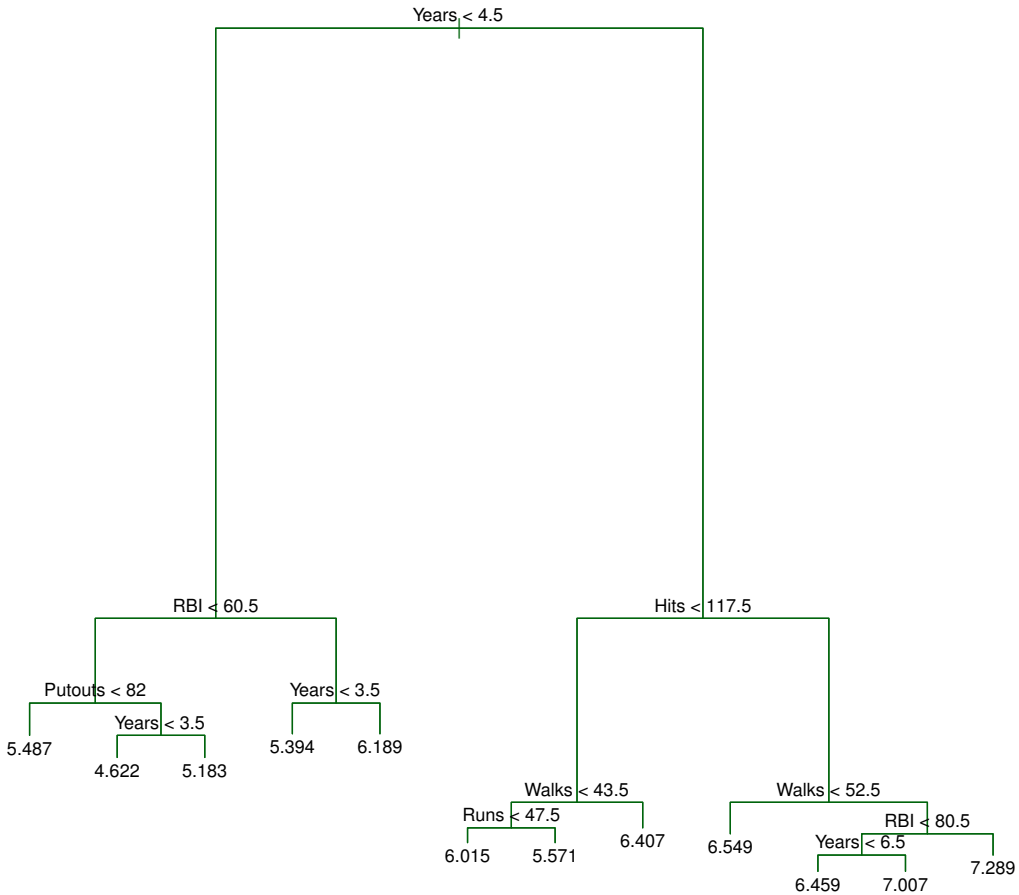
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation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

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

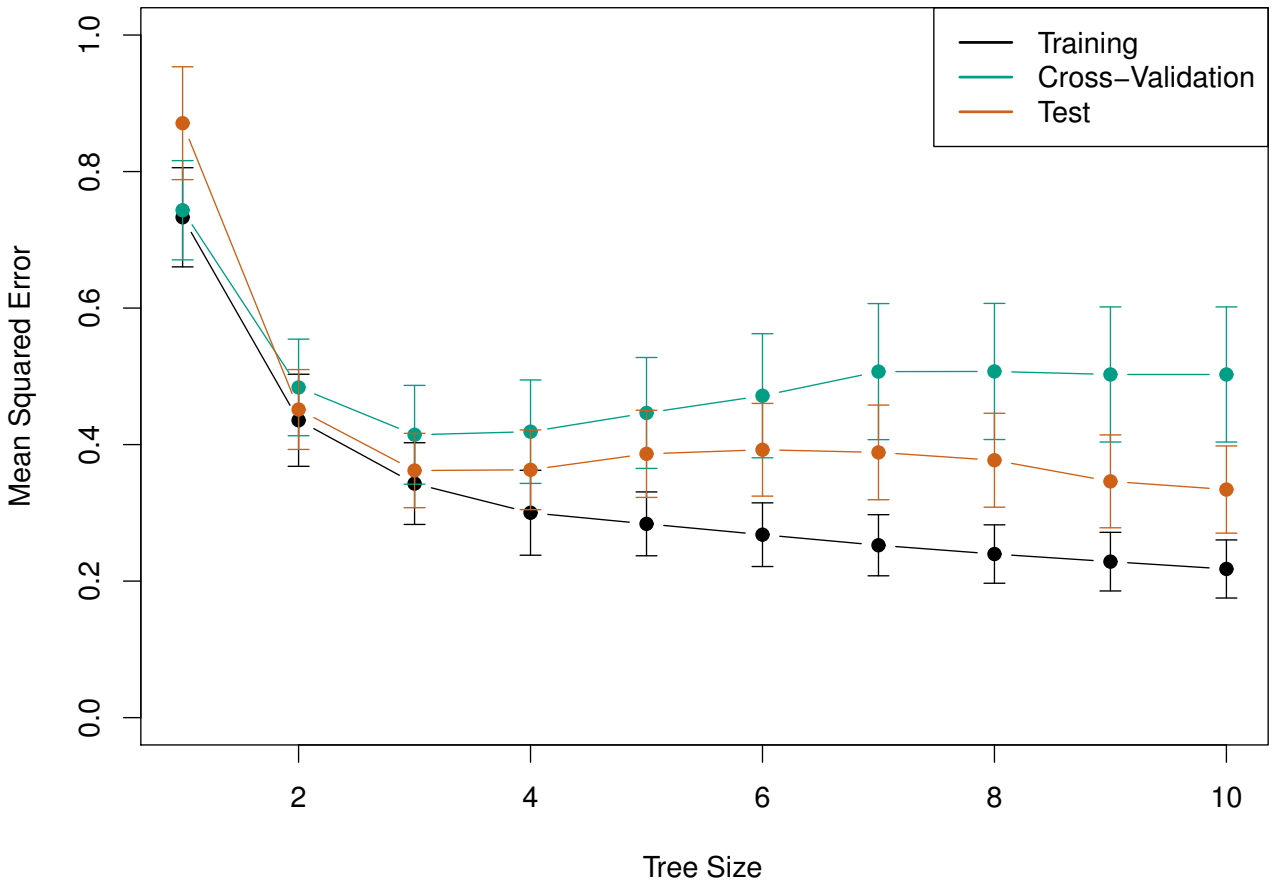
Baseball example continued

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

Baseball example continued



Baseball example continued



- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the *classification error rate*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

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

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

- The *Gini index* is defined by

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

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

- The *Gini index* is defined by

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

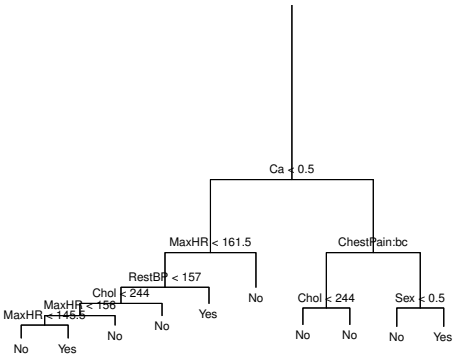
  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from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

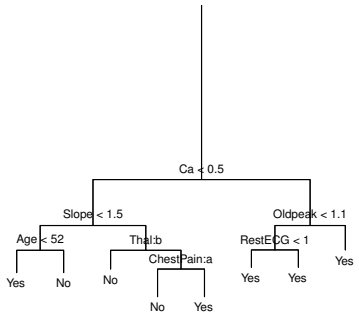
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

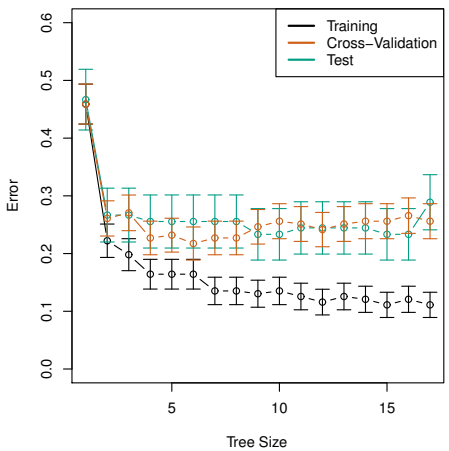
- It turns out that the Gini index and the cross-entropy are very similar numerically.

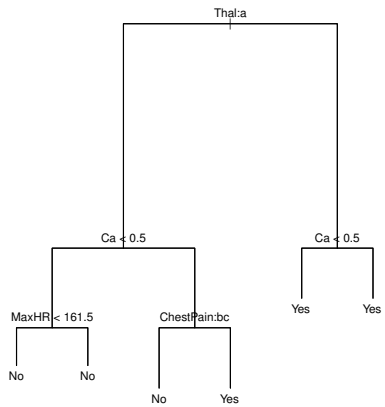
- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement),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

Classication Trees

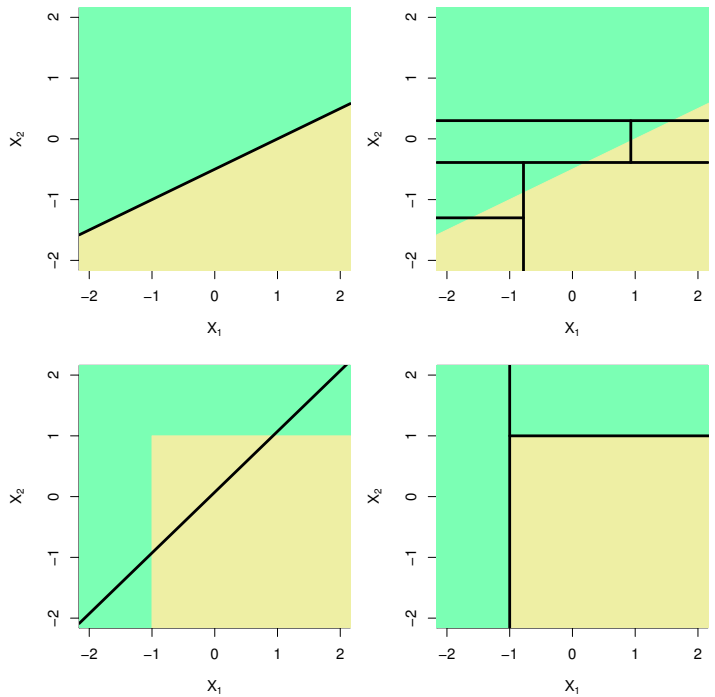








Revist: Advantages and Disadvantages of Trees



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

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

Revist: Advantages and Disadvantages of Trees

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

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

Appendix – Ensemble learning (Bagging, random forests, and boosting).

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

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

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

This is called *bagging*.

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

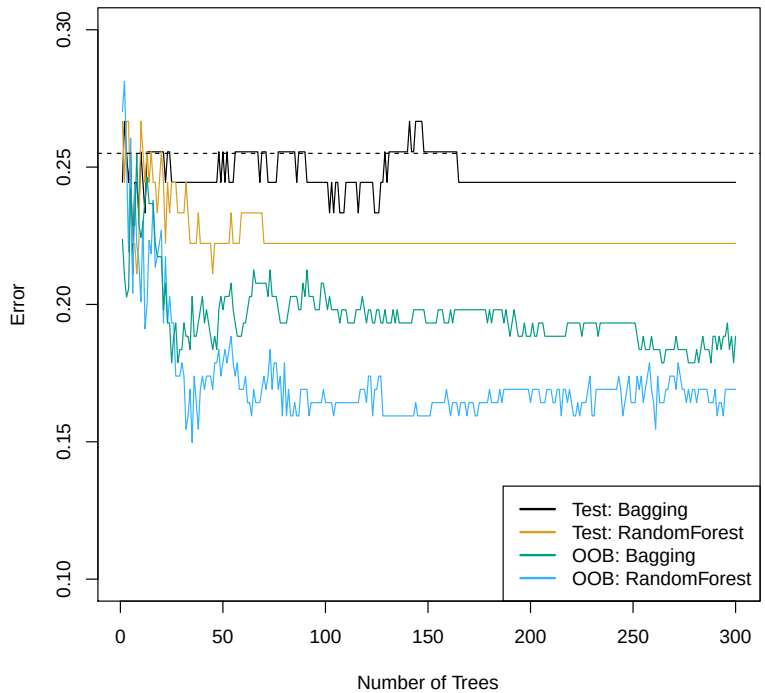
Out-of-Bag Error Estimation

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

Example: heart data

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

Bagging the heart data



Bagging the heart data

Bagging and random forest results for the Heart data.

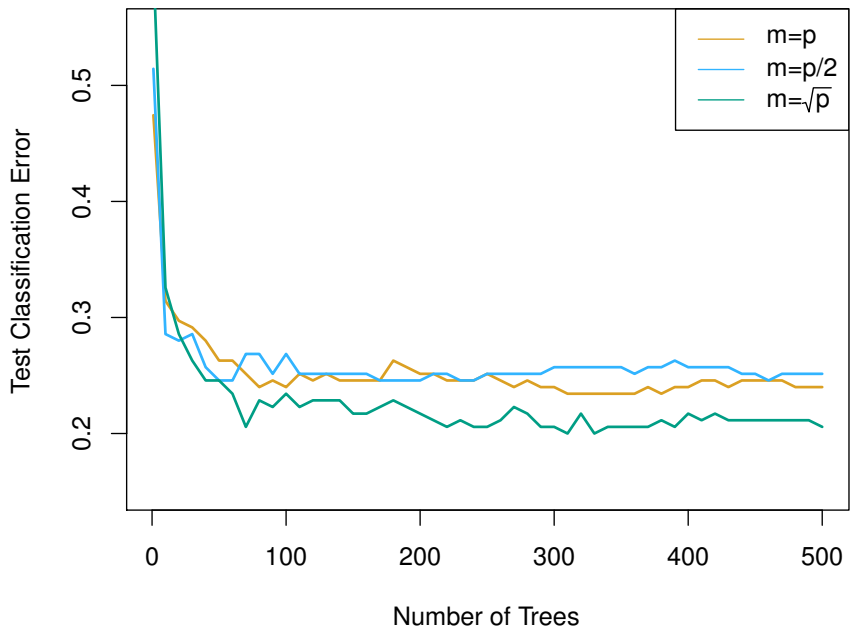
- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- Random forests were applied with $m = \sqrt{p}$.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower

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

Results: gene expression data

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

Results: gene expression data



Results: gene expression data

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

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

Boosting algorithm for regression trees

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

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

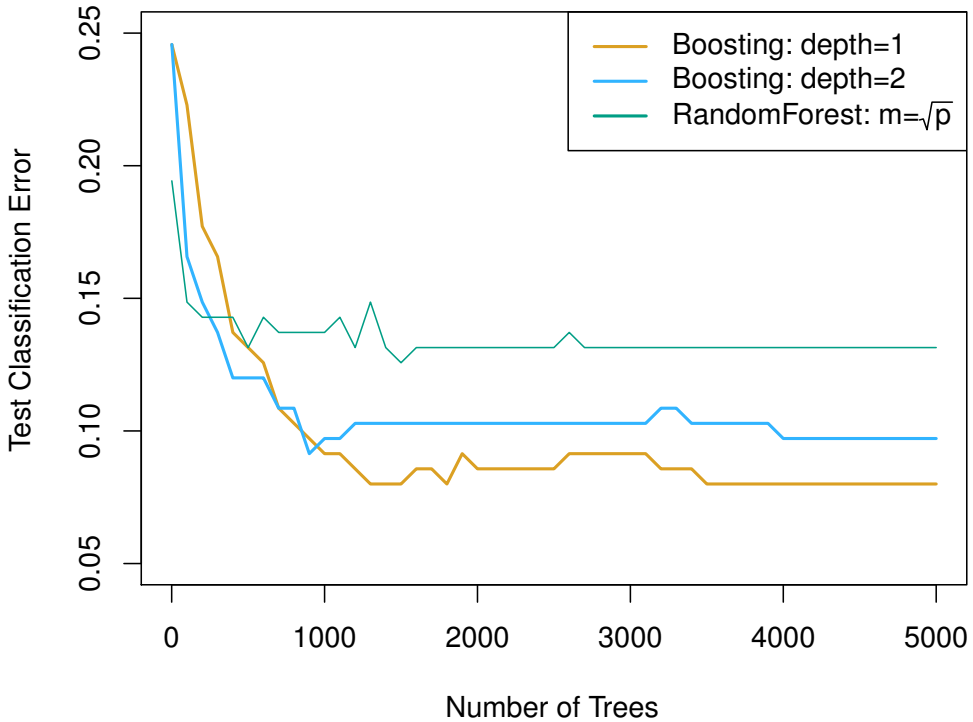
What is the idea behind this procedure?

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

Boosting for classication

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- Students can learn about the details in *Elements of Statistical Learning, chapter 10.*
- The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



Gene expression data continued

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

Tuning parameters for boosting

1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

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.

Tuning parameters for boosting

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

Summary

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

Acknowledgement

Some slides are in courtesy of
(1)Chap 8 of James *et. al.* "An Introduction to Statistical Learning with applications in R", 2011;
(2) Lecture 6, CSC 411 by Raquel Urtasun & Rich Zemel, University of Toronto.