Lecture 5: Classification, Tree-Based Methods

Tree-based methods

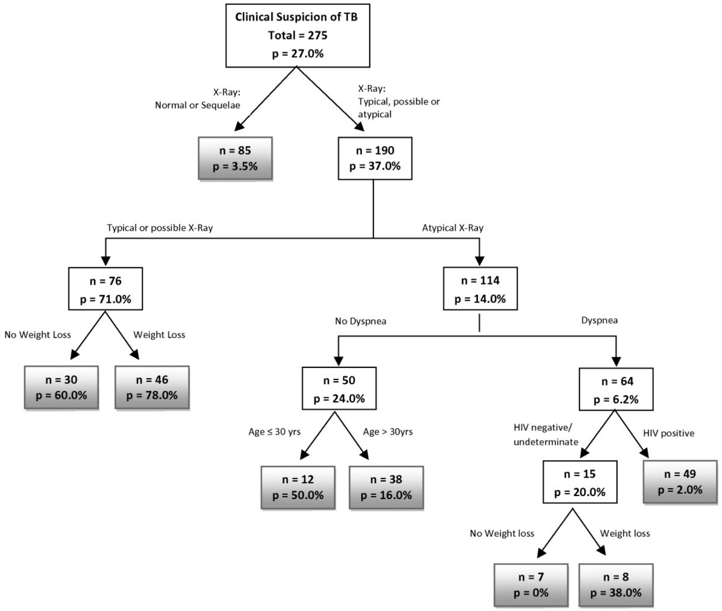
Prof. Alexandra Chouldechova
95-791: Data Mining

Agenda for Week 5

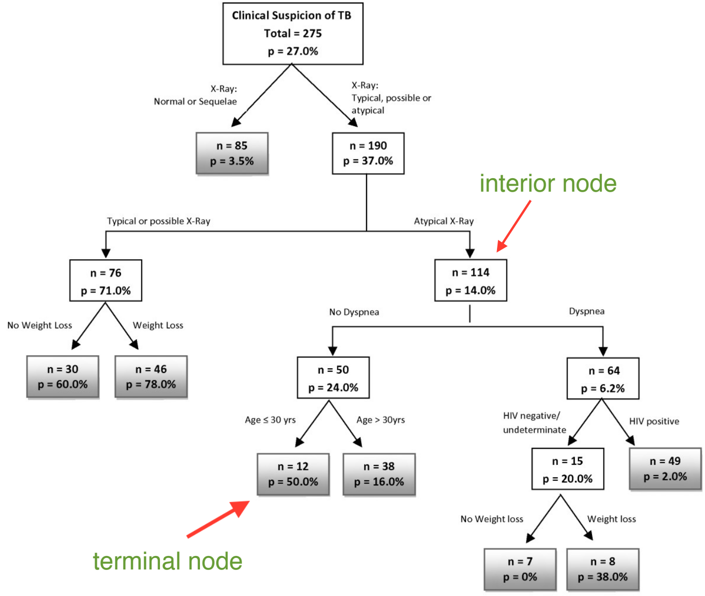
- **Decision Trees (CART)**
- **Bagging**
- **Random Forests**
- **Final project**

Back to methods...

Let's grow some Trees

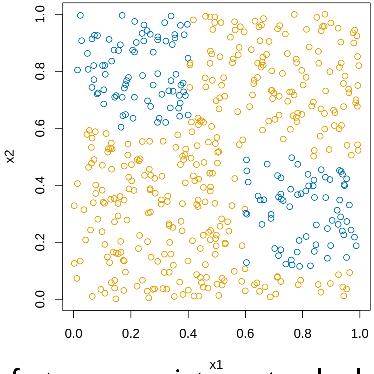


[source: Classification and regression tree (CART) model to predict pulmonary tuberculosis in hospitalized patients, Aguiar et al]

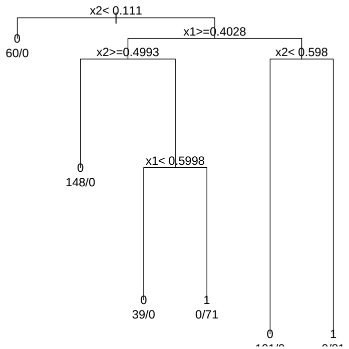

Clinical Suspicion of TB
Total = 275
p = 27.0%
X-Ray: Normal or Sequelae
X-Ray: Typical, possible or atypical
n = 85
p = 3.5%
n = 190
p = 37.0%
Typical or possible X-Ray
Atypical X-Ray
interior node
n = 76
p = 71.0%
n = 114
p = 14.0%
No Weight Loss
Weight Loss
No Dyspnea
Dyspnea
n = 30
p = 60.0%
n = 46
p = 78.0%
n = 50
p = 24.0%
n = 64
p = 6.2%
Age ≤ 30 yrs
Age > 30yrs
HIV negative/ undeterminate
HIV positive
n = 12
p = 50.0%
n = 38
p = 16.0%
n = 15
p = 20.0%
n = 49
p = 2.0%
No Weight loss
Weight loss
terminal node
n = 7
p = 0%
n = 8
p = 38.0%

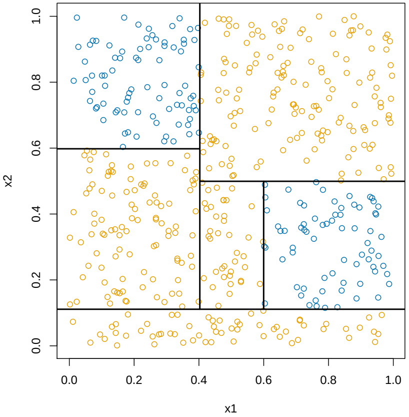
Overview: Tree-based methods

- Tree-based based methods operate by dividing up the feature space into rectangles
- Each rectangle is like a *neighbourhood* in a Nearest-Neighbours method
- You predict using the *average* or classify using the *most common class* in each rectangle



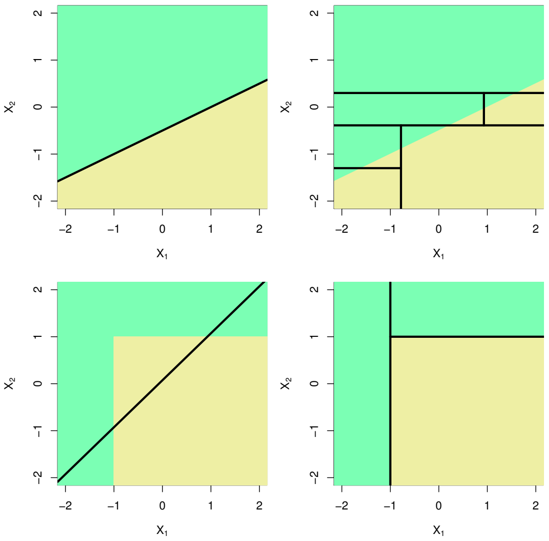
Does dividing up the feature space into rectangles look like it would work here?





- Trees are built up via a greedy algorithm: Recursive binary partitioning
- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data
  - In prediction, you choose splits to minimize the RSS
  - In classification, choose splits to maximize node purity (minimize Gini index)

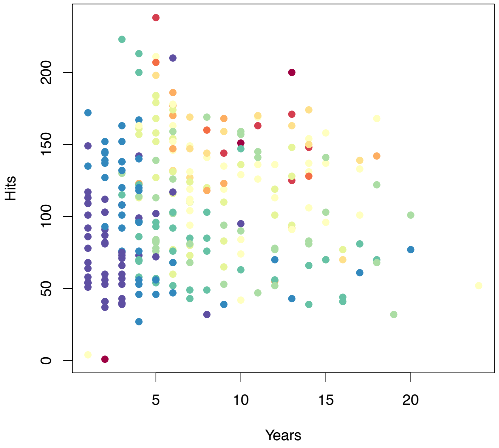
Classification trees vs. Linear models



ISL Figure 8.7. Trees are bad when the boundary is linear, but very good when the boundary is well-described by a simple rectangular partition.

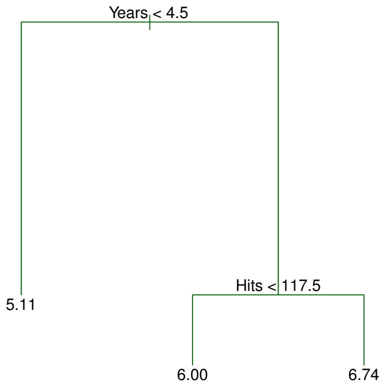
Decision trees in Prediction

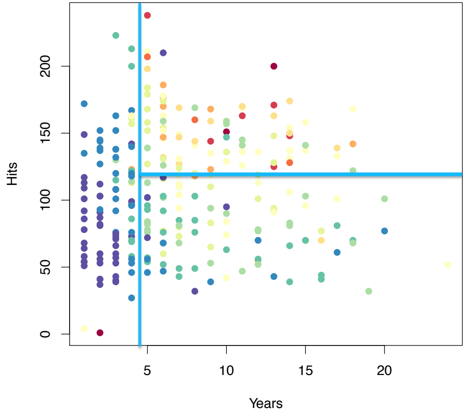
Here's a Prediction example (Y = `Salary` in millions)



Low salary (blue, Green)

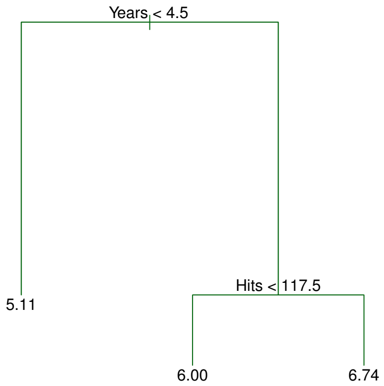
High salary (orange, red)

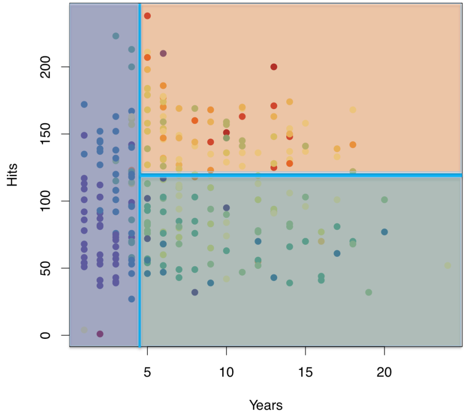




Low salary (blue, Green)
High salary (orange, red)

Within each of the 3 rectangles, we predict `Salary` using the average value of `Salary` in the training data

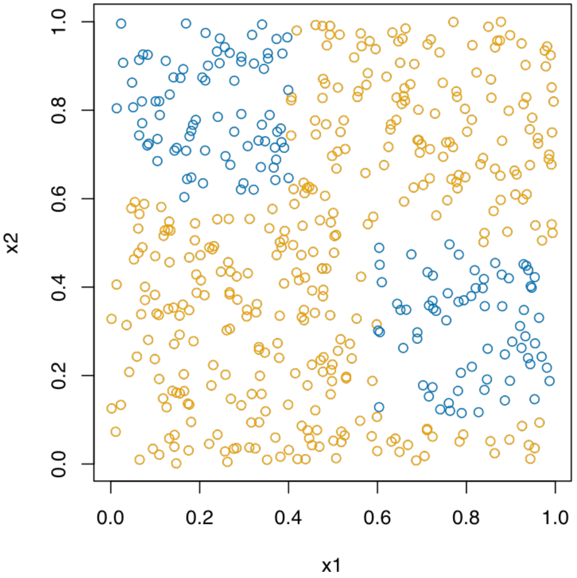




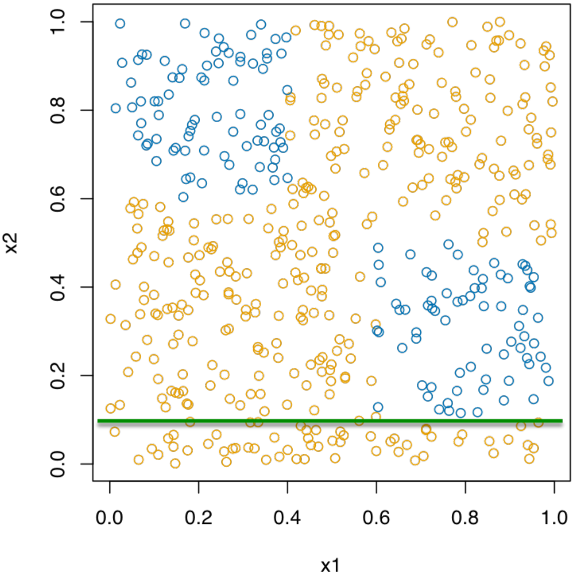
Low salary (blue, Green)
High salary (orange, red)

Within each of the 3 rectangles, we predict `Salary` using the average value of `Salary` in the training data

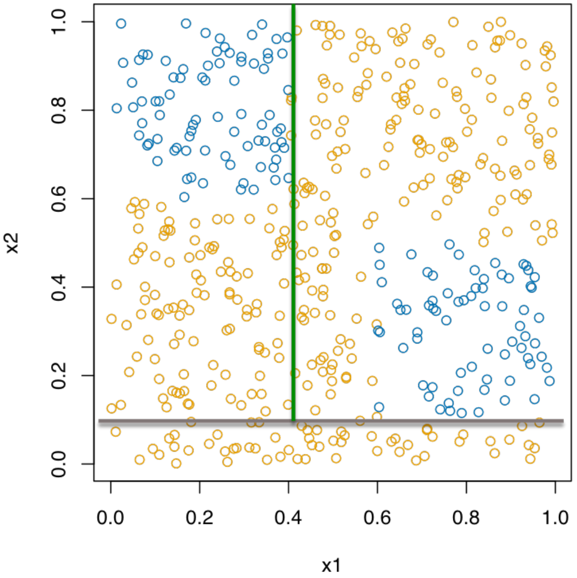
Recursive binary partitioning



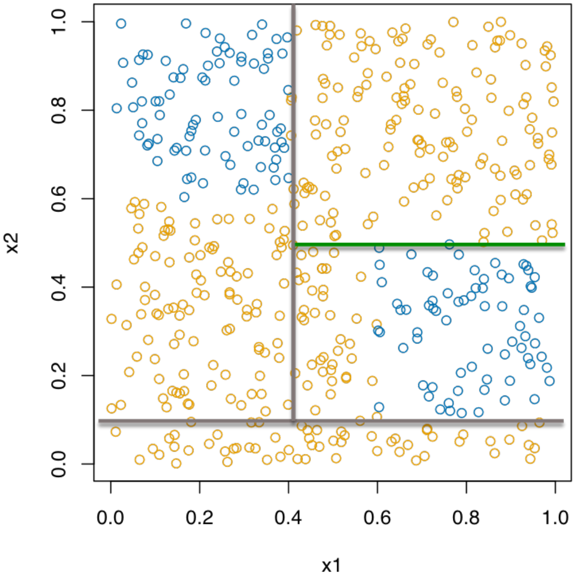
Recursive binary partitioning



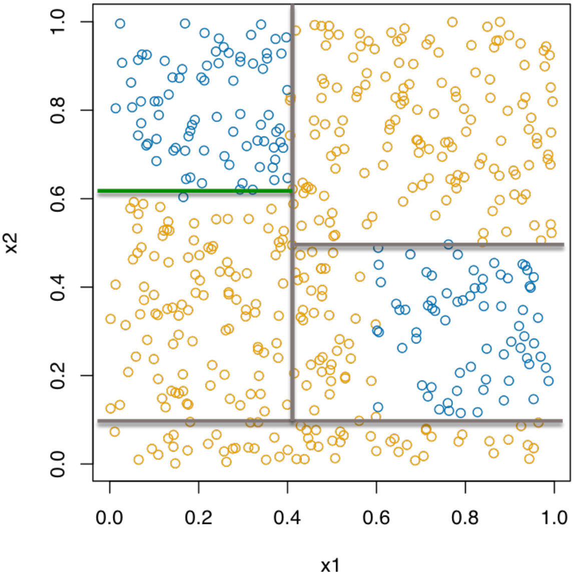
Recursive binary partitioning



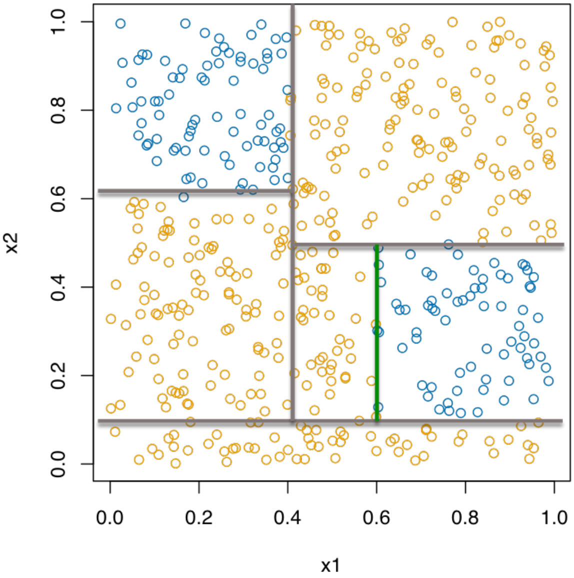
Recursive binary partitioning



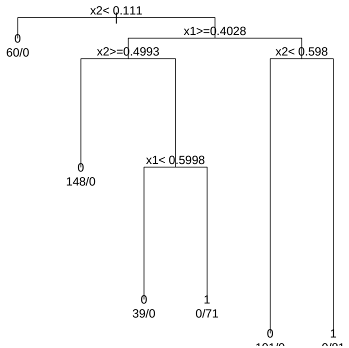
Recursive binary partitioning

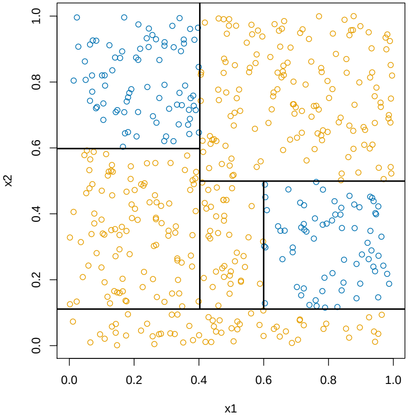


Recursive binary partitioning



The final product





Recursive binary partitioning

- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data

- In prediction, you choose splits to minimize the RSS

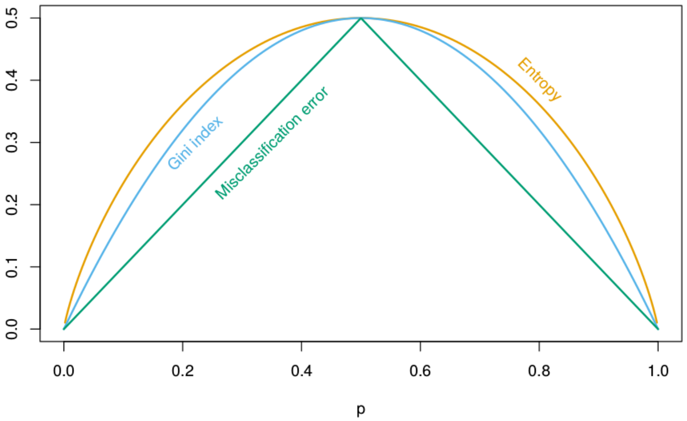
- In classification, choose splits to maximize node purity (minimize Gini index)

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

  where $\hat{p}_{mk}$ is the proportion of *training observations* in the mth region that are from the kth class

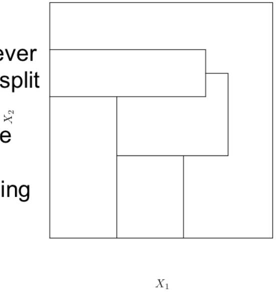
- G is small if all the $\hat{p}_{mk}$ are close to 0 or 1

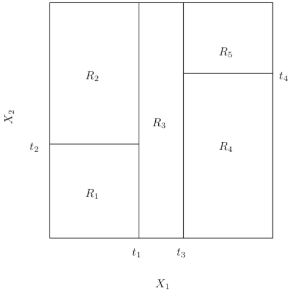
Why not minimize the misclassification error?

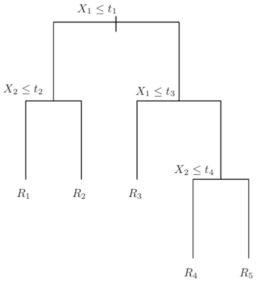


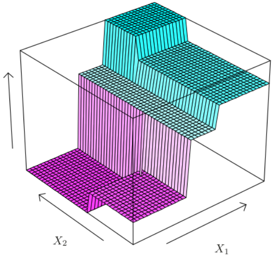
- Misclassification rate is poor at pushing for really pure nodes
- With Gini: going from $\hat{p}_{mk} = 0.8$ to $\hat{p}_{mk} = 0.9$ is better than going from $\hat{p}_{mk} = 0.5$ to $\hat{p}_{mk} = 0.6$
- With Misclassification error, these are considered equal improvements

You'll never get this split from recursive binary partitioning



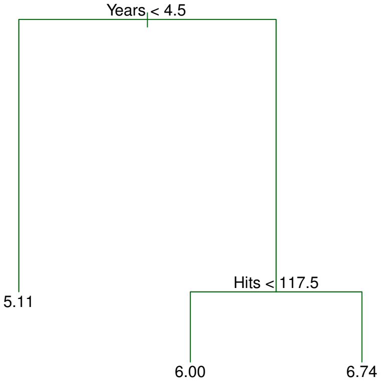


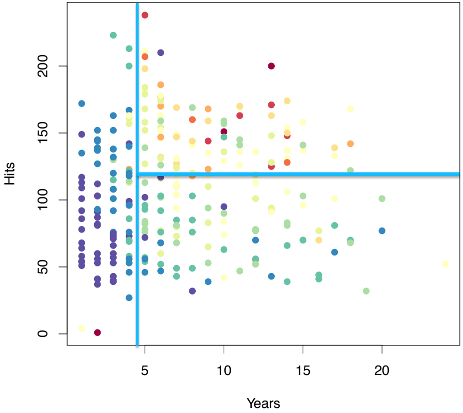




Tree pruning

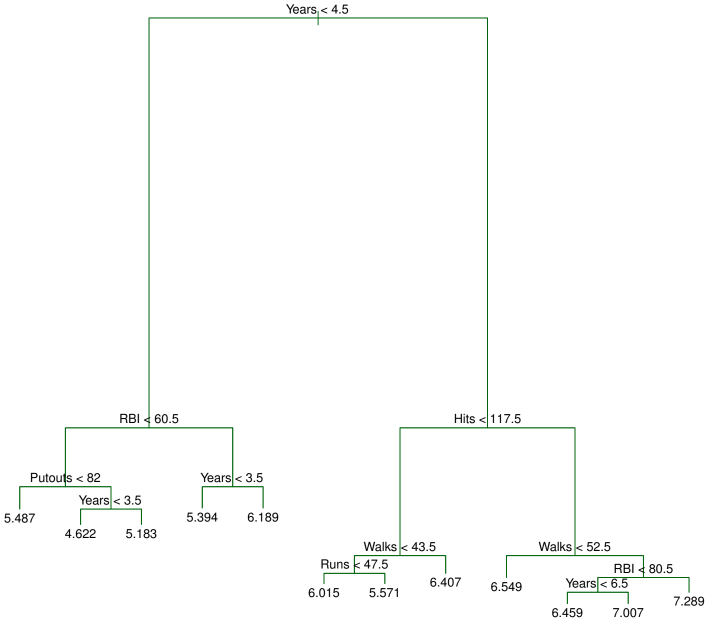
Why did we stop here? Why not keep partitioning?





Low salary (blue, Green)
High salary (orange, red)

We could just keep going...



Tree pruning

- If we just keep going, we're going to overfit the training data, and get poor test performance
- We could stop as soon as we can't find a split to reduce RSS or Gini index by at least some pre-specified amount
- But this strategy is short-sighted: A seemingly worthless split early on might be followed by a really good split later
- Solution: Grow a very large tree T_0, and then prune it back

Cost complexity pruning

- Here's the regression tree version of cost complexity pruning aka weakest link pruning

- For each α, find the subtree $T \subset T_0$ that minimizes

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

  where $|T|$ is the number of terminal nodes (leaves) in tree T, and R_m is the rectangle corresponding ot the mth terminal node. $\hat{y}_{R_m}$ is just the mean of the training observations in R_m

- This is familiar. It has the form:

$$RSS(T) + \alpha|T|$$

  model error + a penalty on model complexity

Cost complexity pruning

For each α, find the subtree $T \subset T_0$ that minimizes

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

- How do we pick α?
- Use Cross-validation

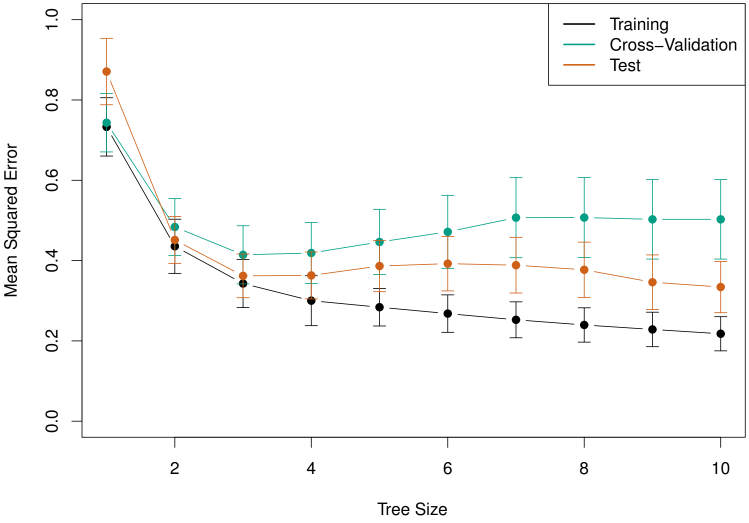
Pruning details

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.

[source: ISL Chapter 8 slides] 21 / 78

Tree pruning



Looks like the small 3-leaf tree has the lowest CV error.

Good things about trees

- Trees are the most easily interpretable method we've talked about in this class
  - You can explain a decision tree to even your least technical colleague
- Arguably, trees more closely *mirror human decision-making*
- Trees are easy to display graphically.
  - You can print off your tree and easily obtain predictions by hand
- Trees can handle qualitative predictors and ordered qualitative predictors without needing to create dummy variables
- Trees handle missing values very nicely, without throwing out observations
  - When a value is missing, they split on a surrogate variable: E.g., if a user's *years of job experience* is missing, they'll split on an optimally chosen correlated variable like *age*.

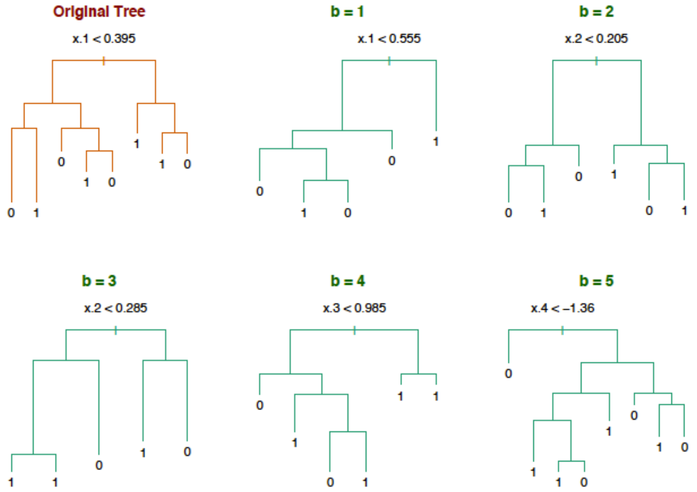
A summary of our methods so far

Method	Interpretable	Flexible	Makes assumptions?
Logistic regression	Yes	Extensible	Yes
k-NN	No	Highly	No
LDA/QDA	Sometimes	No	Yes
Trees	Extremely	Somewhat	No

- Decision trees are perhaps the most Interpretable method we've seen so far
- Trees don't assume any particular relationship between the response Y and the inputs X_j, and large trees are quite flexible
- So what's the catch?
- Turns out, Trees tend to be rather poor predictors/classifiers!
- We can fix this, if we're willing to give up Interpretability

Why are Decision trees poor predictors?

- Decision trees tend to have high variance. A small change in the training data can produce big changes in the estimated Tree.



How are we going to fix this?

- Let's think back to Cross-Validation, and why it gives much better results than the Validation Set Approach

- The Validation Set Approach tends to overestimate the error, but it also gives highly variable estimates
    - If you pick a different random split, you can get *wildly* different estimates of Test error

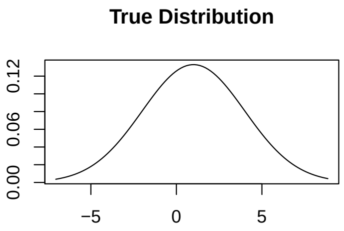
- The K-fold Cross-validation produces much more stable error estimates by averaging over K separate estimates of error (one from each fold).

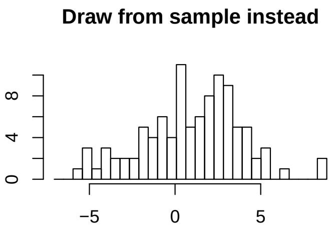
- The idea of Bagging (Bootstrap AGGregatING) has a similar motivation: To decrease the variance of a high-variance predictor, we can average across a bunch of estimates

The Bootstrap

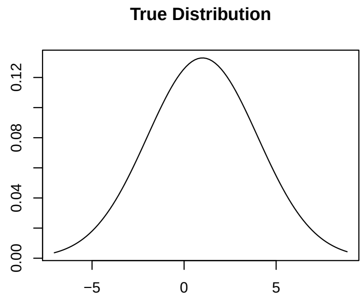
- The Bootstrap[1] is a fundamental resampling tool in statistics.
- Basic idea: We're going to create resampled data sets of size n by sampling from our observed data with replacement
- More formally this idea says: We're going to use the empirical distribution of our data to estimate the true unknown data-generating distribution

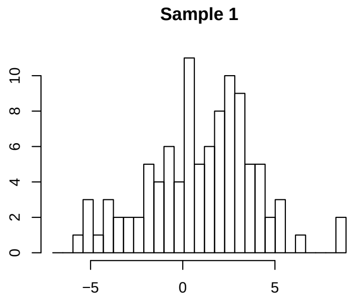
[1] Efron (1979), "Bootstrap Methods: Another Look at the Jackknife"

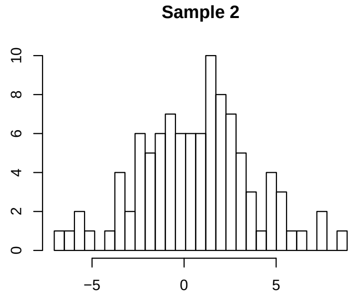


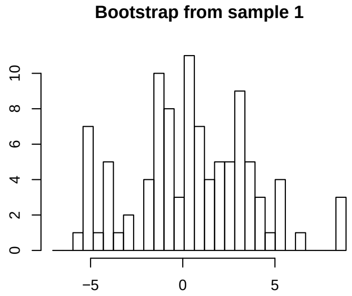


- We'd love to be able to generate more data from the true distribution. This would solve all of our problems.
- But we can't do that. We only get to see a sample of size n
- So we'll approximate sampling from the true distribution by re-sampling from our observed data instead.
  - A bootstrap sample of size n is a data set (x_i^*, y_i^*) $i = 1, \ldots, n$ where each (x_i^*, y_i^*) are sampled uniformly at random with replacement from our observed data $(x_1, y_1), \ldots, (x_n, y_n)$
  - We're (re-)sampling rows of our data, with replacement.


True Distribution
0.00
0.04
0.08
0.12
−5
0
5


Sample 1
0
2
4
6
8
10
−5
0
5


Sample 2
0
2
4
6
8
10
−5
0
5


Bootstrap from sample 1
0
2
4
6
8
10
−5
0
5

What shows up?

- Not all of the training points will appear in each sample
- Each bootstrap sample contains roughly 63.2% of the observed data points
    - The points that randomly get left out points feel like a validation set…we'll return to this later
- If we bootstrap sample B times, we get B data sets of size n, and we can estimate whatever we want on each dataset

Bagging: Classification trees

- Given a training data (x_i, y_i), $i = 1, \ldots n$, bagging[2] averages the predictions from classification trees over a collection of bootstrap samples.
- Here we'll describe how to apply Bagging to Classification Trees
  1. For $b = 1, \ldots, B$, get a bootstrap sample of size n from the training data: (x_i^{*b}, y_i^{*b}), $i = 1, \ldots n$
  2. Fit a classification tree $\hat{f}^{\text{tree},b}$ on each sample
  3. Classify a new point x_0 by taking the plurality vote across all B bootstrapped trees:

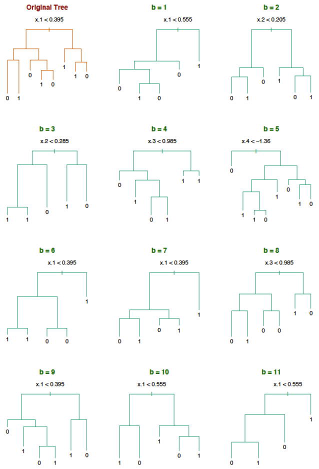
$$\hat{y}_0^{\text{bag}} = \underset{k=1,\ldots,K}{\operatorname{argmax}} \sum_{b=1}^{B} I\left(\hat{f}^{\text{tree},b(x_0)} = k\right)$$

- Step (3) amounts to letting each of the B trees *vote*, and then choosing whichever class has the most votes
- Typically, in Step (2) the trees are grown very large, with no pruning. Why are we less worried about tuning each tree?

[2]Breiman (1996), "Bagging Predictors"

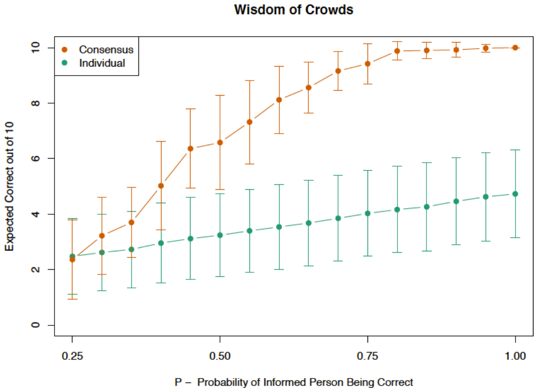
Example: bagging

Example (from ESL 8.7.1): $n = 30$ training data points, $p = 5$ features, and $K = 2$ classes. No pruning used in growing trees:



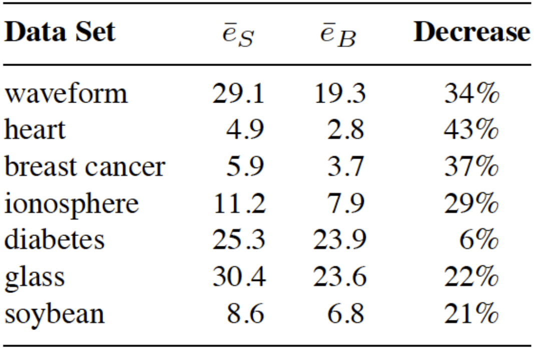
How could this possibly work?

- You may have heard of the Wisdom of crowds phenomenon
- It's a concept popularized outside of statistics to describe the idea that the collection of knowledge of a group of independent people can exceed the knowledge of any one person individually.
- Interesting example (from ESL page 287):



Example: Breiman's bagging

Example from the original Breiman paper on bagging: comparing the misclassification error of the CART tree (pruning performed by cross-validation) and of the bagging classifier (with $B = 50$):



Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Voting probabilities are not estimated class probabilities

- Suppose that we wanted *probability estimates* $\hat{p}_k(x)$ out of our bagging procedure.
- What if we tried using:

$$\hat{p}_k^{\text{vote}}(x) = \frac{1}{B} \sum_{b=1}^{B} \left(\hat{f}^{\text{tree},b}(x) = k \right)$$

  This is the proportion of bootstrapped trees that voted for class k.

- This can be a bad idea
- Suppose we have two classes, and the true probability that $y_0 = 1$ when $X = x_0$ is 0.75.
- Suppose each of the bagged trees $\hat{f}^{\text{tree},b}(x)$ correctly classifies x_0 to class 1
- Then $\hat{p}_1^{\text{vote}}(x) = 1$... that's wrong
- What if we used each tree's estimated probabilities instead?

Alternative form: Probability Bagging

- Instead of just looking at the class predicted by each tree, look at the *predicted class probabilities* $\hat{p}_k^{\text{tree},b}(x)$
- Define the bagging estimate of class probabilities:

$$\hat{p}_k^{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{p}_k^{\text{tree},b}(x) \quad k = 1, \ldots K$$

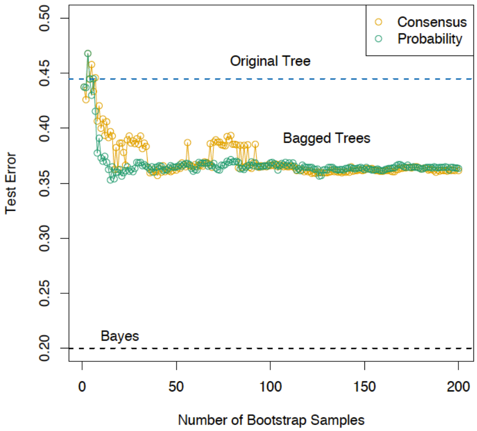
- We can use $\hat{p}_k^{\text{bag}}(x)$ itself as an *alternative* to plurality voting of the trees.
- Given an input vector x_0, we can classify it according to

$$\hat{y}_0^{\text{bag}} = \underset{k=1,\ldots K}{\operatorname{argmax}} \; \hat{p}_k^{\text{bag}}(x)$$

- This form of bagging is preferred if we want to estimate class probabilities, and it may improve overall classification accuracy

Comparison of the two bagging approaches

The probability form of bagging produces misclassification errors shown in green. The Consensus version is what we first introduced. It's not as well behaved.



The Test error eventually stops decreasing past a certain value of B because we hit the limit in the variance reduction bagging can provide

Out-of-Bag (OOB) Error Estimation

- Recall, each bootstrap sample contains roughly 2/3 ($\approx 63.2\%$) of the of the training observations

- The remaining observations not used to fit a given bagged tree are called the out-of-bag (OOB) observations

- Another way of thinking about it: Each observation is OOB for roughly $B/3$ of the trees. We can treat observation i as a test point each time it is OOB.

- To form the OOB estimate of test error:
  - Predict the response for the ith observation using each of the trees for which i was OOB. This gives us roughly $B/3$ predictions for each observation.
  - Calculate the error of each OOB prediction
  - Average all of the errors

Random Forests

- Random forests provide an improvement over bagged trees by incorporating a small tweak that decorrelates the individual trees
  - This further reduces variance when we average the trees

- We still build each tree on a bootstrapped training sample

- But now, each time a split in a tree is considered, the tree may only split on a predictor from a randomly selected subset of m predictors

- A fresh selection of m randomly selected predictors is presented at each split... not for each tree, but for each split of each tree

- $m \approx \sqrt{p}$ turns out to be a good choice
  - E.g., if we have 100 predictors, each split will be allowed to choose from among 10 randomly selected predictors

Bagging vs. Random Forests

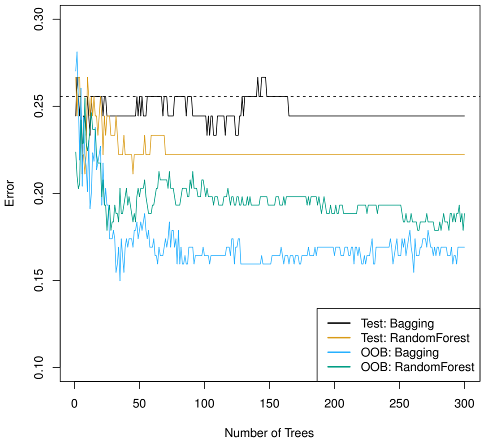


Figure 8.8 from ISL. Various fits to the `Heart` data

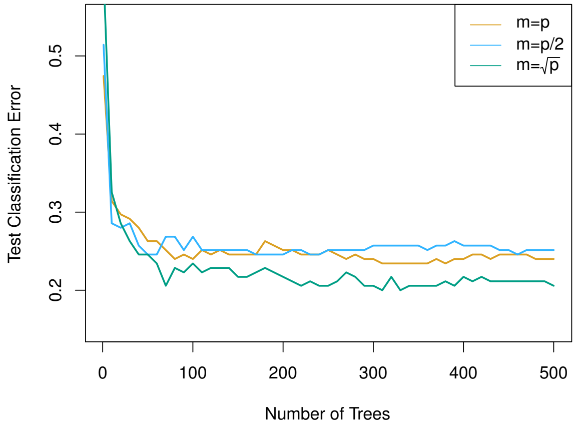
Dashed: Error from a single Classification tree

Random forest fit with $m = 4 \approx \sqrt{13} = \sqrt{p}$

A big data example: Gene expression data

- $p = 4{,}718$ genetic measurements from just 349 patients
- Each patient has a qualitative label. $K = 15$ possible labels
  - Either normal, or one of 14 types of cancer
- Split data into training and testing, fit Random forest to training set for 3 different choices of number of splitting variables, m.
- First, filter down to the 500 genes that have the highest overall variance in the training set

Test error: Gene expression data



- Curves show Test misclassification rates for a 15-class problem with $p = 500$ predictors and under 200 observations used for training
- x-axis gives number of trees (number of bootstrap samples used)
- $m = p$ corresponds to bagging.
- A single classification tree has an error rate of 45.7%.

Acknowledgements

All of the lectures notes for this class feature content borrowed with or without modification from the following sources:

- 36-462/36-662 Lecture notes (Prof. Tibshirani, Prof. G'Sell, Prof. Shalizi)
- 95-791 Lecture notes (Prof. Dubrawski)
- *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
- *Applied Predictive Modeling*, (Springer, 2013), Max Kuhn and Kjell Johnson