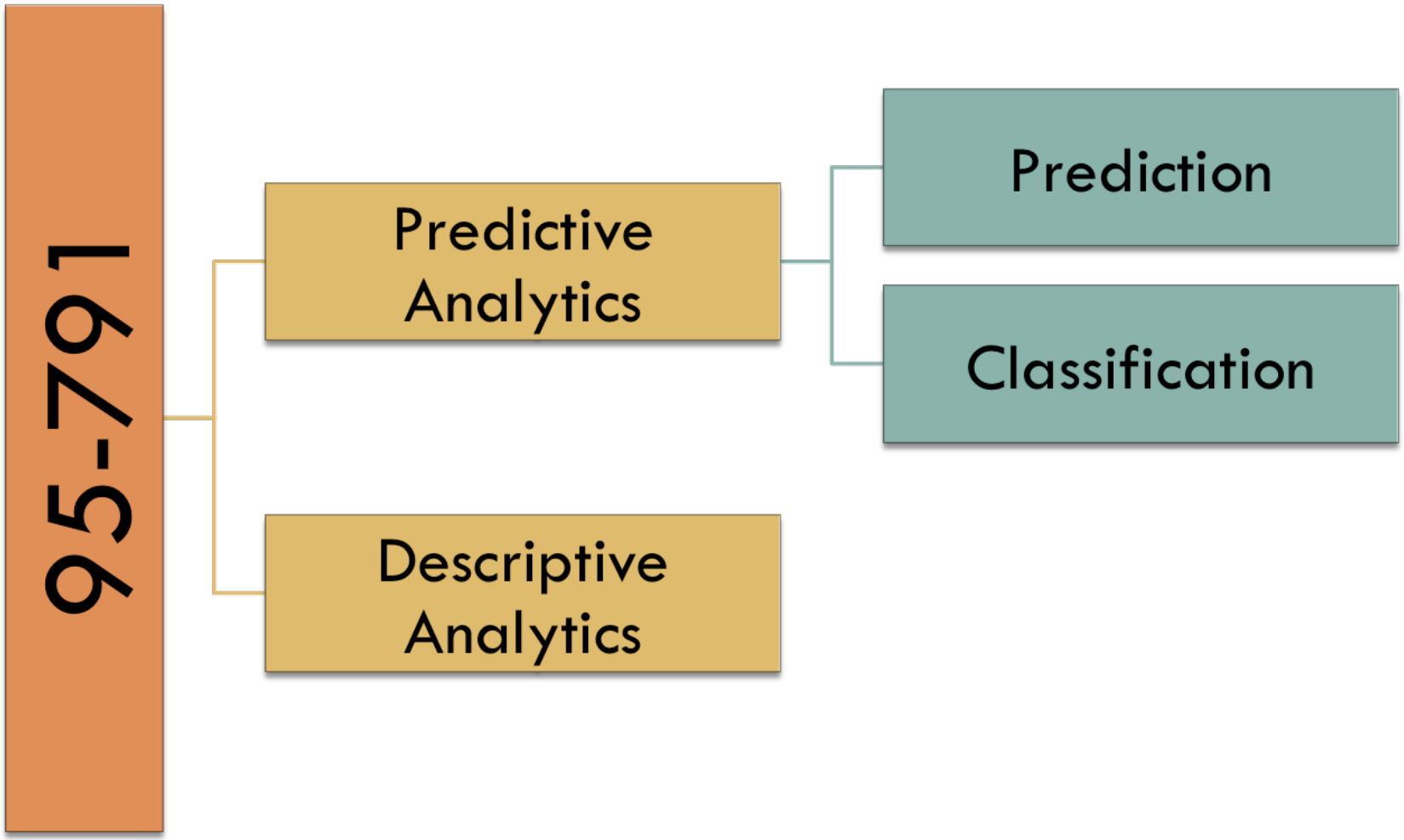
Lecture 9: Classification, Trees

Assessing Performance of Classification Models, Tree-Based methods

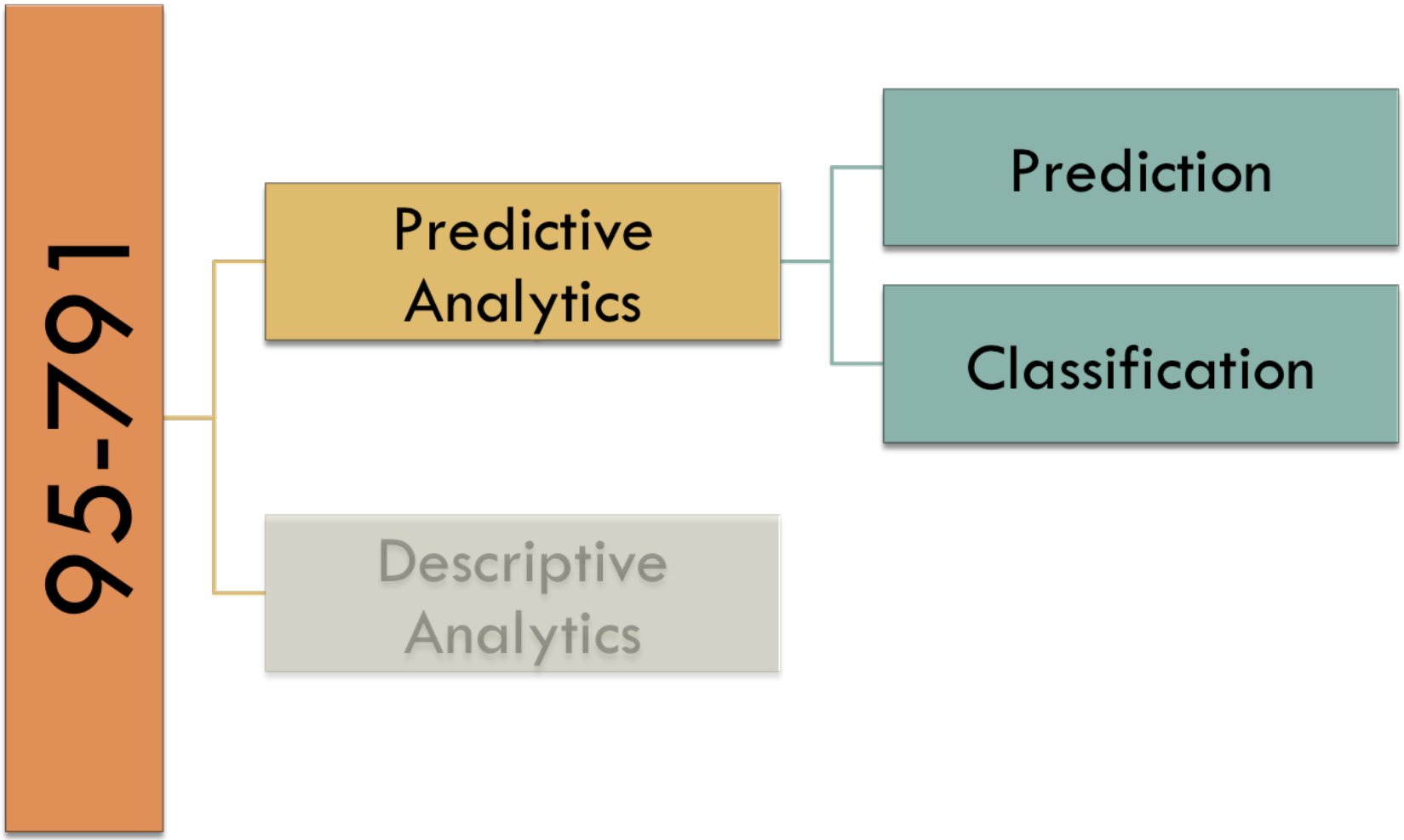
Prof. Alexandra Chouldechova
95-791: Data Mining

April 12, 2016

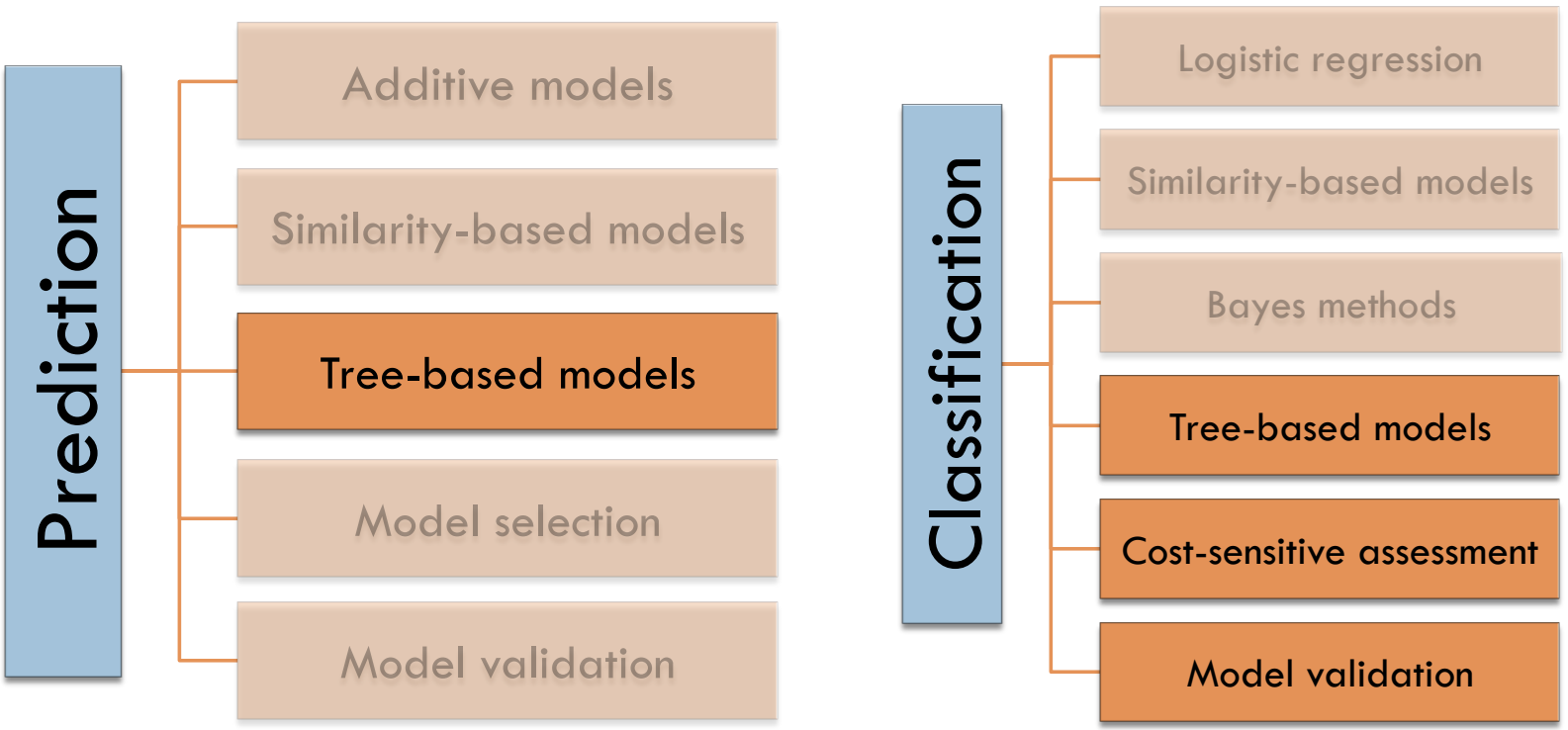
Course Roadmap



Course Roadmap



Today's topics



Agenda

- **Assessing performance of Classification Models**
  - Calibration plots
  - Confusion matrices
  - Sensitivity, Specificity, Accuracy, Precision, Recall
  - Cost-based criteria
  - ROC curves

- **Final project**

Assessing the performance of Classifiers

The confusion matrix

- Let's focus again on the binary classification setting:
  - $Y = 1$: if the event happened
  - $Y = 0$: if the event did not happen
- The primary building block of essentially all approaches to evaluating a Classifier is the confusion matrix

Predicted	Observed	
	Event	Nonevent
Event	TP	FP
Nonevent	FN	TN

- In **R**, it's more natural to form confusion matrices with the Non-event and Event headings swapped.

		Observed	
		No	Yes
Predicted	No	TN	FN
	Yes	FP	TP

[source: Applied Predictive Modeling]

Probabilities as ranking functions

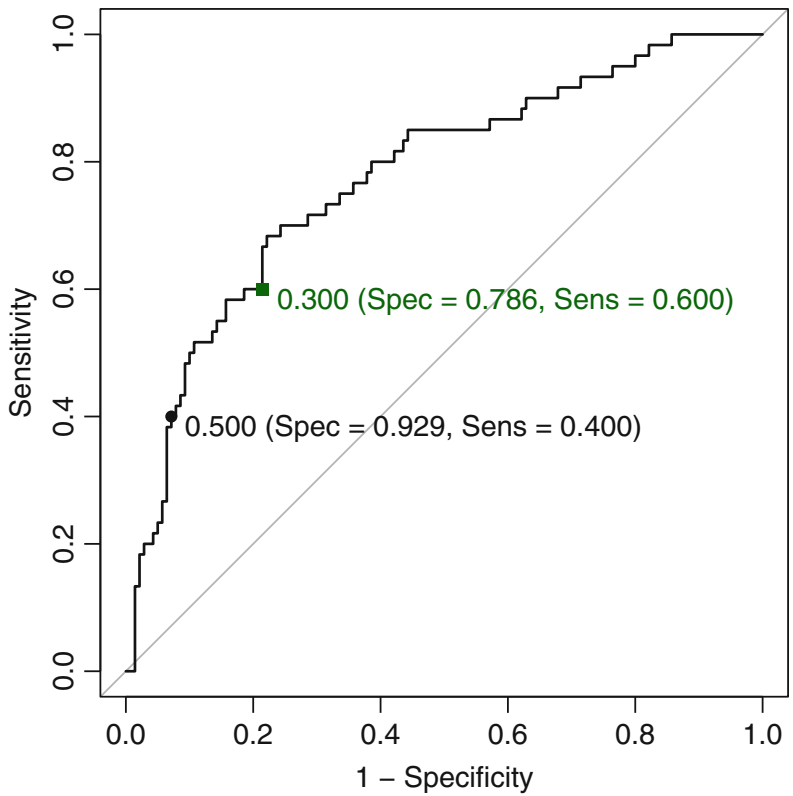
- Suppose we have a probability estimate $\hat{p}(x)$
- We can use $\hat{p}(x)$ to order the observations from *most likely to have the Event* to *least likely*

i	$\hat{p}(x_i)$	y_i
45	0.975	1
12	0.824	0
191	0.762	1
77	0.754	1
$\vdots$	$\vdots$	$\vdots$

- We can think about how well $\hat{p}(x)$ performs by asking: When we order the y_i according to $\hat{p}(x)$, do most of the observations with $y_i = 1$ appear at the top of the list?
- A perfect ranking function will score all of the observations where $y_i = 1$ higher than those where $y_i = 0$
- We're now going to discuss various approaches for visualizing how well $\hat{p}(x)$ does at ranking observations

ROC Curves

- As we vary our probability cutoff α, we get different classification rules and hence different values of all of our performance metrics
- You can think of getting a different confusion matrix at each α
- It's useful to plot the values of various performance metrics as you vary the cutoff α
- Perhaps the most widely used plot is the ROC Curve



[source: Applied Predictive Modeling]

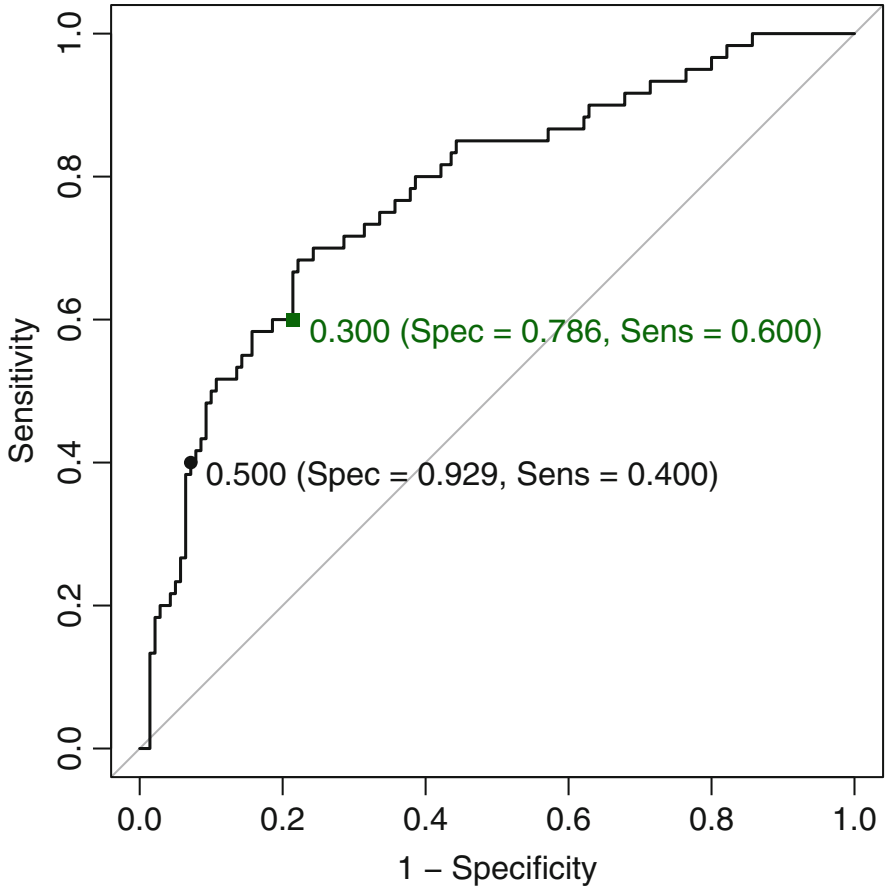
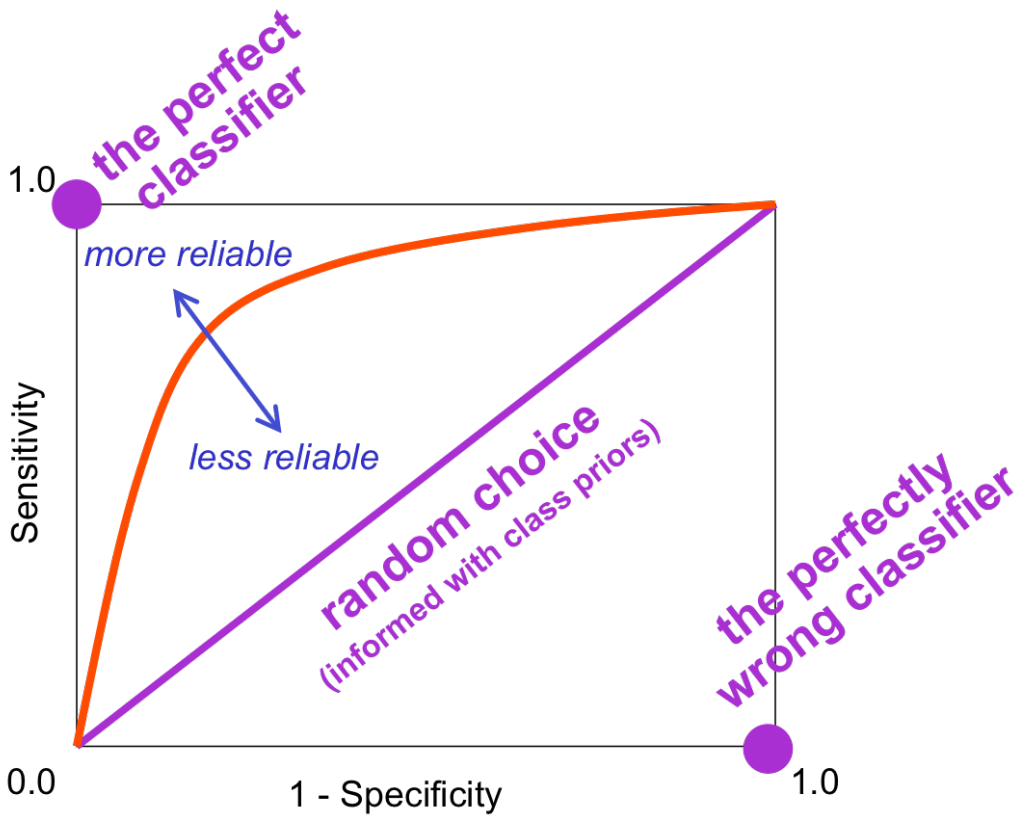


Fig. 11.6: A receiver operator characteristic (ROC) curve for the logistic regression model results for the credit model. The dot indicates the value corresponding to a cutoff of 50 % while the green square corresponds to a cutoff of 30 % (i.e., probabilities greater than 0.30 are called events)

Each point on the curve corresponds to the value of (1−Specificity, Sensitivity) calculated at a particular choice of cutoff α

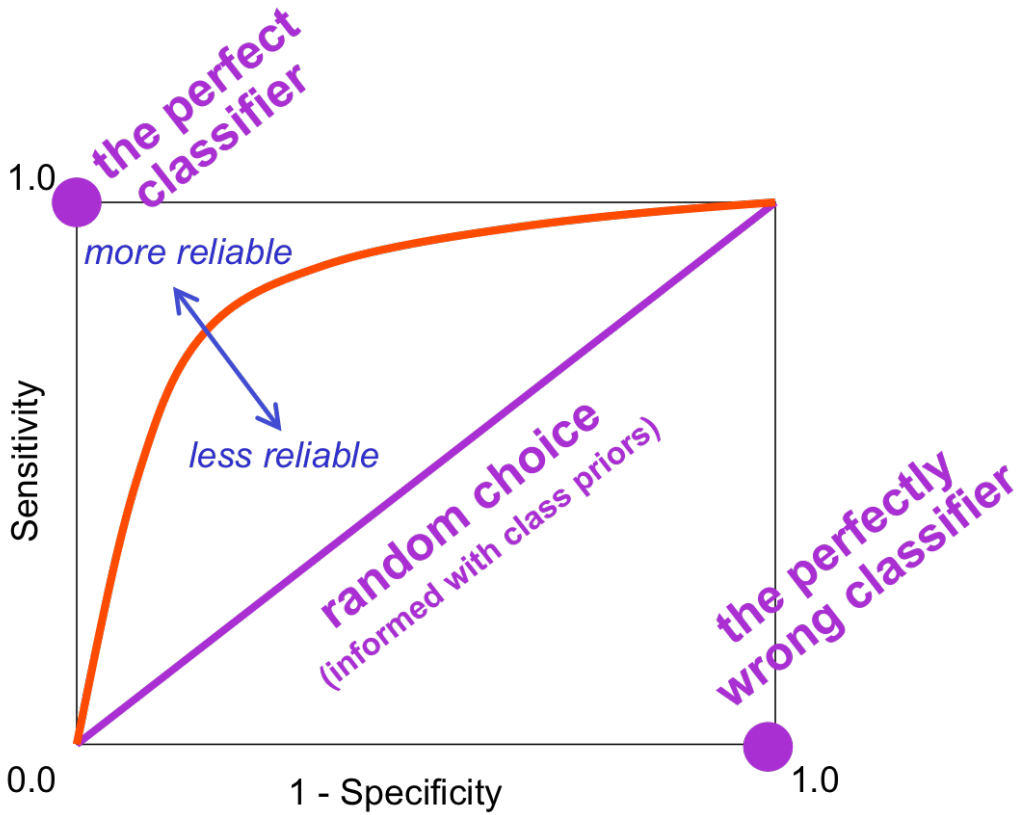
[source: Applied Predictive Modeling]

ROC



- The diagonal is the ROC you would get from randomly picking proportion π_k of the observations to classify to class k
- Higher ROC is better
- The perfect classifier has $(1 - \text{Specificity}, \text{Sensitivity}) = (0, 1)$

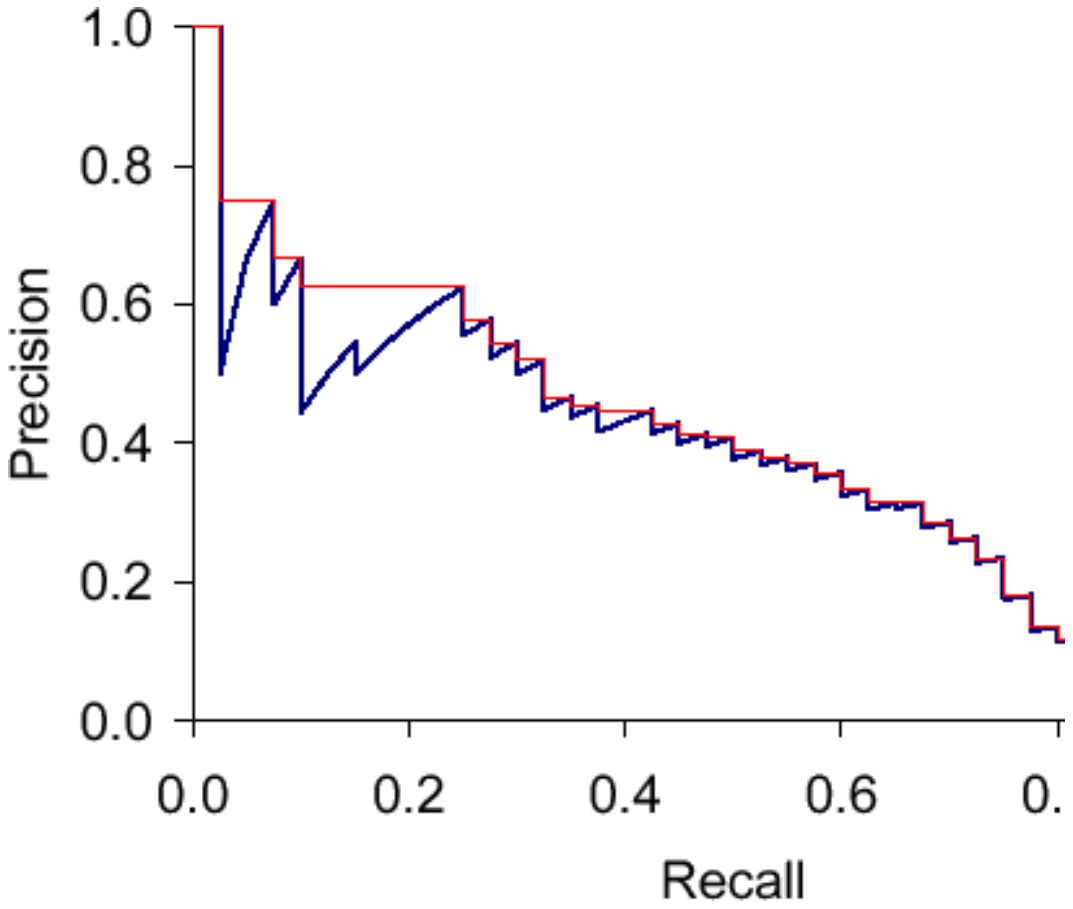
Area under the curve



- The AUC is the *area under the ROC curve*

- AUC has a nice interpretation: The AUC is the probability that the classifier will rank a *randomly selected* observation where $y_i = 1$ higher than a *randomly selected* observation where $y_i = 0$

Precision-Recall curves



- Precision: $TP/(TP+FP)$ (aka, PPV)
- Recall: $TP/(TP+FN)$ (aka, Sensitivity)
- Precision @50% Recall is a common performance metric

[source: Introduction to Information Retrieval, Manning et al.]

Lift charts

- Lift charts are kind of like ROC curves, but may be more useful depending on the application

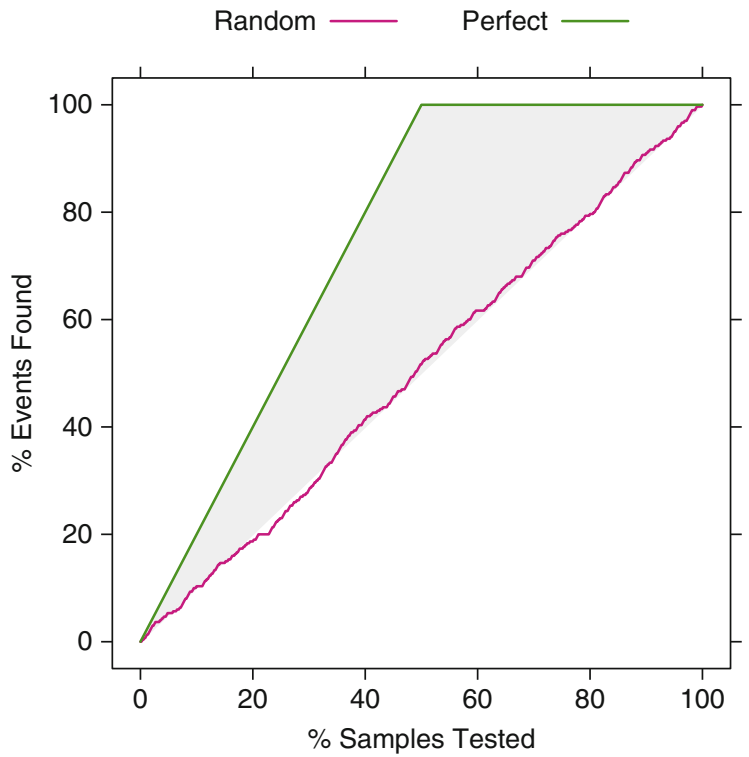
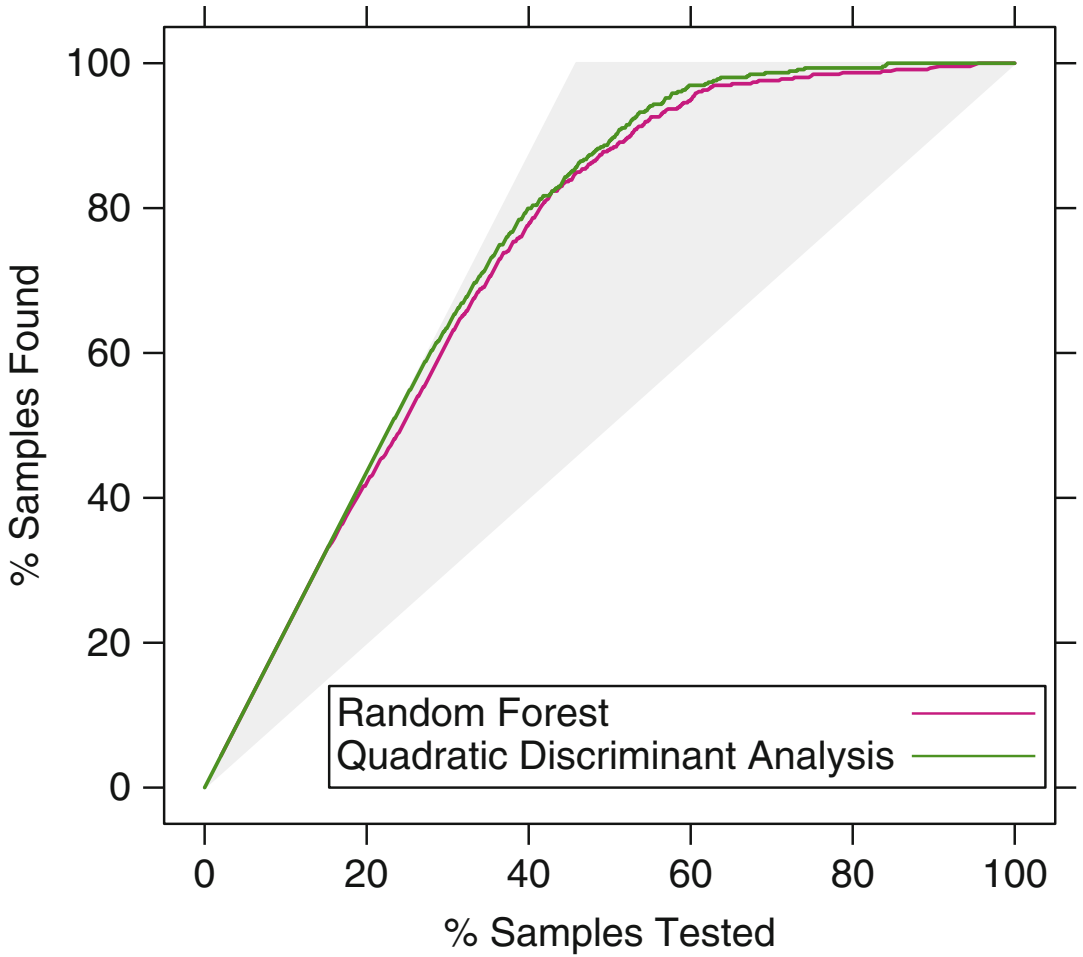


Fig. 11.7: An example lift plot with two models: one that perfectly separates two classes and another that is completely non-informative

- y-axis: Recall (Sensitivity)
- x-axis: $\#\{i : \hat{p}(x_i) > \alpha\}/n = (FP + TP)/n$

[source: Applied Predictive Modeling]

Lift charts

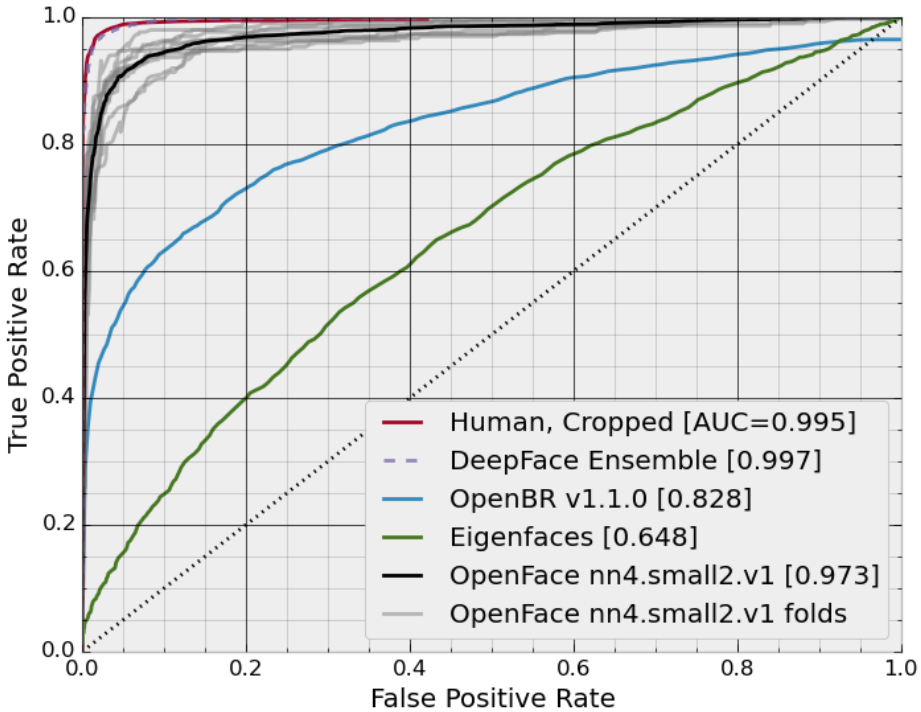


- In this example, of the top 40% of observations ordered according to $\hat{p}(x)$, essentially all of them have $y_i = 1$. This is great!

[source: Applied Predictive Modeling]

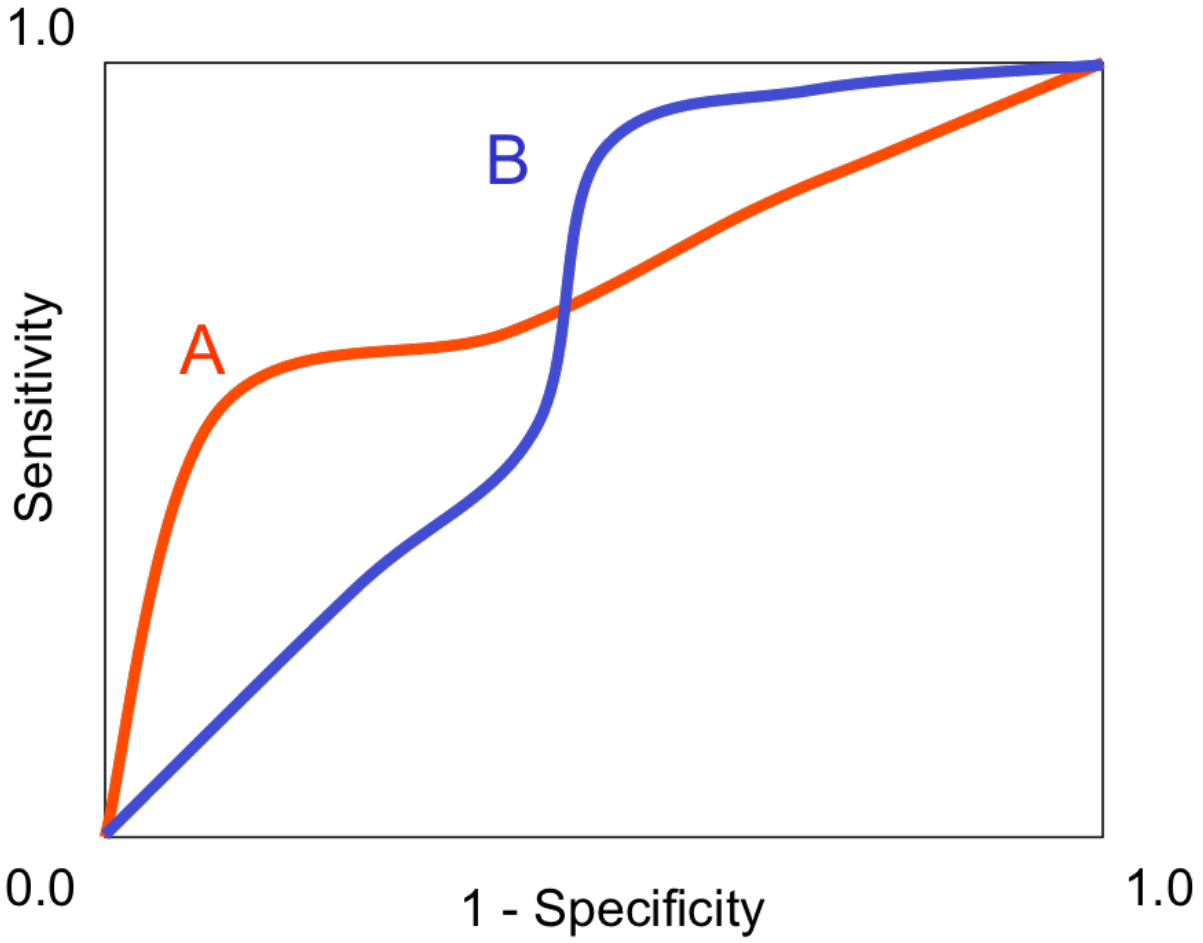
How do we pick the best classifier?

You can overlay the ROC curves from a bunch of different methods.
Here's a facial recognition example. [source: cmusatyalab/openface GitHub]



- The DeepFace Ensemble method is amazing
- The proposed method, OpenFace nn4.small2.v1 does really well. Its ROC curve is the solid black line.
  - 10 grey curves are Test Fold ROC curves from 10-Fold CV

Now which one is better?



It depends on what region of the curve you care most about.

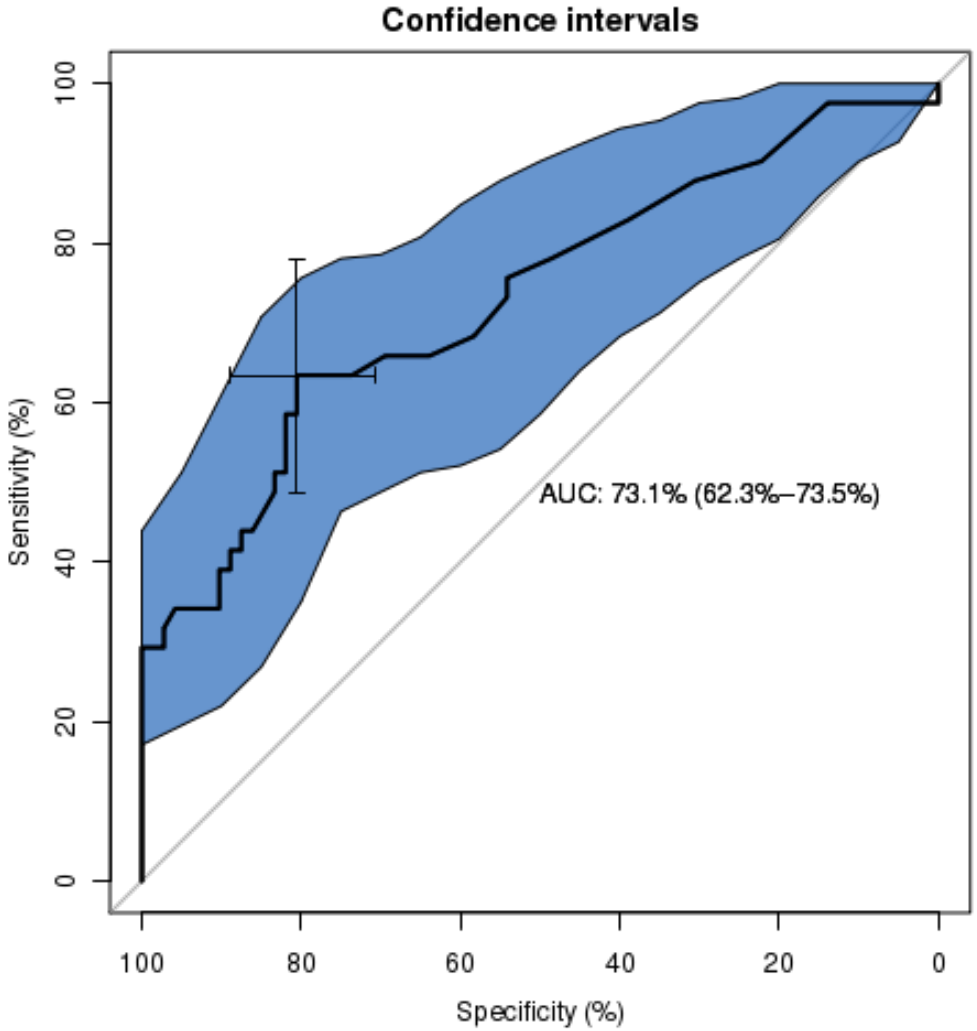
Example: Cancer diagnosis

We can think of two settings: Screening, and Referred examinations

- Screening: E.g., we want annual screening of all people above age 40
  - Most people we test will *not* have cancer (high ratio of non-Events to Events)
  - False positives more costly than False negatives
  - Focus on: performance at High Specificity (small x-axis values)
- Referred examination: E.g., your doctor feels a bump under your skin, and refers you for a biopsy to get it tested for cancer
  - Many referred individuals *will* have cancer (*low* or *equal* ratio of non-Events to Events)
  - False negatives are really costly
  - Focus on: performance at High Sensitivity (high y-axis values)

Putting confidence bands on ROC curves

- We like putting standard error bars on our curves so that we can visually discern which trends/differences are *statistically significant*



Using Cross-validation

- In the Prediction setting, we focussed entirely on MSE as our performance metric

- To validate our model, we would use K-Fold CV to estimate the Test MSE

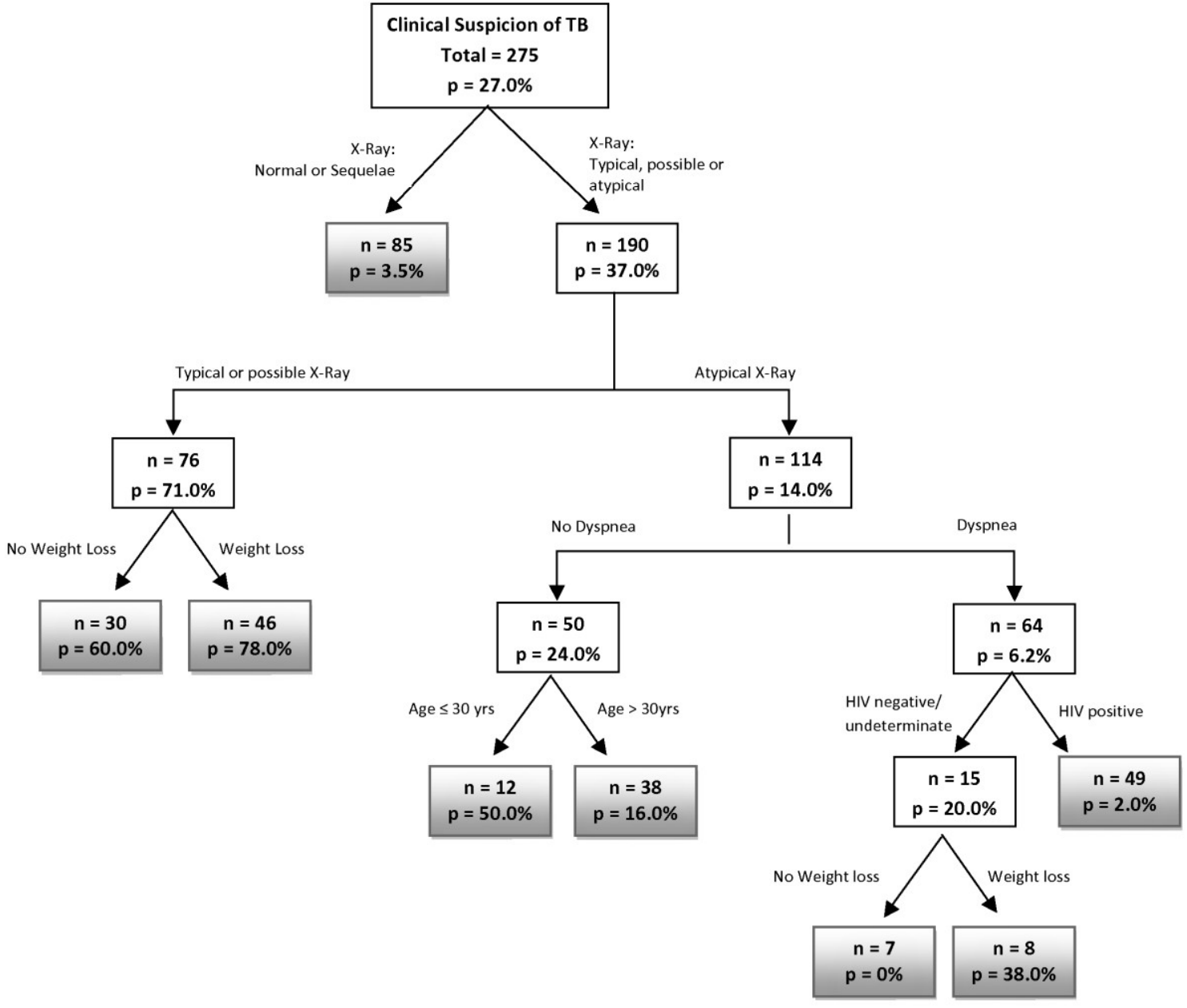
- In the Classification setting, there are *many* metrics out there. The set I presented is by no means exhaustive.

- To estimate Test performance:
  1. Pick a metric (E.g., Accuracy, profit, AUC, Sensitivity @x% Specificity, Precision @x% Recall, etc.)
  2. Calculate the metric on each fold of K-fold CV
  3. Average over all of the folds

- For ROC and Precision-Recall curves, you may want to show the curve you get from each Test fold. This gives a visual representation of the variability of the curve estimates.

Back to methods...

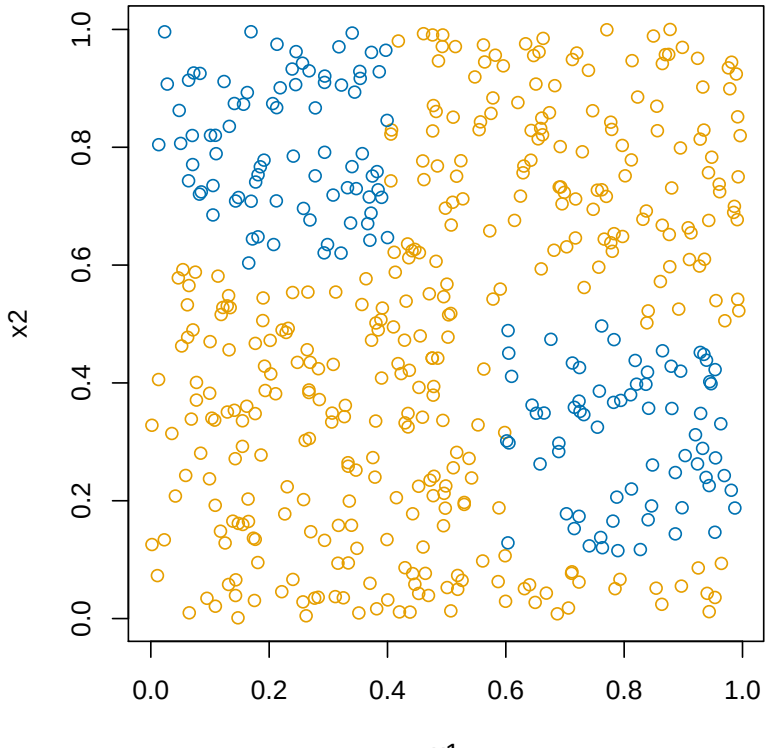
Let's grow some Trees



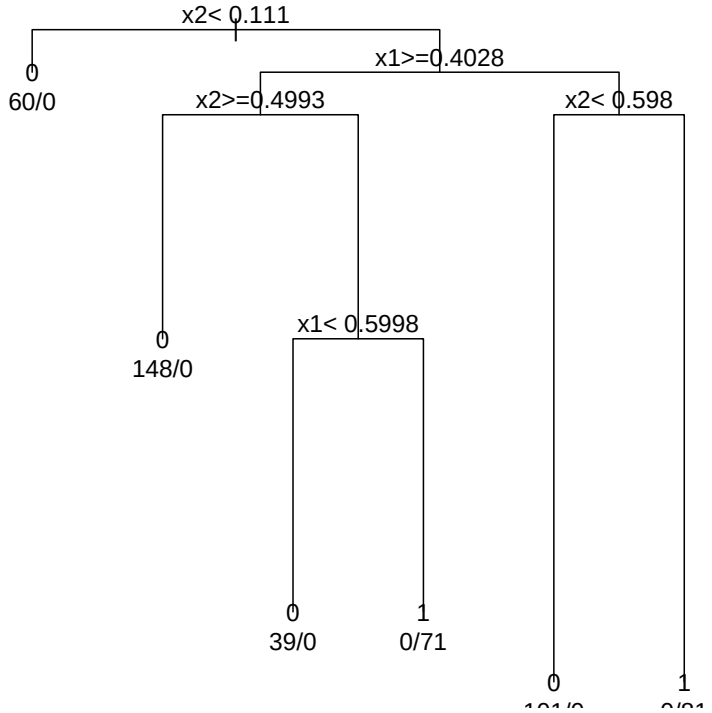
[source: Classification and regression tree (CART) model to predict pulmonary tuberculosis in hospitalized patients, Aguiar et al]

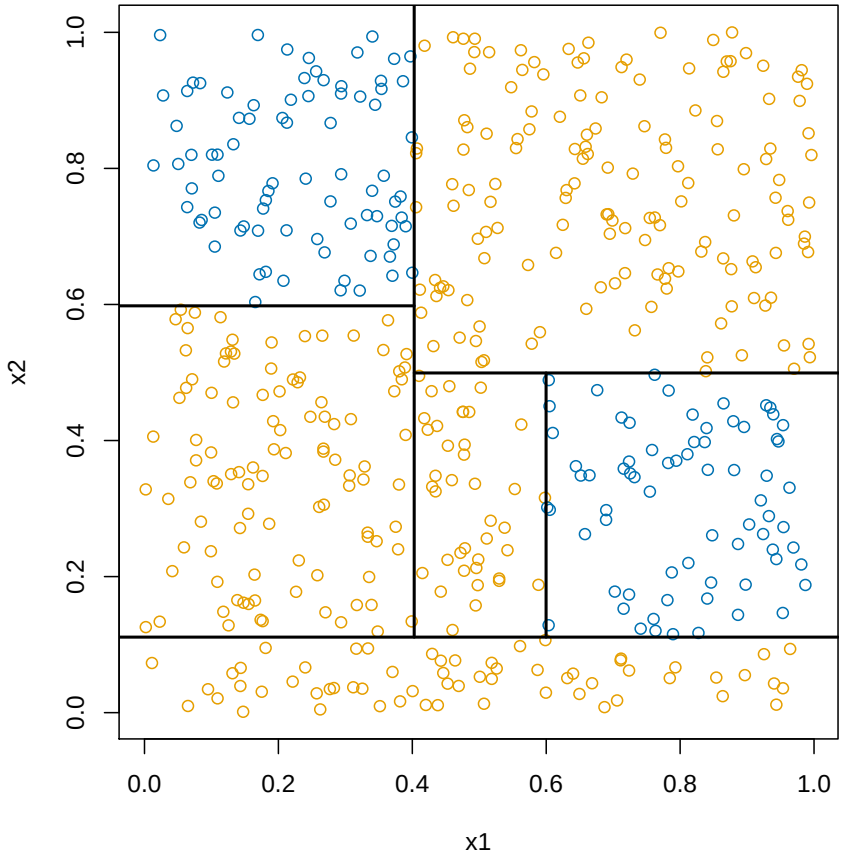
Overview: Tree-based methods

- 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?

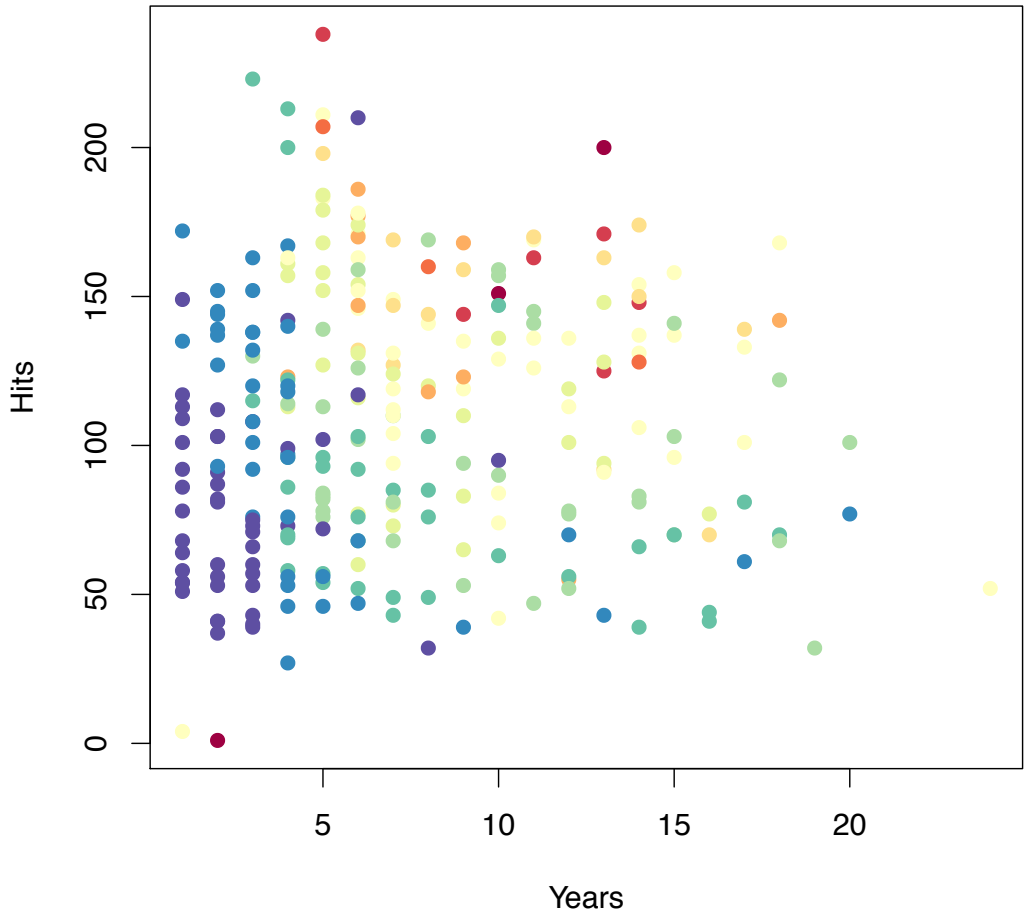




- Trees are built up via a greedy algorithm: Recursive binary partitioning
- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data
  - In prediction, you choose splits to minimize the RSS
  - In classification, choose splits to maximize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

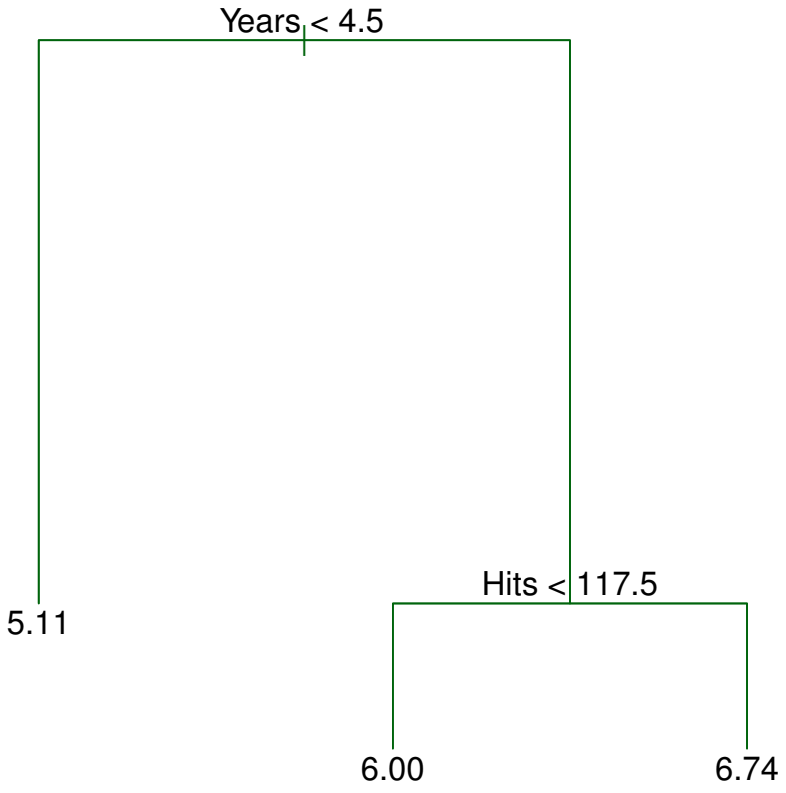
Decision trees in Prediction

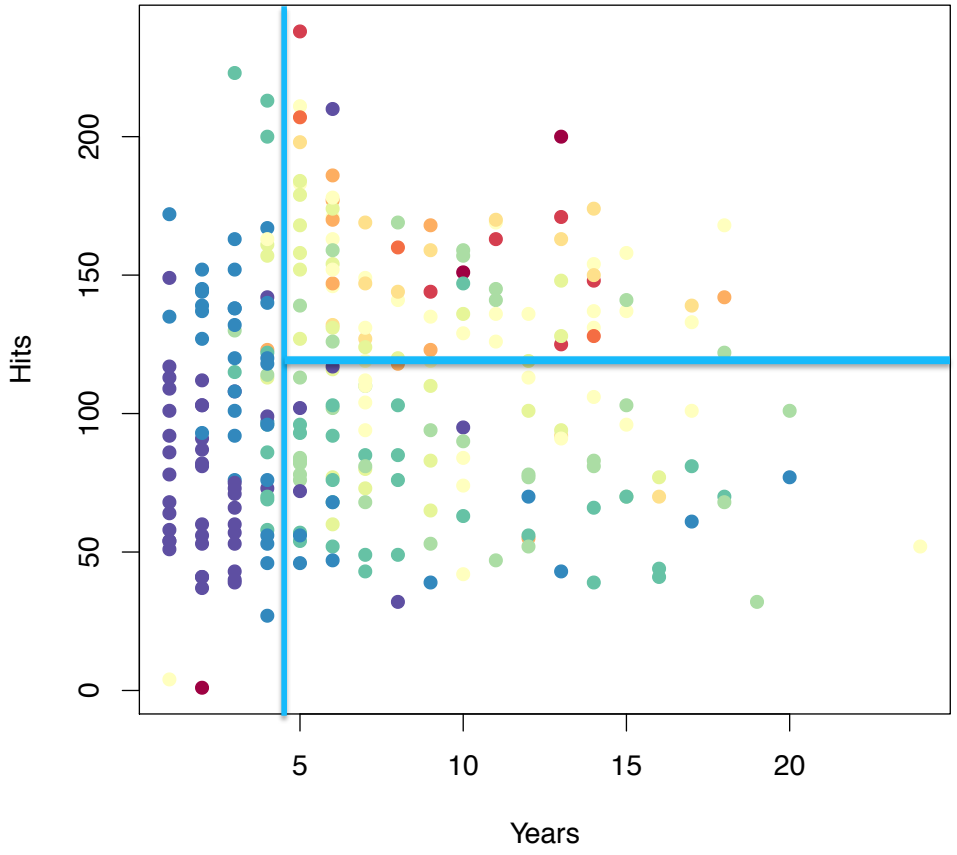
Here's a Prediction example (Y = `Salary` in millions)



Low salary (blue, Green)

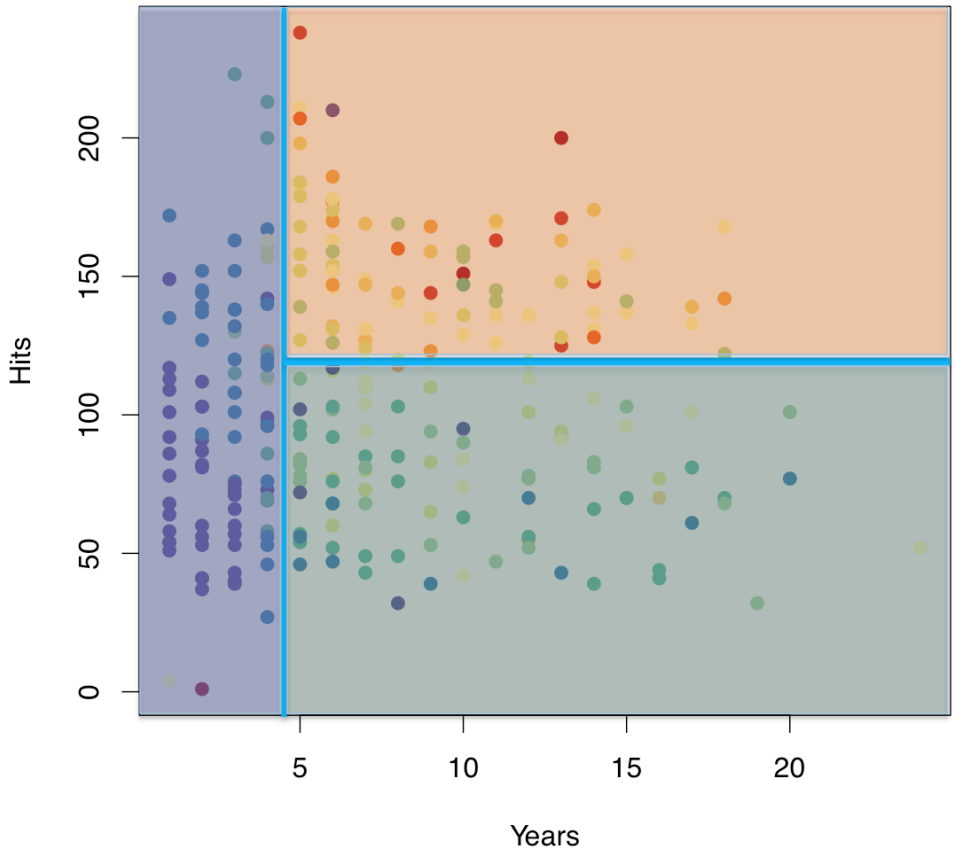
High salary (orange, red)





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

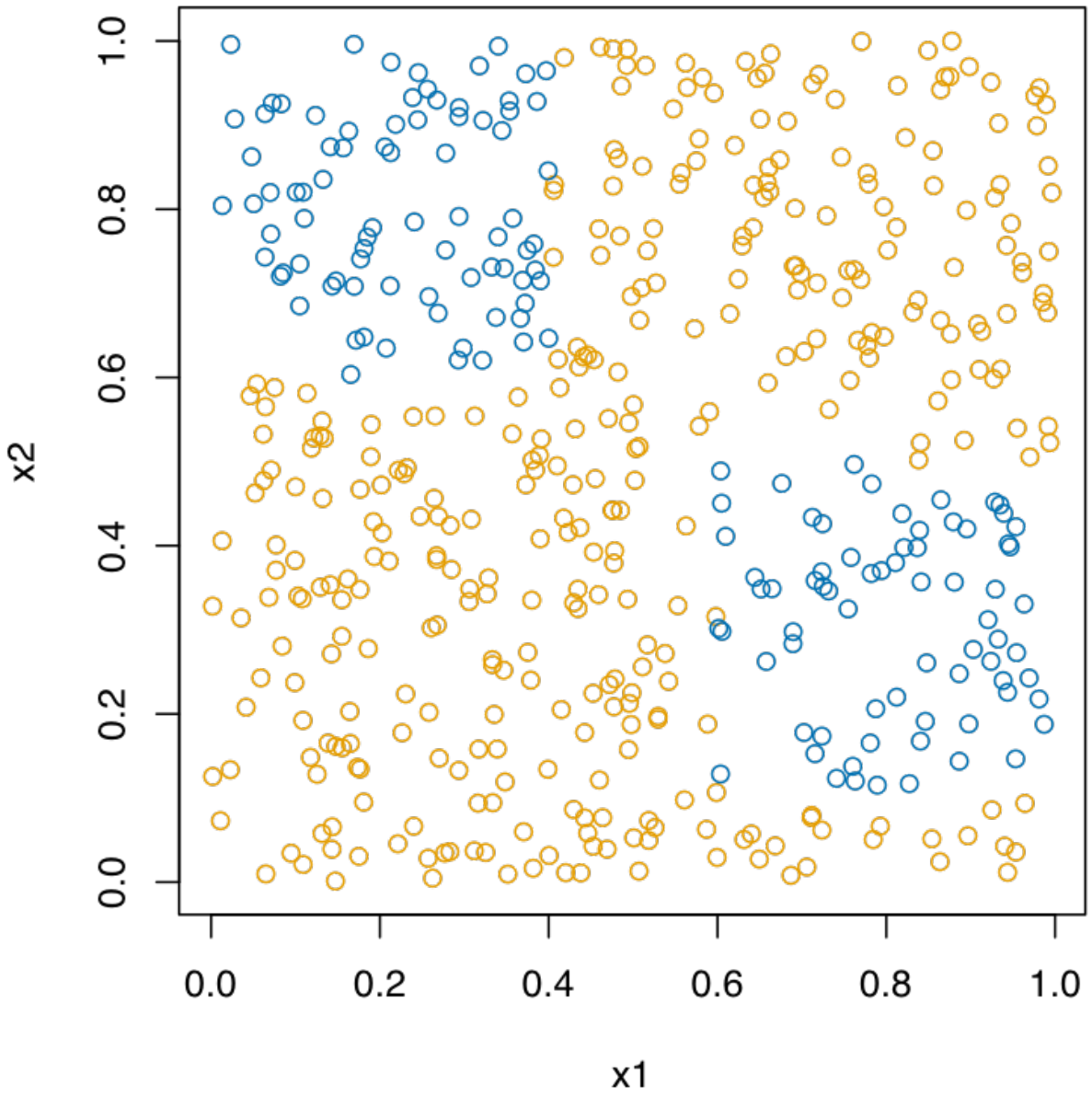
Within each of the 3 rectangles, we predict `Salary` using the average value of `Salary` in the training data



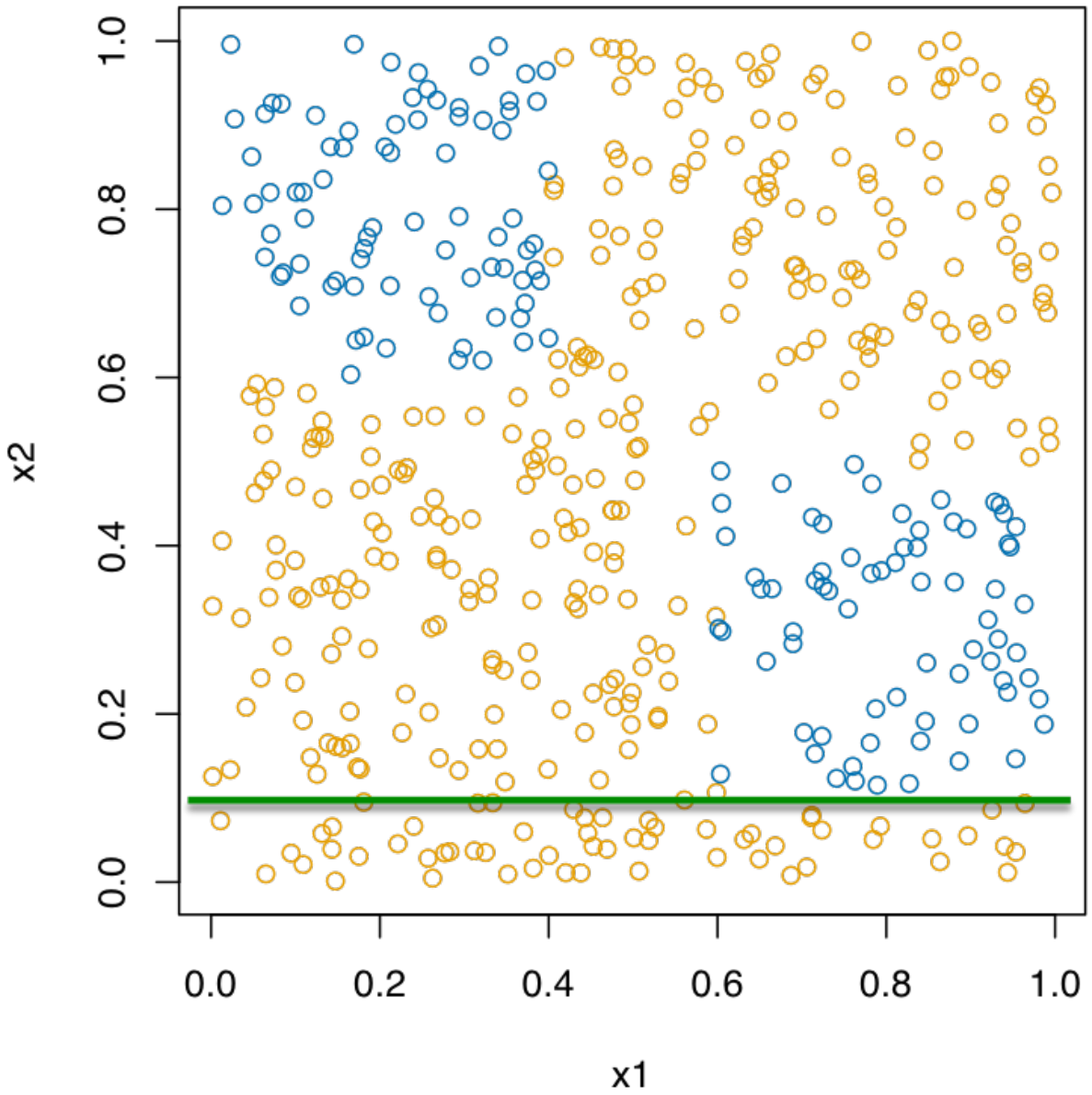
Low salary (blue, Green)
High salary (orange, red)

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

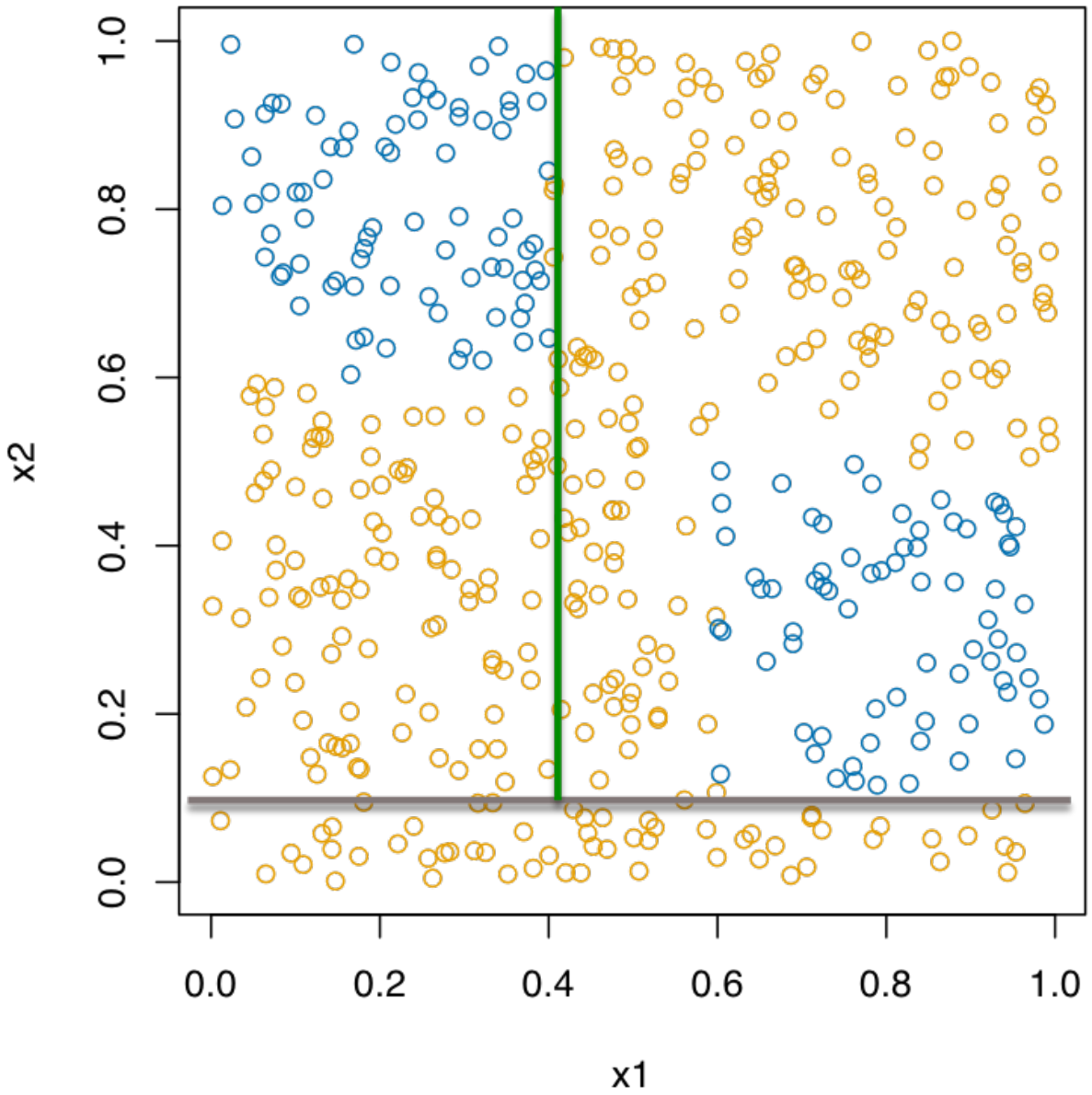
Recursive binary partitioning



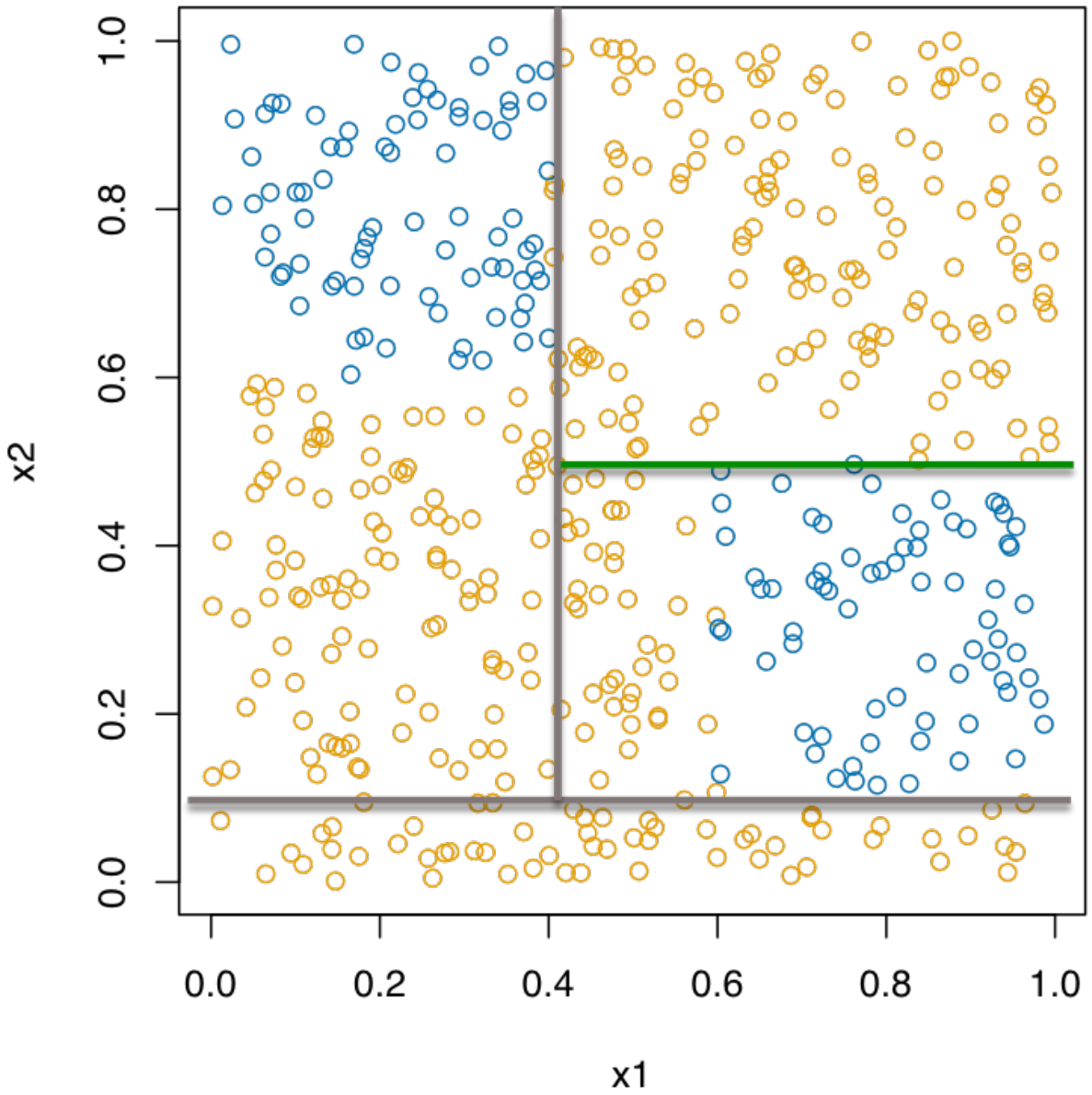
Recursive binary partitioning



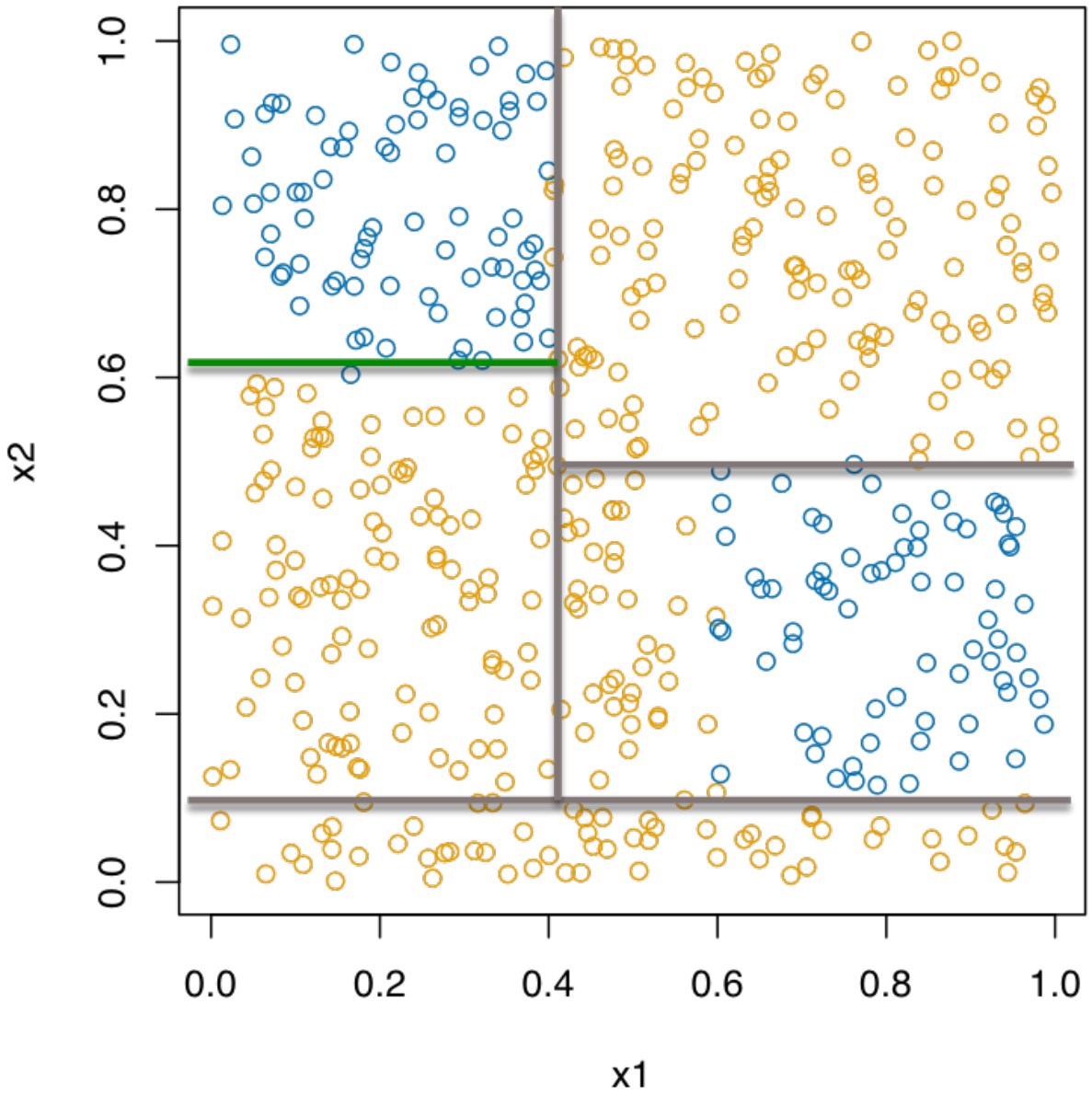
Recursive binary partitioning



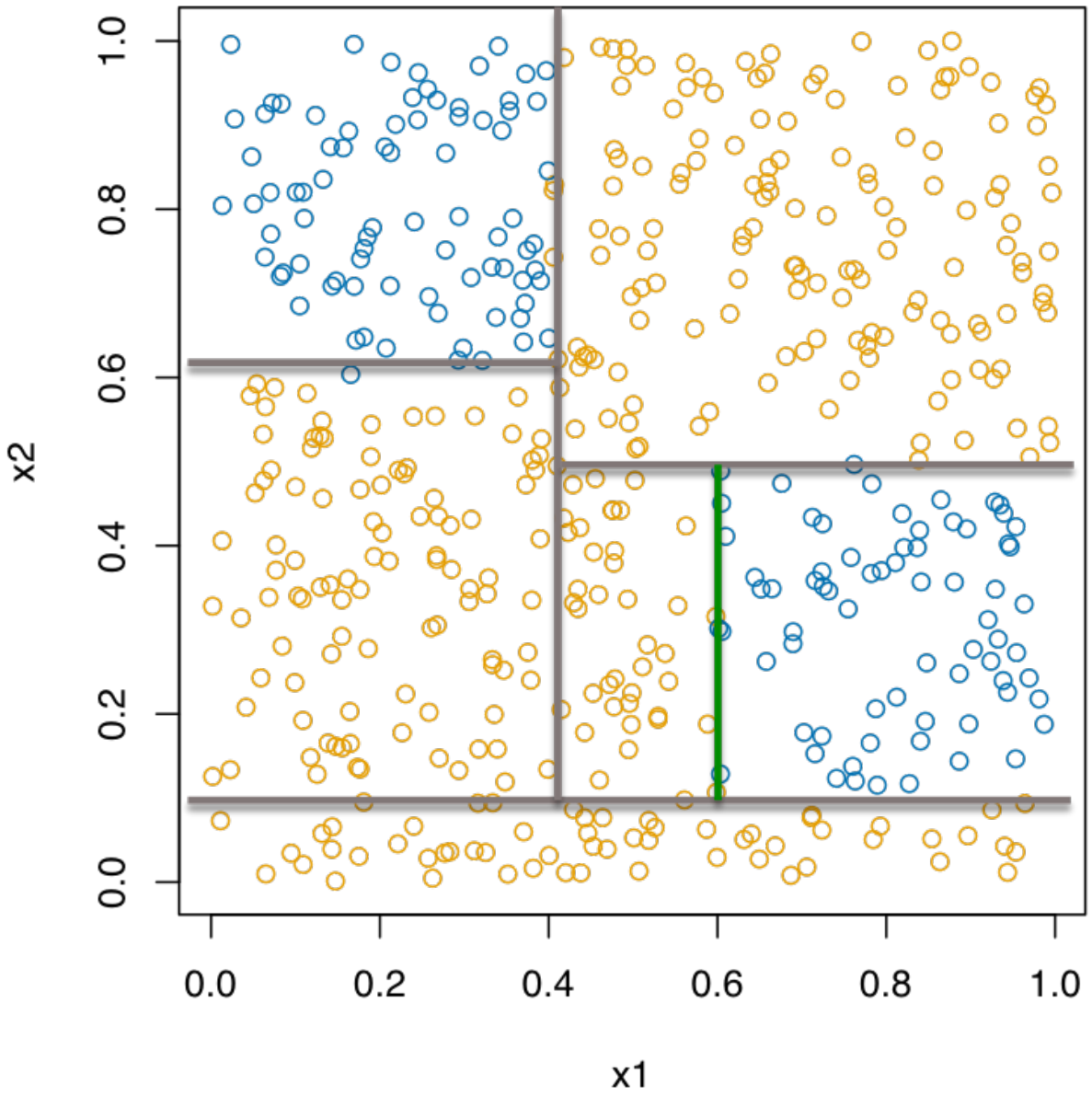
Recursive binary partitioning



Recursive binary partitioning



Recursive binary partitioning



Recursive binary partitioning

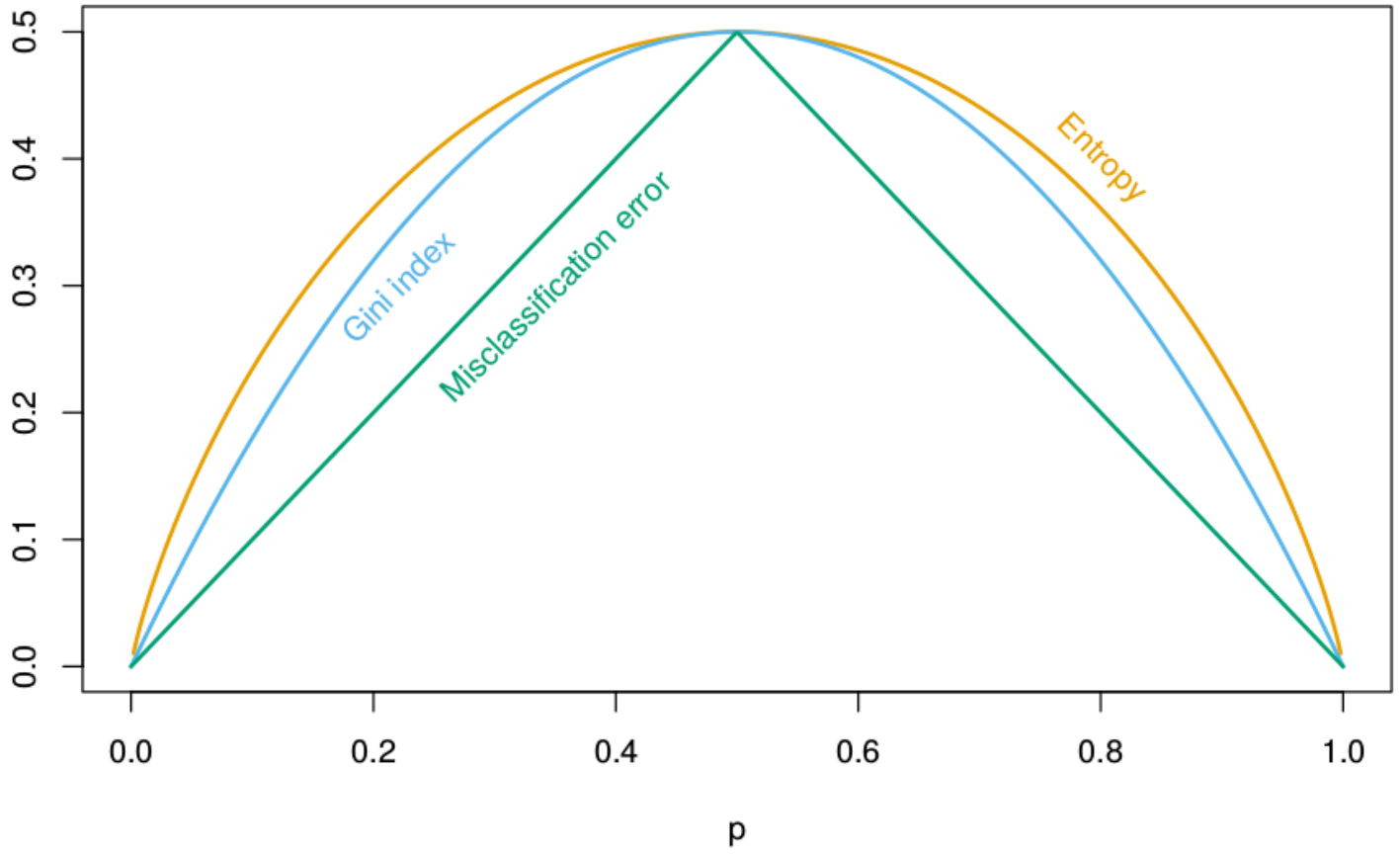
- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data
- In prediction, you choose splits to minimize the RSS
- In classification, choose splits to maximize node purity (minimize Gini index)

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

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

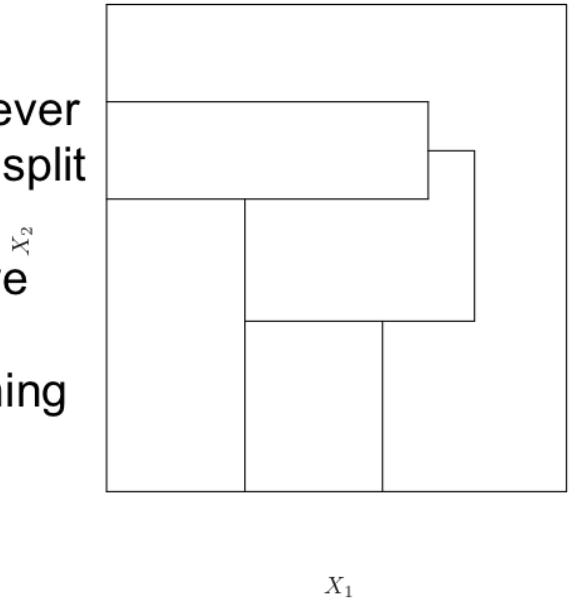
- G is small if all the $\hat{p}_{mk}$ are close to 0 or 1

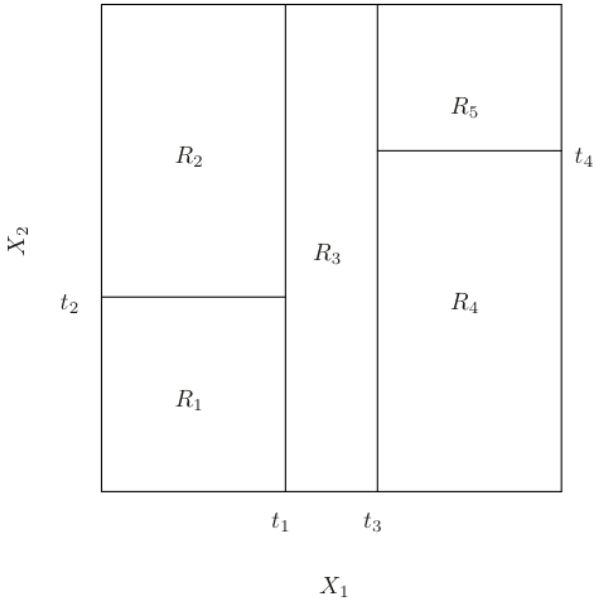
Why not minimize the misclassification error?

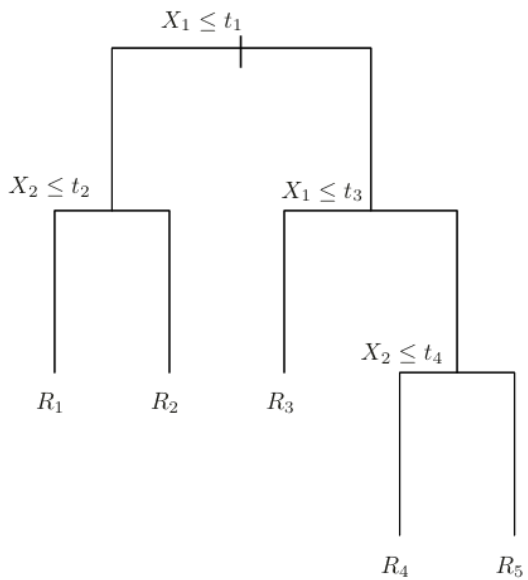


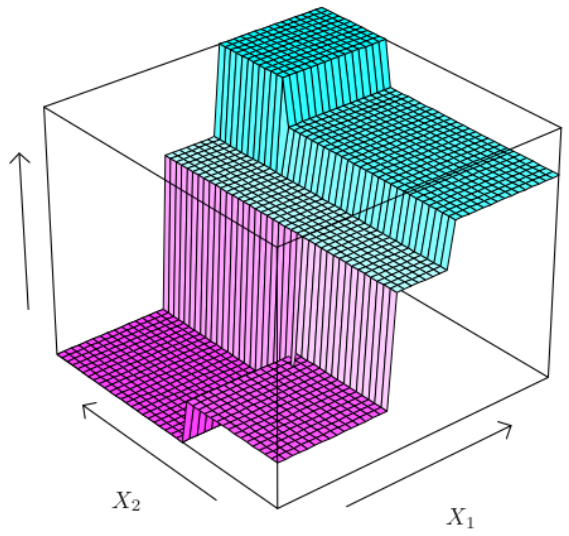
- Misclassification rate is poor at pushing for really pure nodes
- With Gini: going from $\hat{p}_{mk} = 0.8$ to $\hat{p}_{mk} = 0.9$ is better than going from $\hat{p}_{mk} = 0.5$ to $\hat{p}_{mk} = 0.6$
- With Misclassification error, these are considered equal improvements

You'll never get this split from recursive binary partitioning



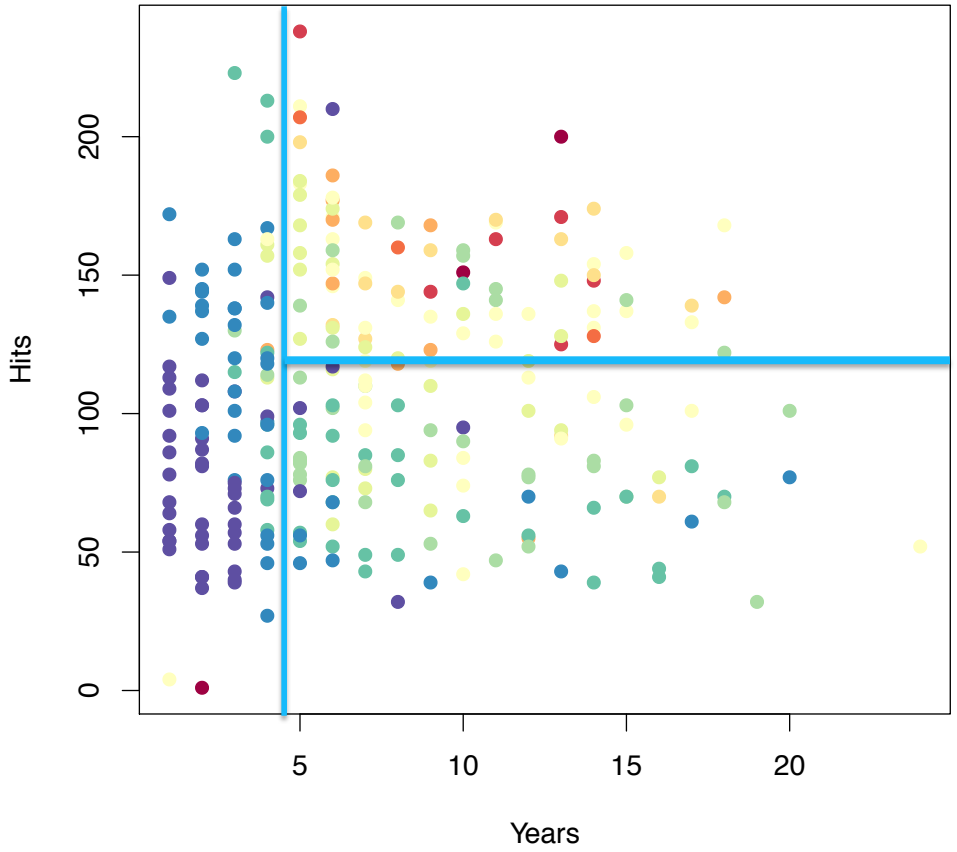






Tree pruning

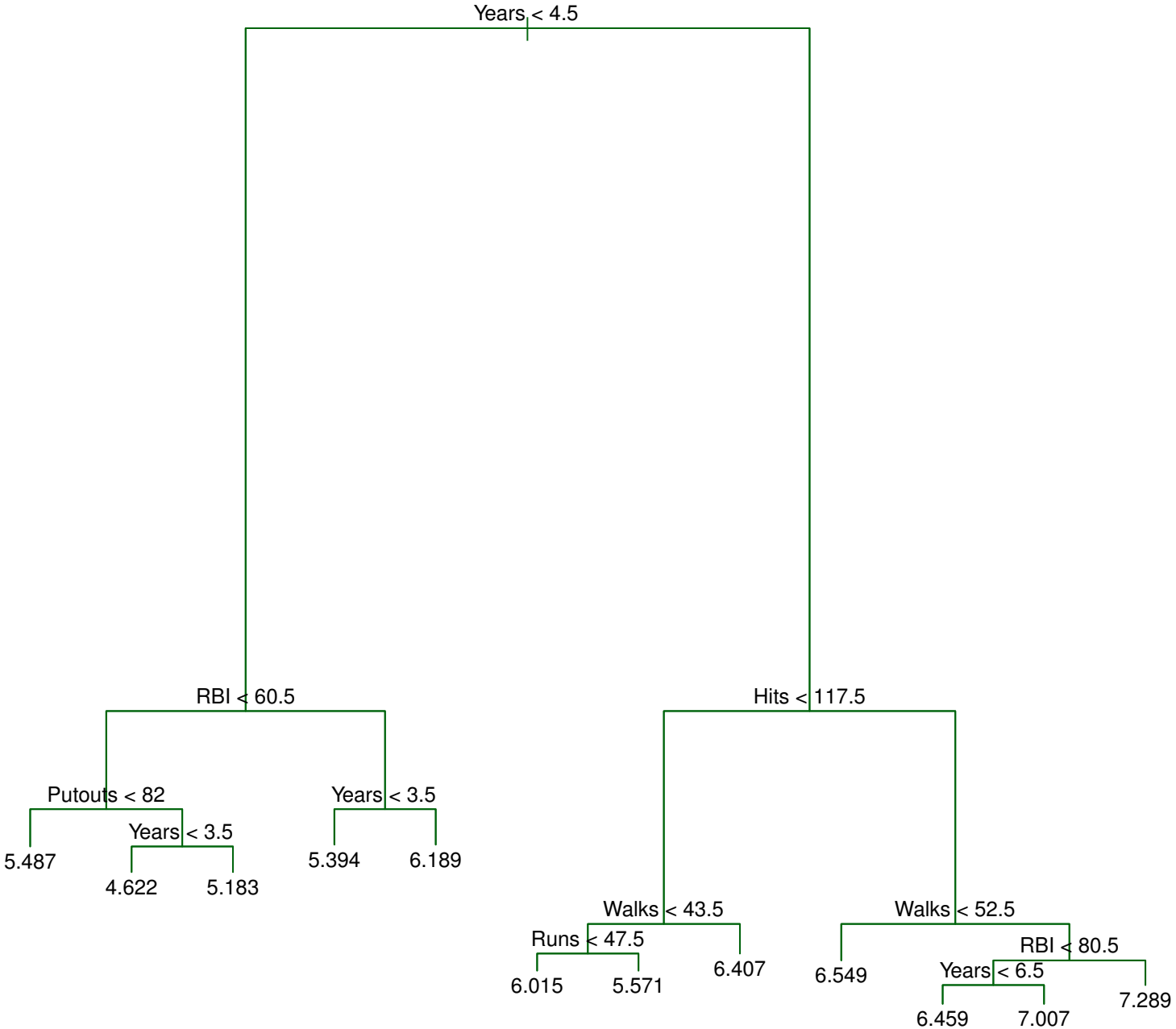
Why did we stop here? Why not keep partitioning?



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* 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

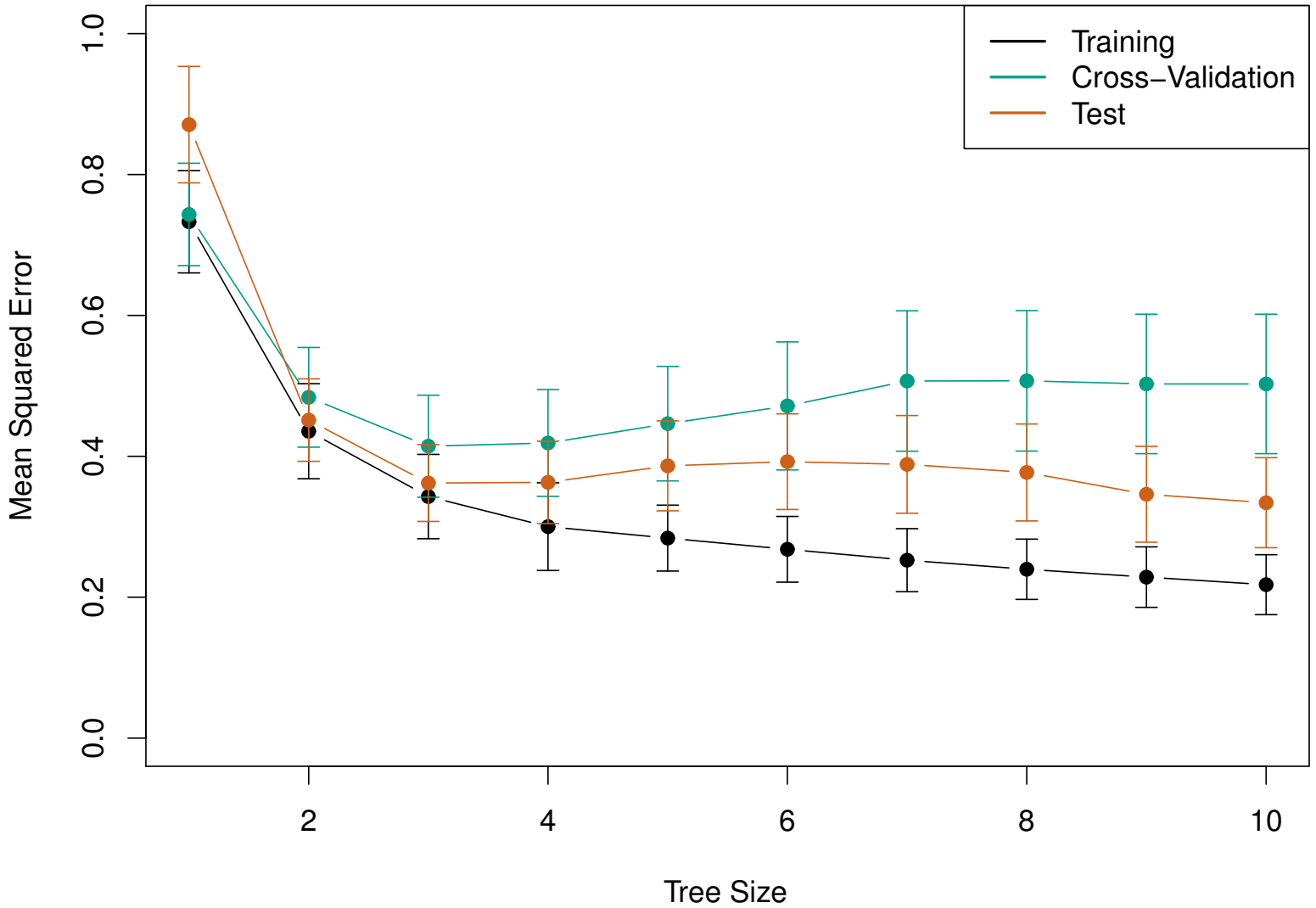
Pruning details

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.

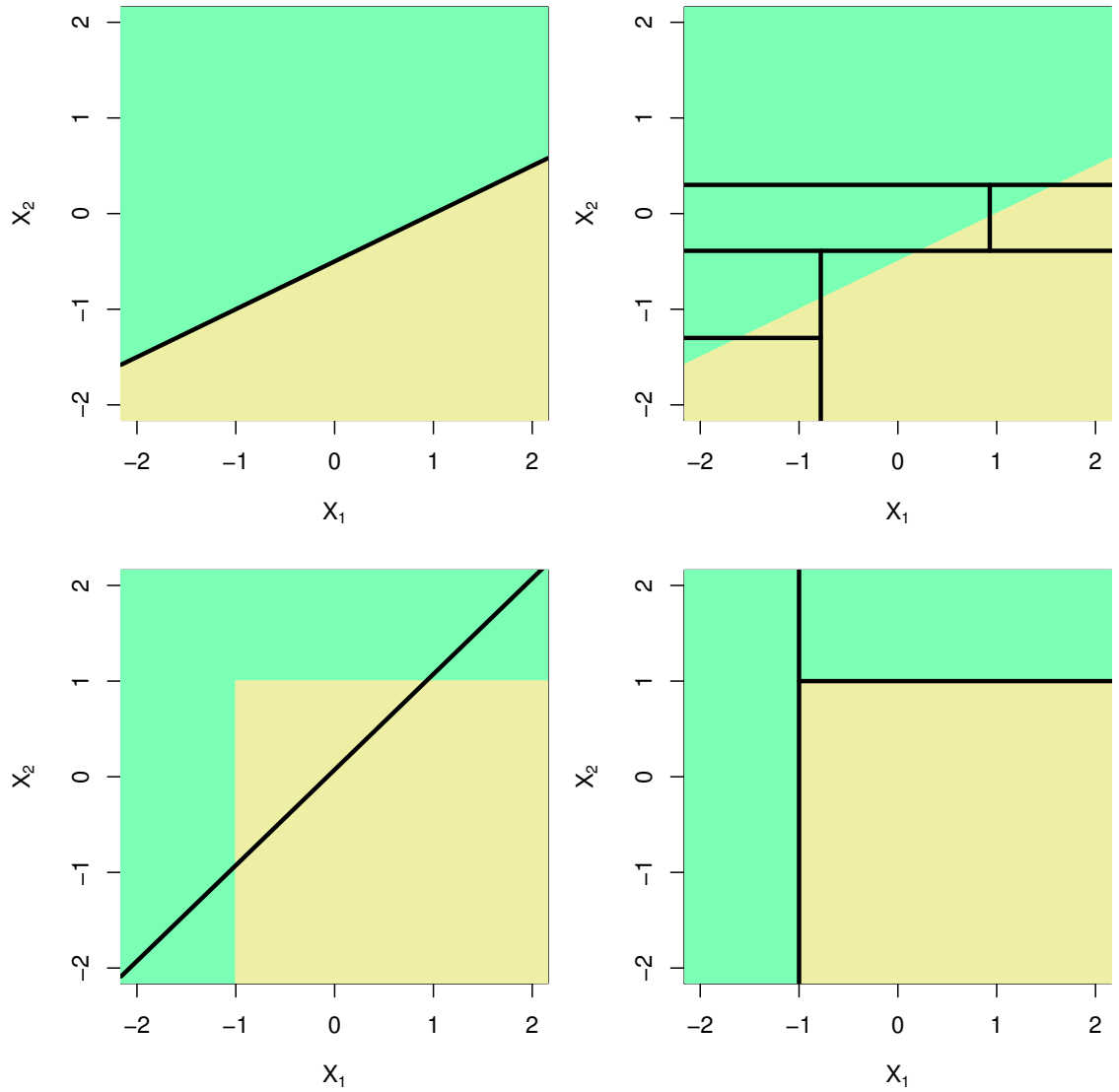
[source: ISL Chapter 8 slides]

Tree pruning



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

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.

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!
- **Coming soon:** Forests and boosted trees

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