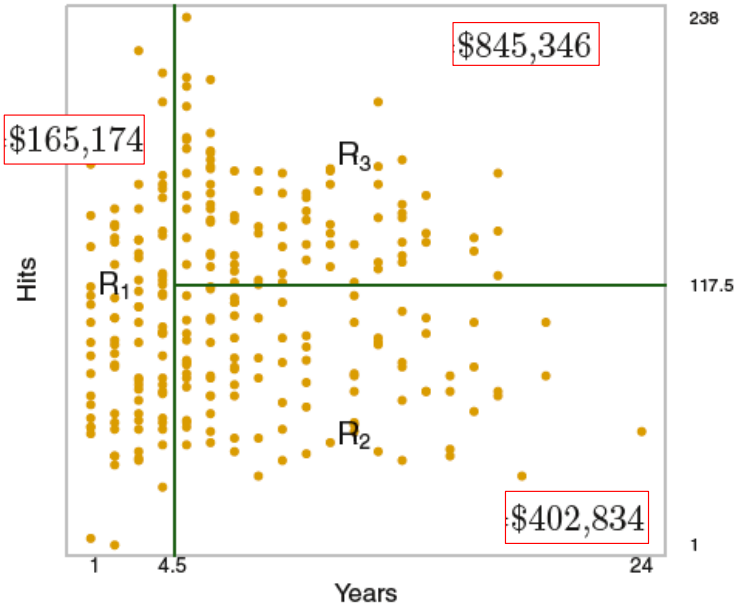
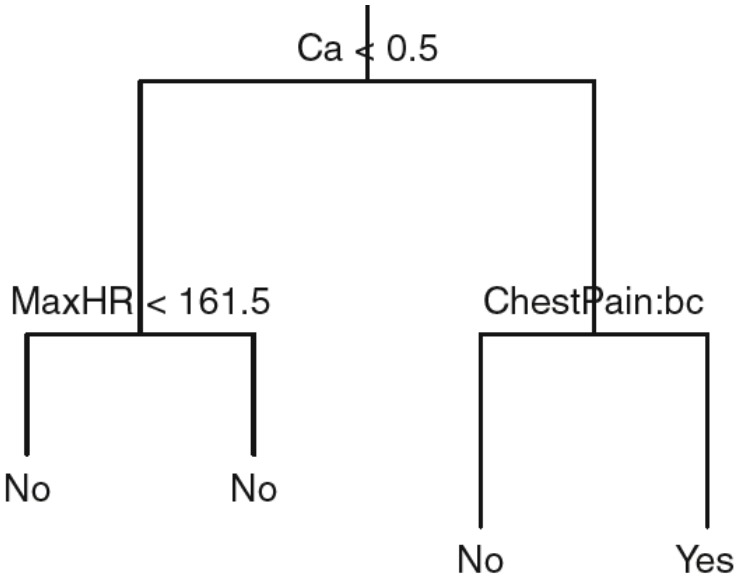
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2020f/

Decision Trees





Prof. Mike Hughes

Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Objectives for day 14
Decision Trees

- Decision Tree Regression
  - How to predict
  - How to train
    - Greedy recursive algorithm
  - Possible cost functions

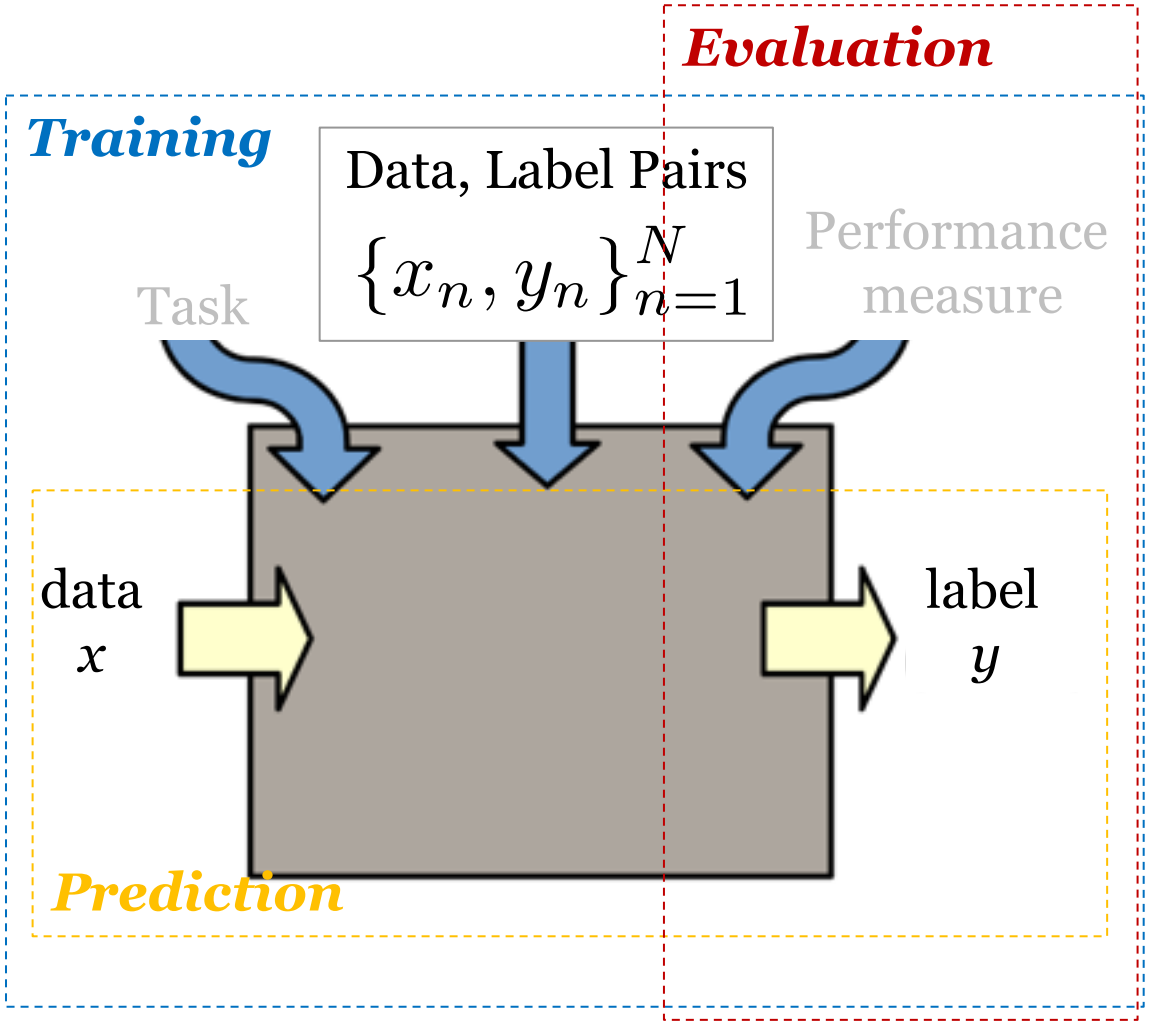
- Decision Tree Classification
  - Possible cost functions
  - Comparisons to other methods

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Task: Regression

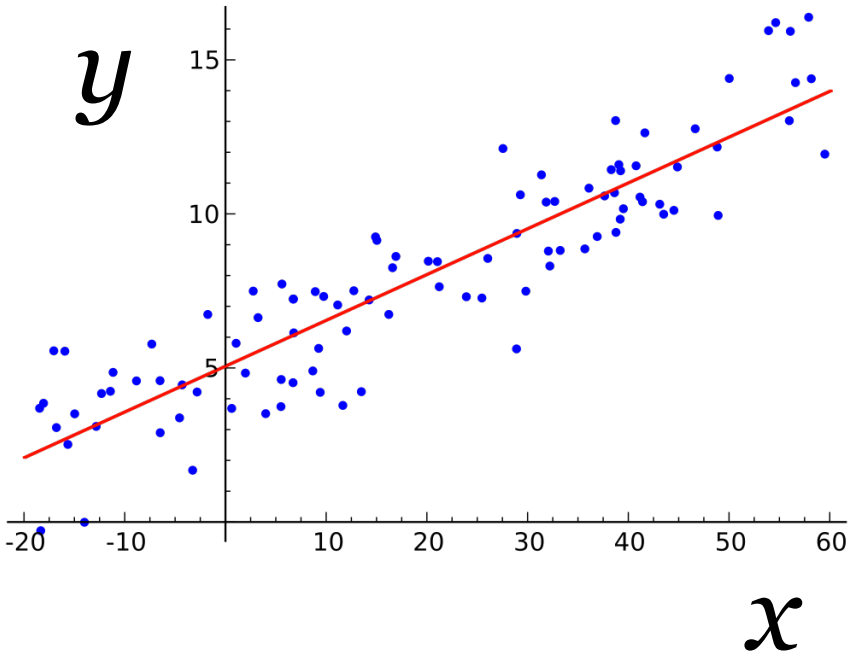
Supervised Learning

regression

Unsupervised Learning

Reinforcement Learning

y is a numeric variable
e.g. sales in $$



Salary prediction for Hitters data

A data frame with 322 observations of major league players on the following variables.

`AtBat`
: Number of times at bat in 1986

`Hits`
: Number of hits in 1986

`HmRun`
: Number of home runs in 1986

`Runs`
: Number of runs in 1986

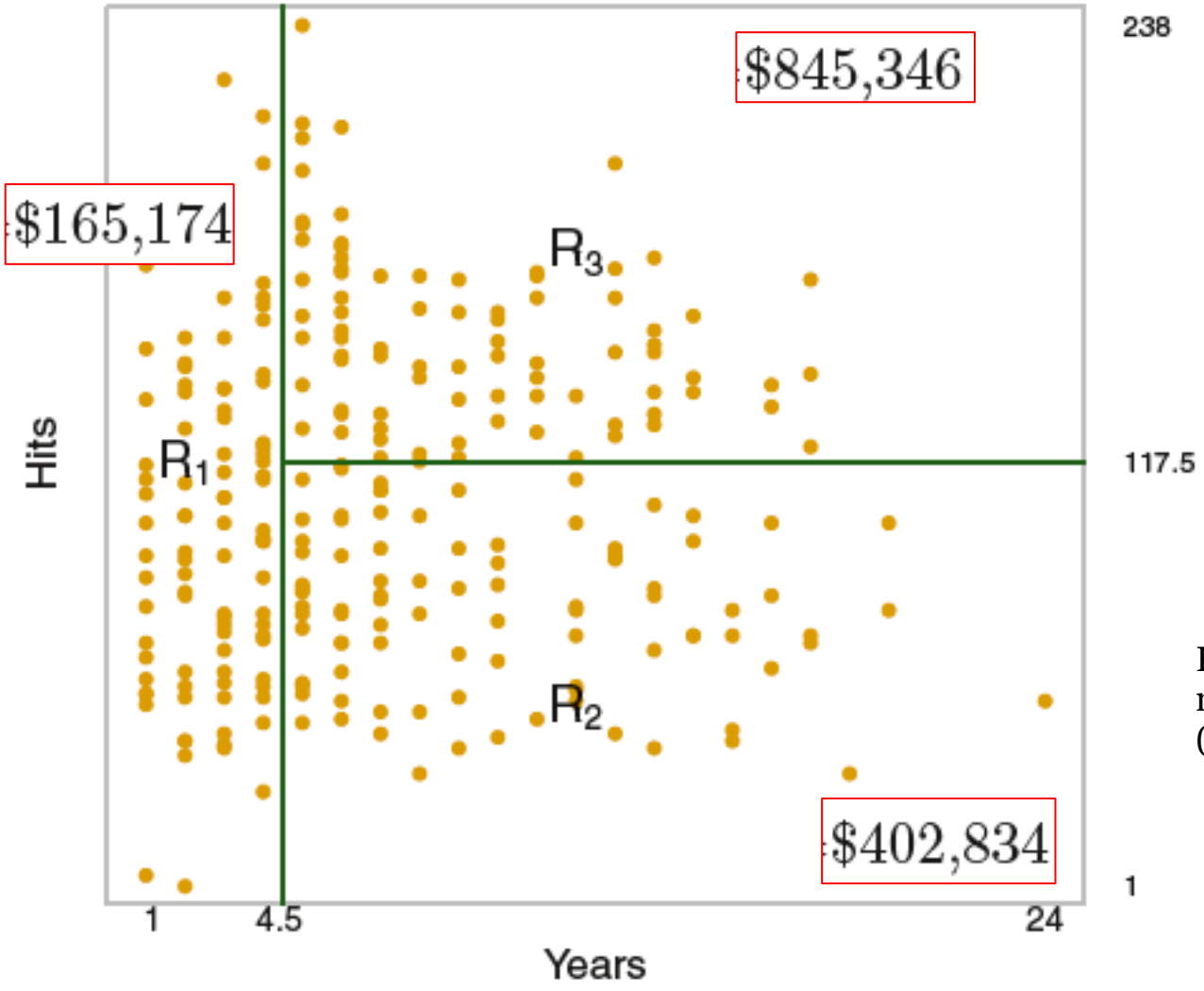
`RBI`
: Number of runs batted in in 1986

`Walks`
: Number of walks in 1986

`Years`
: Number of years in the major leagues

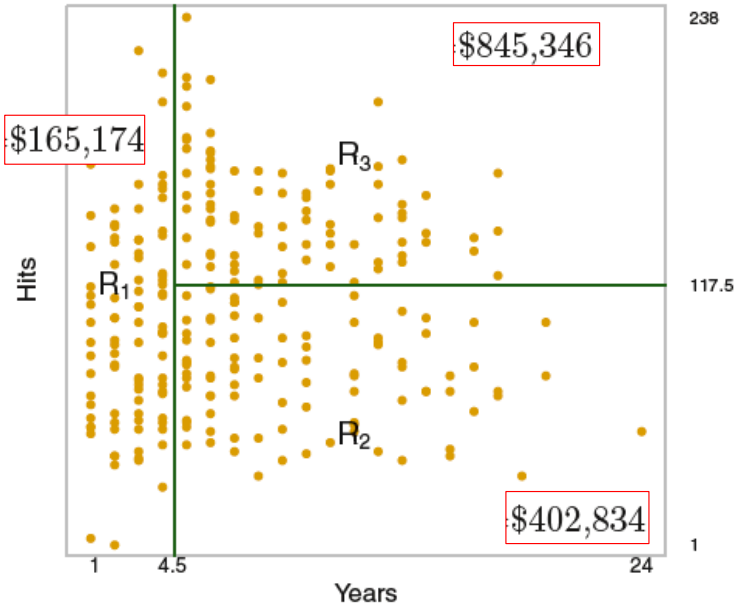
Salary Prediction by "Region"

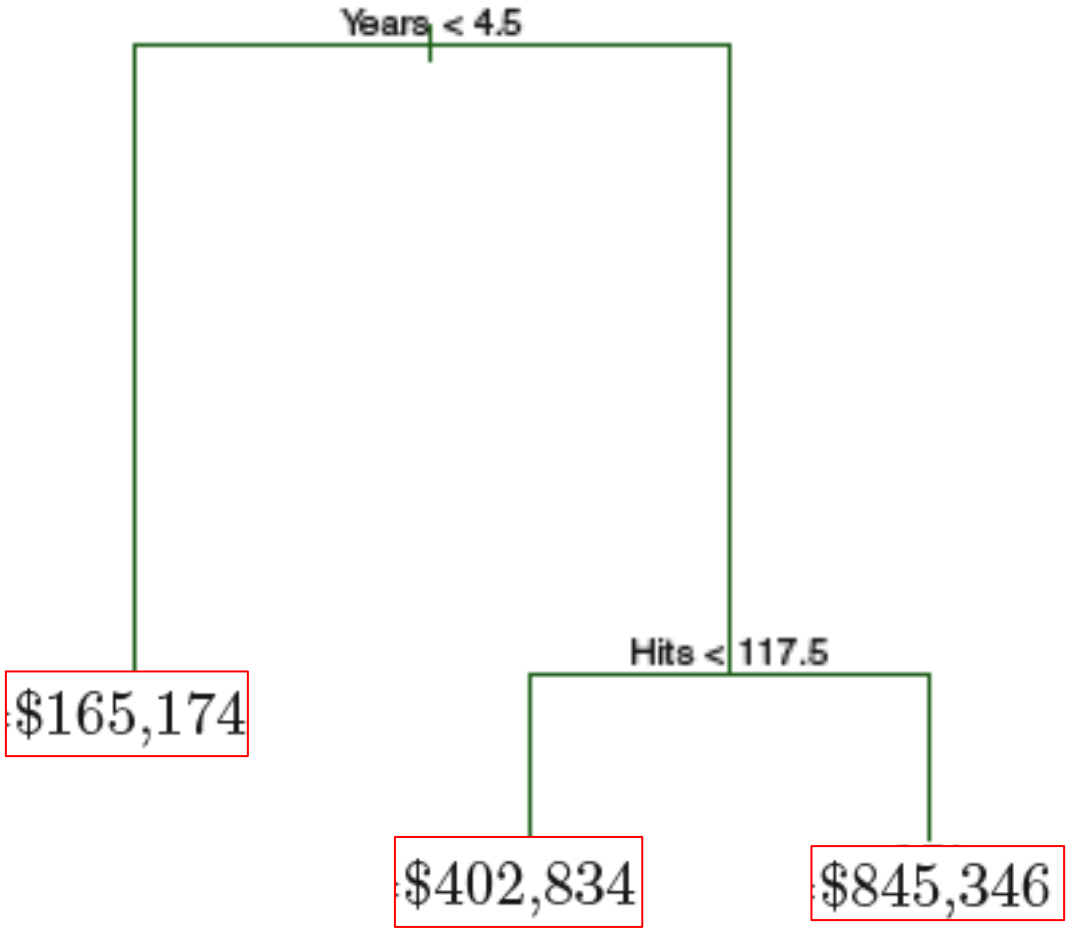
Divide x space into regions, predict constant within each region



Regions are rectangular ("axis aligned")

From Regions to Decision Tree





Decision tree regression

Classification and Regression Trees by Breiman et al (1984)

Parameters:

- tree architecture (list of nodes, list of parent-child pairs)
- *at each internal node: x* variable id and threshold value
- *at each leaf:* scalar *y* value to predict

Hyperparameters

- max_depth, min_samples_split

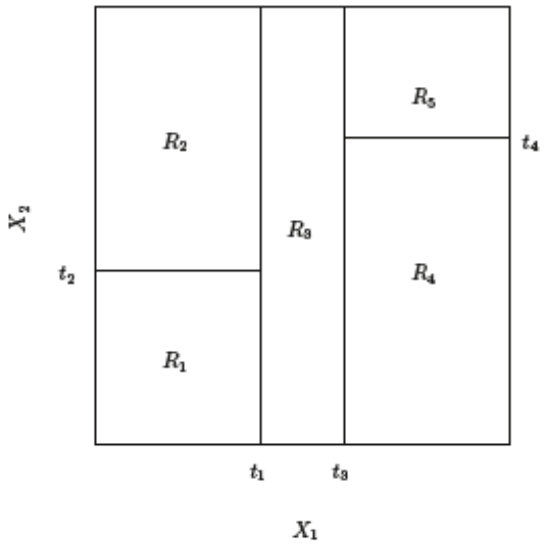
Prediction procedure:

- Determine which leaf (region) the input features belong to
- Guess the *y* value associated with that leaf

Training procedure:

- minimize error on training set
- use greedy heuristics (build from root to leaf)

Ideal Training for Decision Tree



$$\min_{R_1, \ldots R_J} \sum_{j=1} \sum_{n: x_n \in R_j} (y_n - \hat{y}_{R_j})^2$$

Search space is too big (so many regions)! Hard to solve exactly…
… let's break it down into subproblems

Key subproblem: Within 1 region, how to find best binary split?

Given a big region R, find the best possible binary split into two subregions (best means minimize **mean squared error)**

$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\}$$

$$\min_{j,s,\hat{y}_{R_1},\hat{y}_{R_2}} \sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

We can solve this subproblem efficiently!

For each feature index j in 1, 2, ... F:

- find its best possible cut point s[j] and its cost[j]

j <- argmin(cost[1] ... cost[F])

return best index j and its cut point s[j]

Let `binary_split` denote this procedure

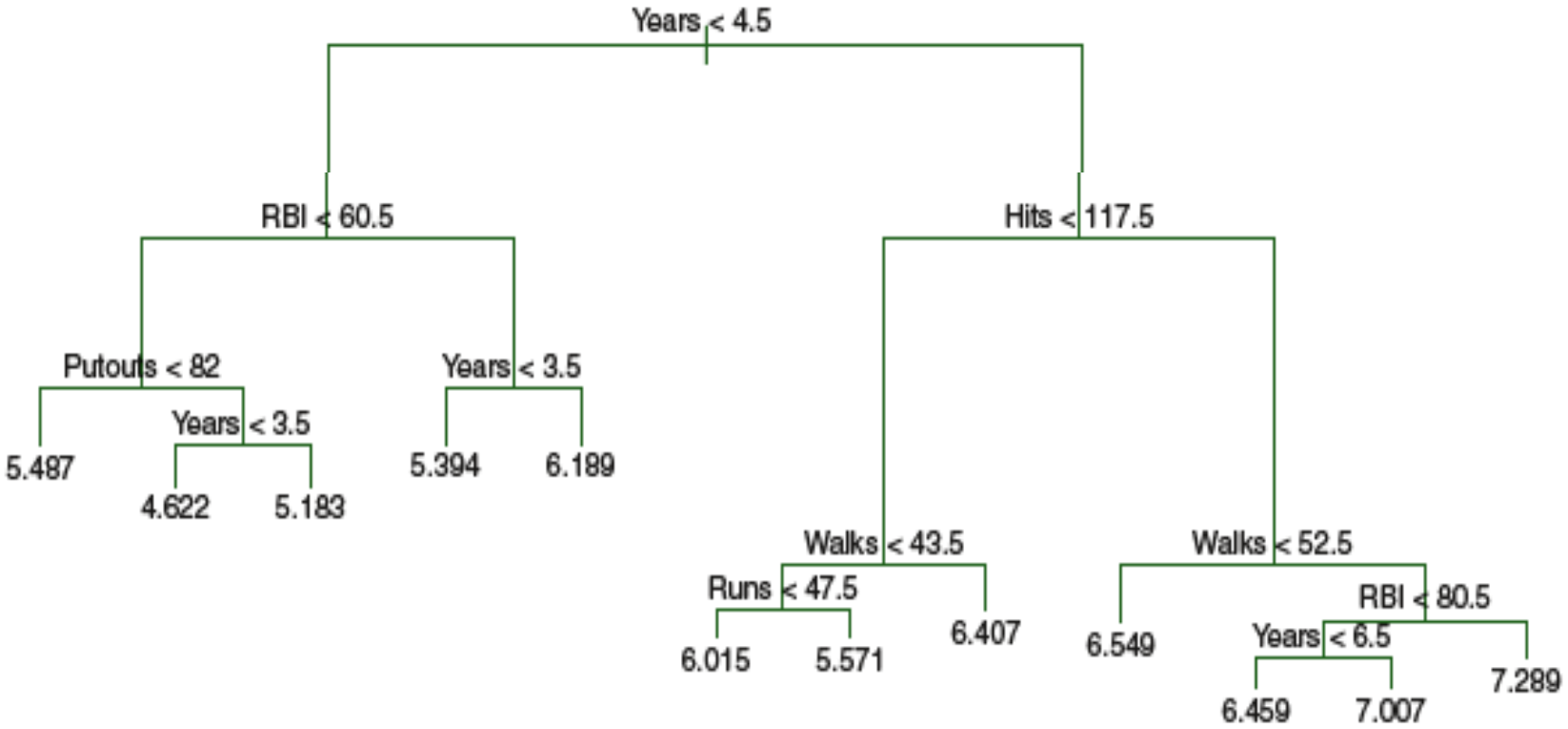
Greedy top-down training

Training is a **recursive** process.
Returns a *tree* (by reference to its root node)

Hyperparameters controling complexity
- MAX_DEPTH
- MIN_INTERNAL_NODE_SIZE
- MIN_LEAF_NODE_SIZE

```
def train_tree_greedy(x_NF, y_N, depth d):
    if d >= MAX_DEPTH:
        return LeafNode(x_NF, y_N)
    elif N < MIN_INTERNAL_NODE_SIZE:
        return LeafNode(x_NF, y_N)
    else:
        # j : integer index indicating feature to split
        # s : real value used as threshold to split
        # L / R : number of examples in left / right region
        j, s, x_LF, x_RF, y_L, y_R = binary_split(x_NF, y_N)
        if no split possible:
            return LeafNode(x_NF, y_N)
        left_child = train_tree_greedy (x_LF, y_L, d+1)
        right_child = train_tree_greedy(x_RF, y_R, d+1)
        return InternalNode(x_NF, y_N, j, s, left_child, right_child)
```

Greedy Tree for Hitters Data



Cost functions for *regression* trees

- **Mean squared error**
  - Assumed on previous slides, very common
  - How to solve for region's best guess?

$$\min_{\hat{y}_{R_1}} \sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2$$

Optimal solution:
Guess **mean** **output value** of the region:

$$\hat{y}_R = \frac{1}{|R|} \sum_{i: x_i \in R} y_i$$

- **Mean absolute error**
  - Possible! (supported in sklearn)
  - How to solve for region's best guess?

Optimal solution:
Guess **median** **output value** of the region:

$$\hat{y}_R = \text{median}(\{y_i : x_i \in R\})$$

Task: Binary Classification

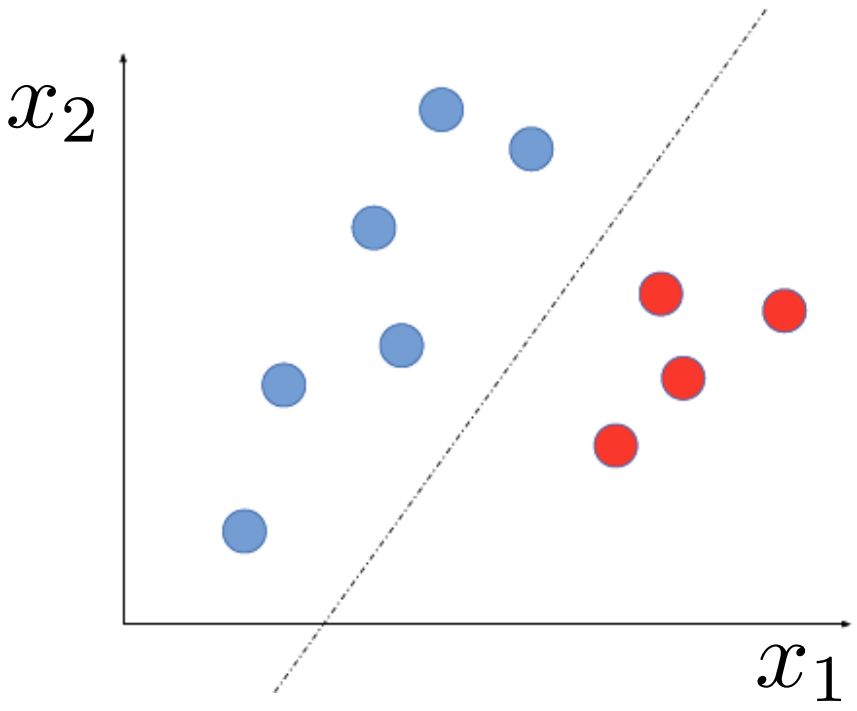
Supervised Learning

binary classification

Unsupervised Learning

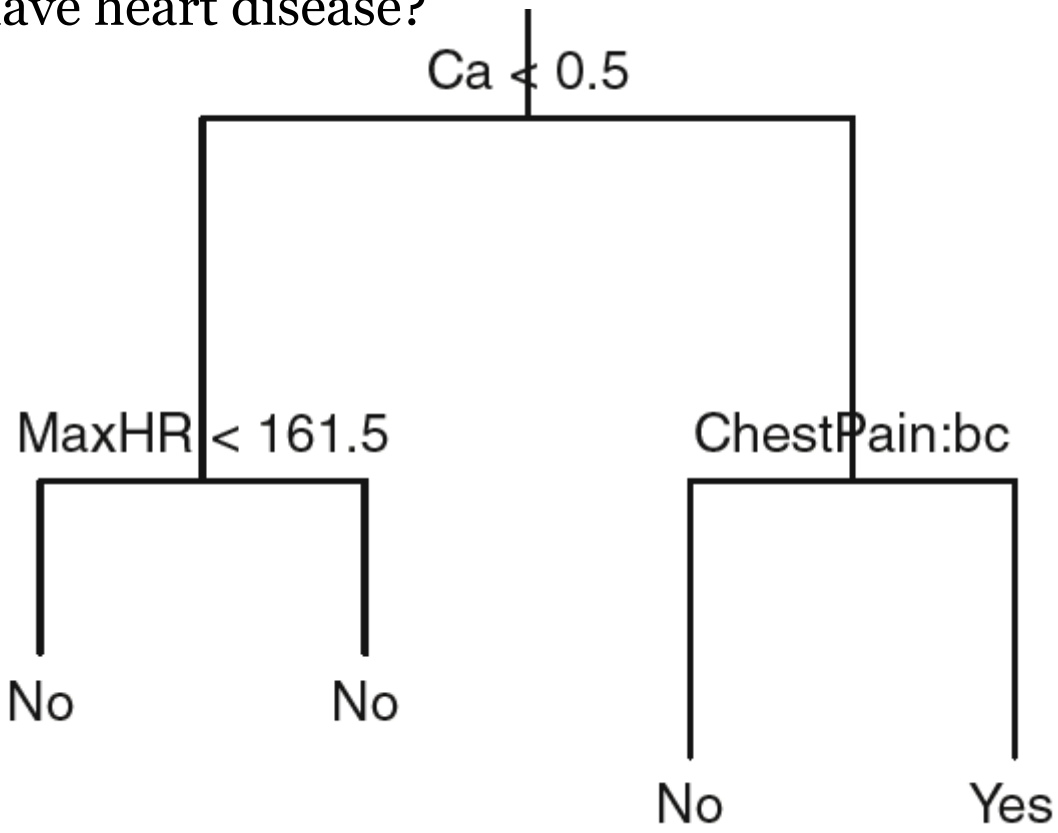
Reinforcement Learning

y is a binary variable (red or blue)



Decision Tree Classifier

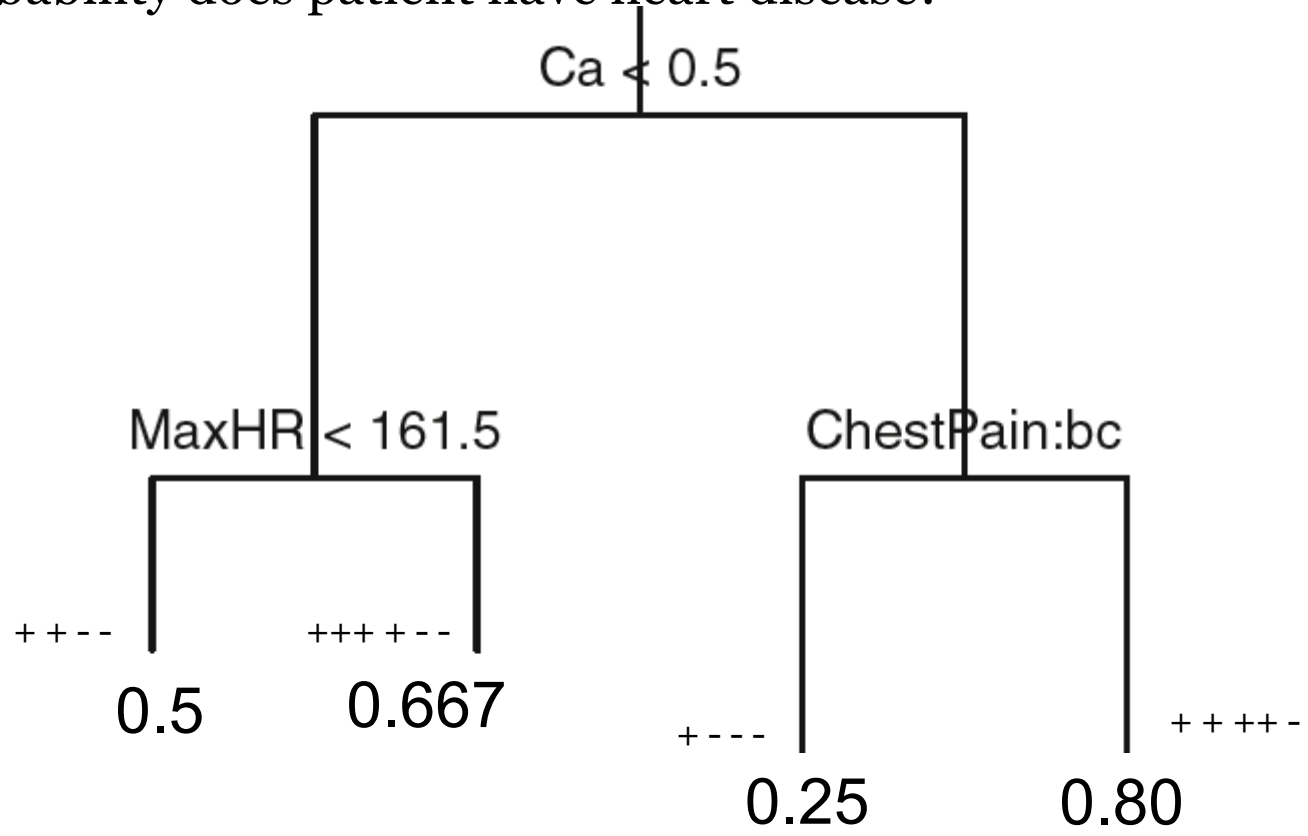
Goal: Does patient have heart disease?



Leaves make binary predictions!

Decision Tree Probabilistic Classifier

Goal: What probability does patient have heart disease?



Leaves count samples in each class! Then return fractions!

Decision Tree Classifier

Classification and Regression Trees by Breiman et al (1984)

Parameters:

- tree architecture (list of nodes, list of parent-child pairs)
- *at each internal node: x* variable id and threshold
- *at each leaf:* number of examples in each class

Hyperparameters:

- max_depth, min_samples_split

Prediction:

- identify rectangular region for input x
- `predict`: **most common** label value in region
- `predict_proba`: **fraction** of each label in region

Training:

- minimize **cost** on training set
- greedy construction from root to leaf

Cost functions for *classification* trees

criterion : *{"gini", "entropy"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

- Information gain or "entropy"
  - Cost for a region with N examples of C classes

$$\text{cost}(x_{1:N}, y_{1:N}) = -\sum_{c=1}^{C} p_c \log p_c, \qquad p_c \triangleq \frac{1}{N} \sum_{n} \delta_c(y_n)$$

- Gini impurity

$$\text{cost}(x_{1:N}, y_{1:N}) = \sum_{c=1}^{C} p_c(1 - p_c)$$

Advantages of Decision 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.

+ Can handle heterogeneous datasets (some features are numerical, some are categorical) easily without requiring standardized scales like penalized linear models do

+ Flexible non-linear decision boundaries

+ Relatively few hyperparameters to select

Limitations of Decision Trees

- 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.

Axis-aligned assumption *not always a good idea*

- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

Summary of Classifiers

	Knobs to tune	Function class flexibility	Interpret?
Logistic Regression	**L2/L1 penalty on weights**	**Linear**	**Inspect weights**
MLPClassifier	L2/L1 penalty on weights Num layers, num units Activation functions GD method: SGD or LBFGS or … Step size, batch size	**Universal (with enough units)**	Challenging
K Nearest Neighbors Classifier	**Number of Neighbors** **Distance metric**	**Piecewise constant**	**Inspect neighbors**
Decision Tree Classifier	**Max. depth** **Min. leaf size**	**Axis-aligned Piecewise constant**	**Inspect tree**