Basics of Decision Trees

DS 301

Iowa State University

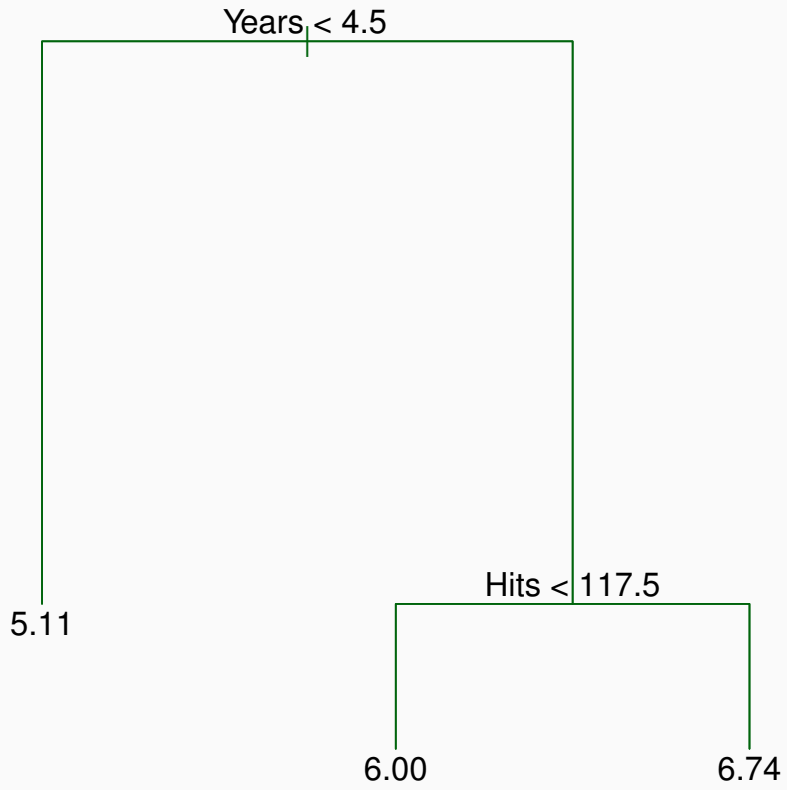
Basics of Decision Trees

Tree-based methods

- Tree-based methods are supervised learning methods that can be used for regression and classification.
- These methods involve stratifying or segmenting the predictor space into a number of simple regions.
- Tree-based methods are simple and useful for interpretation.
- However, they typically are not competitive with the best supervised learning approaches.
- Instead, we use methods that involve combining **multiple trees**: bagging, random forests, and boosting.

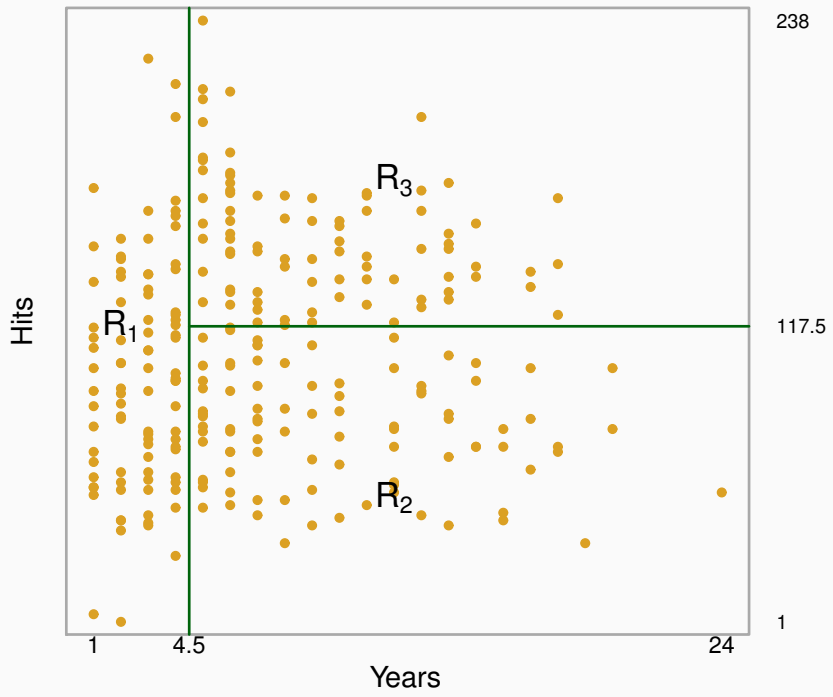
Regression Trees

Example: `Hitters` dataset. We want to predict a baseball player's `Salary` based on `Years` and `Hits`. `Salary` here is log-transformed so it is more bell-shaped.



Regression Trees

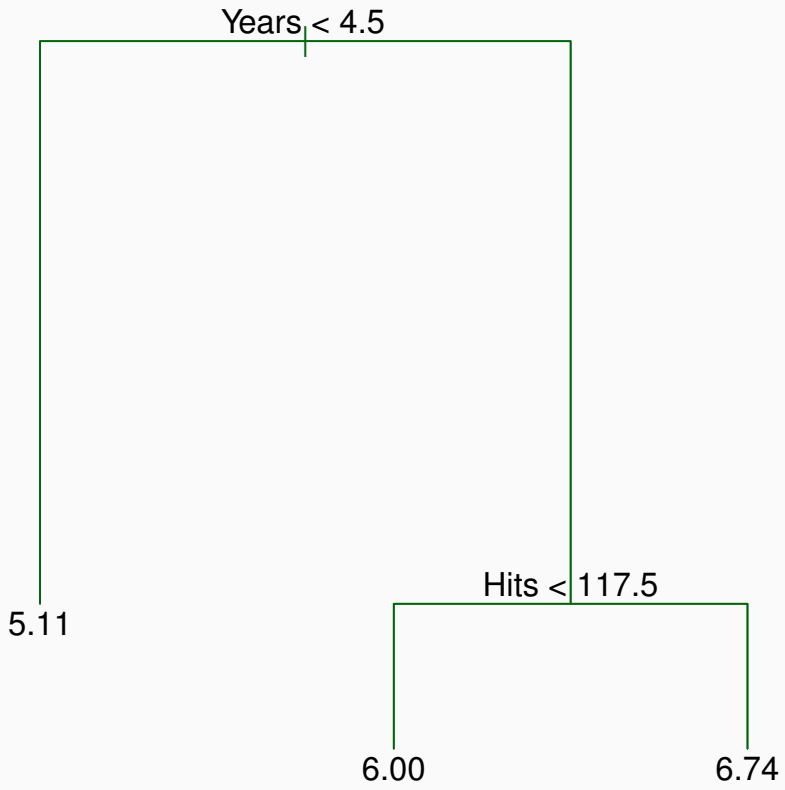
The tree segments (or stratifies) players into 3 regions of the predictor space:



Regression Trees

- Regions R_1, R_2, R_3 are known as **terminal nodes** or **leaves** of the tree.
- Points along the tree where the predictor space is split are referred to as **internal nodes**.
- Segments of the trees that connect the nodes are known as **branches**.

Interpretation of Regression Trees



Prediction via Stratification of the Feature Space

Regression tree building process:

1. We divide the predictor space (the set of possible values for $X_1, X_2, X_3, \ldots, X_p$), 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.

How do we construct the regions $R_1, \ldots, R_J$?

- In theory, regions could have any shape.
- However, we choose to divide the predictor space into **boxes** for simplicity and easy of interpretation.
- Then the goal is to find boxes $R_1, \ldots, R_J$ that minimize the (training) 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.

How do we construct the regions $R_1, \ldots, R_J$?

- Computationally infeasible to consider every partition of the feature space into J boxes.

Recursive binary splitting

- Top-down recursive greedy approach.
- Top-down recursive: start with all observations and split into two branches at each level of the tree.
- Greedy: best split is made at each step without looking ahead.

Recursive binary splitting

- Choose a predictor X_j and a cutpoint s that minimizes the training RSS for the resulting tree:

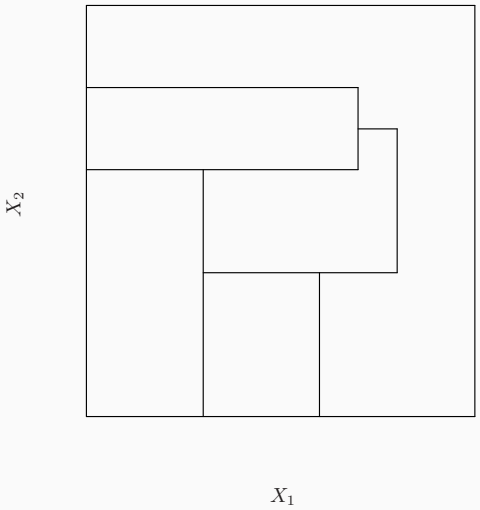
$$R_1(j,s) = \{X|X_j < s\}, \quad R_2(j,s) = \{X|X_j \geq s\}$$

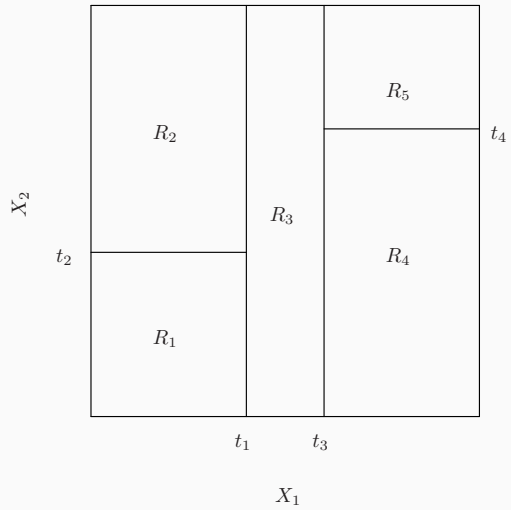
$$RSS = \sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

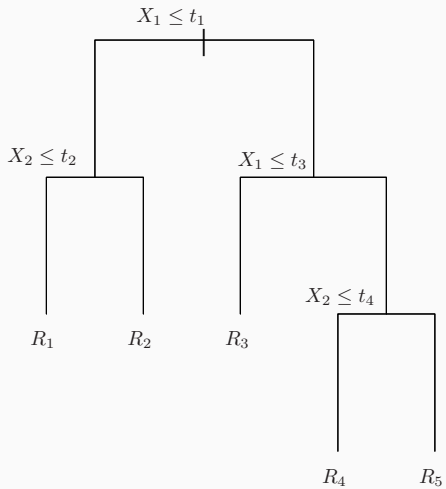
Recursive binary splitting

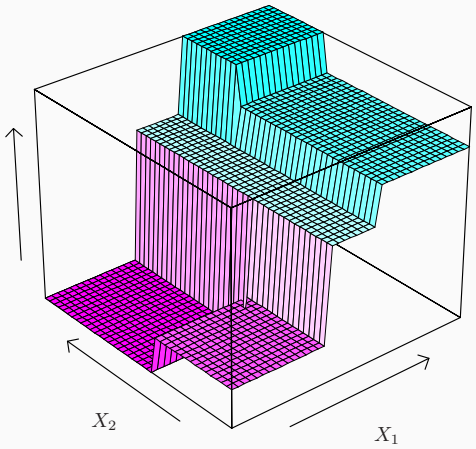
- Repeat the process: look for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each resulting region.
- Now instead of splitting entire predictor space, we split one of the previously identified regions.
- This process continues until a stopping criteria is reached (i.e. no regions contains more than 5 observations).

Recursive binary splitting









Tree Pruning

- Process described is a good start.
- But it is likely to overfit the data, leading to poor test set performance.
- This is because the resulting tree might be too complex.
- A smaller tree with fewer splits (that is, fewer regions $R_1, R_2, \ldots, R_J$) might lead to lower variance at the cost of a little bias.
- Our strategy: grow a very large tree T_0 and then prune it back in order to obtain a smaller tree (*subtree*).

Cost complexity pruning

- Idea: we apply a penalty to the tree.
- For every value of α, there is 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.
- $|T|$ denotes the number of leaves in the tree.
- α is a tuning parameter: controls the trade-off between the subtree's complexity and its fit to the training data.
- When $\alpha = 0$, we just have the original tree T_0.
- When α increases, what happens to our subtree?
- Reminiscent of lasso.

Algorithm for building a regression tree

1. Use recursive binary splitting to a grow a large tree on the training data, stopping according to a criteria.
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 $k = 1, \ldots, K$:
   a. Repeat Steps 1 and 2 on all but the kth fold of the training data.
   b. Evaluate the mean squared prediction error on the data in the left-out kth fold.

   Average the results for each value of α and pick α to minimize the k-fold CV error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

See R script: `decisiontreesbasics.R`