Chapter 8: Tree-Based Methods

nonparametric supervised methods.

quantitative y categorical y.

We will introduce *tree-based* methods for regression and classification.

These involve segmenting the predictor space into a number of simple regions.

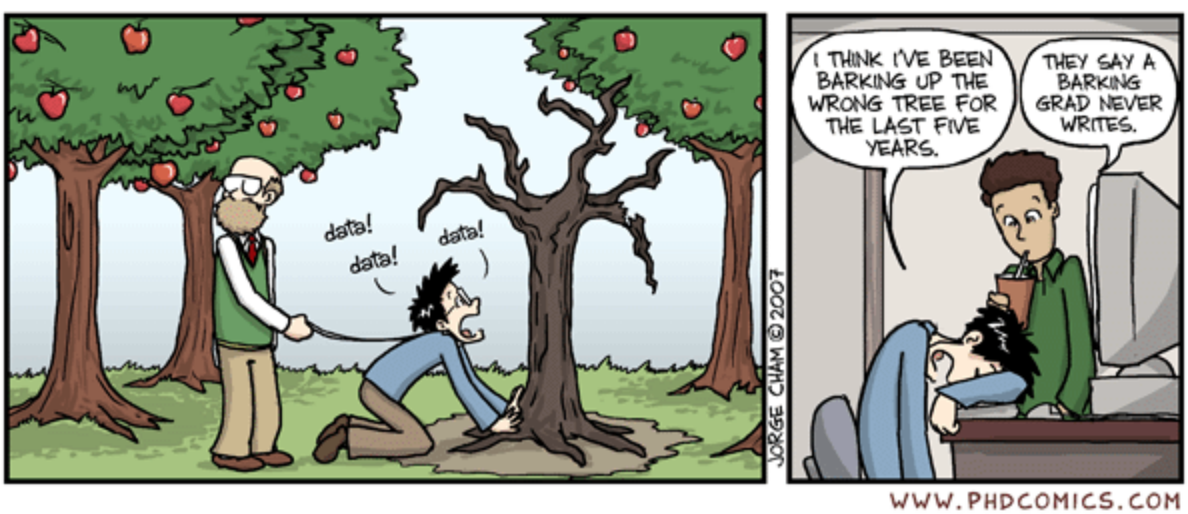
to make a prediction for an observation, we use mean or mode of training observations in the region to which it belongs.

The set of splitting rules can be summarized in a tree ⇒ "decision trees".

- simple and useful for interpreting.
- not competitive w/ other supervised approach (e.g. lasso) for prediction.

boosting, bagging, random forests

Combining a large number of trees can often result in dramatic improvements in prediction accuracy at the expense of interpretation.



Credit: http://phdcomics.com/comics.php?f=852

Decision trees can be applied to both regression and classification problems. We will start with regression.

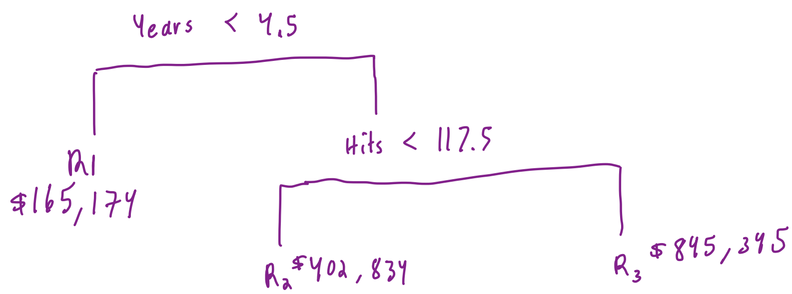
1 Regression Trees

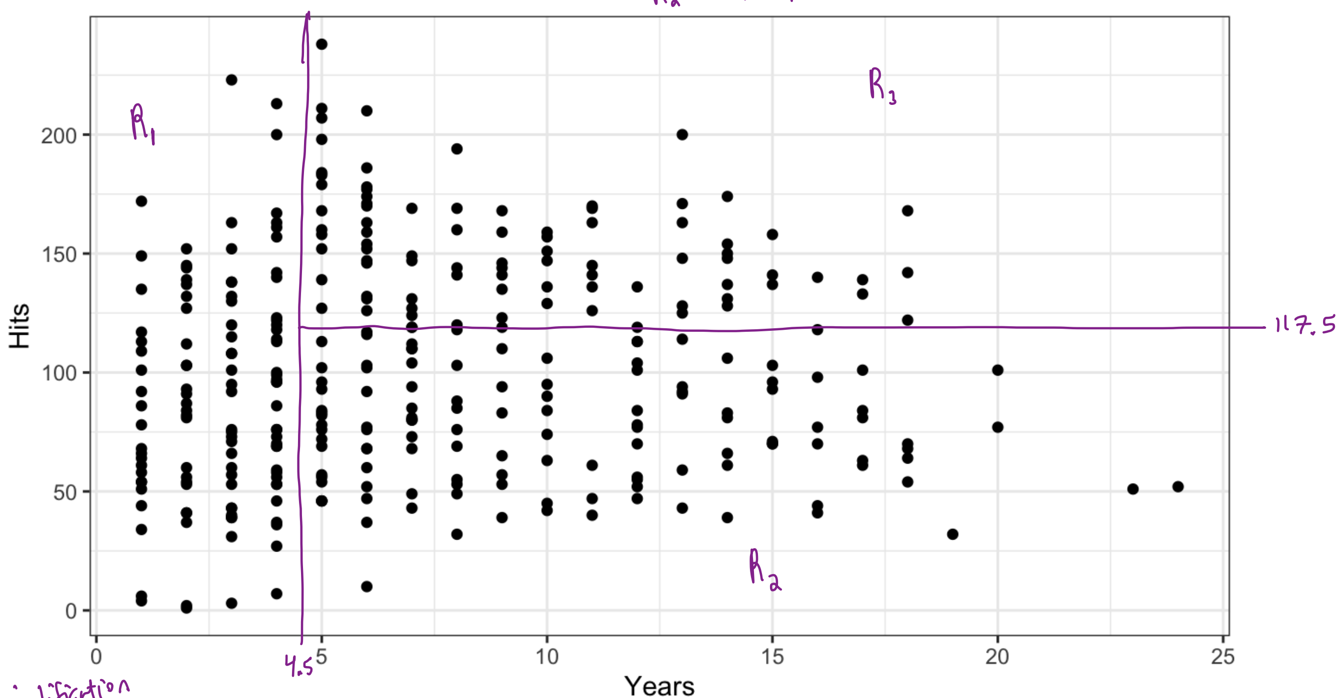
start w/

Example: We want to predict baseball salaries using the `Hittters` data set based on `Years` (the number of years that a player has been in the major leagues) and `Hits` (the number of hits he made the previous year).

→ more details on how later.

We can make a series of splitting rules (according a tree fit to this data) to create regions and predict salary as the mean of training obs. in each region.





probably an oversimplification but is easy to interpret and has nice graphical representation.

The predicted salary for players is given by the mean response value for the players in that box. Overall, the tree segments the players into 3 regions of predictor space.

terminology: R_1, R_2, R_3 = terminal nodes or leaves of the tree.

points along tree where predictor space is split = internal nodes

segments of tree that connect node = branches

interpretation: Years is the most important factor in determining salary

↳ given that a player has less experience, # hit in previous year plays little role in his salary.

↳ among players who have been in the league 5+ years, # hits does affect salary: ↑ hits, ↑ salary.

quantitative Y

We now discuss the process of building a regression tree. There are t~~o~~wo steps:

set of possible values for $X_1, \dots, X_p$

1. Divide <u>predictor space</u>

   into J distinct and non-overlapping regions $R_1, \dots, R_J$

2. Predict

   For every observation that fall into region R_j we make the same prediction — the mean of the response Y for training values in R_j.

How do we construct the regions $R_1, \dots, R_J$? How to divide predictor space?

Regions could be any shape, but that is too hard (to do & to interpret)

$\Rightarrow$ divide predictor space into high dimensional rectangles (boxes).

The goal is to find boxes $R_1, \dots, R_J$ that minimize the RSS. $= \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$ where $\hat{y}_{R_j}$ = mean response of training data in R_jth box.

Unfortunately it is computationally infeasible to consider every possible partition.

$\Rightarrow$ take a <u>greedy</u>, <u>top-down</u> approach called <u>recursive binary splitting</u>.

The approach is *top-down* because

We start at top of the tree (where all observations are in a single region) and successively split the predictor space.

$\hookrightarrow$ each split is indicated via two new branches down the tree.

The approach is *greedy* because

at each step of the tree building process, the best split is made at <u>that particular step</u>.

not looking ahead to make a split that will lead to a better tree later.

In order to perform recursive binary splitting,

(1) Select the predictor and cutpoint s s.t. splitting the predictor space into regions $\{X : X_j < s\}$ and $\{X : X_j \geq s\}$ leads to the greatest possible reduction in RSS.

↳ We consider all possible $X_1, \ldots, X_p$ and all possible cutpoints s then choose predictor and cutpoint w/ lowest RSS.

i.e. consider all possible half-planes $R_1(j,s) = \{X : X_j < s\}$ and $R_2(j,s) = \{X : X_j \geq s\}$ we seek j and s that minimize

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

← finding j and s can be quickly done if p is not too large.

(2) Repeat process, looking for next best j and s combo but instead of splitting entire space, we split $R_1(j,s)$ and $R_2(j,s)$ to minimize RSS.

(3) Continue until stopping criteria is met (i.e. no region contains more than 5 obs).

(4) predict using mean of training obs in the region to which the test falls.

The process described above may produce good predictions on the training set, but is likely to overfit the data.

because the resulting tree may be too complex

A smaller tree, with less splits (less regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

Idea: Only split a tree if it results in a "large enough" drop in RSS.

bad idea: because a seemingly worthless split early in the tree might be followed by a good split.

A strategy is to grow a very large tree T_0 and then *prune* it back to obtain a *subtree*.

Better idea

How to prune the tree?

goal: select a subtree that leads to lowest test error rate. → could use CV to estimate error for every subtree, but this is expensive.

solution: "cost complexity pruning", aka "weakest link pruning"

consider a sequence of subtrees indexed by a nonnegative tuning parameter α.

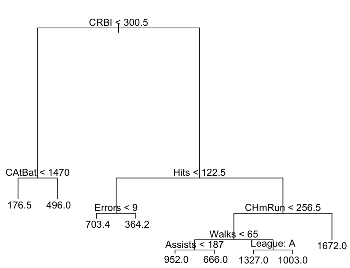
For each value of α, $\exists$ a corresponding subtree $T \subset T_0$ st.

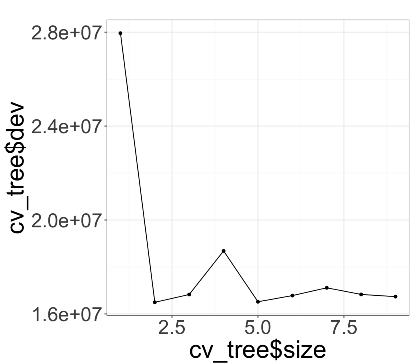
$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha \underbrace{|T|}_{\#\text{ terminal nodes in the tree.}}$$

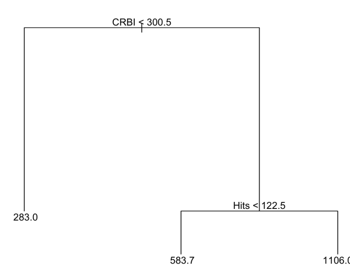
is as small as possible.

Select α by CV.

Algorithm for building a regression tree:







2 Classification Trees

A *classification tree* is very similar to a regression tree, except that it is used to predict a categorical response.

For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observation in the region to which it belongs.

The task of growing a classification tree is quite similar to the task of growing a regression tree.

It turns out that classification error is not sensitive enough.

When building a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split.

3 Trees vs. Linear Models

Regression and classification trees have a very different feel from the more classical approaches for regression and classification.

Which method is better?

3.1 Advantages and Disadvantages of Trees

4 Bagging

Decision trees suffer from *high variance*.

Bootstrap aggregation or *bagging* is a general-purpose procedure for reducing the variance of a statistical learning method, particularly useful for trees.

So a natural way to reduce the variance is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.

Of course, this is not practical because we generally do not have access to multiple training sets.

While bagging can improve predictions for many regression methods, it's particularly useful for decision trees.

These trees are grown deep and not pruned.

How can bagging be extended to a classification problem?

4.1 Out-of-Bag Error

There is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation.

4.2 Interpretation

5 Random Forests

Random forests provide an improvement over bagged trees by a small tweak that decorrelates the trees.

As with bagged trees, we build a number of decision trees on bootstrapped training samples.

In other words, in building a random forest, at each split in the tree, the algorithm is not allowed to consider a majority of the predictors.

The main difference between bagging and random forests is the chouce of predictor subset size m.

6 Boosting

Boosting is another approach for improving the prediction results from a decision tree.

While bagging involves creating multiple copies of the original training data set using the bootstrap and fitting a separate decision tree on each copy,

Boosting does not involve boostrap sampling, instead each tree is fit on a modified version of the original data set.

Boosting has three tuning parameters:

1.

2.

3.