nonparametric supervised methods.

Chapter 8: Tree-Based 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

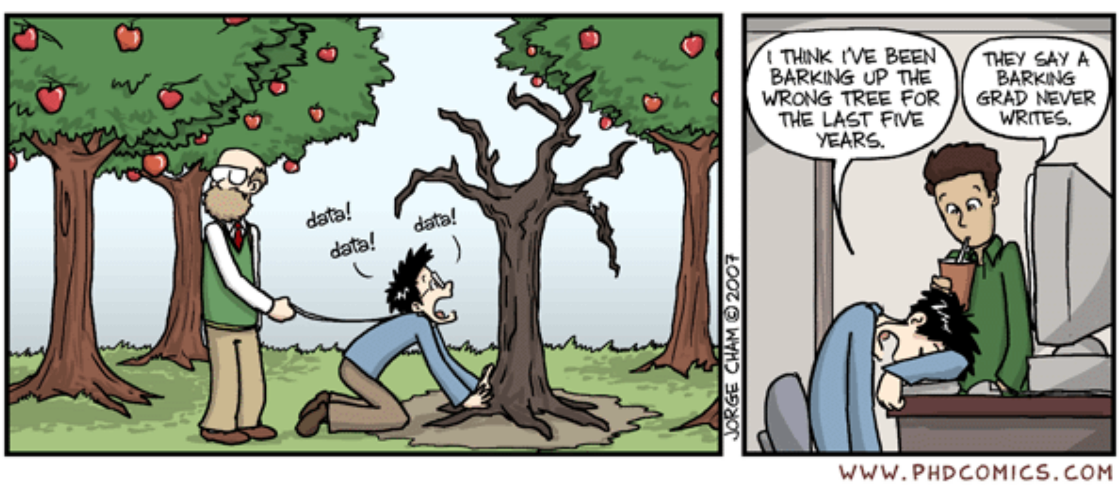
To make a prediction for an observation, we will use the mean or mode of the training observations in the regions to which it belongs.

The set of splitting rules can be summarized in a tree $\Rightarrow$ "decision trees".

- simple and useful for interpretation
- not competitive w/ other supervised approaches (eg. lasso) for prediction.

bagging, random forests, boosting.

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

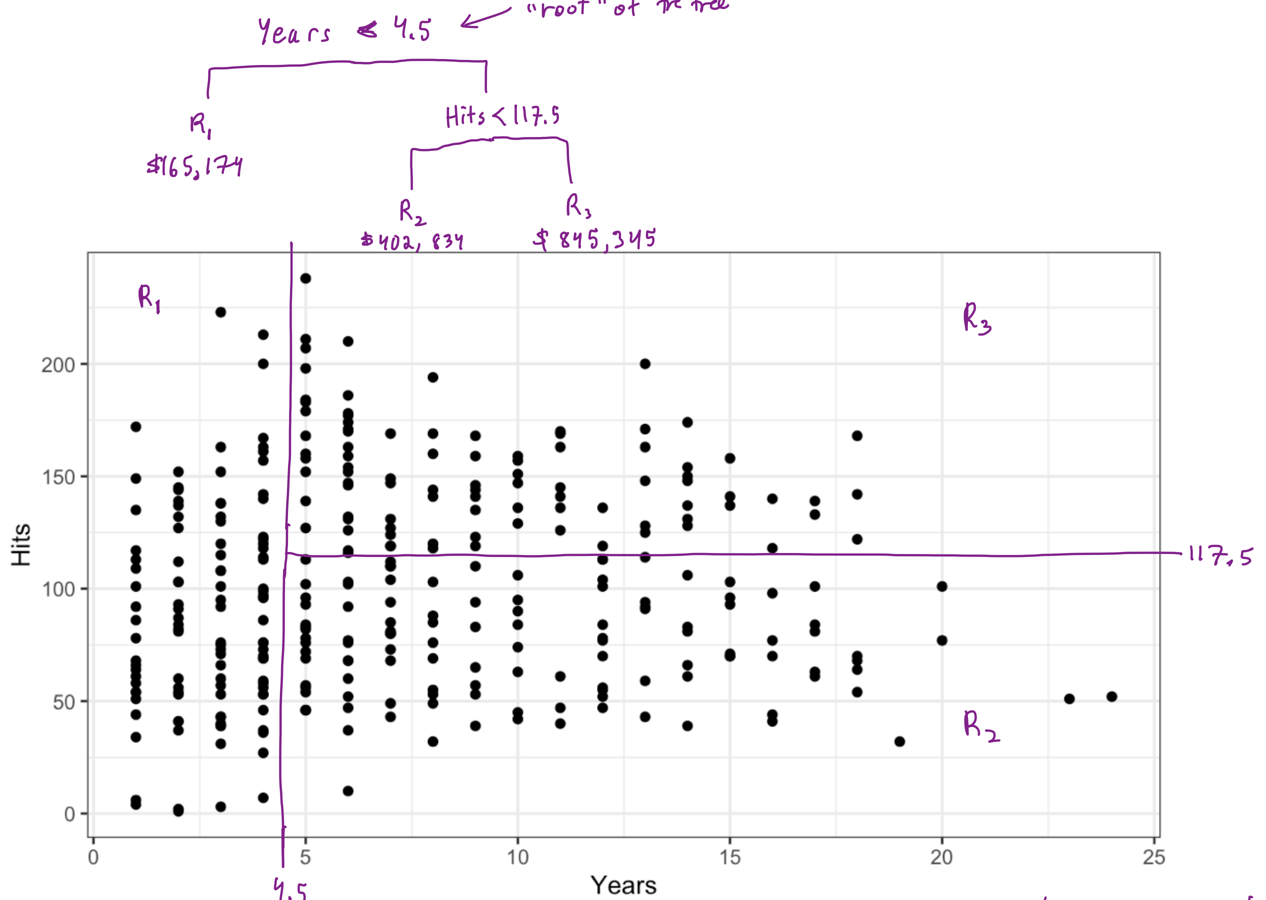
start with

Example: We want to predict baseball salaries using the `Hitters` 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).

We could make a series of splitting rules to create regions and predict salary as the mean in each region.

↓ according to a tree fit to this data (more on how in a minute).

"root" of the tree



probably/definitely an oversimplification but easy to interpret and has a 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 the tree where space is split = internal nodes

segments of tree that connect nodes = branches.

interpretation: Years is the most important factor in determining salary

↳ given that a player is experienced, # hits in previous year plays a role in his salary: ↑ hits, ↑ salary.

↳ given that a player is not experienced, # hits does not affect your salary.

quantitative y

We now discuss the process of building a regression tree. There are to steps:

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

1. Divide predictor space

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

2. Predict

   For every observation that falls 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 the predictor space?

regions could have any shape: but that is too hard (to do + interpret)

$\Rightarrow$ divide predictor space into high dimensional rectangles or 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 box R_j

Unfortunately it is computationally infeasible to consider every possible partition.

$\Rightarrow$ take top-down, greedy approach called recursive binary splitting

The approach is *top-down* because

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

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

The approach is *greedy* because

at each step of the building process, the best split is made at that particular step.

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

In order to perform recursive binary splitting,

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

region of predictor space where x_j takes values $< s$

↳ We consider all possible $X_1, \ldots, X_p$ and all possible cutpoints then choose predictor & cutpoint so tree has lowest RSS.

i.e. consider all possible half planes $R_1(j,s) = \{X \mid X_j < s\}$, $R_2(j,s) = \{X : X_j \geq s\}$.

we seek j and s that minimize $\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$ ← can be quickly done if p not too large.

(2) Repeat process looking for next best j, s combo, but instead of splitting entire space, we will 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 observations).

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

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

because our resulting tree will be too complex.

⇒ less regions.

A smaller tree, with less splits might lead to lower variance and better interpretation at the cost of a little bias.

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

bad idea because a seemingly worthless split early in tree can be followed by a good split!

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

How to prune the tree?

goal: select a subtree that leads to lowest test error rate → Could use CV to estimate error for every possible subtree, but this is expensive (large # of possible subtrees).

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

consider a sequence of trees indexed by a nonnegative tuning parameter α

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

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

$R_m = m^{th}$ terminal node region

$\hat{y}_{R_m} =$ predicted response for R_m

α controls tradeoff between subtree's complexity & fit to training data

when $\alpha = 0$ $T = T_0$

$\alpha \uparrow \Rightarrow$ price to pay for having more terminal nodes ↑ ⇒ smaller tree.

Select α via CV, then use full training dataset & chosen α to get subtree.

Algorithm for building a regression tree:

① Use recursive binary splitting to grow a large tree on training data set, stopping when each terminal node has fewer than some minimum # of observations

② Apply cost complexity pruning to the large tree to get a sequence of best trees as a function of α.

③ Use K-fold CV to choose α.

Divide training data into K folds, for each $k = 1, \ldots, K$

(a) Repeat steps ① and ② on all but k^{th} fold

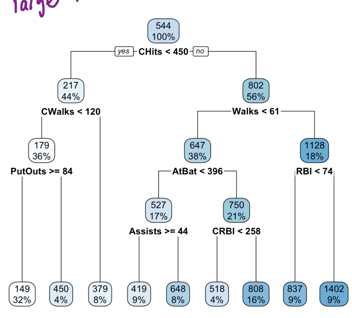
(b) Evaluate MSE on data in k^{th} fold

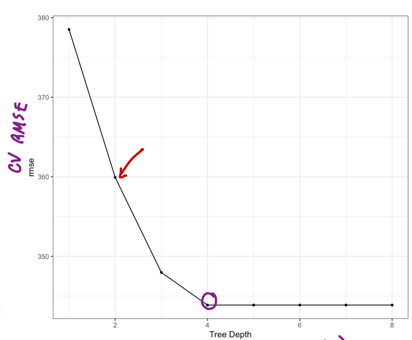
Average results for each value of α and pick α to minimize CV error.

④ Return the subtree from ② that corresponds to α from ③.

Example: Fit regression tree to Hitters using 9 features → 50% train/test split.

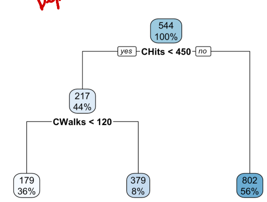
① large tree T_0





(how many splits)

subtree w/ Depth = 2



2 Classification Trees

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

Recall from regression trees, predicted response for an observation is given by the mean response of training obs. in that region.

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 mode

We are often also interested in class proportions that fall into each terminal node.

↳ this can give us some idea of how reliable the prediction is

e.g. terminal node w/ 100% class 1 vs. 55% class 1, 45% class 2.

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

Use binary recursive splitting to grow a classification tree

but RSS cannot be used as criterion for splitting.

Instead natural alternative is classification error rate

= fraction of training obs that do not belong to most common class

$= 1 - \max_k (\hat{p}_{mk})$ ← prop. of training obs in m^{th} region from k^{th} class.

It turns out that classification error is not sensitive enough. for tree growing.

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 choice 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 bootstrap sampling, instead each tree is fit on a modified version of the original data set.

Boosting has three tuning parameters:

1.

2.

3.