nonparametric supervised methods.

Chapter 8: Tree-Based Methods

We will introduce *tree-based* methods for regression (quantitative y) and classification (categorical y).

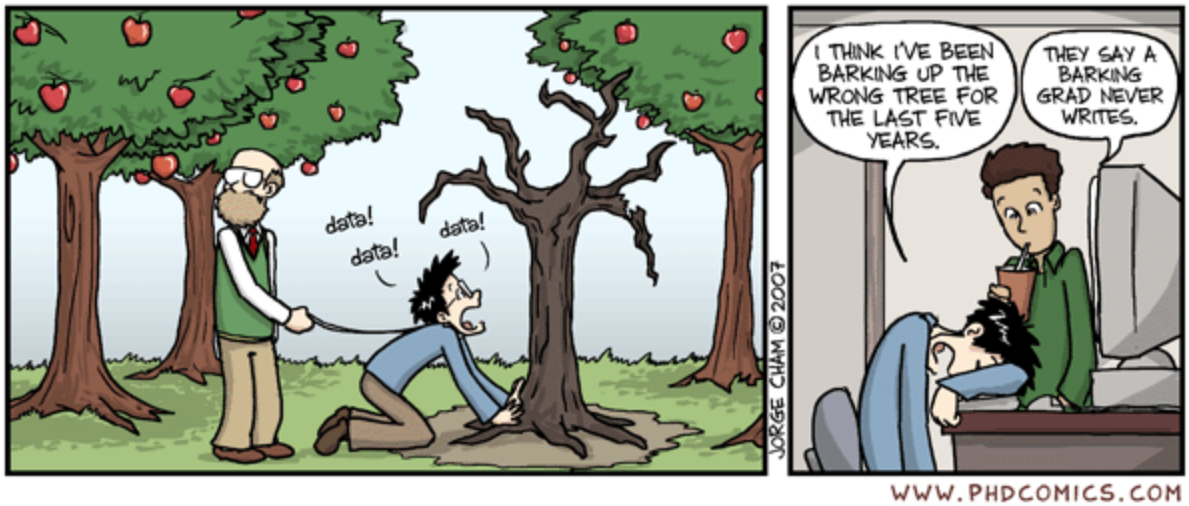
These involve segmenting the predictor space into a number of simple regions.

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 interpretty.
- not competitive w/ other supervised approach (e.g. lasso) for prediction.

Combining (boosting, bagging, random forests) 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.

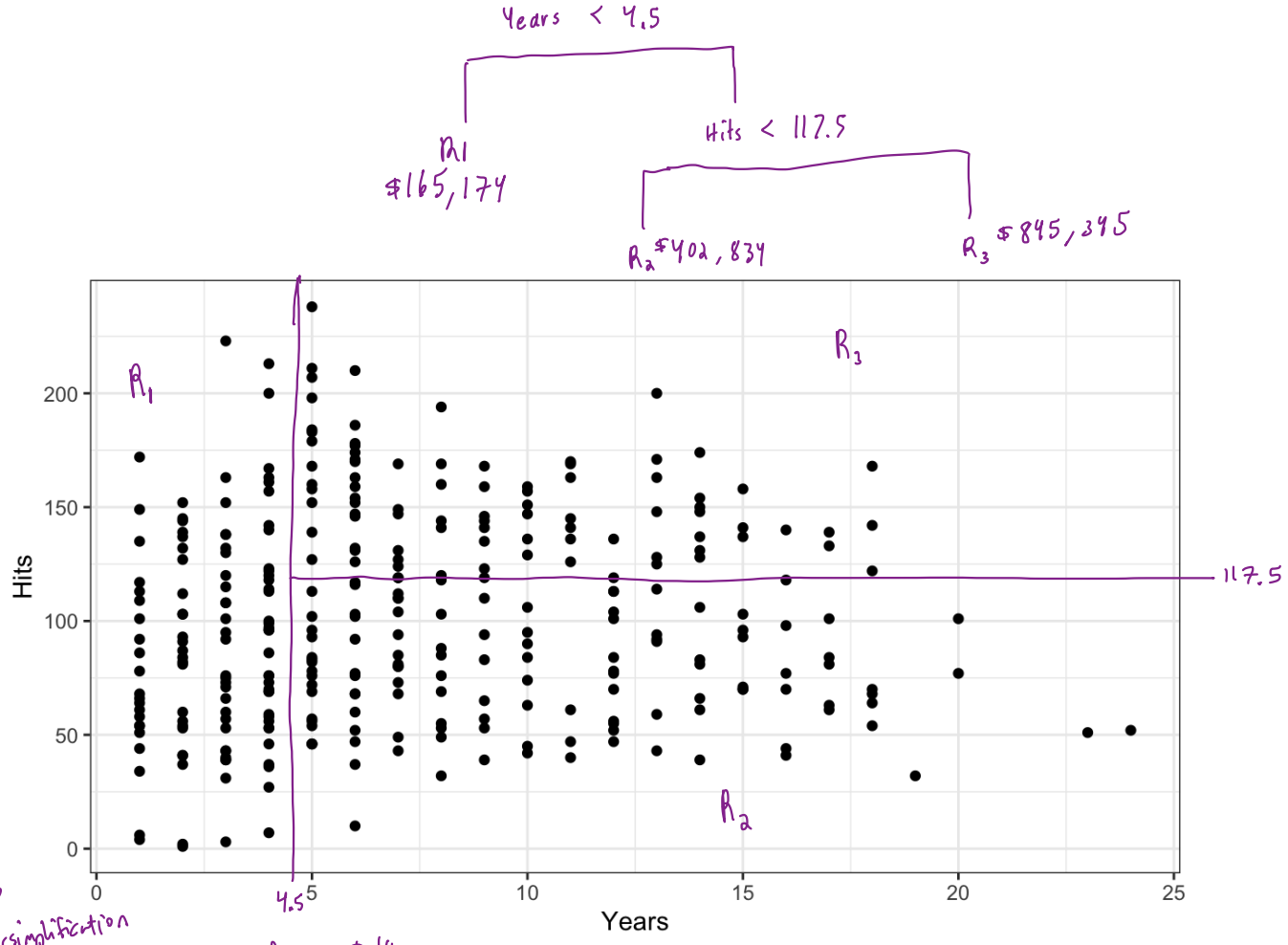
1 Regression Trees

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 two steps:

1. Divide predictor space (set of possible values for $X_1, \ldots, X_p$)
   into J distinct and non-overlapping regions $R_1, \ldots, R_J$

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 $R_1, \ldots, R_J$? 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 $R_1, \ldots, R_J$ 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_j^{th} box.

Unfortunately it is computationally infeasible to consider every possible partition.

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

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.
↳ 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 that particular step.
↙ 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 $X_1, \dots, X_p$ 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 $R_1, \dots, R_J$) 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 $T \subset T_0$ s.t.

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible.

$|T|$ = # terminal nodes in the tree.

α controls trade off between subtree's complexity and fit to training data.

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

$\alpha \uparrow \Rightarrow$ price to pay for having many terminal nodes $\uparrow$ $\Rightarrow$ smaller tree.

Select α by CV.

↳ Then use full data set & chosen α to get subtree.

Algorithm for building a regression tree:

(1) Use recursive binary splitting to grow a large tree on training data, stop only when each terminal node has fewer than some minimum # of observations.

(2) Apply cost complexity pruning to large tree to get a sequence of best trees (as 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 1 and 2 on all but k^{th} fold.

(b) Evaluate the MSPE on data in k^{th} fold as function of α.

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

(4) Return subtree from (2) that corresponds to α from (3).

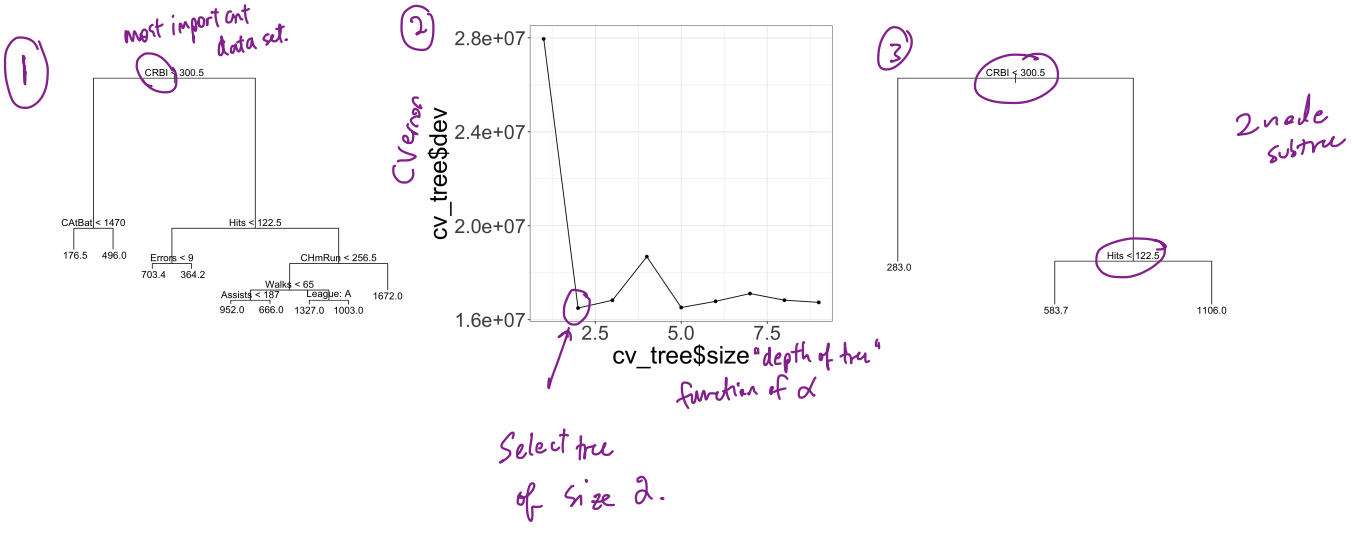
Ok too.

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

(1) is the large tree

(2) CV error to estimate test error as function of α.

(3) subtree selected



2 Classification Trees

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

Recall for a regression tree, the predicted response for an observation is given by the mean response of training observations that belong to a terminal node.

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 the class prediction proportions that fall into each terminal node.

↳ this gives us some idea of how reliable the prediction is

e.g. terminal node w/ 100% class 1 vr. 55% class 1, 45% class 2 — both predict "class 1."

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

Use recursive binary splitting to grow classification tree

But RSS cannot be used as criterion for splitting.

Instead a natural alternative is classification error rate.

= fraction of training obs that do not belong to most common class in terminal node.

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

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

preferred measures

more sensitive to node purity than classification error rate

(1) Gini index $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$ measure of total variance across K classes.

↳ takes small values if all $\hat{p}_{mk}$ close to 0 or 1. => measure of "node purity"

↓G => nodes contain primarily values from 1 class.

(2) Entropy $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

↳ will take values near 0 if $\hat{p}_{mk}$ close to 0 or 1 => ↓D when nodes more "pure"

Gini and Entropy are actually quite similar numerically

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

↑ note: neither work work well w/ unbalanced class data.

There are other options out there to split on.

Any of the 3 methods can be used for pruning, but if prediction accuracy of final pruned tree is the goal => classification error rate should be used for pruning.

3 Trees vs. Linear Models

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

eg. linear regression $f(x) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$

regression tree $f(x) = \sum_{m=1}^{M} c_m \mathbb{I}(x \in R_m)$.

where $R_1, \ldots, R_M$ are partition of the feature space.

Which method is better? It depends on the problem.

- If relationship between features & response is approximately linear, linear regression > regression tree.
- If high non-linear relationship (complex), decision trees may be better.

Also trees may be preferred because of interpretation or visualization.

3.1 Advantages and Disadvantages of Trees

Advantages

- easy to interpret (easier than linear regression)
- Some people think decision trees more closely mirror human decision making.
- can be displayed graphically (easy to interpret for non-experts - especially if small).
- can handle categorical predictors easily.

Disadvantages

- do not have same level of predictive performance as other methods we've seen.
- Not robust: small change in data can have large change in estimated tree (high variability)

↓

we can aggregate many trees to try to improve this!

"Bootstrap aggregation"

4 Bagging

Decision trees suffer from *high variance*.

i.e. if we split data in half randomly, fit decision tree to both halves, results could be quite different.

vs. low variance will yield similar results if applied repeatedly to distinct datasets (from same population)

↳ e.g. linear regression when $n > p$.

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

Recall: for a given set of n indep obs. $Z_1, \dots, Z_n$ each w/ variance σ^2,

$$\mathrm{Var}(\bar{Z}) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) \overset{\text{indep.}}{=} \frac{1}{n^2}\sum_{i=1}^{n} \mathrm{Var}\, Z_i = \frac{1}{n^2}\cdot n\cdot\sigma^2 = \frac{\sigma^2}{n}.$$

i.e., averaging a set of observations reduces variance.

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.

i.e. take B training sets,

Calculate $\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$.

obtain low-variance statistical learning model:

$$\hat{f}_{AVG}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x).$$

Of course, this is not practical because we generally do not have access to multiple training sets. Collecting training data can be expensive.

Instead we could take repeated samples (w/ replacement) from the training data set.

(these are called "bootstrapped" training data sets, i.e. "pulling ourselves up by our bootstraps")

↳ assumes empirical dsn from sample is similar to population dsn, i.e. we have a "good" (representative) sample

Then we could train our method on b^{th} bootstrapped data set to get $\hat{f}^{*b}(x)$ and average:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

This is called "bagging", short for bootstrap aggregation.

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

To apply bagging to regression trees,
① construct B regression trees using B bootstrapped data sets
② average resulting predictions.

These trees are grown deep and not pruned.
=> each tree has low bias & high variance
averaging trees reduces the variance by combining hundreds or thousands of trees!
↳ this can be slow.
won't lead to overfitting (because $E\varepsilon = 0$).

How can bagging be extended to a classification problem? (can't mean averaging no longer an option).
for a given test observation, record class predicted by each of the B bootstrapped trees and take a majority vote: overall prediction is the class that occurs most often.

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.

Key: trees are repeatedly fit to bootstrapped subsets of original training obs.
↳ on average each tree uses $\approx \frac{2}{3}$ of the data to fit the tree. (has to do with the prob of being selected in the bootstrap as $B \to \infty$.)
i.e. $\approx \frac{1}{3}$ of observations NOT used to fit the tree (out-of-bag OOB observations).

idea: We predict the response for ith observation using all trees in which that obs was OOB.
This will be $\approx \frac{B}{3}$ predictions of ith observation.
Average (or majority vote) of these predictions to get single OOB prediction for ith observation.

We can use these OOB predictions for each training obs to get OOB MSE (or OOB classification error) which is an estimate of test error!

This is valid because we only ever use predictions from models (trees) that did not use those data points in fitting!

4.2 Interpretation

Bagging typically results in improved accuracy in predictions over a single tree!

But it can be difficult to interpret a resulting model

↳ one of the biggest advantages of decision trees :(

↳ no longer possible to represent the resulting procedure using a single tree

⇒ no longer clear which variables are the most important to predict the response.

Bagging improves prediction at the total expense of interpretability.

What can we do?

We can obtain an overall summary of the importance of each predictor using RSS (or Gini index)

- record total amount of RSS (or Gini) is decreased due to splits over a given predictor averaged over B trees.
- a large value indicates an important predictor.

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.

But when building the trees, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
↳ the split is only allowed to use one of those predictors.
↳ fresh sample of predictors taken at each split
↳ typically $m \approx \sqrt{p}$.

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. Why?

Suppose there is one strong predictor in data set and a number of moderately strong predictors.
In the collection of trees, most or all trees will use strong predictor as the top split.
⇒ all of the bootstrapped trees will look quite similar!
⇒ predictions will be highly correlated.
and averaging (bagging) correlated values doesn't lead to much variance reduction.

Random forests overcome this by forcing each split to consider a subset of predictors.
⇒ on average $\frac{(p-m)}{p}$ of the splits will not even consider strong predictor ⇒ other predictors will have to be chosen.

The main difference between bagging and random forests is the choice of predictor subset size m. If $m=p$ ⇒ random forest = bagging.

Using small m will typically help when we have a lot of correlated predictors.

– As with bagging, we don't need to worry about overfitting w/ large B.
– And we can examine importance of each variable in the same way.

randomForest package in R.

6 Boosting

* very popular right now (see AdaBoost or XGboost).

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

Again the idea of boosting is a general approach that can be applied to many statistical learning models. We will use it w/ decision trees.

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 grows trees sequentially using information from previously grown trees.

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.