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

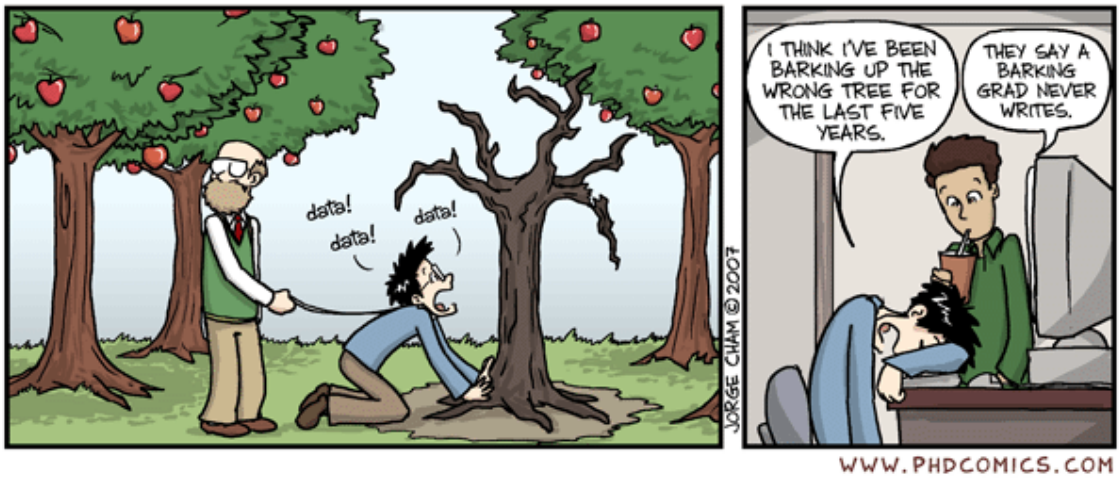
To make a prediction for an observation, we will use the mean or mode of the training observations in the regions to which it belongs.

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

- simple and useful for interpretation
- not competetive 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.

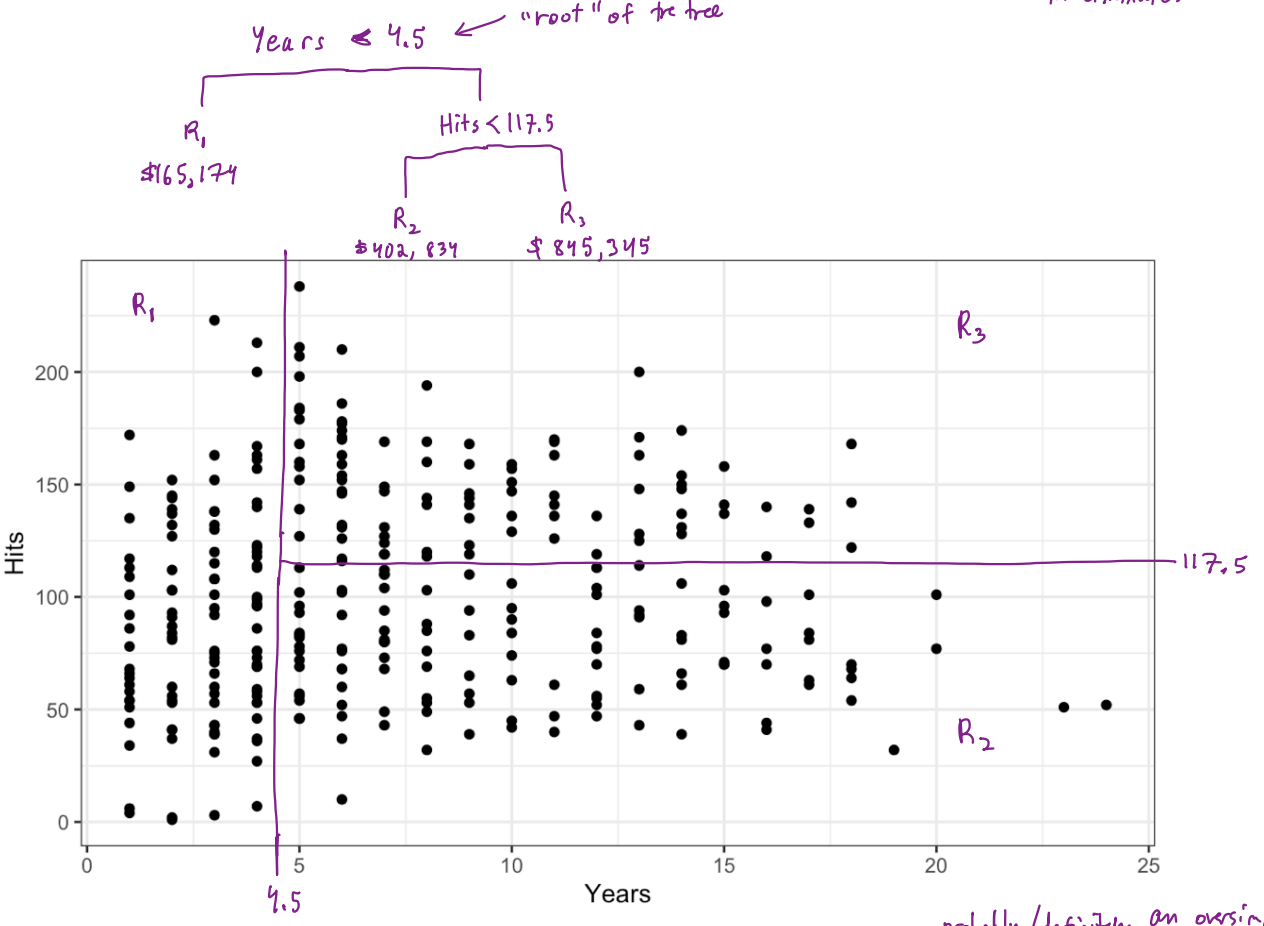
# 1 Regression Trees

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).




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 $X_1, \ldots, X_p$

1. Divide predictor space

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

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 $R_1, \ldots, R_J$? 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 $R_1, \ldots, R_J$ 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 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.

↳ 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.

↳ 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 cut point 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 $X_1, \dots, X_p$ 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 $\alpha$

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

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

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

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

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

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

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

$\alpha$ controls tradeoff between subtree's complexity & fit to training data

Select $\alpha$ via CV, then use full training dataset & chosen $\alpha$ 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 $\alpha$.

(3) Use K-fold CV to choose $\alpha$.

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

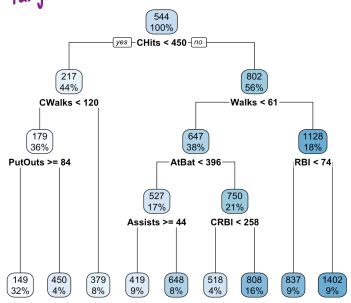
Average results for each value of $\alpha$ and pick $\alpha$ to minimize CV error.

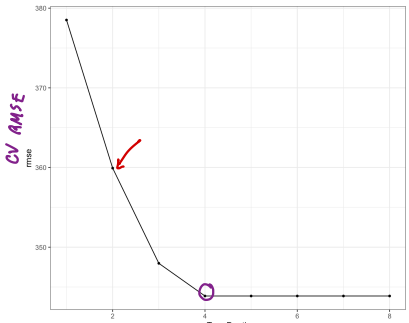
(4) Return the subtree from (2) that corresponds to $\alpha$ from (3).

---

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

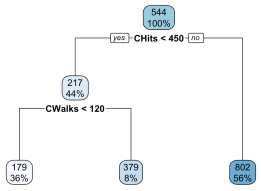
(1) large tree $T_0$




(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.

more sensitive to node purity than classification error rate.

Preferred measures:

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 $\hat{p}_{mk}$'s are close to 0 or 1. ⇒ measure of "node purity"
↓ G ⇒ nodes contain observations primarily from one class.

② Entropy $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$
↳ will take values near zero if $\hat{p}_{mk}$'s close to 0 or 1. ⇒ ↓D ⇒ 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 Gini nor entropy works well w/ unbalanced data. There are other options out there to split on.

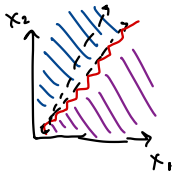
Any of the 3 methods can also be used for pruning.

If prediction accuracy of final tree is the goal,

classification error rate should be used.

# 3 Trees vs. Linear Models

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




e.g. linear regression: $f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$

regression trees: $f(X) = \sum_{m=1}^{M} c_m \mathbb{I}(X \in R_m)$. where $R_1, \ldots, R_m$ is a partition of the predictor space.

Which method is better? It depends on the problem

- If the relationship btw/ features and response is approximately linear, then a linear model will outperform a tree.
- If the relationship is highly non-linear, decision tree may be better.

Trees nice interpretation / visualization.

## 3.1 Advantages and Disadvantages of Trees

Advantages

- easy to explain, even easier than linear regression.
- (?) some people think decision trees more closely mirror human decision making.
- can be displayed graphically (good for nonexperts)
- can handle categorical predictors without needing to make dummy variables.

Disadvantages

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

↓

We can aggregate many trees to try and improve this!

# 4 Bagging

"Bootstrap aggregation"

Decision trees suffer from *high variance*.

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

vs. low variance will yield similar results if applied repeatedly to different samples from same population.

↳ 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 $n$ independent observations $Z_1, \dots, Z_n$ each w/ variance $\sigma^2 < \infty$.

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

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

obtain low-variance statistical learning model:

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

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 training data set.
(these are called "bootstrapped" training data sets because we are "pulling ourselves up by our bootstraps").

↳ assumes empirical dsn in sample is similar to population dsn, i.e. have representative sample.

Then we could fit your method on $b^{\text{th}}$ bootstrapped training data set to get $\hat{f}^{*(b)}(x)$ and average

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

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

To apply to regression trees: ① construct B trees using 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 variance by combining hundreds or thousands of trees!
↳ won't lead to overfitting (but can be slow).

How can bagging be extended to a classification problem? (averaging no longer an option).

For a given test observation, record the class predicted by each bootstrapped tree and take a majority vote: overall prediction is class 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 observations.

⇒ on average each tree uses $\approx \frac{2}{3}$ of the data to fit the tree.
↖ has to do w/ prob. of being selected in the bootstrap.

i.e. $\approx \frac{1}{3}$ of observations are NOT used to fit the tree. (out-of-bag OOB observations).

idea: We can predict response for the $i^{th}$ observation using all trees in which that observation was OOB.
This will lead to $\approx B/3$ predictions for $i^{th}$ observation.
Then average (or majority vote) of these predictions to get a single OOB prediction for $i^{th}$ observation.

We can use each of the OOB predictions for each training obs to obtain OOB MSE (or OOB classification error)
Which is an estimate of test error!

This is valid because only use predictions from models (trees) that did not use that observation in fitting!

## 4.2 Interpretation

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

But it can be difficult to interpret the resulting model!

↳ one of the biggest strengths of decision trees ☹

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

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

⇒ bagging improves prediction at the expense of interpretability.

What can we do?

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

- record total amount 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 from the full set of $p$ predictors as split candidates.

↳ the split is only allowed to use one of those $m$ 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 bagging, most or all trees will select the strong predictor as the top split.

⇒ all of the bagged trees will look quite similar.

⇒ predictions will be highly correlated

and averaging highly correlated values does not lead to much variance reduction!

Random forests overcome this by forcing each split to consider a subset of predictors.

⇒ on average $(p-m)/p$ of the splits will not even consider the strong predictor ⇒ other predictors will have a chance!

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

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

- As with bagging, we will not have overfitting with large $B$.
- And we can examine the importance of each variable in the same way.

# 6 Boosting

* very popular right now (see Ada boost and XGboost).

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

idea is a general approach can be applied to many models.

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

Regression:

idea - the boosting approach learns slowly to avoid overfitting.

> Given a current model we fit a decision tree to residuals from current model and add the decision tree to the fitted function to update.

> each tree is very small (just a few terminal nodes)

⇒ slowly improving $\hat{f}$ in areas where it is not currently performing well.

Boosting has three tuning parameters:

1.

2.

3.