Chapter 8: Tree-Based Methods

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

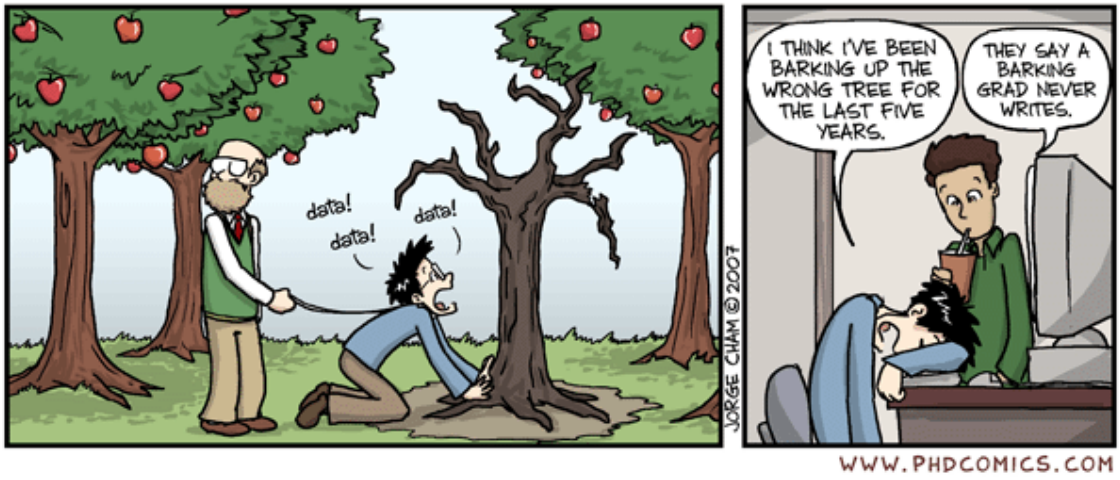
These involve segmenting the predictor space into a number of simple regions.

To make a prediction for an observation, we use the 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 interpretation
- not competitive w/ other supervised approaches (e.g. lasso) for prediction.

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

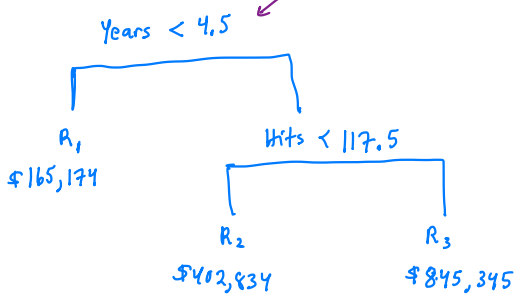
start with

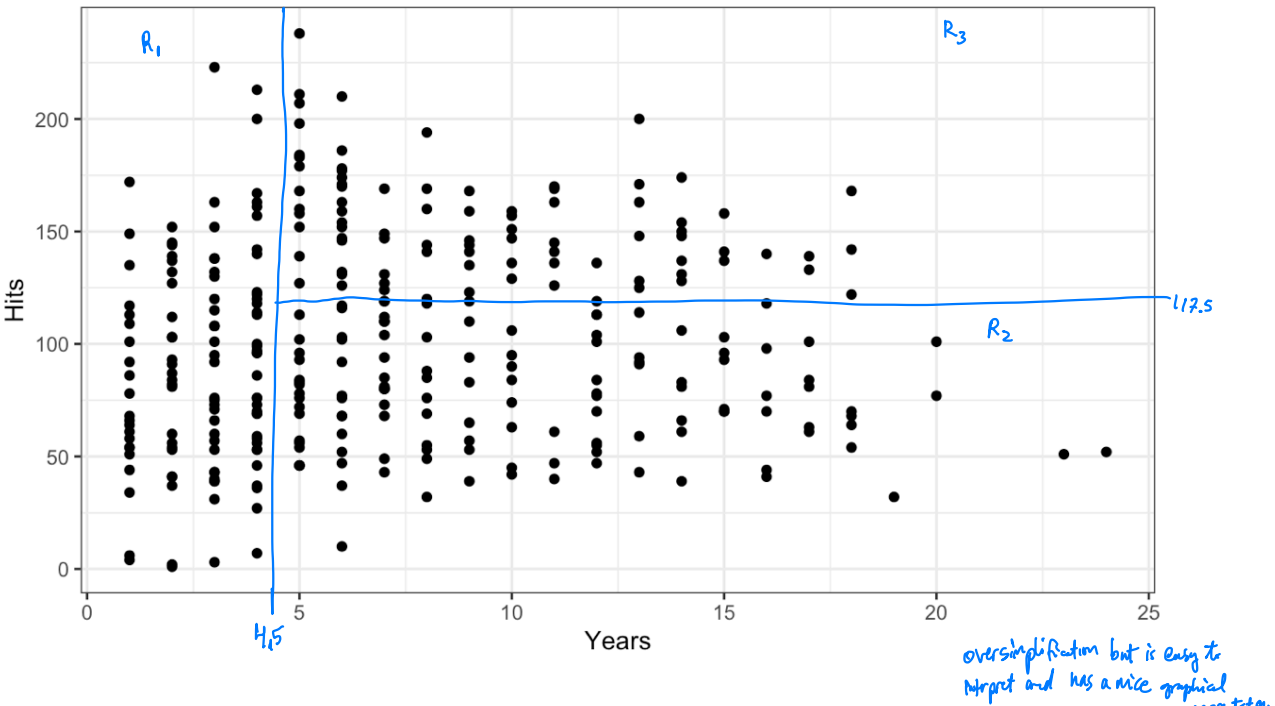
Example: We want to predict baseball salaries using the `Hitters` data set based on `Years` (the number of years that a player has been in the major leagues) and `Hits` (the number of hits he made the previous year).

← according to a tree fit to this data (more later).

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

"root" of the tree





oversimplification but is 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 predictor 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 has less experience, # hits in previous year play little role in his salary.

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

↳ quantitative Y

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

↗ set of possible values for $X_1, \ldots, X_p$

1. Divide the predictor space

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

2. Predict

   For every observation that falls in 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, & to 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. $= \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2$ where

Unfortunately, it is computationally infeasible to consider every possible partition.

$\hat{y}_{R_j}$ = mean response of training data in R_jth box.

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

The approach is *top-down* because

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

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 X_j and cutpoint s s.t. splitting the predictor space into regions $\{X \mid X_j < s\}$ and $\{X \mid X_j \geq s\}$ leads the greatest reduction in RSS.

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

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

(2) Repeat process looking for next best j,s combo, but instead of splitting the 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 observations).

(4) predict using mean of training observations in the region that test obs. fell.

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

because resulting tree too complex.

⇒ less regions $R_1, \ldots R_J$

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 large enough drop in RSS.

Bad idea because seemingly worthless splits early in the tree might 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 the lowest test error rate. → could use CV for every subtree but too expensive (large # of possible subtrees).

Solution: "cost complexity pruning"

Consider a sequence of trees indexed by a nonnegative tuning parameter α.

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

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

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

R_m = m^{th} terminal node region

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

α controls the trade-off between subtree's complexity & fit to training data. → When $\alpha = 0$, $T = T_0$, $\alpha \uparrow$ ⇒ price to pay for having many terminal nodes $\uparrow$ ⇒ smaller tree

Select α via CV!

Then go back and use full data & chosen α to get subtree.

Algorithm for building a regression tree:

(1) Use recursive binary splitting to grow a large tree on training data, stopping only when each terminal node has fewer than some # (5?) 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 $k = 1, \ldots, K$

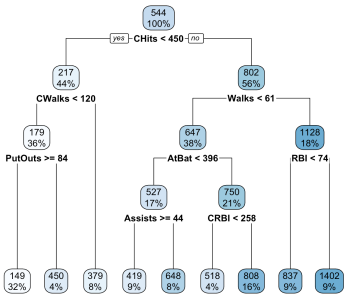
(a) repeate (1) and (2) on all but k^{th} fold

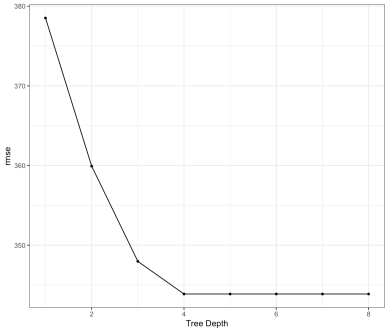
(b) Evaluate the MSE on data in k^{th} fold as a function of α.

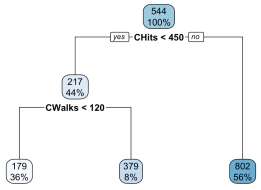
Average results for each value of α to get CV error. Choose α which minimizes CV_k

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

Example: Fit regression tree to Hitters use 9 features







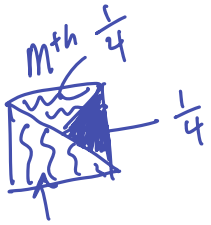
2 Classification Trees

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

Regr: Numeric, continuous

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.

often interested class prediction proportions w/in each region (Terminal Nodes)



① 100% training obs in Class A. ② 55% training obs in A, 45% .. B

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

$$RSS = \sum_{j=1}^{J} \underbrace{\sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2}_{\text{all error in a region}}$$

Binary Recursive Splitting

Can't use RSS

1/2, 1 − 1/2 ≤ 1/2

① Classification Error Rate = fraction of training obs. that don't belong to the most common class in a region

$$= 1 - \max_k (\hat{p}_{mk})$$

$\hat{p}_{mk}$ = proportion of training obs in the m^{th} region for the k^{th} class

It turns out that classification error is not sensitive enough.

② Gini Index: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$ = measure of total variance across all classes

- small: $\hat{p}_{mk}$ close to 0 or 1
- measure node purity: ↓G ⇒ nodes contain primarily 1 class

NOTE: ②, ③ don't work well in unbalanced class situation

③ Entropy: $D = -\sum \hat{p}_{mk} \log(\hat{p}_{mk})$

- values near 0 ⇒ node is more "pure"

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

use any of these 3 methods for pruning

for good prediction accuracy – use classification error rate

3 Trees vs. Linear Models

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

Linear Regression: $f(x) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$

Reg tree: $f(x) = \sum_{m=1}^{M} c_m \mathbb{I}(x \in R_m)$ $R_1, \ldots, R_m$ partition predictor space

Which method is better?

it depends.

linear relationships → linear regression

Tree-based
highly nonlinear
interpretation ✓
visualization

3.1 Advantages and Disadvantages of Trees

Pros

- interpretation / explanation
- some people think imitates human decision-making
- Graphical interpretion
- Can handle categorical responses
- categorical predictors
  - easier

Cons

- lower predictive performance
- Not Robust → Variability
  High Variance

Can be improved by Aggregating

4 Bagging

Decision trees suffer from *high variance*.

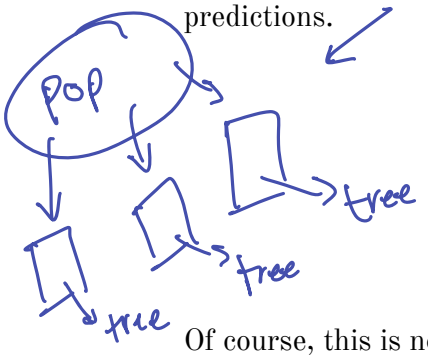
Bootstrap aggregation or *bagging* is a general-purpose procedure for reducing the variance of a statistical learning method, particularly useful for trees.

$Z_1, \ldots, Z_n$ iid $\quad Var(Z_i) = \sigma^2 \quad \bar{Z} = \frac{1}{n}\sum Z_i$

But $\bar{Z}$: $Var(\bar{Z}) = \frac{1}{n^2}\sum Var(Z_i) = \frac{1}{n^2}\sum \sigma^2$

$= \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$

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.



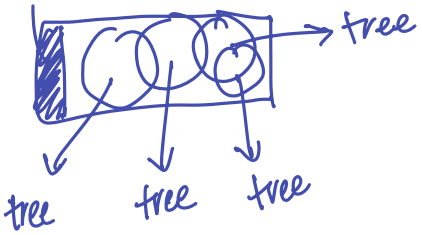
$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \hat{f}^{(3)}(x)$

→ Average $\frac{1}{n}\sum_{i=1}^{*} f^{(i)}(x)$

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

EXPENSIVE

↳ sol: Repeated Samples



(1) get bootstrap samples repeated sampling

(2) build tree for each sample

(3) predict for new obs w/ each tree

(4) Average all tree predictions

ASSUMPTION: Training data looks like population

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

These trees are grown deep and not pruned.

1 tree

- low bias
- High variance

} Average →

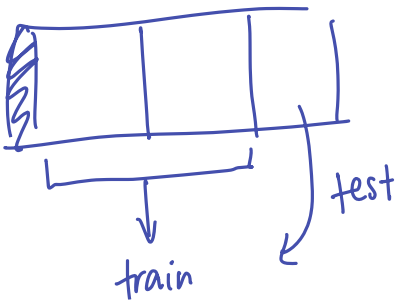
100's trees 1000's trees
↳ slow
- won't lead to overfitting

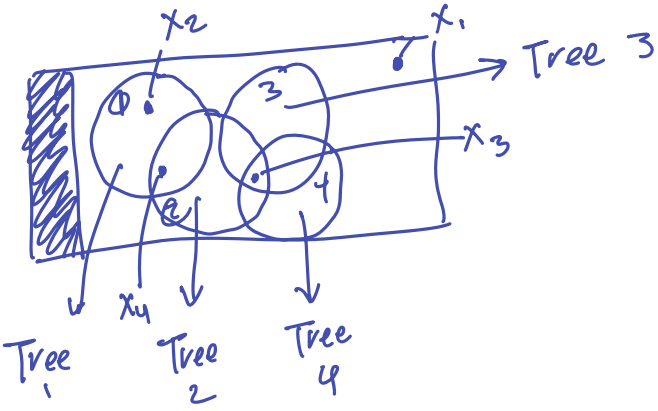
How can bagging be extended to a classification problem?

- can't average predictions
- majority vote

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.





OOB Error

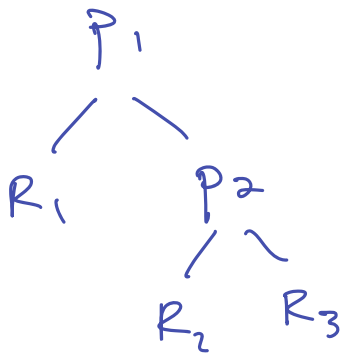
X_1 : 1, 2, 3, 4 → Average Error

X_2 : 2, 3, 4 →

X_3 : 1 →

X_4 : 3, 4 →

4.2 Interpretation (Bagging)



Bagging

- improves Accuracy
- cost is in interpretation – more difficult
  - Recall: interpretation w/ single tree
  - Now: Can't represent procedure w/ 1 tree

→ Reduction in RSS (or another loss function) determines importance
(gini/entropy...)

- obtain overall summary of importance using loss
  - idea: Average reduction in loss across all trees for each variable (due to splits)

- greatest average decrease in loss ⟹ Most important variable

5 Random Forests

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

? Why are the trees correlated?

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

$m < p$

randomly sample m variables as split candidates for each tree

- tree only allowed to split on sampled predictors
- diff sample of m predictors for each tree
- $m \approx \sqrt{p}$ (?)

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.

$Var(X_i) = \sigma^2 \quad Var(\bar{X}) = \frac{\sigma^2}{n}$ } ASSUMES UNCORRELATE X_i

$Var\left(\frac{X_1 + X_2}{2}\right) = \frac{1}{4}\left(Var(X_1) + Var(X_2) + 2Cov(X_1, X_2)\right)$

WHY?

if we have a very strong predictor & several strongish predictors

Bagging: ① all trees split similarly ② predictions highly correlated. → can't guarantee a reduction in variance

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

on Avg: $\frac{(p-m)}{p}$ of splits won't even consider the strong predictor

$m = p$: RF = Bagging

- won't have overfitting B (# of trees)
- estimate OOB error / importance the same way as bagging

6 Boosting

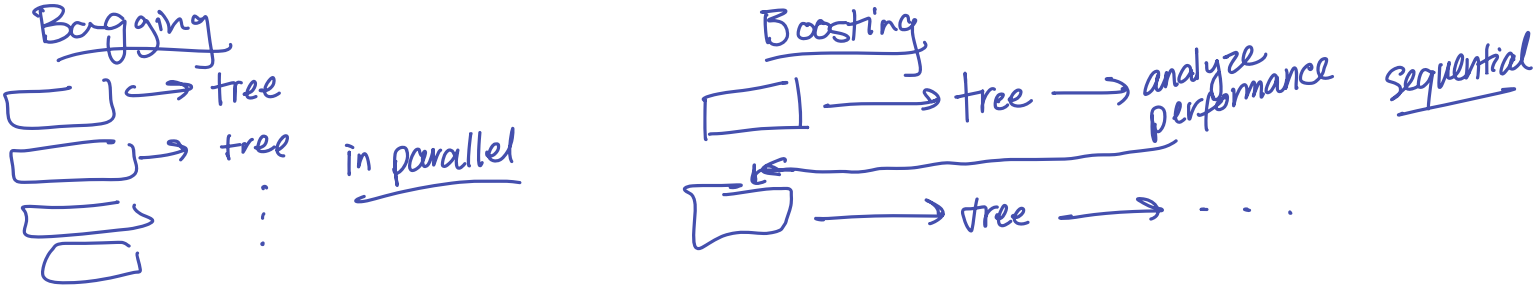
very cool. very popular

Adaptive boosting (AdaBoost) extreme gradient boosting (XGboost)

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

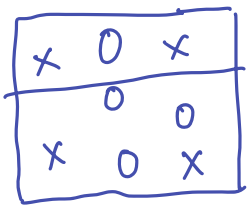
- general method/idea that can be applied to other models
- used to improve accuracy

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.

- grows trees sequentially using info from previously grown trees.



Regression

idea: learn slowly to avoid overfitting

Given a current model, fit a tree to the residuals & add decision tree to the fitted function to update.

→ each tree is very small → slowly improve $\hat{f}$ in areas where it predicts poorly.

Algorithm

① Set $\hat{f}(x) = 0$ $r_i = y_i$ $\forall$ i in our training set.

② $b = 1, \ldots, B$ repeat:

a.) fit a tree $\hat{f}^b$ w/ d splits to $r_i \sim x$ (x, r)

b.) update $\hat{f}$ by adding a shrunken version of $\hat{f}^b$

$$\hat{f} \to \hat{f}(x) + \lambda \hat{f}^b(x)$$

c.) update residuals $r_i = r_i - \lambda \hat{f}^b(x_i)$

(3) output boosted model

Classification: similar idea but more complex

Boosting has three tuning parameters:

1. B : # of trees

Unlike bagging/RF boosting can overfit w/ large B — use CV

(B, λ)

2. λ : shrinkage / learning rate — use CV.

- small positive number
- controls learning rate (bigger ≠ better)
- very small λ can require large B to achieve good performance
- depends on problem

3.

d - # of splits in each tree

- controls model complexity
- d=1 work well (stumps)

Generally d is the interaction depth

w/ d splits → @ least d variables d < p

Cool thing about boosting

- it works well
- fits nicely into "Decision Theory" framework
  - ↳ guarantees on behavior