Chapter 8: Tree-Based Methods

→ non parametric supervised.

↗ quantitative response Y ↗ categorical response Y.

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

These involve segmenting the predictor space into a number of simple regions
↳ all values $X_1, \dots, X_p$ can take.

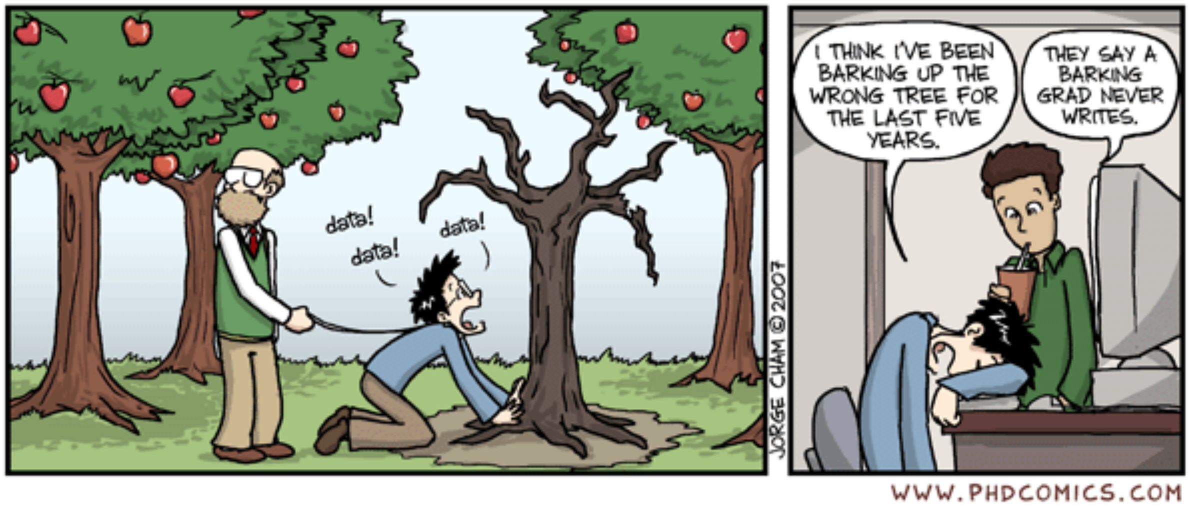
To make a prediction for an observation, we use the mean or mode of training observations in the region to which it belongs.
mean ↓ regression; mode ↘ classification.

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.

↗ boosting, bagging, random forests (later)

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


data!
data!
data!
JORGE CHAM © 2007
I THINK I'VE BEEN BARKING UP THE WRONG TREE FOR THE LAST FIVE YEARS.
THEY SAY A BARKING GRAD NEVER WRITES.
WWW.PHDCOMICS.COM

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

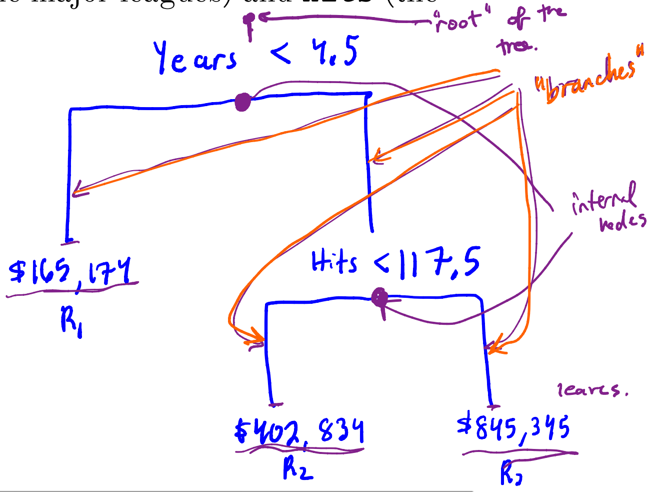
Decision trees can be applied to both regression and classification problems. We will start with regression.

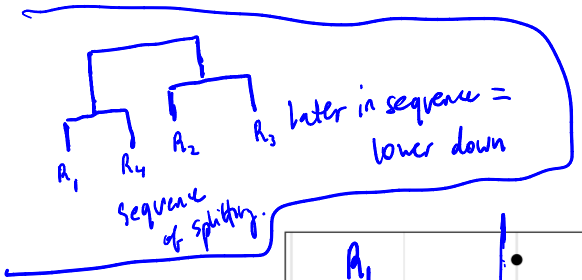
1 Regression Trees

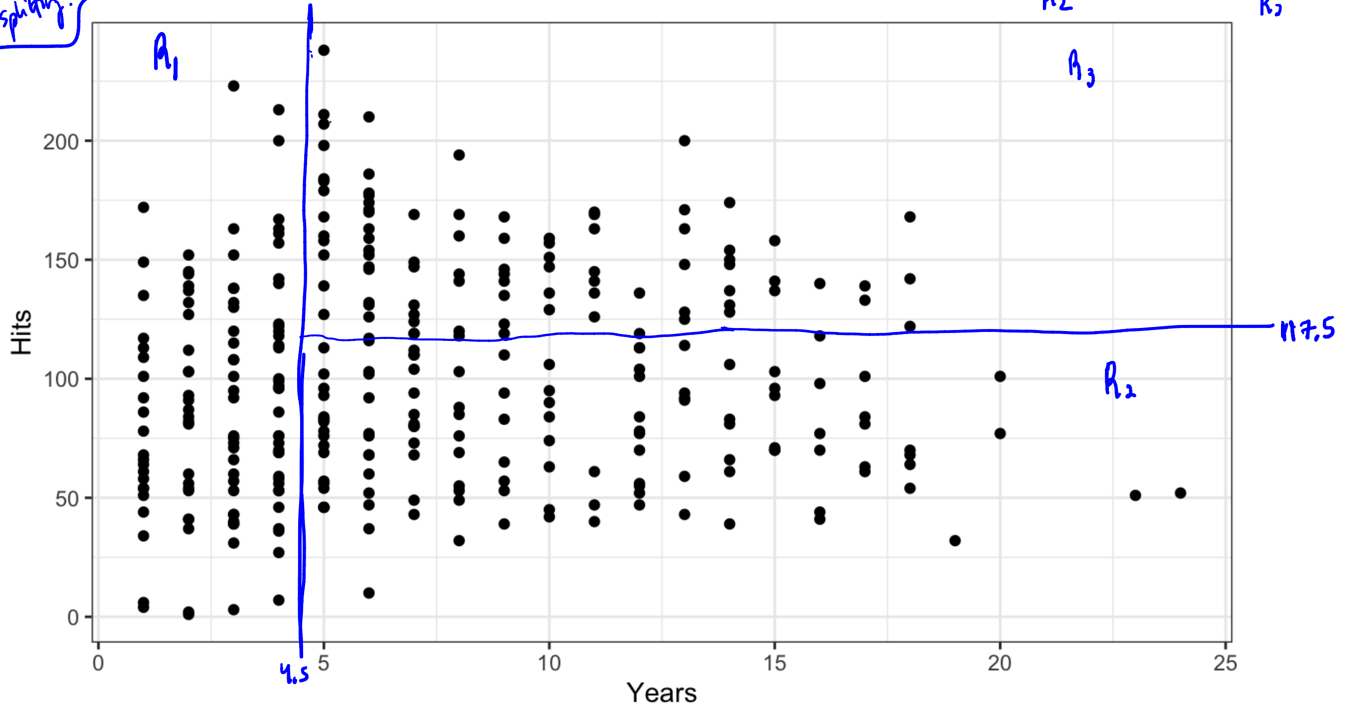
start with

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







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 nodes = "branches"

Interpretation Years is most important factor in determining salary.

↳ given that a player has less experience (< 4.5 years), # hits plays very little role in his salary

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

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

We now discuss the process of building a regression tree. $\nearrow$ quantitative response Y. There are t^w^o steps:

1. Divide predictor space $\nearrow$ set of possible values for $X_1, \dots, X_p$

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

2. Predict

   For every observation that falls into the region R_j we make the same prediction

   = mean of response Y for every training observation 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's 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_j^{th} box.

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 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 down the tree.

The approach is *greedy* because

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

region of predictor space where X_j takes values $< s$.

consider all possible $X_1, \dots, X_p$ and cutpoints s (based on training data), then choose predictor + cutpoint that result in lowest 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, 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, s can be quickly done when p not too large.

② Repeat process, looking for next best j, s combo but instead of splitting entire space, split $R_1(j,s)$ or $R_2(j,s)$ 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 observations in the region to which test observation 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 the tree if it resulted in "large enough" drop in RSS.

bad idea because a seemingly "worthless" split early in the tree might be followed by a good split later (large drop in RSS).

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. $\rightarrow$ could use CV to estimate the error for every possible subtree. This is expensive! (large # of potential 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 $T \subset T_0$ s.t.

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

$|T|$: # of terminal nodes in tree T

penalizing complexity in the tree

R_m = mth terminal node region

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

α controls trade off between subtree complexity + fit to training data

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

$\alpha \uparrow \Rightarrow$ pay for having many terminal nodes $\Rightarrow$ smaller subtree.

Select α via CV, then use full dataset + chosen α to get subtree T.

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 min. # of observations.

2. Use cost complexity pruning 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 1 and 2 on all data but k^{th} fold.

   (b) evaluate the predicted MSE on k^{th} fold as a function of α.

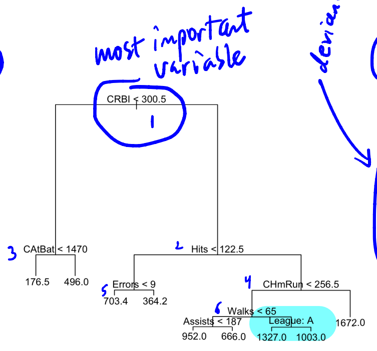
   Average results for each value of α and pick α that minimizes CV error.

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

<u>Example</u>: Fit regression tree to $\overset{\text{subset of}}{\vee}$ Hitters using 9 features. predicting salary.

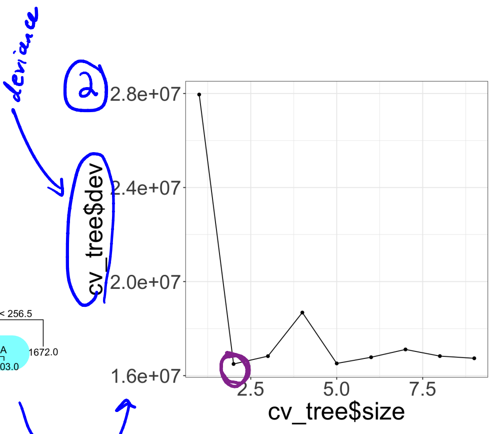
1. is the large tree
2. CV error to estimate test MSE as a function of α
3. subtree selected.

1



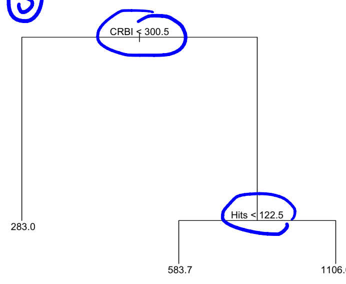
apply cost-complexity pruning

2



size of tree (one-to-one relationship w/ α)

3



2 inner node subtree.

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 regression tree, the predicted response for an observation is given by the mean response of the training observations that belong to the same 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 also often interested in the class prediction proportions that fall into each terminal node.

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

e.g. terminal node — training data in terminal node is 100% Class 1 vs. 55% Class 1, 45% Class 2 — "node purity"

both terminal nodes will predict as "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 a classification tree.

But RSS cannot be used as criterion for splitting.

Instead, natural alternative is classification error rate.

= fraction of training observations that do not belong to the most common class.

$$= 1 - \max_k \{ \underbrace{\hat{p}_{mk}}_{\text{proportion of training observations in the } m\text{th region from } k\text{th class}} \}$$

It turns out that classification error is not sensitive enough to use for growing the tree.
^ as splitting criteria

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.

↳ will take small values if all $\hat{p}_{mk}$'s are close to zero or one $\Rightarrow$ good measure of node purity, $\downarrow G \Rightarrow$ nodes contain primarily obs. from 1 class.

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

↳ will take small values if $\hat{p}_{mk}$ close to 0 or 1 $\Rightarrow \downarrow D$ when nodes are more pure.

note: neither Gini nor entropy work well w/ unbalanced classes in data.

There are other options out there to split on.

Gini and entropy are actually quite similar.

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

minimize. $\left(\text{measure of distance in data} + \alpha|T|\right)$

Any of 3 methods (classification error, Gini, or entropy) can also be used for pruning

But if prediction accuracy of final pruned tree is the goal, classification error rate should be used.

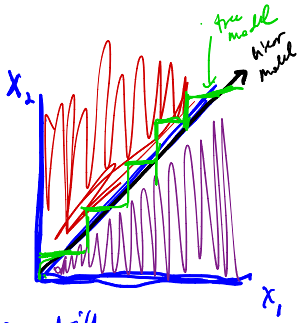
3 Trees vs. Linear Models

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

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 $R_1, \dots, R_M$ are partitions of the feature space.



Which method is better? It depends on the problem.

- If the true relationship between features and response is approximately linear, will out perform a regression tree.
- If highly nonlinear and complex relationships, decision trees may be better.

Also, trees may be preferred because of interpretation and 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, easy to interpret for nonexpert (especially if small).
- can handle categorical predictors without creating dummy variables.

Disadvantages

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

$\downarrow$

We can aggregate many trees to try and improve this! (Next).

4 Bagging

Decision trees suffer from *high variance.*

i.e. if we split data in half (randomly) and fit decision trees to each half, results could be quite differente

vs. low variance will yield similar results if applied repeatedly to distinct datasets.

↳ linear regression is low variance $n >> p$.

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

Recall: For a given set of n independent observations $Z_1, \dots, Z_n$ each w/ variance σ^2,

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

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 a 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" because we are bootstrapping samples from the population using only one training data set, i.e. "pulling ourselves up from our bootstraps")

↳ assuming empirical distribution in sample is similar to population dsn, i.e. we have a representative sample.

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

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

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 predictors

These trees are grown deep and not pruned.

$\Rightarrow$ each tree have low bias & $\uparrow$ variance.

averaging trees reduces variance by combining hundreds or thousands of trees!

$\hookrightarrow$ won't lead to overfitting, but can be slow.

How can bagging be extended to a classification problem?

For a given test observation, record class prediction from each tree and take 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 fit to bootstrapped subsets of observations.

$\Rightarrow$ on average each tree uses $\approx \frac{2}{3}$ of the data to fit the tree $\leftarrow$ has to do w/ probability of being selected in a bootstrap sample as $B \uparrow \infty$

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

idea: predict the response for i^{th} observation using all the trees in which that observation was OOB.

This will lead to $\approx B/3$ predictions for i^{th} observation.

Then average (or majority vote) these predictions to get single OOB prediction for i^{th} observation.

We can then get OOB prediction for each training observation to get OOB MSE (OOB classification error), which is an estimate of test error!

$\uparrow$ because we only use predictions from trees that didn't use those data points in the fitting.

4.2 Interpretation

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

But it can be difficult to <u>interpret</u> the resulting model!

↳ one of the biggest advantages of trees !!

↳ no longer represent model using a single tree.

⟹ no longer clear which variables are the <u>most important</u> to predict the response!

Bagging improves prediction at the cost 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 RSS (or Gini) is decreased due to splits for a given predictor, averaged over B trees.

- large value indicates an important predictors.

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 trees, a random sample of m predictors is chosen a split candidates from the full set of predictors.

↳ each split is allowed only to use those chosen 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 the data set and a number of moderately strong predictors.

In the collection of trees, most or all will use the strong predictors as the top split.

$\Rightarrow$ all of of The bagged trees will look quite similar

$\Rightarrow$ predictions will be highly correlated.

(bagging) 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.

$\Rightarrow$ on average $\frac{(p-m)}{p}$ of The splits will not even consider the strong predictor $\Rightarrow$ other predictors will have higher chance of being split on.

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

If $m=p \Rightarrow$ random forest = bagging.

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

- As with bagging, we will not have overfitting w/ large B

- And we can examine variable importance in the same way.

Ensemble models

6 Boosting

* very popular right now (Adaboost and XGBoost).

Boosting is another approach for improving the prediction results from a decision tree. ↱ can be more general.

↳ general approach can be used w/ 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, (independent trees on each bootstrap dataset)

Boosting grows the 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.

regression:

idea: the boosting approach learns slowly

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

> each tree is quite small (just a few terminal nodes) ⟹ slowly improve $\hat{f}$ in areas where it does not perform well.

Algorithm

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

② $b = 1, \dots, B$ repeat

(a) Fit a tree $\hat{f}^b$ w/ d splits ($d+1$ terminal nodes) to training data (x, r) residuals

(b) Update $\hat{f}$ by adding a shrunken version of $\hat{f}^b$

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

helps us avoid overfitting.

(c) Update residuals

$$r_i = r_i - \lambda \hat{f}^{(b)}(x_i)$$

③ Output the boosted model: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.

Boosting classification trees is similar, but the details are more complex.

Boosting has three tuning parameters:

1. B - the # of trees
   Unlike bagging and RF, boosting can overfit the data with large B.
   We can use CV to select B.

2. λ - shrinkage parameter (small pos. #)
   this controls the rate at which boosting learns.
   Typical values: $\lambda = 0.01$ or $\lambda = 0.001$
   Very small λ can require large B to achieve good performance.
   depends on problem / data.

3. d - # of splits in each tree
   controls complexity of the whole model
   Often $d=1$ works well (stumps)
   $\hookrightarrow$ if $d=1$, the boosted ensemble is additive.
   "Ada Boost"
   Generally d is the interaction depth and controls the interaction order of the boosted model since d splits $\Rightarrow$ at most d variables.

One of the coolest things about boosting (IMO) is not only does it work well, but it fits nicely into a statistical framework called "decision theory" meaning we have some understanding and guarantees about its behaviors.