Supervised Learning: Trees, Bagging, Random Forests, Boosting

Daniela Witten & Noah Simon

July 19–21, 2017
Summer Institute for Statistics of Big Data
University of Washington

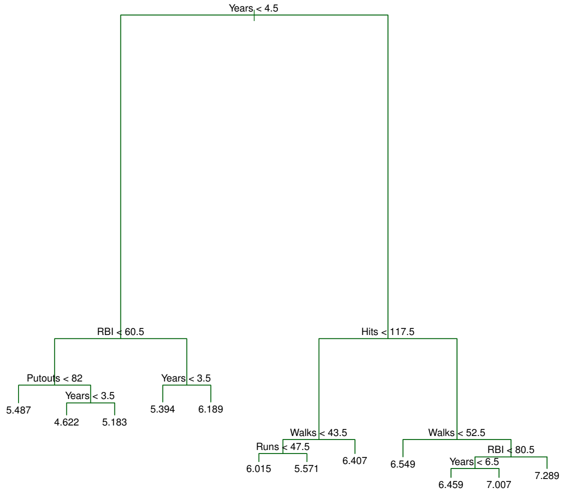
Decision Trees

- A completely different approach to non-linear modeling!
- Partition the range of X's into boxes.
- Then make a prediction for Y within each box.
- Leads to very easily interpretable models . . . though with some loss of prediction accuracy compared to other approaches.

Baseball Data

Want to predict a baseball player's Salary based on various characteristics of his past performance.

A Regression Tree for Baseball Data



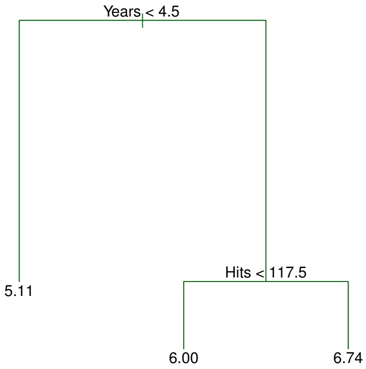
Regression Trees

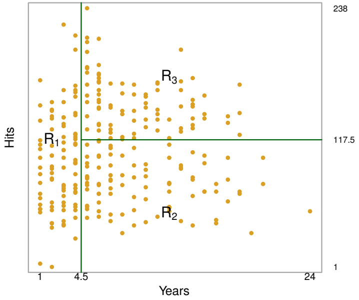
1. Partition the range of X into M disjoint regions, $R_1, \ldots, R_M$.
2. Within the mth region, predict Y as

$$\hat{f}(X) = \sum_{m=1}^{M} c_m I(X \in R_m)$$

where c_m is the mean response value among the training observations in R_m.

A Regression Tree for Baseball Data

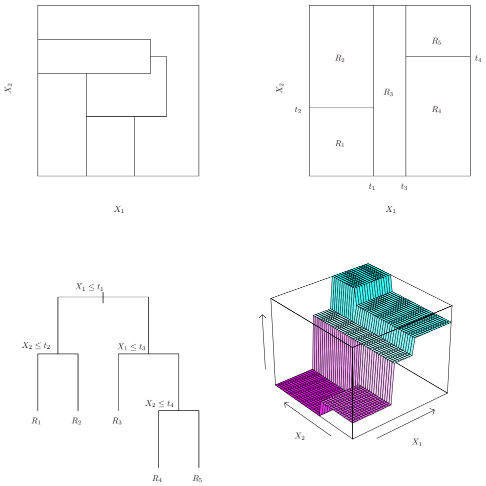




Recursive Binary Splitting

Q: How do we determine the regions $R_1, \ldots, R_M$?
A: Recursive binary splitting.

Recusive Binary Splitting



How Do We Get the Regions?

- Seek the variable j and cut-point s that minimizes

$$\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2,$$

  where $R_1(j,s) = \{X \mid X_j \leq s\}$ and $R_2(j,s) = \{X \mid X_j > s\}$.
- For any choice of j and s, we get

$$\hat{c}_1 = \mathrm{Ave}(y_i \mid x_i \in R_1(j,s)), \qquad \hat{c}_2 = \mathrm{Ave}(y_i \mid x_i \in R_2(j,s)).$$

- For any variable j, computing the split point s is very fast. So scanning through the variables to find the best (j,s) is fast.
- Once we find the best (j,s), we partition the data into the two regions, and repeat splitting in each region.
- Continue along these lines.

How Do We Make Predictions?

- Consider a partition into M regions, $R_1, \ldots, R_M$.
- We model the response as a constant within each region,

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

- To minimize sum of squared errors, the best $\hat{c}_m$ is

$$\hat{c}_m = \mathrm{Ave}(y_i \mid x_i \in R_m).$$

How Big Should We Grow the Tree?

- A large tree might overfit the data . . . high variance.
- A small tree might miss important structure . . . high bias.
- Tree size is a tuning parameter that governs model complexity.
- Optimal tree size should be chosen adaptively from the data.
- Tempting to grow the tree until the decrease in sum-of-squares due to further splits exceeds some threshold. But this strategy is too short-sighted.
- Instead, we build a very big tree, and then prune it down.

Cost-Complexity Pruning, Part I

- Define a subtree $T \subset T_0$ to be a tree obtained by pruning T_0.
- The mth terminal node represents region R_m.
- Let $|T|$ denote number of terminal nodes in T.
- Pruning the tree is complicated because there are so many subtrees!
- Need a way to navigate tree spacee: cost-complexity pruning.

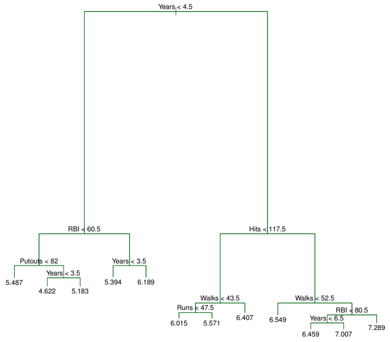
Cost-Complexity Pruning, Part II

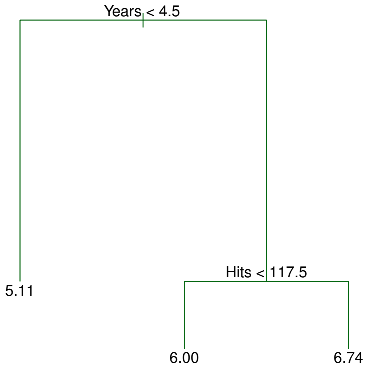
- Let $N_M = \#\{x_i \in R_m\}$, $\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$, and $Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$.
- Define the cost complexity criterion as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|.$$

- For each α, find the subtree $T_\alpha \subset T_0$ that minimizes $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ governs the trade-off between tree size and its goodness of fit to the data.
  - When $\alpha = 0$ we get the full tree.
  - When α is large we get a pruned tree.
- Choose α by cross-validation.

Pruned Tree for Baseball Data





Classification Trees, Part I

- Suppose Y is qualitative with values in $1, 2, \ldots, K$.
- Need to adjust criteria for splitting nodes and pruning tree.
- For the mth node, representing the region R_m containing N_m observations, define

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

  the proportion of class k observations in node m.
- Classify the observations in node m to the majority class,

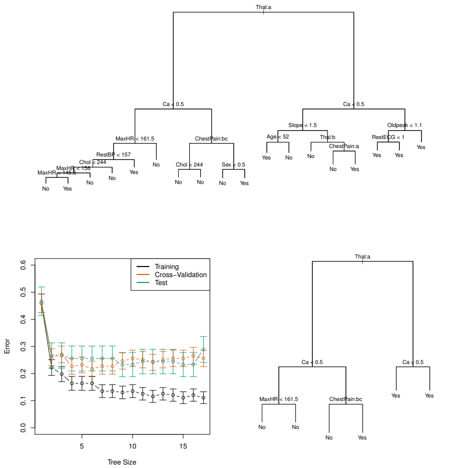
$$k(m) = \mathrm{argmax}_k \hat{p}_{mk}.$$

Classification Trees, Part II

- Three possible measures of node impurity:
  - Misclassification Error: $1 - \hat{p}_{mk(m)}$.
  - Gini Index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.
  - Cross-Entropy or Deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.
- CE and Gini are more sensitive to changes in node probabilities, and hence are a better choice than MCE for growing the tree.
- Misclassification error is typically used for tree pruning.

Classification Tree

Predict presence of heart disease for 303 patients with chest pain.



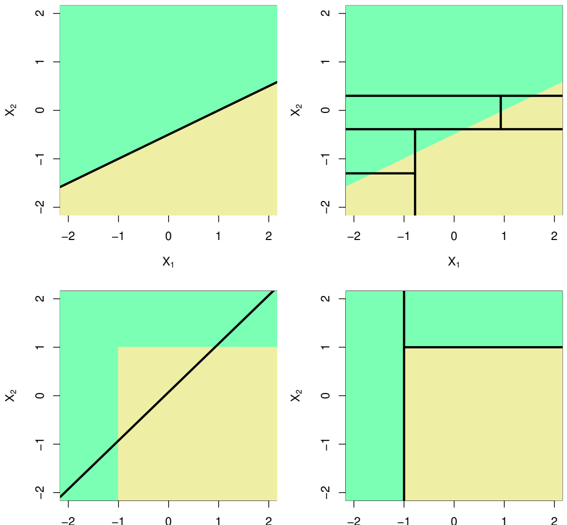
Advantages & Disadvantages of Trees

- Easy to interpret — nice visual!
- Thought to model human decision-making.
- Tends to have lower accuracy than other approaches, like generalized additive models.
- Can suffer from extremely high variance — a small change in data can result in a very different series of splits.
- Difficulty in capturing additive structure: e.g.

$$Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \epsilon.$$

- Accuracy can be improved using bagging and boosting.

Trees Versus Linear Models



Bootstrap Aggregation

- Suppose we fit a model to training data $\mathbf{Z} = \{(x_1, y_1), \dots, (x_N, y_N)\}$, to obtain prediction $\hat{f}(x)$ at x.
- Bootstrap aggregation, or bagging, averages this prediction over many bootstrap samples in order to reduce its variance.
- The bagging estimate is defined as

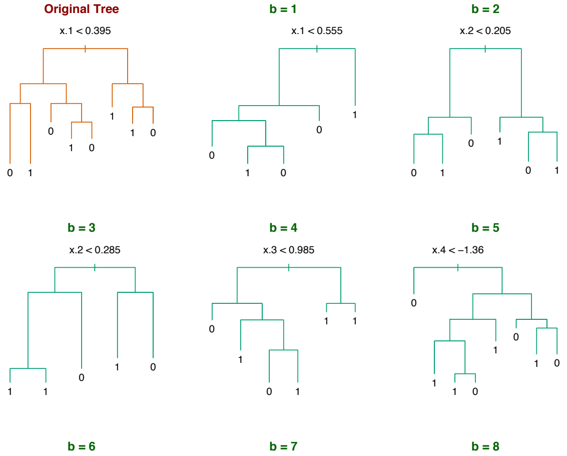
$$\hat{f}_{\mathrm{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x),$$

where the prediction $\hat{f}^{*b}(x)$ was obtained using the bootstrap sample $\mathbf{Z}^{*b}$, $b = 1, \dots, B$.

More on Bagging

- If $\hat{f}(x)$ is a linear function of the data, then you don't accomplish much . . . for B sufficiently large, $\hat{f}_{\mathrm{bag}}(x)$ and $\hat{f}(x)$ will be very similar.
- However, bagging very non-linear fits — like those from a regression tree — can lead to a substantial reduction in variance, and very improved predictions.
- If Y is a qualitative response, then the bagged prediction can be obtained by 1 of 2 options:
  - majority vote: $\mathrm{argmax}_k \hat{f}_{\mathrm{bag}}(x)$; or
  - averaging probabilities across the B bootstrap fits.

Bagged Trees
ESL, Ch 8



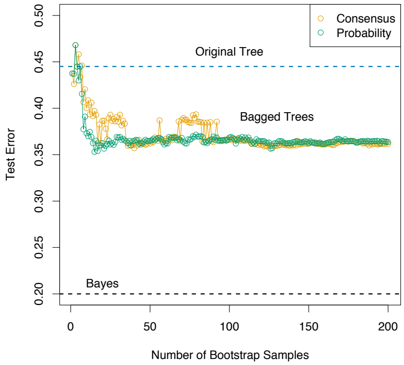


FIGURE 8.10. *Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.*

Bagging

- In regression, bagging can only help you!
- Not so for classification: bagging a good classifier can help, but bagging a bad classifier can hurt.
- Suppose $Y = 1$ for all X, and

$$\hat{Y}(X) = \begin{cases} 1 \text{ with probability } 0.4 \\ 0 \text{ with probability } 0.6 \end{cases} .$$

- Then the misclassification error of $\hat{Y}(X) = 0.6$, and that of the bagged classifier is 1.

Wisdom of Crowds

- If the classifiers are not bad, then bagging can help.
- To see this, a simplified example: the wisdom of crowds.
- Suppose that in a two-class example, $Y(x) = 1$.
- Suppose we have B independent weak learners, $Y^*_b(x)$, each with error rate $e_b = e < 0.5$.
- Let $S_1(x) = \sum_{b=1}^{B} I(Y^*_b(x) = 1)$.
- Weak learners are independent, so $S_1(x) \sim \mathrm{Bin}(B, 1-e)$, and therefore $\Pr(S_1(x) > B/2) \to 1$ as B gets large.
- Bagged predictions are dependent, but the idea still holds.

Wisdom of Crowds

ESL, Ch 8

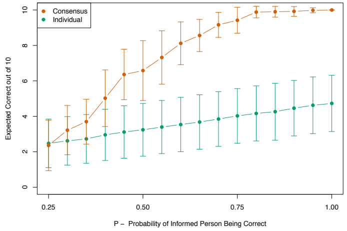


FIGURE 8.11. *Simulated academy awards voting.* 50 *members vote in 10 categories, each with* 4 *nominations. For any category, only* 15 *voters have some knowledge, represented by their probability of selecting the "correct" candidate in that category (so* $P = 0.25$ *means they have no knowledge). For each category, the* 15 *experts are chosen at random from the* 50*. Results show the expected correct (based on* 50 *simulations) for the consensus, as well as for the individuals. The error bars indicate one standard deviation. We see, for example, that if the* 15 *informed for a category have a* 50% *chance of selecting the correct candidate, the consensus doubles the expected performance of an individual.*

The Price of Bagging

Bagging trees can lead to improved predictions, but
...

The Price of Bagging

Bagging trees can lead to improved predictions, but
...interpretability is lost!

Out Of Bag Samples

- No need to do cross-validation to estimate prediction error for bagging.
- Instead, use out of bag (OOB) samples to compute the error.
- That is, for each observation $z_i = (x_i, y_i)$, predict the response *by taking the average of the predictions corresponding to bootstrap samples in which z_i didn't appear*.
- Gives similar results to cross-validation.

Random Forests

- Modification of bagging that builds a large collection of de-correlated trees.
- Very good predictive performance, though limited interpretability.

Variance of an Average

- The average of B uncorrelated r.v.'s, each with variance σ^2, has variance σ^2/B.
- But what about B r.v.'s with pairwise correlation ρ?
  - The average has variance

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

  - As B increases, the variance of the average approaches $\rho\sigma^2$.
- Therefore, bagging can only improve the variance so much if the bootstrapped predictions are correlated.
- Idea behind random forests is to reduce correlation among bootstrapped predictions.

Random Forests

- Perform bagging on trees, but when building each tree, limit yourself to $m \leq p$ variables for each split.
- Then,
  - average the resulting B trees for regression
  - take a majority vote of the resulting B trees for classification.

More On Random Forests

- Excellent prediction; very little tuning required.
- Great “out-of-box” method.
- Loss of interpretability relative to trees.
- Recommendations of the inventors:
  - For regression, take $m = p/3$, and minimum node size 5.
  - For classification, take $m = \sqrt{p}$, and minimum node size 1.

  However, better to treat these as tuning parameters. Select using OOB error estimate.
- Using more trees usually won’t overfit (though this can depend on the size of the trees used). Often a few hundred to a few thousand trees are used.

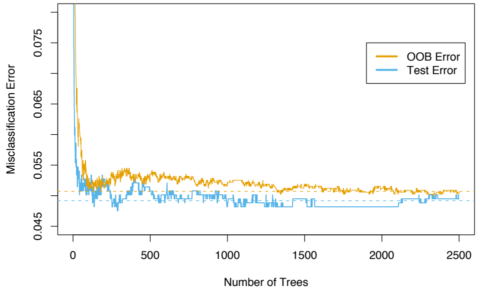


FIGURE 15.4. OOB *error computed on the* spam *training data, compared to the test error computed on the test set.*

Why Don't We Perform Random Forests for Other Types of Estimators?

- We typically perform random forests for trees, but the same idea could be applied more generally.
- But not all estimators can be improved by shaking up the data like this.
- Highly non-linear estimators, like trees, benefit the most.
- Doesn't really help linear estimates.

Interpreting The Results

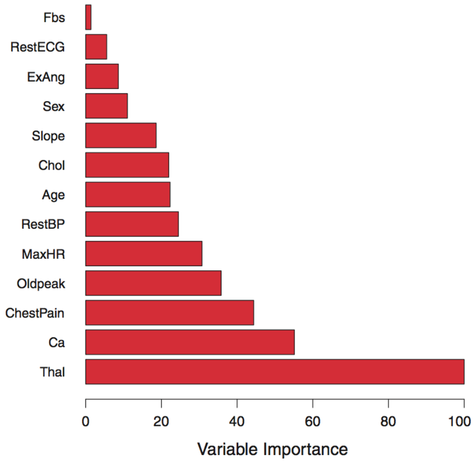
- Unfortunately, bagging and random forest result in models that are not easily interpretable:
  - it is unclear how each variable affects each of the tree models, and how we should interpret the final model

Interpreting The Results

- Unfortunately, bagging and random forest result in models that are not easily interpretable:
  - it is unclear how each variable affects each of the tree models, and how we should interpret the final model
- A very useful tool for gaining insight about individual variables is the variable importance plot
  - The plot shows the total amount of improvement in RSS/Gini Index/entropy resulting from splits over a given predictor, averaged over all B trees

Variable Importance Plot

For the `Heart` data



Let's Try It Out in R!

Chapter 8 R Lab
www.statlearning.com

Boosting

- Boosting is another approach for improving the performance of tree-based methods

Boosting

- Boosting is another approach for improving the performance of tree-based methods
- Like bagging and random forests, boosting uses multiple trees. The main difference is that
  - bagging and random forests aggregate trees based on multiple copies of the data
  - boosting aggregates trees based on a modified version of the same data

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees
- Instead of fitting a single large decision tree, which can potentially overfit the data, in boosting we learn slowly
  - Each tree is small, with just a few terminal nodes
  - Given the current model, we fit a decision tree to the residuals from the previous model as the response
  - We then add this new decision tree into the fitted function and update the residuals

Boosting: Main Idea

- Boosting does not involve bootstrap sampling, and everything is based on a single data set
- In boosting, the trees are grown sequentially: each tree is grown using information from previously grown trees
- Instead of fitting a single large decision tree, which can potentially overfit the data, in boosting we learn slowly
  - Each tree is small, with just a few terminal nodes
  - Given the current model, we fit a decision tree to the residuals from the previous model as the response
  - We then add this new decision tree into the fitted function and update the residuals
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well

Boosting: The Algorithm

Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.    (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$.    (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.    (8.12)
```

Boosting: As a Local Descent Algorithm

Gradient descent logic for minimizing a function:

- Should step in a greedy, locally steepest, descent direction
- As we move, locally steepest direction will change
- Take small enough steps to ensure downhill traversal

Boosting logic

- Should adjust our fitted function, by choosing splits greedily (to most reduce RSS)
- As we adjust fitted function, best greedy split choice changes
- Add in only small amounts of each tree to always use nearly locally-optimal greedy splitting.

Tuning Parameters

- Boosting has 3 tuning parameters:

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B
  - The shrinkage parameter λ: a small positive number that controls the rate of learning (typically 0.01 or 0.001). λ slows down the learning process, allowing more and different shaped trees to attack the residuals. A very small λ may require a very large B to achieve good performance.

Tuning Parameters

- Boosting has 3 tuning parameters:
  - The number of trees B: unlike bagging and random forests, boosting can overfit if B is too large, though this happens slowly. We use cross-validation to select B
  - The shrinkage parameter λ: a small positive number that controls the rate of learning (typically 0.01 or 0.001). λ slows down the learning process, allowing more and different shaped trees to attack the residuals. A very small λ may require a very large B to achieve good performance.
  - The number of splits in each tree d: controls the complexity of the boosted ensemble. Often a single split $d = 1$ works well (each tree is a stump). When $d = 1$, we are fitting an additive model, and in general d is the interaction depth.

Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome

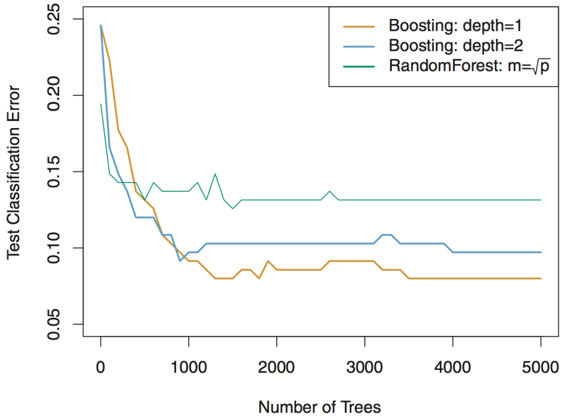
Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome
- The key in boosting is that each individual tree has to be small

Boosting: Some Remarks

- Note that in each step, we fit the model using current residuals, rather than the original outcome
- The key in boosting is that each individual tree has to be small
- A potential advantage of boosting over bagging and random forests is interpretability especially if we use small trees (using stumps gives an additive model)

Comparison on Gene Expression Data



Aggregating Models

Boosting/Bagging/Random Forests are primarily used for aggregating similar models (eg. trees).

What if we want combine a boosted tree model, a linear-Lasso, and a sparse-additive model?

One way to "combine" is just use cross-validation to find the *best* model; and then use that model in the future.

However, a linear combination of the models may perform better than any single model.

Stacking is a method that uses cross-validation to find that combination.

Stacking

The stacking algorithm (using LOO CV):

1. For each modeling procedure, k, and each observation, i:
    1.1 Build a predictive model $\hat{f}_{k,-i}$ using all obs except i
    1.2 Evaluate the model on the left out obs $\hat{f}_{k,-i}(x_i)$
2. Solve the linear regression problem in α:

$$\hat{\alpha}_k, \ldots, \hat{\alpha}_K \leftarrow \text{argmin} \sum_i \left(y_i - \sum_k \alpha_k \hat{f}_{k,-i}(x_i) \right)^2$$

3. Refit our models to all the data, to obtain $\hat{f}_k$ for each k
4. In the future, use the model $\hat{f}(x) \leftarrow \sum_k \hat{\alpha}_k \hat{f}_j(x)$

Illustrating the Algorithm

Suppose we want to combine a boosted tree model, and a lasso model:

- For each observation *i*, we would fit our two models to the rest of the data.
- *fitting the two models* will require additional internal CV for tuning parameter selection (eg. λ in the lasso, tree-depth, etc. for boosting). For each *i*, the choices determined by CV may be different.
- After determining the optimal weights we refit our models (including any internal CV) on all the observations to determine the final models used in our mixture

The tuning parameter choices will likely differ at each stage of CV; and from choices for the final mixture.

Cross-validation in Stacking

- By splitting up the optimization into 2 stages, we avoid overfitting
- Can use k fold CV, or training-test split rather than LOO-CV.
- Can be very computationally intensive — still need to evaluate performance.