STA314, Lecture 9

November 15, 2019
Stanislav Volgushev

Regression trees

Regression trees: Introduction

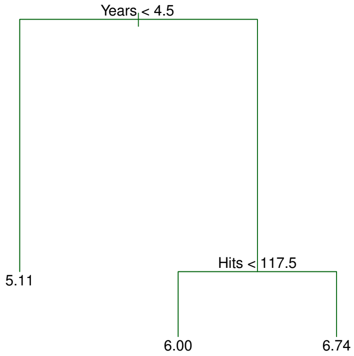
Motivation: build models that allow for non-linear impact of predictors without using basis functions or other simple transformations we used so far.

We will see that 'vanilla' regression trees are very interpretable but typically do not perform well (compared to some methods discussed so far) in terms of prediction.

Regression trees provide the fundamental building block for some of the best 'black box' prediction procedures available today.

Regression trees

Regression trees predict outcomes based on a sequence of simple binary (yes/no) decisions.

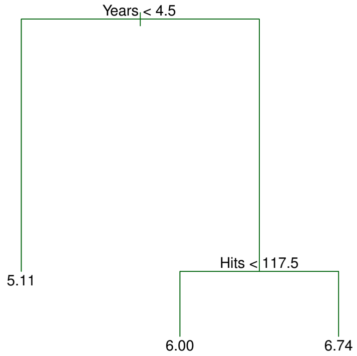


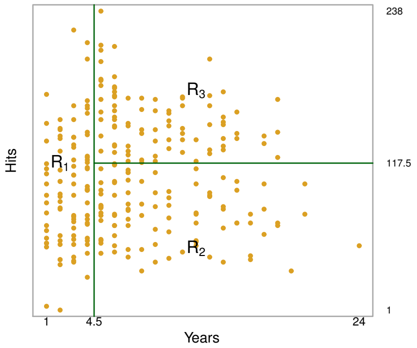
Read from top to bottom, if condition satisfied 'go left'. Predictions from tree above

- Prediction of salary for Years = 3, Hits = 120: 5.11
- Prediction of salary for Years = 5, Hits = 120: 6.74

Regression trees and partitions of predictor space

A regression tree is equivalent to *successively* partitioning the predictor space into rectangles by making cuts parallel to one of the axis, one at a time.





Example of a simple tree and the corresponding partition of the predictor space. 'End points' correspond to rectangles in predictor space. Explanation, additional examples, definition of the notion of *leafs, root, branches, terminal nodes*: see lectures.

How do we grow a tree?

Or: how do we find a good way to partition the predictor space? For trees with many splits, it is computationally impossible to maximize over all possible trees to find the one that minimizes the RSS. One simple approach: *recursive binary splitting*.

---

1. Consider regions $R_1(j,s) := \{X \in \mathbb{R}^d : x_j < s\}$, $R_2(j,s) := \{X \in \mathbb{R}^d : x_j \geq s\}$ $(j = 1, ..., d)$ and find $j, s \in \mathbb{R}$ such that

$$RSS(j,s) := \sum_{i:X_i \in R_1(j,s)} \left(y_i - \bar{y}_{R_1(j,s)}\right)^2 + \sum_{i:X_i \in R_1(j,s)} \left(y_i - \bar{y}_{R_2(j,s)}\right)^2$$

   is minimized. Here $\bar{y}_R$ is mean of $\{y_i : X_i \in R\}$.

2. For each of the two regions from step 1, find the split that minimizes the resulting RSS. Split the region that leads to larger reduction in RSS. This gives 3 regions.

   $\vdots$

K. For each of the K regions from step $K-1$, find the best split. Among those K splits, select the one that leads to largest reduction in RSS.

See blackboard in lectures for additional explanations.

Pruning trees

Growing trees with too many leafs will lead to high variance. A typical approach to improve this is by first growing a large tree and then *pruning* it.

Start with a large tree T_0. A tree T is a *sub-tree* of T_0 if it can be obtained by turning some of the internal nodes of T_0 into terminal nodes (i.e. by pruning away branches).

A computationally efficient way to do pruning is to consider *cost complexity pruning*, i.e. minimize over sub-trees $T \subseteq T_0$

$$T^{\alpha} = \arg \min_{T \subseteq T_0} \sum_{m=1}^{|T|} \sum_{i : X_i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T|$$

where $|T|$ is number of leafs, $R_1, ..., R_{|T|}$ correspond to leafs and $\bar{y}_{R_m}$ mean of response with predictors in R_m.

- The first part is the training error of tree T.
- The second term measures the complexity of the tree.
- α is a tuning parameter that can be selected by cross-validation.

Food for thought: why do we grow large trees and later prune? Why not just grow a smaller tree directly?

Pruning trees II

$$T^\alpha = \arg\min_{T \subset T_0} \sum_{m=1}^{|T|} \sum_{i:X_i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T|$$

Easy to show (exercise): *if tree T_0 was grown by recursive binary splitting*, increasing α will lead to a sequence of nested trees, i.e for $\alpha_1 > \alpha_2$ we have $T^{\alpha_2} \subseteq T^{\alpha_1}$.

Cross-validation approach to selecting α for growing a tree:

---

1. Split the data into K folds of roughly equal size.
2. For $k = 1, ..., K$ take out k'th fold and on remaining data
   2.1 Grow a large tree $T_0(k)$ by binary splitting.
   2.2 For each α find $T^\alpha(k)$ as above.
   2.3 Compute $cv_k(\alpha)$ as MSE of predicting data in k'th fold from tree $T^\alpha(k)$.
3. Select α that minimizes $\sum_k cv_k(\alpha)$. Return this α.

In the final step, we grow a large tree on the complete data set and return T^α obtained from that with α selected by cross-validation.

Regression trees: closing remarks

- Regression trees are very easy to interpret and explain, in particular 'small' regression trees.
- Predictive performance is typically not that great. Part of the trouble: high variance, location of splits depends on data a lot.
- Pruning helps a bit, but does not really lead to performance that is competitive with generalized additive models. Ways to improve that: next.

Some R examples - file `Sta314-Lecture09-Trees`

Bagging

Bagging

Bagging is short for **B**ootstrap **Agg**regat**ing**.

Definition Given a sample $(X_1, y_1), ..., (X_n, y_n)$, a *bootstrap sample* is obtained by randomly sampling n of the pairs with replacement.

- If you heard of the bootstrap before, it probably was related to inference or variance estimation.
- Motivation: we would like to draw from the population, but we can't. So why don't we draw from the sample, pretending it is the population? One can prove that often this actually works!
- We will be using the bootstrap in a different way than usually, but the main motivation that we are generating new samples from the population (or something that is close) remains.

Bagging regression trees

1. For $b = 1, ..., B$ repeat the following
   1.1 Draw a bootstrap sample $(X_1^{(b)}, y_1^{(b)}), ..., (X_n^{(b)}, y_n^{(b)})$ from the original data.

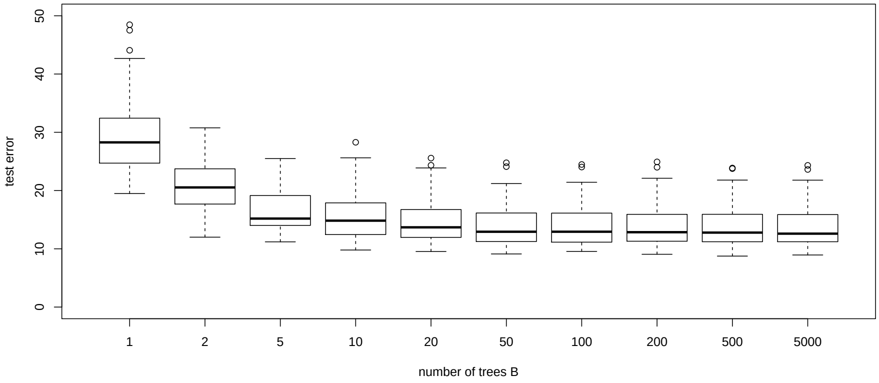
   1.2 Using the bootstrap sample, grow a large regression tree $T^{(b)}$ (do not prune).

2. For new predictor X_0, each of the trees $T^{(1)}, ..., T^{(B)}$ from step 1 (*a bag of trees*, hence bagging) gives a prediction $\hat{y}^{(1)}, ..., \hat{y}^{(B)}$. Use the average for prediction, i.e. the predicted response is

$$\hat{y} = B^{-1} \sum_{b=1}^{B} \hat{y}^{(b)}.$$

- Motivation: each individual tree has high variance. Taking averages leads to a reduction of variance, so the above procedure should take care of high variance of large trees without using pruning.

Bagging regression trees: performance for Boston data



Plot above shows test MSE vs number of trees B

- Choosing the number of trees B: no overfitting for large numbers! Except for computation time, there is no disadvantage in choosing larger values of B. In this example $B = 100$ would have been enough.

Bagging regression trees: variable importance measures I

Compared to trees, bagging leads to substantial improvements in prediction performance but sacrifices interpretation. A single tree is easy to interpret, but an average over many trees is not. Still, one interesting aspect of bagging is that it allows to obtain *variable importance measures* which give some information about which variables are useful for predicting the response.

There are two versions of measuring variable importance implemented in `R`. To describe the first version, we begin by measuring variable importance in a single tree.

---

1. Each internal node of a tree T corresponds to splitting along one predictor. For internal node $m = 1, ..., M$, denote index of this predictor by i_m.
2. For $m = 1, ..., M$, denote by Δ_m the total reduction in RSS from splitting that node into two regions along predictor i_m.
3. The total reduction of RSS due to using predictor $X_{i,k}$ is given by

$$\sum_{m:i_m=k} \Delta_m$$

See blackboard for more detailed explanation in pictures.

Bagging regression trees: variable importance measures II

1. For each tree in bag of trees $T^{(1)}, ..., T^{(B)}$ and each predictor $1, ..., p$, compute total reduction of RSS resulting from that predictor using algorithm on previous slide. Denote those reductions by $\{E^{(b)}(k)\}_{b=1,\ldots,B,k=1,\ldots,p}$.
2. For each predictor $1, ..., p$, compute averaged (over trees in bag) RSS improvement
$$e_k = \frac{1}{B}\sum_{b=1}^{B} E^{(b)}(k)$$
3. The output is e_k for $k = 1, ..., p$. Larger values of e_k correspond to larger reduction in RSS and thus 'more important' variables.

Interpretation: if splitting a predictor leads to large reduction in RSS, this means that predictor is informative about the response and using this predictor is helpful to reduce prediction error.

Bagging regression trees: permutation variable importance

An alternative to the variable importance measure defined previously is based on *permutation variable importance*

---

1. For $k = 1, ..., p$ and $b = 1, ..., B$ do the following steps

   1.1 Create new predictors $Z_i = (x_{i,1}, .., x_{i,k-1}, \tilde{x}_{i,k}, x_{i,k+1}, ..., x_{i,p})^\top$ where $\tilde{x}_{1,k}, ..., \tilde{x}_{n,k}$ is a random permutation of $x_{1,k}, ..., x_{n,k}$.

   1.2 For tree $T^{(b)}$ let $(X_{i_1}, y_{i_1}), ..., (X_{i_{n_b}}, y_{i_{n,b}})$ denote the variables that were not used to grow that tree (*out-of-bag observations*). Compute

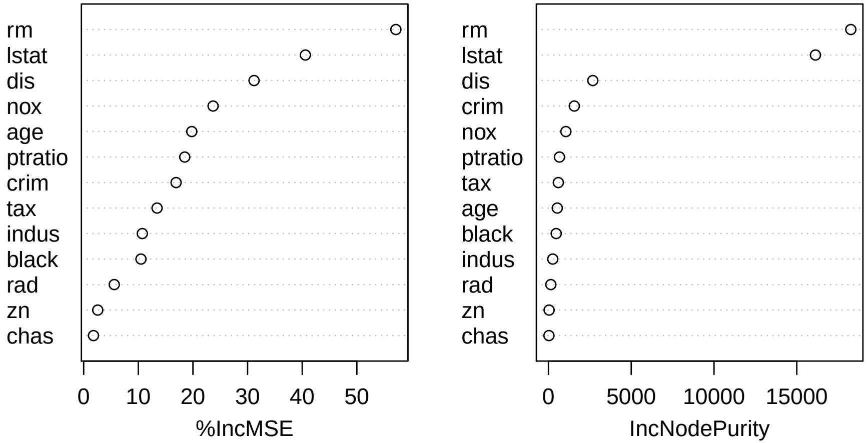
   $$RSS_k^{(b)} := \sum_{j=1}^{n_b} (\hat{f}^{(b)}(X_{i_j}) - y_{i_j})^2, \quad \widetilde{RSS}_k^{(b)} := \sum_{j=1}^{n_b} (\hat{f}^{(b)}(Z_{i_j}) - y_{i_j})^2$$

   where $\hat{f}^{(b)}$ is the regression function corresponding to tree $T^{(b)}$.

2. Output for $k = 1, ..., p$: $\sum_{b=1}^{B} (\widetilde{RSS}_k^{(b)} - RSS_k^{(b)}) / \sum_{b=1}^{B} RSS_k^{(b)}$.

Interpretation: by permuting one of the predictors, we destroy any influence this predictor has on the response. If this leads to a large increase in RSS, this means that the predictor was useful.

Variable importance example: Boston data



Example of R output for importance of variables in predicting median housing prices for Boston data. Left: 'permutation variable importance'. Right: first option we discussed. Some sources recommend the permutation variable importance more highly. Recent research suggests that there are issues with both methods.

Random forests

Improving bagged regression: Random forests

The idea of random forests is very similar to bagging, but there is one modification. At each step of growing a tree, use only $m \leq p$ *randomly selected* predictors (m is a tuning parameter).

---

1. For $b = 1, ..., B$ repeat the following
   1.1 Draw a bootstrap sample $(X_1^{(b)}, y_1^{(b)}), ..., (X_n^{(b)}, y_n^{(b)})$ from the original data.

   1.2 Using the bootstrap sample, grow a large regression tree $T^{(b)}$ using only m randomly selected predictors at each split (do not prune).
2. For new predictor X_0, each of the trees $T^{(1)}, ..., T^{(B)}$ from step 1 gives a prediction $\hat{y}^{(1)}, ..., \hat{y}^{(B)}$. Use the average for prediction, i.e. the new prediction is

$$\hat{y} = B^{-1} \sum_{b=1}^{B} \hat{y}^{(b)}.$$

See example on blackboard for additional explanation.

Improving bagged regression: Random forests

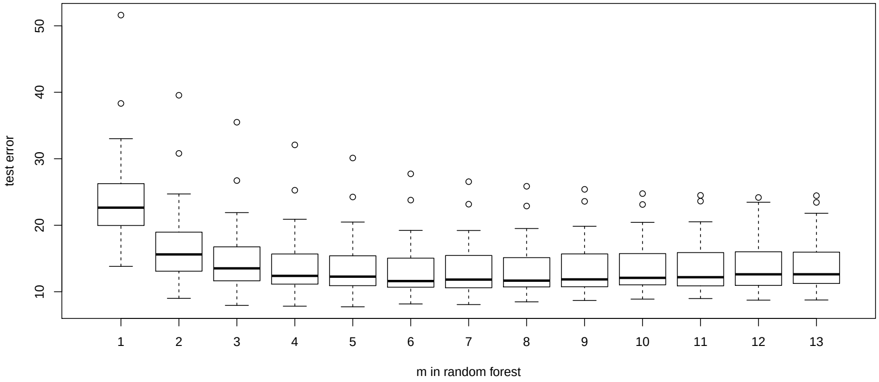
Using only some of the predictors in each step can have several advantages.

- **Weaker predictors also get a chance**: if one of the predictors has strong relation to the response, it is likely that most splits when growing trees in bagging will be along this predictor. In random forests, if $m < p$, we sometimes split without using this predictor. If other predictors are informative they will also be used for growing trees.

- **Reducing dependence between trees**: the main motivation for bagging is to reduce variance by averaging over trees. If only a subset of predictors is allowed to enter each tree, this makes trees even more independent.

There is no universal rule for choosing m. A rule of thumb for regression is $m \approx p/3$, but in general it is a good idea to consider cross-validation.

Random forests: impact of m for Boston data

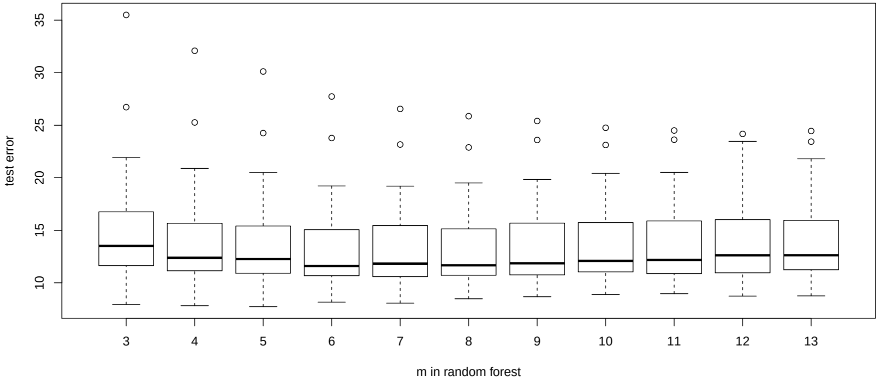


Above: plot of test error of random forest against number of trees for `Boston` data set.

- Performance for $m = 1$ is not that great.
- Best performance seems to be in the middle. Take a closer look on next slide.

Question: Where on the above plot is the performance of bagging?

Random forests: impact of m for Boston data



Above: plot of test error of random forest against number of trees for `Boston` data set.

- Overall, $m = 6$ seems to perform best (not by a large margin). A bit better compared to $m = 13$ which uses all predictors.

Some R examples - file `Sta314-Lecture09-Bagging-and-Forests`

Boosting

Motivation

In last lecture, we discussed how regression trees can be (substantially) improved by means of randomization, leading to *bagging* and *random forests*.

Now: alternative method that can also lead to substantial improvements of single trees (and other methods) called *boosting*.

Two (distinct!) fundamental ideas:

1. *Fit models to residuals*: once we learn a regression function $\hat{f}$, we have residuals $\hat{\varepsilon}_i = y_i - \hat{f}(X_i)$. If $\hat{f}$ did not capture all dependence between y_i and X_i, it might be worth to fit another model using residuals $\hat{\varepsilon}_i$ as response.

2. *Slow learning*: instead of attempting to learn complete information form data in one step, learn a bit, then a bit more etc. Will be made precise in examples later.

Basic boosting algorithm for regression trees

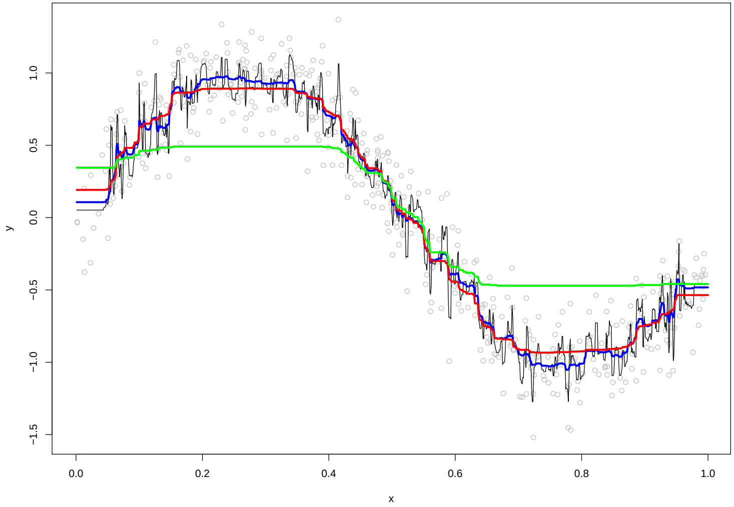
There are 3 tuning parameters: number of trees B, shrinkage parameter $\lambda \in (0, 1]$, number of internal nodes d in each tree.

---

1. Set $\hat{f}^{(0)}(x) \equiv 0$ and $r_i^{(0)} = y_i$. For $b = 1, ..., B$, repeat the following steps.
    1.1 Grow a tree with d internal nodes using $(X_1, r_1^{(b-1)}), ..., (X_n, r_n^{(b-1)})$ as data. Here X_i are the predictors and $r_i^{(b-1)}$ are treated as response. Denote corresponding regression function by $\hat{g}$.
    1.2 Update $\hat{f}(x)$ using the rule
    $$\hat{f}^{(b)}(x) = \hat{f}^{(b-1)}(x) + \lambda \hat{g}(x)$$
    1.3 Define new residuals $r_i^{(b)} = y_i - \hat{f}^{(b)}(X_i)$.
2. The output is $\hat{f}^{(B)}$.

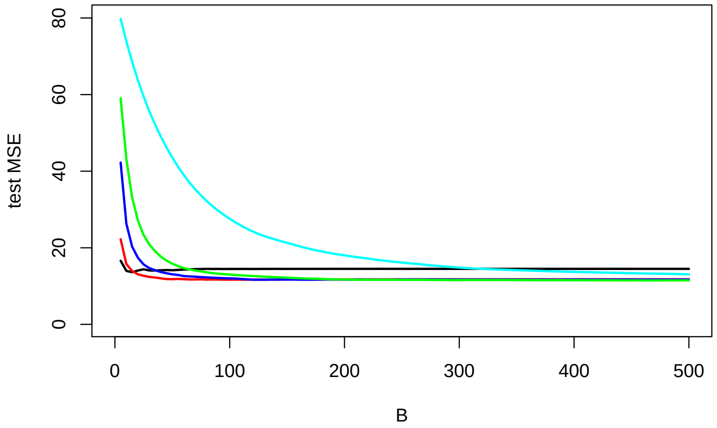
- Although the idea of adding $\lambda \hat{g}(x)$ instead of $\hat{g}(x)$ seems surprising, this is what really makes boosting work well.
- R example: `STA314-Lecture10-BoostingExample`

Large B can lead to overfitting, small B not good either



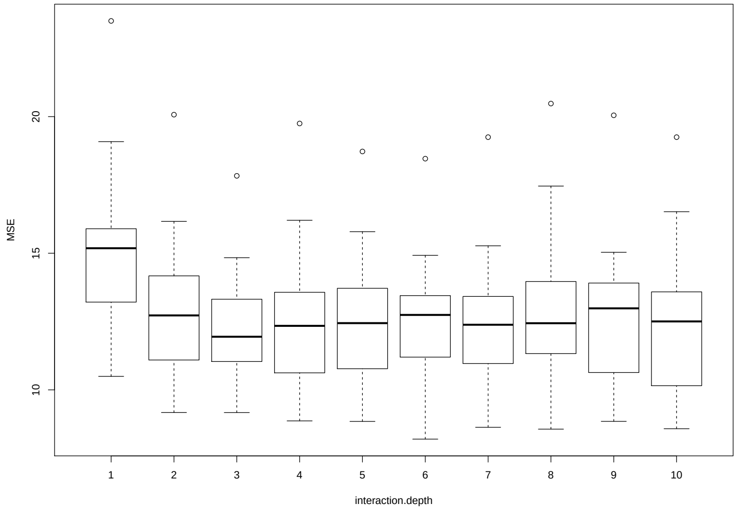
$\lambda = 0.01$, $d = 2$. Green: $B = 100$, Red: $B = 500$, Blue: $B = 2500$, Black: $B = 50000$

Basic boosting for Boston data: effect of λ for one split of data set



$d = 8$ on Boston data, different colours correspond to $\lambda \in \{0.5, 0.25, 0.1, 0.05, 0.01\}$. Which is which?

Basic boosting for Boston data: effect of d for one split of data set



The effect of d (number of terminal nodes) for the Boston data. $\lambda = 0.1$, number of trees selected by 5-fold cross-validation.

Basic boosting: selecting tuning parameters

Unfortunately, there is no universal rule for choosing tuning parameters which works universally well. Some general comments and rules of thumb:

- In contrast to random forests, choosing large values of B can lead to worse performance. Should be selected by cross-validation.
- In contrast to random forest, it is not helpful to grow large trees. A rule of thumb recommended in some places is $d \approx 4 - 10$.
- Usually it is a good idea to choose $\lambda < 1$. Recommended range: λ between 0.001 and 0.1. Usually no harm in choosing smaller λ except for computation time.
- Smaller values of λ require choosing larger values of B.
- Larger values of B are more expensive from a computational point of view.

Boosting algorithm for regression trees with randomization

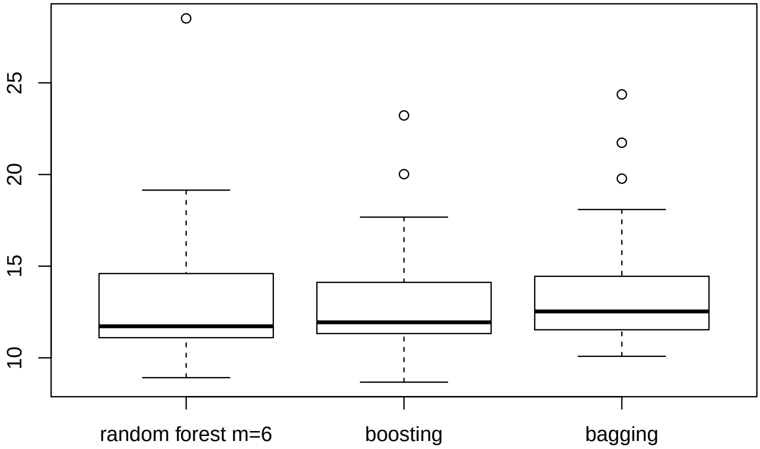
Now there are 4 tuning parameters: number of trees B, shrinkage parameter $\lambda \in (0, 1]$, number of internal nodes d in each tree, sub-sample proportion p.

---

1. Set $\hat{f}^{(0)}(x) \equiv 0$ and $r_i^{(0)} = y_i$. For $b = 1, ..., B$, repeat the following steps.
   1.1 Randomly sample pn observations from $(X_1, r_1^{(b-1)}), ..., (X_n, r_n^{(b-1)})$.
   1.2 Grow a tree with d internal nodes using the sample from step 1 as data. Denote corresponding regression function by $\hat{g}$.
   1.3 Update $\hat{f}(x)$ using the rule
   $$\hat{f}^{(b)}(x) = \hat{f}^{(b-1)}(x) + \lambda \hat{g}(x)$$
   1.4 Define new residuals $r_i^{(b)} = y_i - \hat{f}^{(b)}(X_i)$.
2. The output is $\hat{f}^{(B)}$.

- Recommended range for p: $0.2 - 0.5$, although again this is just a rule of thumb. Usually < 1 is better.
- Smaller p can speed up computation.
- Recommendation for choosing B, d, λ as before.

Performance on Boston data



Boosting: $\lambda = 0.002$, $d = 8$, $p = 0.5$, B based on cross-validation.

Some useful functions in package `gbm` I

The `plot.gbm` function in package `gbm` can use the output of the function `gbm` to provide plots of the *average dependence* of response on individual variables while averaging over values of other variables.

More precisely: denote by $\hat{f}$ the function that boosting learned from data. Then `plot.gbm` provides plot of the function

$$x \mapsto \hat{g}(x) := \frac{1}{n} \sum_{i=1}^{n} \hat{f}(X_{i,1}, X_{i,2}, ..., X_{i,j-1}, x, X_{i,j+1}, ..., X_{i,d}),$$

i.e. holding the j'th entry component fixed and averaging over all other components based on the sample. Similar if x is two-dimensional.

See R example `STA314-Lecture09-BoostingBoston`

Some useful functions in package `gbm` II

The `summary` function in package `gbm` provides ranking of variable importance. Similar idea as for bagging, but scaled version (so that all values sum up to one) of the first variable importance measure we discussed.

Setting the option `cv.folds=5` (5 can be replaced by other number) when running `gbm` and running `gbm.perf(boost.boston,method="cv")` can be used to select number of trees based on cross-validation.

See R example `STA314-Lecture09-BoostingBoston`