STATISTICS 452/652: Statistical Learning and Prediction

October 24, 2020

Lecture 16: Ensembles3 - Boosting

(Reading: Section 8.2.3)

1 Goals of lecture

- We introduce one last ensemble method
- Like bagging, it can be applied to any existing prediction machine, but works especially well with regression trees
- However, its construction is not at all like bagging

2 Boosting

- Bagging/random forest builds an ensemble using independent resamples from the original data.
  - This is a "parallel" process, because we could estimate all trees simultaneously:
    - Generate all B resamples at once
    - Fit each tree on B different computers or computer cores
    - Average them together at the end.
  - In particular, the order in which the trees are fitted doesn't matter.
  - Also, we want each tree to explain the true structure as fully as possible
- *Boosting fits trees in sequence*
  - Each tree is fitted to the residuals left by all the trees that have been fitted previously
  - The idea is that each tree should try to explain *only a little bit* of the true structure

- The next tree in sequence tries to explain a little bit of what is left behind by previous trees in the sequence
- Trees need to be relatively small
  - No one tree looks very much like the true structure
  - We use *many, many trees* to construct a potentially very complex structure
- Needless to say, there is huge potential to overfit, so need to use heavy shrinkage on trees to create picture slowly
- Basically this is like impressionist art or how any computer image is composed of tiny pixels of colour
- Construct a complex surface from an ensemble of weak learners
  - This general concept is very powerful

2.1 Boosting algorithm

The algorithm cycles through three steps in sequence: create residuals, fit a tree, and update the function. Start by choosing values for three parameters, all of which should be tuned:

- The number of trees in the sequence, denoted as B
- The size of trees to be fit at each step, denoted by d. There are several ways to measure "size":
  - One is number of splits made, where each added split creates 1 added terminal node ($d + 1$ terminal nodes in total).
  - One is the "depth" of the trees to be used, meaning how many *levels* of splits will be done below the root node. Then there are up to 2^d total terminal nodes. (This is what the R function `gbm()` uses.)
  - Constraints on terminal node sizes may limit the potential sizes of trees, and can be a minor tuning parameter
  - It doesn't particularly matter which definition is used. You just need to understand is done in the implementation you are using.
- The shrinkage parameter that controls how much we allow the updated function to change at each iteration of the cycle, denoted by λ (see algorithm).

Once these choices are made, we proceed with the algorithm (warning: notation is different from ISLR):

1. We start the algorithm by setting the initial $\hat{f}^{(0)}(X)$ to be the simplest possible prediction function, the sample mean of the training data $\bar{y}$.
2. Compute residuals for each observation in the training data, $r_i^{(0)} = y_i - \hat{f}^{(0)}(x_i) = y_i - \bar{y}$, $i = 1, \ldots, n$.
3. Now start the loop over the iterative cycle. For $b = 1, \ldots, B$:

(a) Fit a tree of size d to the data using the usual RSS-based splitting algorithm from Lecture 13. Call it $T_b(X)$

i. Terminal node means measure how far that node wants to be from the overall mean $\bar{y}$

(b) Update the prediction function by adding in a regularized amount of $T_b(X)$ into the current function $\hat{f}^{(b-1)}(X)$

$$\hat{f}^{(b)}(x_i) = \hat{f}^{(b-1)}(x_i) + \lambda T_b(x_i)$$

(c) Compute a new residual, $e_i^{(b)} = y_i - \hat{f}^{(b)}(x_i) = e_i^{(b-1)} - \lambda T_b(x_i)$

(d) Increment b

4. The final prediction function is $\hat{f}^{(B)}(X)$

2.2 Tuning the Machine

We see three tuning parameters described in the algorithm:

- Number of trees B
  - Often need thousands
  - Occasionally tens of thousands
  - Depends on tree size and shrinkage
    - The weaker the learners, $\lambda T_{b(X)}$, the more trees are needed
  - `gbm()` default is 100, which is almost *never* enough.
  - *Maybe* try something like 1000, 3000, 5000, 10000???
  - Larger values increase computation time, so may struggle to do larger numbers in large grids
- Tree size d
  - Larger d makes stronger learners, so don't want it "too big"
  - Each level of a tree corresponds to an order of potential interaction
    - $d = 1$ forbids any interaction, so limited use to non-interactive structures! (but this is the default in `gbm()`)
    - $d = 2$ allows 2-way interaction, $d = 3$ allows 3-variable interaction, etc.
    - So can choose values of d that represent possible complexities of surface
    - I've never needed $d > 8$ or so.
    - Maybe try $d = 1, 2, 4, 8$ and see what happens???
- Learning rate or shrinkage λ
  - Usually want to build surface slowly, so want to multiply tree by something much less than 1
  - Common to try "different orders of magnitude", $\lambda = 0.0001, 0.001, 0.01, 0.1$
  - `gbm()` default is 0.001

Other parameters

There are additional parameters in the algorithm that can sometimes make a difference in performance on test data:

- The minimum number of observations in a terminal node (`gbm()` default is 10)
  - Especially if training n is relatively small, setting this to a small value like 5 (or smaller??) may help.
  - *Occasionally* a larger value can help as well, if there are enough data to support it.
- There is a variant of boosting that adds resampling to each iteration
  - After computing residuals, but before fitting new tree
  - Fit each new tree to a different fraction of the data
  - Keeps tree from overfitting, but slows learning,
    - Need more trees if this is set lower
  - `gbm()` default is 0.5—50% of data are used in each tree
    - ISLR does not explain this!
  - Usually surface is less sensitive to this than other parameters, but can make small difference sometimes.
    - I've only ever tuned between 0.5 and 0.9
  - Side effect: Some predicted values in each tree are out of bag
    - Can create a version of OOB error and use it to tune B within one run!
    - See whether a smaller number of trees would work.
    - But for some reason seems to underestimate best number; maybe double its recommendation???

Practically speaking, tuning ALL FIVE of these parameters in a giant grid may take too long.

- Can just tune main three parameters
- Can use a smaller grid of 2-3 values each
  - But doing this increased the chances that best result is on a boundary
  - When this happens, often want to explore further in that direction

Example: Boosting on Prostate Data (L16 - Boosting Prostate.R)

The package `gbm` ("gradient boosted models") can do boosted regression trees and a variety of other boosted models. Lots of tuning parameters (B, d, λ,and others) can all be specified. It is worth playing around with them, because combinations of them matter. The `gbm()` function has built-in CV tuning on B if you want (`cv.folds=`).[1] However, it's easier to use the internal OOB error, which is available without any extra calculation. The `gbm.perf()` function plots the training and OOB error and estimates the optimal B, for the given values of the other tuning parameters.[2]

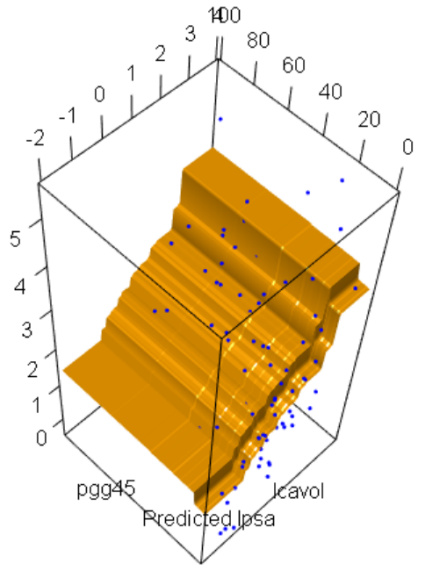
There is a variable importance measure based on the mean reduction in training error, like in a random forest. It seems like it should be possible to implement the permutation method for variable importance, but I haven't seen it in `gbm()`. There is also a function that can add trees to an existing `gbm()` object, which is faster than than starting over if the value of B is too small. You can use `gbm.more()` on the previously fitted object, as long as the original fit includes the option `keep.data=TRUE`.

I use boosting on the Prostate data using just two variables, `lcavol` and `pgg45`, and a few different combinations of tuning parameters. I only show one surface below, in Figure 1 because others are very similar. We see a very similar pattern to what other methods have shown us: increasing with respect to `lcavol`, and only a small effect of `pgg45` near 0. It turns out that this data set does not need large trees, because there is little interaction.

[1] Beware: it tries to use multiple cores automatically, so may be slowed by additional parallelization. Can also set `n.cores=1` to prevent this. There also may be a problem on Macs. See https://github.com/gbm-developers/gbm/issues/53.

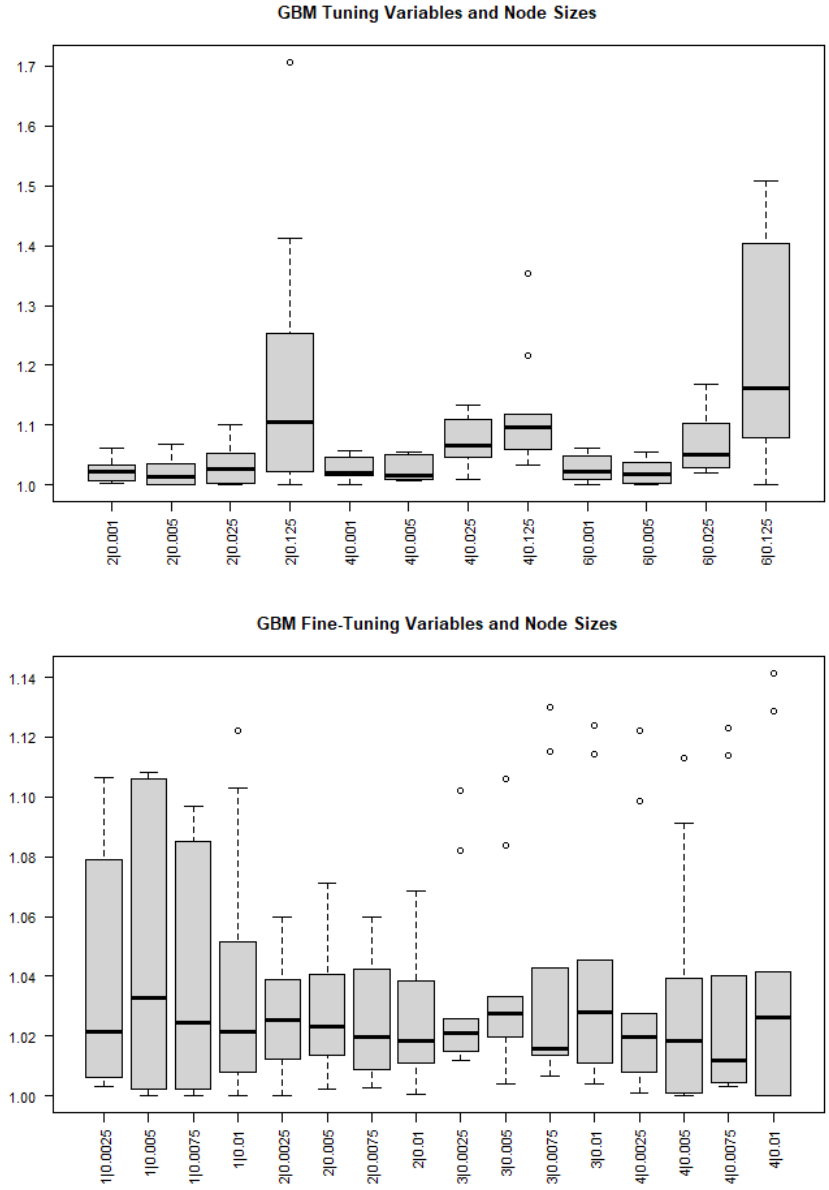
[2] The OOB error is not exactly the same as in RF, due to the sequential nature of boosting. The OOB measurement is an estimate of how much each tree improves OOB error. You aim to stop when that number reaches 0, but it varies quite a bit from one tree to the next, so the process is a little complicated. A warning is printed suggesting that it underestimates the optimal number of trees. I have seen examples where CV estimates that roughly twice as many treed are needed, so as *a complete hack* I might just take $2\times$ the OOB estimate.

Figure 1: Fitted boosted tree surface for 2 variables, `lcavol` and `pgg45` in Prostate data.



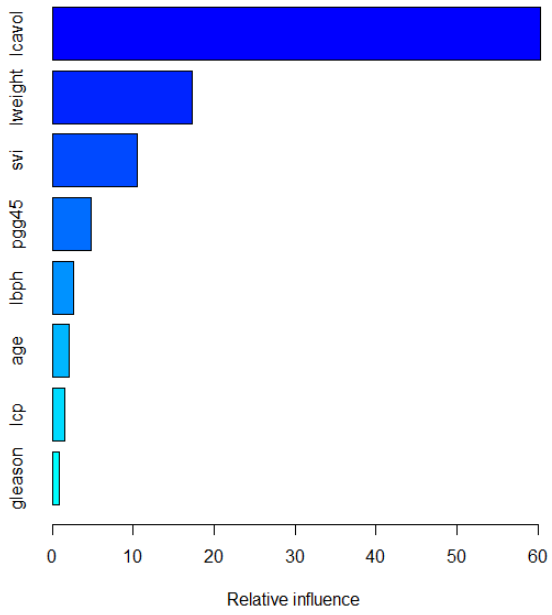
Next I use all 8 explanatory variables and tune the booster. I use the OOB error to estimate the number of trees and tune only on the other two most important variables, d and λ. I use 2 replicates of 5-fold CV. The relative root-MSPE boxplots results are in Figure 2. In this example, my optimal parameters are around $d = 2, \lambda = 0.005$ ("2|.005"), with $B = 638$, the average selected by doubling the OOB estimate across the 10 total folds. Since tuning did not take too long, I tine-tuned in a new grid around these values as shown in the program. The boxplots from the second grid are also shown in Figure 2. There are lots of similar results here, but anything with $d = 2$ seems to have a lower box. I'll take $\lambda = .0075$ because it has the smallest mean (not shown). For this combination, the average optimal B is about 500.

Figure 2: Relative root-MSPE plots from boosting on Prostate data. Top: Initial grid, Bottom: Fine-tuned grid.



I then refit the boosted trees using these optimal parameters and computed the variable importance measures. In this case they are presented as proportions of total RSS reduction due to each variable. See Figure 3. Clearly, most of the variability in the ensemble is explained by splits on `lcavol`, as other methods have indicated. We do see that `lweight` is next as expected, followed by `svi`.

Figure 3: Relative variable importance for tuned boosting with $d = 2, \lambda = 0.0075, B = 500$.



Note that I have not tried to tune the subset size for each tree in the sequence, fixing `bag.fraction=0.80`. In larger data sets, the default value of 0.5 might work, but here the sample size is so small that having only 50% of the data would not allow fitting some of the larger tree sizes.

---

3 What to take away from this

1. Boosting is another excellent tool for prediction
   (a) With RF and NN, among the top methods we have
2. Tuning is necessary and potentially very complicated
   (a) 3 parameters NEED to be tuned, and two more can add some improvement if tuned.

4 Exercises

Application

Refer to the Air Quality data described previously, and the analyses we have done with `Ozone` as the response variable, and the five explanatory variables (including the two engineered features).

1. Use boosting to model the relationship between `Ozone` and all ONLY THE THREE ORIGINAL VARIABLES. Tune on an initial grid of $\lambda = 0.001, 0.005, 0.025, 0.125$ and $d = 2, 4, 6$, and select trees optimally using twice the number suggested by OOB error. Use two reps of 5-fold CV.

   (a) **Report the mean root-MSPE for each combination**

   (b) **Show relative root-MSPE boxplots**

   (c) **What combination do you prefer? Explain.**

   (d) **If you were to do more tuning, what values might you consider? Explain.**

2. Add tuned versions of boosting into the CV analysis comparing previous methods. Use the same grid used above for tuning. ALSO, make two versions of boosting: one with only the three original variables, and one with the two extra engineered features.

   (a) **For the each of the 10 folds, report the chosen best values of the boosting tuning parameters in both ensembles, including numbers of trees.** (I expect that they will vary from one fold to the next).

   (b) **Compare the mean MSPE from the best-tuned booted ensemble using 3 variables to the one with 5 variables. Does it help to have the extra variables?**

   (c) **ADD two tuned boosting ensembles to the relative MSPE boxplots made previously.**

      i. **Comment on how well they perform compared to other methods.**