Optimal Subtrees

Stephen Vardeman
Analytics Iowa LLC
ISU Statistics and IMSE

Seeking optimal tree complexity

It is, of course, possible to continue splitting rectangles/adding branches to a tree until every distinct **x** in the training set has its own rectangle. But that is not helpful in a practical sense, in that it corresponds to a very "low bias/high variance"/complex predictor. So how does one find a tree of appropriate size? How does one choose a size at which to stop growing a tree, or more generally, prune a large tree back to a good size? (This latter is more general in that pruning a tree can produce subtrees not met in a sequence of trees as built up to a final one.) It turns out that it is possible to efficiently find a nested sequence of "optimal" subtrees of a large tree, and that methodology can in turn be used in cross-validation.

Penalizing training error

For T a subtree of some fixed large tree T_0 (e.g. grown until the cell with the fewest training $\mathbf{x}_i$ contains 5 or less such points or in classification contexts until the training error is 0) write

$$E(T) = N \cdot \overline{\text{err}} = \sum_{i=1}^{N} L\left(\hat{f}(\mathbf{x}_i), y_i\right)$$

(the total training error for the tree predictor based on T). For $\alpha > 0$ define the quantity

$$C_\alpha(T) = |T| + \alpha \cdot E(T)$$

(for, in the obvious way, $|T|$ the number of final nodes in the candidate tree).

Alternate parameterization and best tree

It is, of course, equivalent to consider a quantity $E(T) + \lambda |T|$ for a $\lambda > 0$ in notation more like that used in other contexts like the ridge regression problem. Using α as a weight on $E(T)$ to define C_α, is equivalent to using $\lambda = 1/\alpha$ as a weight on $|T|$. The penalized form $E(T) + \lambda |T|$ is probably more often used (at least for user interface) in software implementations. The present C_α is more natural in the context of the following development of optimal subtrees.

Write

$$T(\alpha) = \underset{\text{subtrees } T}{\arg\min}\, C_\alpha(T)$$

and let $\hat{f}_\alpha$ be the corresponding predictor.

Search

The question of how to find a subtree $T(\alpha)$ optimizing $C_\alpha(T)$ without making an exhaustive search over subtrees for every different value of α has a workable answer. There is a relatively small number of nested candidate subtrees that are the only ones that are possible minimizers of $C_\alpha(T)$, and as α decreases one moves through that nested sequence of subtrees from the largest/original tree to the smallest.

First candidate sub-tree

One may quickly search over all "pruned" versions of T_0 (subtrees T created by removing a node where there is a fork and all branches that follow below it) and find the one with minimum

$$\frac{E(T) - E(T_0)}{|T_0| - |T|}$$

(This is the per node–of the lopped off branch of the first tree–increase in E.) Call that subtree T_1. T_0 is the optimizer of $C_\alpha(T)$ over subtrees of T_0 for every $\alpha \geq (|T_0| - |T_1|) / (E(T_1) - E(T_0))$, but at $\alpha = (|T_0| - |T_1|) / (E(T_1) - E(T_0))$, the optimizing subtree switches to T_1.

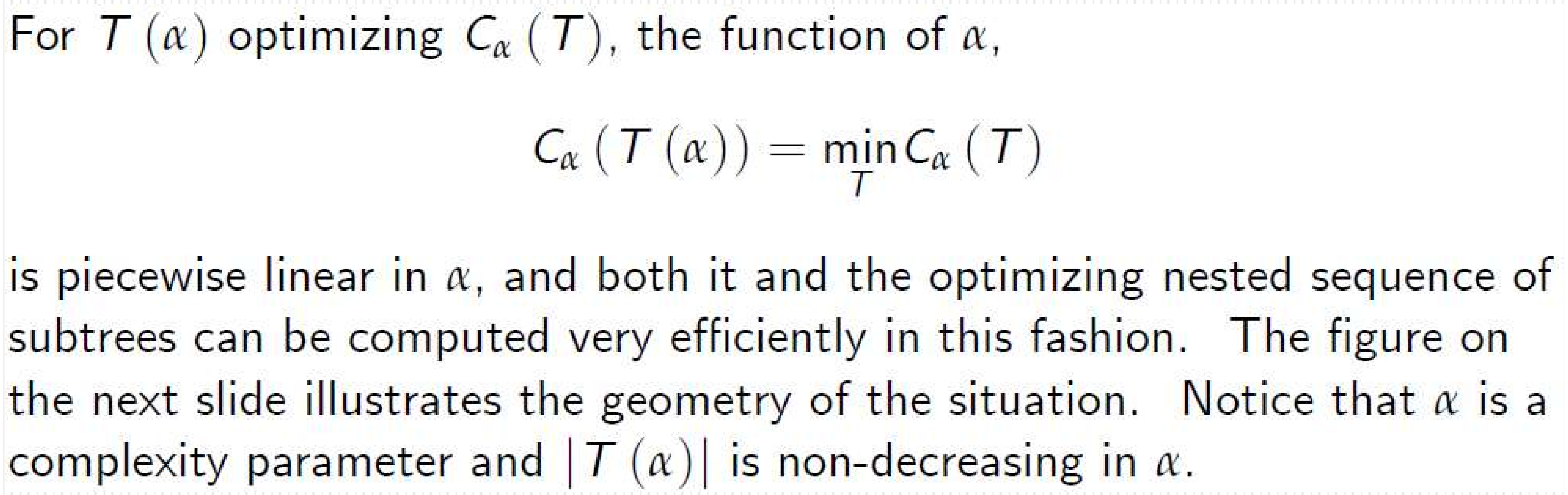
Second (and later) candidate sub-trees

One then may search over all "pruned" versions of T_1 for the one with minimum

$$\frac{E(T) - E(T_1)}{|T_1| - |T|}$$

and call it T_2. T_1 is the optimizer of $C_\alpha(T)$ over subtrees of T_0 for every $(|T_1| - |T_2|) / (E(T_2) - E(T_1)) \leq \alpha \leq$ $(|T_0| - |T_1|) / (E(T_1) - E(T_0))$, but at $\alpha = (|T_1| - |T_2|) / (E(T_2) - E(T_1))$ the optimizing subtree switches to T_2, and so on. (If there ever happens to be a tie among subtrees in terms of a minimizing a ratio of increase in total training error per decrease in number of nodes, one chooses the subtree with the smaller $|T|$.)

Optimal subtree and cost as a function of α

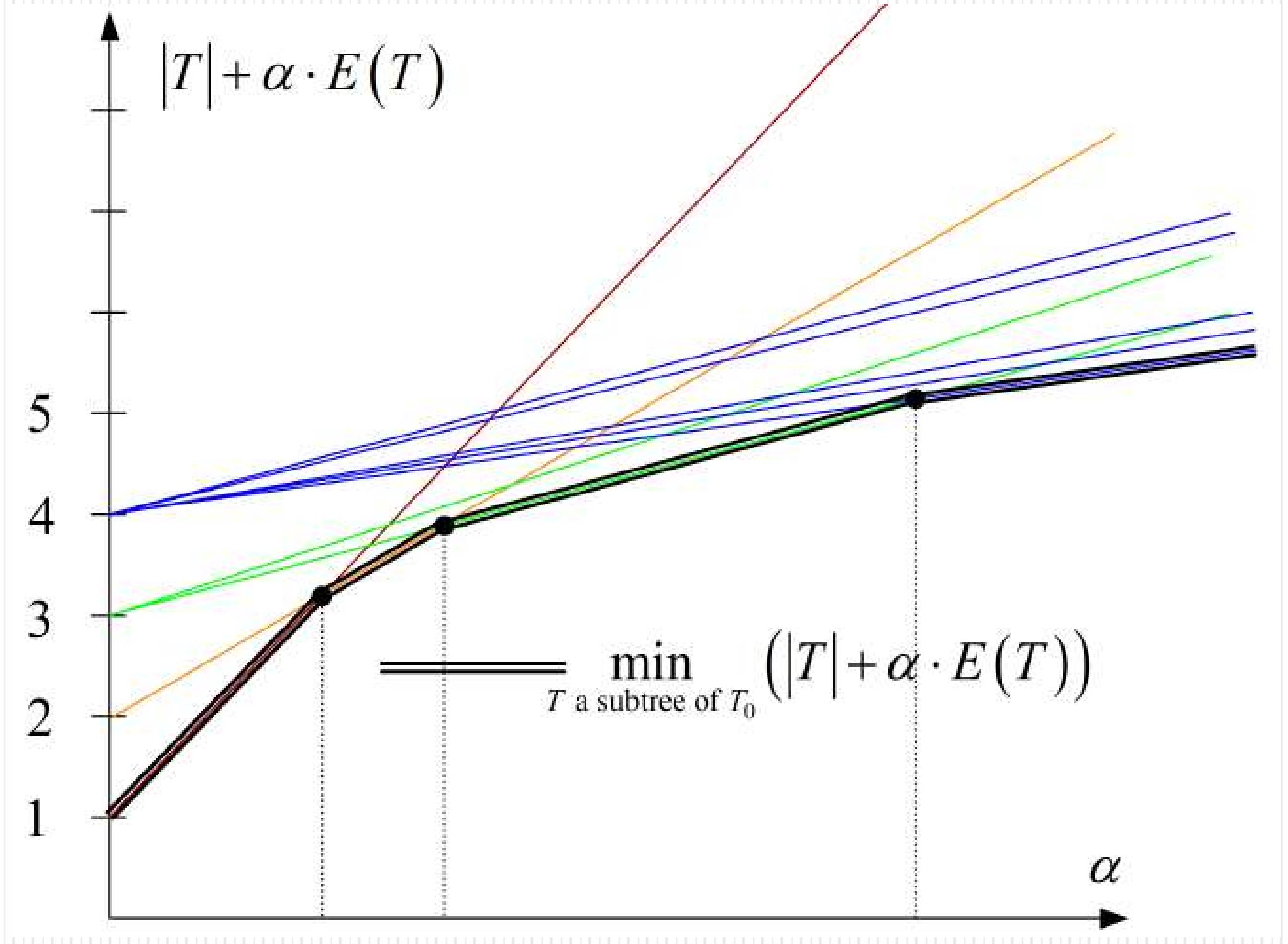


For $T(\alpha)$ optimizing $C_\alpha(T)$, the function of α,

$$C_\alpha(T(\alpha)) = \min_T C_\alpha(T)$$

is piecewise linear in α, and both it and the optimizing nested sequence of subtrees can be computed very efficiently in this fashion. The figure on the next slide illustrates the geometry of the situation. Notice that α is a complexity parameter and $|T(\alpha)|$ is non-decreasing in α.

Generic geometry of optimization



Cross-validation choice of tree predictor

One can then employ K-fold cross-validation to choose α as follows. For each of the K remainders $\mathbf{T} - \mathbf{T}_k$

1. grow an appropriate large tree (on a given dataset), then
2. "prune" the tree in 1. back by for each $\alpha > 0$ (a complexity parameter, weighting the remainder-in-training-sample error total E_k (for $\mathbf{T} - \mathbf{T}_k$) against complexity defined in terms of tree size) minimizing over choices of subtrees, the quantity

   $$C_\alpha^k(T) = |T| + \alpha \cdot E_k(T)$$

   (for $E_k(T)$ the error total for the corresponding tree predictor). Write

   $$T_k(\alpha) = \underset{\text{subtrees } T}{\arg\min}\, C_\alpha^k(T)$$

   and let $\hat{f}_\alpha^k$ be the corresponding predictor.

Cross-validation choice of tree cont.

Then letting $k(i)$ be the index of the fold $\mathbf{T}_k$ containing training case i, one computes the cross-validation error

$$CV(\alpha) = \frac{1}{N}\sum_{i=1}^{N} L\left(\hat{f}_{\alpha}^{k(i)}(\mathbf{x}_i), y_i\right)$$

For $\widehat{\alpha}$ a minimizer of $CV(\alpha)$, one then operates on the entire training set, growing a large tree T and then finding the subtree, say $T(\widehat{\alpha})$, optimizing $C_{\widehat{\alpha}}(T) = |T| + \hat{\alpha} \cdot E(T)$, and using the corresponding predictor $\hat{f}_{\hat{\alpha}}$ (in a pick-the-winner fashion).