Tree based methods

These involve **stratifying or segmenting the predictor space** into a number of simple regions. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as **decision-tree methods.**

Bagging, Random forests, and Boosting: These methods grow multiple trees which are then combined to yield a single consensus prediction. Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

Decision Tree:

Algorithm:

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Divided into splits.

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into boxes. For this reason, we take a top-down,greedy approach that is known as recursive binary splitting.
The approach is top-down because it begins at the top of the tree and then successively splits the predictor space;each split is indicated via two new branches further down on the tree.
It is greedy because at each step of the tree-building process, the best split is made at that particular step,rather than looking ahead and picking a split that will lead to a better tree in some future step.

Pruning a tree

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. Grow a very large tree To, and then prune it back in order to obtain a subtree.

Cost complexity pruning— also known as weakest link pruning— is used to do this we consider a sequence of trees indexed by a nonnegative tuning parameterα. For each value of α there corresponds to a subtreeT⊂To; Such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here |T| indicates the number of terminal nodes of the treeT, Rm is the rectangle (i.e. the subset of predictor space) corresponding to them terminal node, and ˆyRm is the mean of the training observations in Rm.

Classification Trees

Used to predict a qualitative response rather than a quantitative
ach observation belongs to the most commonly occurring class of training observations in the region to which it belongs

To find error or dissimilarity RSS cannot be used in the classification case.

We use the Gini index as a measure of node purity— a small value indicates that a node contains predominantly observations from a single class.

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

A measure of total variance across the Kclasses. The Gini index takes on a small value if all of the ˆpmk's are close to zero or one.

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

An alternative to the Gini index is cross-entropy, given by
It turns out that the Gini index and the cross-entropy are very similar numerically.

Advantages and disadvantages of trees

- Easy to explain
- Mirror human decision making
- Handle qualitative and quantitative measures
- Easy to visualize
- Limitation: Does not always have the same level of accuracy (overcome by aggregating trees)

Bagging

Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

Given a set of 'n' independent observations Z1,...,Zn, each with variance σ2, the variance of the mean $\bar{Z}$ of the observations is given by σ2/n.
In other words, averaging a set of observations reduces variance. This is not practical because we generally do not have access to multiple training sets.

In bootstrap we take repeated samples from the (single) training data set.

- generate 'B' different bootstrapped training data sets.
- then train our method on the 'bth' bootstrapped training set in order to get ˆf∗b(x), the prediction at a point x.
- then average all the predictions to obtain

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

Out-of-Bag Error Estimation

A way to estimate the test error of a bagged model.

Bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. On average, each bagged tree makes use of around two-thirds of the observations.
The remaining one-third of the observations not used to fita given bagged tree are referred to as the out-of-bag (OOB) observations. The response for the ith observation using each of the trees in which that observation was OOB. This Will yield aroundB/3 predictions for the ith observation, which we average. This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This Reduces the variance when we average the trees.

For decision trees, each time a split in a tree is considered,a random selection of m predictors are chosen as split candidates from the full set of predictors. The split is allowed to use only one of those m predictors.

typically $m \approx \sqrt{p}$: the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Boosting

Bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. In boosting this is done sequentially; each tree is grown using information from previously grown trees.

Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting,the boosting approach instead learns slowly.
By fitting small trees to the residuals, we slowly improveˆfin areas where it does not perform well. The shrinkageparameterλslows the process down even further, allowing more and different shaped trees to attack the residuals.
Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.

Algorithm:

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:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

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

   2.3 Update the residuals,

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

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Parameters for boosting:

B: number of trees, use cross-validation to select.

Λ: a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001

d: the interaction depth/number of splits, and controls the interaction order of the boosted model, since d splits can involve at most d variables. (stump=d=1)