Boxi Huang's Reading Note bh563@cornell.edu

Page Covered: PP 303 - 323

The Basic of Decision Tree

Decision trees can be applied to both regression and classification problems.

Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree. The points along the tree where the predictor space is split are referred to as **internal nodes**. We refer to the segments of the trees that connect the nodes as **branches**. The leave nodes are called **terminal nodes**.

Prediction via Stratification of the Feature Space

We now discuss the process of building a regression tree. Roughly speaking, there are two steps.

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

The goal is to find boxes R1, ..Rj that minimize the RSS, given by

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

Where y^hat (Rj) is the mean response for the training observations within the jth box. Due to heavy computation, 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 (greatest reduction in RSS) is made at that

particular step.

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\},$$

and we seek the value of j and s that minimize the equation

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

Tree Pruning:

To prevent overfit and increase test set performance, we can perform **Cost complexity pruning (Weakest link pruning).** We consider a sequence of trees indexed by a nonnegative tuning parameter a.

Algorithm 8.1 *Building a Regression Tree*

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. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \tag{8.4}$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response associated with R_m—that is, the mean of the training observations in R_m.

Classification Tree (Criterion: Classification Error Rate**):**
For a classification tree, we predict that each observation belongs to the most **commonly occurring class** of training observations in the region to which it belongs.

The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{8.6}$$

a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node *purity*—a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is *entropy*, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \tag{8.7}$$

Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$. One can show that the entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the entropy will take on a small value if the mth node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

If there is a highly non-linear and complex relationship between the features and the response, then the decision tree is better than linear models.

Bagging
Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method. Take repeated samples from the single training set from the population, build a separate prediction model using each training set, and average the resulting predictions.

In other words, we could calculate f^1(x), f^2(x),...f^n(x) using B separate training sets and average them in order to obtain a single low-variance model.

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

In **bootstrap regression tree** setting, each tree is not pruned and so there are all deep with high variance, low bias. By averaging all trees' predictions, we can obtain a low variance. In the **bootstrap classification tree** setting, we just take a majority vote for the final prediction.

Out-of-Bag Error Estimation
On average, each bagged tree makes use of around ⅔ of the observations. The remaining ⅓ is not used to fit a given bagged tree are referred to as **the out-of-bag (OOB) observations**. We can use OOB to predict the response for ith response from each tree observation. We will then get a total of B/3 response. We can get a single OOB prediction with averaging or majority vote and the OOB prediction is a valid estimate of the test error for the bagged model.

Random Forests
Random Forest provides an improvement over bagged trees by way of a small tweak that decorrelates the trees. When building Random Forest, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose m ~= square root of (total predictors)

WHY random forest is better than the bagged trees?
Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. In bagged trees, most or all trees will use the strong predictor in the top trees, resulting in identical trees with high variance for all. Random Forest decorrelates variables by letting each split to random takes a subset of features and only that specific split can predict that subset of features. On average, (p-m)/p of the split will not have a strong predictor.

Boosting
Tree are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap, instead, each tree is fit on a modified version of the original data set.

Boosting approach learns slowly. Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals. We then add

this new decision tree into the fitted function in order to update the residuals. And in this way, we slowly improve f in areas where it does not perform well.

Boosting has three tuning parameters:

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

3. The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the *interaction depth*, and controls

Feb. 3rd, 2021
bh563@cornell.edu

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:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$