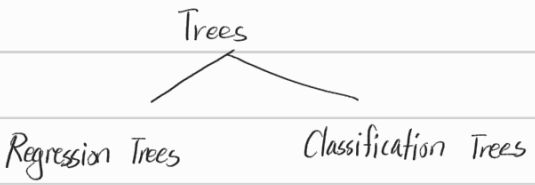
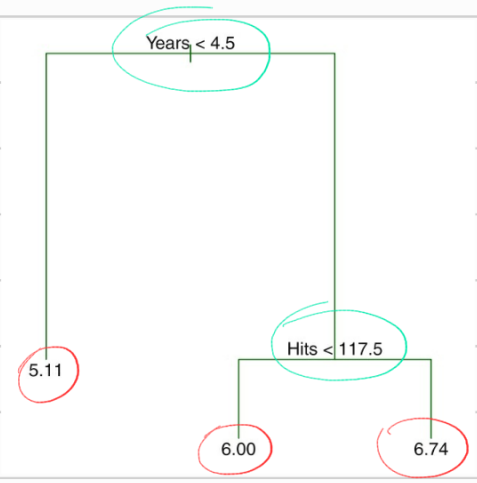
- Tree-based methods involve stratifying or segmenting the predictor space into a number of simple regions
- Tree-based methods are simple and useful for interpretation.
  ↳ but their prediction accuracy is less reliable compared to other methods

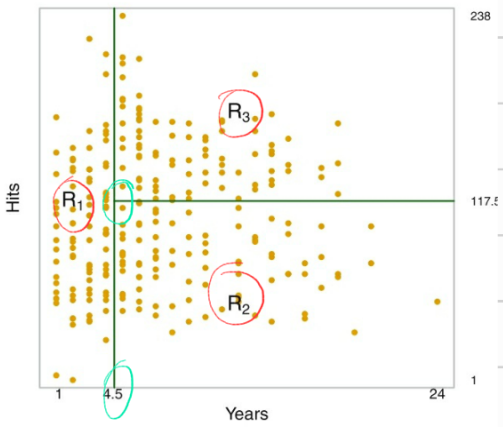
8.1 The Basics of Decision Trees



8.1.1 Regression Trees

- Each region is called terminal nodes or leaves of the tree
- The points along the tree where the predictor space is split are referred to as internal nodes.
- The segments of the trees that connect the nodes as branches.





O - terminal nodes
O - internal nodes

Prediction via Stratification of the Feature Space

- Process of building a regression tree :

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.

- We choose to divide the regions (the predictor space) into high-dimensional rectangles, or boxes. The goal is to find boxes $R_1, \dots, R_J$ that minimize the RSS,

$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, where $\hat{y}_{R_j}$ is the mean response for the training observations within the j^{th} box.

- However, since considering every combination of points is infeasible, recursive binary splitting such as greedy algorithm is used.

> jth box. Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J 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 (at which point all observations belong to a single region) 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.

- The general process explained at pdf 307

Tree Pruning

- Growing large / complicated trees will give the best tree for the training data.
- instead, it would be better to grow a very large tree T_0, and then prune it back in order to obtain a subtree.
- However, it is still computationally infeasible to consider all subtrees.
- As for Cost Complexity Pruning, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

> For each value of α there corresponds a subtree $T \subset T_0$ such that
>
> $$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \qquad (8.4)$$
>
> 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.

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.

EX)

tree on the Hitters data, using nine of the features. First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set. We then built a large regression tree on the training data and varied α in (8.4) in order to create subtrees with different numbers of terminal nodes. Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α. (We chose to perform six-fold cross-validation because

8.1.2 Classification Trees

- As for regression trees, the predicted value of a region is the mean of all points within the specific region. As for classification trees, the predicted response is the most commonly occurring class of training observations in the region to which it belongs.
- We are interested to check the class prediction and class proportions among the training observations that fall into that region.
- Instead of RSS, we use classification error rate, the fraction of the training observations in that region that do not belong to the most common class.

  $E = 1 - \max_k (\hat{P}_{mk})$, where $\hat{P}_{mk}$ represents the proportion of training observations in the m^{th} region that are from the k^{th} class.
- Because the classification error is not sufficiently sensitive for tree-growing, we use Gini Index, a measure of total variance across the K classes.

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

생각해보니 기말범위가 아니네?

8.2 Bagging, Random Forests, Boosting

8.2.1 Bagging

- is used for reducing the variance of a statistical learning method.

samples from the (single) training data set. In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, and finally average all the predictions, to obtain

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

This is called bagging.

While bagging can improve predictions for many regression methods, it is particularly useful for decision trees. To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance. Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

- For a given test observation, we can record the class predicted by each of the B trees, and take a majority vote.

* The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting

Out-of-Bag Error Estimation

- Observations that were not used for building a model for each bootstrap sample are called out-of-bag observation, OOB
- Error Estimation for regression:
  → Using each OOB set as a test set, compute the average responses of OOB
- Error Estimation for classification:
  → majority rule with OOB's

– to a single OOB prediction for the ith observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each

Variable Importance Measures

8.2.2 Random Forests

- each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
  - $m \approx \sqrt{p}$

8.2.3 Boosting

- each tree is grown using information from previously grown trees; each tree is fit on a modified version of the original data set.

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

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 the interaction order of the boosted model, since d splits can involve at most d variables.