1 Tree-Based Models

1.1. Tree-based models involve *stratifying* or *segmenting* the predictor space into regions. The general template is as follows.

1. Divide the predictor space—the set of possible values for features $X_1, \dots, X_p$—into J distinct and non-overlapping regions $R_1, \dots, R_J$.
2. For every observation that falls into the region R_i, we predict its response based on the response values for the training observations in R_j:
   - For regression problems, we use the mean of response values for the training observations in R_j.
   - For classification problems, we use the most commonly occurring label for the training observations in R_j.

1.1 Constructing the Regions.

1.2. So how exactly do we construct those regions in step 1? Recall our goal is to find boxes $R_1, \dots, R_J$ that minimizes 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 box j. However, it is computationally infeasible to consider every possible partition of the feature space. For this reason, we use a top-down, greedy approach known as **recursive binary splitting**.

1.3. Intuitively, for each iteration, we select a predictor X_j and a cutpoint s to split one of the previously identified region (initially there is only one region: the entire predictor space) into two sub-regions that leads to the greatest possible reduction in RSS.

1.4. More precisely, for any $j \in [J]$ and s, we define the pair of half-planes

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

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

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_2 \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for the training observations in $R_1(j,s)$ and $R_2(j,s)$, respectively. We repeat this process, looking for the best predictor and best cutpoint in order to split the data further. This process continues until a stop criterion is matched.

1.2 Tree Pruning.

1.5. The above procedure is likely to overfit the data, because the resulting tree might be too complex. To fix this, we grow a very large tree T_0 and then prune it to obtain a subtree. Our goal is to select a subtree that leads to the lowest test error rate.

1.6. The naive solution is to estimate the test error using cross-validation for each subtree, but there are just too many of them. A better solution is to use **cost complexity pruning** (also known as **weakest link pruning**): rather than considering every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter α.

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

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

is minimized. Here, $|T|$ indicates the number of leaves/regions of the tree T, R_m is the rectangular region corresponding to the mth terminating node, and $\hat{y}_{R_m}$ is the predicted response (mean or mode) associated with R_m.

1.8. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, the subtree T will simply equal T_0. As α increases, there is a price to pay for having many leaves, so we obtain a smaller subtree.[*] This value α can be selected using cross-validation. We then return to the full data set and obtain the subtree corresponding to α.

1.9. The following is a complete algorithm that builds a regression tree:

Algorithm 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 to obtain a sequence of best
   subtrees, as a function of $\alpha$.
3: Use cross-validation to choose $\alpha$ which minimizes the average error.
4: Return the subtree from step 2 that corresopnds to the chosen value of $\alpha$.
```

[*]Compare this with lasso regression.

1.3 More Remarks.

1.10. So far, we have been using RSS for regression trees. Naively, we could use classification error rate for classification trees:

$$E = 1 - \max_k(\hat{p}_{mk}),$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, this is often not good enough. In practice, we prefer the following two measures:

- **Gini index**, a measure of total variance across the K classes:

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

- **Entropy**, a common measure from information theory:

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

1.11. We now compare decision trees with linear models. In general, decision trees are easier to explain, mirror human decision-making process, can be graphically displayed, and have no need for dummy variables. However, they have less predictive power and are non-robust. We now cover techniques to address these issues.

1.12. Vanilla decision trees discussed above suffer from high variance. This means the model we trained highly depend on the training observations. Recall given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. In other words, averaging a set of observations reduces variance. Based on this result, we introduce three techniques: bagging, random forests, and boosting.

1.4 Bagging and Random Forests.

1.13. Bagging involves repeatedly taking samples from the training data set. This generates B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$ and average all predictions to obtain

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

1.14. At a high level, we construct B regression trees using B bootstrapped training sets, then average the resulting predictions. These trees are grown deep, unpruned. Each individual tree has high variance but low bias. Averaging them reduces the variance. For classification trees, we simply take the majority vote when predicting.

1.15. Bagging improves prediction accuracy at the expense of interpretability as the collection of bagged trees is much more difficult to interpret than a single tree. Still, one can obtain an overall summary of the importance of each predictor using the RSS (for regression trees) or the Gini index (for bagging classification trees): record the total amount that RSS/GI is decreased due to splits over a given predictor, average over all B trees; a large value indicates an important predictor.

1.16. Suppose there is one very strong predictor in the data set along with some moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split (due to our greedy approach). Consequently, all of the bagged tree will look quite similar to each other. This causes problem as averaging highly correlated quantities does not lead to a large reduction in variance.

1.17. To fix this problem, we aim to decorrelate the trees. **Random forest** overcome this problem by forcing each split to consider only a subset of the predictors. More precisely, 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. The split is allowed to use only one of those m predictors. Therefore, on average $(p - m)/p$ of the splits will not consider the strong predictor and other predictors will have more of a chance. Note that when $m = p$, this is reduced to bagging.

1.18. In summary, this process decorrelates the trees, thereby making the average of the resulting trees less variable and hence more reliable.