Trees and Random Forest

ST 697 | Fall 2017
University of Alabama

Classification and Regression Trees

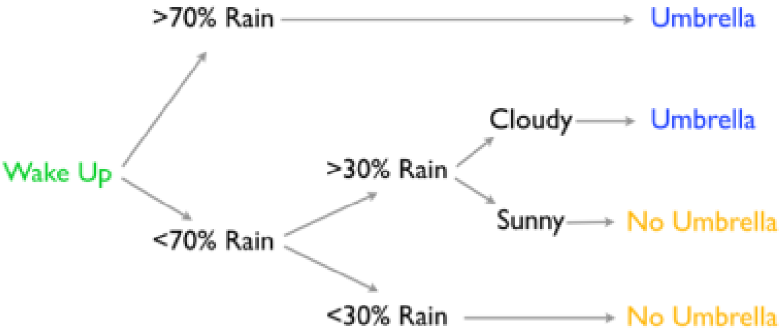
Classification and Regression Trees

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful.

- Main Characteristics: very flexible, very intuitive, non-model based, hierarchical nature, natural graphical display, easy to interpret
- Main Implementations: CART (Classification and Regression Trees) by Breiman, Friedman, Olshen, Stone (1984) and C4.5 Quinlan (1993).

Decision Trees

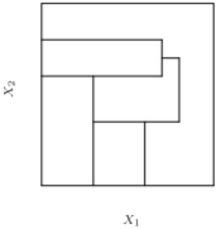
http://www.20q.net/



Building Trees

As usual, we want find the tree that minimizes the loss.

- Classification trees have class probabilities at the leaves (e.g., the probability I'll be in heavy rain is 0.9). Use binomial likelihood.
- Regression trees have a mean response at the leaves. (e.g., the expected amount of rain is 2in). Use squared error.

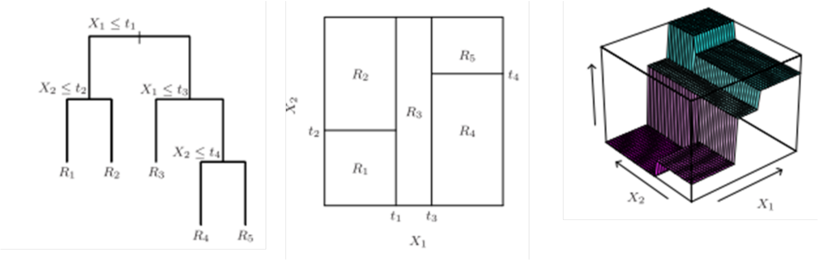


Note: This looks something like adaptive binning or nearest neighbor

Recursive Binary Partition

Because the number of possible trees is too large, we usually restrict attention to recursive binary partition trees (CART).

- These are also easy to interpret



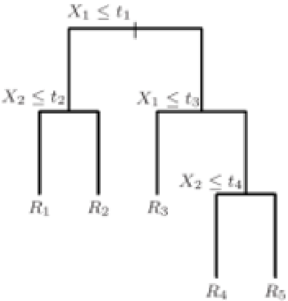
Model the response as a *constant* in each region

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m)$$

Basis Expansion Interpretation

$$f(x) = \sum_{m=1}^{M} \theta_m b_m(x)$$

$$b_1(x_1, x_2) = \mathbb{1}(x_1 \leq t_1)\mathbb{1}(x_2 \leq t_2)$$
$$b_2(x_1, x_2) = \mathbb{1}(x_1 \leq t_1)\mathbb{1}(x_2 > t_2)$$
$$b_3(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 \leq t_3)$$
$$b_4(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 > t_3)\mathbb{1}(x_2 \leq t_4)$$
$$b_5(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 > t_3)\mathbb{1}(x_2 > t_4)$$



Growing a Tree

CART uses a greedy algorithm to grow a tree.

- Split the feature space into two pieces and model response in each region
  - Find the predictor j (out of $1, 2, \ldots, p$) and split point t (from unique ordered values of X_j or categories) to minimize the loss function
  - Produces two regions: $R_1(j,t) = \{x : x_j \leq t\}$ and $R_2(j,t) = \{x : x_j > t\}$ or $R_1(j,t) = \{x : x_j \in A_j\}$ and $R_2(j,t) = \{x : x_j \notin A_j\}$
- Repeat this step for each child region
- Continue until stopping criteria met, e.g.
  - Minimum number of observations in region
    - Loss function has minimal improvement
- The final regions are called leaf nodes

Stopping and Pruning

- Tree Size
  - A large tree (many leaf nodes with few observations) can overfit
  - A small tree can not capture important structure
  - Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data
- Early Stopping
  - Stop splitting when loss function has insignificant improvement (like forward stepwise), but a seemingly worthless initial split can lead to a good split further down the tree (short-sighted)
- Pruning
  - Grow a full tree (minimum node size) and prune back (like backwards stepwise)

Cost Complexity Pruning

Let N_m be the number of observations in node R_m and $Q_m(T)$ be the loss in region m for a given tree T.

- e.g., $Q_m(T) = \frac{1}{N_m} \sum_{\{i:x_i \in R_m\}} (y_i - \hat{c}_m)^2$

Weakest link pruning: Successively collapse the internal node that produces the smallest per-node increase in $\sum_{m=1}^{|T|} N_m Q_m(T)$ until you reach the single node (root) tree. This produces a finite sequence of sub-trees.

For each sub-tree T, define its cost complexity

$$C_\lambda(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda |T|$$

where the m cover all terminal nodes in T and λ is a penalty on tree size $|T|$.

Penalty Tuning

- For each λ, there is a unique smallest sub-tree T_λ that minimizes $C_\lambda(T)$.
- The sequence of sub-trees from weakest link pruning contains every T_λ
- The tuning parameter, λ can be chosen from cross-validation

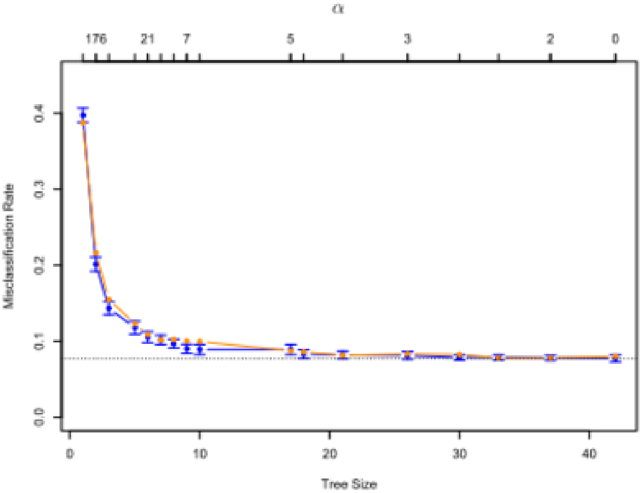


FIGURE 9.4. *Results for* spam *example. The blue curve is the 10-fold cross-validation estimate of misclassification rate as a function of tree size, with standard error bars. The minimum occurs at a tree size with about 17 terminal nodes (using the "one-standard-error" rule). The orange curve is the test error, which tracks the CV error quite closely. The cross-validation is indexed by values of α, shown above. The tree sizes shown below refer to $|T_\alpha|$, the size of the original tree indexed by α.*

Categorical Predictors

- If categories are ordered (e.g., likert scale) then split like continuous response
- If unordered, then first order the categories by mean response and treat like an ordered factor
- For a large number of categories this can overfit too fast - recommended to combine levels or drop such predictors

Missing Predictor Values

- For categorical predictors, create an additional level “missing"
  - This can reveal important patterns from the missing values
- Surrogate splits
  - For every split, create a list of surrogate splits that mimics the original splits
  - For prediction, if an observation has a missing value use the surrogate splits to determine which child group it should go into
- Alternatively, we could omit observations with missing values or impute

Binary Splitting

- Multiway splits are possible, but could partition the data too quickly (overfit)
- Multiway splits could be achieved from a combination of binary splits

Tree Advantages

- Handles categorical and continuous data in consistent manner
- Automatic variable selection (any predictor not split)
- Automatically discover interactions between multiple predictors
- Because the partitions are made from the data, trees give a *locally adaptive* estimate
- Invariant to monotone transformations
- Can be robust to outliers (splitting on sample quantiles)
- Easy to interpret

Tree Limitations

- Instability (high variance) due to greedy hierarchical structure; one change at top split can change remaining tree.
- Would be difficult to use trees for making inferences due to stepwise search (see above)
- Difficulty capturing additive structure

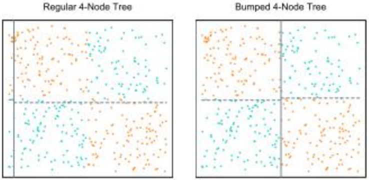


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm 20 times.*

Trees in R

R packages: `tree` and `rpart` and `party`

Bagging Trees

Better Trees

- Because of the instability of trees, they are great candidates for methods like bagging that will reduce the variance.
- Grow a set of B trees from a bootstrap sample and average their predictions:

  $$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x; \Theta_b)$$

  where Θ_b are all the parameters for fitting the tree (split variables, cutpoints, and terminal (leaf) node values to the bootstrap sample b.
- Lots of detail and discussion can be found in Breiman (1996 - Machine Learning).

Bagging Trees

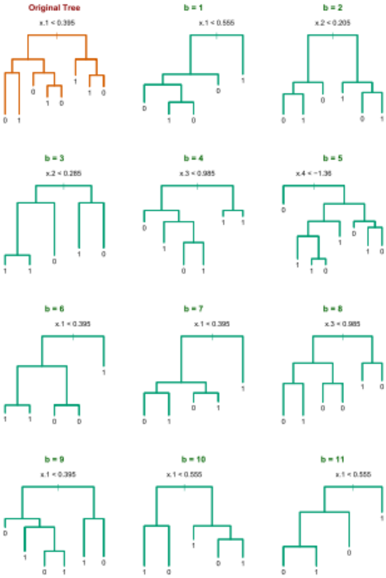
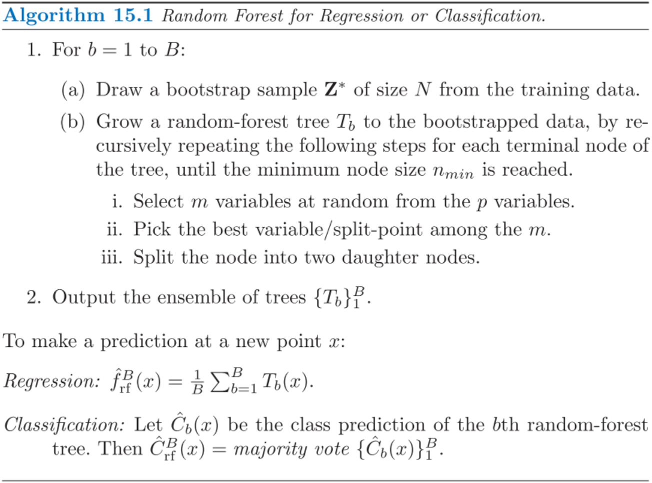


FIGURE 8.9. *Bagging trees on simulated dataset.*

Random Forest

Random Forest

Random Forest is a modification of bagging that attempts to build de-correlated trees and then average them



Algorithm 15.1 *Random Forest for Regression or Classification.*

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random Forest Tuning

There are two primary tuning parameters for Random Forest:

1. m controls the number of predictors that are evaluated for each split
2. $min.obs$ or $depth$ controls how large the tree grows

How do these relate to the bias/variance trade-off?

Note:

- For classification, the default value is $m = \lfloor \sqrt{p} \rfloor$ and $min.obs = 1$.
- For regression, the default value is $m = \lfloor p/3 \rfloor$ and $min.obs = 5$.

OOB error

For each observation (x_i, y_i), construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which observation i did not appear.

$$\hat{f}(x_i) = \frac{1}{N_B(i)} \sum_{b=1}^{B} \mathbb{1}(x_i \in \mathrm{OOB(b)}) \cdot \mathrm{T(x_i; \Theta_b)}$$

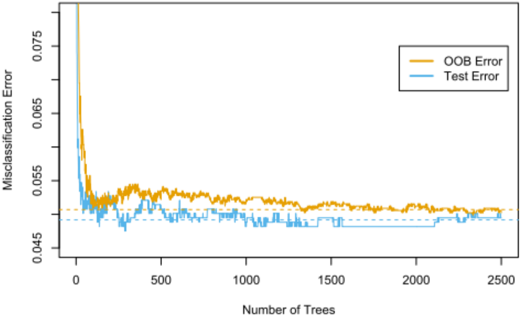


Figure 15.4 in ESL

Variable Importance

At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

The importance of predictor j in a single tree T:

$$\mathcal{I}_j^2(T) = \sum_t i_j^2 \cdot \mathbb{1}(v(t) = j)$$

The importance of predictor j in a forest:

$$\mathcal{I}_j^2 = \frac{1}{B} \sum_{b=1}^{B} \mathcal{I}_j^2(T_b)$$

Random Forest and k-NN

Random Forests (especially with fully grown trees) are similar to k-NN methods, but adaptively determines the neighbors.

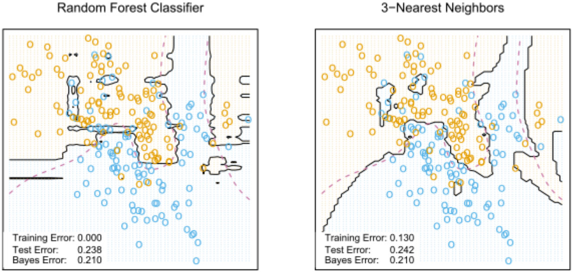


FIGURE 15.11. *Random forests versus 3-NN on the mixture data. The axis-oriented nature of the individual trees in a random forest lead to decision regions with an axis-oriented flavor.*