Tree Methods

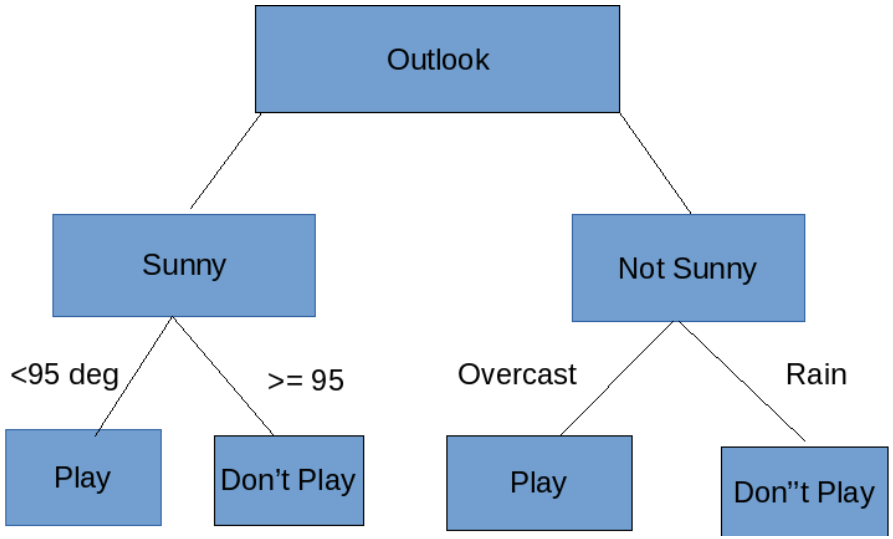
Eric Roemmele Josh Lambert

January 25, 2018

Motivation

Golf

- Should I play golf today?
- I can ask myself a series of questions to determine this - make a sort of flowchart.



Motivation Cont.

- That is the general idea of trees - recursive binary splits of factors to where we say if we fall in this "region", classify the observation as "sick" or the predicted value is 5 (for example).
- More specifically, we partition the sample space (or feature space) into disjoint regions and in those regions, are fitted values are the mean (of all the observations that fall into that region) for a continuous outcome, or the majority vote (of all the observations that fall into that region) for a categorical outcome
- Really seems to mimic human decision making, and are highly interpretive models
- Nonparametric - no distributional assumptions

Example

Wanted to relate the number of points a basketball player scored by his height and weight.

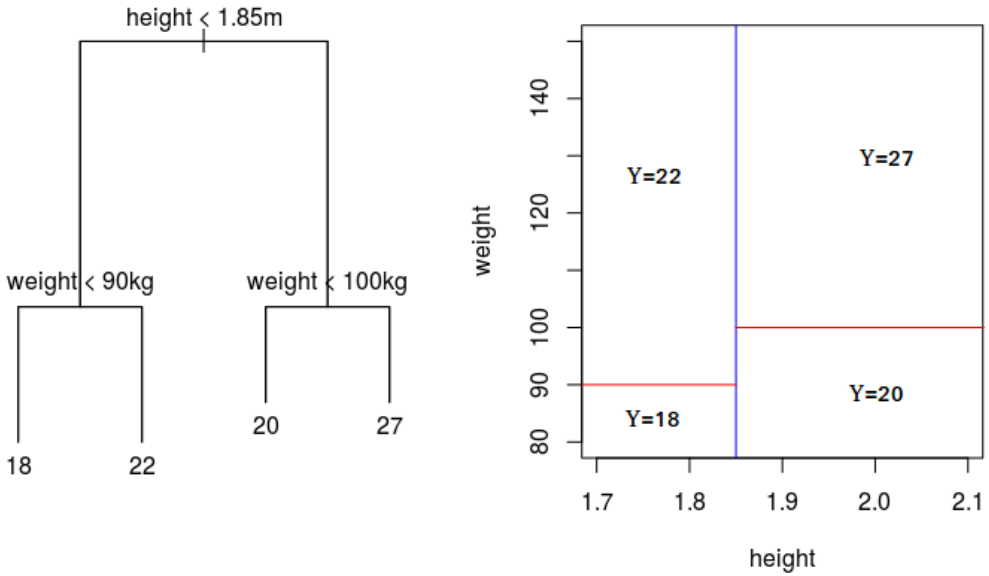


Figure: https://www.r-bloggers.com/how-random-forests-improve-simple-regression-trees/

Algorithm

- Suppose we observe (y_i, x_i) for $i = 1, \ldots, N$ and $x_i = (x_{i1}, \ldots, x_{ip})$
- Algorithm needs to decide on the splitting variable and split points, as well as the topology (shape) of the tree
- We want to partition the feature space into M distinct regions $R_1, \ldots, R_M$, and we model the response as a constant c_m in each region
  - So, $f(x) = \sum_{i=1}^{M} c_m I\{x \in R_m\}$
- If we assume squared error loss, i.e $L(y, f(\cdot)) = \sum_{i=1}^{N} (y_i - f(x_i))^2$, then it follows that $c_m = \text{ave}(y_i | x_i \in R_m)$.
- Finding the best binary partition in terms of minimizing squared error is computationally unfeasible. Thus, use what is called a *greedy* algorithm.

Algorithm Cont.

Algorithm

1. Starting with all the data, consider a splitting variable j and split point s, and define the half-planes to $R_1(j,s) = \{X|X_j \leq s\}$ and $R_2(j,s) = \{X|X_j > s\}$
2. Find the splitting variable j and split point s that solve

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right] \quad (1)$$

3. For any choice of (j,s), the inner minimization is solved by $\widehat{c_1} = \text{ave}(y_i|x_i \in R_1(j,s)$ and $\widehat{c_2} = \text{ave}(y_i|x_i \in R_2(j,s)$.
4. For each (potential) splitting variable, do a grid search of the best splitting point s. The pair (j,s) that minimizes (1) above is the "best split".
5. Partition the data into the two regions, and then repeat the splitting process on each of the two regions. But how long do we do this for?

Cost Complexity Pruning

- How large should we grow a tree. A large tree would overfit, while a too small tree might miss important structures.
- We could stop splitting when the decrease in SSE is too small at an iteration. This is called pre-pruning.
  - Issue - too short-sighted, a poor split at one iteration might lead to a good split below it.
- Our strategy : Grow a large tree T_0, stopping the process only when some minimum node size is reached. Then, imposing a penalty, prune the branches.
- Define a subtree $T \subset T_0$ to be a tree that can be obtained by pruning T_0, that is, collapsing its internal nodes.
- Index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T

Cost Complexity Pruning Continued

Cost Complexity Pruning

Let

$$N_m = \#\{x_i \in R_m\}$$

$$\widehat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \widehat{c}_m)^2$$

we define the cost complexity criterion

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

CC Pruning Cont.

- We write a tree as $T(x;\theta) = \sum_{j=1}^{M} c_m I\{x \in R_j\}$, where θ consists of the nodes and split points.
- Find, for α, the subtree $T_\alpha \subset T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter α governs the tradeoff between tree size and its goodness of fit.
- Large $\alpha \rightarrow$ smaller tree; small $\alpha \rightarrow$ larger tree
- How to choose α? Use validation set (if we have enough data), or use k-fold (usually 5 or 10) cross validation.

Cross-Validation

CV Process

1. Select a grid of values for α (i.e. $\alpha \in$ *lbrace*$.1, .2, .3, .4, \dots\}$
2. Partition the training data into K separate sets of equal size.(Usually K=5 or 10)
3. For each $k = 1, 2, \dots, K$, train $\widehat{f}_{-k}^{(\alpha)}(x)$ on the set excluding the k-th fold T_k.
4. Compute the fitted values for the observations in T_k, based on the training data that excluded this fold.
5. Compute the cross-validation error associated with the k-th fold.

$$CV_k^{(\alpha)} = n_k^{-1} \sum_{x_i, y_i \in T_k} (y_i - \widehat{f}_{-k}^{(\alpha)}(x))^2$$

where n_k is the number of observations in T_k

CV Continued

CV Continued

6. The overall CV error is

$$CV^{(\alpha)} = K^{-1} \sum_{k=1}^{K} CV_k^{(\alpha)} \tag{2}$$

7. Of the alpha in the grid points, choose the one that minimizes the above.

If we have enough data, split the data into training (T), validation (V), and test. For each α, train the model on the training data, then evaluate $CV^{(\alpha)} = \sum_{(x_i, y_i \in V} (y_i - \widehat{f}^{(\alpha)}(x_i))^2$, where $\widehat{f}(\cdot)$ is the tree trained on T. Choose the α that has the smallest $CV^{(\alpha)}$. We'd choose α before we fit the overall tree and prune.

Classification

- Suppose y is categorical and takes values as $1, \ldots, K$. Same algorithm as before, but we need a different impurity measure $Q_m(T)$ since squared error is not suitable for classification.
- In node m, define
$$\widehat{p}_{mk} = N_m^{-1} \sum_{x_i \in R_m} I\{y_i = k\}$$
- Classify the observation in node m to class $k(m) = \underset{k}{argmax} \widehat{p}_{mk}$; or the majority vote
- Common node impurity measures are

$$\text{Misclassification Error: } N_m^{-1} \sum_{i \in R_m} I\{y_i \neq k(m)\} = 1 - \widehat{p}_{mk}$$

$$\text{Cross-Entropy: } -\sum_{k=1}^{K} \widehat{p}_{mk} \log \widehat{p}_{mk}$$

- Misclassification Error makes the most practical sense, but is not differentiable. Cross-Entropy is more wieldly.

Example : Spam Data

- The data consists of 4601 email messages, the goal is to screen for spam email.
- The response is binary - 1 for spam and 0 for email
- 57 predictors
  - 48 quantitative predictors - percentage of words in the email that mathch a given word (such as business, address, etc.)
  - 6 quantitative predictors - percentage of characters in the email that match a given character (such as ;,$,etc.)
  - Average length of uninterrupted capital letters , length of the longest uninterrupted sequence of capital letters, and sum of the length of uninterrupted sequences of capital letters.

Bagging

Bagging improves prediction by averaging, which thereby reduces variance.

Algorithm

1. Let $\boldsymbol{Z} = \{(x_i, y_i)\}_{i=1}^{n}$. Sample a bootstrap data set $\boldsymbol{Z}^{*b}$ for $b = 1, \ldots, B$.
2. On each bootstrap data set, $\boldsymbol{Z}^{*b}$, fit a tree $\widehat{f}^{*b}(x)$.
3. The bagged estimate is then,

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

4. If the outcome is categorical, $\widehat{f}_{bag}(x) = \underset{k}{argmax}\,[p_1(x), \ldots, p_K(x)]$, where $p_k(x)$ is the proportion of bootstrapped trees predicting class k at x.

Why Bag?

1. Reduce the variance by averaging, but bias is unchanged.
2.

$$\begin{aligned}\mathbb{E}\left[Y-\widehat{f}^{*b}(x)\right]^2 &= \mathbb{E}\left[Y-\mathbb{E}\widehat{f}^{*b}(x)+\mathbb{E}\widehat{f}^{*b}(x)-\widehat{f}^{*}(x)\right]^2 \\ &= \mathbb{E}\left[Y-\mathbb{E}\widehat{f}^{*b}(x)\right]^2+\mathbb{E}\left[\widehat{f}^{*b}(x)-\mathbb{E}\widehat{f}^{*b}(x)\right]^2 \qquad (3)\\ &\geq \mathbb{E}\left[Y-\mathbb{E}\widehat{f}^{*}(x)\right]^2\end{aligned}$$

   This essentially says that bagging will never increase MSE.
3. **Warning** - The above argument does not hold for classification because we are typically not using squared loss. Its possible that bagging a poor classifier can make it worse.

Boosting Trees

- I recommend reading Chapter 10 (or there are plenty of blogs on boosting) to get a deeper comprehension. This is great as well `http://scikit-learn.org/stable/modules/ensemble.html`

AdaBoost for Classification

The general idea is to fit a sequence of weak learners to repeatedly modified versions of the data. At each iteration, observations that were misclassified in the previous step receive more weight. Each succesive classifier is therby enforced to concentrate on observations that are hard to classify.

Algorithm

1. Initialize the observation weights $w_i = 1/N$
2. For $m = 1$ to M
   - Fit a weak learner to the data using weights w_i.
   - Compute
   $$err_m = \frac{\sum w_i I\left(y_i \neq G_m(x_i)\right)}{\sum w_i}$$
   - Compute $\alpha_m = \log\left((1 - err_m)/err_m\right)$
   - Set $w_i \leftarrow w_i \cdot \exp(\alpha_m I(y_i \neq G_m(x_i)))$
3. Output $G(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$

Gradient Boosting

- AdaBoost is great for classification. Gradient Boosting allows us to extend to regression problems (but this method can still be applied to classification).
- Similar idea - Fit a sequence of weak learners, but then look at the residuals at each iteration.
- Consider models of $G(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$, where h_m is a weak learner.

Algorithm

Let L be a loss function.

Gradient Boosting

1. Initialize $f_o = \underset{\gamma}{argmin} \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M
   - For $i = 1, \ldots, N$, compute the pseudo residuals

$$r_{im} = -\left.\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right|_{f=f_{m-1}}$$

   - Fit a regression tree to r_{im} giving regions R_{jm}, $j = 1, \ldots, J_m$.
   - For $j = 1, \ldots, J_m$ compute

$$\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I\{x \in R_{jm}\}$
3. Output $\widehat{f}(x) = f_M(x)$

Random Forests

- Recall in bagging we average over noisy, but unbiased models, to reduce the variance.
  - Each tree is identically distributed.
- In contrast, bagging trees are grown in an adaptive way to remove bias, and are not id
- The variance of the average of B i.d. variables is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
  - As B increases the second term disappears, but the first remains, and the size of the correlation of pairs of bagged trees limits the benefits of averaging.
- **Random Forests** - Improve the variance reduction of bagging by reducing ρ, without increase σ^2 too much.

Random Forests Cont

This is done by random selection of the inputs.

Algorithm

1. For b =1 to B
   - Draw a bootstrap sample Z_b^* from the training data.
   - Grow a tree T_b to Z_b^* drawing $m < p$ randomly selected features.
2. Output the ensemble T_b for $b = 1, \ldots, B$.

Then, for regression $\widehat{f}_{rf}^B(x) = B^{-1} \sum_{b=1}^B T_b(x)$. For classification, take the majority vote at a fixed feature x across the trees.

Common value of m are 1 and $\sqrt{p}$.