Machine Learning for Social Sciences
Part 3: Tree-based Models

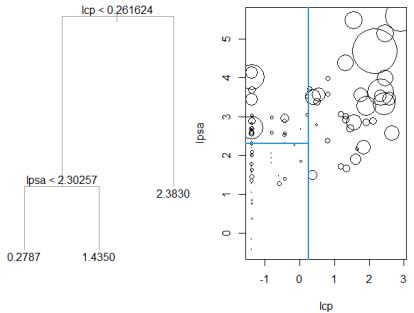
Felix Hagemeister

TUM School of Social Sciences and Technology
April 2022

Introduction

OLS seems often too restrictive as it imposes a linear relationship between input and output (check scatter plot of fitted values and actual values to gauge performance).

- **Classification and Regression Trees (CART)** is a fully *nonparametric* method that can better account for non-linear relationships.
- Trees and Random Forest are default prediction tools for data with $n > \frac{p}{4}$



Tree

Suppose we have input X and output y. A tree is characterised by

- a logical system for mapping inputs to outputs,
- hierarchical use of series of ordered steps (*nodes*) to reach a decision,
- each node contains a *split* on available input data,
- leaf nodes containing the final decision (prediction rule), i.e. the predicted $\hat{y}$, the average of sample y values that end up in that leaf.

The goal is to have a sequence of decision nodes that combine for a good final choice, that is to minimize a loss function.

Tree

Given a *parent node* containing data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, the optimal split is location x_{ij} – dimension j on observation i – that makes the *child* sets as *homogeneous* in response y as possible. The child sets are denoted as

left: $\{\mathbf{x}_k, y_k : x_{kj} \leq x_{ij}\}$ and right: $\{\mathbf{x}_k, y_k : x_{kj} > x_{ij}\}$.

For a regression with squared error loss, this means that you would want to minimize the function

$\sum_{k \in left}(y_k - \bar{y}_{left})^2 + \sum_{k \in right}(y_k - \bar{y}_{right})^2$

CART Algorithm (Breiman et al., 1984)

We start at the root containing the full (training) sample. For each node:

1. Determine the single error-minimizing split for this data (i.e. we try across all dimensions i and all observations j)
   - For each dimension X_j:
     - for each potential split point i:
       calculate prediction and loss function, choose split point minimizing loss
     - choose X_j with lowest loss
2. Split parent node in left and right children accordingly.
3. Apply steps 1 and 2 to each child node.

Continue until stopping rule applies (*greedy forward search*).

Stopping rule

Stopping rules can be (a combination of):

- Minimum observations in terminal leaf node (e.g. 10 observations in leaf)
- Minimum loss improvement
- Maximum depth

For CART, it doesn't matter whether inputs are logged or transformed in any mean preserving way. Transformation of the response variable does matter, though.

Flexibility vs. Stability

Potential Overfit: do we need all nodes or are we fitting too much noise?

- **Solution 1: Pruning and cross-validation**
  - fit an overgrown tree and prune backward iteratively by removing leaf splits with lowest in-sample error reduction
  - thus obtain a set of candidate trees
  - choose best tree using cross-validation
- **Solution 2: Minimal Cost-complexity Pruning**
  - find a subtree of T that minimizes cost-complexity $R_\alpha(T) = R(T) + \alpha N$ where
    - $R(T)$ is loss function at terminal nodes (e.g. misclassification rate)
    - N number of terminal nodes in (sub)tree
    - α complexity parameter to be tuned by cross-validation
  - increasing number of terminal nodes decreases $R(T)$ but increases complexity penalized by αN.

Exercise

Let's try to predict cancer volume using medical data and trees.

You can find the R code here.

Random Forest

Random Forest

Bootstrap Aggregating = Bagging = Model Averaging

- obtain many trees by fitting CART on multiple with-replacement samples from our data
- then average over bootstrapped sample of trees to get mean fit β = average model fit

Random Forest = bagging of CART-trees

- Intuition: by averaging over multiple trees, we preserve the stable relationship that persists across bootstrap samples, while the noise averages out. We reduce

Random Forest Algorithm

Suppose B is the bootstrap size (i.e. number of trees). For $b = 1...B$:

1. Sample with-replacement n observations from data
2. Fit a CART tree – T_b – to this sample

This results in a set of trees $T_1...T_B$. Random Forest predictions are averages of individual tree predictions:

- If $\hat{y}_{ib}$ is the prediction given x_i from tree T_b, random forest prediction is $\hat{y}_i = \frac{1}{B}\sum_b \hat{y}_{ib}$.

Technicality: Grow trees until OOS error stabilizes, i.e. adding trees does not improve OOS performance

Random Forest Algorithm

Advantages

- Very good out-of-the-box performance, even with all hyperparameters set to default.
- Among more popular ML methods, random forest has the least variability in their prediction accuracy when tuning.
- Parallel and partitioned computing makes random forests deployable on big data on industrial scale
  - *Parallel Computing:* doing many computations at once
  - *Distributed algorithms:* work with subsets of data
- Can be used to get causal trees for heterogeneous treatment effect (Athey and Imbens, 2016)

Disadvantage

- Lack of illustration (but can report variable importance defined by increase in error when omitting variable)

Random Forest Hyperparameters

Main hyperparameters include:

1. The number of trees in the forest
2. The number of features to consider at any given split
3. The complexity of each tree
4. The sampling scheme
5. The splitting rule to use during tree construction

where 2. is most important, 2.,3., and 5. have only small impact on accuracy.

One can assess combiations of hyperparameters using *full cartesian grid search* or *random grid search* with the h2o package.

Exercise

Let's use random forests to predict median home values in 20,640 census tracts in California.

The code is here.

This tutorial might also be of interest if you would like to go into more detail.

Boosting

Random forests is an *ensemble* method: it is a bagging method for trees. Bagging reduces variance.

There is another ensemble method called **boosting** which also reduces bias. It does so by training subsequent models with the errors of the previous models.

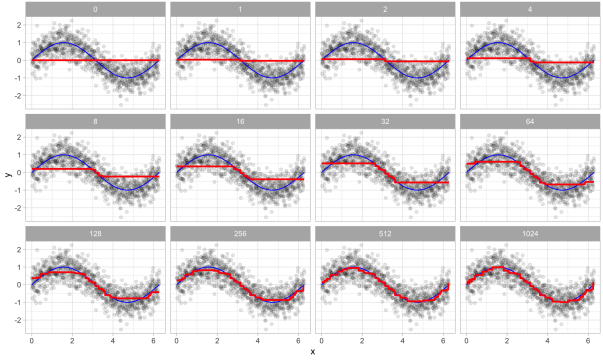
There are two main algorithms:

- *AdaBoost:* this is the original algorithm; you tell subsequent models to punish more heavily observations mistaken by the previous models using weights
- *Gradient boosting:* you train each subsequent model using the residuals (the difference between the predicted and true values)

Boosting

Example of boosting with decision stumps:

- A single predictor x has a true underlying sine wave relationship (blue line) with y
- First tree is a single decision stump (i.e. a single split)
- Each successive decision stump thereafter is fit to the previous one's residuals.



Exercise

Let's implement Extreme Gradient Boosting (XGBoost) with this example code.

You might find this tutorial on XGBoost helpful.

You can also read more about boosting algorithms in R here.