A Short Introduction to CARTs

Andreas Kagoshima, *M. Sc. Student, Karlsruhe Institute of Technology KIT*

First of all some important definitions:

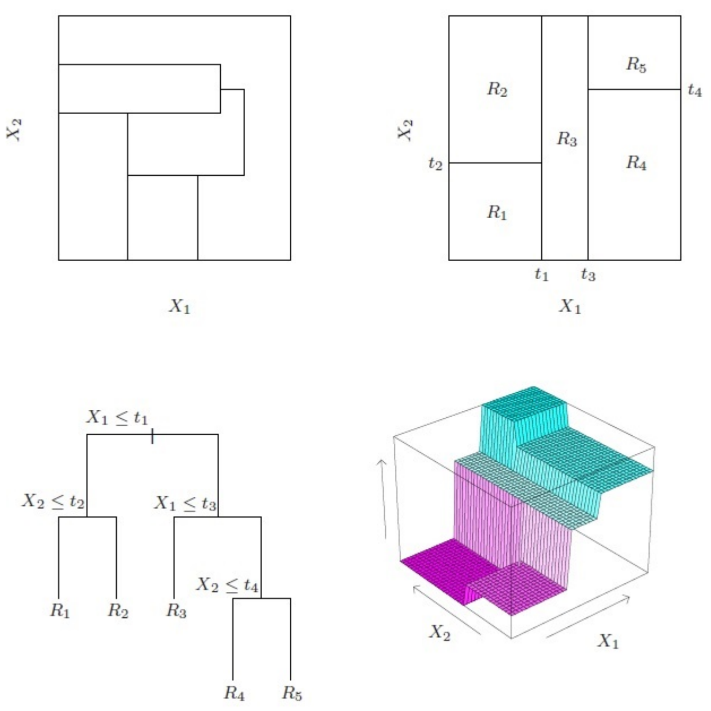


Figure 1: CART: Hastie, Tibshirani, Friedman- The Elements of Statistical Learning

CARTs (Classification and Regression Trees)

- They are called **CARTs**, because they can be used for either: Cassification and Regression.
- We will focus on **regression trees**.
- Regression trees partition the predictor space into simple regions and do a constant linear regression for each region. (Figure 1: Top left: non-admissible regions, top right: admissible regions).
- Since the set of splitting rules used to partition the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss the **ensemble methods**: **Bagging**, **random forests**, and **boosting**.
- Given a partition $R_1, \dots, R_J$ and data-points $[x^{(i)}, y^{(i)}]^T \in \mathbb{R}^{(p+1)}$. Then the resulting (fitted) regression model is given by

$$f(x) = \sum_{j=1}^{J} \hat{c}_j \mathbb{1}_{\{x \in R_j\}},$$

where the $\hat{c}_j$ are given by

$$\hat{c}_j = \text{ave}(y^{(i)} \mid x^{(i)} \in R_j) = \frac{\sum_{i=1}^N y^{(i)} \mathbb{1}_{x^{(i)} \in R_j}}{\sum_{i=1}^N \mathbb{1}_{x^{(i)} \in R_j}}.$$

This is the optimal choice under quadratic loss (quadratic scoring function).

Tree-building process

- To find the best partition $R_1, \dots, R_J$, under quadratic loss

$$\sum_{j=1}^J \sum_{\{i\colon\, x^{(i)} \in R_j\}} (y^{(i)} - \hat{c}_j)^2$$

  is computationally infeasible, because one would have to consider every possile partition of the feature space into J boxes.

- For this reason, one often uses the **top-down greedy approach** that is also known as **recursive binary splitting**.

- The approach is called top-down because it begins at the top of the tree 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 the particular step, rather then looking ahead and picking a split that will lead to a better tree in some future step.

- Formally this algorithm can be described as follows: Let x_j (for some $j \in \{1, \dots, p\}$) be the j-th row of the vector $x \in \mathbb{R}^p$ in the prediction space (x_j = **splitting variable**) and let $s \in \mathbb{R}$ be an one-dimensional bound (s = **split point**). We can then define a pair of half-planes by the splitting rule

$$R_1(j,s) = \{x \in \text{prediction space} \mid x_j \leqslant s\}, \qquad R_2(j,s) = \{x \in \text{prediction space} \mid x_j > s\}.$$

  We then seek the splitting variable j^* and the split point s^*, so that

$$\begin{aligned}(j^*, s^*) &= \text{argmin}_{(j,s)} \left[\min_{c_1} \sum_{\{i\colon\, x^{(i)} \in R_1(j,s)\}} (y^{(i)} - c_1)^2 + \min_{c_2} \sum_{\{i\colon\, x^{(i)} \in R_2(j,s)\}} (y^{(i)} - c_2)^2 \right] \\ &= \text{argmin}_{(j,s)} \left[\sum_{\{i\colon\, x^{(i)} \in R_1(j,s)\}} (y^{(i)} - \hat{c}_1(j,s))^2 + \sum_{\{i\colon\, x^{(i)} \in R_2(j,s)\}} (y^{(i)} - \hat{c}_2(j,s))^2 \right].\end{aligned}$$

  The inner minimization problems are always solved by

$$\hat{c}_1(j,s) = \text{ave}(y^{(i)} | x^{(i)} \in R_1(j,s)), \qquad \hat{c}_2(j,s) = \text{ave}(y^{(i)} | x^{(i)} \in R_2(j,s)).$$

- For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the splitting variables x_j, determination of the best pair (j,s) is feasible.

- One usually uses binary splitting, stopping only, when each terminal node has fewer that some minimum number of observations.

Pruning large trees

- The process above may produce good predictions on the training data, but is likely to overfit the data, leading to poor test set performance. Why?

- A smaller tree with fewer splits (that is, fewer regions $R_1, \dots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

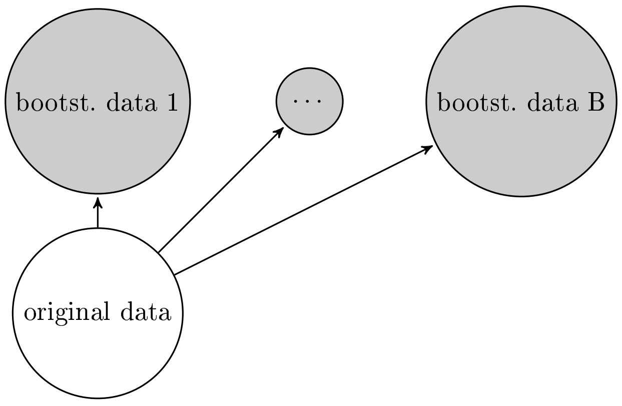
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: A seemingly worthless split early on in the tree might be followed by a very good split- that is, a split that leads to a large reduction in RSS later on.
- A better strategy is to grow a very large tree T_0 and then prune it back in order to obtain a subtree.
- Cost complexity pruning- also known as weakest link pruning- is used to do this.
- We consider a sequence of trees indexed by a non-negative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{\{i:\ x^{(i)} \in R_m\}} (y^{(i)} - \hat{c}_m) + \alpha|T|$$

  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 m-th terminal node, and $\hat{c}_m$ is the mean of the training observations in R_m.
- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree correspoinding to $\hat{\alpha}$.

Bagging

- **Bagging** = Bootstrap aggregation (we use trees in an ensemble).
- Usually the bootstrap is used to estimate the standard error or the bias of an estimator.
- **Bagging** is a general purpose procedure for reducing the variance of a statistical learning method; we introduce it here, because it is particularly useful and frequently used in the context of **decision trees**.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n.
- In other words, averaging a set of observations reduces the variance. Of course, this is not practical because we generally do not have access to multiple training sets.
- Instead, we create **bootstrap data-sets**. (This means we create data-sets of the same size, consisting out of uniformly drawn data-points of the original data-set).



Usually we create a few hundred **bootstrap data-sets**.

- On each **bootstrap data-set** we now create a big tree without pruning. Hence each tree has high variance, but low bias.
- Averaging these B trees reduces the variance. This is an **ensemble method**.
- In formulas: We generate B different **bootstrapped data-sets**. We then train our method on the b-th bootstrapped training set in order to get $\hat{f}^{*(b)}(x)$, the prediction at a point x. We then average all the predictions to obtain

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

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped sub-sets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the **out-of-bag** (OOB) observation.
- We can predict the response for the i-th observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the i-th observation, which we average.
- The estimate is essentially the LOO cross-validation error for bagging, if B is large.

Random Forests (RFs)

- **Random forests** is also an **ensemble method**. **RFs** provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on **bootstrapped data-set**.
- But when building these decision trees, each time a split in a tree is considered, a random selection 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.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$- that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Boosting

- **Boosting** is an **ensemble method**. Like **bagging**, **boosting** is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating **bootstrapped data-sets**. Then a seperate decision tree is fit to each copy and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a **bootstrap data-set**, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: Each tree is grown using information from previously grown trees.
- Boosting algorithm 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 f by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   (c) Update the residuals
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x).$$
3. Output the boosted model
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

- This is also an **ensemble method**.

What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to **fitting the data hard** and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ shows the process down even further, allowing more and different shaped trees to attack the residuals.

Tuning parameters for boosting

- The number of trees B: Unlike **bagging** and **random forests** (**RF**), 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.
- The shrinkage parameter λ is a small positive number. This ontrols 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.
- The number of splits d 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 and resulting in an additive model. 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.

CARTs and RFs for Density Forecasting

- First of all Regression trees and Random Forests are typically used for regression.
- But in return they can also be used for density forecasting.
- Consider a tuple $[x, y]^T \in \mathbb{R}^{p+1}$ and consider the case of a regression tree. Then
$$P[y = y^{(k)} | x \in \text{node t}] \qquad (k \hat{=} \text{"class" } k)$$
can be estimated with the estimator
$$\hat{f}(y = y^{(k)} | x \in \text{node t}) := \frac{\#\{\text{data-points } y: \ y = y^{(k)} \wedge x \in \text{node t}\}}{\#\{\text{data-points } y: \ x \in \text{node t}\}}.$$

- Consider a tuple $[x, y]^T \in \mathbb{R}^{p+1}$, B bootstrap data-sets and consider the case of a RF. Then
$$P[y = y^{(k)} | x \in \text{node t}] \qquad (k \hat{=} \text{"class" } k)$$
can be estimated with the estimator (created from the b-th bootstrap data-set)
$$\hat{f}^{(b)}(y = y^{(k)} | x \in \text{node t}) := \frac{\#\{\text{data-points } y \in \ b\text{-th bootstrap data-set} : \ y = y^{(k)} \wedge x \in \text{node t}\}}{\#\{\text{data-points } y \in \ b\text{-th bootstrap data-set} : \ x \in \text{node t}\}},$$
and through an ensemble approach one could use
$$\hat{f}_{\text{RF}}(y = y^{(k)} | x \in \text{node t}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{(b)}(y = y^{(k)} | x \in \text{node t})$$
as a density forecast.

Literature

- The Elements of Statistical Learning- Hastie, Tibshirani, Friedman,
- An Introduction to Statistical Learning- James, Witten, Hastie, Tibshirani,
- Lecture Notes: "Statistisches Lernen"- Universität Freiburg, 2014,
- Stanford University Online Lecture: "Statistical Learning".