Quant III

Lab 12: Tree-based Method

Junlong Aaron Zhou

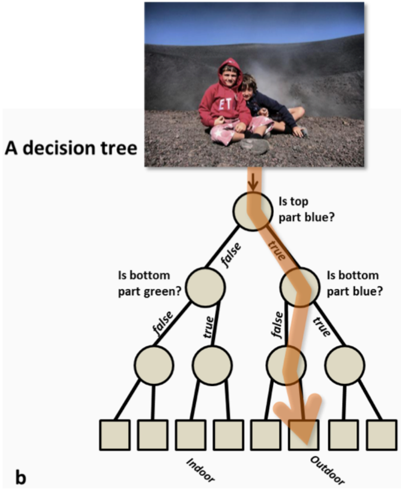
December 11, 2020

Announcement

- Final Exam:
  - I will distribute it Friday, December 18th, 9:45 am (**EST**)
  - Due: Sunday, December 20th, 11:59 pm (**EST**)
- Replication: due Friday, December 18th, 11:59 pm (**EST**)
- Final homework:
  - Due: Friday, December 18th, 11:59 pm (**EST**)
  - You should submit a pdf and a r object,
  - which includes a object for user: The input being a design matrix, and output being predictions.
- I add an extra office hour for next week, check **here**

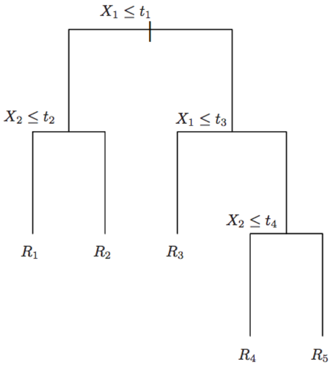
A Binary Decision Tree

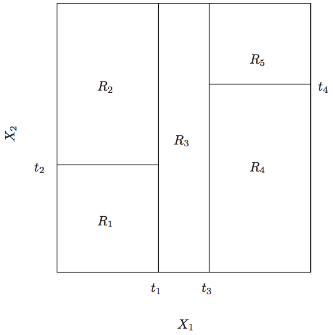
- binary tree: each node has either 2 children or 0 children



Binary Decision Tree on R^2

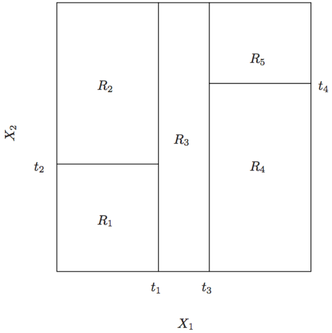
- Consider a binary tree on $\{(X_1, X_2) \mid X_1, X_2 \in R\}$

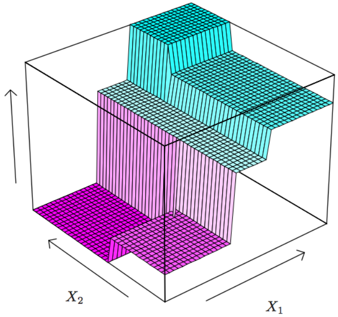




Binary Regression Tree on R^2

- Consider a binary tree on $\{(X_1, X_2) \mid X_1, X_2 \in R\}$





Recursive binary splitting

1. Split the predictor space on one dimension (on predictor) into two sets.
2. Split each of the resulting sets into two sets again.
3. Repeat until you have only a small number of observations left in each of the terminal sets.

Recursive binary splitting

- For a given j and s, split the space of each predictor x_j, $j = 1, \ldots, p$, into two regions:

$$R_-(j, s) = \{\boldsymbol{x} | x_j < s\}$$
$$R_+(j, s) = \{\boldsymbol{x} | x_j \geq s\}$$

$R_-(j, s)$ is the region of predictor space where $x_j < s$.

- The splitting is done by finding j and s that minimizes

$$\sum_{i:\boldsymbol{x}_i \in R_-(j,s)} (y_i - \hat{y}_{R_-})^2 + \sum_{i:\boldsymbol{x}_i \in R_+(j,s)} (y_i - \hat{y}_{R_+})^2$$

where $\hat{y_R} = E(y|x_i \in R)$ - Each resulting region is then split further until each region contains no more than N^* observations (where N^* is a small number).

Classification tree

- Suppose that the outcome variable is nominal $y_i \in \{c_1, ..., c_K\}$ so that y_i can be one of K categories.
- Prediction for observation i: $\hat{p_{ik}}$
- Prediction in that region R_j: $\hat{p_{jk}}$
- The idea is exactly the same except that when calculating $\hat{y}_{iR_m}$ we use the most frequent outcome category for observations in the set R_m.
- The only difference is that RSS is not used to find splits or to calculate the optimal complexity parameter α.

Impurity Measures

- Classification error:

$$1 - \max_k(\hat{p}_{jk})$$

where $\hat{p}_{jk}$ is the proportion of observations in the region R_j that are from the class k. If a region R_j contains observations from many classes, then it does not classify them well. - Gini index

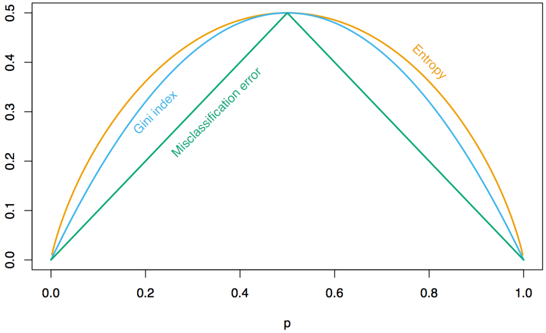
$$\sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

- Cross-entropy:

$$-\sum_{k=1}^{K} \hat{p}_{jk} \ln \hat{p}_{jk}$$

Two-Class Node Impurity Measures

- Consider binary classification
- Let p be the relative frequency of class 1.
- Here are three node impurity measures as a function of p



Bagging

- Combine predictions from multiple regression trees to improve their individual predictive capacity:

  1. Take a bootstrap sample $b = 1, ..., B$ from your data (with replacement).
  2. Estimate a deep regression tree for sample b (without prunning).
  3. Calculate prediction $\hat{g}^b(\boldsymbol{x})$ given that tree.
  4. Calculate the *ensemble prediction*

$$\hat{g}(\boldsymbol{x}) = \sum_{b=1}^{B} \hat{g}^b(\boldsymbol{x}) / B$$

- Averaging over a large number of high-variance low-bias trees results in lower-variance bag of trees.

Random Forest

- Two disadvantages of bagging:
  1. Computationally complex
  2. Each tree are too similar
- Random forest solve these issues:
  1. Take a bootstrap sample $b = 1, ..., B$ from your data (with replacement).
  2. Build regression tree by randomly selecting at each step $\mathbf{m} < \mathbf{p}$ predictors (usually $m \approx \sqrt{p}$).
  3. Calculate prediction $\hat{g}^b(\boldsymbol{x})$ from that tree.
  4. Calculate the *ensemble prediction*

$$\hat{g}(\boldsymbol{x}) = \sum_{b=1}^{B} \hat{g}^b(\boldsymbol{x})/B$$

- Mathematically, RF should recover true $g(x)$

Different Forest

- You may see different forests, e.g.
  - Generalized Random Forest (Athey, et al.)
- Most the difference lies in the spliting criterion.

Optional: Boosting

- Ensemble methods combine multiple models
- **Parallel ensembles**:
  - each model is built independently
  - e.g. bagging and random forests
  - Main Idea: Combine many (high complexity, low bias) models to reduce variance
- **Sequential** ensembles:
  - Models are generated sequentially
  - Try to add new models that do well where previous models lack

Intuition

- Suppose we want to solve a classification problem (spam or not)
- A weak learner is a classifier that does slightly better than random.
    - Weak learners are like “rules of thumb”:
    - If an email has “Viagra” in it, more likely than not it’s spam.
    - Email from a friend is probably not spam.
    - A linear decision boundary.
- Can we combine a set of weak classifiers to form **single** classifier that makes accurate predictions
- Yes! Boosting solves this problem. [Rob Schapire (1990).]
- In general, you can use different classifier. (e.g., Grimmer et al, 2017)

Basic Idea

- Linear combination of basis functions:

$$f(x) = \sum_{m=1}^{M} \nu_m g_m(x)$$

- We'll consider learning by **empirical risk minimization**:

$$\hat{f} = \underset{f \in \mathbb{F}}{argmin} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f\left(x_i\right)\right),$$

  for some **loss function** $\ell(y, \hat{y})$.

- Write ERM objective function as

$$J(v_1, \ldots, v_M, h_1, \ldots, h_M) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, \sum_{m=1}^{M} v_m h_m(x)\right).$$

- How to optimize J? i.e. how to learn?

Gradient-Based Methods

- **Suppose** our base hypothesis space is parameterized by $\Theta = R^b$:

$$J(v_1, \ldots, v_M, \theta_1, \ldots, \theta_M) = \frac{1}{n} \sum_{i=1}^{n} \ell \left(y_i, \sum_{m=1}^{M} v_m h(x; \theta_m) \right).$$

- Can we can differentiate J w.r.t. v_m's and θ_m's? Optimize with SGD?
- For **some** hypothesis spaces and typical loss functions, yes!
- Neural networks fall into this category! ($h_1, \ldots, h_M$ are neurons of last hidden layer.)
- We have gradient boost machine, Adaboost, etc.
- For trees, we have gradient tree boosting