Trees

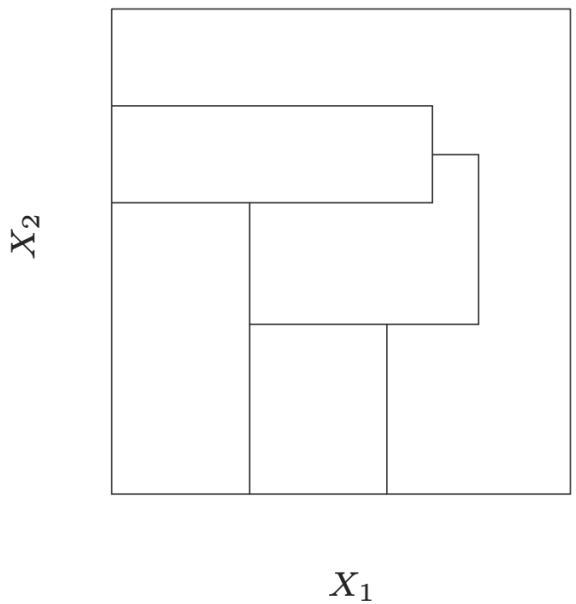
ORIE 4741

October 8, 2020

Table of Contents

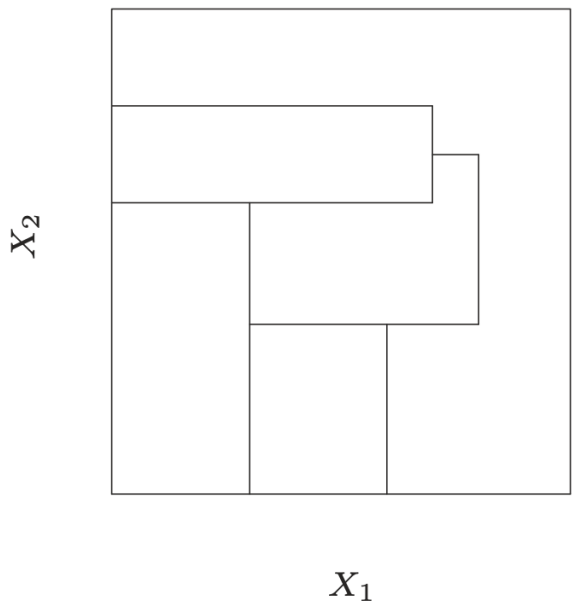
Split the feature space

Consider a regression problem with continuous response Y and features X_1 and X_2, each taking values in the unit interval:



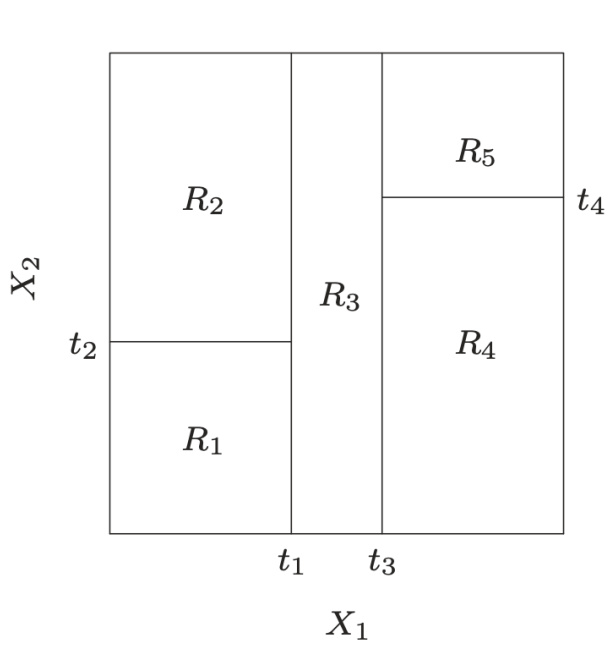
Split the feature space

Consider a regression problem with continuous response Y and features X_1 and X_2, each taking values in the unit interval:

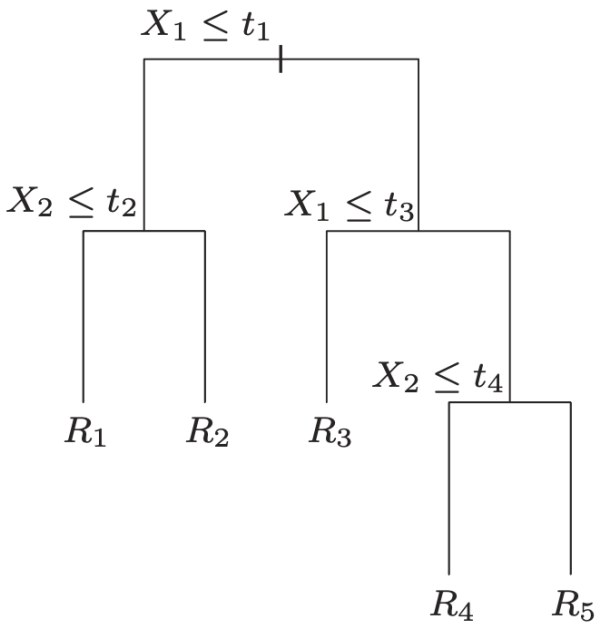


hard to describe each small feature space

Recursive binary splitting



(a) Recursive binary split



(b) Description for each small feature subspace

Figure 1: Recursive binary tree. For each region R_m, we can predict Y with a constant c_m: $\hat{f}(X) = \sum_{m=1}^{5} c_m \mathrm{I}(\{(X_1, X_2) \in R_m\})$.

Regression trees

Suppose we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

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

What are model parameters here?

Regression trees

Suppose we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

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

What are model parameters here?

- Hyperparameter to choose **beforehand**: the number of regions M.
- The divided regions to choose **using data**: $R_1, ..., R_M$.
- Constants to choose **using data**: $c_1, \ldots, c_M$.

Regression trees

Suppose we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

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

What are model parameters here?

- Hyperparameter to choose **beforehand**: the number of regions M.
- The divided regions to choose **using data**: $R_1, ..., R_M$.
- Constants to choose **using data**: $c_1, \ldots, c_M$.

Ideal case: find the partition which gives the smallest loss (ℓ_2 loss or sum of squares).

Construct a regression tree

Reality: finding the best binary partition $R_1, \ldots, R_m$ is computationally infeasible. We can only find approximately "best" partition.

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

Construct a regression tree

Reality: finding the best binary partition $R_1, \ldots, R_m$ is computationally infeasible. We can only find approximately "best" partition.

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

Given R_m, can you find the best c_m?

Construct a regression tree

Reality: finding the best binary partition $R_1, \ldots, R_m$ is computationally infeasible. We can only find approximately "best" partition.

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

Given R_m, can you find the best c_m?

- the best c_m is just the mean of y_i in region R_m:

$$\hat{c}_m = \mathrm{ave}(y_i | x_i \in R_m)$$

Greedy algorithm to construct a regression tree

Search over splitting variable j and split point s:

$$R_1(j,s) = \{X \mid X_j \leq s\}, R_2(j,s) = \{X \mid X_j > s\}$$

Greedy algorithm to construct a regression tree

Search over splitting variable j and split point s:

$$R_1(j,s) = \{X \mid X_j \leq s\}, R_2(j,s) = \{X \mid X_j > s\}$$

Seek the pair that minimizes the prediction error:

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

- Inner minimization is solved by using the regional average of y_i.
- Searching s can be done very quickly.

Classification tree

For a classification problem with possible outcome $1, ..., K$, define $N_m = \#\{x_i \in R_m\}$ and $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$.

Classification tree

For a classification problem with possible outcome $1, ..., K$, define $N_m = \#\{x_i \in R_m\}$ and $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$.

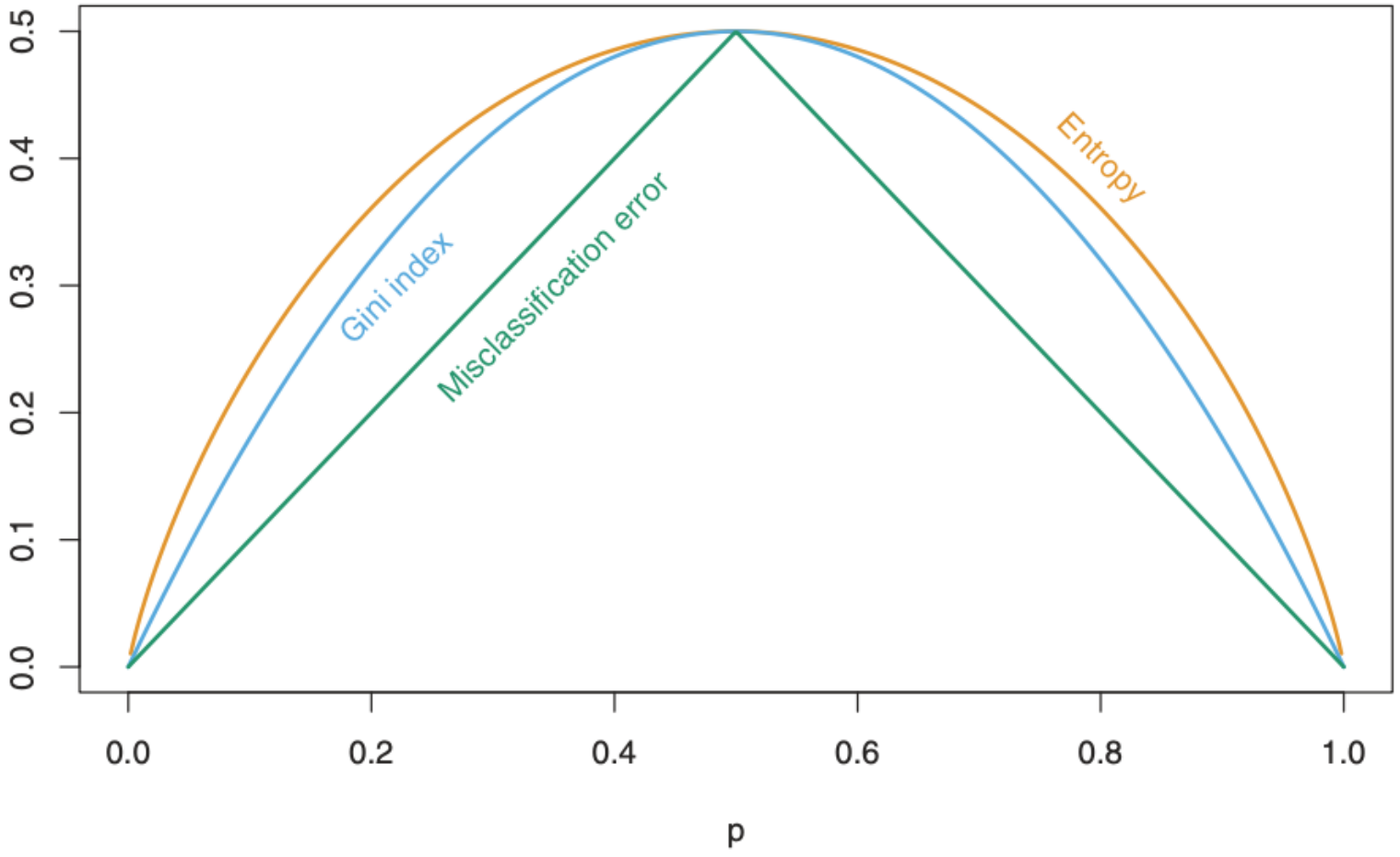
Replace the squared error loss with loss functions for classification.

Misclassification error: $\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$.

Gini index: $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.

Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

Loss functions for classification trees



Hyperparameter: tree size

- Tree size controls the model complexity.
- How large should we grow the tree?

Hyperparameter: tree size

- Tree size controls the model complexity.
- How large should we grow the tree?
  - A very large tree might overfit the data and thus have high variance.
  - A small tree might not capture the important structure.

Hyperparameter: tree size

- Tree size controls the model complexity.
- How large should we grow the tree?
  - A very large tree might overfit the data and thus have high variance.
  - A small tree might not capture the important structure.
  - The optimal tree size should be adaptively chosen from the data

Pros and Cons

Pros and Cons

- Pros: Interpretability
- Cons: Instability (large variance): an error in the top split is propagated down to all of the splits below it.

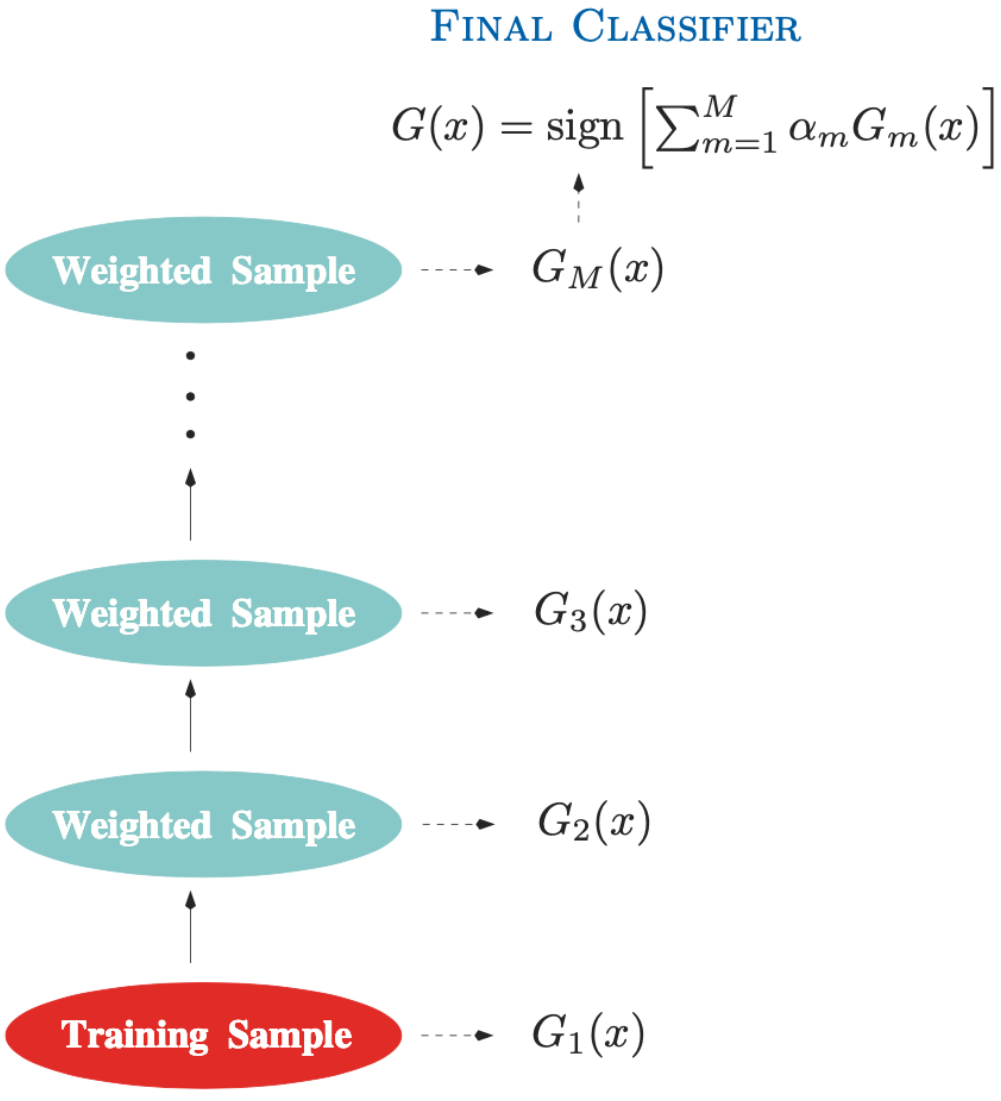
Boosting methods

- Intuition: combines the outputs of many "weak" learners to produce a powerful "committee."

Boosting methods

- Intuition: combines the outputs of many "weak" learners to produce a powerful "committee."
- Methodology:
  - Sequentially apply the weak learners to repeatedly modified versions of the data.
  - Produce a sequence of weak learners $G_m(x), m = 1, 2, \ldots, M$.
  - At step m, observations that were predicted worse by $G_{m-1}(x)$ will have larger weights.
  - Observations that are difficult to predict receive ever-increasing influence.
  - Combine all prediction $G_m(x), m = 1, 2, \ldots, M$ to a single weighted average.

Boosting Visualization



Adaboost

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Gradient Boosting Regression Tree

GBRT in Pseudo Code

Input: ℓ, α, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{A}$
$H = 0$
for $t{=}1{:}T$ **do**
  $\forall i : t_i = y_i - H(\mathbf{x}_i)$
  $h = \text{argmin}_{h \in \mathbb{H}} (h(\mathbf{x}_i) - t_i)^2$
  $H \leftarrow H + \alpha h$
end
return H

Model hyperparameters

- Number of weak learners (trees)
  - Can overfit with too many trees.
- Tree size
  - Mostly below 10.

Bagging methods

Main idea: improve model prediction through Bootstrap.

- Generate B bootstrap examples.
- Fit your model (tree) on each of the bootstrap example.
- Average all prediction into a single one.

Each bootstrap tree will typically involve different features than the original, and might have a different number of terminal nodes.

Pros and Cons

Pros: average many noisy but approximately unbiased trees, and hence reduce the variance.

Cons: constructed trees still have high correlation

- Boosting appears to dominate bagging on most problems, and became the preferred choice.

Random forest: improve the variance reduction of bagging

Given B identically distributed random variables (R.V.) with variance σ^2:

- If all B R.V. are independent, the average has variance σ^2/B.
- If all B R.V. are dependent and have positive correlation ρ, the average has variance $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
  - cannot decrease the variance below $\rho\sigma^2$.

Random forest: improve the variance reduction of bagging

Given B identically distributed random variables (R.V.) with variance σ^2:

- If all B R.V. are independent, the average has variance σ^2/B.
- If all B R.V. are dependent and have positive correlation ρ, the average has variance $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
  - cannot decrease the variance below $\rho\sigma^2$.

Remedy:

- Before each split, select $m \leq p$ of the features at random as candidates for splitting.
  - Typical values of m are $\sqrt{p}$ or even as low as 1.

Random Forest Algorithm

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 $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Model hyperparameters

- Number of trees
  - Hardly overfit
- Number of variables to randomly select from at each split

Bagging, random forest, and gradient boosting in real data

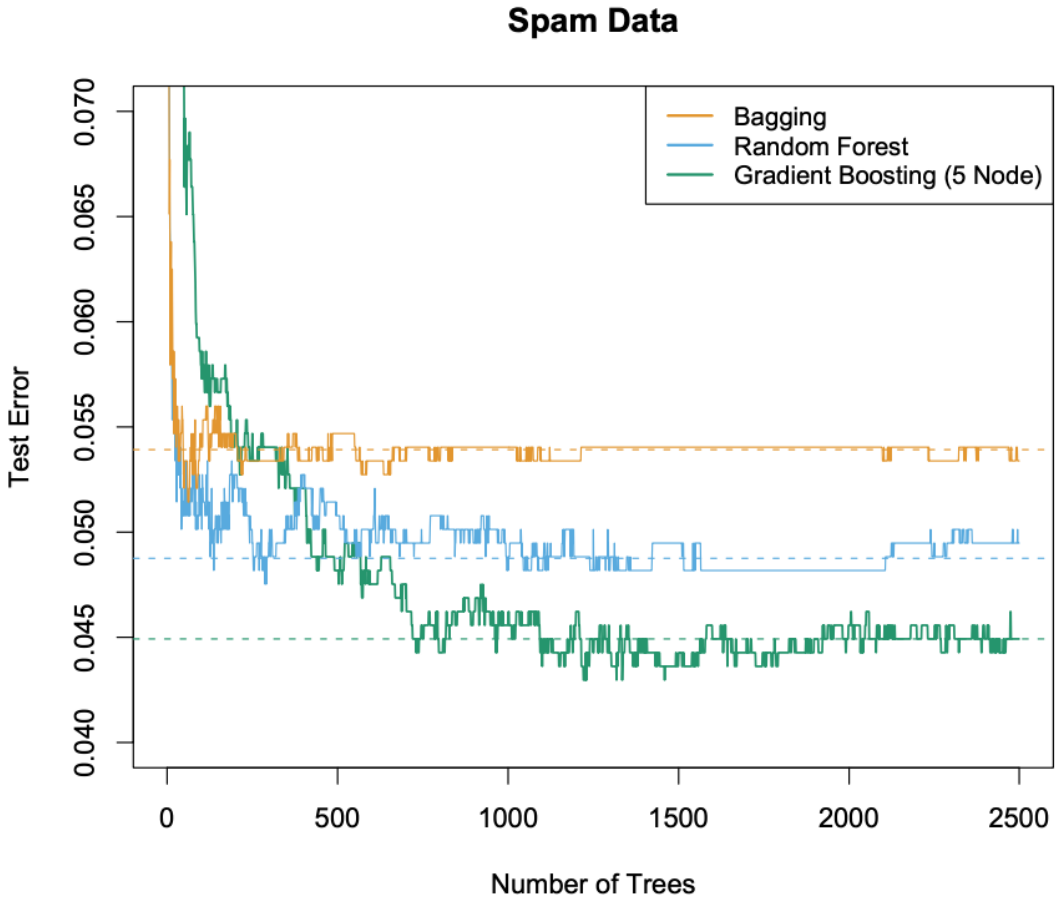


Figure 2: For boosting, 5-node trees were use.

Reference

- Chapters 8.7, 9 and 15 of Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani and Jerome Friedman, free at https://web.stanford.edu/ hastie/ElemStatLearn/.