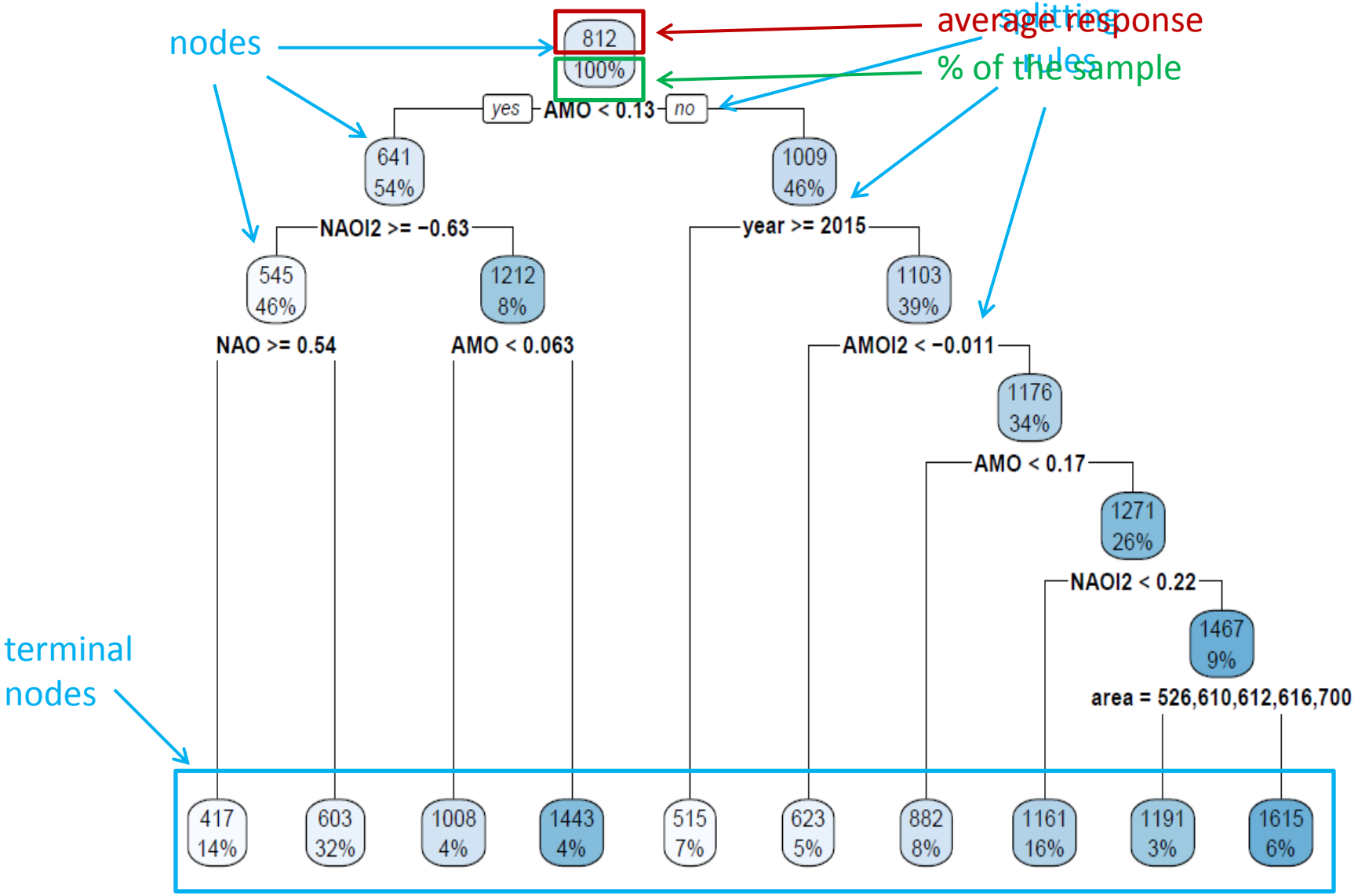
# Tree-based methods of machine learning

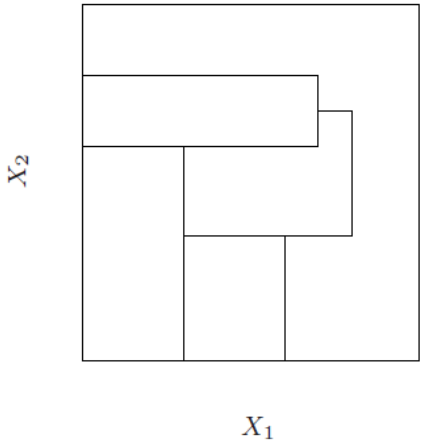
V. Lyubchich

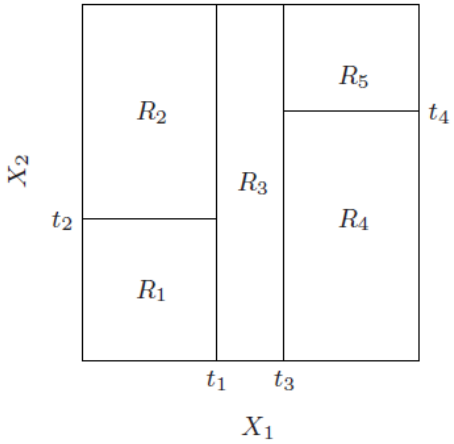
2020-03-12

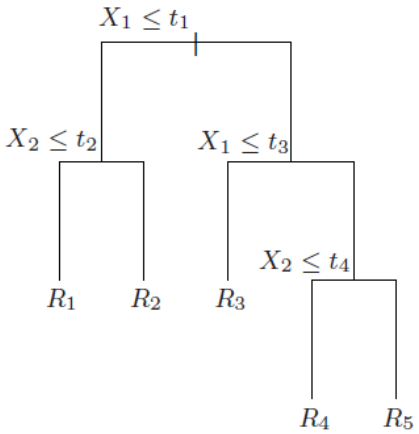
# CART = Classification and Regression Tree

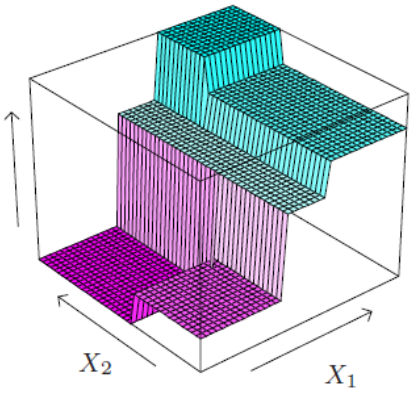


# Partitions in CART



If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of $y_i$ in region $R_m$:

$$\hat{c}_m = \text{ave}(y_i|x_i \in R_m). \tag{9.11}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable $j$ and split point $s$, and define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}. \tag{9.12}$$

Then we seek the splitting variable $j$ and split point $s$ that solve

$$\min_{j,\ s}\Big[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\Big]. \tag{9.13}$$

For any choice $j$ and $s$, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i|x_i \in R_2(j,s)). \tag{9.14}$$

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

The preferred strategy is to grow a large tree $T_0$, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using *cost-complexity pruning*, which we now describe.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning $T_0$, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes by $m$, with node $m$ representing region $R_m$. Let $|T|$ denote the number of terminal nodes in $T$. Letting

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{9.15}$$

we define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{9.16}$$

The idea is to find, for each $\alpha$, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$. The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data. Large values of $\alpha$ result in smaller trees $T_\alpha$, and conversely for smaller values of $\alpha$. As the notation suggests, with $\alpha = 0$ the solution is the full tree $T_0$.

# Tree Problems

- Instability of trees
- Lack of smoothness
- Difficulty in capturing additive structure

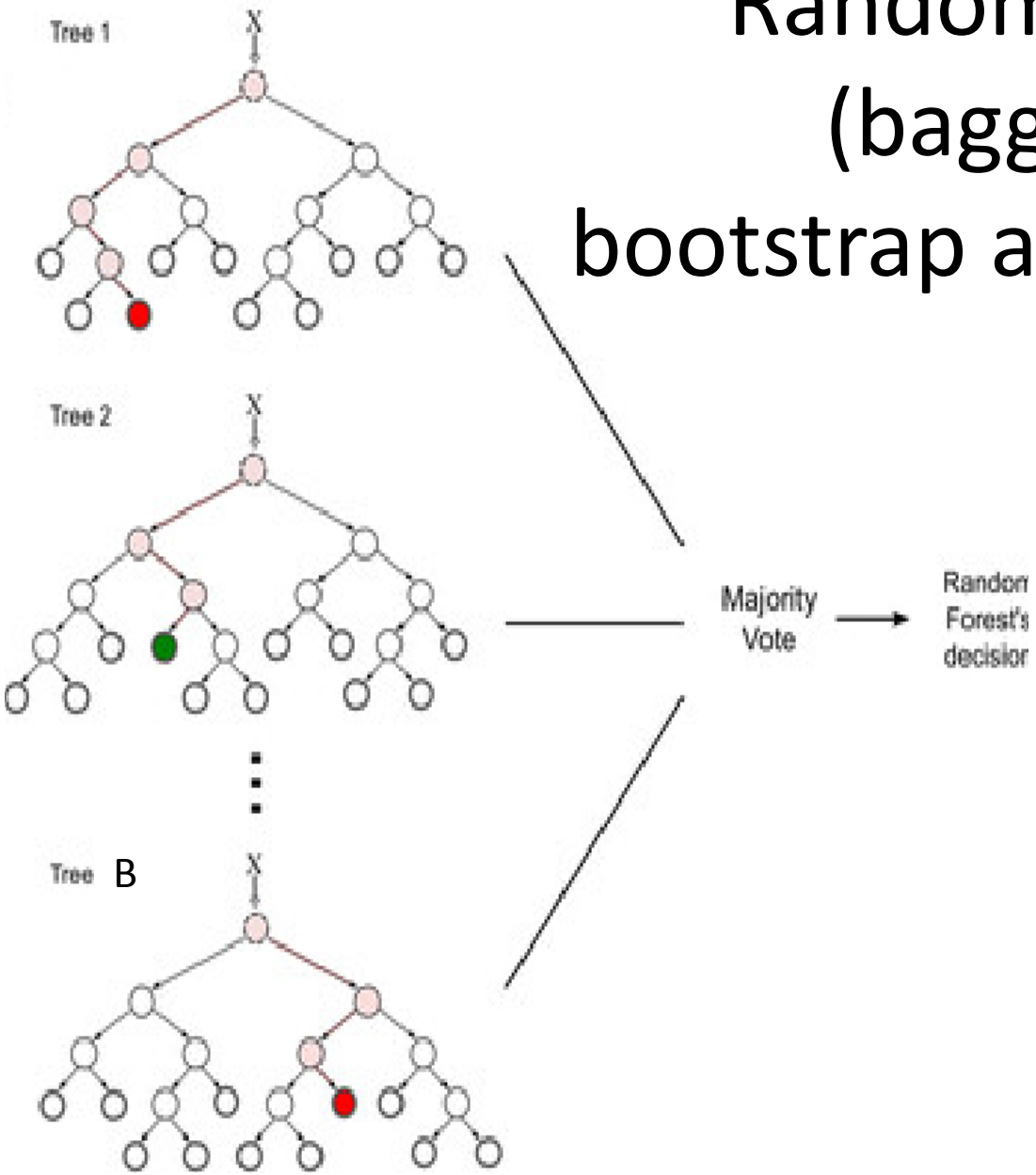
$$Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \varepsilon$$

For example,

$$Income = \$10\, I(age > 18) + \$30 I(schoolyrs > 12)$$

has 2 thresholds, but CART would likely need 3.

# Random Forest (bagging = bootstrap aggregation)



**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.

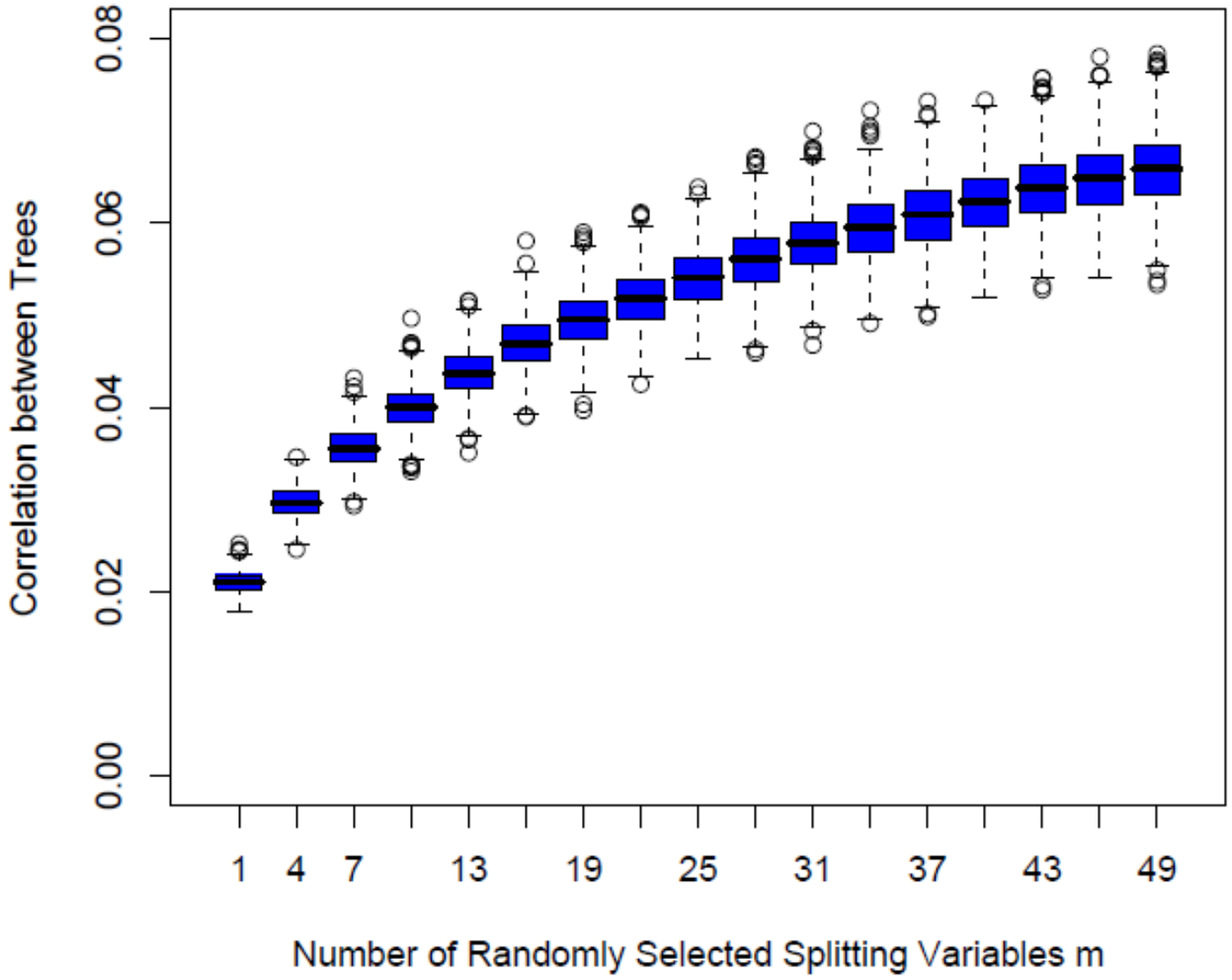
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$:

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

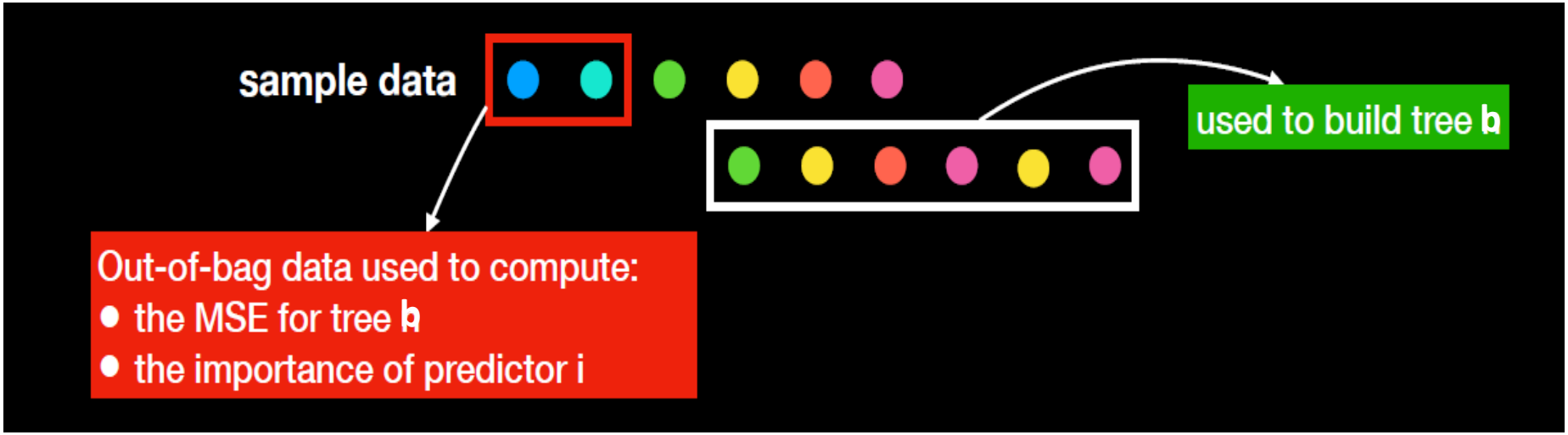
*Classification:* Let $\hat{C}_b(x)$ be the class prediction of the $b$th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

# Decorrelation



# Random Forest = many trees

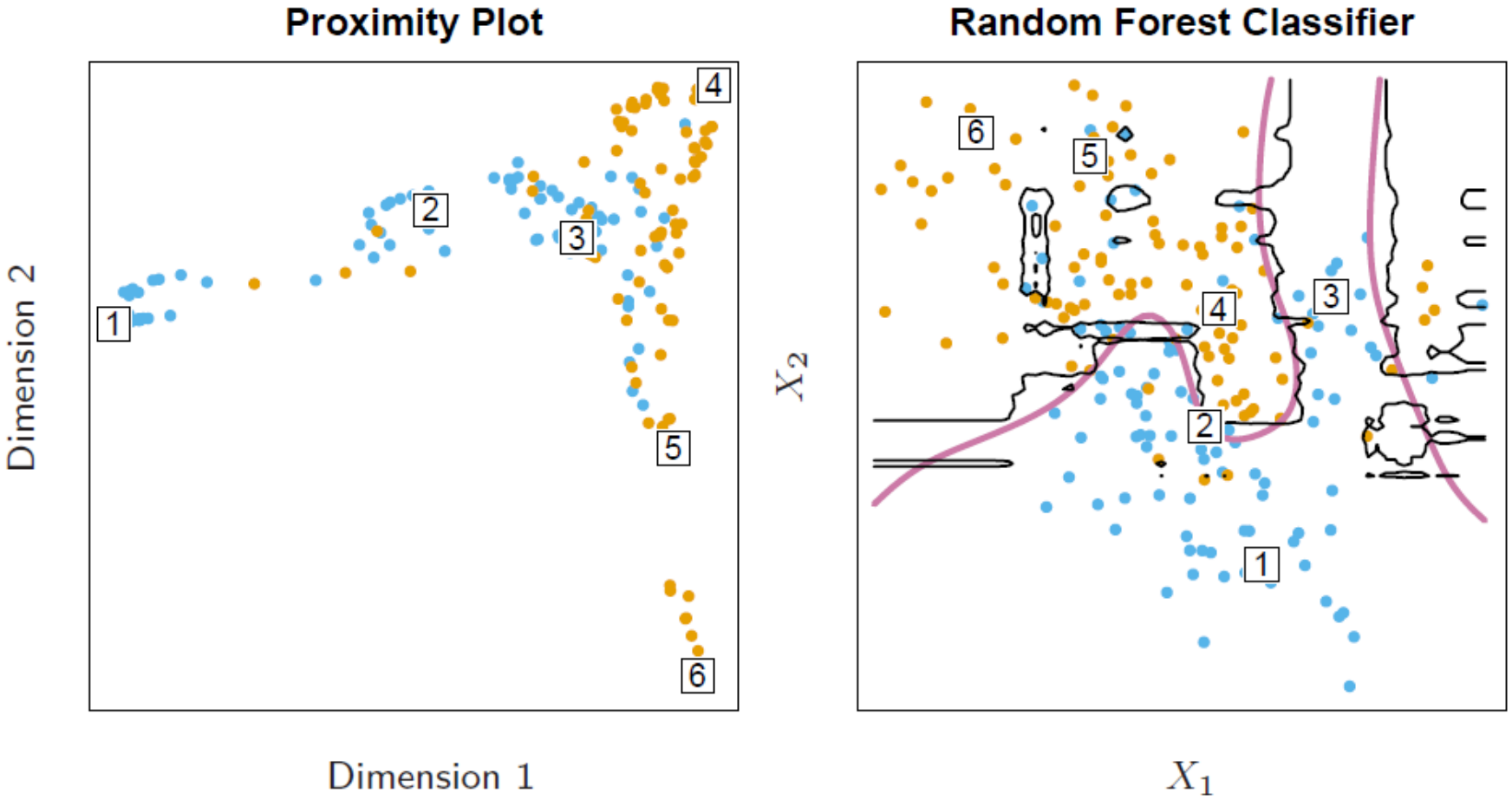
Bootstrap sample of data cases (sample with replacement, of the same size as the original)




$$I_i = B^{-1} \sum_{b=1}^{B} ({}^{\mathrm{RF}}MSE(i)_b - {}^{\mathrm{RF}}MSE_b)$$

At each split, subsample (sample without replacement, smaller than the original, about 1/3) variables for consideration,
to make trees more different from one another.

# Proximitv Plot




In growing a random forest, an N × N proximity matrix is accumulated for the training data. For every tree, any pair of oob observations sharing a terminal node has their proximity increased by one. This proximity matrix is then represented in two dimensions using multidimensional scaling. The idea is that even though the data may be high-dimensional, involving mixed variables, etc., the proximity plot gives an indication of which observations are effectively close together in the eyes of the random forest classifier.

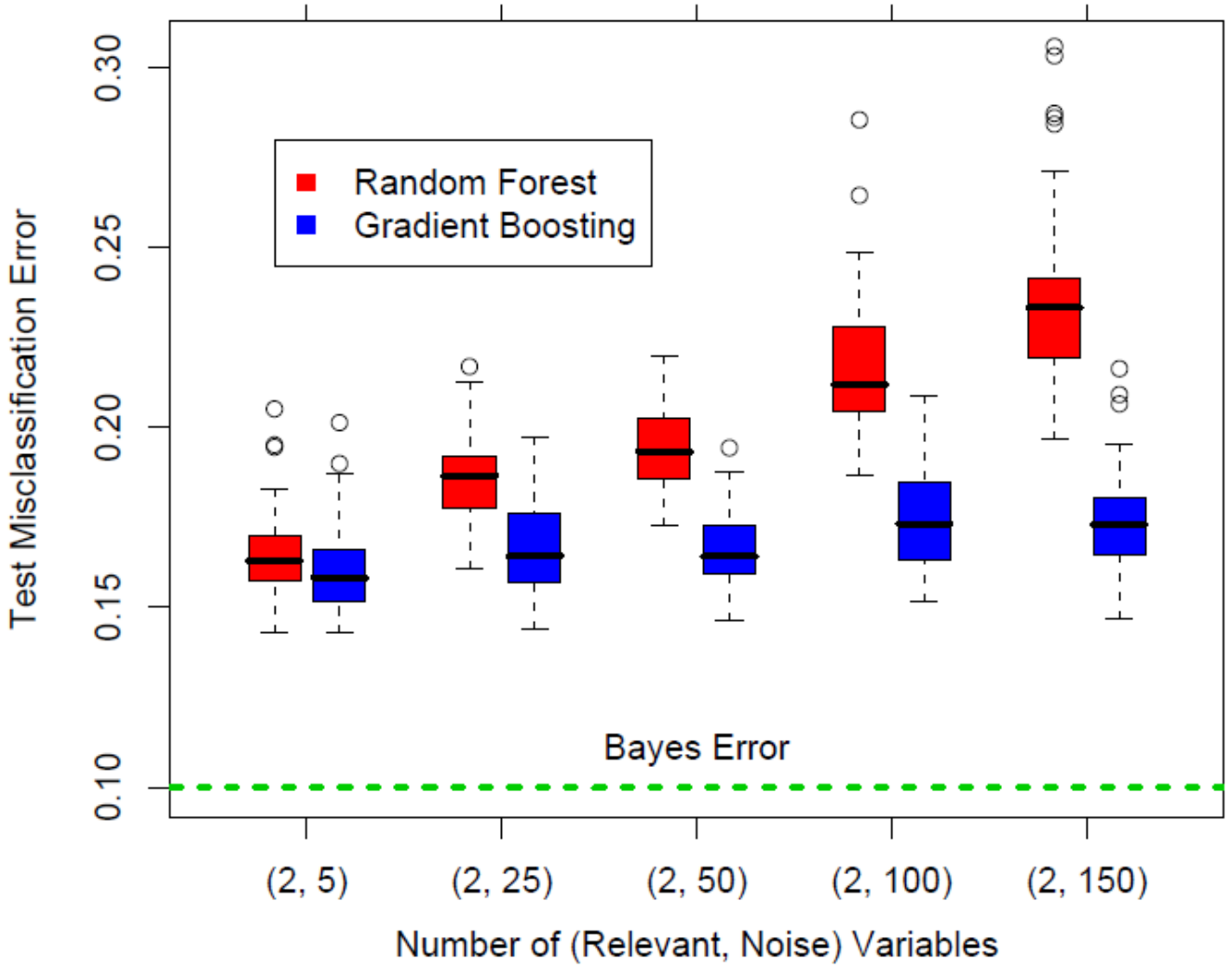
# Random Forests and Overfitting

Another claim is that random forests "cannot overfit" the data. It is certainly true that increasing $B$ does not cause the random forest sequence to overfit; like bagging, the random forest estimate (15.2) approximates the expectation

$$\hat{f}_{\mathrm{rf}}(x) = \mathrm{E}_{\Theta} T(x;\Theta) = \lim_{B\to\infty} \hat{f}(x)_{\mathrm{rf}}^{B} \tag{15.3}$$

with an average over $B$ realizations of $\Theta$. The distribution of $\Theta$ here is conditional on the training data. However, *this limit can overfit the data*; the average of fully grown trees can result in too rich a model, and incur unnecessary variance. Segal (2004) demonstrates small gains in performance by controlling the depths of the individual trees grown in random forests. Our experience is that using full-grown trees seldom costs much, and results in one less tuning parameter.

# RF v. Gradient Boosting



# AdaBoost

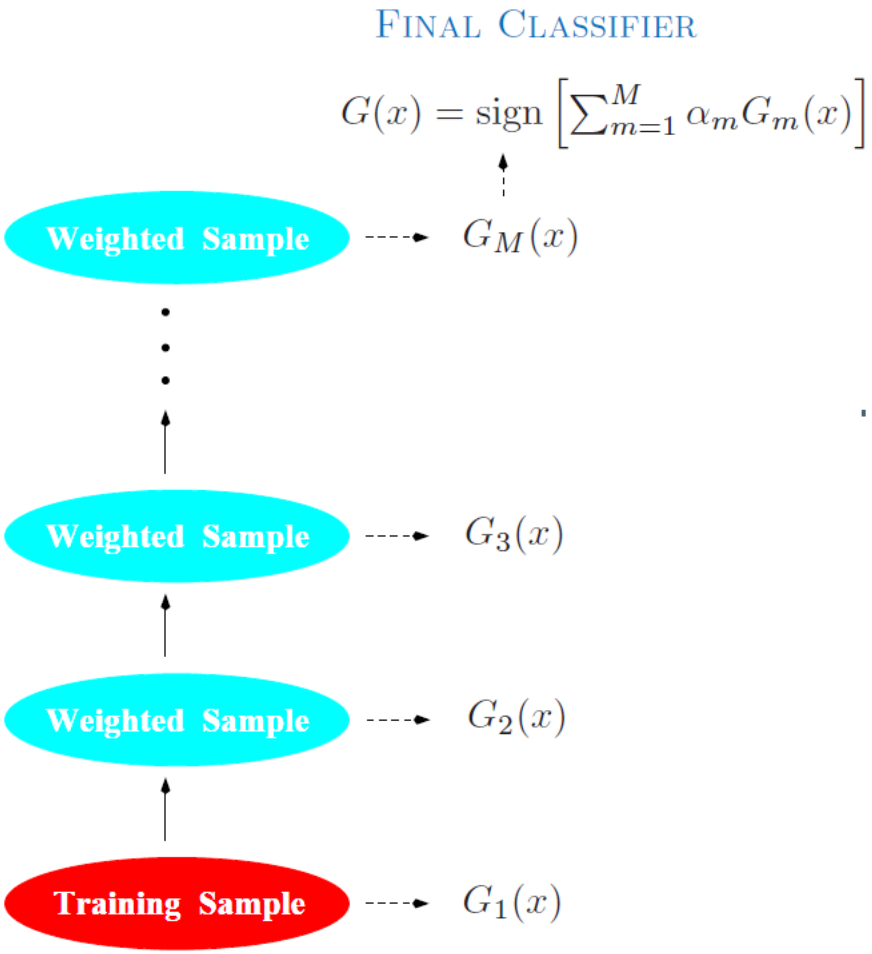


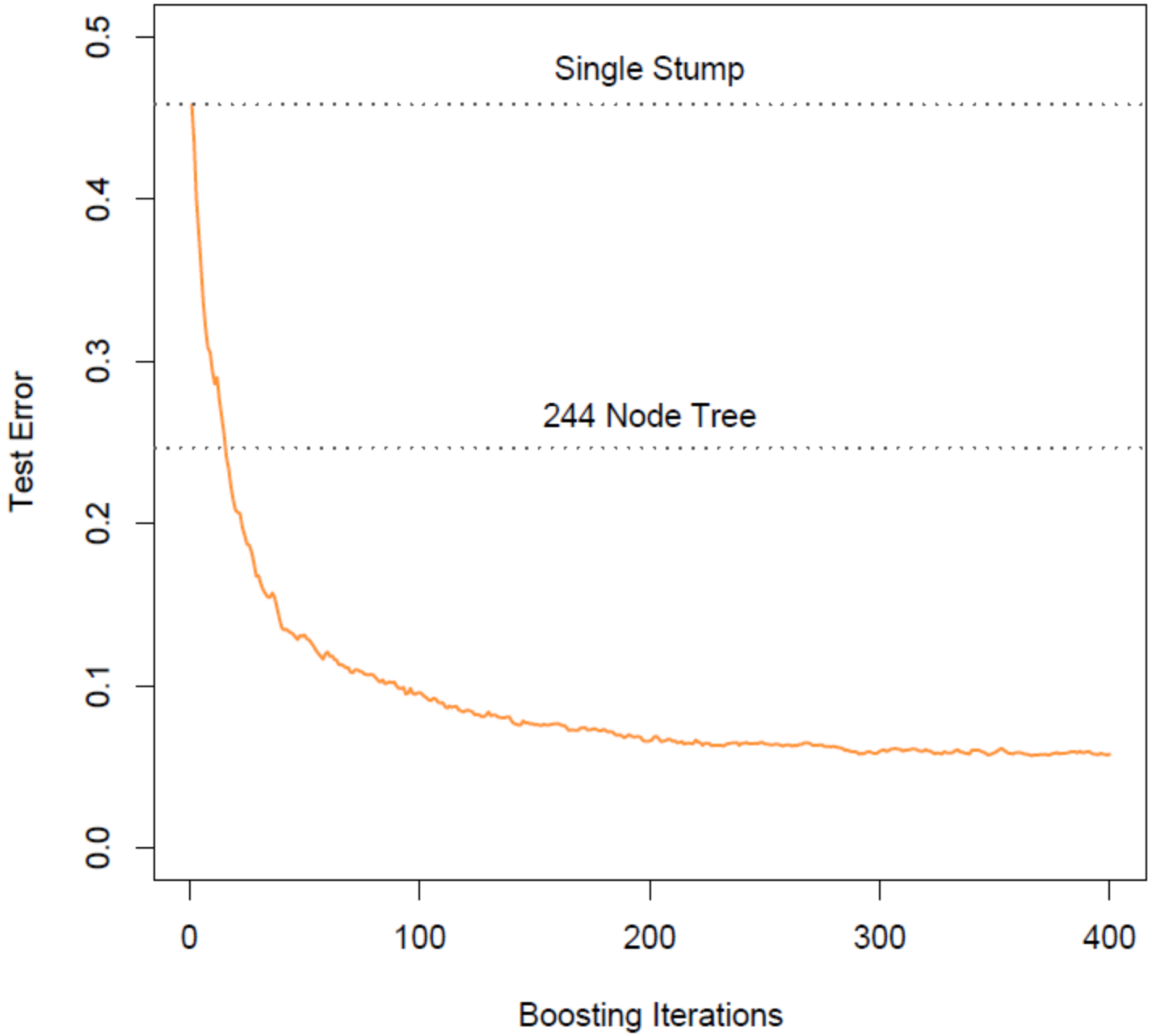

**FIGURE 10.1.** *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

**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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

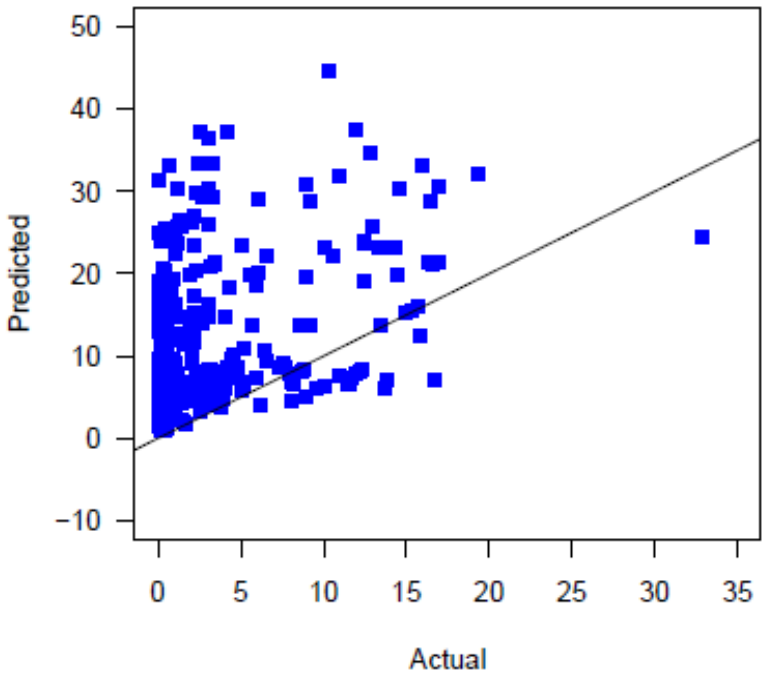
**Algorithm 10.3** *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

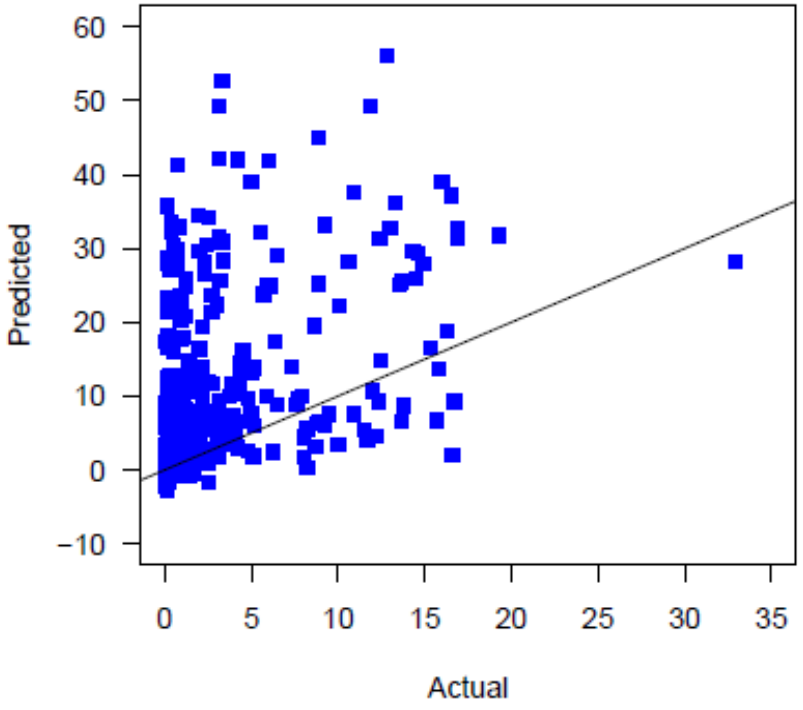

Single Stump
244 Node Tree
Test Error
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
Boosting Iterations

# Own Examples

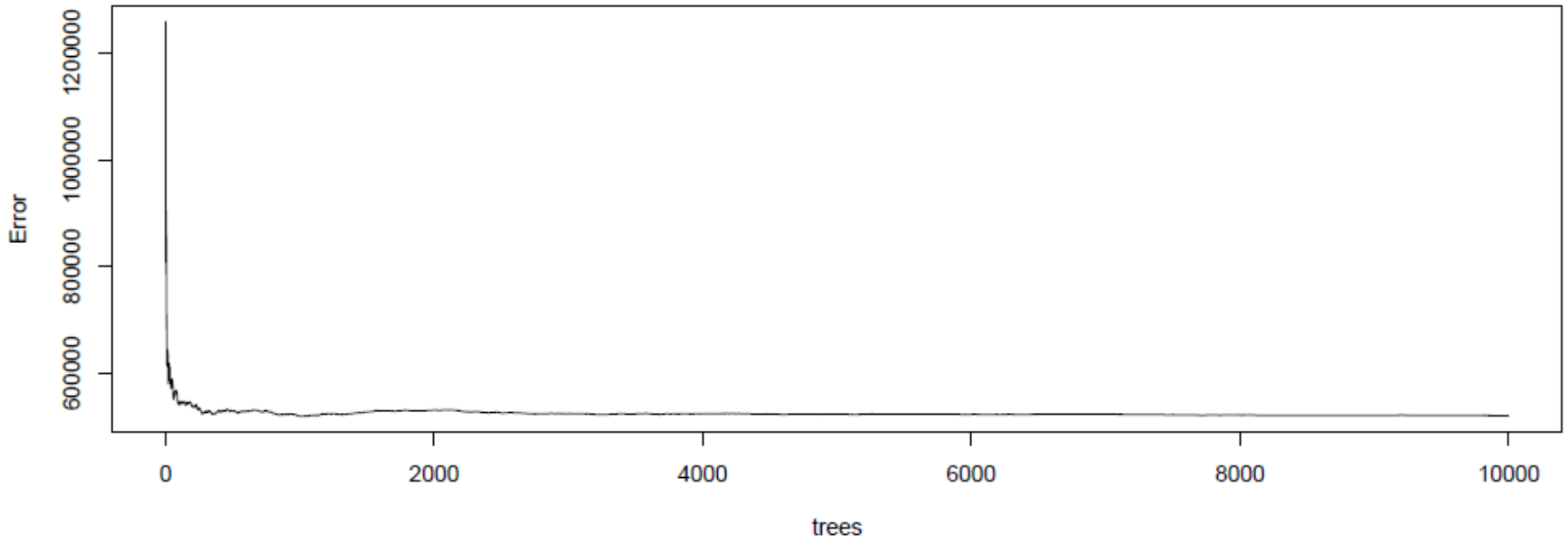
For random forest, use R packages **randomForest** (has partial dependence plots) and **ranger** (much faster, has options to calculate p-values for variable importance).




(a) Random forest




(b) Boosted regression




```
set.seed(300000)
rf2 <- randomForest(y = DATAnoNA[,RESPONSE],
                    x = DATAnoNA[, v])
print(rf2)
plot(rf2)
```

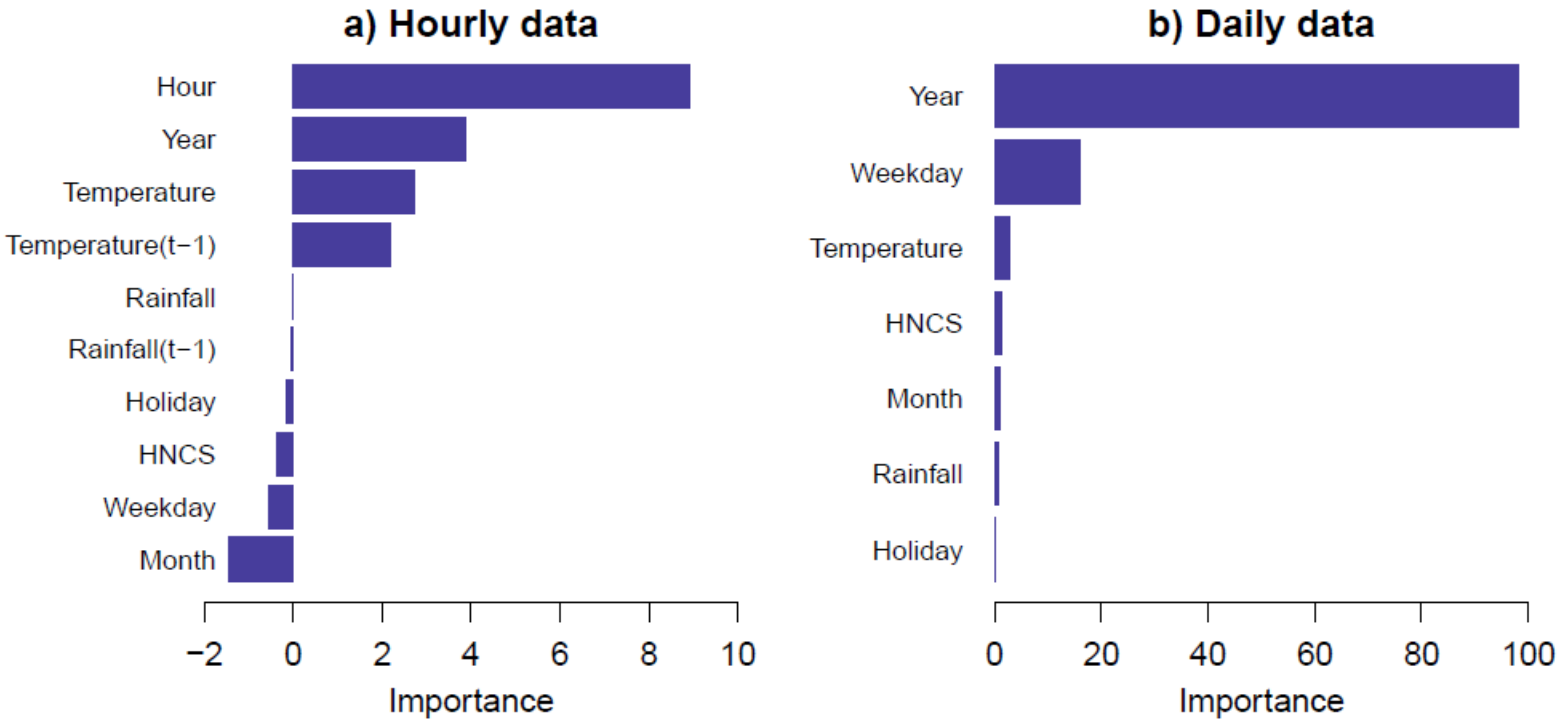
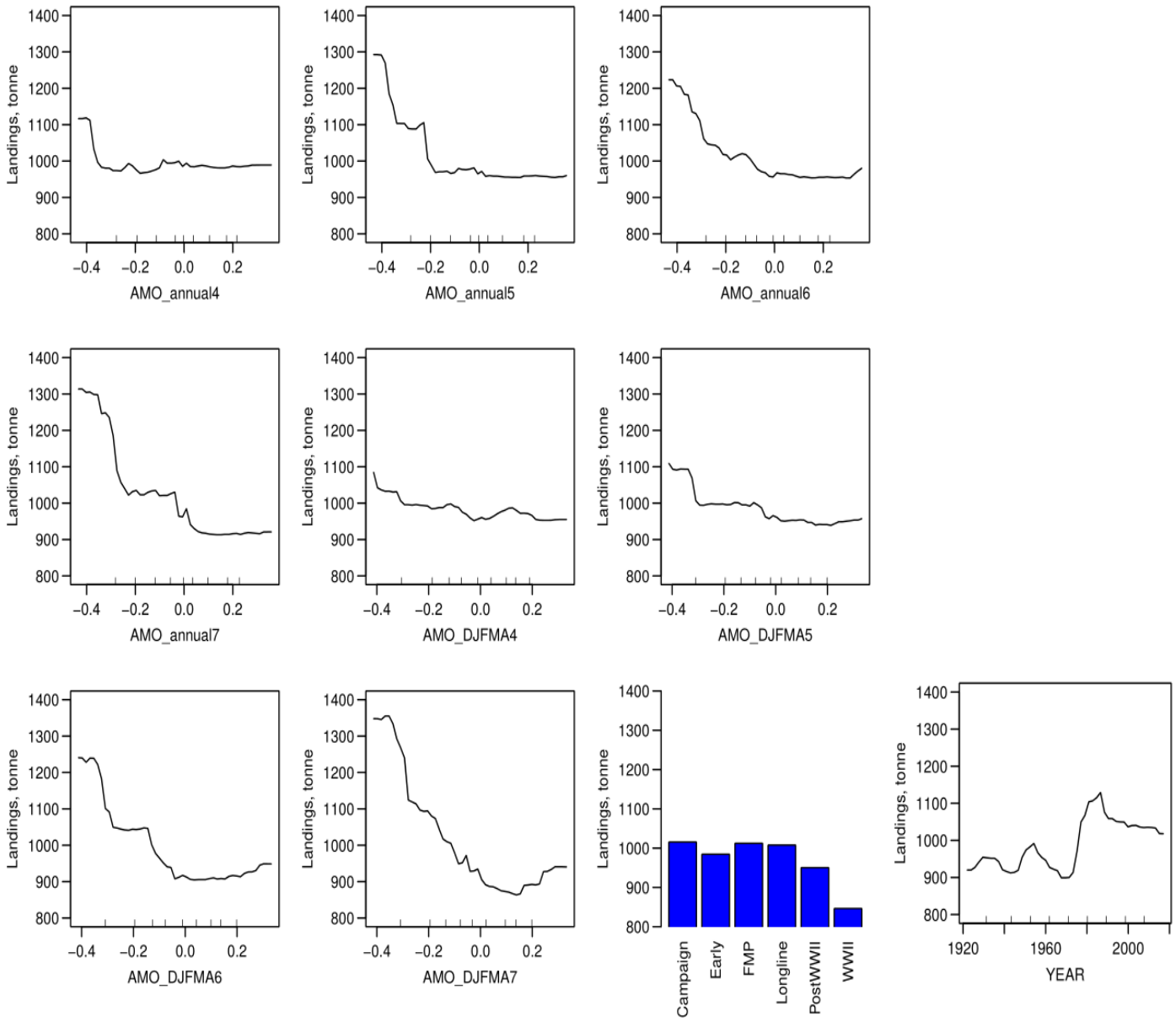



FIGURE 14.6: Relative importance of the variables in random forests.

# Example code: variable importance

```
set.seed(10000)
ran <- ranger(dependent.variable.name = RESPONSE, data = DATAnoNA,
          importance = 'impurity_corrected',
          min.node.size = 5, respect.unordered.factors = 'partition',
          num.trees = 500)
#for predictions
ran2 <- ranger(dependent.variable.name = RESPONSE, data = DATAnoNA,
           # importance = 'impurity_corrected',
           min.node.size = 5, respect.unordered.factors = 'partition',
           num.trees = 500)
print(ran)
ranimp <- importance_pvalues(ran, method = "altmann",
                      num.permutations = 500,
                      formula = as.formula(paste(RESPONSE, ".", sep = " ~ ")),
                      data = DATAnoNA)
ranimp <- ranimp[order(ranimp[,1]),]
tmp <- ranimp[,1]
barplot(tmp,
        beside = TRUE, las = 1, #xlim = c(0, 20),
        xlab = "Importance",
        col = 1, border = NA, cex.names = 0.6,
        horiz = TRUE)
```

# Partial Dependence Plots



# Example code: partial dependence

```
set.seed(300000)
RF <- randomForest(y = DATAnoNA[,RESPONSE],  x = DATAnoNA[, v])
preds <- sort(rownames(rf2$importance))  #get number of predictors
par(mfrow = c(ceiling(length(preds)/3), 3)) #set plots in 3 columns
par(bty = "L", mar = c(5, 4, 1, 1) + 0.1, mgp = c(2, 0.7, 0))
for(i in 1:length(preds)) {
   if(preds[i] == "Time_Block") { #use different settings for categorical predictor(s)
      partialPlot(RF, pred.data = Dtrain, x.var = preds[i],
                  las = 2, xlab = "", ylab = "", main = "", xpd = F
                  #,ylim = c(400, 3100)
      )
   } else {
      partialPlot(RF, pred.data = Dtrain, x.var = preds[i],
                  las = 1, xlab = preds[i], ylab = "", main = "", xpd = F
                  #,ylim = c(0.55, 1.05)
      )
   }
}
```

# library(plotmo) #for interaction plots

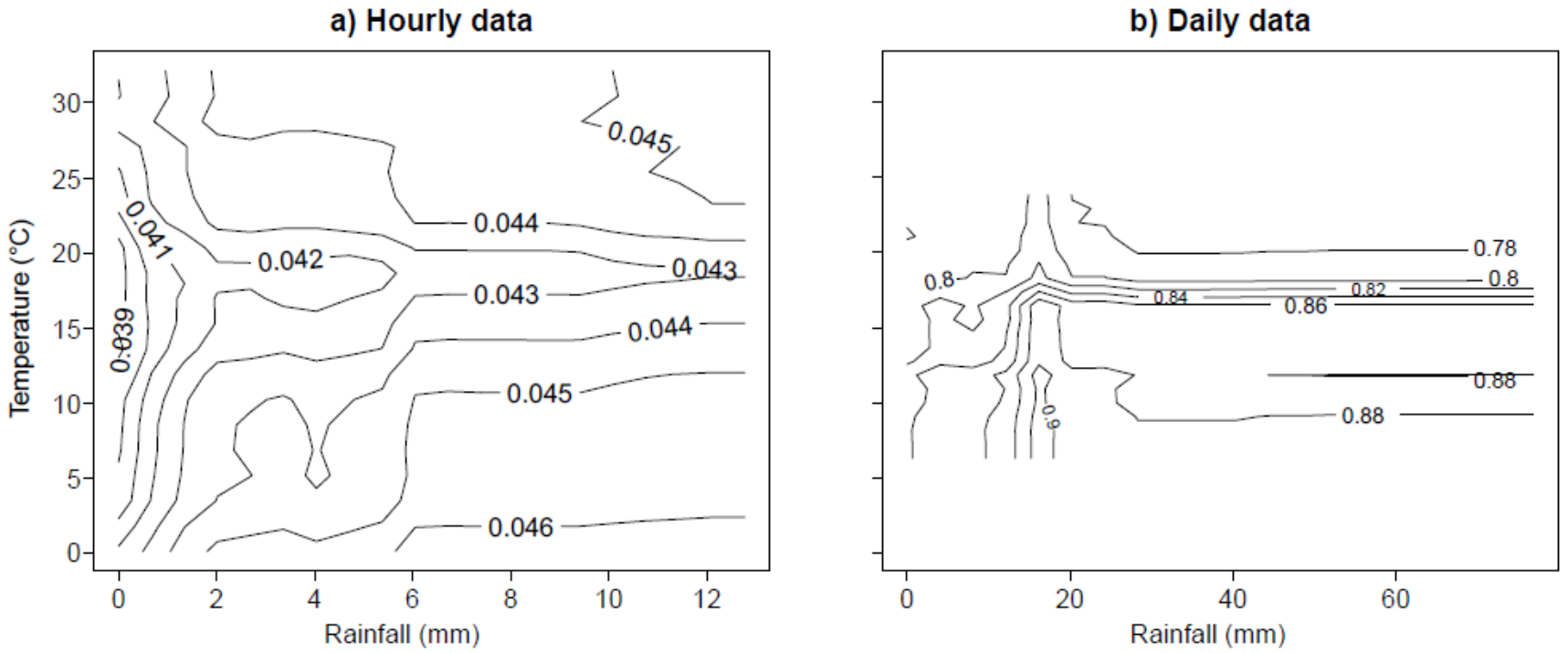



FIGURE 14.7: Partial dependence interaction plots from the random forests. The numbers show average accident rate (non-centered).

# To consider

- Out-of-sample predictions
- Cross-validation (for tuning hyper-parameters)

# References


- Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and regression trees. CRC press; 1984.
- Breiman L. Random forests. Machine learning. 2001;45(1):5-32.
- Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media; 2009.
- Berk RA. Statistical learning from a regression perspective. 2nd ed. New York: Springer; 2016.