Ensemble of Trees: Boosting, Bagging, and Random Forest

Outline

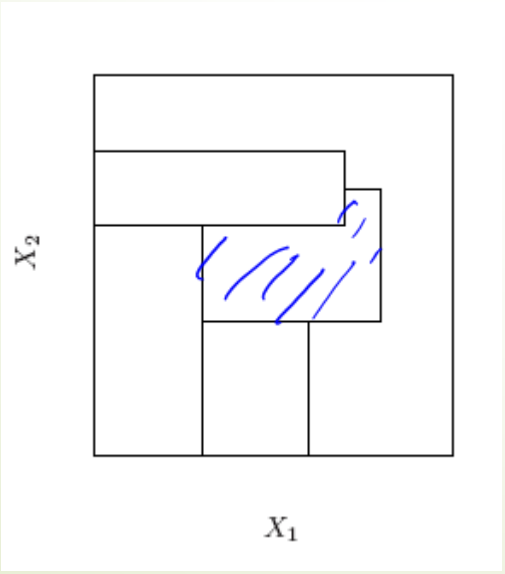
- Tree based methods
  - CART
  - MARS
- Boosting
  - Adaboost for Classification
  - Gradient Boosting Method
- Bagging
- Random Forest

Tree based methods: CART

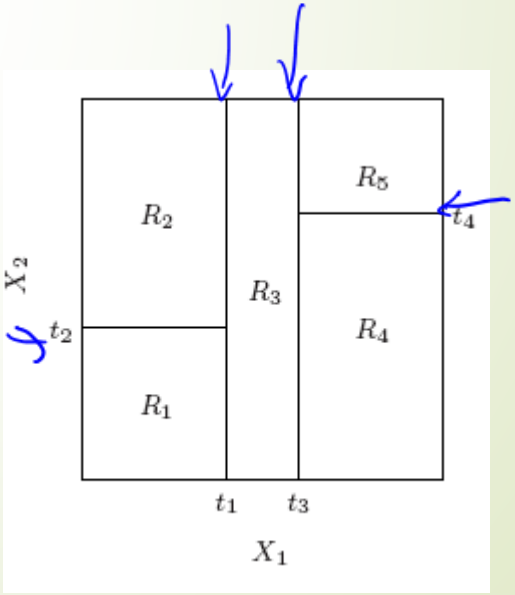
- Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}.$$

Haar wavelet basis

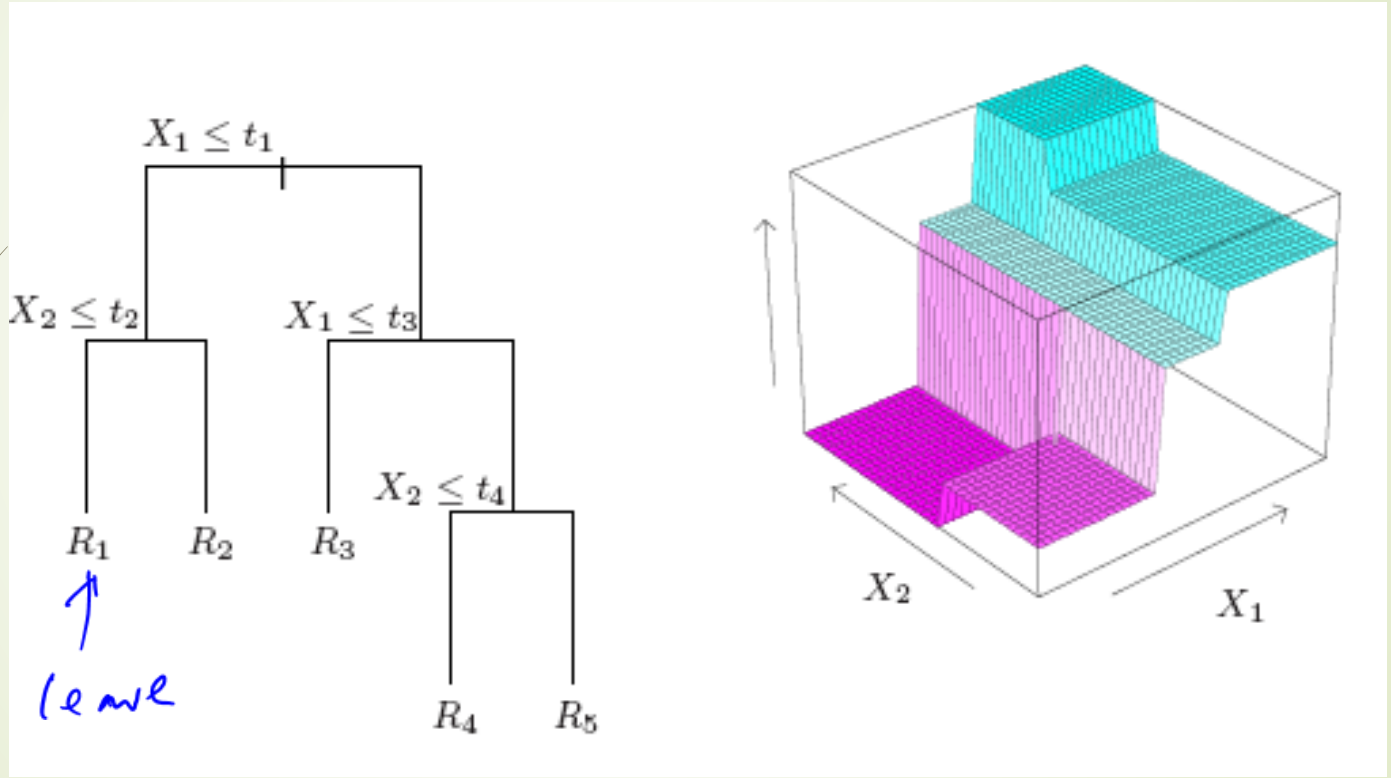


Nonrecursive partition



A recursive partition

Tree based methods



Left: Tree representation for the recursive partition above; Right: piecewise constant function on the tree

Regression Trees

- P inputs and a response

  (x_i,y_i),i=1,2,...,N

$y_i \in \mathbb{R} \quad x_i \in \mathbb{R}^p$

- Goal: automatically decides on the splitting variables and split points, and also the topology (shape) of the tree
- Alg: suppose we know the partition
- R_1,R_2,...,R_M

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

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

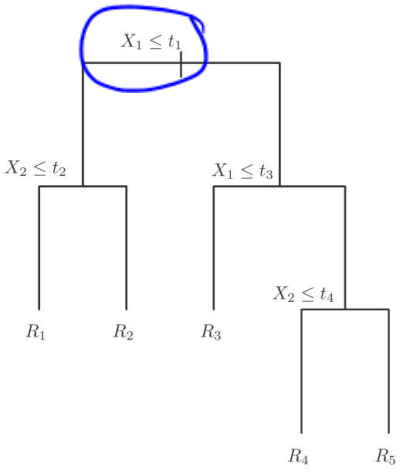
Regression Trees

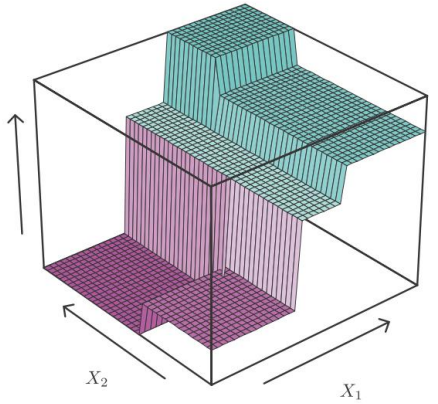
- Recursive binary splitting of variable in greedy way

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

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

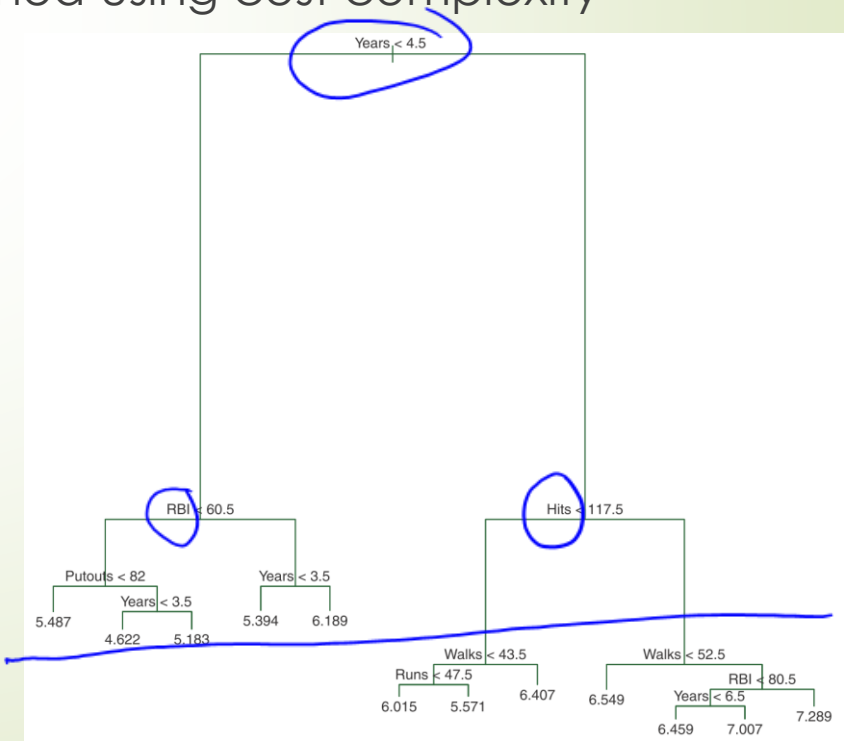
$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j,s)).$$





Tree size

- Large tree might over-fit the data
- Small tree might not capture the important structure
- Tree size is a tuning parameter
- Stop the splitting process only when some minimum node size (say 5) is reached
- Then this large tree is pruned using cost complexity pruning



Optimal Pruning

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error. α^*

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Classification Trees

- The target variable is a classification outcome taking values 1,2,...,K
- The only change needed is the criteria for splitting nodes and pruning the tree.
- Measures of node impurity: $Q_m(T)$

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \qquad k(m) = \arg\max_k \hat{p}_{mk},$$

purity

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

Library("tree")

- Section 8.3, Introduction to Statistical Learning with R
  - Classification tree
  - Regression tree
  - model = tree(y~., data)

↑
formula

lm

glm

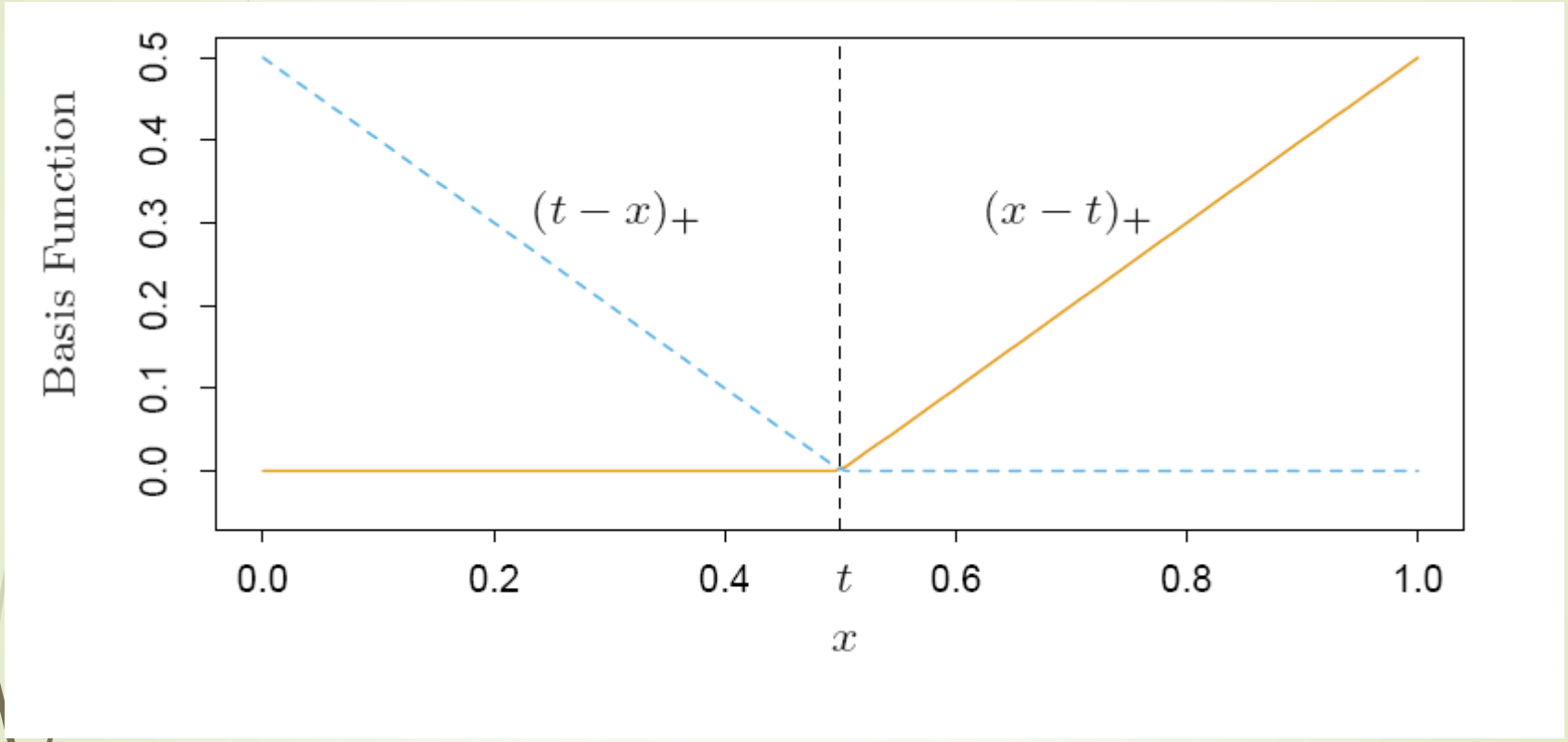
MARS

- Multivariate Adaptive Regression Splines
- Well suited for high-dim problems
- A generalization of stepwise linear regression
- It uses expansions in piecewise linear basis functions of the form:

$$(x-t)_+ = \begin{cases} x-t, & \text{if } x > t, \\ 0, & \text{otherwise,} \end{cases} \quad \text{and} \quad (t-x)_+ = \begin{cases} t-x, & \text{if } x < t, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{C} = \{(X_j - t)_+,\ (t - X_j)_+\} \quad \begin{array}{l} t \in \{x_{1j}, x_{2j}, \ldots, x_{Nj}\} \\ j = 1, 2, \ldots, p. \end{array}$$

MARS



A reflected pair

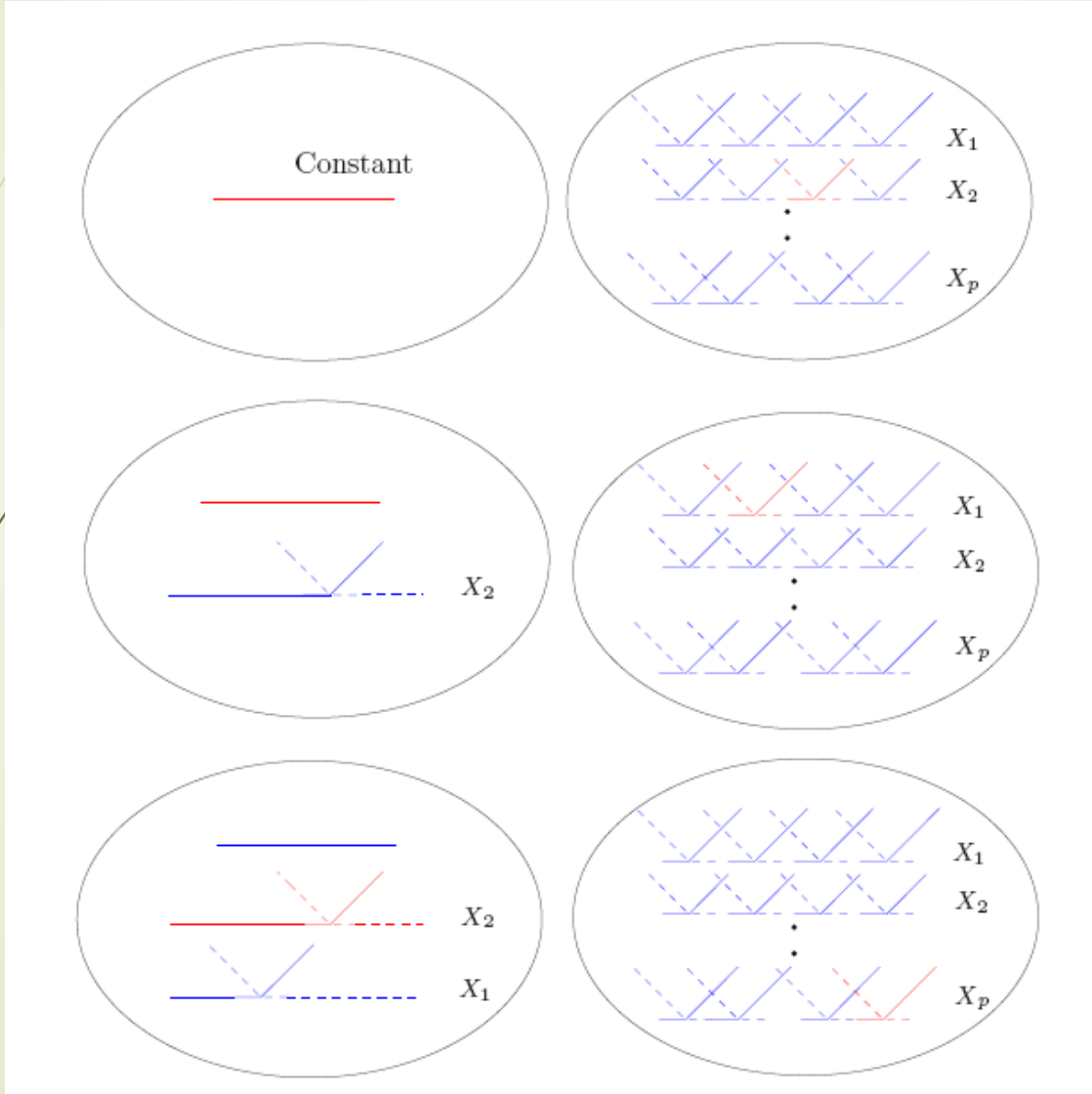
MARS

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X),$$

- Where $h_m(X)$ is a function in $\mathcal{C}$, or a product of two or more such functions.
- Adaptive way to add basis functions:

$$\hat{\beta}_{M+1} h_\ell(X) \cdot (X_j - t)_+ + \hat{\beta}_{M+2} h_\ell(X) \cdot (t - X_j)_+, \; h_\ell \in \mathcal{M},$$

MARS



MARS

- The size of terms matters

$$\mathrm{GCV}(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2}.$$

- M is the effective number of parameters in the model: this accounts both for the number of terms and the number of parameters when selecting the positions of the knots.

Boosting

- Originally designed for classification problems
- Can be extended to regression problems
- Motivation: combines the output of many weak classifiers to produce a powerful "committee" (How to boost a weak classifier which is slightly better than random guess to a strong one – Leslie Valiant problem)
  - What are 'weak' classifiers?
  - How to combine them?

Here we present a statistical point of view.

What's weak classifiers

- In theory, weak classifiers have expected error rate <1/2, but impossible to design with finite samples
- In practice, Rob Schapire points out two elements about 'weak' classifiers:
  - Each classifier is simple, decided locally by small number of samples
  - Classifiers are independent, not highly correlated through using all samples
- So CART (classification-and-regression-trees) are good candidates for weak classifiers

Adaboost

- Consider a two-class problem:

$$X \qquad Y \in \{-1, 1\}$$

- G(X) is the classifier
- Error rate:

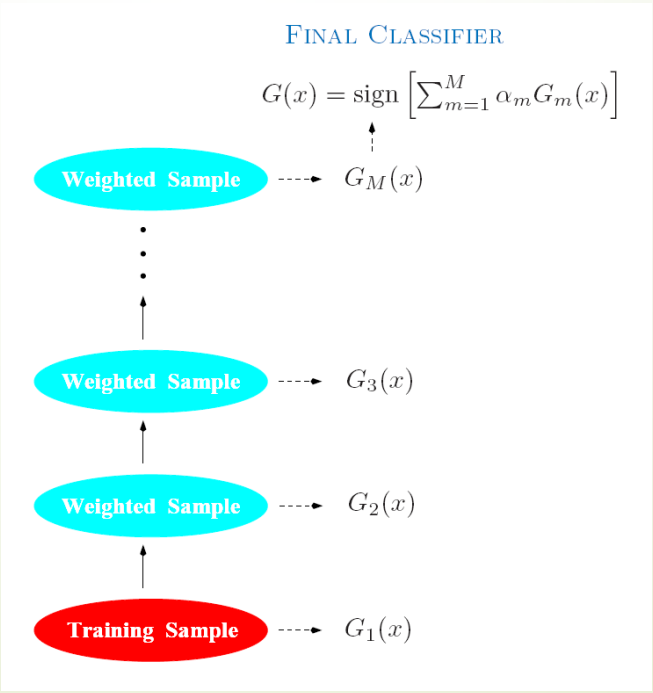
$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

- Weak classifier: error rate is only slightly better than random guessing.

Adaboost

- Boosting sequentially apply the week classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers:

$$G_m(x), m = 1, 2, \ldots, M.$$



Adaboost.M1

Freund and Schapire 1997

Algorithm 10.1 *AdaBoost.M1.*

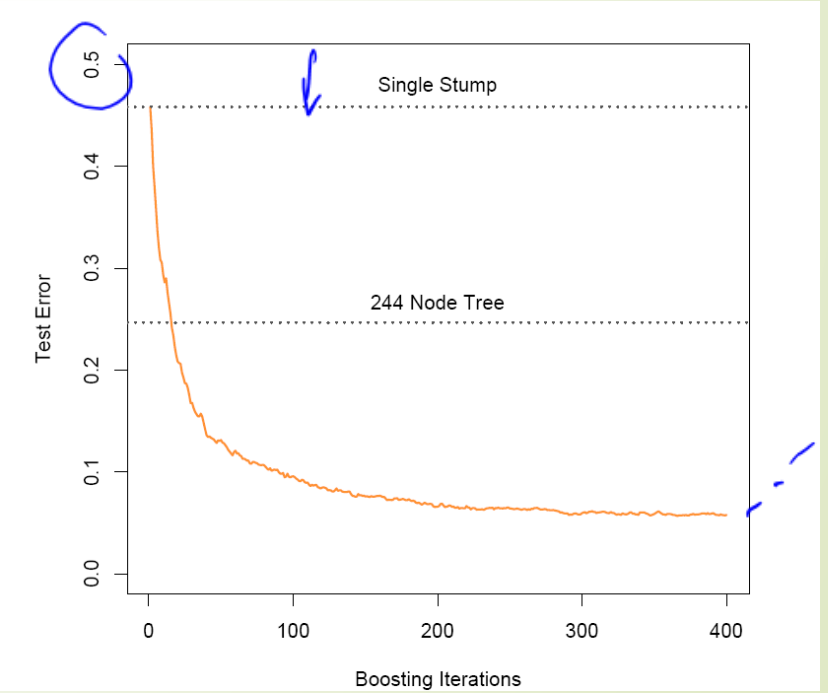
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1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

learning from errors

Example: Decision Stumps

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5), \\ -1 & \text{otherwise.} \end{cases}$$

- 2000 trains and 10,000 tests
- Weak classifier is just a stump: a two terminal node classification tree.
- Error rate: 48.5%



Boosting Fits and Additive Model

- Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions.
- For Adaboost, basis functions are weak classifiers

$$G_m(x) \in \{-1, 1\}$$

- More generally,

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m),$$

Additive Model:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right).$$

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.

2. For $m = 1$ to M:

   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Exponential Loss and AdaBoost

- Adaboost is equivalent to forward additive modeling using the followina loss function:

$$L(y, f(x)) = \exp(-y\, f(x)).$$

- Forward step:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i (f_{m-1}(x_i) + \beta\, G(x_i))]$$

$$= \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

$$w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$$

- Joint optimization

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i)) \qquad (10.9)$$

$$w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$$

- Step I: Optimizer over G:

claim

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

Step II: Optimizer over β

$$\sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

$$= e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

$$= (e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \cdot \sum_{i=1}^{N} w_i^{(m)}$$

solving for β one obtains

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m},$$

where err_m is the minimized weighted error rate

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}.$$

Finally adaBoost

The approximation is then updated

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x),$$

which causes the weights for the next iteration to be

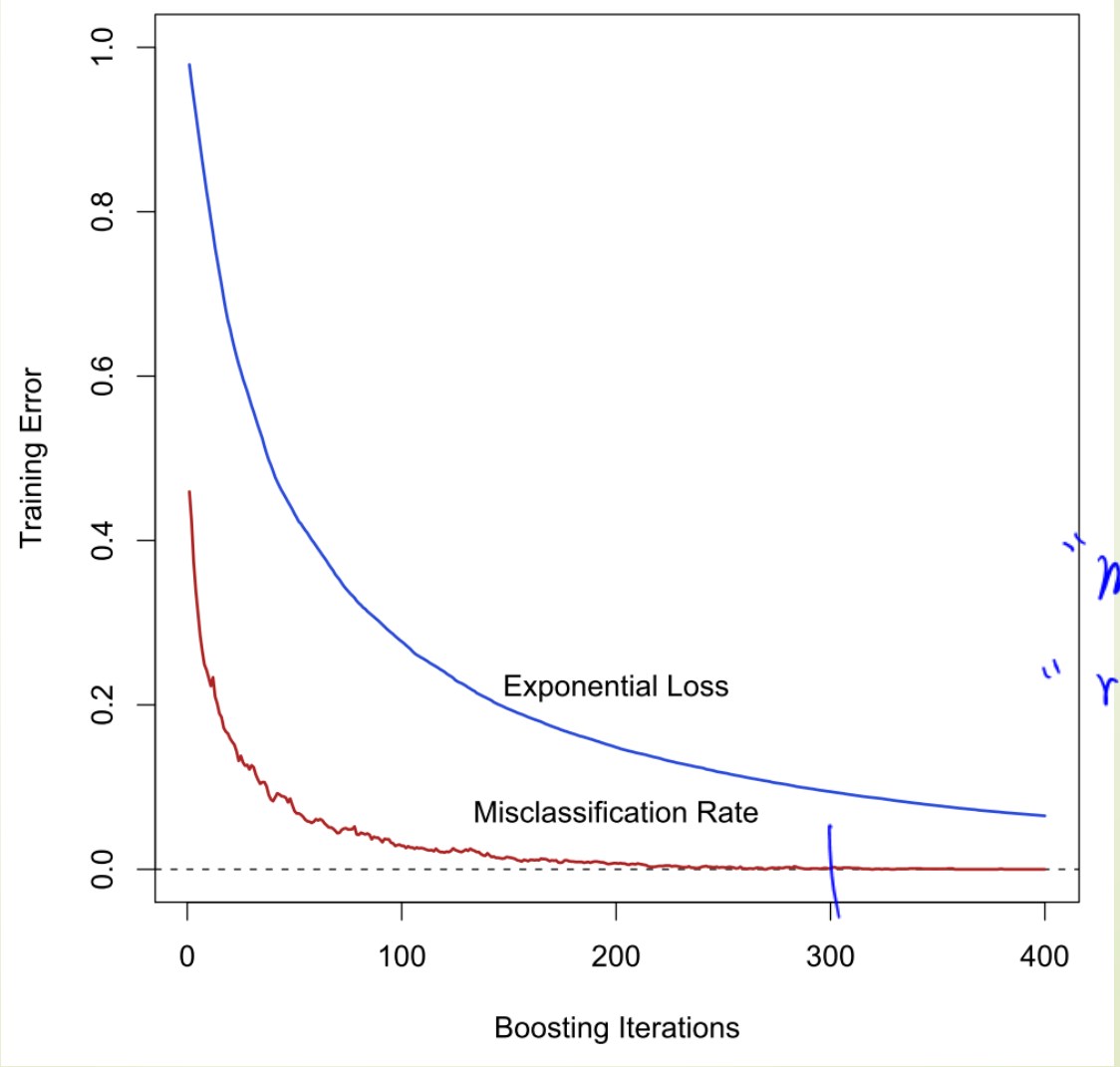
$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)}. \tag{10.14}$$

Using the fact that $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, (10.14) becomes

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m}, \tag{10.15}$$

where $\alpha_m = 2\beta_m$ is the quantity defined at line 2c of AdaBoost.M1 (Algorithm 10.1). The factor $e^{-\beta_m}$ in (10.15) multiplies all weights by the same value, so it has no effect. Thus (10.15) is equivalent to line 2(d) of Algorithm 10.1.

Exponential Loss reduction goes further than misclassification error



Why Exp Loss? Boosting the Margin

- Computational reason
- Leads the simple reweighting scheme
- Question: what does adaBoost estimate?
  - Look at population minimizer…

AdaBoost vs. LogitBoost

Friedman

- In population, AdaBoost

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)},$$

$$\Pr(Y=1|x) = \frac{1}{1+e^{-2f^*(x)}}$$

- LogitBoost: It is the same maximal binomial log-likelihood (deviance) in population

$$p(x) = \Pr(Y=1\,|\,x) = \frac{e^{f(x)}}{e^{-f(x)}+e^{f(x)}} = \frac{1}{1+e^{-2f(x)}}$$

$$-l(Y,f(x)) = \log\left(1+e^{-2Yf(x)}\right).$$

Loss functions and Robustness: classification

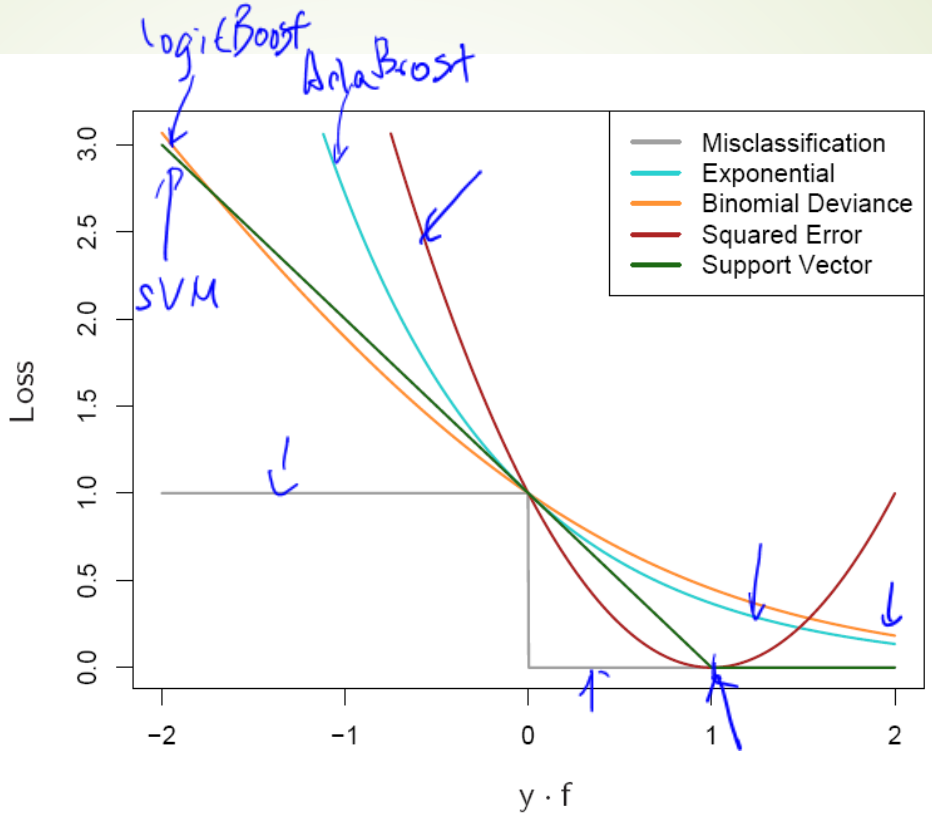
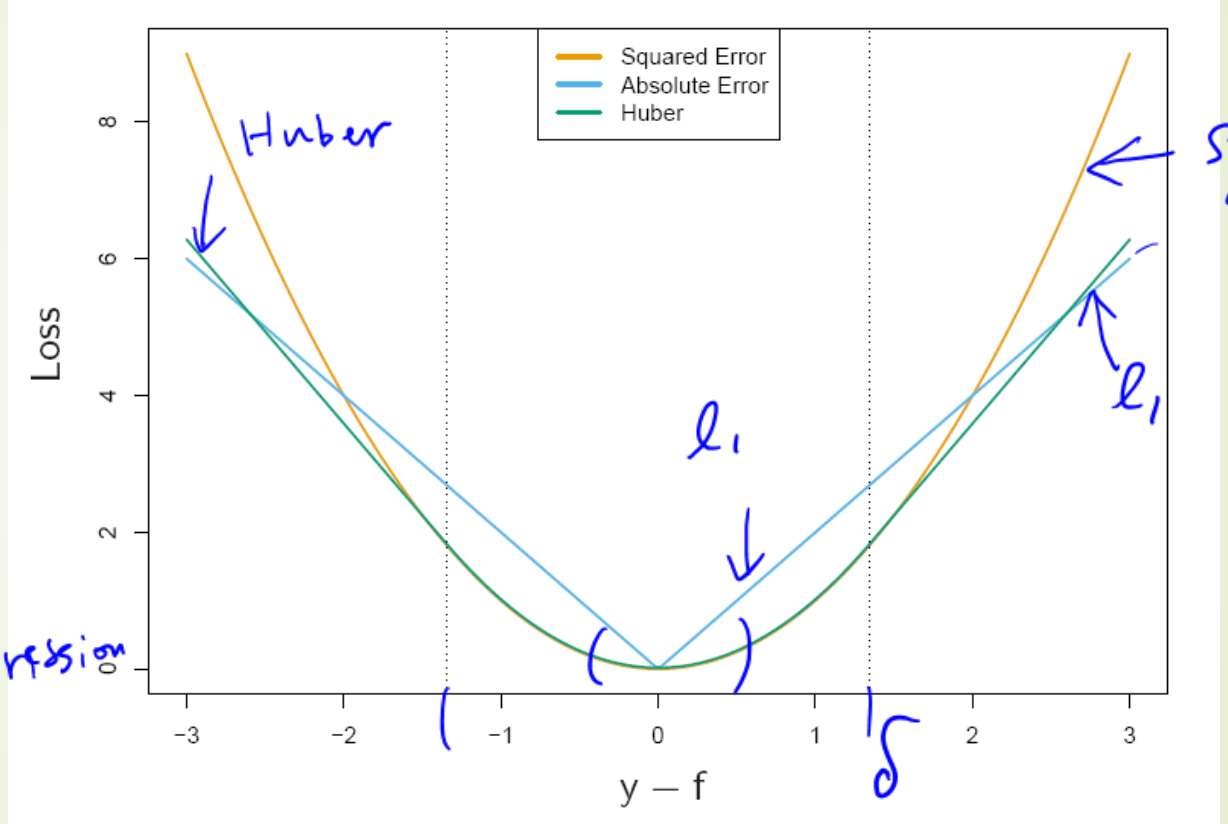


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Loss functions and Robustness: Regression



robust Regression

Huber Loss

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for } |y - f(x)| \le \delta, \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise.} \end{cases}$$

Huber's M-Estimator in Regression

$$\beta_0 = \arg\min E_0 \rho(Y - \mathbf{Z}^T \beta)$$

ρ convex, symmetric about 0. p fixed

$\hat{\beta}_\rho \equiv \arg\min \frac{1}{n} \sum_{i=1}^n \rho(Y_i - \mathbf{Z}_i^T \beta)$

Thm (Huber) If $\psi \equiv \rho'$ is smooth, p is fixed, $n \to \infty$, $E_0\psi^2(e) < \infty$, $E_0\psi'(e) \neq 0$ and $\mathbf{Z}$ is full dimensional, $E\mathbf{Z}\mathbf{Z}^T$ non singular

$$\hat{\beta}_\rho = \beta_0 - \frac{1}{n} \sum_{i=1}^n \mathbf{Z}_i \frac{\psi(e)}{E_0\psi'(e)} + o_P(n^{-\frac{1}{2}})$$

$$\sqrt{n}(\hat{\beta} - \beta_0) \Longrightarrow N_p(\mathbf{0}, \left[E_0(\mathbf{Z}\mathbf{Z}^T)\right]^{-1} \sigma^2(\rho))$$

$$\sigma^2(\rho) = \frac{E_0\psi^2(e)}{[E_0\psi'(e)]^2}$$

E.g.: $\psi(t) = t$ LSE not robust against heavy tails

$$\psi(t) = h_k(t) = t, \ |t| \leq k \text{ (Huber)}$$
$$= k \ \mathrm{sgn}(t), |t| > k$$
$$\psi(t) = \mathrm{sgn}(t) \qquad (L1)$$

Other Interpretations on Boosting

- Leo Breiman, Freund and Schapire, Game Theoretic View
- Tong Zhang, Bin Yu, Early Stopping Regularization in convex optimization
- Peter Buhlman, Bin Yu; Yuan Yao, Andrea Caponnetto and Lorenzo Rosasco, Early Stopping in RKHS
- Peter Bartlett, Schapire et al.; Liwei Wang, Margin distribution boosting

Gradient Boosted Models Library("gbm")

- Section 8.3.4, ISLR.
- model = gbm(y~x, data, distribution,n.trees,interaction.depth)
  - distribution = ["gaussian"] | "Bernoulli"
  - n.trees = 1000, number of trees
  - interaction.depth = 2, the depth limit of each tree

Bagging (Leo Breiman 1996)

- Bootstrap can be used to assess the accuracy of a prediction or parameter estimate
- Bootstrap can also be used to improve the estimate or prediction itself.

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

- Reduce variances of the prediction

Bagging in Reduction of Variance

$$\mathrm{E}_{\hat{\mathcal{P}}}\hat{f}^*(x), \text{ where } \mathbf{Z}^* = (x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_N^*, y_N^*)$$

$$(x_i^*, y_i^*) \sim \hat{\mathcal{P}}.$$

- If $\hat{f}(x)$ is linear in data, then bagging is just itself.
- Take cubic smooth spline as an example.
- Property:

$$\begin{aligned}
\mathrm{E}_{\mathcal{P}}[Y - \hat{f}^*(x)]^2 &= \mathrm{E}_{\mathcal{P}}[Y - f_{\mathrm{ag}}(x) + f_{\mathrm{ag}}(x) - \hat{f}^*(x)]^2 \\
&= \mathrm{E}_{\mathcal{P}}[Y - f_{\mathrm{ag}}(x)]^2 + \mathrm{E}_{\mathcal{P}}[\hat{f}^*(x) - f_{\mathrm{ag}}(x)]^2 \\
&\geq \mathrm{E}_{\mathcal{P}}[Y - f_{\mathrm{ag}}(x)]^2.
\end{aligned}$$

x fixed

$$f_{\mathrm{ag}}(x) = \mathrm{E}_{\mathcal{P}}\hat{f}^*(x).$$

Bagging Classifiers

- Classifier

$$\hat{f}(x) = e_j(x), \qquad 1 \le j \le k$$

- Bagging the classifier:

$$Z^{*b} = (x_i^{*b}, y_i^{*b})_{i=1}^{N} \Rightarrow \hat{f}^{*b}, \qquad b = 1, \ldots, B$$

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b \in [1,B]} \hat{f}^{*b}(x) = (p_1, \cdots, p_k)$$

$$\hat{G}(x) = \max_{1 \le j \le k} \hat{f}_{bag}(x) \Big|_j$$

Bagging Classifiers: independent weak learners

good classifier can make it better, but bagging a bad classifier can make it worse. Here is a simple example, using a randomized rule. Suppose $Y = 1$ for all x, and the classifier $\hat{G}(x)$ predicts $Y = 1$ (for all x) with probability 0.4 and predicts $Y = 0$ (for all x) with probability 0.6. Then the misclassification error of $\hat{G}(x)$ is 0.6 but that of the bagged classifier is 1.0.

For classification we can understand the bagging effect in terms of a consensus of independent *weak learners* (Dietterich, 2000a). Let the Bayes optimal decision at x be $G(x) = 1$ in a two-class example. Suppose each of the weak learners G_b^* have an error-rate $e_b = e < 0.5$, and let $S_1(x) = \sum_{b=1}^{B} I(G_b^*(x) = 1)$ be the consensus vote for class 1. Since the weak learners are assumed to be independent, $S_1(x) \sim \text{Bin}(B, 1 - e)$, and $\Pr(S_1 > B/2) \to 1$ as B gets large. This concept has been popularized outside of

Wisdom of Crowds

Bumping

As in bagging, we draw bootstrap samples and fit a model to each. But rather than average the predictions, we choose the model estimated from a bootstrap sample that best fits the training data. In detail, we draw bootstrap samples $\mathbf{Z}^{*1}, \ldots, \mathbf{Z}^{*B}$ and fit our model to each, giving predictions $\hat{f}^{*b}(x)$, $b = 1, 2, \ldots, B$ at input point x. We then choose the model that produces the smallest prediction error, averaged over the *original training set*. For squared error, for example, we choose the model obtained from bootstrap sample $\hat{b}$, where

$$\hat{b} = \arg\min_{b} \sum_{i=1}^{N} [y_i - \hat{f}^{*b}(x_i)]^2. \qquad (8.60)$$

Example: Bumping

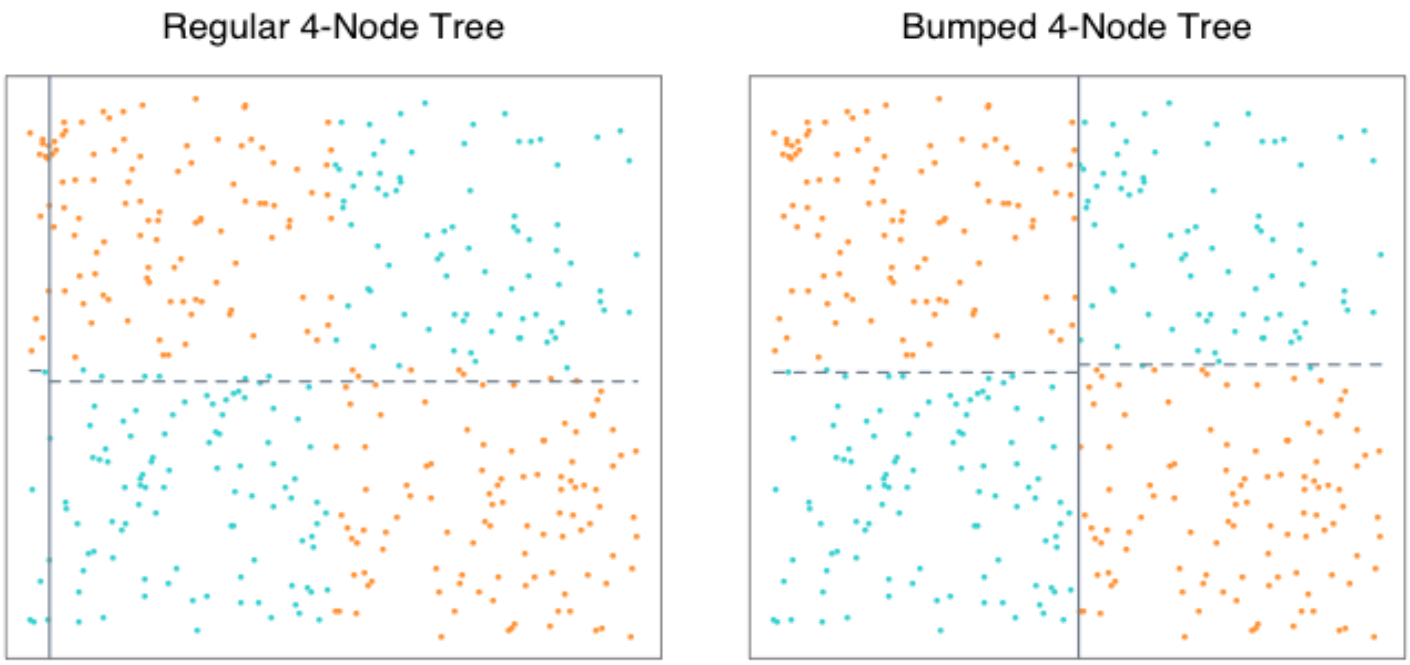


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm* 20 *times.*

Random Forest

- Suppose there is a very strong predictor, along with a small set of moderately strong ones: most of bagged trees will split this strong predictor on the top, since they look at all the predictors in every bootstrap – correlation of bagged trees
- Random forests provide an improvement over bagging byway of decorrelating the trees

Before each split, select $m \leq p$ *of the input variables at random as candidates for splitting.*

Random Forest (Leo Breiman 2001)

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.

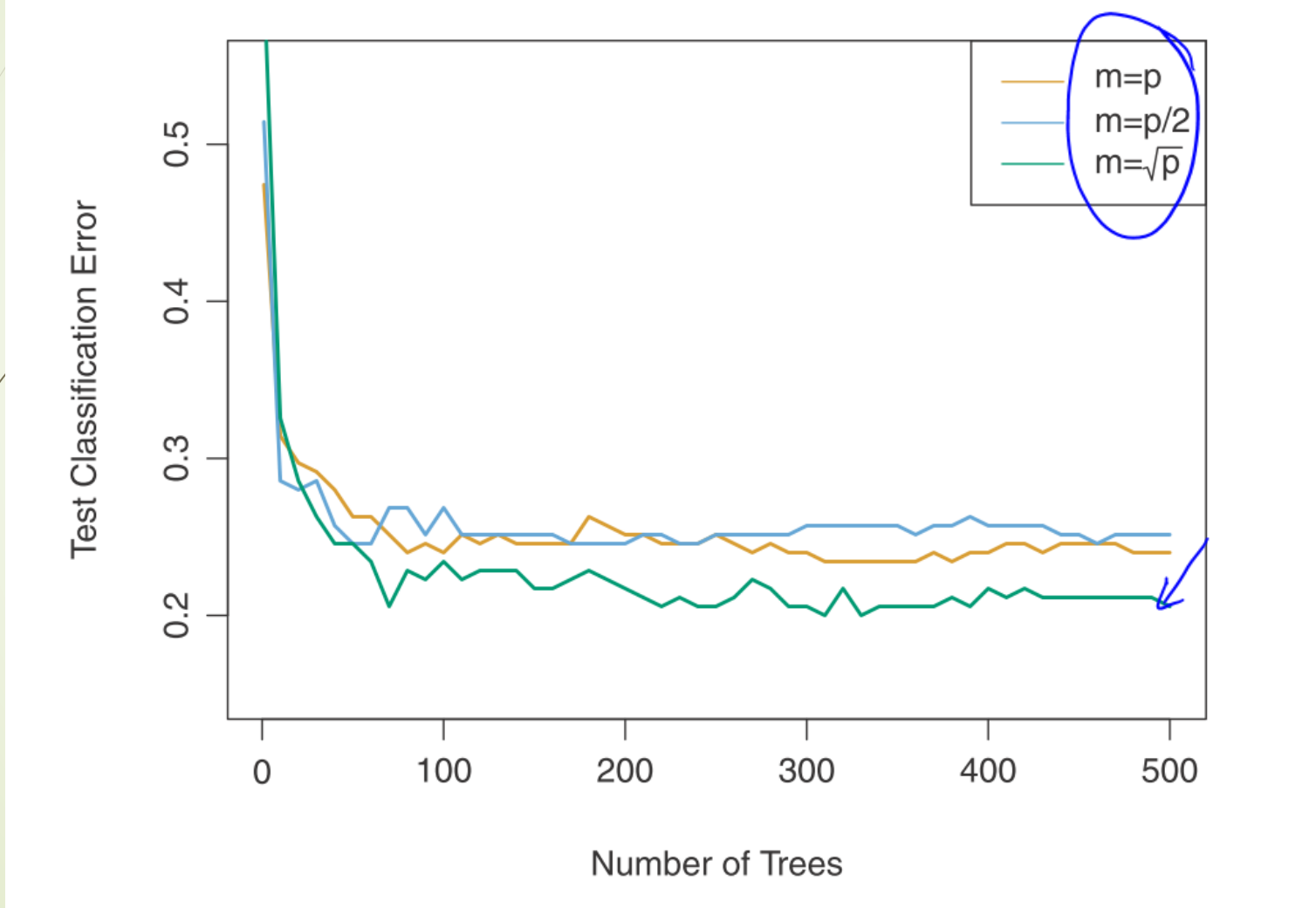
      iii. Split the node into two daughter nodes.

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

To make a prediction at a new point x:

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

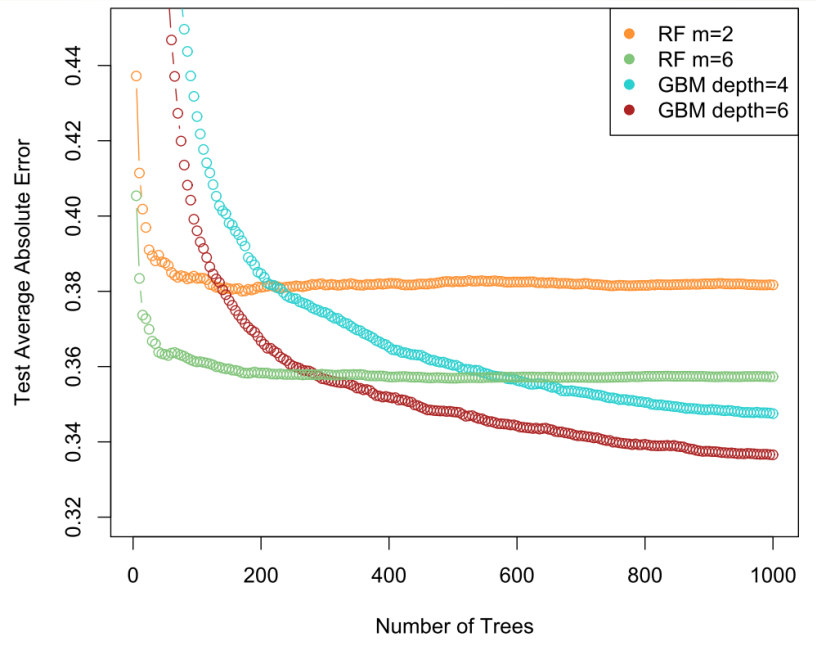
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.


Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees
m=p
m=p/2
m=√p

How to choose m?

- For classification, the default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one.
- For regression, the default value for m is $\lfloor p/3 \rfloor$ and the minimum node size is five.

In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters. In Figure 15.3 $m = 6$ performs much better than the default value $\lfloor 8/3 \rfloor = 2$.



Error Estimate: Out-Of-Bag

For each observation $z_i = (x_i, y_i)$, *construct its random forest predictor by averaging* only *those trees corresponding to bootstrap samples in which* z_i did not *appear.*

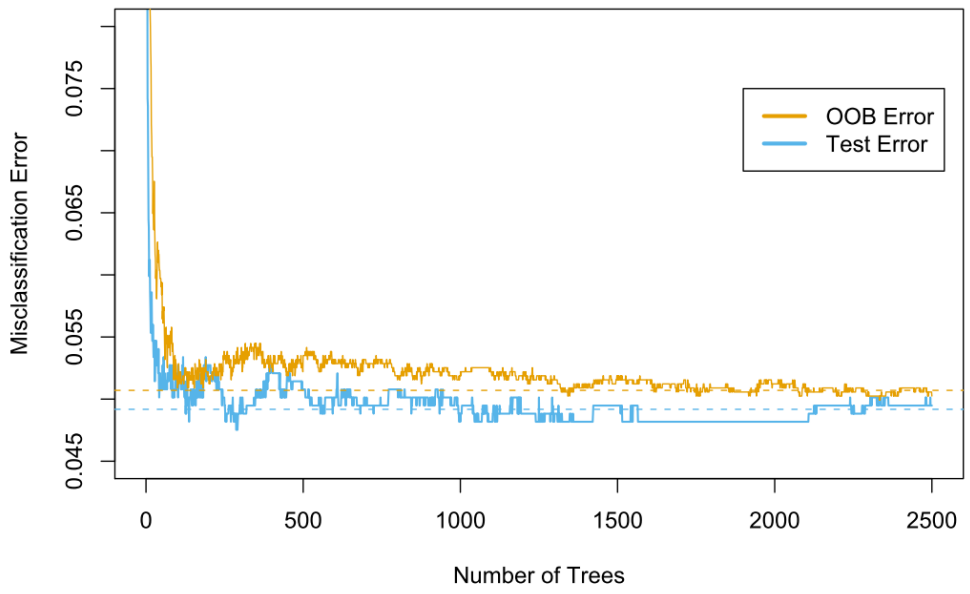


FIGURE 15.4. OOB *error computed on the* `spam` *training data, compared to the test error computed on the test set.*

Variable Importance

- Gradient-Boosted Methods: at each split in each tree, the *improvement* in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable
- OOB prediction power:
  - for the b-th tree, use OOB samples to estimate the prediction accuracy;
  - then the values for the j-th variable are *randomly permuted* in the OOB samples and recompute the prediction accuracy;
  - The accuracy decrease of such a permutation is averaged over all trees, as a measure of the importance of variable j in the random forest

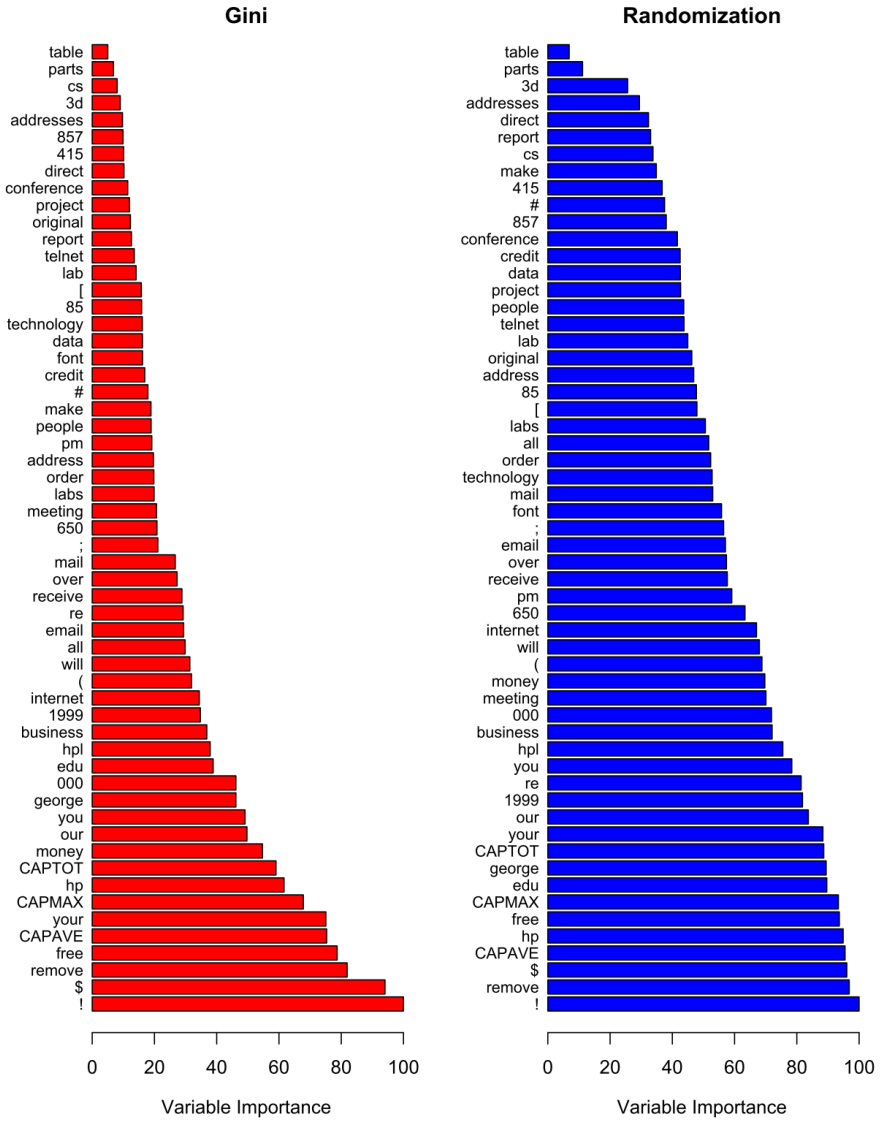


FIGURE 15.5. *Variable importance plots for a classification random forest grown on the* `spam` *data. The left plot bases the importance on the Gini splitting index, as in gradient boosting. The rankings compare well with the rankings produced by gradient boosting (Figure 10.6 on page 354). The right plot uses* OOB *randomization to compute variable importances, and tends to spread the importances more uniformly.*

Library("randomForest")

- http://www.math.usu.edu/~adele/forests
- model = randomForest(formula, data, mtry, ntree, importance, subset)
  - formula = y~.
  - mtry = [p/3] | [\sqrt{p}] | <=p, m is the number of variables to explore in random forest, if m=p it gives Bagging
    - p/3 for regression
    - \sqrt{p} for classification
  - importance=True | False
- importance(model): show the variable importance

m=p