Decision Trees, and Related Methods

gam Generalized Additive Models (advanced)

CART! Classification and regression trees

! cost sensitive pruning (α's)

PRIM Patient Rule Induction Method (a note)

MARS Multivariate Additive Regression Splines (advanced)

Additive Models, Trees, and Related Methods

- **Generalized additive models (GAMs)** are automatic flexible statistical methods that may be used to identify and characterize nonlinear regression effects.
- GAM has the form

$$\mathbb{E}(Y|X_1,\ldots,X_p) = \alpha + f_1(X_1) + \ldots + f_p(X_p)$$

  - where f_j's are unspecified smooth functions
  - X_j predictors, Y the outcome.
- We use a cubic smoothing spline, local polynomial regression or a kernel smoother
- **we simultaneously estimate all p functions**.

GAM for non Gaussian distributions

- Denote $\mu(X) = P(Y = 1|X)$ in a two class classification with 0-1 encoding and recall the logistic regression

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + \beta_1 X_1 + \ldots + \beta_p X_p,$$

- **Additive logistic regression** model replaces the linear terms by the smoothers

$$\log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + f_1(X_1) + \ldots + f_p(X_p),$$

- The conditional mean $\mu(X)$ of a response Y is related to an additive function of the predictors via a **link function** g

$$g[\mu(X)] = \alpha + f_1(X_1) + \ldots + f_p(X_p).$$

- Examples of classical link functions are the following
  - $g(\mu) = \mu$ the identity link, used for linear and additive models for Gaussian response data.
  - $g(\mu) = \mathit{logit}(\mu)$ as above
  - $g(\mu) = \mathit{probit}(\mu)$ **probit** link function for modeling binomial probabilities is the inverse of Gaussian cumulative distribution function $\mathit{probit}(\mu) = \Phi^{-1}(\mu)$
  - $g(\mu) = \log(\mu)$ for log-linear or **log-additive models** for Poisson count data.

Models with Feature Interactions

- The categorical variables are usually treated like identifiers (0-1 or -1,1)
- in the logistic regression, it leads to a 'constant' β_j fitted for the variable
- the slope β_{-j} of others does not depend on the identifier
- We can extend to a **semiparametric model**, that keeps the effect of the kth predictor and the effect of the predictor Z additive

  $$g(\mu) = X^T\beta + \alpha_k + f(Z).$$

- To allow different slopes/shapes of Z based on qualitative variable V we need an interaction term of two features

  $$g(\mu) = f(X) + g_k(Z)$$

- Generally, we may add a function $g_{ZW}(Z, W)$ of two or more features

  $$g(\mu) = f(X) + g_{ZW}(Z, W).$$

- Note: logit, probit, log, gamma and negative-binomial distributions belong the an exponential family, therefore have some nice properties (fit together).

Fitting Additive Model

The backfitting algorithm for additive models

1: **procedure** GENERALIZED ADDITIVE MODEL FITTING:($\mathbf{X}$, $\mathbf{y}$)
2: $\hat{\alpha} = \frac{1}{N}\sum_1^N y_i$, $\hat{f}_j \equiv 0$ initialize $\forall i, j$.
3: **repeat** for $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots$
4: $\hat{f}_j \leftarrow \mathcal{S}_j\left[\{y_i - \hat{\alpha} - \sum_{j\neq k}\hat{f}_k(x_{ij})\}_1^N\right]$,
5: $\hat{f}_j \leftarrow \hat{f}_j - \sum_{i=1}^N \hat{f}_j(x_{ij})$.
6: **until** the functions $\hat{f}_j$ change less than a prespecified threshold.
7: **end procedure**

- $\mathcal{S}_j$ denotes the smoother, for example the smoothing spline with predefined degrees of freedom.
- All $\hat{f}_j$ should have zero mean, the constant is fitted by α.
- Re-normalization is recommended because of rounding errors.

Generalized Additive Logistic Regression×

1: **procedure** ADDITIVE LOGISTIC REGRESSION:(**X**, **y** in 0-1 encoding)
2: $\hat{y} = \frac{1}{N}\sum_1^N y_i, \hat{\alpha} = \log(\frac{\hat{y}}{1-\hat{y}})$, $\hat{f}_j \equiv 0$ initialize $\forall j$.
3: $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = \frac{1}{1+\exp(-\hat{\eta}_i)}$
4: **repeat**
5: Construct the working target variable

$$z_i = \hat{\eta}_i + \frac{y_i - \hat{p}_i}{\hat{p}_i(1-\hat{p}_i)}.$$

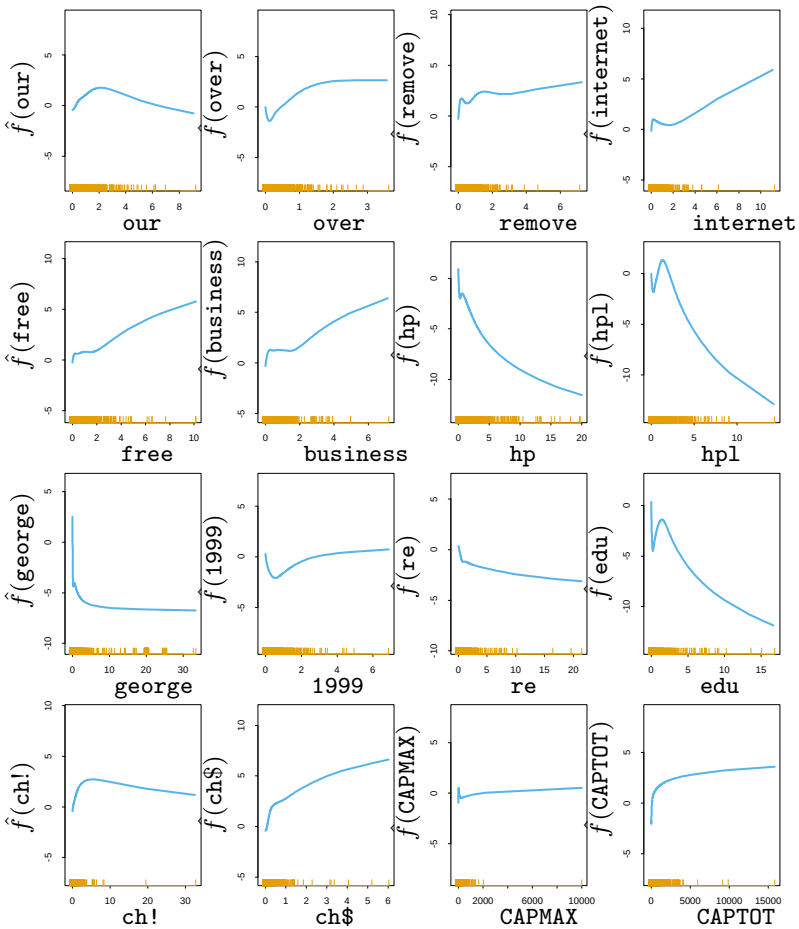
6: Construct the weight $w_i = \hat{p}_i(1-\hat{p}_i)$.
7: Fit an additive model to the targets z_i with weights w_i, using a weighted backfitting algorithm. This gives new estimates $\hat{\eta}_j, \hat{f}_j \ \forall j$
8: **until** the functions change less than a prespecified threshold.
9: **end procedure**

Spam Example

- Email classification as email/spam.
- word frequency as X features.
- Important features:

TABLE 9.2. *Significant predictors from the additive model fit to the spam training data. The coefficients represent the linear part of $\hat{f}_j$, along with their standard errors and Z-score. The nonlinear P-value is for a test of nonlinearity of $\hat{f}_j$.*

Name	Num.	df	Coefficient	Std. Error	Z Score	Nonlinear P-value
Positive effects						
our	5	3.9	0.566	0.114	4.970	0.052
over	6	3.9	0.244	0.195	1.249	0.004
remove	7	4.0	0.949	0.183	5.201	0.093
internet	8	4.0	0.524	0.176	2.974	0.028
free	16	3.9	0.507	0.127	4.010	0.065
business	17	3.8	0.779	0.186	4.179	0.194
hpl	26	3.8	0.045	0.250	0.181	0.002
ch!	52	4.0	0.674	0.128	5.283	0.164
ch$	53	3.9	1.419	0.280	5.062	0.354
CAPMAX	56	3.8	0.247	0.228	1.080	0.000
CAPTOT	57	4.0	0.755	0.165	4.566	0.063
Negative effects						
hp	25	3.9	−1.404	0.224	−6.262	0.140
george	27	3.7	−5.003	0.744	−6.722	0.045
1999	37	3.8	−0.672	0.191	−3.512	0.011
re	45	3.9	−0.620	0.133	−4.649	0.597
edu	46	4.0	−1.183	0.209	−5.647	0.000

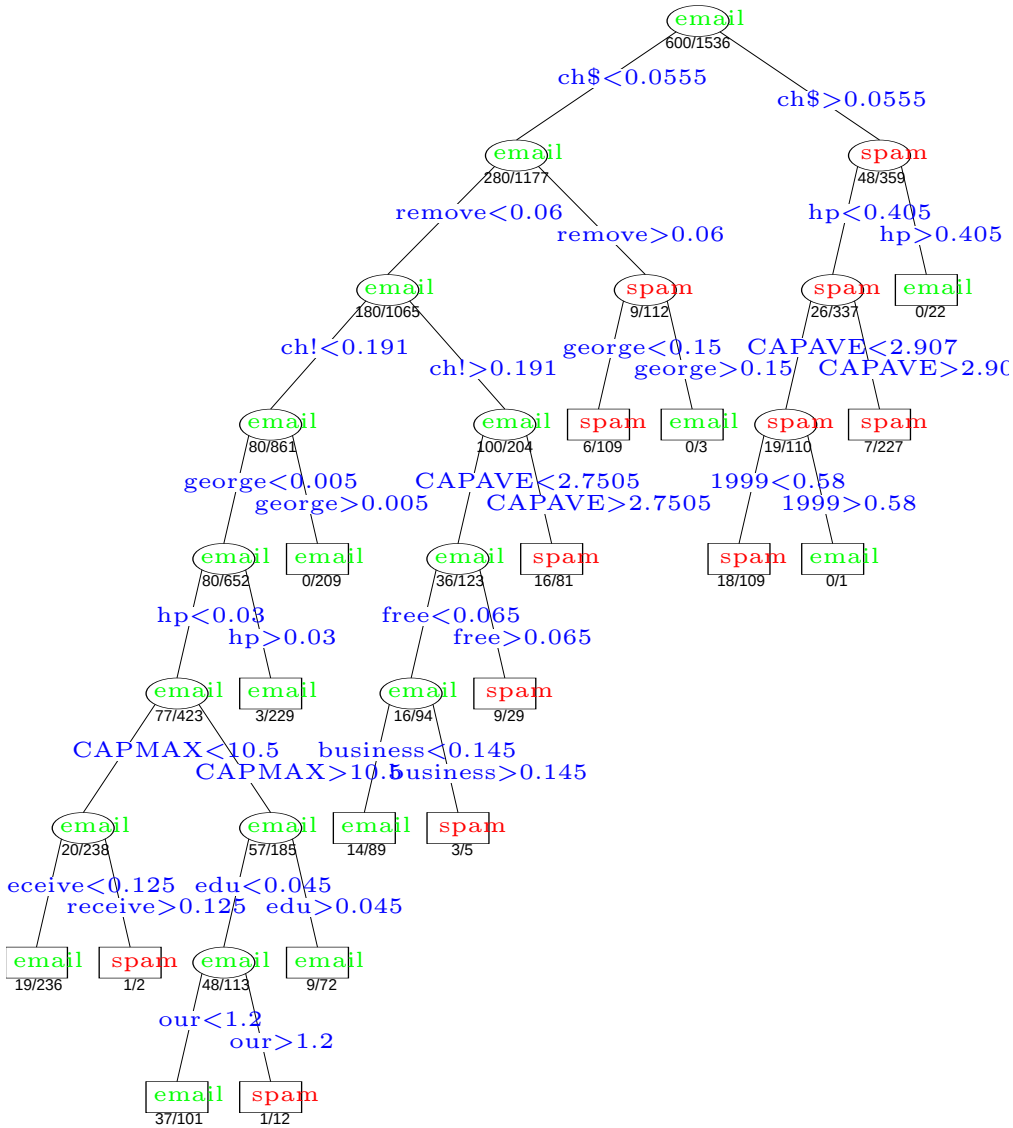


Decision Trees

Decision tree for a given goal attribute G is a rooted tree with

- a root and inner nodes labeled by attributes; for each possible value of the attribute there is an outgoing edge from the node;
- leaves are labeled with predicted goal class $g \in G$ assuming other attributes have values as labeled on the path from the root.

Attributes not present on the path from the root to the leaf are irrelevant.



Construction

Tree construction idea:

- select an attribute; create a node and split the data according the value of the attribute
- for each attribute value construct a subtree based on the appropriate part of the data
- stop if there is a unique value of the goal G in the data or no attributes to split, create a leaf labeled by the most common class $g \in G$.

The criterion to select an attribute follows.

Entropy

The entropy of an attribute A ('uncertainty', negative information) we would like:

- to be zero for the pure data (only one value of the attribute A)
- the highest entropy for uniform distribution on values of A (no information at all)
- two step split leads to the same result as split at once:

$$H([2,3,4]) = H([2,7]) + \frac{7}{9} \cdot H([3,4])$$

Definition

Entropy These properties has the **entropy** $H([p_1, \ldots, p_k]) = -\sum_{i=1}^{k} p_i \log p_i$, the base of the logarithm usually e, sometimes 2.

If we do not normalize we get the entropy multiplied by $\sum_{i=1}^{k} p_i$.

The lower index A denotes the attribute to calculate the entropy H_A, for the goal attribute H_G.

The Entropy for a binary attribute

x axis: p_i, y axis: entropy.
$Gini = 1 - \sum_i (p_i)^2$

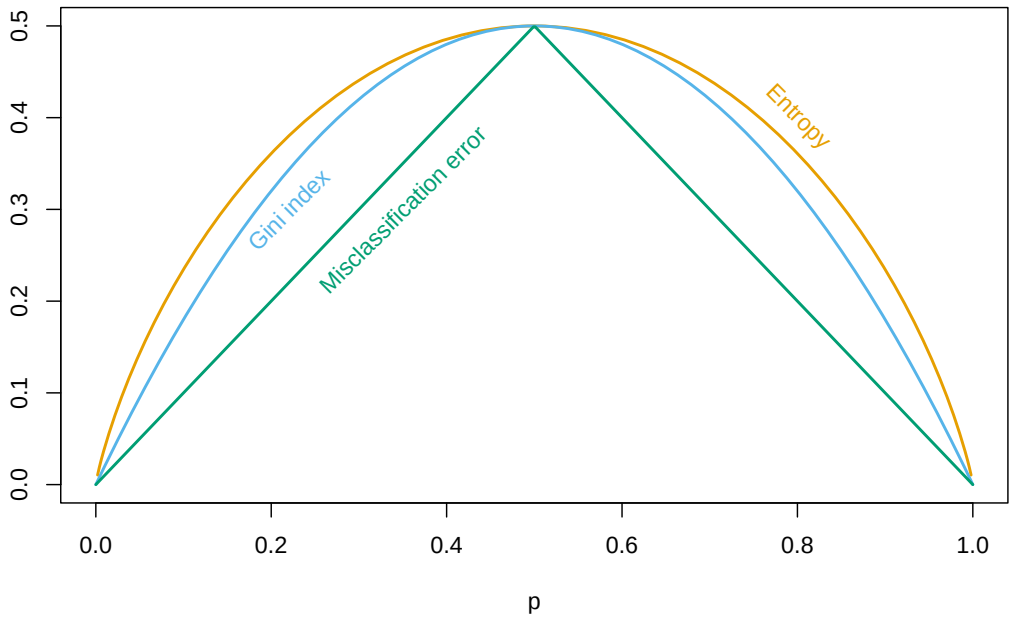


FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

ID3 algorithm

We select an attribute with the maximal **information gain**, defined for the *data* and an attribute X_j:

$$Gain(data, X_j) = H_G(data) - \sum_{x_j \in X_j} \frac{|data_{X_j=x_j}|}{|data|} H_G(data_{X_j=x_j})$$

where $data_{X_j=x_j}$ is a subset of *data* where $X_j = x_j$, the entropy is defined

$$H_G(data) = \sum_{g \in G} -\frac{|data_{G=g}|}{|data|} \cdot log_2 \frac{|data_{G=g}|}{|data|} = \sum_{i=1}^{|G|} -p_i \cdot log_2 p_i$$

where p_i denotes the ratio of $G = g_i$ in the *data*.

It is equivalent to minimize the weighted entropy after the split, that is

$$\arg\min_{X_i} \sum_{x_j \in X_j} \frac{|data_{X_j=x_j}|}{|data|} \sum_{g \in G} -\frac{|data_{G=g \& X_j=x_j}|}{|data_{X_j=x_j}|} \cdot log_2 \frac{|data_{G=g \& X_j=x_j}|}{|data_{X_j=x_j}|}$$

Entropy or 0-1 Error

It is equivalent to minimize the weighted entropy after the split, that is

$$\arg\min_{X_i} \sum_{x_j \in X_j} \frac{|data_{X_j=x_j}|}{|data|} \sum_{g \in G} -\frac{|data_{G=g \& X_j=x_j}|}{|data_{X_j=x_j}|} \cdot log_2 \frac{|data_{G=g \& X_j=x_j}|}{|data_{X_j=x_j}|}$$

- The attributes are A and B.
- Both splits have the same 0-1 error =200.
- Entropy and Gini criterion prefers the 'pure' leaf in the split B.

	$Y == 1$	$Y == 0$	leaf entropy	weighted entropy	0-1 error
A-left	300	100	0.81	0.5 * 0.81	100
A-right	100	300	0.81	0.5 * 0.81	100
A-split				0.81	200
B-left	400	0	0	0.5 * 0	0
B-right	200	200	1	0.5 * 1	200
B-split				0.5	200

1: **procedure** ID3 ALGORITHM:(*data*, *G* goal, *Attributes* attributes)
2: Create the root *root*
3: **if** all data have the same g **then**
4: label the root g and **return** g
5: **end if**
6: **if** no attributes *Attributes* **then**
7: label the *root* by the most frequent g in the *data* and **return** g
8: **end if**
9: $X_j \leftarrow$ the attribute from *Attributes* with the maximal $Gain(data, X_j)$
10: label *root* as X_j
11: **for** each possible value x_j of X_j **do**
12: add an edge from *root* labeled $X_j = x_j$
13: $data_{X_j=x_j} \leftarrow$ the subset of the *data* with $X_j = x_j$
14: **if** $data_{X_j=x_j}$ is empty **then**
15: add a leaf labeled by the most common class g in *data*
16: **else** add a subtree $ID3(data_{X_j=x_j}, G, Attributes \setminus \{X_j\})$
17: **end if**
18: **end for**
19: **return** *root*
20: **end procedure**

Categorical Attribute Notes

- CART in the sklearn DecisionTreeClassifier does not support categorical attributes
  - uses just binary splits.
- It is requires exponential complexity with respect to the number of categories to find optimal binary split.
  - The recommended heuristic is to sort categories according to the goal class probabilities and search the split in a linear time.
- We should avoid the split into too many branches.
  - ID3 used penalization $Gain^*(X_i, data) = \frac{Gain(X_i, data)}{H(X_i)}$
  - so for the identifier with unique values $Gain^*(X_i, data) = \frac{Gain(X_i, data)}{\log N}$.
  - min_samples_split, min_samples_leaf, min_weight_fraction_leaf can do it too.

Prunning Introduction

To avoid overfitting we try to remove unnecessary nodes

- **postprunning** – build a tree, prune afterwards;
  - usuall way
- **preprunning** – prune during the construction
  - This seems nice but we could prune two attributes combined by XOR since both has information gain (close to) zero.

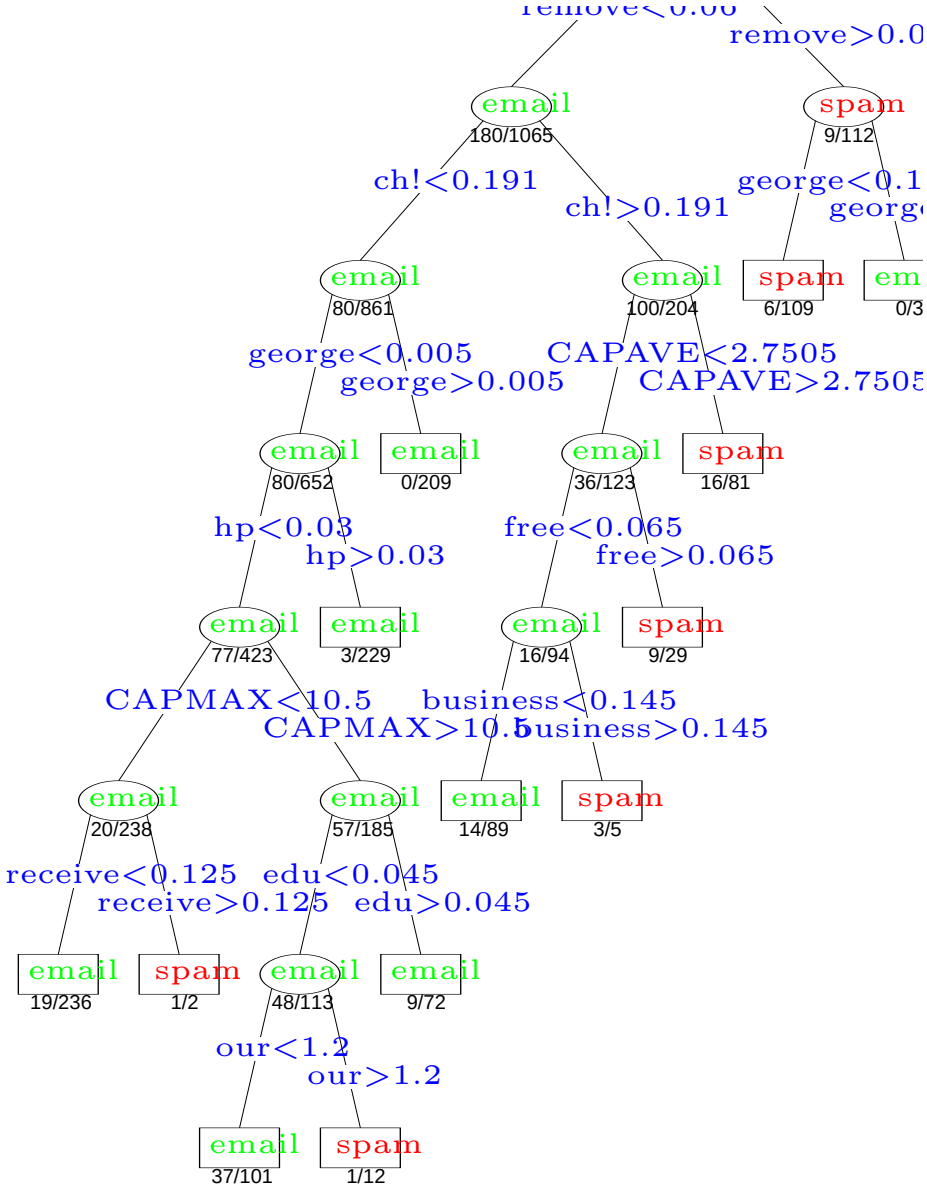
Postprunning

- **subtree replacement** – select a tree and replace it by a leaf;
  - it increases the training error
  - it may decrease the error on validation data
  - step by step, we try to prune each subtree: we prune if we do not increase validation error.
- **subtree raising** – remove an inner node. Used in C4.5. The data samples must be re-send to the remaining branch, it is time consuming.
  - Usually checked only for the most frequent branch in the tree.

Minimal cost-complexity pruning

Reduced Error Pruning!

- **reduced error pruning** – we keep part of the data for validation (pruning).
- for each inner node compare
  - validation error with this node as a leaf
  - validation error with the (pruned) subtree of this node
- select whatever gives the lower error.



- there exist also a criterion based on the training data
- **Minimal cost-complexity pruning** CART - few slides later.

Numerical attributes

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no,yes	yes,yes	no	yes	yes	no

- we require a binary split
- 11 split points
- for each split we calculate the information gain
  - $$H([9,5]) - H([4,2],[5,3]) = H([9,5]) - (\frac{6}{14} \cdot H([4,2]) + \frac{8}{14} \cdot H([5,3]))$$

    $= 0.940 - 0.939$ bits.
- We allow multiple splits based on this attribute.

Regression trees - numerical prediction

- **Model tree** has linear fit in the leaves
  - not so popular as regression trees; increases complexity and discontunuity
- **CART**
  - use the decrease of the square error loss to select an attribute
  - binary splits
  - predict the average value in the leaves.

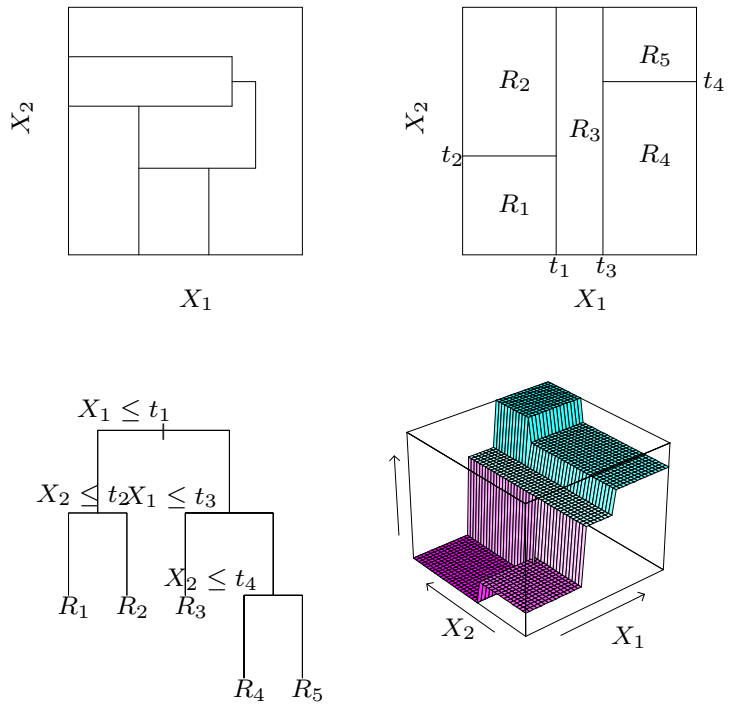


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right*

CART (Classification and) Regression Trees

- Regions t_m (=leaves), we predict a constant c_m inside any region.

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

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in t_m} y_i.$$

Single Regression Tree for CART

- Start with all data in one region t_0
- Select the best attribute j and its value s for the split:

$$\min_{j,s}[min_{c_1} \sum_{x_i \in t_1(j,s)} (y_i - c_1)^2 + min_{c_2} \sum_{x_i \in t_2(j,s)} (y_i - c_2)^2]$$

  - Inner minimum is the average $\hat{c}_1 = ave(y_i | \ x_i \in t_1)$.
- iterate until stop (number of samples in the leaf $\leq n_0$).

One Tree Minimal Cost-Complexity Pruning

- The error for any node t taken as a leaf t is

$$R(t) = \sum_{x_i \in t} \left(y_i - \frac{1}{|x \in t|} \sum_{x_i \in t} y_i \right)^2,$$

- Cost of the tree with α penalty for the number of leaves

$$R_\alpha(T) = \sum_{t \in leaves(T)} R(t) + \alpha |T|.$$

- For each single node $t \in T$ as a leaf the cost complexity is

$$R_\alpha(t) = R(t) + \alpha$$

- For some α_{eff}, this costs is the same as the cost $R(T_t)$ of the subtree rooted at t,

$$\alpha_{eff}(t) = \frac{R(t) - R(T_t)}{|T_t| - 1}$$

- A non-terminal node with the smallest value of α_{eff} is the weakest link and will be pruned

Minimal cost-complexity pruning

- Split the data into K folds
- For each fold k:
  - use all except fold k to train the tree T
  - Build a sequence of subtrees $T^k \supset T_1^k \supset T_2^k \ldots \supset T_{|T|}^k$
    - always join two leaves with the minimal increase in the training error
  - use fold k do calculate the crossvalidation error of each tree
  - consider the error function $R_\alpha(T^k)$ as a function of α
- Select $\alpha \leftarrow \mathit{argmin}_\alpha \sum_k R_\alpha(T^k)$
- Build a tree on the full training data
- Return the subree corresponding to the optimal α.

Missing values,Class and Samples Weights

- Trees can handle missing data well.
- Often we cannot **omit** missing data since many samples have missing values.
  - Furthermore, missing values in unused attributes are irrelevant.
- If the value is **not missing at random** then take the missingness as another value of the attribute
  - example: very small and very high wages are more ofter missing
- If the data are **missing at random**
  - **split the instance**
  - according the data ratio following each branch
  - weight and average the predictions on leaves.
- Similarly, we use weighted information gain to select the attribute.
  - by setting setting class_weight
  - fit(X, y, sample_weight=None).

Complexity considerations

CART

- Let us have N instances with p attributes.
- Assume a reasonably balanced tree with the tree depth $O(logN)$.
- To build the tree we need $O(p \cdot N^2 \cdot logN)$ time.
  - At each depth, each instance occurs exactly ones, $logN$ depth levels, p attributes on each level, the time $O(p \cdot N^2 \cdot \log N)$.
- Subtree replacement $O(N)$, Subtree raising $O(N(logN)^2)$.
- Naive tree construction comlpexity is $O(p \cdot N^2 \cdot logN) + O(N(logN)^2)$.
- With sorted features and clever indexing
  - Overall tree construction comlpexity is $O(p \cdot N \cdot logN) + O(N(logN)^2)$.

Decision Trees, and Related Methods

gam Generalized Additive Models (advanced)

CART! Classification and regression trees

! cost sensitive pruning (α's)

PRIM Patient Rule Induction Method (a note)

MARS Multivariate Additive Regression Splines (advanced)

Decision Rules from Decision Trees

- We can represent a tree as a set of rules
  - one rule for each leaf.
- These rules may be improved by testing each attribute in each rule
  - Has the rule without this test a better precision than with the test?
  - Use validation data
  - May be time consuming.
- These rules are sorted by (decreasing) precision.

Patient Rule Induction Method PRIM = Bump Hunting

- Rule induction method
- We iteratively search regions with the high Y values
  - for each region, a rule is created.
- CART runs of data after aproximately $\log_2(N) - 1$ cuts.
- PRIM can affort $-\frac{\log(N)}{\log(1-\alpha)}$.
  For $N = 128$ data samples and $\alpha = 0.1$ it is 6 and 46 respectively 29, since the number of observations must be a whole number.

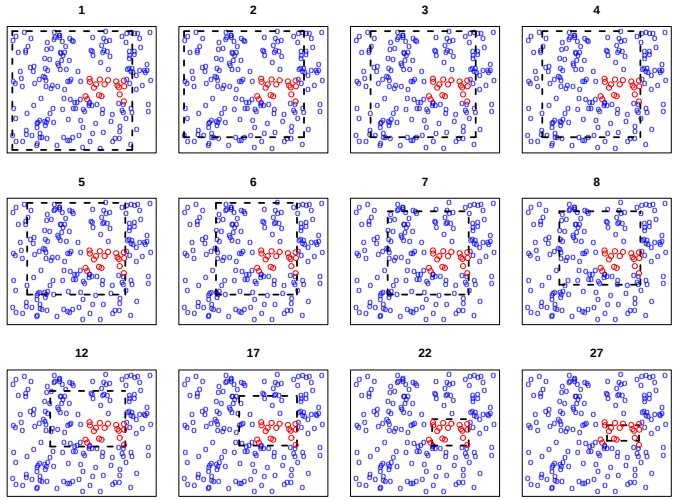
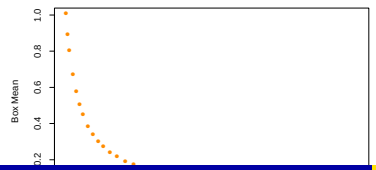


FIGURE 9.7. *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectangle (broken black lines) surrounding all of the data, and then peels away points along one edge by a prespecified amount in order to maximize the mean of the points remaining in the box. Starting at the top left panel, the sequence of peelings is shown, until a pure red region is isolated in the bottom right panel. The iteration number is indicated at the top of each panel.*

PRIM Patient Rule induction Algorithm

PRIM

- Consider the whole space and all data. Set $\alpha = 0.05$ or 0.10.
- Find X_j and its upper or lower boundary such that the cut of $\alpha \cdot 100\%$ observations leads to the maximal mean of the remaining data.
- Repeat until less then 10 observations left.
- Enlarge the region in any direction that increases the mean value.
- Select the number of regions by the crossvalidation. All regions generated 1-4 are considered.
- Denote the best region B_1.
- Create a rule that describes B_1.
- Remove all data in B_1 from the dataset.
- Repeat 2-5, create B_2 continue until final condition met.



CART Weaknesses

- the high variance
  - the tree may be very different for very similar datasets
  - ensemble learning addresses this issue
- the cuts are perpendicular to the axis
- the result is not smooth but stepwise.
  - MARS (Multivariate Adaptive Regression Splines) addresses this issue.
- it is difficult to capture an additive structure

$$Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \ldots + c_k I(X_k < t_k) + \epsilon$$

  - MARS (Multivariate Adaptive Regression Splines) addresses this issue.

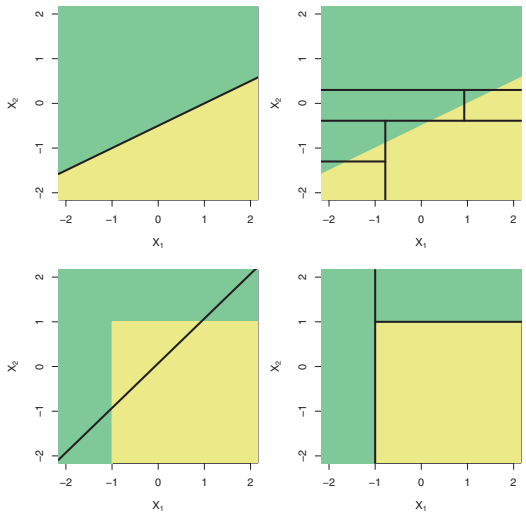


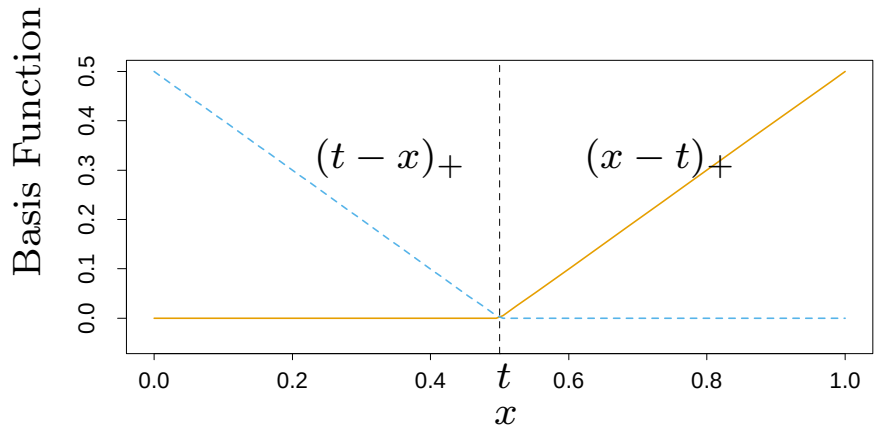
FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a de-*

MARS Multivariate Adaptive Regression Splines

- generalization of linear regression and decision trees CART
- for each feature and each data point we create a **reflected pair** of basis functions
- $(x - t)_+$ and $(t - x)_+$ where $+$ denotes non–negative part, minimum is zero.
- we have the set of functions

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1,j}, x_{2,j}, \ldots, x_{N,j}\}, j=1,2,\ldots,p}$$

- that is $2Np$ functions for non–duplicated data points.

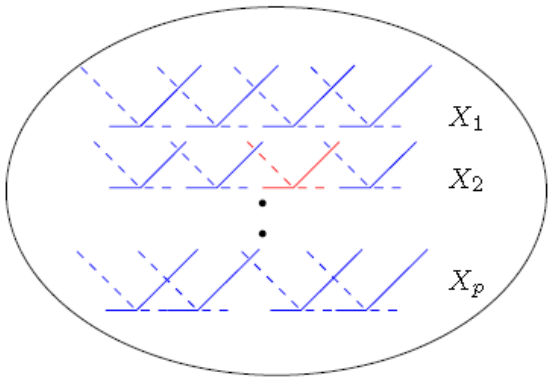


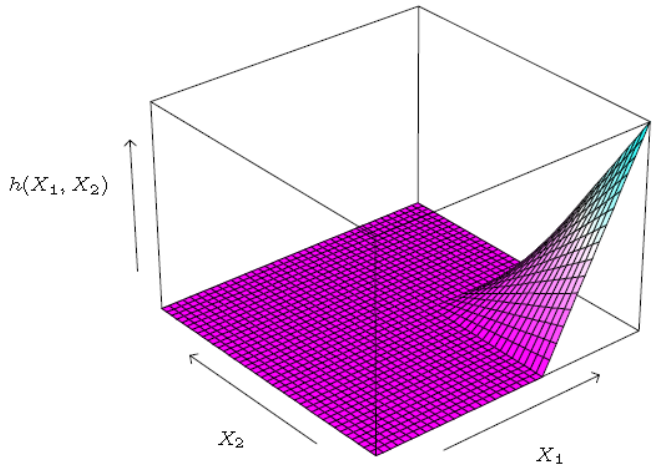
MARS – continuation

- our model is in the form

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

  where $h_m(X)$ is a function from $\mathcal{C}$ or a product of any amount of functions from $\mathcal{C}$
- for a fixed set of h_m's we calculate coefficients β_m by usual linear regression (minimizing RSS)
- the set of functions h_m is selected iteratively.



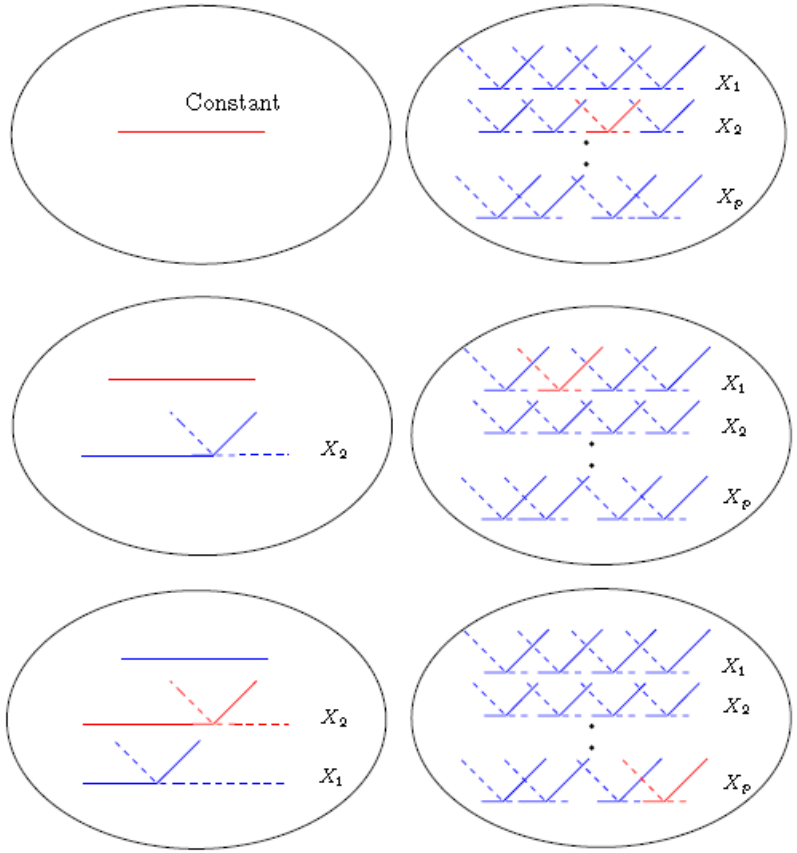


MARS – basis selections

- We start with $h_0 = 1$, we put this function into the model $\mathcal{M} = \{h_0\}$.
- We consider the product of any member $h_\ell \in \mathcal{M}$ with any pair from $\mathcal{C}$,

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

  we select the one minimizing training error RSS (for any product candidate, we estimate $\hat{\beta}$).
- Repeat until predefined number of functions in $\mathcal{M}$



MARS – model pruning

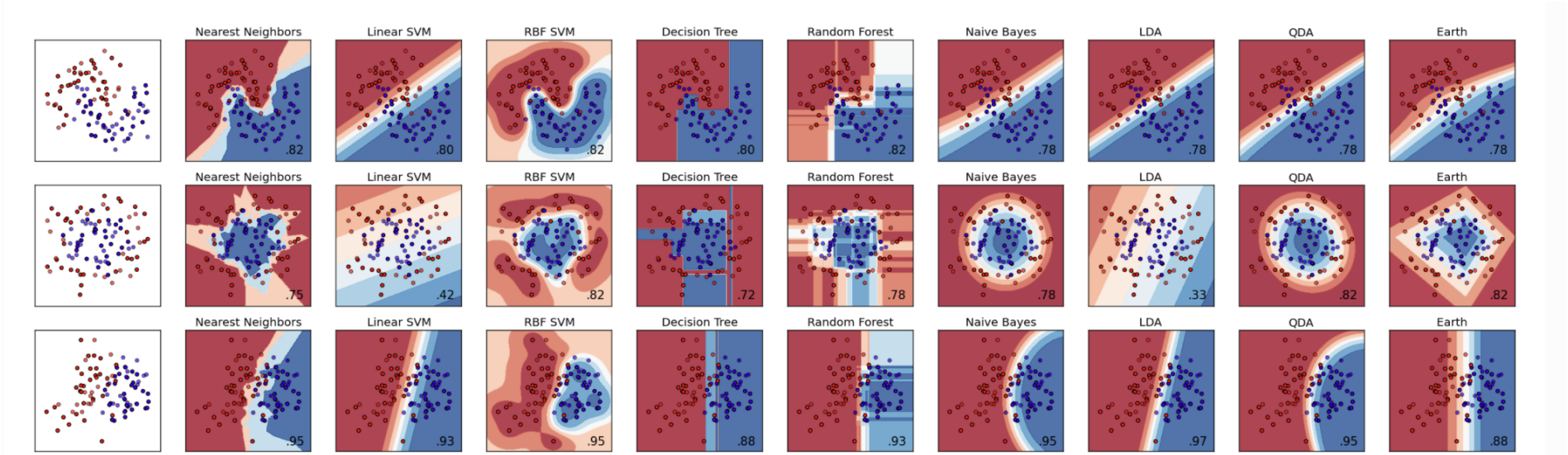
- The model is usually overfitted. We select (remove) iteratively the one minimizing the increase of training RSS. We have a sequence of models $\hat{f}_\lambda$ for different numbers of parameters λ.
- (we want to speed–up cross-validation for computational reasons)
- we select λ (and the model) minimizing **generalized cross-validation**

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

- where $M(\lambda)$ is the number of effective parameters, the number of function h_m (denoted r) plus the number of knots K, the authors suggest to multiply K by 3: $M(\lambda) = r + 3K$.

MARS is a generalization of CART

- We select piecewise constant functions $I(x - t > 0)$ and $I(x - t \leq 0)$
- If h_m uses multiplication we remove this function from the candidate list. It cannot be used any more.
  - This guarantees binary split.
- Its CART.



https://contrib.scikit-learn.org/py-earth/auto_examples/plot_classifier_comp.html

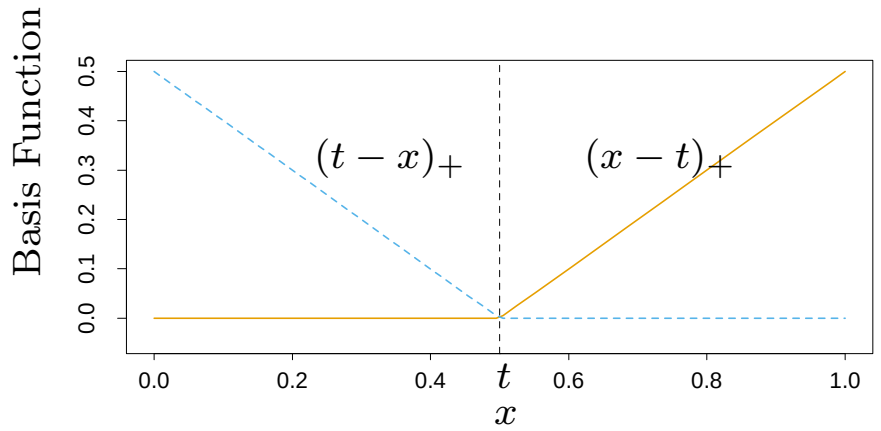
https://contrib.scikit-learn.org/py-earth/auto_examples/index.html

MARS Multivariate Adaptive Regression Splines

- generalization of linear regression and decision trees CART
- for each feature and each data point we create a **reflected pair** of basis functions
- $(x - t)_+$ and $(t - x)_+$ where $+$ denotes non–negative part, minimum is zero.
- we have the set of functions

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1,j}, x_{2,j}, \ldots, x_{N,j}\}, j=1,2,\ldots,p}$$

- that is $2Np$ functions for non–duplicated data points.

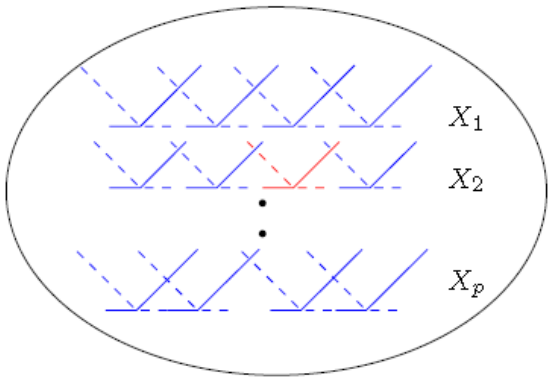


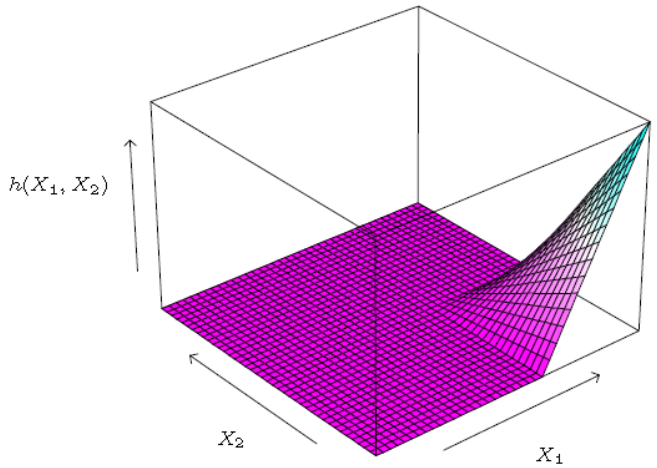
MARS – continuation

- our model is in the form

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

  where $h_m(X)$ is a function from $\mathcal{C}$ or a product of any amount of functions from $\mathcal{C}$
- for a fixed set of h_m's we calculate coefficients β_m by usual linear regression (minimizing RSS)
- the set of functions h_m is selected iteratively.



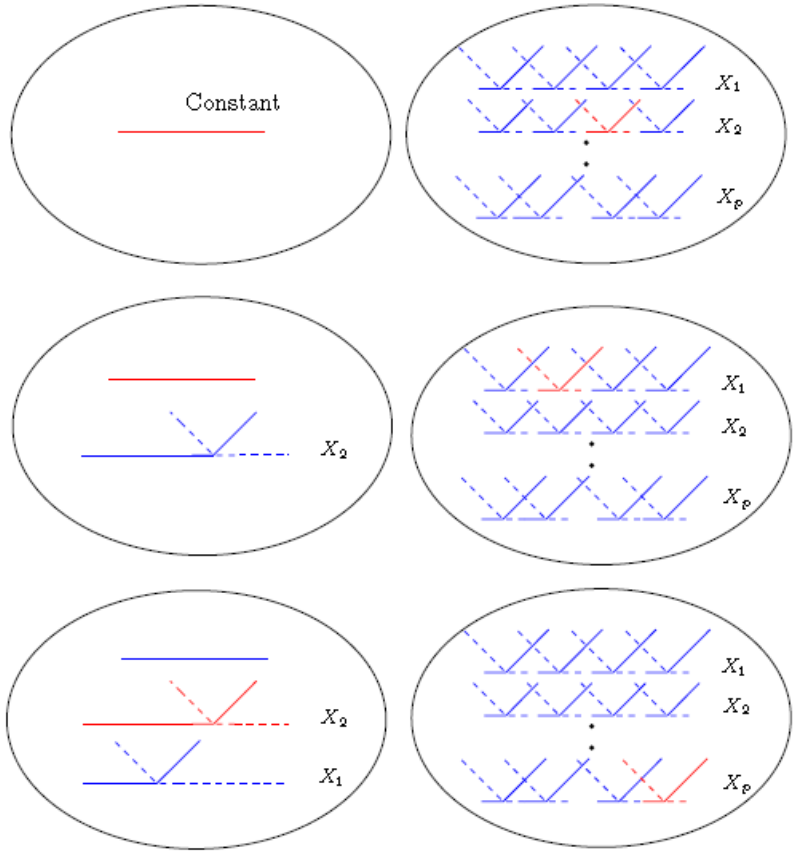


MARS – basis selections

- We start with $h_0 = 1$, we put this function into the model $\mathcal{M} = \{h_0\}$.
- We consider the product of any member $h_\ell \in \mathcal{M}$ with any pair from $\mathcal{C}$,

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

  we select the one minimizing training error RSS (for any product candidate, we estimate $\hat{\beta}$).
- Repeat until predefined number of functions in $\mathcal{M}$



MARS – model pruning

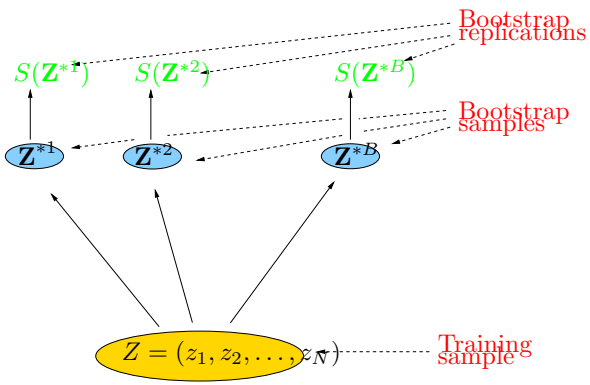
- The model is usually overfitted. We select (remove) iteratively the one minimizing the increase of training RSS. We have a sequence of models $\hat{f}_\lambda$ for different numbers of parameters λ.
- (we want to speed–up crossv-alidation for computational reasons)
- we select λ (and the model) minimizing **generalized cross-validation**

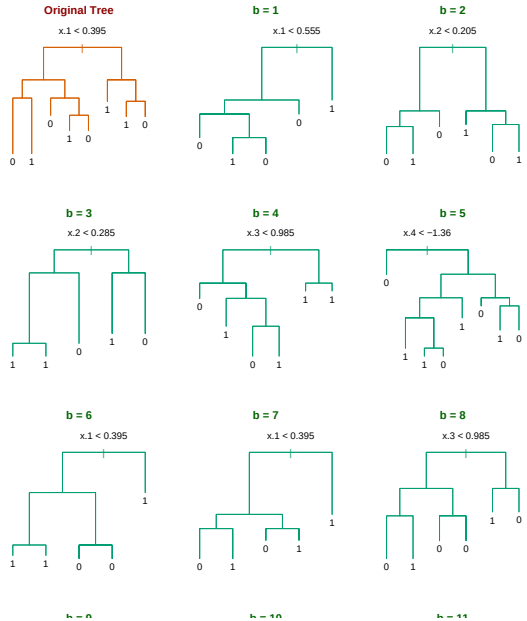
$$GCV(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2}.$$

- where $M(\lambda)$ is the number of effective parameters, the number of function h_m (denoted r) plus the number of knots K, the authors suggest to multiply K by 3: $M(\lambda) = r + 3K$.

Ensemble Methods

- Random forest (+ Bagging)
- Boosting
  - Adaboost - classification
  - Gradient boosting - regression and classification
- Stacking
- MARS (=earth).





Bootstrap

- Select elements with replacement.
- We have N data samples, we select with replacement N samples – some are selected more than one, some are not selected at all. *The not selected are used for testing.*
- The probability of not-selecting a sample is $\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0.368$.
- Selected samples used to learn a model (usually a tree).
- These are used for the **OutOfBag** error computation.

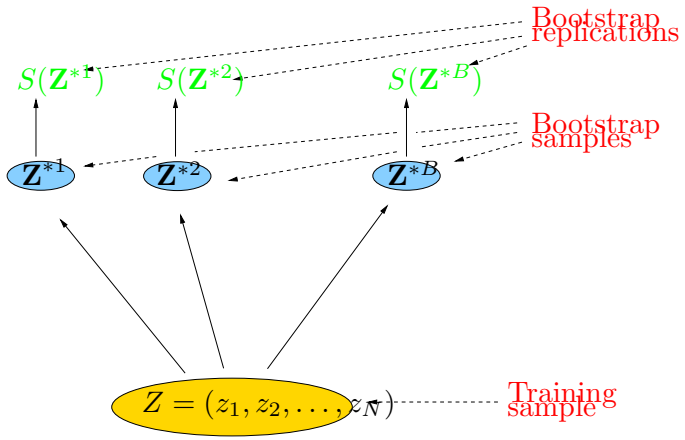


FIGURE 7.12. *Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(\mathbf{Z})$ computed from our dataset. B training sets $\mathbf{Z}^{*b}$, $b = 1, \ldots, B$ each of size N are drawn with replacement from the original dataset. The quantity of interest $S(\mathbf{Z})$ is computed from each bootstrap training set, and the values $S(\mathbf{Z}^{*1}), \ldots, S(\mathbf{Z}^{*B})$ are used to assess the statistical accuracy of $S(\mathbf{Z})$.*

Random Forest for Regression or Classification

procedure RANDOM FOREST:(X, y training data)
  for $b = 1, 2, \ldots, B$ **do**
    Draw a bootstrap sample $\mathbf{Z}^*$ of size N
    Grow a random forest tree T_b
    repeat
      Select m variables at random from p variables.
      Pick the best variable/split–point among the m
      Split the node into two daughter nodes.
    until the minimum node size n_{min} is reached.
  end for
  Output the ensamble of trees $\{T_b\}_1^B$.
end procedure

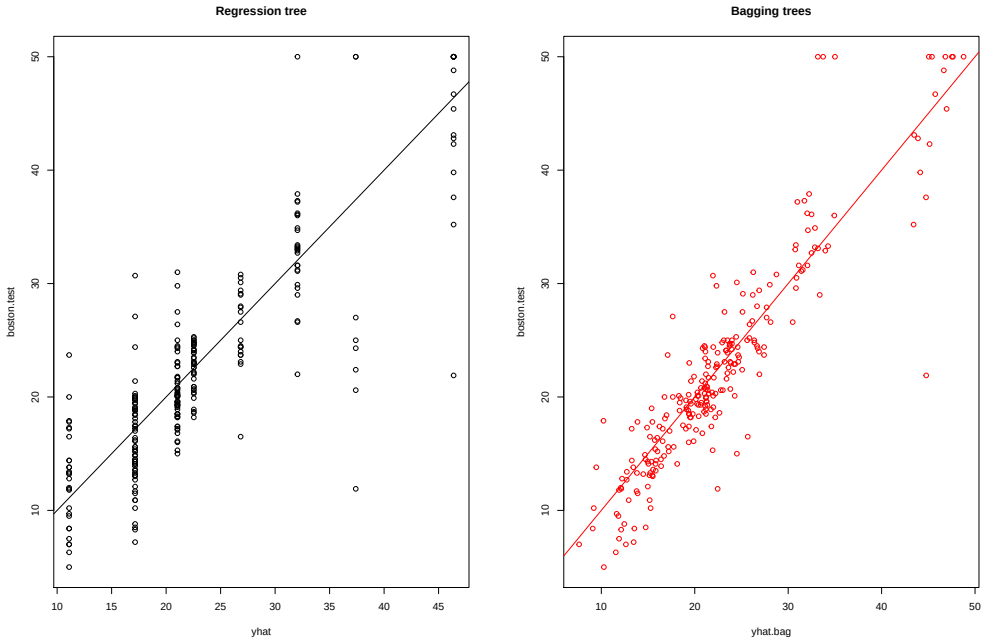
To make a prediction at a new point x:

- Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.
- Classification: Let $\widehat{C}_b(x)$ be the class prediction of the bth random–forest tree.
  - Predict $\widehat{C}_{rf}^B(x) = \textit{majority vote}\ \{\widehat{C}_b(x)\}_1^B$.

Bagging (Bootstrap aggregating)

- It is a Random Forest, where we use all predictors, that is $m = p$.
- both regression and classification.
- Training data $\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

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



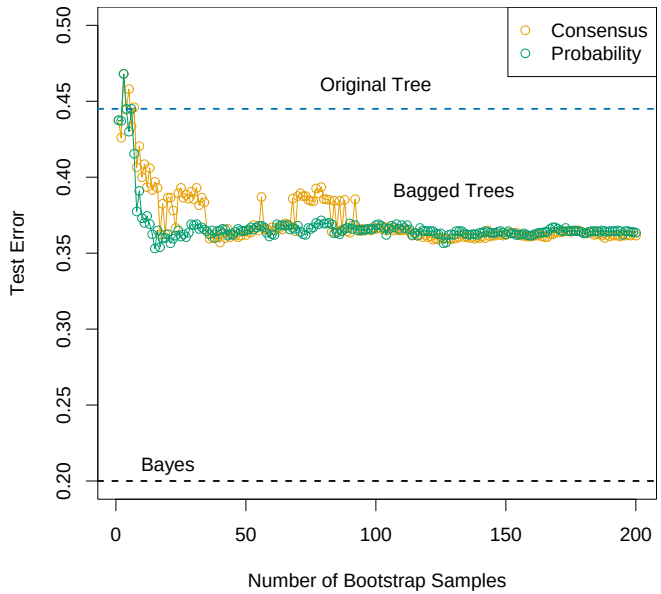
Bagging for Classification

- Training data $\mathbf{Z} = \{(x_1, g_1), (x_2, g_2), \ldots, (x_N, g_N)\}$
- for each bootstrap sample, $b = 1, 2, \ldots, B$, we fit our model, giving prediction $\hat{f}^{*b}(x)$.
- Take either
  - predict probabilities of classes and find the class with the highest predicted probability over the bootstrap samples

$$\hat{G}(x) = \mathit{argmax}_k \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

  - predict class and

$$\hat{G}_{bag}(x) = \mathit{majority\ vote}\ \{\hat{G}^{*b}(x)\}_{b=1}^{B}.$$



Behind Random Forest

The variance of the random forest estimate $Var(\hat{f}_{rf}^{B}(x)) = \mathbb{E}(\hat{f}(x) - \mathbb{E}\hat{f}(x))^2$ is

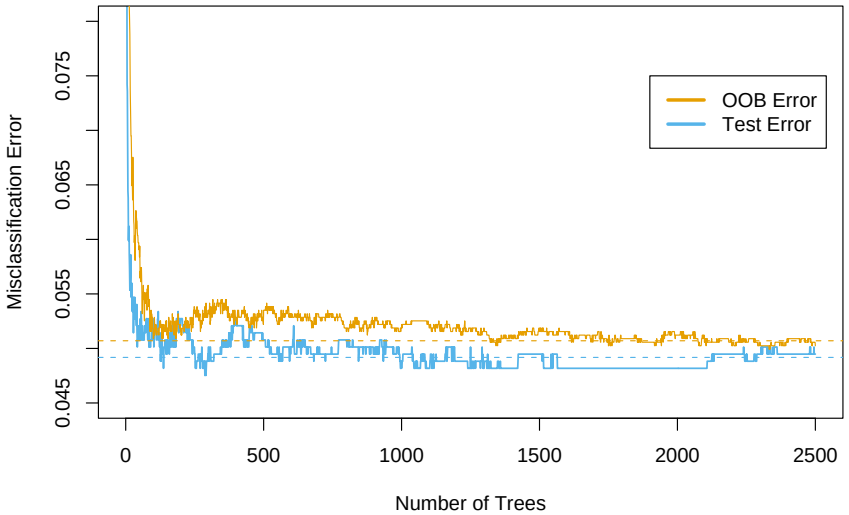
- iid data variables, independent features, each with variance σ^2:
  - $\frac{1}{B}\sigma^2$
- id identically distributed data, each with variance σ^2 with positive pairwise correlation ρ:
  - $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
- The second part is addressed by bagging.
- The idea behind random random forest is to address the first part of the formula.
  - Before each split, select $m \leq p$ variables as candidates for splitting.
  - $m \leftarrow \sqrt{p}$ for regression, even as low as 1. $\frac{p}{3}$ for classification.
- Bagging does not change linear estimates, such as the sample mean
  - The pairwise correlation between bootstrapped means is about 50%.

OOB Error

Definition (Out of bag error (OOB))

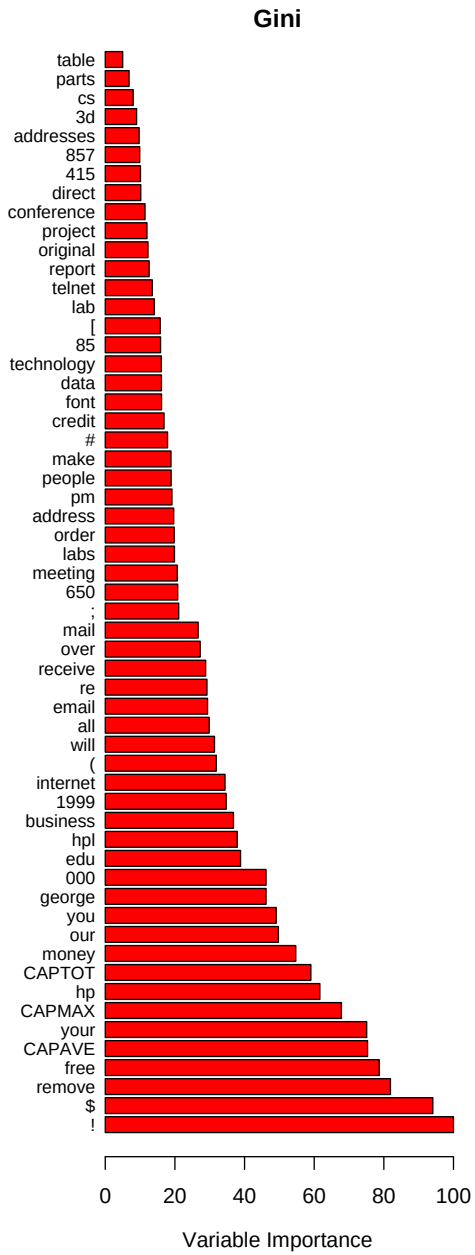
For each observation $z_i = (x_i, y_i)$, construct is random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear.

- An OOB error estimate is almost identical to that obtained by N-fold crossvalidation.
- Unlike many other nonlinear estimators, random forests can be fit in one sequence.



Variable Importance (Gini, RSS)

- **Variable Importance** of a predictor X_ℓ in a single tree T is
  $I_\ell^2(T) = \sum_{t=1}^{J} \hat{i}_t^2 \cdot I(v(t) = \ell)$
  - For each internal node t of the tree, we calculate the *Gini* or *RSS* gain
  - where $\hat{i}_t^2$ is the Gini/RSS improvement of the predictor in the inner node t.
    - Gini $\hat{p}_k(t)(1 - \hat{p}_k(t))$ before and after the split
    - for K goal classes, a separate tree for each class against others
    - weighted by the probability of reaching the node t.
- For a set of trees, we average over M all trees
  $I_\ell^2 = \frac{1}{M} \sum_{i=1}^{M} I_\ell^2(T_m)$.
- Usually scaled to the interval $(0, 100)$.



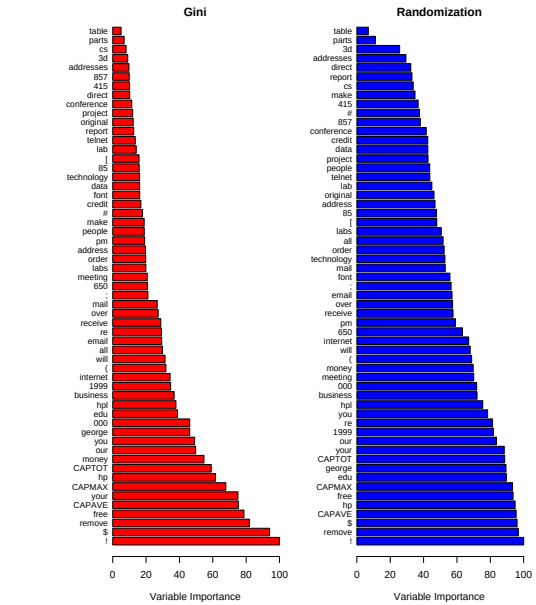
OOB Variable Importance

OOB Variable Importance

procedure OOBN VarImportance:(data)
  for $b = 1, 2, \ldots, B$ **do**
    Draw a bootstrap sample $\mathbf{Z}^*$ of size N
    Grow a random forest tree T_b
    Calculate accuracy on OOB samples
    for $j = 1, 2, \ldots, p$ **do**
      permute the values for the jth variable randomly in the OOB samples
      Calculate the decrease in the accuracy
    end for
  end for
  Output average accuracy gain for each $j = 1, 2, \ldots, p$.
end procedure

Alternative Variable Importance

with quite different results



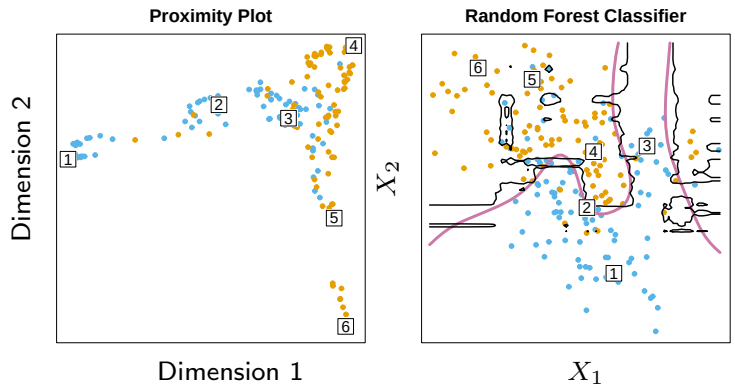
- The randomization voids the effect of a variable.

Proximity plot

Proximity plot

1: **procedure** PROXIMITY PLOT(X, y training data)
2: **for** $b = 1, 2, \ldots, B$ **do**
3: Draw a bootstrap sample $\mathbf{Z}^*$ of size N
4: Grow a random forest tree T_b
5: **Calculate prediction accuracy on OOB samples**
6: **for** any pair of OOB samples sharing the same leaf **do**
7: **increase the proximity by one.**
8: **end for**
9: **end for**
10: **end procedure**

- Distinct samples usually come from the pure regions
- Samples in the 'star center' are close to the decision boundary.

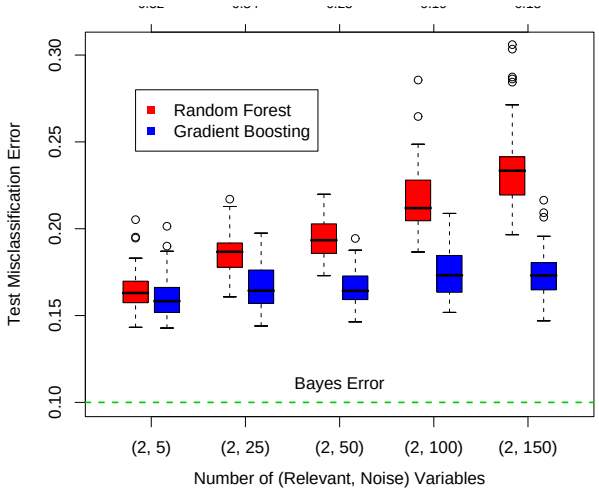


Overfitting

- Though the random forest cannot overfit the limit distribution

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

- the limit distribution (the average of fully grown trees) may overfit the data.
- Small number of relevant variables with many irrelevant hurts the random forest approach.
- With higher number of relevant variables RF is quite robust.
- 6 relevant and 100 noisy variables, $m = \sqrt{6+100} \sim 10$
- probability of a relevant variable being selected at any split is 0.46.



- Seldom the pruning improves the random forest result
- usually, fully grown trees are used.
  - Two additive vars, 10 noisy,
  - plus additive Gaussian noise.

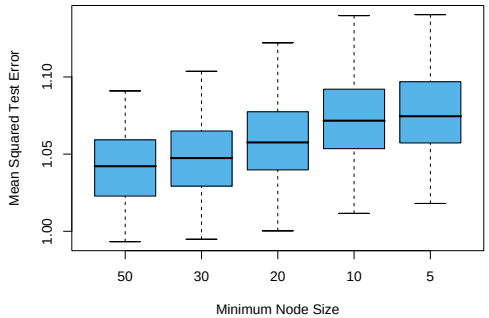


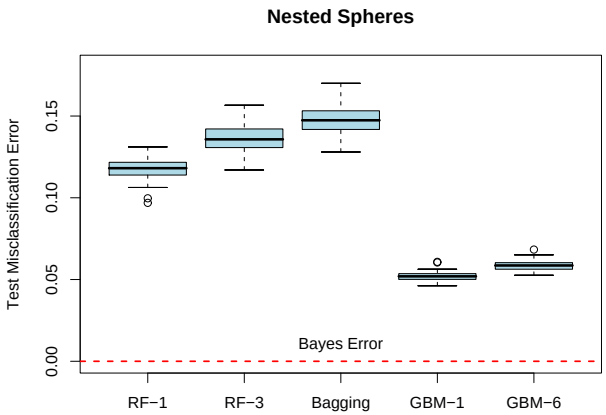
FIGURE 15.8. *The effect of* tree size *on the erro*

Random Forest Experiments

Spam example misclassification error

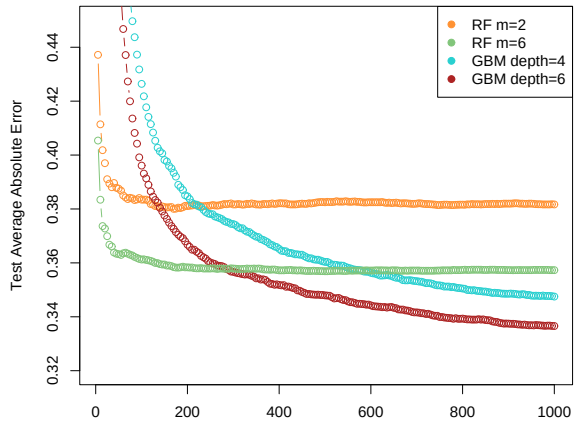
- bagging 5.4%
- random forest 4.88%
- gradient boosting 4.5%.

Nested spheres in $\mathbb{R}^{10}$, 2500 trees, the number selected by 10–fold crossvalidation



California housing data

- Random forests stabilize at about 200 trees, while at 1000 trees boosting continues to improve.
  - Boosting is slowed down by the shrinkage
  - the trees are much smaller (decision stumps, interaction depth=1 or 2).
- Boosting outperforms random forests here.



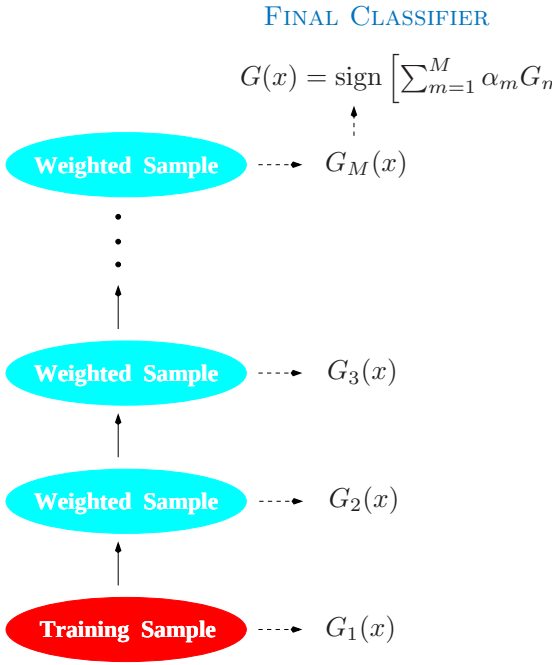
Boosting

! Use a week classifier as a decision stump (a decision tree with the depth= 1).

AdaBoost.M1

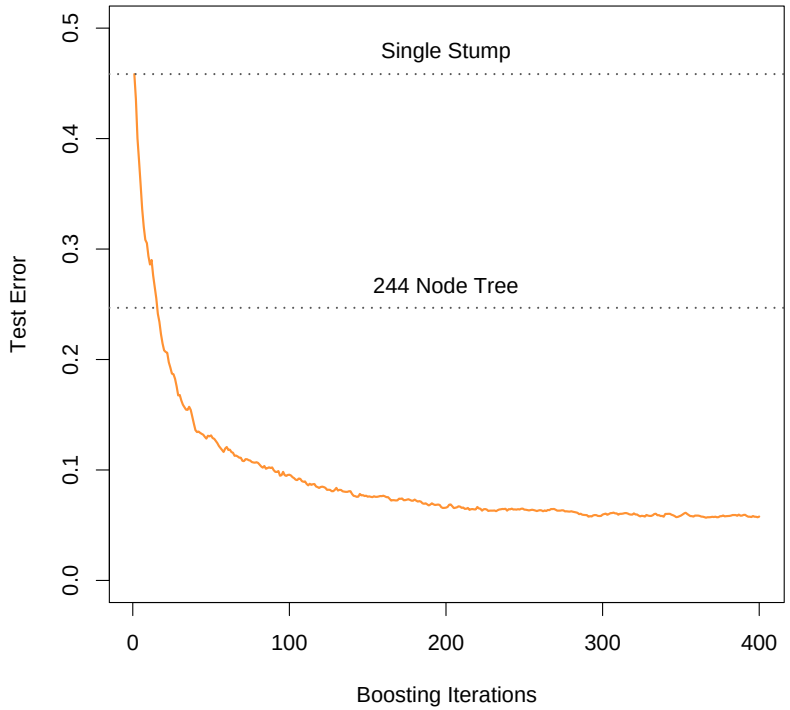
1: **procedure** ADABOOST CLASSIFIER(X, G)
2: Initialize the observation weights $w_i \leftarrow \frac{1}{N}$.
3: **for** $m = 1, 2, \ldots, M$ **do**
4: Fit a classifier $G_m(x)$ to the training data using weights w_i
5: compute $err_m \leftarrow \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
6: compute $\alpha_m \leftarrow log\frac{(1-err_m)}{err_m}$
7: Set $w_i \leftarrow w_i \cdot e^{I(y_i \neq G_m(x_i)) \cdot \alpha_m}$
8: (normalize weights)
9: **end for**
10: Output $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$.
11: **end procedure**

- Two class problem with encoding $Y \in \{-1, 1\}$
- $\overline{err} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i))$.



Nested Spheres Example

- The features $X_1, \ldots, X_{10}$ are standard independent Gaussian
- deterministic target
  - $Y = 1$ iff $\sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) = 9.34$,
  - $Y = -1$ otherwise.
- 2000 training cases
- 10000 test observations.
- Decision stumps.



Additive Model

- We encode the binary goal by $Y \in \{-1, +1\}$.
- Boosting fits an additive model:

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

- where β_m for $m = 1, \ldots, M$ are the expansion coefficients
- $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x
  - characterized by a set of parameters γ.
- For trees, γ parametrizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes.
- Forward stagewise Additive Modeling sequentially adds one new basis function without adjusting the parameters and coefficients of the previously fitted.
- For squared–error loss

$$L(y, f(x)) = (y - f(x))^2,$$

we have

$$\begin{aligned} L(y_i, f_{m-1}(x) + \beta_m b(x_i; \gamma_m)) &= (y_i - f_{m-1}(x) - \beta_m b(x_i; \gamma_m))^2 \\ &= (r_{im} - \beta_m b(x_i; \gamma_m))^2 \end{aligned}$$

Exponential Loss and AdaBoost

- Let us use the $Y \in \{-1, 1\}$ encoding and the **exponential loss**

$$L(y, f(x)) = e^{-yf(x)}.$$

- We have to solve

$$\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{N} e^{[-y_i(f_{m-1}(x_i)+\beta G(x_i)]} \\
&= \arg\min_{\beta, G} \sum_{i=1}^{N} e^{[-y_i(f_{m-1}(x_i)]} e^{[-y_i \beta G(x_i)]} \\
&= \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} e^{[-y_i \beta G(x_i)]}
\end{aligned}$$

- where $w_i^{(m)} = e^{[-y_i f_{m-1}(x_i)]}$ does not depend on β nor $G(x)$.
- this weight depends on $f_{m-1}(x_i)$ and change with each iteration m.

Exponential Loss and AdaBoost

- From

$$
\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} e^{[-y_i \beta G(x_i)]} \\
&= \arg\min_{\beta, G} \left[e^{\beta} \cdot \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\beta} \cdot \sum_{y_i = G(x_i)} w_i^{(m)} \right] \\
&= \arg\min_{\beta, G} \left[(e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \cdot \sum_{i=1}^{N} w_i^{(m)} \right]
\end{aligned}
$$

- For any $\beta > 0$ the solution for $G_m(x; \gamma)$ is

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

- Recall the error definition:

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

Adaboost Update

$$\arg\min_{\beta,G}\left[(e^{\beta}-e^{-\beta})\cdot\sum_{i=1}^{N}w_i^{(m)}I(y_i\neq G(x_i))+e^{-\beta}\cdot\sum_{i=1}^{N}w_i^{(m)}\right]$$

- The minimum w.r.t. β_m is:

$$\beta_m=\frac{1}{2}log\frac{1-err_m}{err_m}$$

- The approximation is updated

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

- which causes the weights for the next iteration to be:

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

- using the fact $-y_i G_m(x_i)=2\cdot I(y_i\neq G_m(x_i))-1$ we get

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

Why exponential loss?

- The population minimizer is

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

- therefore

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

- The same function $f^*(x)$ minimizes also deviance (cross–entropy, binomial negative log–likelihood)
  - interpreting f^* as the logit transform. Let:

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

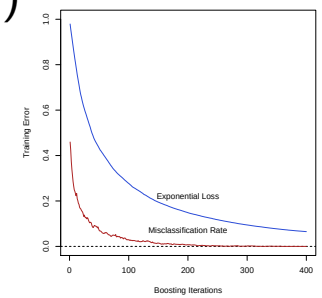
  - and define $Y^| = (Y+1)/2 \in \{0,1\}$. Log–likelihood is

$$\ell(Y, p(x)) = Y^| \log p(x) + (1-Y^|) \log(1-p(x))$$

  - or equivalently the deviance:

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

- Exponential loss decreases long after misclassification loss is stable at zero.



Forward Stagewise Additive Modeling

- A general iterative fitting approach.
- In each step, we select the best function from the dictionary $b(x_i; \gamma)$, fit its parameters γ and the weight of this basis function β_m.
- Stagewise approximation is often faster then iterative fitting of the full model.

Forward Stagewise Additive Modeling

procedure FORWARD STAGEWISE ADDITIVE MODELING(L, X, Y, b)
    Initialize $f_0 \leftarrow 0$.
    for $m = 1, 2, \ldots, M$ **do**
        Compute $(\beta_m, \gamma_m) \leftarrow \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
        Set $f_m(x) \leftarrow f_{m-1}(x) + \beta_m b(x_i; \gamma_m)$
    end for
end procedure

- For example, our basis functions are decision trees, γ represents the splits and fitted values $T(*; \gamma)$).
- For square error loss, any new tree $T(*; \gamma)$ is the best tree fitting residuals $r_i = y_i - f_{m-1}(x_i)$.

Gradient Tree Boosting Algorithm

Gradient Tree Boosting Algorithm

procedure GRADIENT TREE BOOSTING ALGORITHM(X, Y, L)

  Initialize $f_0(x) \leftarrow \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.

  for $m = 1, 2, \ldots, M$ **do**

    for $i = 1, 2, \ldots, N$ **do**

      compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} =^{[*]} y_i - f_{m-1}(x_i)$

    end for

    Fit reg. tree to the target r_{im} giving regions $\{R_{jm}\}_{j=1,\ldots,J_m}$.

    for $j = 1, 2, \ldots, J_m$ **do**

      Compute $\gamma_{jm} \leftarrow \arg\min_\gamma \sum_{i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.

    end for

    Set $f_m(x) \leftarrow f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

  end for

  Output $\hat{f}(x) = f_M(x)$.

end procedure

[*] for square error loss.

Stacking

- Over a set of models (possibly different types) learn a simple model (like a linear regression)
- Assume predictions $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_M(x)$ under square error loss
- Predictors trained without ith example are denoted
  - $\hat{f}_1^{-i}(x), \hat{f}_2^{-i}(x), \ldots, \hat{f}_M^{-i}(x)$
- we can seek weights $w = (w_1, \ldots, w_m)$ such that

$$\hat{w}^{st} = \arg\min_w \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x) \right]^2 .$$

- The final prediction is

$$\hat{f}^{st}(x) = \sum_{m=1}^{M} w_m^{st} \hat{f}_m(x).$$

  - Using cross–validated predictions $\hat{f}_m^{-i}(x)$ stacking avoids giving unfairly high weight to models with higher complexity
  - Better results can be obtained by restricting the weights to be nonnegative and to sum to 1.

List of topics

1. Linear, ridge, lasso regression, k-neares neighbours,(formulas) overfitting, curse of dimensionality, (LARS)
2. Splines - the base, natural splines, smoothing splines; kernel smoothing: kernel average, Epanechnikov kernel.
3. Logistic regression, Linear discriminant analysis, generalized additive models
4. Train/test error and data split, square error, 0-1, crossentropy, AIC, BIC,(formulas) crossvalidation, one-leave-out CV, wrong estimate example
5. decision trees, information gain/entropy/gini, CART prunning,(formulas)
6. random forest (+bagging), OOB error, Variable importance, boosting (Adaboost(formulas) and gradient boosting), stacking, MARS,
7. Bayesian learning: MAP, ML hypothesis (formulas), Bayesian optimal prediction, EM algorithm
8. Clustering: k-means, Silhouette, k-medoids, hierarchical
9. Apriori algorithm, Association rules, support, confidence, lift
10. Inductive logic programming basic: hypothesis space search, background knowledge, necessity, sufficiency and consistency of a hypothesis, Aleph
11. Undirected graphical models, Graphical Lasso procedure, deviance, MRF
12. Gaussian processes: estimation of the function and its variance (figures, ideas).

List of topics

1. Linear, ridge, lasso regression, k-neares neighbours,(formulas) overfitting, curse of dimensionality, (LARS)
2. Splines - the base, natural splines, smoothing splines; kernel smoothing: kernel average, Epanechnikov kernel.
3. Logistic regression, Linear discriminant analysis, generalized additive models
4. Train/test error and data split, square error, 0-1, crossentropy, AIC, BIC,(formulas) crossvalidation, one-leave-out CV, wrong estimate example
5. decision trees, information gain/entropy/gini, CART prunning,(formulas)
6. random forest (+bagging), OOB error, Variable importance, boosting (Adaboost(formulas) and gradient boosting), stacking, MARS,
7. Bayesian learning: MAP, ML hypothesis (formulas), Bayesian optimal prediction, EM algorithm
8. Clustering: k-means, Silhouette, k-medoids, hierarchical
9. Apriori algorithm, Association rules, support, confidence, lift
10. Inductive logic programming basic: hypothesis space search, background knowledge, necessity, sufficiency and consistency of a hypothesis, Aleph
11. Undirected graphical models, Graphical Lasso procedure, deviance, MRF
12. Gaussian processes: estimation of the function and its variance (figures, ideas).

Table of Contens