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.
  - $g(\mu) = logit(\mu)$ as above
  - $g(\mu) = probit(\mu)$ **probit** link function for modeling binomial probabilities is the inverse of Gaussian cumulative distribution function $probit(\mu) = \Phi^{-1}(\mu)$
  - $g(\mu) = \log(\mu)$ 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 $j = 1, 2, \dots, p, \dots, 1, 2, \dots$
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

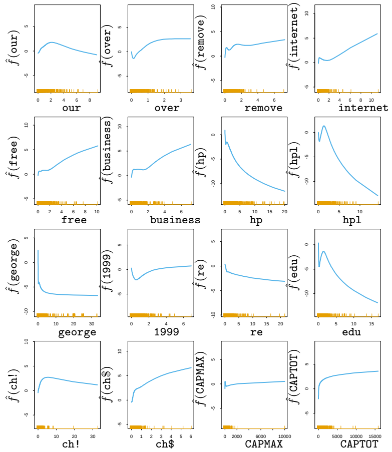
```
1: procedure ADDITIVE LOGISTIC REGRESSION:(X, y in 0-1 encoding)
2:     $\hat{y} = \frac{1}{N}\sum_1^N y_i, \hat{\alpha} = \log(\frac{\hat{y}}{1-\hat{y}})$, $\hat{f}_j \equiv 0$ initialize $\forall j$.
3:     $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = \frac{1}{1+\exp(-\hat{\eta}_i)}$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

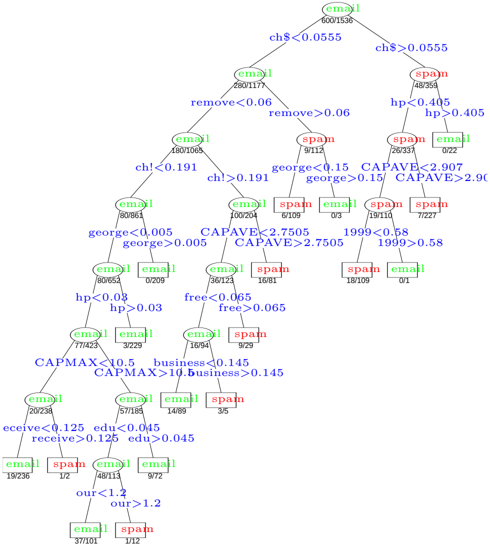


Decision Trees

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.
$Gini = 1 - \sum_i (p_i)^2$

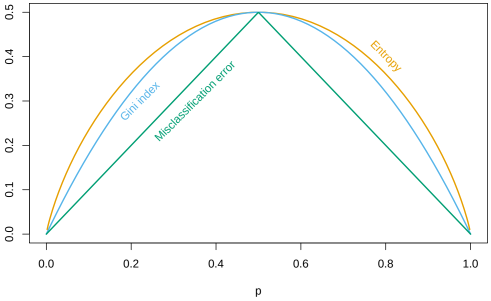


FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through $(0.5, 0.5)$.*

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

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

ID3 algorithm(*data*, *G* goal, *Attributes* attributes)

ID3

```
Create the root root
If all data have the same $g$, label the root $g$ and return,
If no attributes Attributes, label the root
            by the most frequent $g$ in the data and return
otherwise
    $X_j \leftarrow$ the attribute from Attributes with the maximal $Gain(data, X_j)$
    label root as $X_j$
    for each possible value $x_j$ of $X_j$,
            add an edge from root labeled $X_j = x_j$
            $data_{X_j=x_j} \leftarrow$ the subset of the data with $X_j = x_j$
            If $data_{X_j=x_j}$ is empty, add a leaf labeled by
                    the most common class $g$ in data and return
            add a subtree $ID3(data_{X_j=x_j}, G, Attributes \setminus \{X_j\})$
return root
```

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

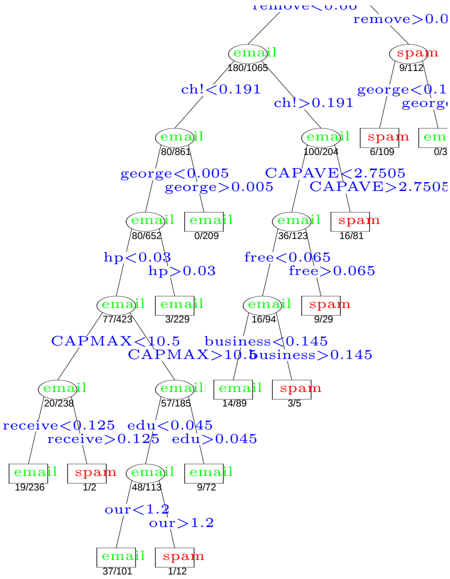
- **subtree replacement** – select a tree and replace it by a leaf;
  - it increases the training error
  - it may decrease the error on validation data
  - step by step, we try to prune each subtree: we prune if we do not increase validation error.
- **subtree raising** – remove an inner node. Used 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.

Reduced Error 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
- **Reduced Cost 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.

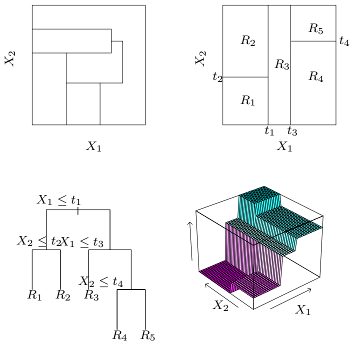


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the 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 R_m, we predict a constant c_m inside any region.

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$
$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i.$$

Single Regression Tree for CART

- Start with all data in one region R_0
- Select the best attribute j and its value s for the split:

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

  - Inner minimum is the average $\hat{c}_1 = ave(y_i|\ x_i \in R_1)$.
- iterate until stop (number of samples in the leaf $\leq n_0$).

Reduced Cost 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 $C_\alpha(T^k)$ as a function of α
- Select $\alpha \leftarrow argmin_\alpha \sum_k C_\alpha(T^k)$
- Build a tree on the full training data
- Return the subree corresponding to the optimal α.

- Average error on a leaf m

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \frac{1}{N_m} \sum_{x_i \in R_m} y_i)^2,$$

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

$$C_\alpha(T) \quad = \quad \sum^{|T|} N_m Q_m(T) + \alpha |T|.$$

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 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.

Loss Matix

The cost of missclassification may be different for each class. The general loss specification is a **loss matrix** $L_{kk^{|}}$, an element represent the cost of classifying k as $k^{|}$. Must be zero at the diagonal, non-negative everywhere.

- we can modify $Gini(m) = \sum_{k \neq k^{|}} L_{kk^{|}} \hat{p}_{mk} \hat{p}_{mk^{|}}$
- or weight the data samples k $L_{kk^{|}}$ times (only in binary classification)
- we classify according to $k(m) = argmin_k \sum_l L_{lk} \hat{p}_{ml}$ in the leaves.

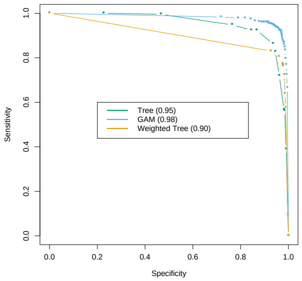


FIGURE 9.6. *ROC curves for the classification rules fit to the* `spam` *data. Curves that are closer to the north-east corner represent better classifiers. In this case the GAM classifier dominates the trees. The weighted tree achieves better sensitivity for higher specificity than the unweighted tree. The numbers in the legend represent the area under the curve.*

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.

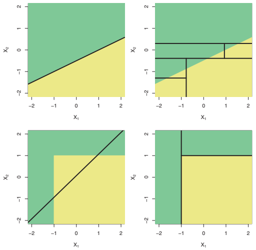


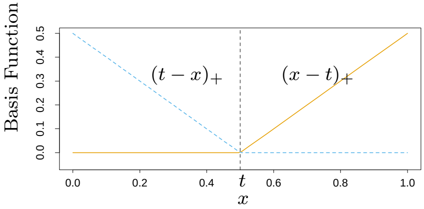
FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a de-*

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.

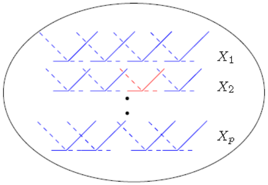


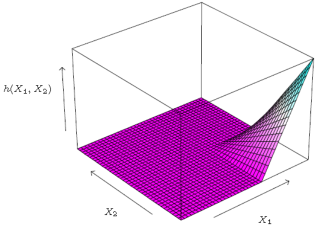
MARS – continuation

- 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.



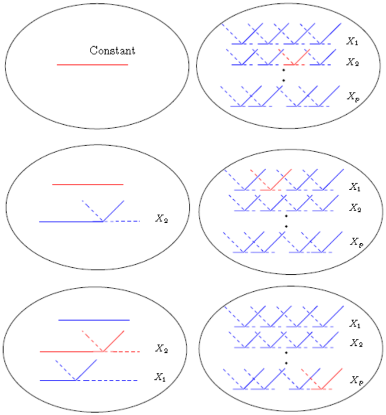


MARS – basis selections

- 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

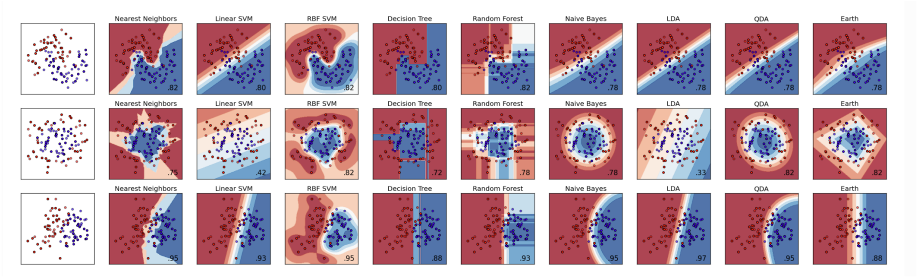
- The model is usually overfitted. We select (remove) iteratively the one minimizing the increase of training RSS. We have a sequence of models $\hat{f}_\lambda$ for different numbers of parameters λ.
- (we want to speed–up cross-validation for computational reasons)
- we select λ (and the model) minimizing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

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.

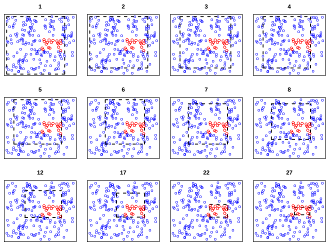


FIGURE 9.7. *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectangle (broken black lines) surrounding all of the data, and then peels away points along one edge by a prespecified amount in order to maximize 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.

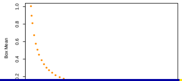


Table of Contens