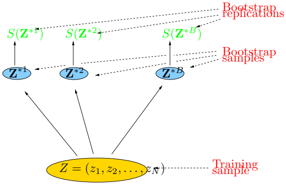
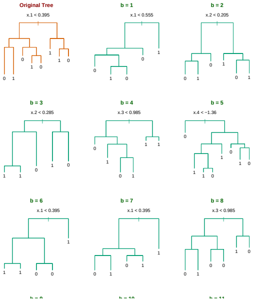
# Ensemble Methods

- 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.

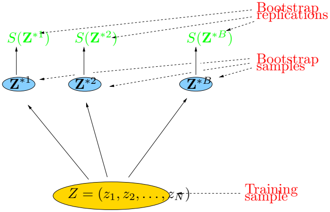



**FIGURE 7.12.** *Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(\mathbf{Z})$ computed from our dataset. $B$ training sets $\mathbf{Z}^{*b}$, $b = 1, \ldots, B$ each of size $N$ are drawn with replacement from the original dataset. The quantity of interest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

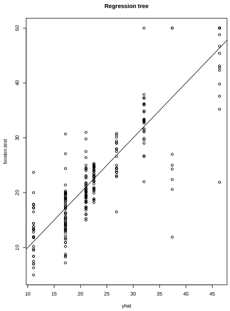
To make a prediction at a new point $x$:

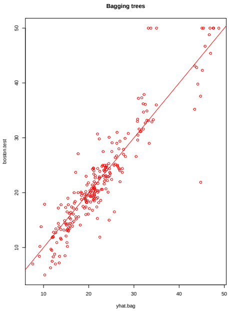
- Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
- Classification: Let $\widehat{C}_b(x)$ be the class prediction of the $b$th random–forest tree.
    - Predict $\widehat{C}_{rf}^B(x) = \textit{majority vote}\ \{\widehat{C}_b(x)\}_1^B$.

# Bagging (Bootstrap aggregating)

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

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



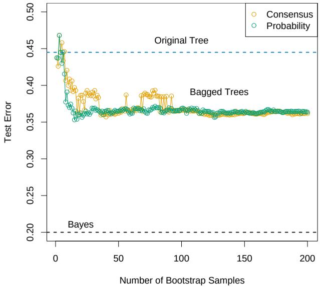
# Bagging for Classification

- 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

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

  - predict class and

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



# 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

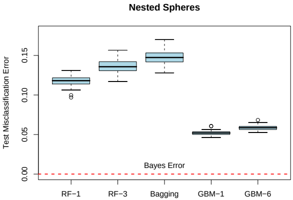
- iid data variables, independent features, each with variance $\sigma^2$:
  - $\frac{1}{B}\sigma^2$
- id identically distributed data, each with variance $\sigma^2$ with positive pairwise correlation $\rho$:
  - $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
- The second part is addressed by bagging.
- 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.
- For boot–strapped trees
  - $\rho$ is typically small (0.05 or lower)
  - $\sigma^2$ is not much larger than for the original tree.
- Bagging does not change linear estimates, such as the sample mean
  - The pairwise correlation between bootstrapped means is about 50%.

# Random Forest Experiments

Spam example misclassification error

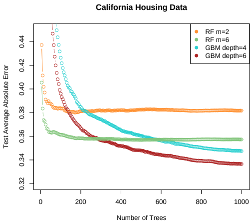
- bagging 5.4%
- random forest 4.88%
- gradient boosting 4.5%

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.

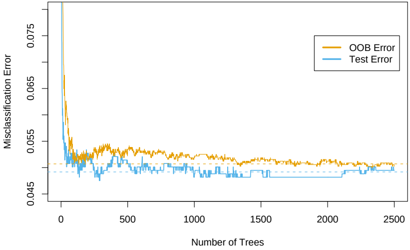


# OOB Error

**Definition (Out of bag error (OOB))**

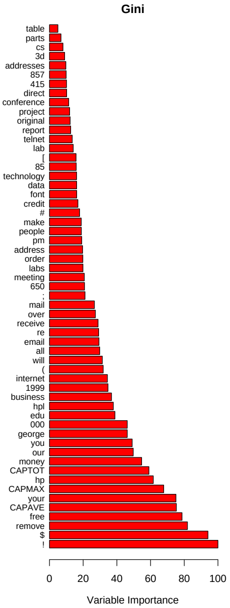
For each observation $z_i = (x_i, y_i)$, construct is random forest predictor by averaging only those trees corresponding to bootstrap samples in which $z_i$ did not appear.

- 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 $X_\ell$ 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 $(0, 100)$.



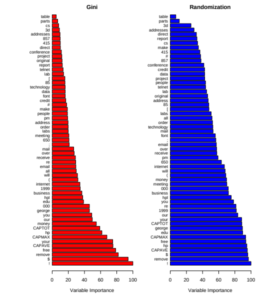
# OOB Variable Importance

## OOB Variable Importance

```
procedure OOBN VarImportance:(data)
    for b = 1, 2, ..., B do
        Draw a bootstrap sample Z* of size N
        Grow a random forest tree T_b
        Calculate accuracy on OOB samples
        for j = 1, 2, ..., 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, ..., p.
end procedure
```

Alternative Variable Importance

with quite different results



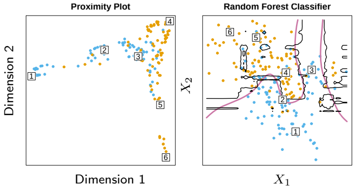

- The randomization voids the effect of a variable.

# Proximity plot

## Proximity plot

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

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

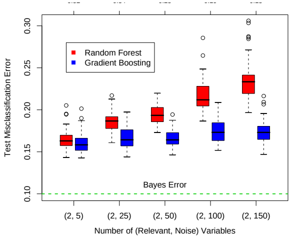


# Overfitting

- 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.

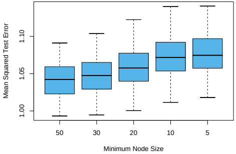



**FIGURE 15.8.** *The effect of tree size on the erro*

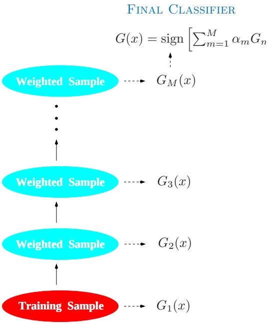
# Boosting

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

**AdaBoost.M1**

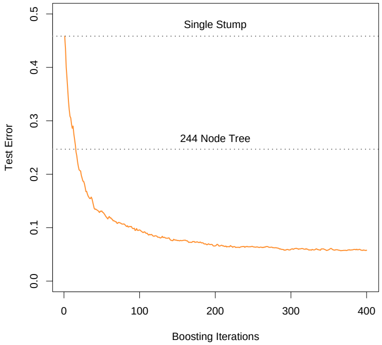
**procedure** ADABOOST CLASSIFIER( $X, G$)
  Initialize the observation weights $w_i \leftarrow \frac{1}{N}$.
  **for** $m = 1, 2, \ldots, M$ **do**
    Fit a classifier $G_m(x)$ to the training data using weights $w_i$
    compute $err_m \leftarrow \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
    compute $\alpha_m \leftarrow log \frac{(1-err_m)}{err_m}$
    Set $w_i \leftarrow w_i \cdot e^{I(y_i \neq G_m(x_i)) \cdot \alpha_m}$
    (normalize weights)
  **end for**
  Output $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$.
**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 $\sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5) = 9.34$,
  - $Y = -1$ 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 $\beta_m$ 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 $\gamma$.
- For trees, $\gamma$ 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 $\beta$ nor $G(x)$.
- this weight depends on $f_{m-1}(x_i)$ and change with each iteration $m$.

# Exponential Loss and AdaBoost

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

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

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

- since

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

# Adaboost Update

- Solving previous equation for $\beta_m$ gives:

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

- 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:

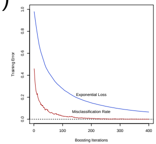
$$p(x) = P(Y=1|x) = \frac{e^{f^*(x)}}{e^{-f^*(x)}+e^{f^*(x)}} = \frac{1}{1+e^{-2f^*(x)}}.$$

  - 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 $\gamma$ and the weight of this basis function $\beta_m$.
- Stagewise approximation is often faster then iterative fitting of the full model.

**Forward Stagewise Additive Modeling**

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

- For example, our basis functions are decision trees, $\gamma$ 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 $i$th 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.

# 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.

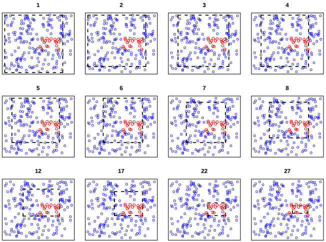
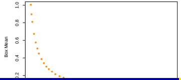



**FIGURE 9.7.** *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectangle (broken black lines) surrounding all of the data, and then peels away points along one edge by a prespecified amount in order to maximize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

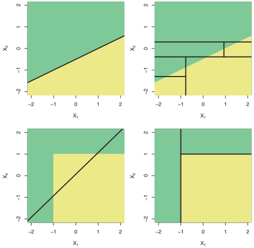


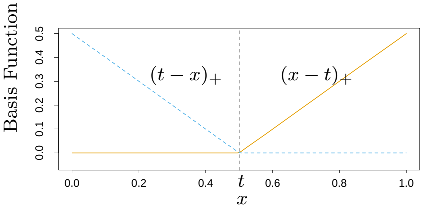
**FIGURE 8.7.** Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a de-*

# 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.

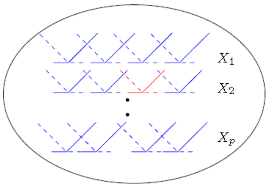


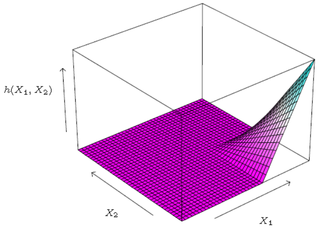
# MARS – continuation

- 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 $\beta_m$ by usual linear regression (minimizing RSS)
- the set of functions $h_m$ is selected iteratively.



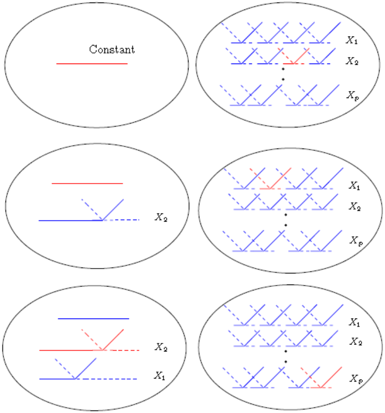



# MARS – basis selections

- 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

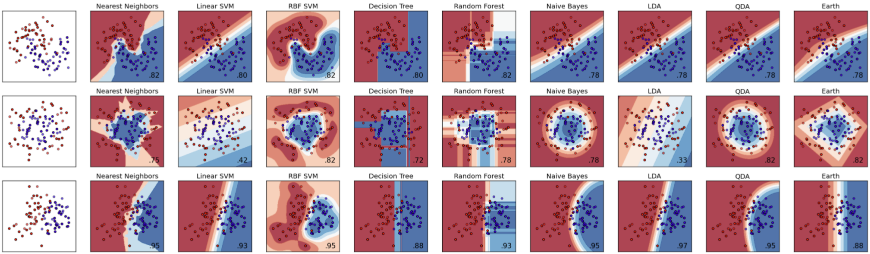
- The model is usually overfitted. We select (remove) iteratively the one minimizing the increase of training RSS. We have a sequence of models $\hat{f}_\lambda$ for different numbers of parameters $\lambda$.
- (we want to speed–up cross-validation for computational reasons)
- we select $\lambda$ (and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

# Table of Contens

1. Overview of Supervised Learning
2. Kernel Methods, Basis Expansion and regularization
3. Linear Methods for Classification
4. Model Assessment and Selection
5. Additive Models, Trees, and Related Methods
6. Ensamble Methods
7. Clustering
8. Bayesian learning, EM algorithm
9. Association Rules, Apriori
10. Inductive Logic Programming
11. Undirected (Pairwise Continuous) Graphical Models
12. Gaussian Processes
13. PCA Extensions, Independent CA
14. Support Vector Machines