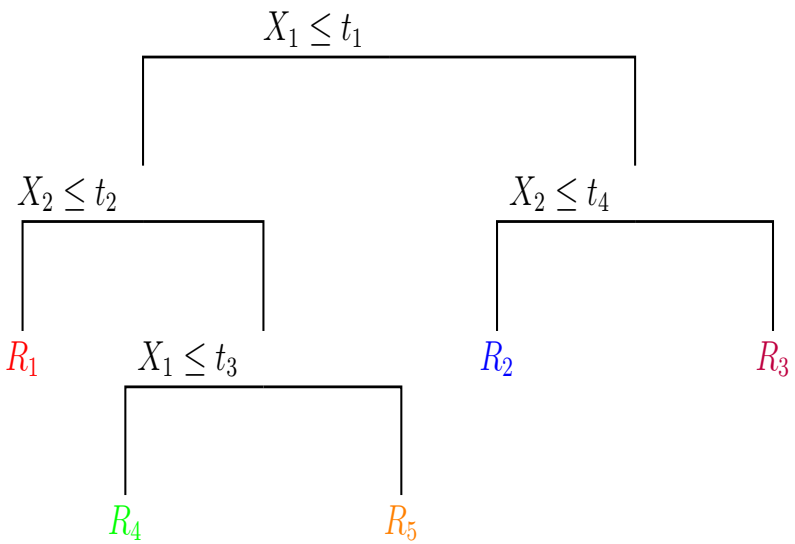
STATS 235: Modern Data Analysis
Classification and Regression Trees

Babak Shahbaba

Department of Statistics, UCI

Tree Models

- Decision trees are models that recursively partition the input space into regions and define a local map between each region and the response variable.



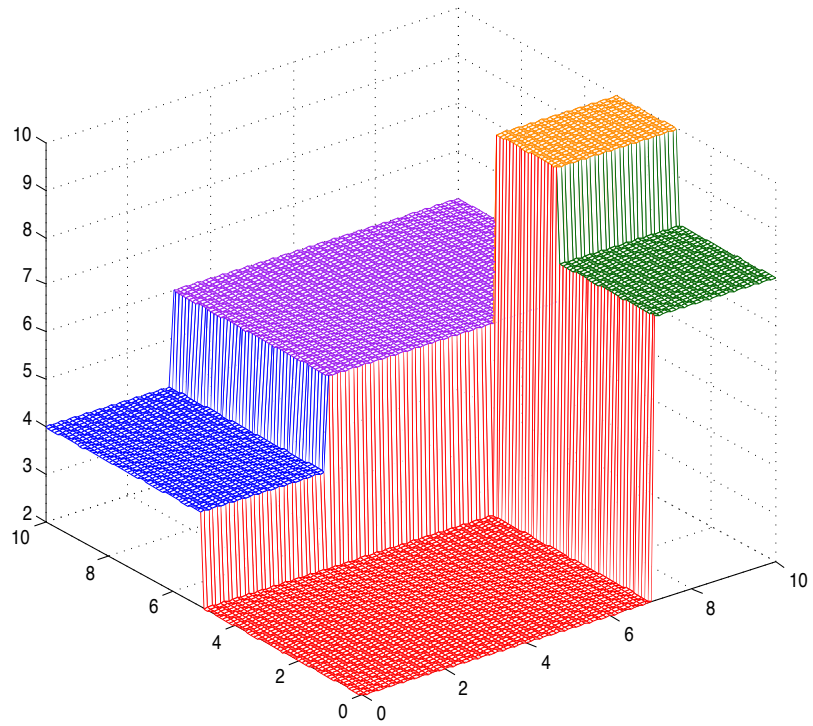


Figure 16.1 in Murphy (2012).

Tree Models

- Starting from the root, when a rule is satisfied, we move to the left branch; otherwise, we move to the right branch.
- When we reach a leaf, we use the subset of data, which fall in the corresponding region, to estimate the response variable.
- The corresponding model can be presented as follows:

$$\hat{y}_i = \sum_{m=1}^{M} \hat{y}_m I(x_i \in R_m)$$

Tree Models

- For regression model, $\hat{y}_m$ can be simply the mean response in region R_m, or the regression estimate using the subset in R_m.
- For classification models, we use the sample proportions, p_{mc}, within region R_m as the estimate for the probability of class c.
- We usually assign a case to the class with the highest probability.

CART

- The most commonly used decision tree method is Classification and Regression Tree (CART).
- To build a CART model, we first *grow* a tree using recursive *binary* splits.
- We usually stop the procedure when some stopping criterion is met. For example: the leaves must have at least m observations.
- The resulting model is typically too complex.
- Next, we *prune* the tree to obtain a simpler model and to avoid overfitting.

Growing a Tree

- To grow a tree,
  - we choose a *cost* function, which typically reflects "impurity"
  - at each node (starting with the whole data at the root), we find the best input variable (feature), j^*, to split the data,
  - and find the best cutoff, t^* for the split.
- For numerical variables, the best (j^*, t^*) is defined as follows:

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,p\}} \min_t \{\text{cost}(x_{ij}, y_i : x_{ij} \leq t) + \text{cost}(x_{ij}, y_i : x_{ij} > t)\}$$

- When x_j is categorical with K categories, the splits are usually based on one group versus all other groups: $x_{ij} = k$ vs. $x_{ij} \neq k$.

Cost Functions

- For regression trees, the square error cost function is commonly used,

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \hat{y}_{\mathcal{D}})^2$$

- $\hat{y}_{\mathcal{D}}$ is the estimate of the response variable (e.g., mean or regression estimate) using the observations in $\mathcal{D}$.

Cost Functions

- For classification trees, commonly used cost functions are
  - misclassification rate

$$\text{cost}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} (y_i \neq \hat{y}_{\mathcal{D}})$$

  - Entropy

$$\text{cost}(\mathcal{D}) = \sum_{\mathcal{D}; c=1}^{C} p_c \log \frac{1}{p_c}$$

  - Gini index

$$\text{cost}(\mathcal{D}) = \sum_{\mathcal{D}; c=1}^{C} p_c (1 - p_c)$$

- p_c is the sample proportion of class c in the subset $\mathcal{D}$.

Cost Functions

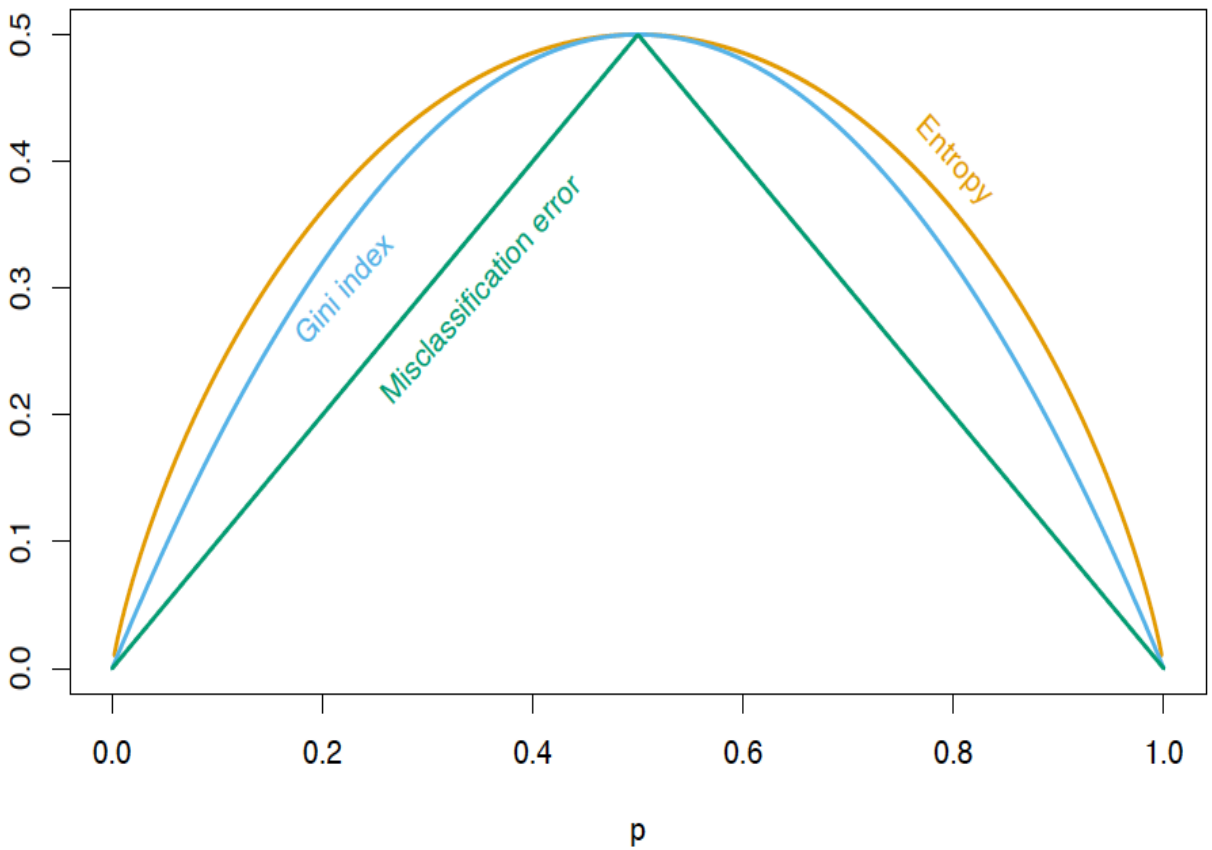


Figure 9.3 in Hastie et al. (2010) for binary classification.

Pruning

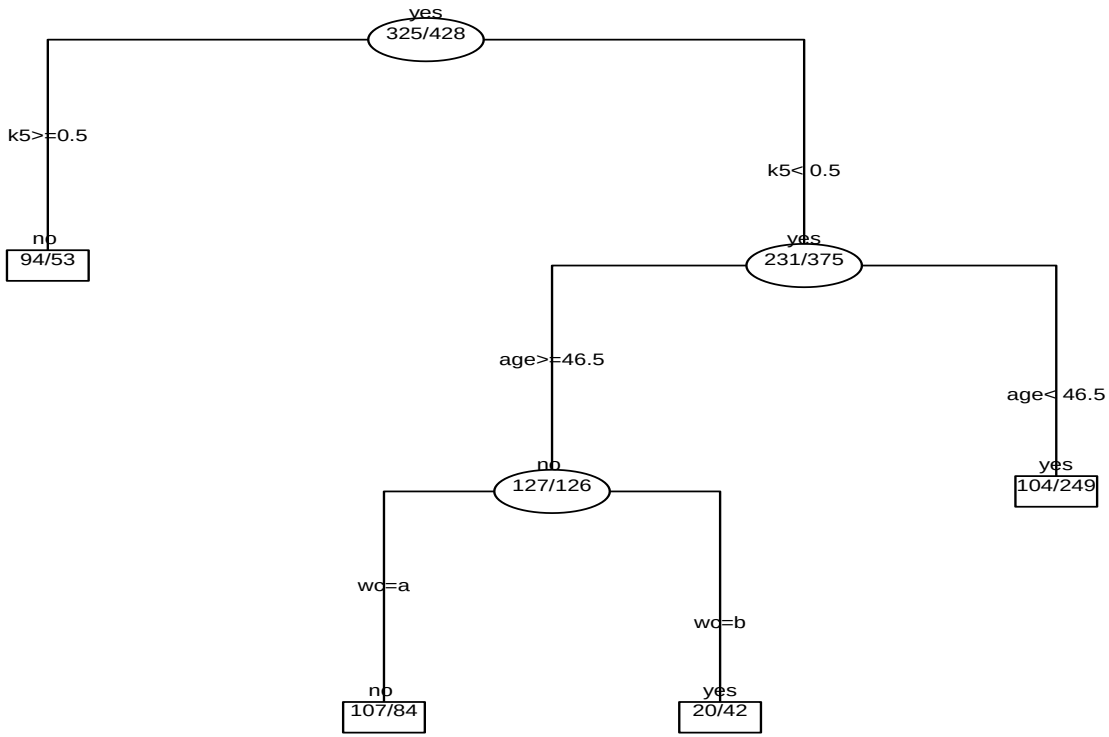
- After growing a full tree, we prune it back to avoid overfitting.
- Pruning involves collapsing some internal nodes to find a subtree without increasing the overall cost substantially.
- We can find the cross-validation costs for all possible subtrees, T, and choose the subtree with the lowest *cost-complexity* value defined as

$$C_\alpha(T) = \sum_{m=1}^{|T|} n_m C_m + \alpha |T|$$

where $|T|$ is the number of terminal nodes for the subtree; n_m and C_m are the number of observations and cost for the mth terminal node; $\alpha > 0$ is a tuning parameter.

US Women's labor-force participation

- The dataset `Mroz` in the package `car` includes the work status of 753 women along with 7 other variables
- We obtain the following model for predicting US women's work status after growing and pruning a tree model



Pros and Cons of Trees

- Prose
  - Easy to interpret
  - Perform automatic variable selection
  - Automatically captures interactions and nonlinear relationships
  - Robust to outliers
- Cons
  - Low predictive power
  - High variability

Bagging

- To reduce the variance of prediction, $\hat{y} = \hat{f}(x)$, we could take a collection of bootstrap samples, build a separate model based on each sample, and aggregate predictions from all models into a single prediction with a similar bias but lower variance

- Given a training, $z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, we generate a collection of B bootstrap samples,

$$z^{(b)} = \{(x_1^*, y_1^*), \ldots, (x_n^*, y_n^*)\}, \quad b = 1, \ldots, B$$

- The overall prediction for the input x is the average over all bootstrap samples,

$$\hat{f}_B(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{(b)}(x)$$

where $\hat{f}^{(b)}(x)$ is the prediction based on the model built on the b^{th} bootstrap sample

Bagging

- This approach is called *bootstrap aggregation* or *bagging*
- Bagging could improve models with low bias and high variance, such as trees
- For regression trees, we simply average over bootstrap trees
- for classification trees, we can either average over class probabilities from bootstrap trees, or if we are just interested in prediction, we can predict the class with the majority "vote"
- The latter resembles the "Wisdom of Crowds" idea (Surowiecki, 2004): making decisions based on the consensus of *independent* and *weakly-informed* voters (here, individual trees)
- Bagged trees, however, are not independent; *random forests* models are designed to address this issue

Random Forests

- *Random forests* models (Breiman, 2001) attempt to reduce variance and improve predictive power by using *bagged* and *de-correlated* (decoupled) trees
- A random forests model is an ensemble of trees, each developed based on a random subset of input variables (features) and a bootstrap sample of observations.
- For regression, we average over the estimates from individual trees.
- For classification, each tree casts a vote, and we choose the class with the highest vote.

Random Forests

for b $= 1$ to B **do**

Bootstrap: Sample n cases with replacement from the data.

Decouple: Use the boot strap sample to grow a random-forest tree T_b, but before each split, sample $m < p$ features randomly (typically $m = \sqrt{p}$) and then find the best feature/cutoff among the m variables.

end for

Output an ensemble of trees $\{T_1, \ldots, T_B\}$.

For regression, average the estimates from individual trees.

For classification, use the majority rule.

Out-of-bag samples

- An advantage of random forests models is their use of out-of-bag (OOB) samples to provide a fast assessment of model performance
- To this end, the response value for the i^{th} case in the training set is estimated by averaging over all bootstrap trees that did not include the i^{th} case in their bootstrap sample
- The resulting prediction accuracy is very similar to that of leave-one-out procedure (without fitting the model n times)

Boosting

- The idea behind bagging and random forests is that an ensemble of *weak learners* can collectively perform well
- *Boosting* extends this idea by sequentially applying a base learner (e.g., a classification tree) to weighted versions of data
- The base learners are *adaptive basis-functions models*, such as GAM, trees, and random forests, with the following general form

$$f(x) = \sum_{m=1}^{M} \beta_m \phi_m(x)$$

where the basis functions, $\varphi(x)$, are learned from the data

- Tree models, for example, can be written as follows:

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

where w_m is the mean of the response variable over the region R_m

Boosting

- For classification trees, we start by applying the weak learner (a tree model performing slightly better than random guessing) to the training set, giving all observations equal weights: $1/n$
- Next, we reapply the weak learner to the data, but this time we give higher weights to cases that were misclassified previously
- We continue this procedure for M iterations, where at each iteration we increase the weights for cases that were misclassified previously and decrease weights for those that were correctly classified

Boosting

- This process results in a sequence of M classifiers: $G_1(x), \ldots, G_M(x)$
- For a binary classification problem with the response variable $y \in \{-1, 1\}$, we combine the individual votes using a weighted majority vote as follows:

$$G(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$$

- The weights $\alpha_1, \ldots, \alpha_M$, are chosen such that more accurate classifiers have higher weights
- The following plot illustrates this approach

Boosting

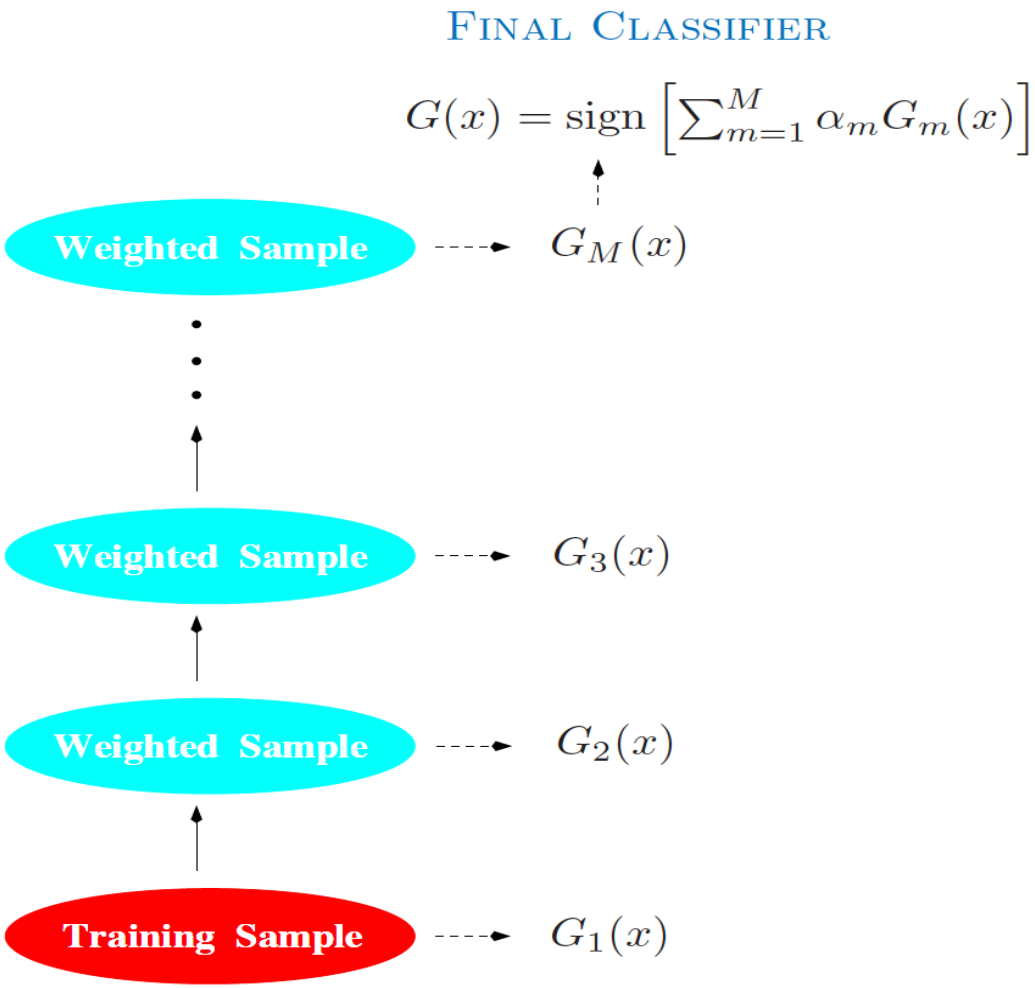


Figure 10.1 in Hastie et al. (2010).

Boosting

- Boosting algorithms attempt to minimize the loss function

$$L(y, f(x))$$

- For binary classification, we could use the 0-1 loss function, $L = I(\text{sign}(f) \neq y)$; however, this is not differentiable

- Alternatively, we could use exponential loss, $L = \exp(-yf)$, or logloss, $L = \log(1 + \exp(-yf))$

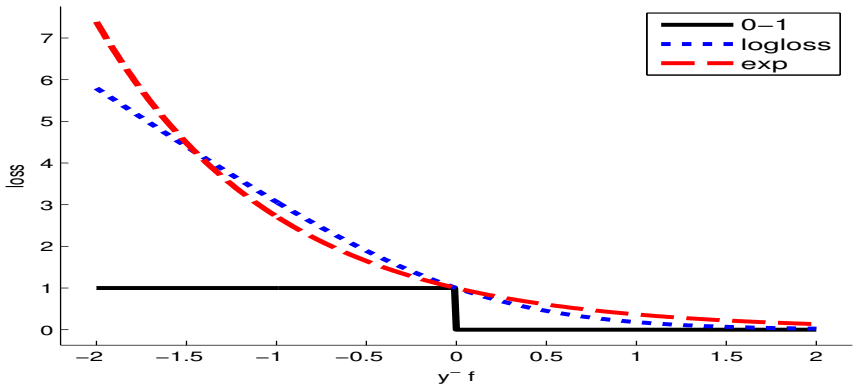


Figure 16.9 in Murphy (2012).

- Using exponential loss leads to one of the most common boosting algorithms called AdaBoost (see Hastie et al. (2010) for more details)

AdaBoost

Initiale the observations weights, $w_i = 1/n$, $i = 1, \ldots, n$

for m $= 1$ to M **do**

Fit a base learner G_m (e.g., tree) to the weighted training data

Find the overall error

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

Calculate the model weights: $\alpha_m = \log((1 - err_m)/err_m)$

Update the observations weights: $w_i \leftarrow w_i \exp(\alpha_m I(y_i \neq G_m(x_i)))$

end for

Output $G(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$

Forward stagewise additive modeling (optional)

- To understand the above algorithm, we note that boosting can be regarded as an additive basis expansion method
- For classification models, each $G_m(x) \in \{-1, 1\}$ is a basis
- In general, we can write them as

$$\begin{aligned} f(x) &= \sum_{m=1}^{M} \beta_m \phi_m(x) \\ &= \sum_{m=1}^{M} \beta_m \phi(x, \gamma_m) \end{aligned}$$

- We estimate $f(x)$ by minimizing a loss function

$$\min \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} \beta_m \phi(x_i, \gamma_m))$$

Forward stagewise additive modeling

- In general, this optimization is computationally intensive; therefore, it is common to find an approximate solution by by sequentially adding a basis function, while keeping the pervious ones (and their corresponding parameters) fixed

- This strategy is known as *forward stagewise additive modeling*:
  1. Initialize $f_0(x)$
  2. For $m = 1, \ldots, M$
     a) Find $(\beta_m, \gamma_m) = \operatorname{argmin} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \beta\phi(x_i, \gamma))$
     b) Set $f_m(x) = f_{m-1}(x) + \beta_m\phi(x_i, \gamma_m)$

Forward stagewise additive modeling

- For AdaBoost, we have

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

and

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

Therefore, optimization is in terms of β and $G_m(x)$ with respect to the exponential loss function

$$\begin{aligned}(\beta_m, G_m) &= \operatorname{argmin} \sum_{i=1}^{n} \exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))] \\ &= \operatorname{argmin} \sum_{i=1}^{n} w_i^{(m)} \exp[-\beta y_i G(x_i))]\end{aligned}$$

where $w_i^{(m)} = \exp(y_i f_{m-1}(x_i))$, i.e., observation weight, which only depends on the previous model, not the optimization parameters

Forward stagewise additive modeling

- We can write the objective function as

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

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

Therefore, for any value of $\beta > 0$, optimization with respect to $G_m(x)$ involves finding a classifier that minimizes the weighted error

$$G_m = \operatorname{argmin} \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(x_i))$$

Forward stagewise additive modeling

- Given G_m, we solve the optimization problem with respect to β,

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

where

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

Using (β_m, G_m), we set

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

Forward stagewise additive modeling

- Using (β_m, G_m), we set

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

Therefore, the next set of observation weights are

$$w_i^{(m+1)} = w_i^{(m)} \exp[-\beta_m y_i G_m(x_i)]$$

Because $-y_i G_m(x_i) = 2I(y_i \neq G_m(x_i)) - 1$, we can rewrite the weights as

$$w_i^{(m+1)} = w_i^{(m)} \exp[\alpha_m I(y_i \neq G_m(x_i))]$$

where

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