BOOSTING

The Elements of Statistical Learning, Chapter 10

¿How does it work?

- Boosting builds a single "strong" classifier by iteratively adding multiple weak classifiers.
- The data weight change after each iteration is the key, i.e., before fitting each new weak classifier.

- In Bagging, each “strong” tree is fitted to a different sample and averaged: Variance Decreasing.

- In Boosting a sequence of weak classifiers (low variance) from all samples are added: Bias Decreasing

Boosting

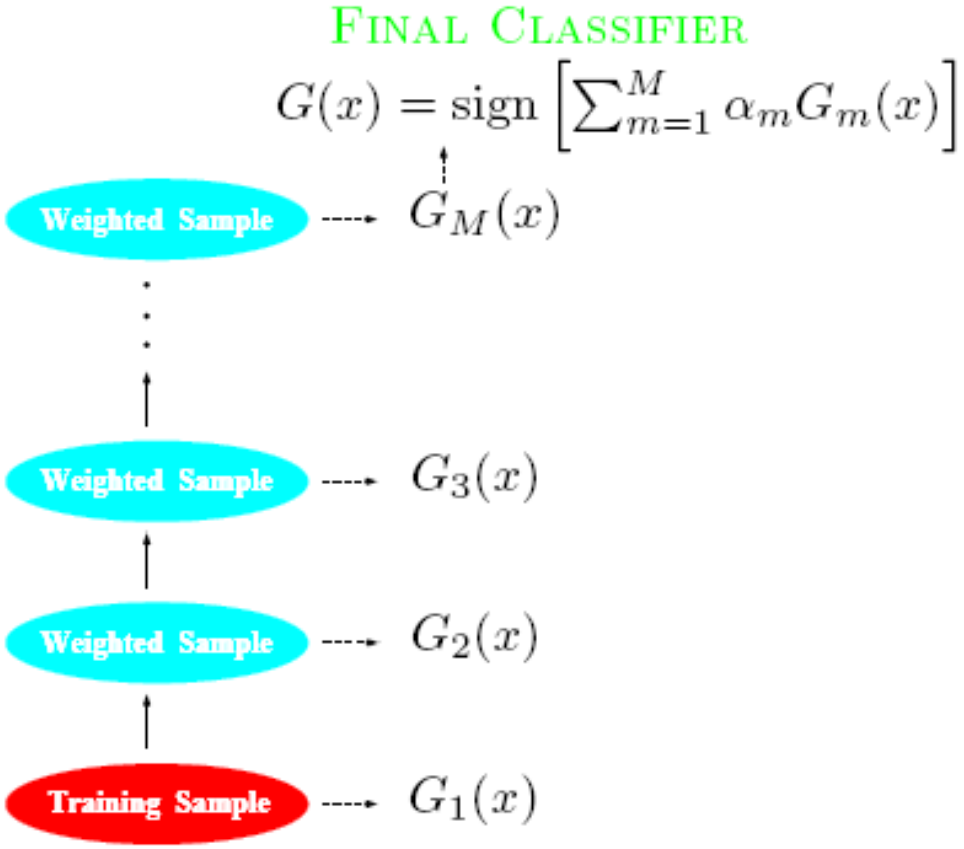


Figure 10.1: *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Ada Boost.M1

- The most popular boosting algorithm – Fruend and Schapire (1997)
- Consider a two-class problem, output variable coded as $Y \in \{-1,+1\}$
- For a predictor variable X, a weak classifier G(X) produces predictions that are in {-1,+1}
- The error rate on the training sample is

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

Ada Boost.M1 (Cont'd)

- Sequentially apply the weak classification to repeatedly modified versions of data
- → produce a sequence of weak classifiers $G_m(x)$ m=1,2,..,M
- The predictions from all classifiers are combined via majority vote to produce the final prediction

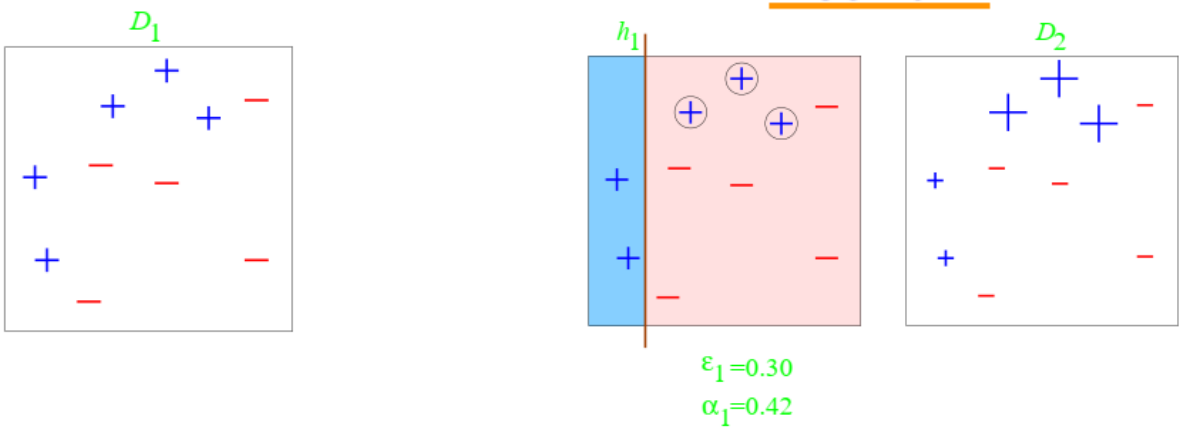
FINAL CLASSIFIER

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

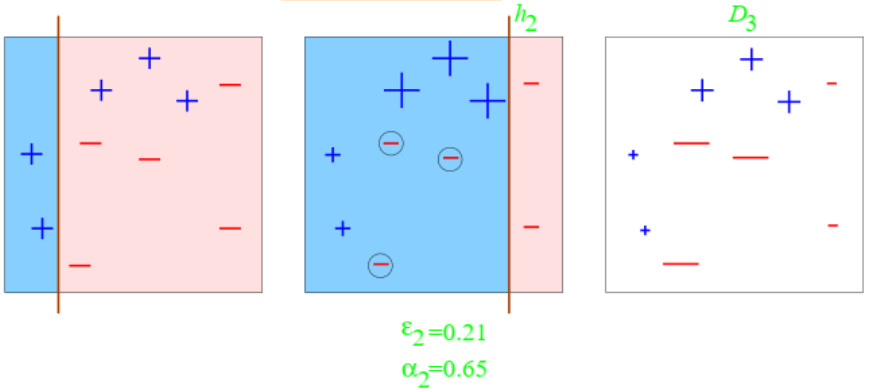
Toy example

weak classifiers = vertical or horizontal half-planes

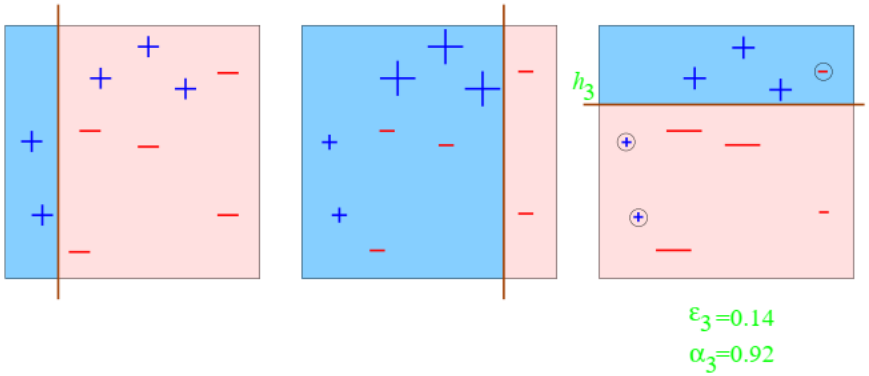
Round 1



Round 2



Round 3

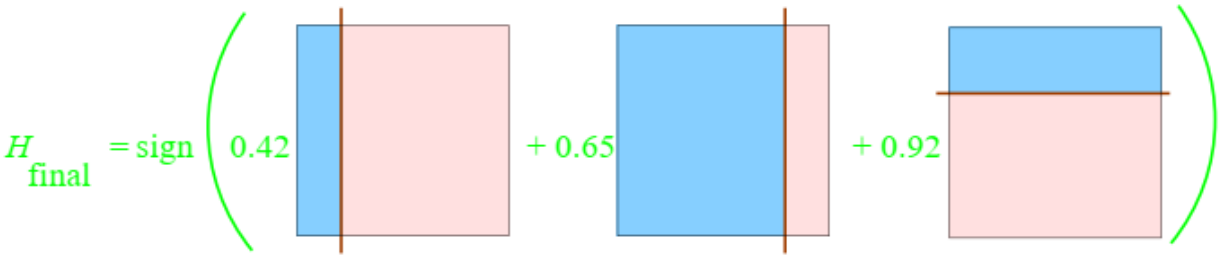


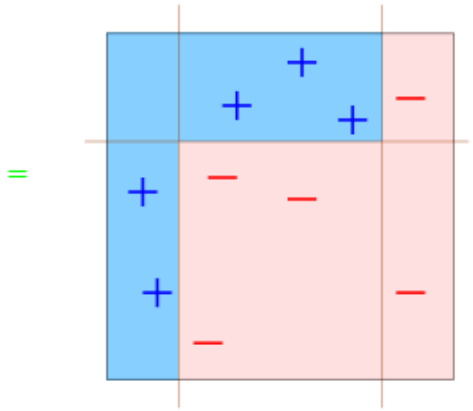
Borrowed from R. Shapire

Toy example

weak classifiers = vertical or horizontal half-planes

Final Classifier





Borrowed from R. Shapire

Algorithm AdaBoost.M1

1. Initialize the observ. weights $w_i^{(1)} = 1/N, i = 1, \ldots, N$.
2. For $m = 1$ to M
   - Fit classifier $G_m(x)$ to the training data using weights $w_i^{(m)}$.
   - Compute $err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$.
   - Compute $\alpha_m = \log((1 - err_m)/err_m)$.
   - $w_i^{(m+1)} = w_i^{(m)} \exp[\alpha_m I(y_i \neq G_m(x_i))], i = 1, \ldots, N$.
3. Compute $G(x) = sign(\sum_{i=1}^{M} \alpha_m G_m(x))$.

Some slides borrowed from http://www.stat.ucl.ac.be/

Example: Adaboost.M1 (Cont'd)

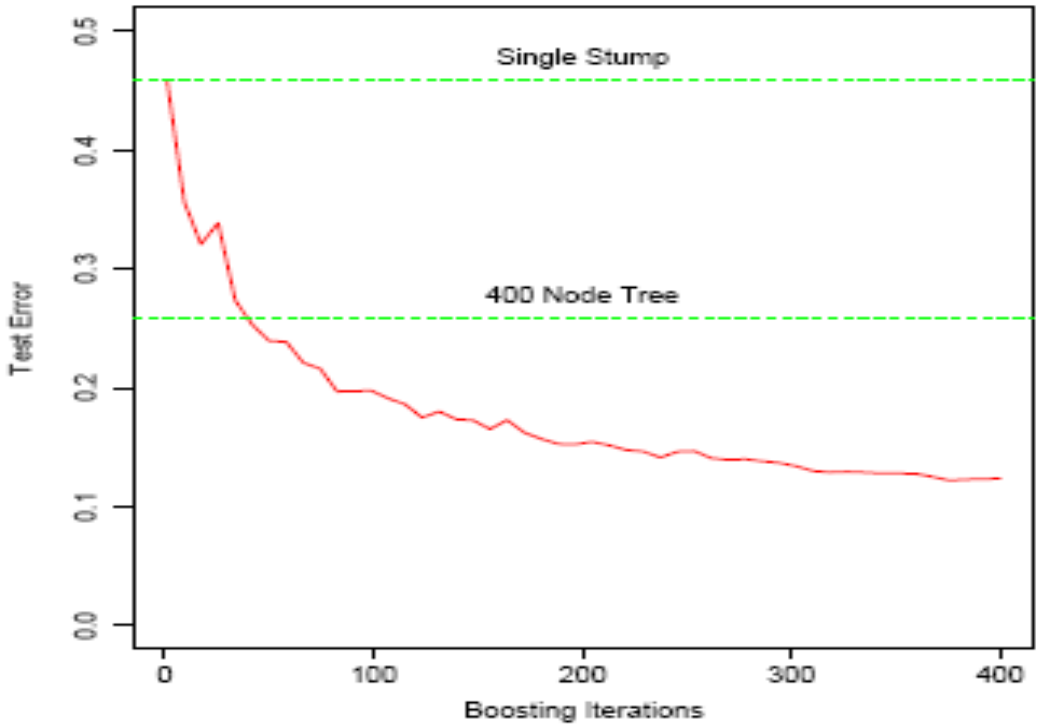


Figure 10.2: *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a* 400 *node classification tree.*

Boosting Fits an Additive Model

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

where $b(x; \gamma_m) = G_m(x) \in \{-1, 1\}$ (for Adaboost) is like a set of elementary "basis functions"

This model is fit by minimizing a loss function L averaged over the training data set:

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

Source http://www.stat.ucl.ac.be/

Forward Stagewise Additive Modeling

- An approximate solution to the minimization problem is obtained via forward stagewise additive modeling (greedy algorithm)

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M
   - Compute
     $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$
   - Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Source http://www.stat.ucl.ac.be/

Why adaBoost Works?

- Adaboost is a forward stagewise additive algorithm using the loss function

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

with

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

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

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

Source http://www.stat.ucl.ac.be/

Why Boosting Works? (Cont'd)

The solution is :

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

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

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

Source http://www.stat.ucl.ac.be/

Practical Advantages of AdaBoost

- fast
- simple and easy to program
- no parameters to tune (except T)
- flexible — can combine with any learning algorithm
- no prior knowledge needed about weak learner
- provably effective, provided can consistently find rough rules of thumb

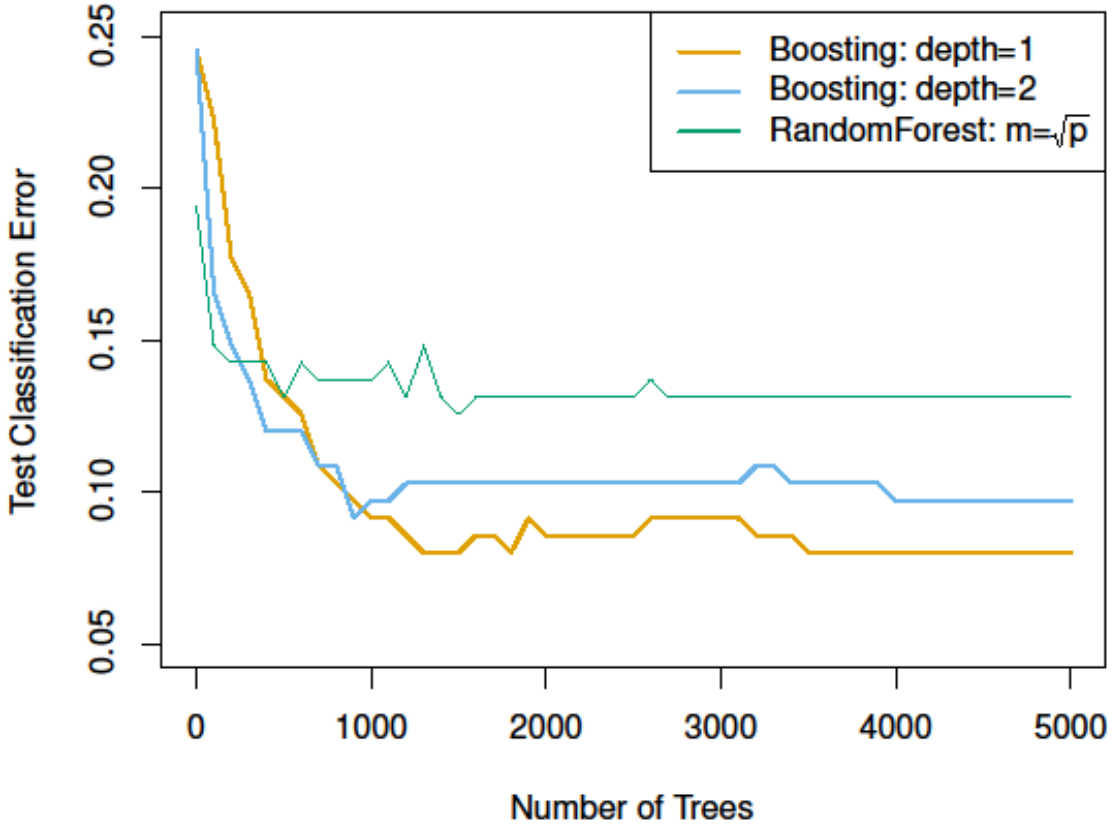
  → shift in mind set — goal now is merely to find classifiers barely better than random guessing
- versatile
  - can use with data that is textual, numeric, discrete, etc.
  - has been extended to learning problems well beyond binary classification

Caveats

- performance of AdaBoost depends on data and weak learner
- consistent with theory, AdaBoost can fail if
  - weak classifiers too complex → overfitting
- weak classifiers too weak ($\gamma_t \rightarrow 0$ too quickly)
  - → underfitting
  - → low margins → overfitting
- empirically, AdaBoost seems especially susceptible to uniform noise

Much more info in the tutorials (PRADO)

RF vs Boosting



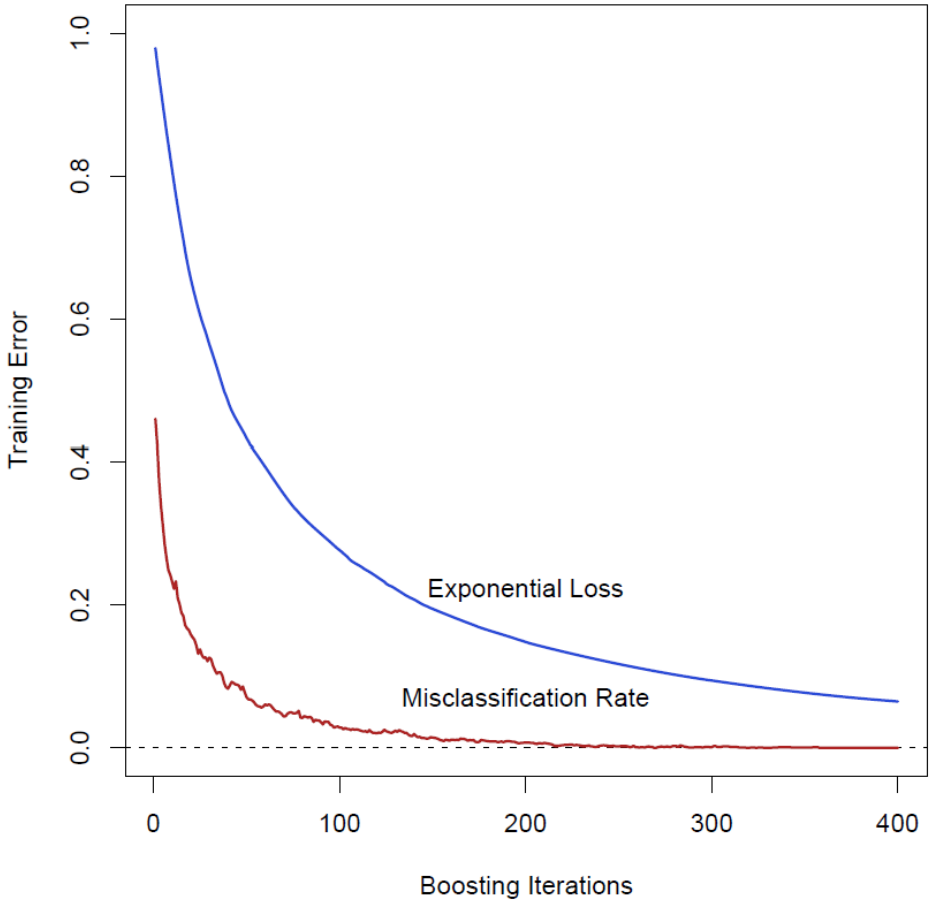
15 –classes dataset

¿What does minimize Adaboost?

- Exponential Loss:

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

- The attached image shows how Adaboost does NOT minimise the global "Training Error" criterion.
- The function that minimises "Exponential Loss" keeps decreasing after the training error is zero.
- The test error also keeps decreasing.
- AdaBoost has an RL model as a probabilistic equivalent.



Loss Functions for Two-Class Classification

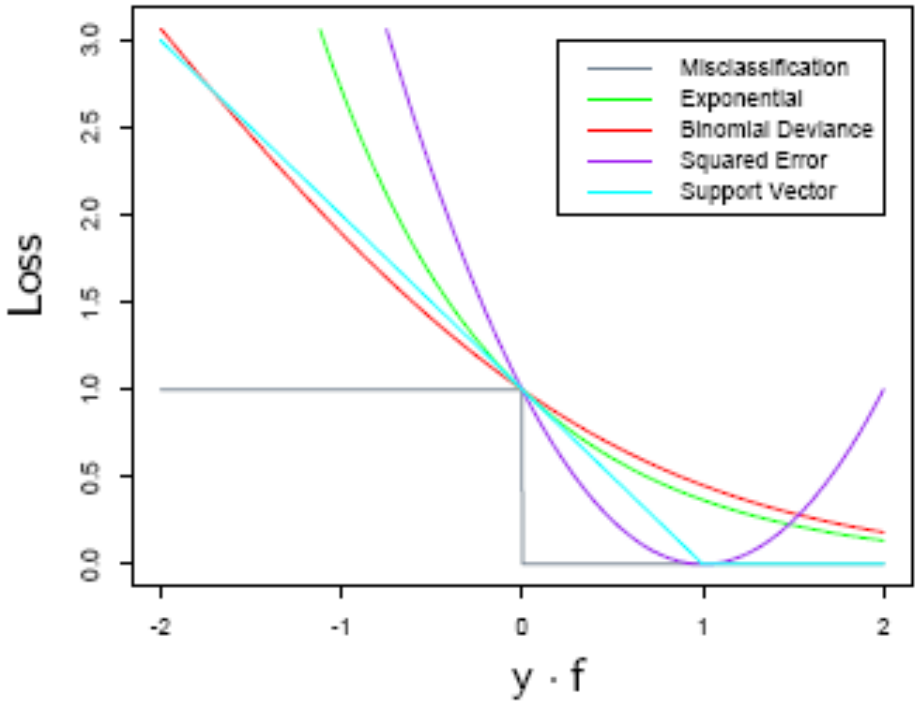


Figure 10.4: *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:*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

Loss Function (Cont'd)

- $yf(x)$ is called the Margin
- The classification rule implies that observations with positive margin $y_i f(x_i) > 0$ were classified correctly, but the negative margin ones are incorrect
- The decision boundary is given by the $f(x) = 0$
- The loss criterion should penalize the negative margins more heavily than the positive ones

- **ADABOOST is a margin maximizing classifier**

Loss Functions (Cont'd)

C. Loss functions for regression

Classification:	exponential loss	$\leftrightarrow$	deviance
Regression :	squared error loss	$\leftrightarrow$	absolute loss
$L(y, f(x))$:	$(y - f(x))^2$	$\leftrightarrow$	$\lvert y - f(x)\rvert$
Minimizer :	$E(Y\lvert x)$	$\leftrightarrow$	Median $(Y\lvert x)$

They are identical for symmetric distribution but lack of robustness for squared error loss.

Compromise between robustness and efficiency : Huber

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

Source http://www.stat.ucl.ac.be/

Gradient Boosting

Algorithm 1: Gradient_Boost

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For $m = 1$ to M do:
3. $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5. $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6. $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7. endFor

end Algorithm

Fast approximate algorithms to parameter estimation. A numerical approach similar to Gradient Descent, But using basic functions instead of derivatives.

J.H. Friedman, Greedy Function Approximation: A Greedy Boosting Machine (1999)

Boosting Trees

- Trees have shown to be the most succesful stump functions

$$f_M(x) = \sum_{m=1}^{M} T(x, \Theta_m)$$

- Forward Stagewise algorithm:
- 1. Initialize $f_0(\mathrm{x}) = 0$
- 2. For $m = 1$ *to* M
  - **Compute**
    - $\widehat{\boldsymbol{\Theta}}_{\boldsymbol{m}} = \mathrm{argmin}_{\Theta_m} \sum_{i=1}^{M} L(y_i, f_{m-1}(x_i) + \mathrm{T}(x_i, \Theta_m), \quad \Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$

- Given the regions R_{jm} finding the optimal constants γ_{jm} in each region is

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_j \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

- Finding the regions is in general difficult except in few special cases of Loss Function.

Simple Regression

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Regression parameters

- Three parameters have to be set:
  - The number of trees B (if it is too large it could over-fit). It can be estimated by cross-validation.
  - The value of the damping parameter λ. Values between 0.01 and 0.001 are typical. If the value of λ is very small we may need a very large value of B.
  - The number of tree partitions d which controls the complexity of each of the individual trees. Normally d=1 and we have trees with only one partition (stump functions).
    - In the case d=1 we have a model that only uses 1 variable at each step and therefore fits an additive model (easily interpretable!).
    - The parameter d is called depth-of-interaction since d partitions could involve d different variables.
    - Although d=1 works well in many applications, in general $1 < d < 4$ works well in the context of boosting.

Gradient Tree Boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Improvement to adapt weigths to each tree región (J.H. Friedman)

Stochastic Gradient Tree Boosting

	Algorithm 2: Stochastic Gradient_TreeBoost
1	$F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} \Psi\left(y_i, \gamma\right)$
2	For $m = 1$ to M do:
3	$\{\pi(i)\}_1^N = \text{rand\_perm}\ \{i\}_1^N$
4	$\tilde{y}_{\pi(i)m} = -\left[\frac{\partial \Psi(y_{\pi(i)}, F(\mathbf{x}_{\pi(i)}))}{\partial F(\mathbf{x}_{\pi(i)})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \ i = 1, \tilde{N}$
5	$\{R_{lm}\}_1^L = L - \text{terminal node}\ tree(\{\tilde{y}_{\pi(i)m}, \mathbf{x}_{\pi(i)}\}_1^{\tilde{N}})$
6	$\gamma_{lm} = \arg\min_\gamma \sum_{\mathbf{x}_{\pi(i)} \in R_{lm}} \Psi\left(y_{\pi(i)}, F_{m-1}(\mathbf{x}_{\pi(i)}) + \gamma\right)$
7	$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
8	endFor

Bagging is used in the fitting function process, see 3 and 4 ($\tilde{N} < N$).
(J.H. Friedman)