Combining Classifiers

- **Goal:** generate a set of simple “weak” classification methods and combine them into a single “strong” method
- **Solution:** Average multiple classifiers by estimate of their reliability. Combine the discriminant functions additively so that the final classifier is the sign of

\[ \hat{g}_n(x) = \alpha_1 h_1(x; w_1) + \cdots + \alpha_m h_m(x; w_m) \]

where the “votes” \( \alpha_i \) emphasize component classifiers that make more reliable predictions than others.
Adaboost

**Final Classifier**

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

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

1. Initialize weight
   1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For each round
   2. For $m = 1$ to $M$:
      a. Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
      b. Compute
         \[
         \text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.
         \]
   3. Compute
      \[
      \alpha_m = \log((1 - \text{err}_m)/\text{err}_m).
      \]
   4. Update distribution/correctly/correct
      (c) Compute
         \[
         \alpha_m = \log((1 - \text{err}_m)/\text{err}_m).
         \]
   5. Compute
      (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.

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

\[
\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)
\]

\[
\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}
\]

\[
w_i \leftarrow w_i \cdot \exp[\alpha_m I(y_i \neq G_m(x_i))] \quad i = 1, 2, \ldots, N.
\]
Adaboost Algorithm

Algorithm 1 (AdaBoost)

1. \textbf{begin initialize} \( D = \{x^1, y_1, x^2, y_2, \ldots, x^n, y_n\}, k_{\text{max}}, W_1(i) = 1/n, i = 1, \ldots, n \)
2. \( k \leftarrow 0 \)
3. \textbf{do} \( k \leftarrow k + 1 \)
4. \hspace{1em} Train weak learner \( C_k \) using \( D \) sampled according to distribution \( W_k(i) \)
5. \hspace{1em} \( E_k \leftarrow \) Training error of \( C_k \) measured on \( D \) using \( W_k(i) \)
6. \hspace{1em} \( \alpha_k \leftarrow \frac{1}{2} \ln[(1 - E_k)/E_k] \)
7. \hspace{1em} \( W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \times \begin{cases} e^{-\alpha_k} & \text{if } h_k(x^i) = y_i \text{ (correctly classified)} \\ e^{\alpha_k} & \text{if } h_k(x^i) \neq y_i \text{ (incorrectly classified)} \end{cases} \)
8. \textbf{until} \( k = k_{\text{max}} \)
9. \textbf{return} \( C_k \) and \( \alpha_k \) for \( k = 1 \) to \( k_{\text{max}} \) (ensemble of classifiers with weights)
10. \textbf{end}
Adaboost Example

- The simple classifiers in our case are *decision stumps*:

\[ h(x; \theta) = \text{sign}(w_1 x_k - w_0) \]

where \( \theta = \{k, w_1, w_0\} \).

Each decision stump pays attention to only a single component of the input vector.
Boosting: example
FIGURE 8.11. Data with two features and two classes, separated by a linear boundary. Left panel: decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. Right panel: decision boundary from boosting the decision rule of the same classifier. The test error rates are 0.166, and 0.065 respectively. Boosting is described in Chapter 10.
Adaboost Summary

- Basic AdaBoost: Combine weak classifiers to make a strong classifier.
- Dynamically weight the data, so that misclassified data weighs more (like SVM pay more attention to hard-to-classify data).
- Exponential convergence to empirical risk (weak conditions).
Boosting with near-Chance Classifiers

**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.
Why does it work?

- Changing algorithm changes problem - REVISED Model representation -
  Generalized additive model (defined in subsequent slide) in a base learner,

REVISED Error criterion -
  Novel exponential loss function.
  However: very similar to the (negative) binomial log-likelihood
  Optimality: The population minimizer of the exponential loss function is shown to be the log-odds of the class probabilities

- Step-wise optimization of loss function
Generalized Additive models

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

Where
\( \beta_m, \quad m = 1, 2, \ldots, M \) are the expansion coefficients
\( b(x; \gamma_m) \) are functions of the multivariate argument \( x \), characterized by a set of parameters \( \gamma_m \).
Learning

Given a Loss function $L$ on training labels $y_i$ and predictions $g(x_i)$

Learning the model involves minimizing the loss function

$$
\min_{\{\text{params}\}} \sum_{i=1}^{N} L(y_i, g(x_i; \text{params}))
$$

$$
\min_{\{\beta_m, \gamma_m\}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m))
$$
Forward Stagewise Additive Modeling

Forward stagewise modeling approximates the solution to by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added.

Algorithm 10.2 Forward stagewise additive modeling.

1. Initialize \( f_0(x) = 0 \).

2. For \( m = 1 \) to \( M \):
   
   (a) Compute
   
   \[
   (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).
   \]

   (b) Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).
Example

For squared-error loss

\[ L(y, f(x)) = (y - f(x))^2, \]

one has

\[ L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \]
\[ = (r_{im} - \beta b(x_i; \gamma))^2, \]
Loss function for AdaBoost

\[ L(y, f(x)) = \exp(-y f(x)) \]

Thus we need to solve:

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

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

Using stagewise forward modeling on this loss leads to the AdaBoost algorithm (see posted Hastie chapter)
Optimality

The principal attraction of exponential loss in the context of additive modeling is computational; it leads to the simple modular reweighting AdaBoost algorithm.

BUT

What does it estimate and how well is it being estimated?

It is easy to show (Friedman et al., 2000) that

\[ f^*(x) = \arg \min_{f(x)} E_{Y|x}(e^{-Yf(x)}) = \frac{1}{2} \log \frac{\Pr(Y = 1|x)}{\Pr(Y = -1|x)}, \]

(10.16)

or equivalently

\[ \Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}}. \]

Thus, the additive expansion produced by AdaBoost is estimating one-half the log-odds of \( P(Y = 1|x) \). This justifies using its sign as the classification rule in (10.1).
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: $(1-yf)I(yf > 1)$ (see Section 12.3). Each function has been scaled so that it passes through the point $(0, 1)$.  

CSCI 5521: Pattern Recognition, Prof. Paul Schrater, Fall 2005