Voted combination of classifiers

- The general problem here is to try to combine many simple “weak” classifiers into a single “strong” classifier.
- We consider voted combinations of simple binary ±1 component classifiers

\[ h_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m) \]

where the (non-negative) votes \( \alpha_i \) can be used to emphasize component classifiers that are more reliable than others.

Voted combination cont’d

- We need to define a loss function for the combination so we can determine which new component \( h(x; \theta) \) to add and how many votes it should receive

\[ h_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m) \]

- While there are many options for the loss function we consider here only a simple exponential loss

\[ \exp \{ -y h_m(x) \} \]

Components: decision stumps

- Consider the following simple family of component classifiers generating ±1 labels:

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

where \( \theta = \{k, w_1, w_0\} \). These are called decision stumps.
- Each decision stump pays attention to only a single component of the input vector.

Modularity, errors, and loss

- Consider adding the \( m^{th} \) component:

\[
\sum_{i=1}^n \exp \{ -y_i[h_{m-1}(x_i) + \alpha_m h(x_i; \theta_m)] \}
= \sum_{i=1}^n \exp \{ -y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m) \}
\]
**Modularity, errors, and loss**

- Consider adding the $m^{th}$ component:
  \[
  \sum_{i=1}^{n} \exp\left\{ -y_i[h_{m-1}(x_i) + \alpha_m h(x_i; \theta_m)] \right\}
  = \sum_{i=1}^{n} \exp\left\{ -y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m) \right\}
  = \sum_{i=1}^{n} \exp\left\{ -y_i h_{m-1}(x_i) \right\} \exp\left\{ -y_i \alpha_m h(x_i; \theta_m) \right\}
  \]

**Empirical exponential loss cont’d**

- To increase modularity we’d like to further decouple the optimization of $h(x; \theta_m)$ from the associated votes $\alpha_m$
- To this end we select $h(x; \theta_m)$ that optimizes the rate at which the loss would decrease as a function of $\alpha_m$
  \[
  \frac{\partial}{\partial \alpha_m} \left|_{\alpha_m=0} \right. \sum_{i=1}^{n} W_i^{(m-1)} \exp\left\{ -y_i \alpha_m h(x_i; \theta_m) \right\} = \]

\[
\begin{bmatrix}
\sum_{i=1}^{n} W_i^{(m-1)} \exp\left\{ -y_i \alpha_m h(x_i; \theta_m) \right\} \\
\sum_{i=1}^{n} W_i^{(m-1)} \exp\left\{ -y_i \alpha_m h(x_i; \theta_m) \right\} \cdot \left( -y_i h(x_i; \theta_m) \right)
\end{bmatrix}
\]

So at the $m^{th}$ iteration the new component (and the votes) should optimize a weighted loss (weighted towards mistakes).

**Empirical exponential loss cont’d**

- We find $h(x; \theta_m)$ that minimizes
  \[
  - \sum_{i=1}^{n} W_i^{(m-1)} y_i h(x_i; \theta_m)
  \]
Empirical exponential loss cont’d

• We find \( h(x; \hat{\theta}_m) \) that minimizes
\[
- \sum_{i=1}^{n} W^{(m-1)}_i y_i h(x_i; \theta_m)
\]

We can also normalize the weights:
\[
- \sum_{i=1}^{n} \frac{W^{(m-1)}_i}{\sum_{j=1}^{n} W^{(m-1)}_j} y_i h(x_i; \theta_m) = - \sum_{i=1}^{n} W^{(m-1)}_i y_i h(x_i; \theta_m)
\]
so that \( \sum_{i=1}^{n} \tilde{W}^{(m-1)}_i = 1 \).

Selecting a new component: summary

• We find \( h(x; \hat{\theta}_m) \) that minimizes
\[
- \sum_{i=1}^{n} \tilde{W}^{(m-1)}_i y_i h(x_i; \theta_m)
\]
where \( \sum_{i=1}^{n} \tilde{W}^{(m-1)}_i = 1 \).
• \( \alpha_m \) is subsequently chosen to minimize
\[
\sum_{i=1}^{n} \tilde{W}^{(m-1)}_i \exp(-y_i \alpha_m h(x_i; \hat{\theta}_m))
\]

The AdaBoost algorithm

0) Set \( \tilde{W}^{(0)}_i = 1/n \) for \( i = 1, \ldots, n \)

1) At the \( m^{th} \) iteration we find (any) classifier \( h(x; \hat{\theta}_m) \) for which the weighted classification error \( \epsilon_m \)
\[
\epsilon_m = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}^{(m-1)}_i y_i h(x_i; \hat{\theta}_m) \right)
\]
is better than chance.

2) The new component is assigned votes based on its error:
\[
\hat{\alpha}_m = 0.5 \log \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
\]
which minimizes the weighted loss when \( h(x; \theta) \in \{-1, 1\} \)
\[
\sum_{i=1}^{n} \tilde{W}^{(m-1)}_i \exp(-y_i \hat{\alpha}_m h(x_i; \hat{\theta}_m))
\]

The AdaBoost algorithm

0) Set \( \tilde{W}^{(0)}_i = 1/n \) for \( i = 1, \ldots, n \)

1) At the \( m^{th} \) iteration we find (any) classifier \( h(x; \hat{\theta}_m) \) for which the weighted classification error \( \epsilon_m \)
\[
\epsilon_m = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}^{(m-1)}_i y_i h(x_i; \hat{\theta}_m) \right)
\]
is better than chance.

2) The new component is assigned votes based on its error:
\[
\hat{\alpha}_m = 0.5 \log \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
\]

3) The weights are updated according to (\( Z_m \) is chosen so that the new weights \( \tilde{W}^{(m)}_i \) sum to one):
\[
\tilde{W}^{(m)}_i = \frac{1}{Z_m} \cdot \tilde{W}^{(m-1)}_i \cdot \exp(-y_i \hat{\alpha}_m h(x_i; \hat{\theta}_m))
\]
Adaboost properties: weighted error

- Weighted error of each new component classifier
  \[ \epsilon_k = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(k-1)} y_i h(x_i; \theta_k) \right) \]
  tends to increase as a function of boosting iterations.

Adaboost properties: exponential loss

- After each boosting iteration, assuming we can find a component classifier whose weighted error is better than chance, the combined classifier
  \[ \hat{h}_m(x) = \hat{\alpha}_1 h(x; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(x; \hat{\theta}_m) \]
  is guaranteed to have a lower exponential loss over the training examples.

Adaboost properties: training error

- The boosting iterations also decrease the classification error of the combined classifier
  \[ \hat{h}_m(x) = \hat{\alpha}_1 h(x; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(x; \hat{\theta}_m) \]
  over the training examples.

Adaboost properties: training error cont’d

- The training classification error has to go down exponentially fast if the weighted errors of the component classifiers, \( \epsilon_k \), are strictly better than chance \( \epsilon_k < 0.5 \)
  \[ \text{err}(\hat{h}_m) \leq \prod_{k=1}^{m} 2\sqrt{\epsilon_k (1 - \epsilon_k)} \]

“Typical” performance

- Training and test errors of the combined classifier
  \[ \hat{h}_m(x) = \hat{\alpha}_1 h(x; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(x; \hat{\theta}_m) \]

- Why should the test error go down after we already have zero training error?
AdaBoost and margin
• We can write the combined classifier in a more useful form by dividing the predictions by the "total number of votes":
\[ \hat{h}_m(x) = \frac{\hat{\alpha}_1 h(x; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(x; \hat{\theta}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m} \]
• This allows us to define a clear notion of "voting margin" that the combined classifier achieves for each training example:
\[ \text{margin}(x_i) = y_i \cdot \hat{h}_m(x_i) \]
The margin lies in \([-1, 1]\) and is negative for all misclassified examples.

Can we improve the combination?
• As a result of running the boosting algorithm for \(m\) iterations, we essentially generate a new feature representation for the data
\[ \phi_i(x) = h(x; \hat{\theta}_i), i = 1, \ldots, m \]
• Perhaps we can do better by separately estimating a new set of "votes" for each component. In other words, we could estimate a linear classifier of the form
\[ f(x; \alpha) = \alpha_1 \phi_1(x) + \ldots + \alpha_m \phi_m(x) \]
where each parameter \(\alpha_i\) can be now any real number (even negative). The parameters would be estimated jointly rather than one after the other as in boosting.