Topics

• Combination of classifiers: boosting
  – decision stumps, reweighting, adaboost
  – examples,
  – margin and performance
Review: combination of classifiers

● We can combine multiple “weak” classifiers to produce a single “strong” classifier:

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

where the predicted label for \( x \) is the sign of the discriminant function \( h_m(x) \).

● If each component classifier returns only \( \pm 1 \) it is beneficial to allow some of them to have more “votes” than others:

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

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

● We wish to “modularize” the estimation problem.
Review: decision stumps as weak classifiers

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

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

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.
Review: measure of classification loss

- We need a loss function that permits us to summarize how the already included components in \( h_{m-1}(x) \) affect the training of a new component \( \alpha_m h(x; \theta_m) \).

- The exponential loss function

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

suffices for this purpose. The loss is large when the \( \pm 1 \) label \( y \) disagrees with the sign of the discriminant function

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

- The empirical exponential loss is “modular”:

\[
\sum_{i=1}^{n} \exp\{-y_i h_m(x_i)\} = \sum_{i=1}^{n} \exp\{-y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m)\} = \sum_{i=1}^{n} \exp\{-y_i h_{m-1}(x_i)\} \exp\{-y_i \alpha_m h(x_i; \theta_m)\} = \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(x_i; \theta_m)\}
\]

The combined classifier based on \(m-1\) iterations defines a weighted loss criterion for the next simple classifier to add.
Enforcing stronger modularity

- We can further simplify the estimation criterion for the new component classifiers (not the votes)

**Rationale:** When $\alpha_m \approx 0$ (low confidence votes)

$$\exp\{-y_i\alpha_m h(x_i; \theta_m)\} \approx 1 - y_i\alpha_m h(x_i; \theta_m)$$

and our empirical loss criterion reduces to

$$\approx \sum_{i=1}^{n} W^{(m-1)}_i \left(1 - y_i\alpha_m h(x_i; \theta_m)\right) =$$

$$= \sum_{i=1}^{n} W^{(m-1)}_i - \alpha_m \left(\sum_{i=1}^{n} W^{(m-1)}_i y_i h(x_i; \theta_m)\right)$$

So we can choose each new component classifier to optimize a weighted classification error.
Boosting

- A Boosting algorithm sequentially estimates and combines classifiers by reweighting training examples
  - each component classifier is presented with a slightly different problem depending on the weights
  - the weights focus each new component on harder examples

- To get started we need:
  - a family of “weak” binary ($\pm 1$) classifiers $h(x; \theta)$ such as decision stumps
  - normalized weights $\tilde{W}_i^{(0)}$ on the training examples, initially set to be uniform ($\tilde{W}_i^{(0)} = 1/n$)
The AdaBoost algorithm

1) At the $k^{th}$ iteration we find any classifier $h(x; \hat{\theta}_k)$ for which the weighted classification error $\epsilon_k$

$$\epsilon_k = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(k-1)} y_i h(x_i; \hat{\theta}_k) \right)$$

is better than chance.
The AdaBoost algorithm

1) At the $k^{th}$ iteration we find any classifier $h(x; \hat{\theta}_k)$ for which the weighted classification error $\epsilon_k$

$$\epsilon_k = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(k-1)} y_i h(x_i; \hat{\theta}_k) \right)$$

is better than chance.

2) The new component classifier is assigned “votes” based on its performance

$$\hat{\alpha}_k = 0.5 \log \left( \frac{(1 - \epsilon_k)}{\epsilon_k} \right)$$

where $\hat{\alpha}_k$ minimizes the weighted exponential loss

$$\sum_{i=1}^{n} \tilde{W}_i^{(k-1)} \exp \{ -y_i \alpha_k h(x_i; \hat{\theta}_k) \}$$
The AdaBoost algorithm

1) At the $k^{th}$ iteration we find any classifier $h(x; \hat{\theta}_k)$ for which the weighted classification error $\epsilon_k$

$$\epsilon_k = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(k-1)} y_i h(x_i; \hat{\theta}_k) \right)$$

is better than chance.

2) The new component classifier is assigned “votes” based on its performance

$$\hat{\alpha}_k = 0.5 \log \left( \frac{(1 - \epsilon_k)}{\epsilon_k} \right)$$

3) The weights on the training examples are updated according to ($c$ is chosen so that the new weights $\tilde{W}_i^{(k)}$ sum to one):

$$\tilde{W}_i^{(k)} = c \cdot \tilde{W}_i^{(k-1)} \cdot \exp \{ -y_i \hat{\alpha}_k h(x_i; \hat{\theta}_k) \}$$
Boosting: example
“Typical” performance

- Weighted error of each new component classifier

\[ \epsilon_k = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(k-1)} y_i h(x_i; \hat{\theta}_k) \right) \]
“Typical” performance cont’d

- 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?

Tommi Jaakkola, MIT CSAIL
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.
AdaBoost and margin

- Successive boosting iterations improve the majority vote or margin for the training examples

\[
\text{margin}(x_i) = y_i \left[ \frac{\hat{\alpha}_1 h(x_i; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(x_i; \hat{\theta}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m} \right]
\]
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.
Can we improve the combination?

- We could use SVMs in a postprocessing step to reoptimize

\[ f(x; \alpha) = \alpha_1 \phi_1(x) + \ldots + \alpha_m \phi_m(x) \]

with respect to \( \alpha_1, \ldots, \alpha_m \). This is not necessarily a good idea.

![Graphs](boosting.png)  ![Graphs](svm_postprocessing.png)

**boosting**  **svm postprocessing**
Topics

- Combination of classifiers: boosting
  - decision stumps, reweighting, adaboost
  - examples,
  - margin and performance

- Complexity and model selection
  - learning and VC dimension
  - structural risk minimization