# The AdaBoost algorithm

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

**1)** At the $m^{th}$ iteration we find (any) classifier $h(\mathbf{x}; \hat{\theta}_m)$ for which the *weighted classification error* $\epsilon_m$

$$\epsilon_m = 0.5 - \frac{1}{2}\left( \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i; \hat{\theta}_m) \right)$$

is better than chance.

**2)** The new component is assigned votes based on its error:

$$\hat{\alpha}_m = 0.5 \log(\, (1 - \epsilon_m)/\epsilon_m \,)$$

**3)** The weights are updated according to ($Z_m$ is chosen so that the new weights $\tilde{W}_i^{(m)}$ sum to one):

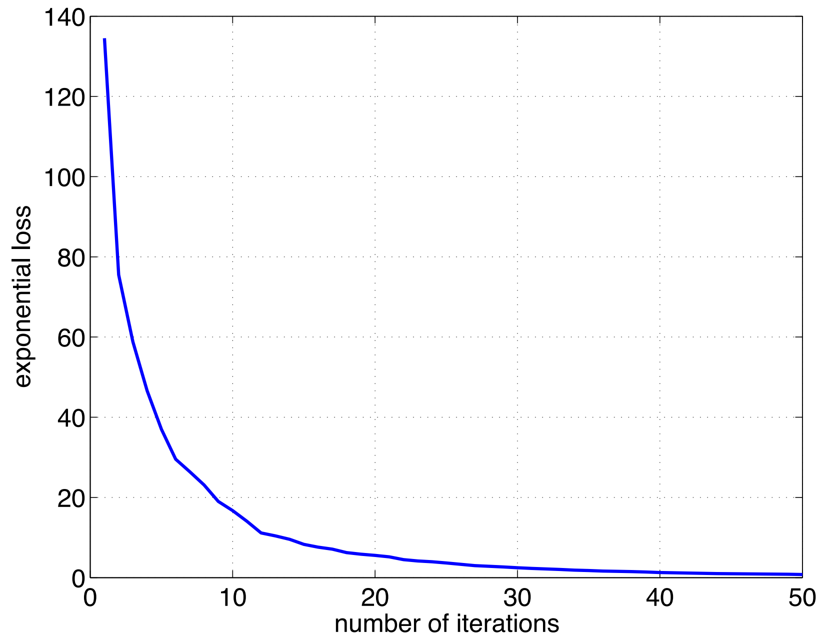
$$\tilde{W}_i^{(m)} = \frac{1}{Z_m} \cdot \tilde{W}_i^{(m-1)} \cdot \exp\{\, -y_i \hat{\alpha}_m h(\mathbf{x}_i; \hat{\theta}_m) \,\}$$

# 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

$$\hat{h}_m(\mathbf{x}) = \hat{\alpha}_1 h(\mathbf{x}; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}; \hat{\theta}_m)$$

is guaranteed to have a lower exponential loss over the training examples

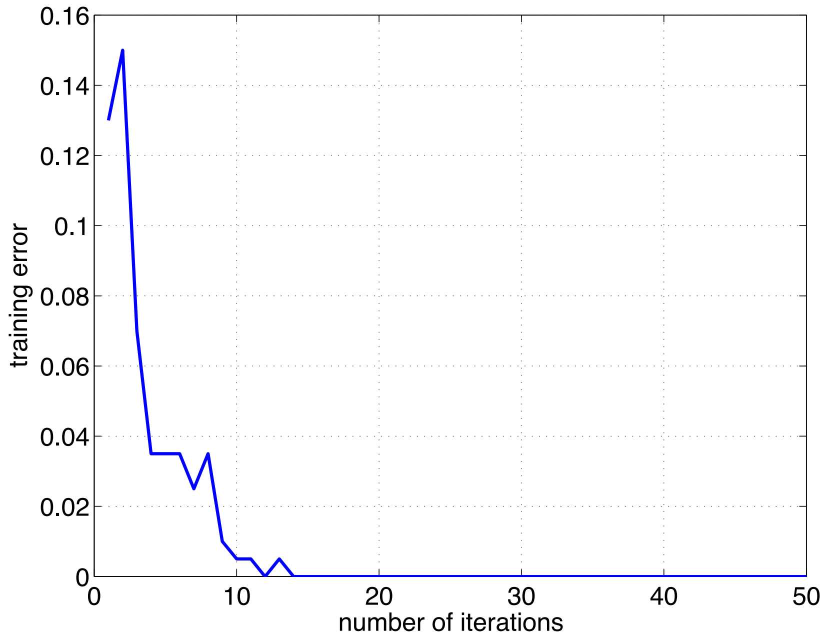


# Adaboost properties: training error

- The boosting iterations also decrease the classification error of the combined classifier

$$\hat{h}_m(\mathbf{x}) = \hat{\alpha}_1 h(\mathbf{x}; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}; \hat{\theta}_m)$$

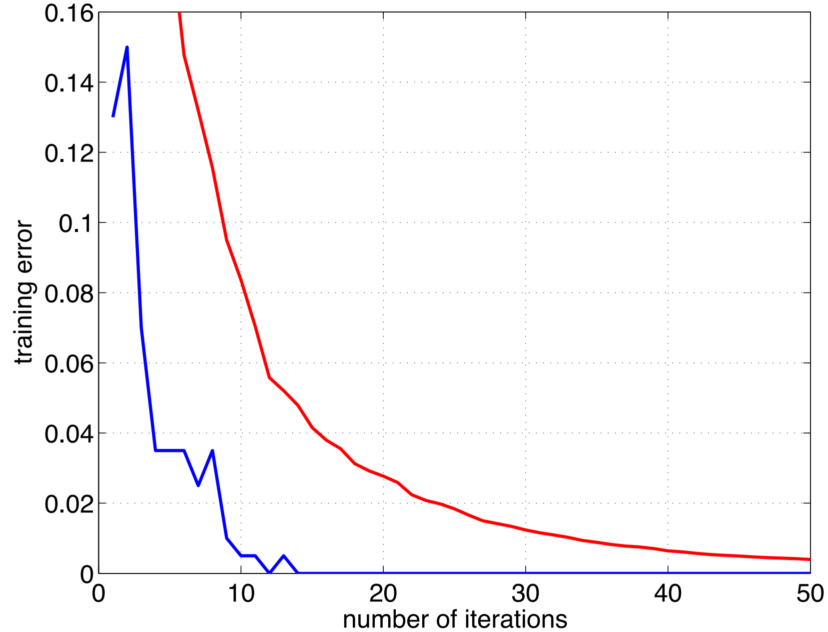
over the training examples.



# 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 $\epsilon_k < 0.5$

$$\text{err}(\hat{h}_m) \leq \prod_{k=1}^{m} 2\sqrt{\epsilon_k(1-\epsilon_k)}$$

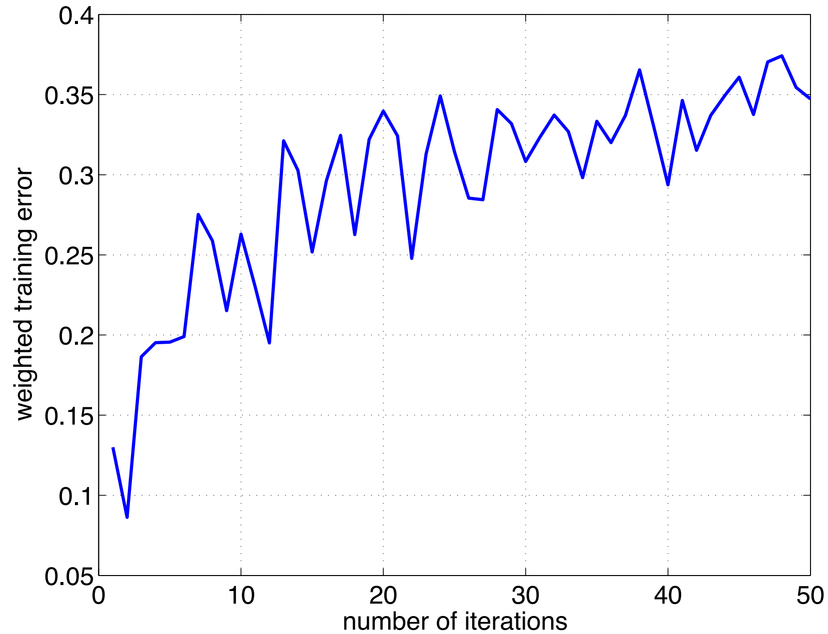


# Adaboost properties: weighted error

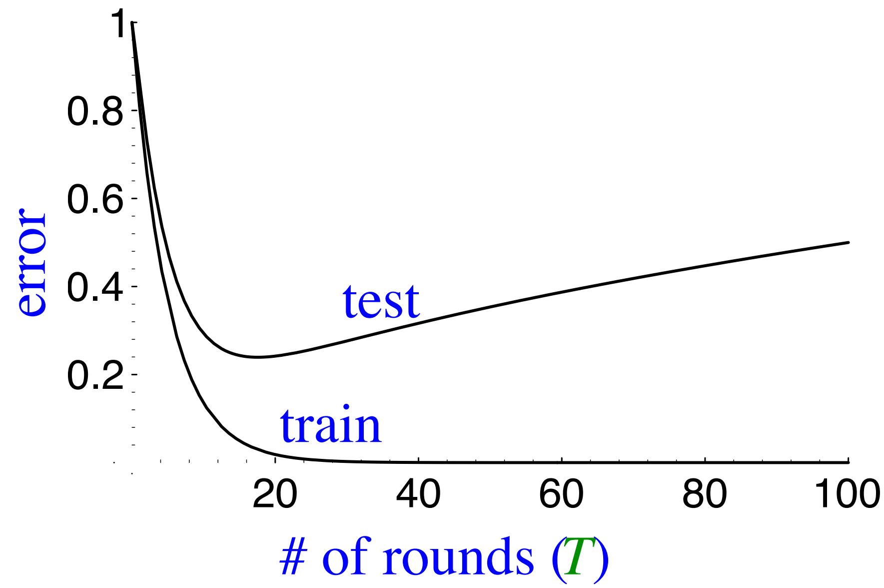
- Weighted error of each new component classifier

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

tends to increase as a function of boosting iterations.



# How Will Test Error Behave? (A First Guess)




expect:

- training error to continue to drop (or reach zero)
- test error to increase when $H_{\mathrm{final}}$ becomes "too complex"
  - "Occam's razor"
  - overfitting
    - hard to know when to stop training

# Technically...

- with high probability:

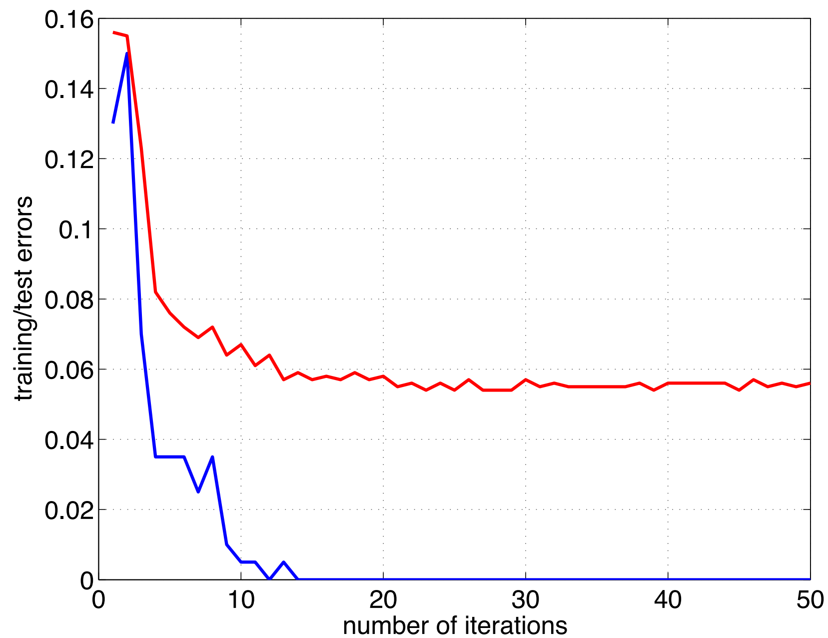
$$\text{generalization error} \leq \text{training error} + \tilde{O}\left(\sqrt{\frac{dT}{m}}\right)$$

  - bound depends on
    - $m = \#$ training examples
    - $d =$ "complexity" of weak classifiers
    - $T = \#$ rounds
- generalization error $= \mathrm{E}$ [test error]
- predicts overfitting

# “Typical” performance

- Training and test errors of the *combined classifier*

$$\hat{h}_m(\mathbf{x}) = \hat{\alpha}_1 h(\mathbf{x}; \hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}; \hat{\theta}_m)$$




- 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":

$$\hat{h}_m(\mathbf{x}) = \frac{\hat{\alpha}_1 h(\mathbf{x};\hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x};\hat{\theta}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m}$$

- This allows us to define a clear notion of "voting margin" that the combined classifier achieves for each training example:

$$\text{margin}(\mathbf{x}_i) = y_i \cdot \hat{h}_m(\mathbf{x}_i)$$

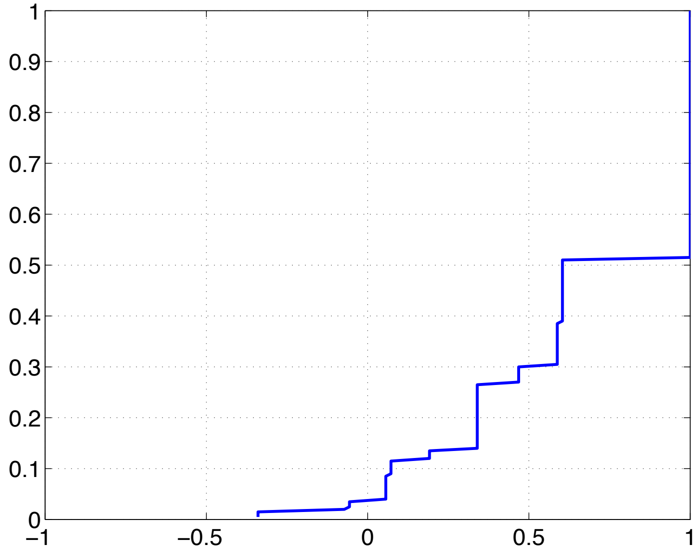
The margin lies in $[-1, 1]$ and is negative for all misclassified examples.

# AdaBoost and margin

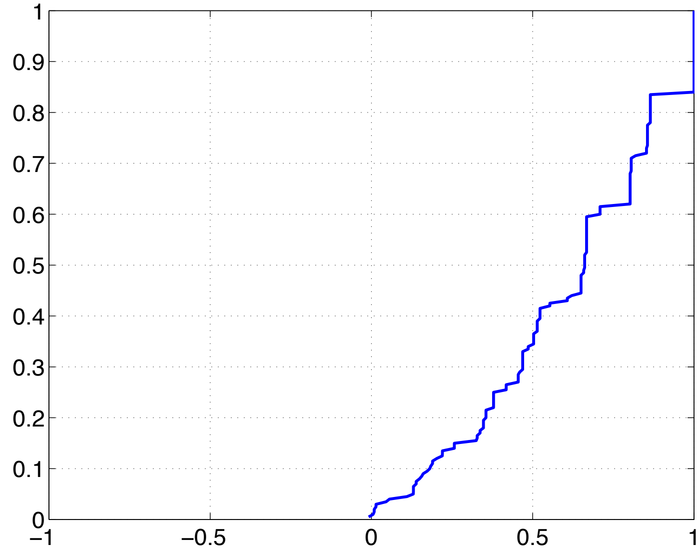
- Successive boosting iterations still improve the majority vote or margin for the training examples

$$\text{margin}(\mathbf{x}_i) \;=\; y_i \left[ \frac{\hat{\alpha}_1 h(\mathbf{x}_i;\hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}_i;\hat{\theta}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m} \right]$$

- Cumulative distributions of margin values:



4 iterations



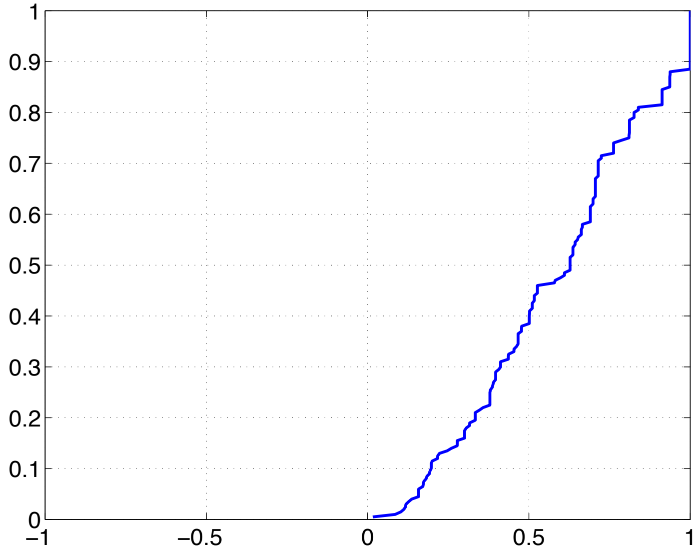
10 iterations

# AdaBoost and margin

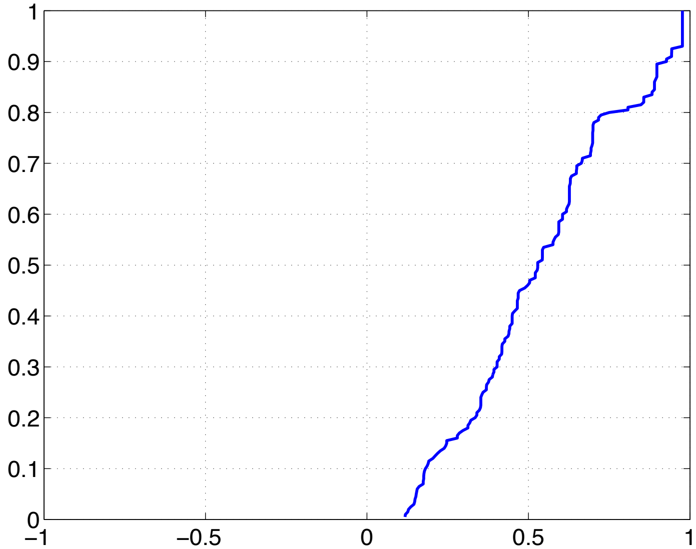
- Successive boosting iterations still improve the majority vote or margin for the training examples

$$\text{margin}(\mathbf{x}_i) = y_i \left[ \frac{\hat{\alpha}_1 h(\mathbf{x}_i;\hat{\theta}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}_i;\hat{\theta}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m} \right]$$

- Cumulative distributions of margin values:




20 iterations




50 iterations

# 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

$$\phi_i(\mathbf{x}) = h(\mathbf{x}; \hat{\theta}_i), i = 1, \ldots, m$$

- 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

$$f(\mathbf{x}; \alpha) = \alpha_1 \phi_1(\mathbf{x}) + \ldots \alpha_m \phi_m(\mathbf{x})$$

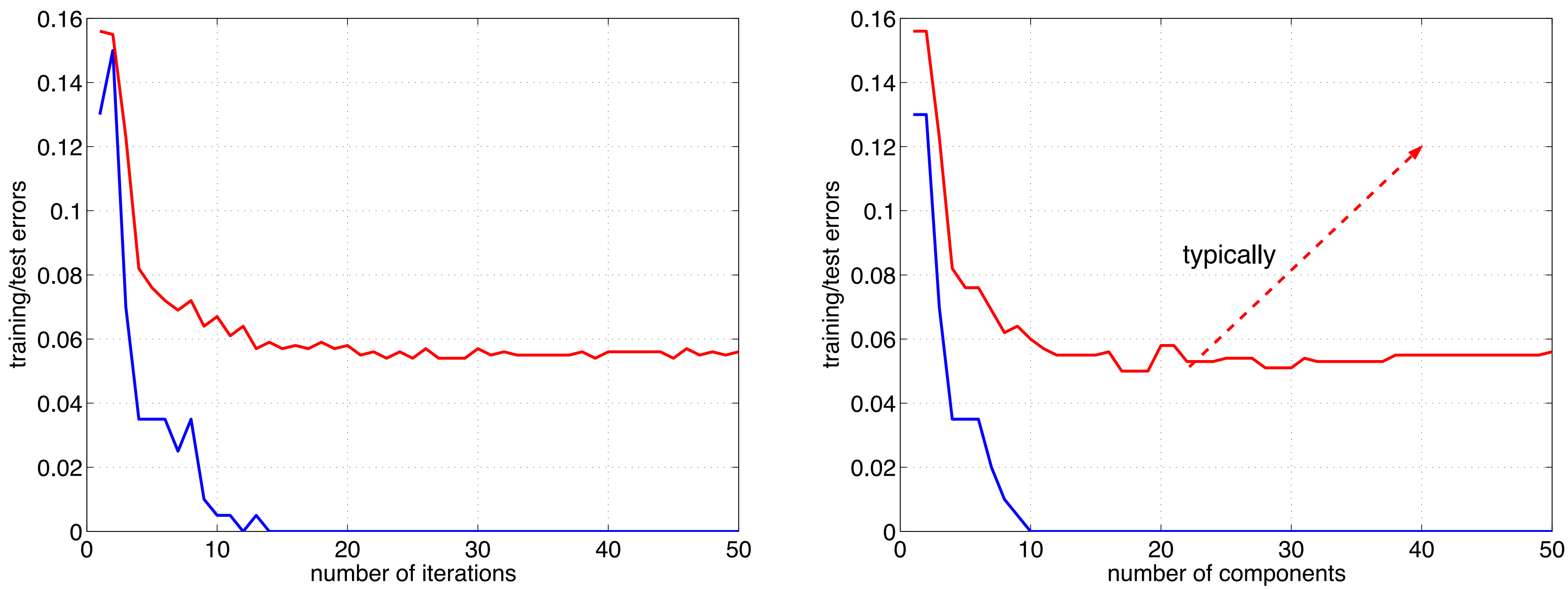
where each parameter $\alpha_i$ can be now any real number (even negative). The parameters would be estimated jointly rather than one after the other as in boosting.

# Can we improve the combination?

- We could use SVMs in a postprocessing step to reoptimize

$$f(\mathbf{x};\alpha) = \alpha_1\phi_1(\mathbf{x}) + \ldots \alpha_m\phi_m(\mathbf{x})$$

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




boosting　　　　svm postprocessing

# 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

    $\rightarrow$ overfitting
  - weak classifiers too weak ($\gamma_t \rightarrow 0$ too quickly)

    $\rightarrow$ underfitting

    $\rightarrow$ low margins $\rightarrow$ overfitting
- empirically, AdaBoost seems especially susceptible to uniform noise

# Multiclass Problems

[with Freund]

- say $y \in Y$ where $|Y| = k$
- direct approach (AdaBoost.M1):

$$h_t : X \to Y$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$H_{\text{final}}(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \alpha_t$$

- can prove same bound on error if $\forall t : \epsilon_t \leq 1/2$
  - in practice, not usually a problem for "strong" weak learners (e.g., C4.5)
  - significant problem for "weak" weak learners (e.g., decision stumps)
- instead, reduce to binary

# The One-Against-All Approach

- break $k$-class problem into $k$ binary problems and solve each separately
- say possible labels are $Y = \{{\color{green}\blacksquare}, {\color{gold}\blacksquare}, {\color{red}\blacksquare}, {\color{cyan}\blacksquare}\}$

| | | | ${\color{green}\blacksquare}$ | | ${\color{gold}\blacksquare}$ | | ${\color{red}\blacksquare}$ | | ${\color{cyan}\blacksquare}$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | ${\color{gold}\blacksquare}$ | | $x_1$ | $-$ | $x_1$ | $+$ | $x_1$ | $-$ | $x_1$ | $-$ |
| $x_2$ | ${\color{red}\blacksquare}$ | | $x_2$ | $-$ | $x_2$ | $-$ | $x_2$ | $+$ | $x_2$ | $-$ |
| $x_3$ | ${\color{cyan}\blacksquare}$ | $\Rightarrow$ | $x_3$ | $-$ | $x_3$ | $-$ | $x_3$ | $-$ | $x_3$ | $+$ |
| $x_4$ | ${\color{gold}\blacksquare}$ | | $x_4$ | $-$ | $x_4$ | $+$ | $x_4$ | $-$ | $x_4$ | $-$ |
| $x_5$ | ${\color{green}\blacksquare}$ | | $x_5$ | $+$ | $x_5$ | $-$ | $x_5$ | $-$ | $x_5$ | $-$ |

- to classify new example, choose label predicted to be “most” positive
- $\Rightarrow$ “AdaBoost.MH” [with Singer]
- problem: not robust to errors in predictions