Machine learning: lecture 10

Tommi S. Jaakkola
MIT CSAIL
tommi@csail.mit.edu

Topics

- Combination of classifiers
  - voted combination of stumps
  - loss, modularity, and weights
  - AdaBoost, properties

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

$$h_m(\mathbf{x}) = \alpha_1 \, h(\mathbf{x}; \theta_1) + \ldots + \alpha_m \, h(\mathbf{x}; \theta_m)$$

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

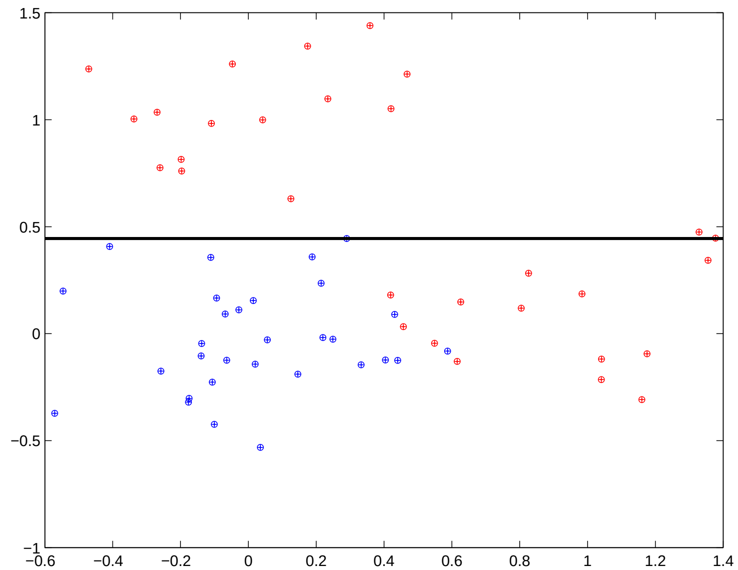
Components: decision stumps

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

$$h(\mathbf{x};\theta) = \text{sign}(\, w_1\, x_k - w_0\,)$$

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



Voted combination cont'd

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

$$h_m(\mathbf{x}) = \alpha_1 h(\mathbf{x};\theta_1) + \ldots + \alpha_m h(\mathbf{x};\theta_m)$$

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

$$\exp\{\, -y\, h_m(\mathbf{x})\, \}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{\, -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\}$$

$$= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i \alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{\, -y_i [h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\}$$

$$= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i \alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\{\, -y_i h_{m-1}(\mathbf{x}_i)\}}_{\text{fixed at stage } m} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{\, -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\}$$

$$= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\{\, -y_i h_{m-1}(\mathbf{x}_i)\}}_{\text{fixed at stage } m} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

$$= \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

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

- To increase modularity we'd like to further decouple the optimization of $h(\mathbf{x};\theta_m)$ from the associated votes α_m
- To this end we select $h(\mathbf{x};\theta_m)$ that optimizes the rate at which the loss would decrease as a function of α_m

$$\frac{\partial}{\partial \alpha_m}\Big|_{\alpha_m=0} \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\} =$$

Empirical exponential loss cont'd

- To increase modularity we'd like to further decouple the optimization of $h(\mathbf{x};\theta_m)$ from the associated votes α_m
- To this end we select $h(\mathbf{x};\theta_m)$ that optimizes the rate at which the loss would decrease as a function of α_m

$$\frac{\partial}{\partial \alpha_m}\Big|_{\alpha_m=0} \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\} =$$

$$\left[\sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\} \cdot \big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]_{\alpha_m=0}$$

Empirical exponential loss cont'd

- To increase modularity we'd like to further decouple the optimization of $h(\mathbf{x};\theta_m)$ from the associated votes α_m
- To this end we select $h(\mathbf{x};\theta_m)$ that optimizes the rate at which the loss would decrease as a function of α_m

$$\frac{\partial}{\partial \alpha_m}\Big|_{\alpha_m=0} \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\} =$$

$$\left[\sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\} \cdot \big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]_{\alpha_m=0}$$

$$= \left[\sum_{i=1}^{n} W_i^{(m-1)} \big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]$$

Empirical exponential loss cont'd

- We find $h(\mathbf{x}; \hat{\theta}_m)$ that minimizes

$$-\sum_{i=1}^{n} W_i^{(m-1)} y_i h(\mathbf{x}_i; \theta_m)$$

Empirical exponential loss cont'd

- We find $h(\mathbf{x};\hat{\theta}_m)$ that minimizes

$$-\sum_{i=1}^{n} W_i^{(m-1)} y_i h(\mathbf{x}_i;\theta_m)$$

We can also normalize the weights:

$$-\sum_{i=1}^{n} \frac{W_i^{(m-1)}}{\sum_{j=1}^{n} W_j^{(m-1)}} y_i h(\mathbf{x}_i;\theta_m)$$

$$= \; -\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i;\theta_m)$$

so that $\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1$.

Selecting a new component: summary

- We find $h(\mathbf{x};\hat{\theta}_m)$ that minimizes

$$-\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i;\theta_m)$$

where $\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1$.

- α_m is subsequently chosen to minimize

$$\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\hat{\theta}_m)\}$$

The AdaBoost algorithm

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

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* ϵ_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.

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* ϵ_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:

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

which minimizes the weighted loss when $h(\mathbf{x}; \theta) \in \{-1, 1\}$

$$\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\{ -y_i \alpha_m h(\mathbf{x}_i; \hat{\theta}_m) \}$$

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* ϵ_m

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

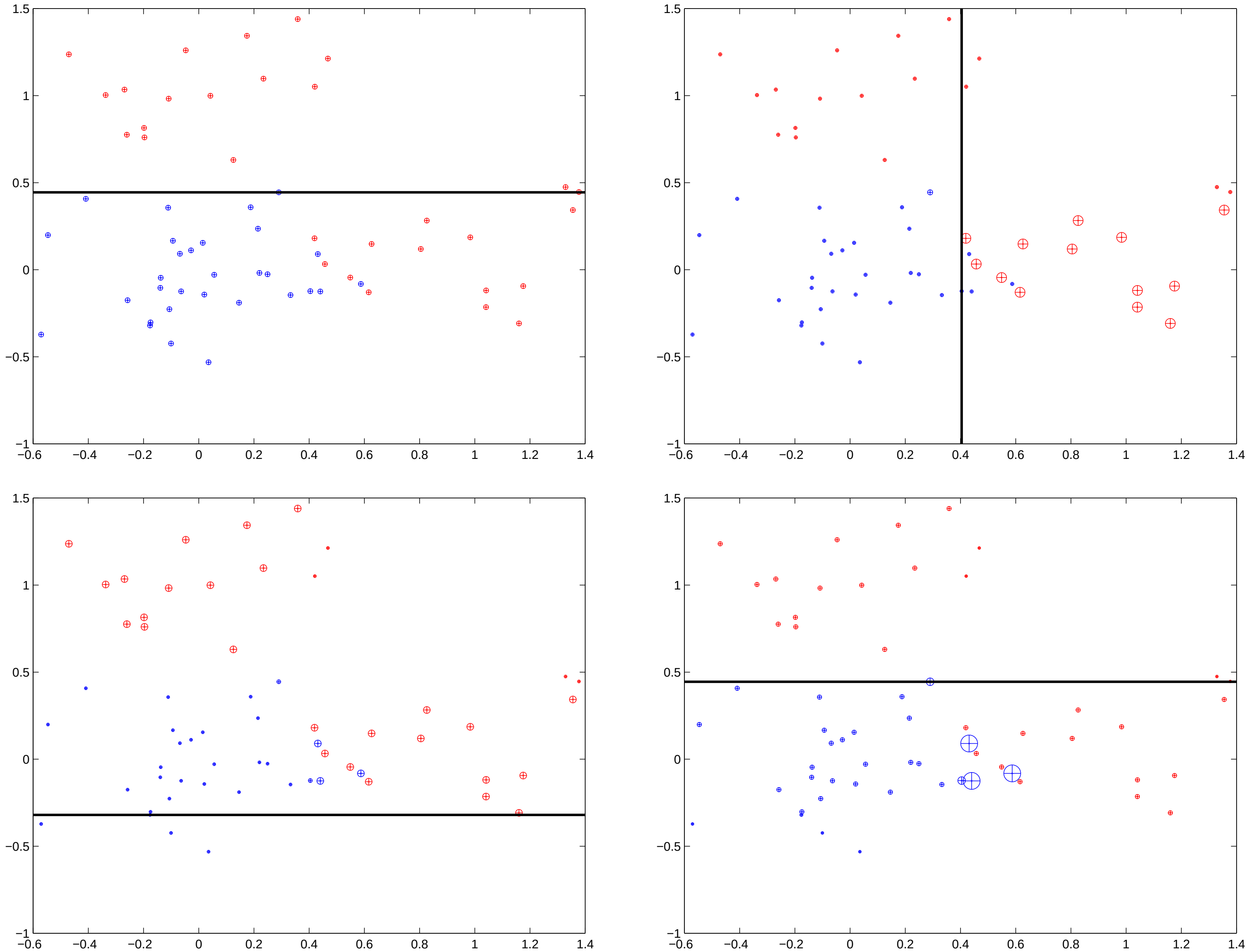
is better than chance.

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):

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



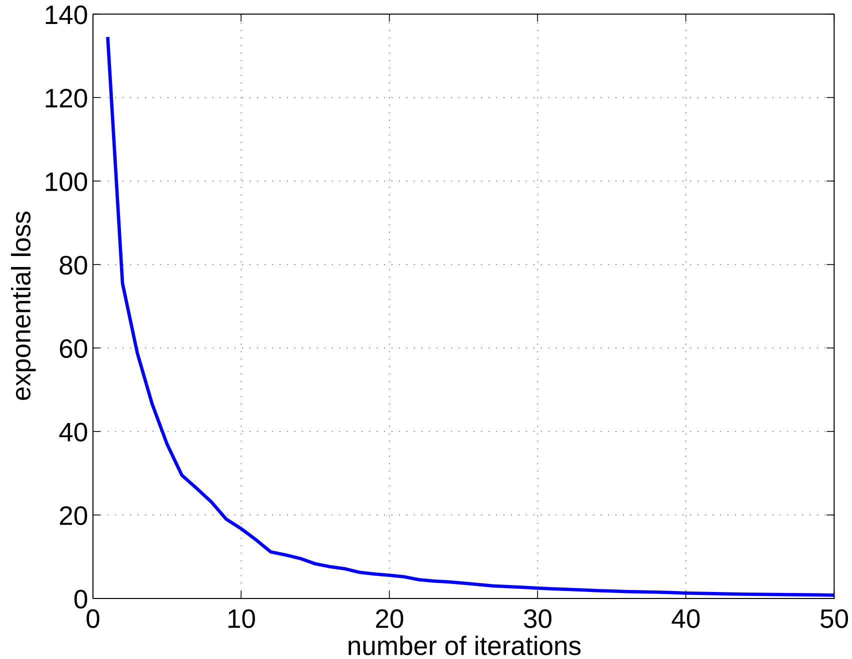
1.5
1
0.5
0
-0.5
-1
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4

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

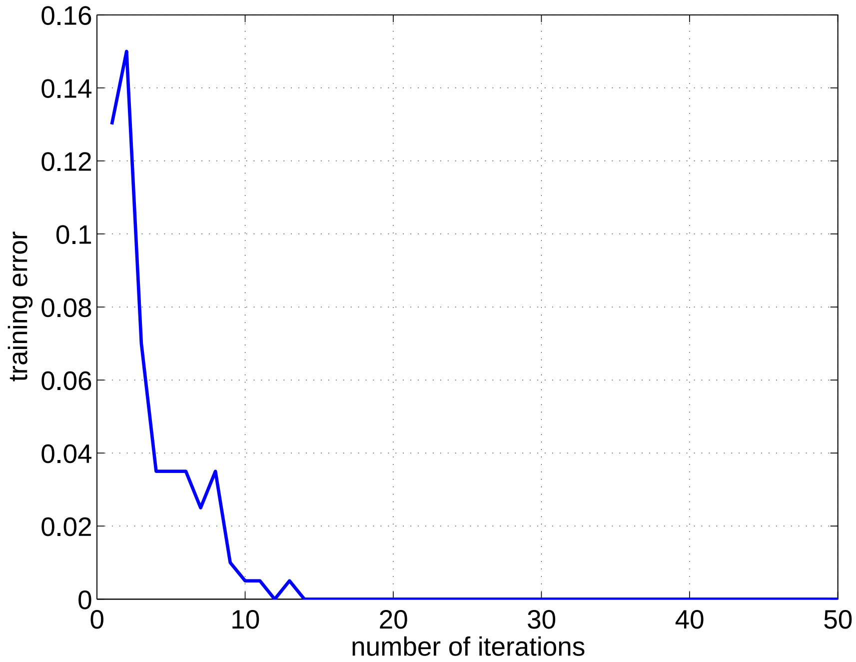


Adaboost properties: training error

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

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

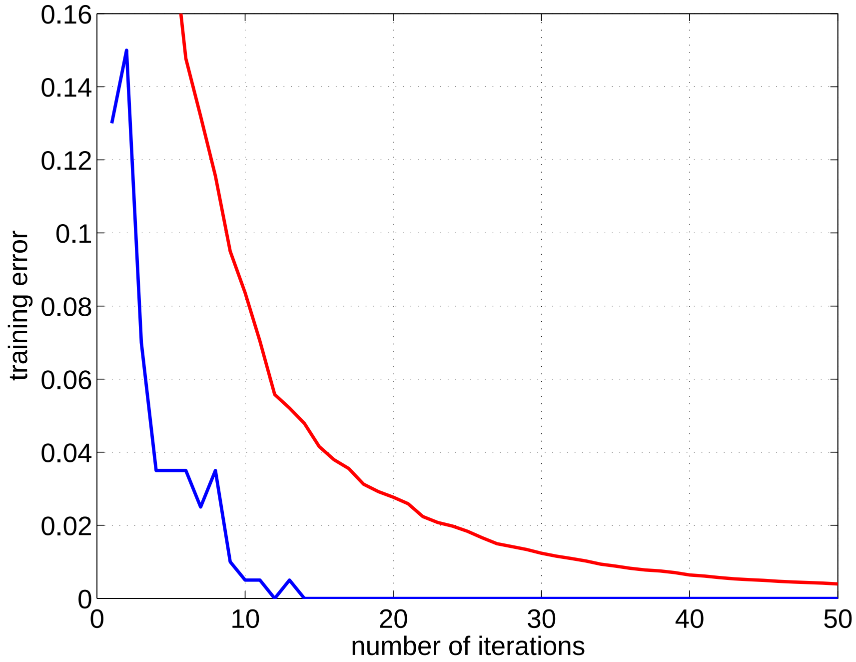
over the training examples.



Adaboost properties: training error cont'd

- The training classification error has to go down exponentially fast if the weighted errors of the component classifiers, ϵ_k, are strictly better than chance $\epsilon_k < 0.5$

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

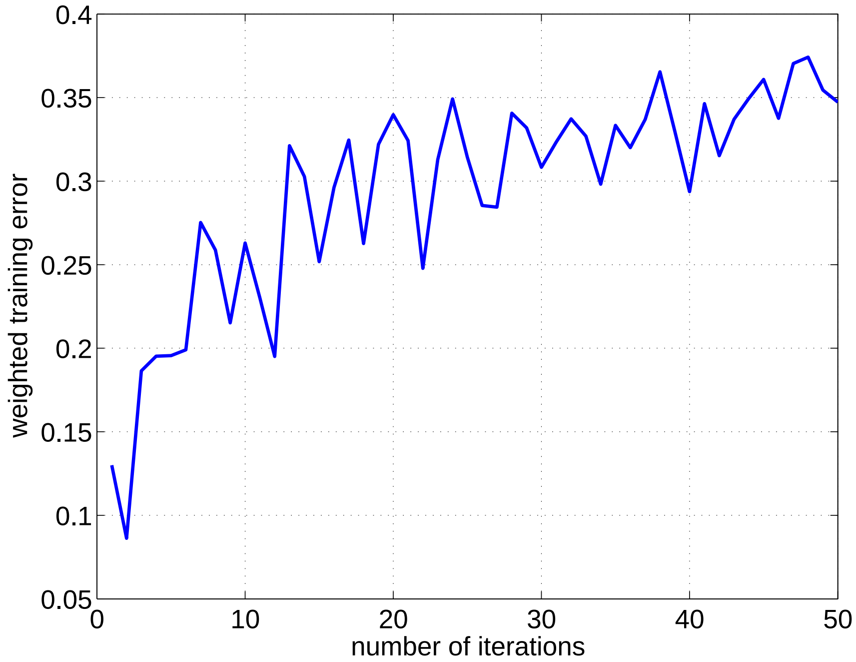


Adaboost properties: weighted error

- Weighted error of each new component classifier

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

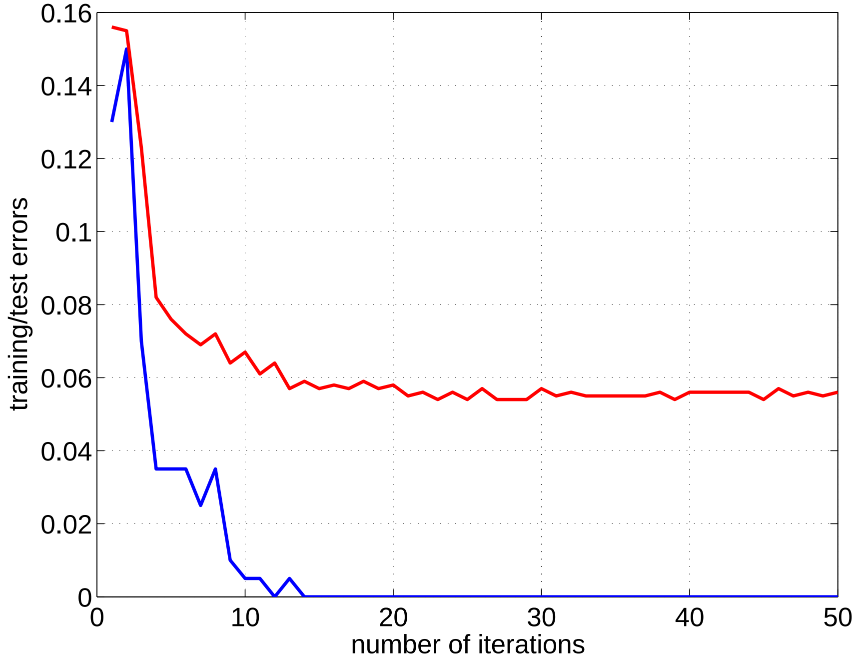
tends to increase as a function of boosting iterations.



"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:

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

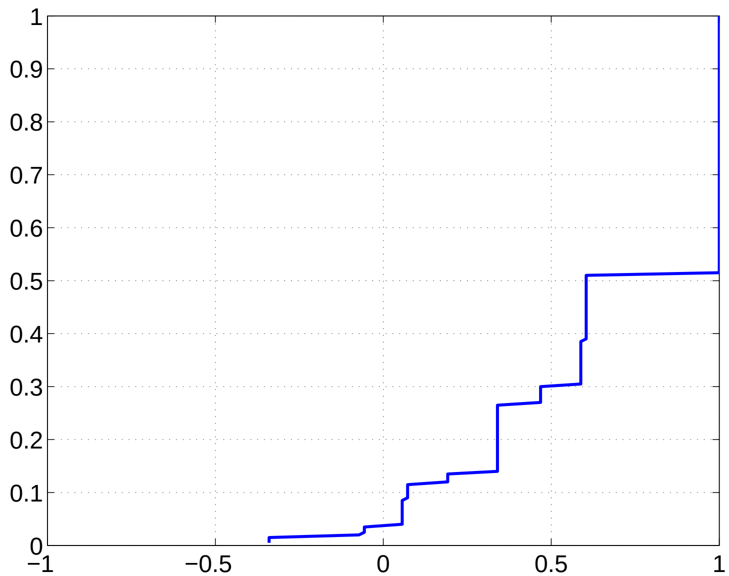
The margin lies in $[-1, 1]$ and is negative for all misclassified examples.

AdaBoost and margin

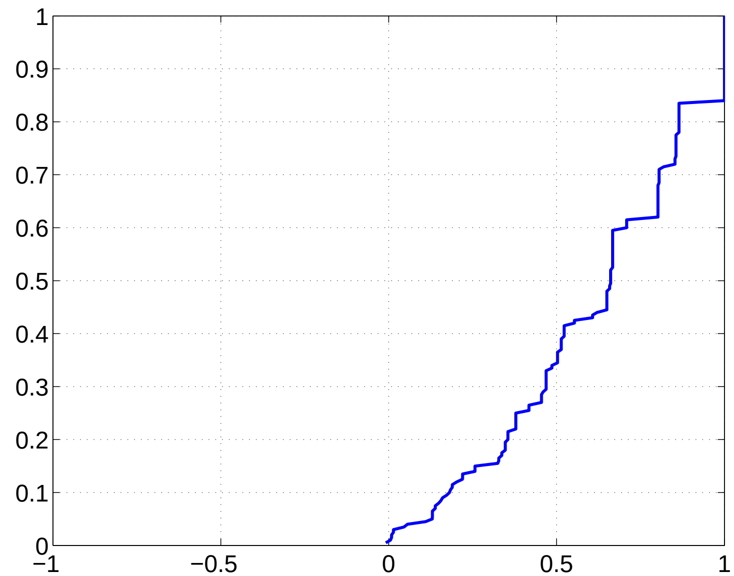
- Successive boosting iterations still improve the majority vote or margin for the training examples

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

- Cumulative distributions of margin values:



4 iterations



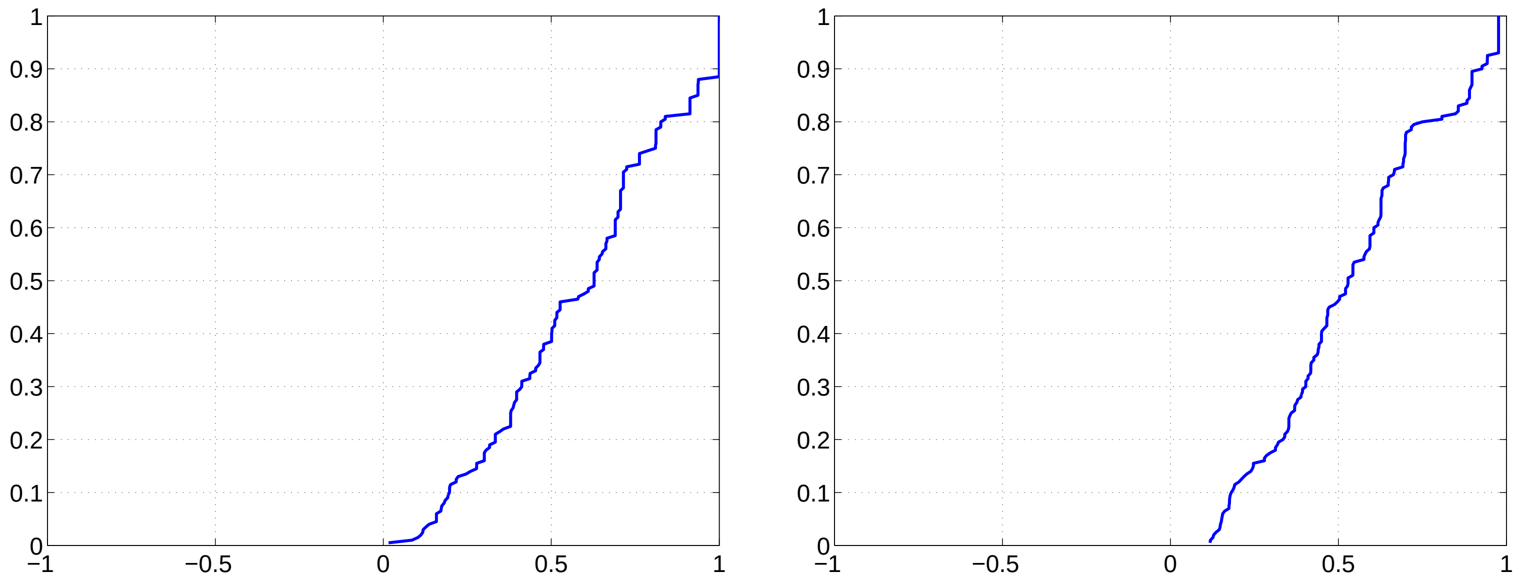
10 iterations

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

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

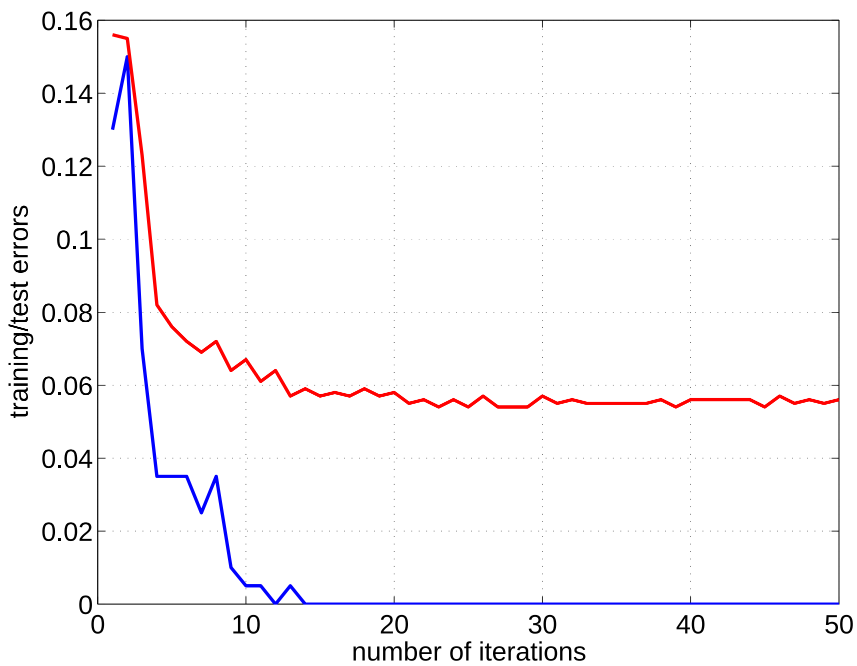
  where each parameter α_i can be now any real number (even negative). The parameters would be estimated jointly rather than one after the other as in boosting.

Can we improve the combination?

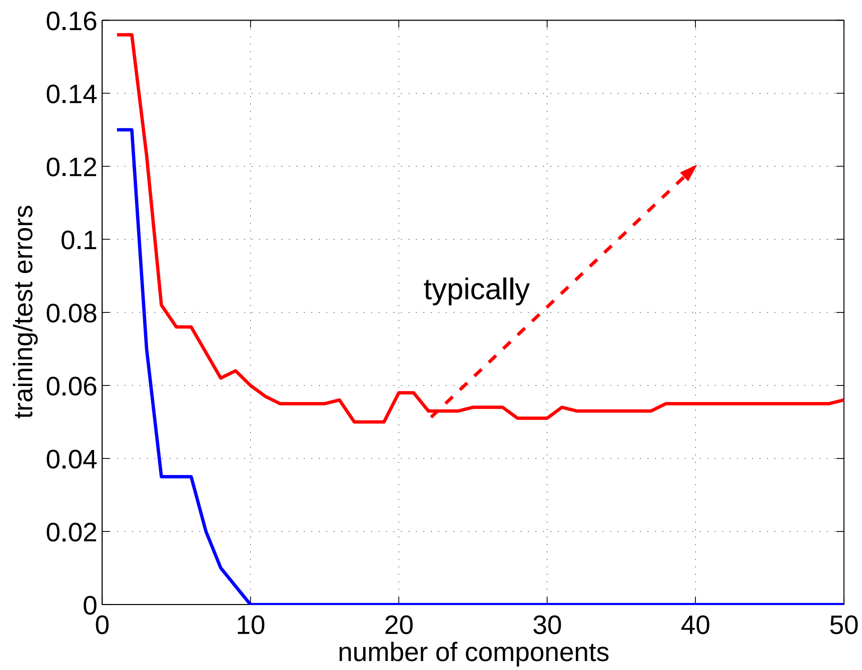
- We could use SVMs in a postprocessing step to reoptimize

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

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



boosting



svm postprocessing