CS9840
Learning and Computer Vision Prof. Olga Veksler

Lecture 9

Boosting

Some slides are due to Robin Dhamankar

Vandi Verma & Sebastian Thrun

Today

- New Machine Learning Topics:
  - Ensemble Learning
    - Bagging
    - Boosting

Ensemble Learning: Bagging and Boosting

- So far we have talked about design of a single classifier that generalizes well (want to "learn" f(x))
- From statistics, we know that it is good to average your predictions (reduces variance)
- Bagging is based on ensemble learning ideas
- Boosting was inspired by bagging

Bagging

- Generate a random sample from training set by selecting ***l*** elements (out of ***N*** elements available) with replacement
- If l = N, the new sampled dataset has, on average, 63.2% of training examples
  - each example has a probability of $1-(1-1/N)^N$ of being selected at least once. For $N \to \infty$, this converges to $(1-1/e)$ or 0.632 [Bauer and Kohavi, 1999]
- Repeat the sampling procedure, getting a sequence of ***k*** independent training sets
- Train classifiers $f_1(x), f_2(x), \ldots, f_k(x)$ for each of these training sets, using the same classification algorithm
- To classify an unknown sample x, let each classifier predict
- The *bagged classifier* $f_{FINAL}(x)$ combines predictions of individual classifiers, frequently by simple voting

$$f_{FINAL}(x) = \mathrm{sign}[\,1/k \,\Sigma\, f_i(x)\,]$$

Boosting: Motivation

- Hard to design accurate classifier which generalizes well
- Easy to find many rule of thumb or weak classifiers
  - a classifier is weak if it is slightly better than random guessing
  - example: if an email has word "money" classify it as spam, otherwise classify it as not spam
    - likely to be better than random guessing
- How combine weak classifiers to produce an accurate classifier?
  - Question people have been working on since 1980's
  - Ada-Boost (1996) was the first practical boosting algorithm
- Boosting
  - Assign different weights to training samples in a "smart" way so that different classifiers pay more attention to different samples
  - Weighted majority voting, the weight of individual classifier is proportional to its accuracy
  - Ada-boost was influenced by bagging, and it is superior to bagging

Ada Boost

- Assume 2-class problem, with labels +1 and -1
  - $\mathbf{y}^i$ in {-1,1}
- Ada boost produces a discriminant function:

$$\mathbf{g}(\mathbf{x}) = \sum_{\mathbf{t}=1}^{\mathbf{T}} \alpha_{\mathbf{t}} \mathbf{h}_{\mathbf{t}}(\mathbf{x}) = \alpha_1 \mathbf{h}_1(\mathbf{x}) + \alpha_2 \mathbf{h}_2(\mathbf{x}) + \ldots\ \alpha_{\mathbf{T}} \mathbf{h}_{\mathbf{T}}(\mathbf{x})$$

- Where $\mathbf{h}_t(\boldsymbol{x})$ is a weak classifier, for example:

$$\mathbf{h}_{\mathbf{t}}(\mathbf{x}) = \begin{cases} -1 & \text{if email has word “money”} \\ 1 & \text{if email does not have word “money”} \end{cases}$$

- The final classifier is the sign of the discriminant function

$$\mathbf{f}_{\text{final}}(\mathbf{x}) = \text{sign}[\mathbf{g}(\mathbf{x})]$$

Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially weights are equal
- Main Idea: at successive iterations, the weight of misclassified examples is increased
- This forces the algorithm to concentrate on examples that have not been classified correctly so far

Idea Behind Ada Boost

- Examples of high weight are shown more often at later rounds
- Face/nonface classification problem:

Round 1

	1/7	1/7	1/7	1/7	1/7	1/7	1/7
best weak classifier:	✓	✗	✓	✓	✗	✓	✗
change weights:	1/16	1/4	1/16	1/16	1/4	1/16	1/4

Round 2

best weak classifier:	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
change weights:		1/8	1/32	11/32		1/2		1/8	1/32	1/32

Idea Behind Ada Boost

Round 3

- out of all available weak classifiers, we choose the one that works best on the data we have at round 3
- we assume there is always a weak classifier better than random (better than 50% error)
-  image is half of the data given to the classifier
- chosen weak classifier **has to** classify this image correctly

More Comments on Ada Boost

- Ada boost is simple to implement, provided you have an implementation of a “weak learner”
- Will work as long as the “basic” classifier $\mathbf{h}_t(\mathbf{x})$ is at least slightly better than random
  - will work if the error rate of $\boldsymbol{h}_t(\boldsymbol{x})$ is less than 0.5
  - 0.5 is the error rate of a random guessing for a 2-class problem
- Can be applied to boost any classifier, not necessarily weak
  - but there may be no benefits in boosting a “strong” classifier

Ada Boost for 2 Classes

Initialization step: for each example **x**, set

$$\mathbf{D}(\mathbf{x}) = \frac{1}{\mathbf{N}}$$, where **N** is the number of examples

Iteration step (for **t** = 1…T):

1. Find best weak classifier $\mathbf{h_t(x)}$ using weights $\mathbf{D(x)}$
2. Compute the error rate $\boldsymbol{\varepsilon_t}$ as

$$\boldsymbol{\varepsilon}_t = \sum_{i=1}^{N} \mathbf{D}(\mathbf{x}^i) \cdot \underbrace{\mathbf{I}[\mathbf{y}^i \neq \mathbf{h}_t(\mathbf{x}^i)]}_{= \begin{cases} 1 & \text{if } \mathbf{y}^i \neq \mathbf{h}_t(\mathbf{x}^i) \\ 0 & \text{otherwise} \end{cases}}$$

3. compute weight α_t of classifier $\mathbf{h_t}$

$$\alpha_t = \log\left((1-\varepsilon_t)/\varepsilon_t\right)$$

4. For each $\mathbf{x}^i$, $\mathbf{D}(\mathbf{x}^i) = \mathbf{D}(\mathbf{x}^i) \cdot \exp\left(\alpha_t \cdot \mathbf{I}[\mathbf{y}^i \neq \mathbf{h}_t(\mathbf{x}^i)]\right)$

5. Normalize $\mathbf{D}(\mathbf{x}^i)$ so that $\sum_{i=1}^{N} \mathbf{D}(\mathbf{x}^i) = 1$

$$\boxed{\mathbf{f}_{\text{final}}(\mathbf{x}) = \text{sign}\left[\sum \alpha_t \mathbf{h}_t(\mathbf{x})\right]}$$

Ada Boost: Step 1

1. Find best weak classifier $\mathbf{h}_t(\mathbf{x})$ using weights $\mathbf{D}(\mathbf{x})$
   - some classifiers accept weighted samples, but most don't
   - if classifier does not take weighted samples, sample from the training samples according to the distribution $\mathbf{D}(\mathbf{x})$

1/16 1/4 1/16 1/16 1/4 1/16 1/4

- Draw **k** samples, each **x** with probability equal to **D**(**x**):

re-sampled examples

Ada Boost: Step 1

1. Find best weak classifier $\mathbf{h}_t(\mathbf{x})$ using weights $\mathbf{D}(\boldsymbol{x})$

- Give to the classifier the re-sampled examples:

- To find the best weak classifier, go through **all** weak classifiers, and find the one that gives the smallest error on the re-sampled examples

weak classifiers	$\mathbf{h}_1(\mathbf{x})$	$\mathbf{h}_2(\mathbf{x})$	$\mathbf{h}_3(\mathbf{x})$	……….	$\mathbf{h}_m(\mathbf{x})$
errors:	0.46	0.36	0.16		0.43

the best classifier $h_t(x)$
to choose at iteration t

Ada Boost: Step 2

2. Compute $\boldsymbol{\varepsilon}_t$ the error rate as

$$\varepsilon_t = \sum_{i=1}^{N} D(x^i)\cdot \overbrace{I\left[y^i \neq h_t(x^i)\right]}^{=\begin{cases}1 & \text{if } \quad y^i \neq h_t(x^i)\\ 0 & \text{otherwise}\end{cases}}$$

1/16	1/4	1/16	1/16	1/4	1/16	1/4
✓	✓	✓	✗	✗	✓	✓

$$\varepsilon_t = \frac{1}{4} + \frac{1}{16} = \frac{5}{16}$$

- $\boldsymbol{\varepsilon}_t$ is the weight of all misclassified examples added
  - the error rate is computed over original examples, not the re-sampled examples
- If a weak classifier is better than random, then $\boldsymbol{\varepsilon}_t < ½$

Ada Boost: Step 3

3. compute weight α_t of classifier $\mathbf{h}_t$

$$\alpha_t = \log\left((1 - \boldsymbol{\varepsilon}_t)/\boldsymbol{\varepsilon}_t\right)$$

In example from previous slide:

$$\varepsilon_t = \frac{5}{16} \quad \Rightarrow \quad \alpha_t = \log \frac{1 - \frac{5}{16}}{\frac{5}{16}} = \log \frac{11}{5} \approx 0.8$$

- Recall that $\boldsymbol{\varepsilon}_t < ½$
- Thus $(1- \boldsymbol{\varepsilon}_t)/ \boldsymbol{\varepsilon}_t > 1 \Rightarrow \alpha_t > 0$
- The smaller is $\boldsymbol{\varepsilon}_t$, the larger is α_t, and thus the more importance (weight) classifier $\mathbf{h}_t(x)$

$$\text{final}(\mathbf{x}) = \text{sign}\left[\sum \alpha_t \mathbf{h}_t(\mathbf{x})\right]$$

Ada Boost: Step 4

4. For each $\mathbf{x}^i$, $\mathbf{D}(\mathbf{x}^i) = \mathbf{D}(\mathbf{x}^i) \cdot \mathbf{exp}(\alpha_t \cdot \mathbf{I}[\mathbf{y}^i \neq \mathbf{h}_t(\mathbf{x}^i)\,])$

from previous slide $\alpha_t = 0.8$

						
1/16	**1/4**	**1/16**	**1/16**	**1/4**	**1/16**	**1/4**
✓	✓	✓	✖	✖	✓	✓
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
1/16	**1/4**	**1/16**	(1/16) exp(0.8)	(1/4) exp(0.8)	**1/16**	**1/4**

- weight of misclassified examples is increased

Ada Boost: Step 5

5. Normalize $\mathbf{D}(\mathbf{x}^i)$ so that $\sum \mathbf{D}(\mathbf{x}^i) = 1$

from previous slide:

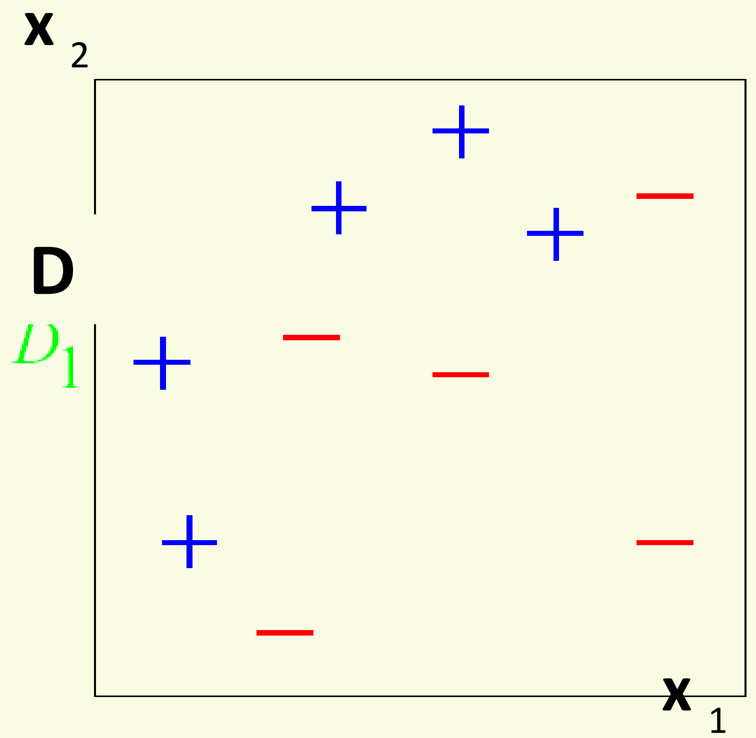
1/16 1/4 1/16 0.14 0.56 1/16 1/4

- after normalization

0.05 0.18 0.05 0.10 0.40 0.05 0.18

AdaBoost Example

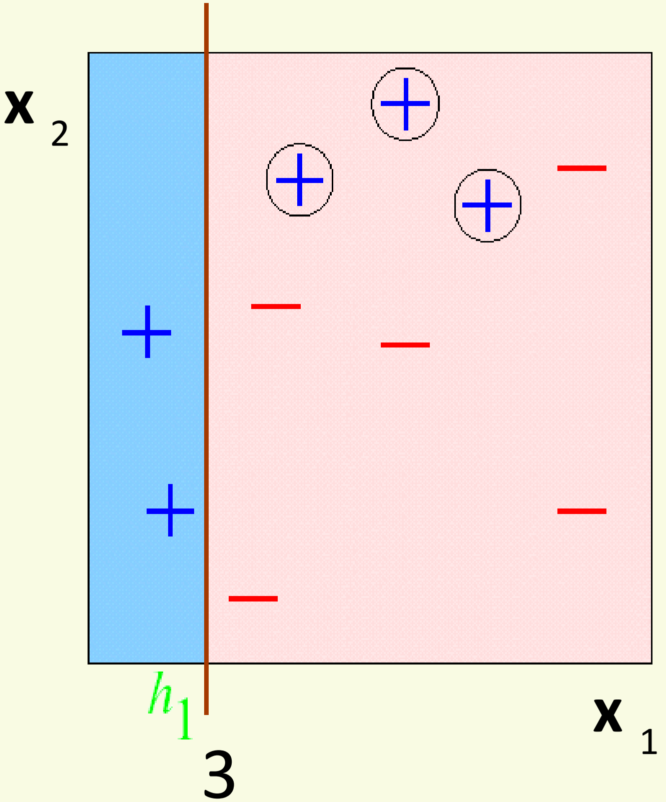
- Initialization: all examples have equal weights



from "A Tutorial on Boosting" by Yoav Freund and Rob Schapire

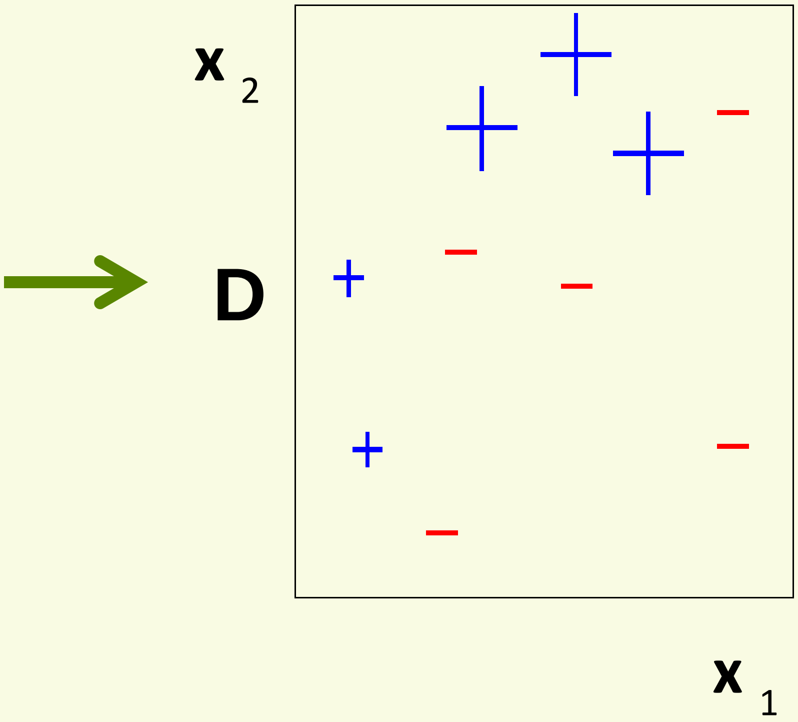
AdaBoost Example

ROUND 1



$\varepsilon_1 = 0.30$

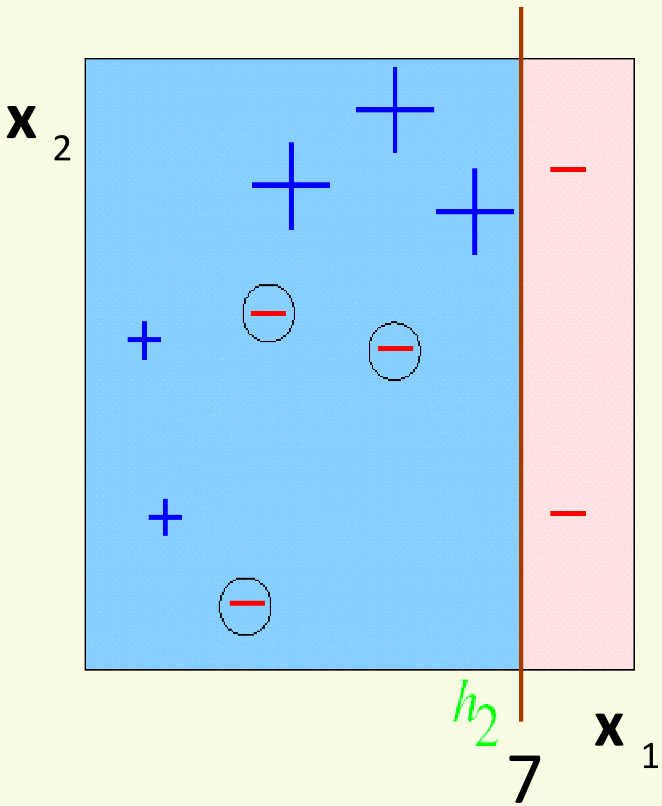
$\alpha_1 = 0.42$



$$h_1(x) = \textbf{sign}\left(3 - x_1\right)$$

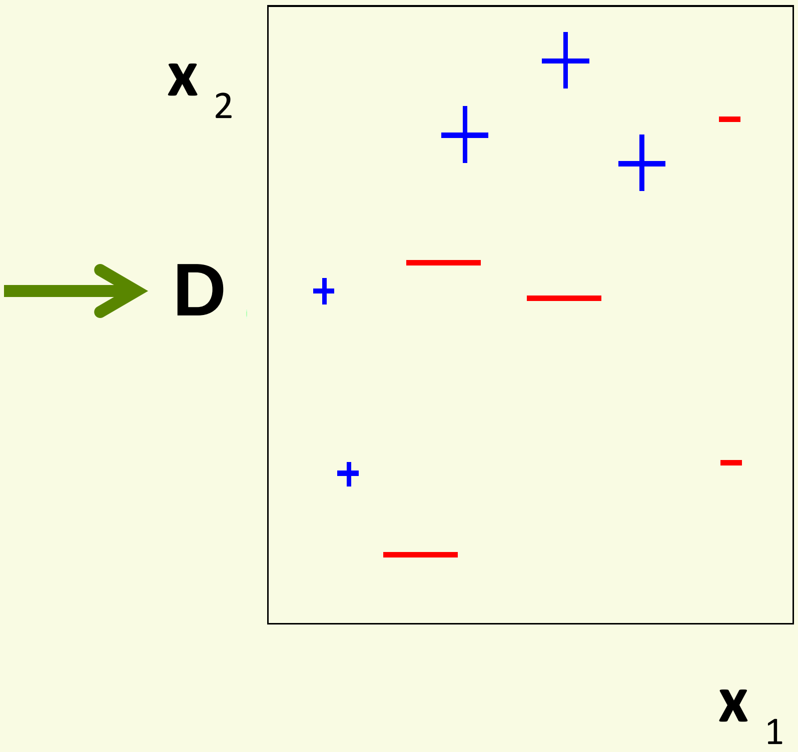
AdaBoost Example

ROUND 2



$\varepsilon_2 = 0.21$

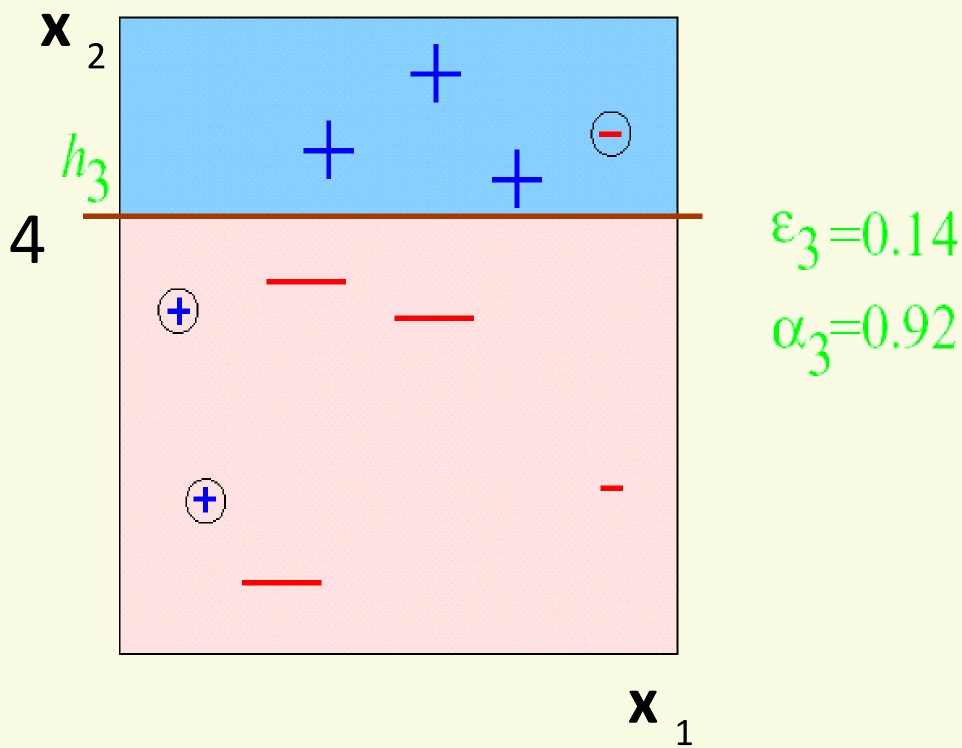
$\alpha_2 = 0.65$



$$h_2(x) = \text{sign}\,(7 - x_1)$$

AdaBoost Example

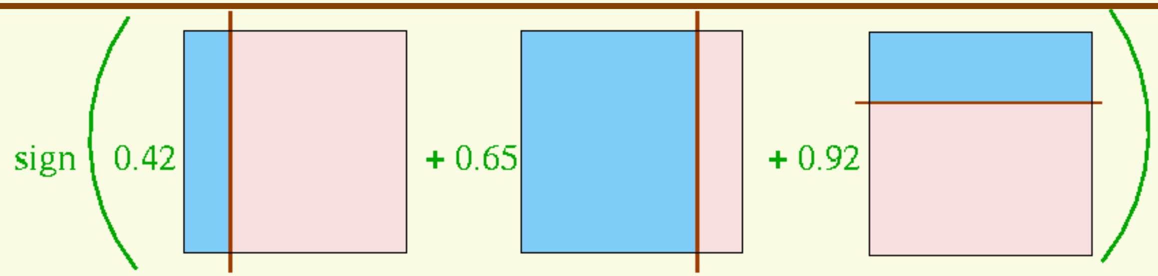
ROUND 3



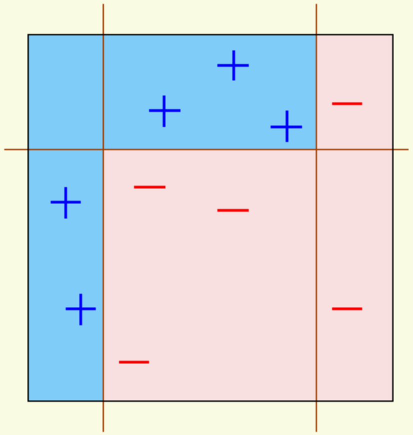
$$h_3(x) = \text{sign}\,(x_2 - 4)$$

AdaBoost Example

$\mathbf{f}_{\text{final}}(\mathbf{x}) =$



=



$$\mathbf{f}_{\text{final}}(\mathbf{x}) =$$

$$\mathbf{sign}\,\left(0.42\,\mathbf{sign}\,(3 - \mathbf{x}_1) + 0.65\,\mathbf{sign}\,(7 - \mathbf{x}_1) + 0.92\,\mathbf{sign}\,(\mathbf{x}_2 - 4)\right)$$

- note non-linear decision boundary

AdaBoost Comments

- Can show that training error drops exponentially fast

$$\mathbf{Err}_{\mathbf{train}} \le \mathbf{exp}\left(-2\sum_{\mathbf{t}} \gamma_{\mathbf{t}}^2\right)$$

- Here $\gamma_t = \varepsilon_t - 1/2$, where ε_t is classification error at round t
- Example: let errors for the first four rounds be, 0.3, 0.14, 0.06, 0.03, 0.01 respectively. Then

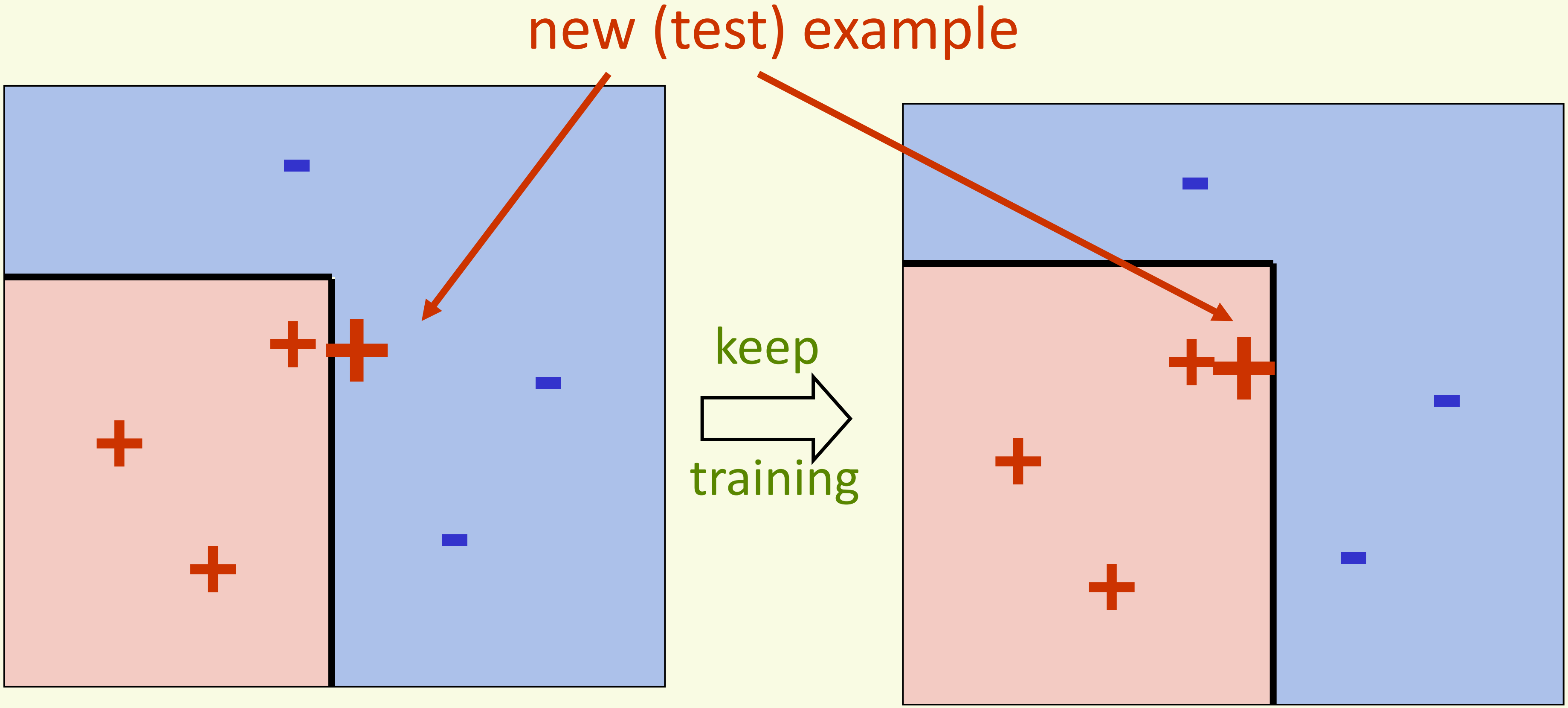
$$\mathbf{Err}_{\mathbf{train}} \le \mathbf{exp}\left[-2\left(0.2^2 + 0.36^2 + 0.44^2 + 0.47^2 + 0.49^2\right)\right]$$

$$\approx 0.19$$

AdaBoost Comments

- We are really interested in the generalization properties of $\mathbf{f}_{FINAL}(\mathbf{x})$, not the training error
- AdaBoost was shown to have excellent generalization properties in practice
  - the more rounds, the more complex is the final classifier, so overfitting is expected as the training proceeds
  - but in the beginning researchers observed no overfitting of the data
  - It turns out it does overfit data eventually, if you run it really long
- It can be shown that boosting increases the margins of training examples, as iterations proceed
  - larger margins help better generalization
  - margins continue to increase even when training error reaches zero
  - helps to explain empirically observed phenomena: test error continues to drop even after training error reaches zero

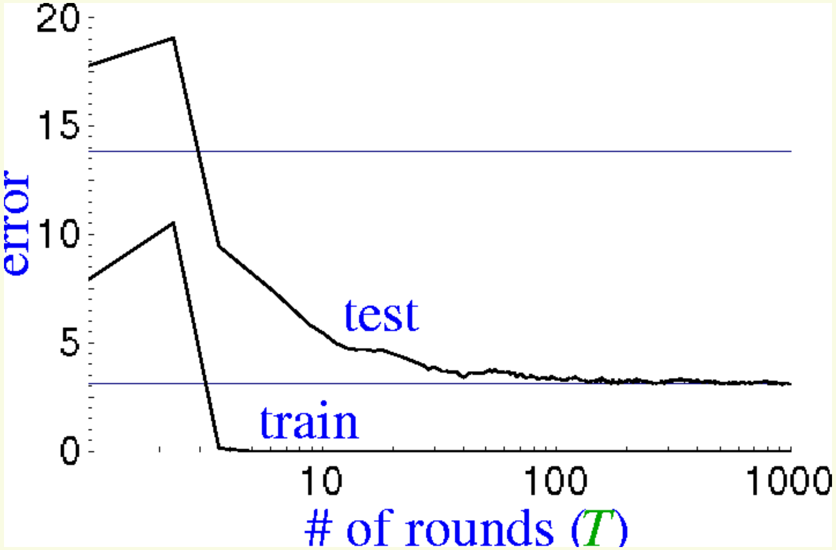
AdaBoost Example

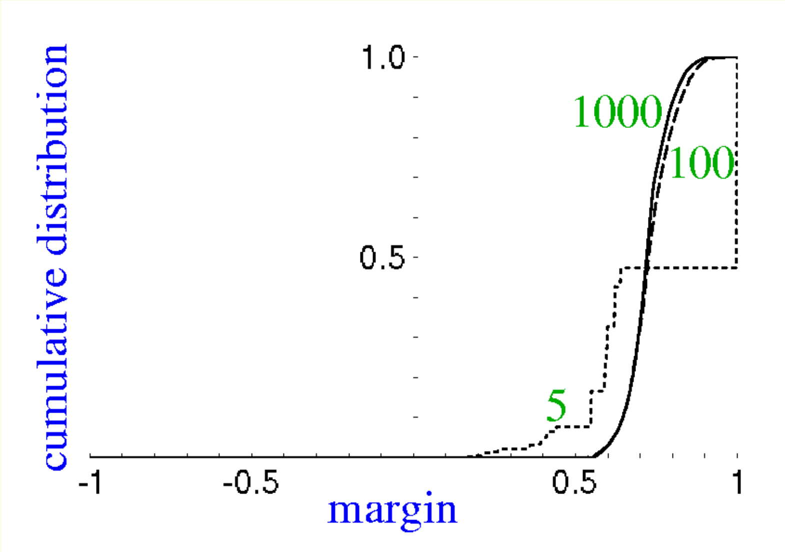


- zero training error

- zero training error
- larger margins helps better genarlization

Margin Distribution





Iteration number	5	100	1000
training error	0.0	0.0	0.0
test error	8.4	3.3	3.1
%margins≤0.5	7.7	0.0	0.0
Minimum margin	0.14	0.52	0.55

Boosting As Additive Model

- The final prediction in boosting *g(x)* can be expressed as an additive expansion of individual classifiers

$$g(x) = \sum_{k=1}^{M} \alpha_k f_k(x; \gamma_k)$$

- Typically we would try to minimize a loss function on the N training examples

$$\min_{\alpha_1, \gamma_1, \ldots, \gamma_M, \alpha_M} \sum_{i=1}^{N} L\left(y_i, \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \right)$$

- For example, under squared-error loss:

$$\min_{\alpha_1, \gamma_1, \ldots, \gamma_M, \alpha_M} \sum_{i=1}^{N} \left(y_i - \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \right)^2$$

Boosting As Additive Model

- Forward stage-wise modeling is iterative and fits the $f_k(x,\gamma_k)$ sequentially, fixing the results of previous iterations

$$\underbrace{g_t(x)}_{\text{model at iteration } t} = \overbrace{\boxed{g_{t-1}(x)}}^{\textit{fixed}} + \overbrace{\alpha_t f_t(x;\gamma_t)}^{\text{fit } \gamma_t,\ \alpha_t \text{ to produce improved } g_t(x)}$$

- Under the squared difference loss function:

$$L(y_i, g_{t-1}(x_i) + \alpha_t f_t(x_i;\gamma_t)) =$$

$$= (\underbrace{\boxed{y_i - g_{t-1}(x_i)}}_{\textit{fixed}} - \alpha_t f_t(x_i;\gamma_t))^2$$

- Forward stage-wise optimization seems to produce classifier with better generalization, doing the process stagewise seems to overfit less quickly

Boosting As Additive Model

$$g(x) = \sum_{k=1}^{M} \alpha_k f_k(x; \gamma_k)$$

- It can be shown that AdaBoost uses forward stage-wise modeling under the following loss function:
  - L(y, g (x)) = exp(-y · g (x))
    - the exponential loss function
  - At stage (or iteration) ***m***, we fit:

$$\arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} L(y_i, g(x_i)) =$$

$$= \arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot [g_{m-1}(x_i) + \alpha_m \cdot f_m(x_i)])$$

$$= \arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot g_{m-1}(x_i)) \cdot \exp(-y_i \cdot \alpha_m \cdot f_m(x_i))$$

Exponential Loss vs. Squared Error Loss

- $L(y, g(x)) = \exp(-y \cdot g(x))$
- $L(y, g(x)) = (y - g(x))^2$

- Squared Error Loss penalizes classifications that are "too correct", with $y \cdot g(x) > 1$, and thus it is inappropriate for classification
- Exponential loss encourages large margins, want $y \cdot g(x)$ large

Logistic Regression Model

- It can be shown that Adaboost builds a logistic regression model:

$$g(x) = \log \frac{Pr(Y = 1 \mid x)}{Pr(Y = -1 \mid x)} = \sum_{k=1}^{M} \alpha_m f_m(x)$$

- It can also be shown that the the training error on the samples is at most:

$$\sum_{i=1}^{N} \exp(-y_i \cdot g(x_i)) = \sum_{i=1}^{N} \exp\left(-y_i \cdot \sum_{k=1}^{M} \alpha_m f_m(x_i)\right)$$

Practical Advantages of AdaBoost

- Can construct arbitrarily complex decision regions
- Fast
- Simple
- Has only one parameter to tune, **T**
- Flexible: can be combined with any classifier
- provably effective (assuming weak learner)
  - shift in mind set: goal now is merely to find hypotheses that are better than random guessing

Caveats

- AdaBoost can fail if
  - weak hypothesis too complex (overfitting)
  - weak hypothesis too weak ($\gamma_t \to 0$ too quickly),
    - underfitting
- empirically, AdaBoost seems especially susceptible to noise
  - noise is the data with wrong labels