Ensemble learning: Bagging & Boosting

Machine Learning

Dr. Hamid R Rabiee – Zahra Dehghanian

Spring 2025

Sharif University
of Technology

What is ensemble learning?

- Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis.
  - combining many *weak learners* in an attempt to produce a *strong learner*.

- *Ensemble* term is usually reserved for methods that generate multiple hypotheses using the **same base learner**.

- *Multiple classifier* (broader term) also covers **combination of hypotheses** that are not induced by the same base learner.

Ensemble Methods

- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

How to generate an ensemble of classifiers?

- Bagging: bootstrap aggregating
- Boosting

Ensemble learning

- We only talk about:
  - **Bagging**: **B**ootstrap **agg**regat**ing**
    - The most famous bagging algorithm: RandomForest
  - **Boosting**
    - One important committee method
    - The most famous boosting algorithm: AdaBoost

Bagging algorithm (Breiman, 96)

- Each member of the ensemble is constructed from a different training dataset
  - samples training data uniformly at random with replacement

- Predictions combined either by uniform averaging or voting over class labels.
  - works best with unstable models (high variance models)

- Despite its apparent simplicity, Bagging is still not fully understood

Bootstrap Sampling

- Bootstrap sampling: Samples the given dataset N times uniformly with replacement (resulting in a set of N samples)
  - Some samples in the original set may be included several times in the bootstrap sampled data
- Bootstrap sampling: like "roll N-face dice N times"

Bagging algorithm

- **Input:** Required ensemble size M

  Training set $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$

- **for** t=1 to T **do**
  - Build a dataset D_t by sampling N items, randomly with replacement from D
  - Train a model h_t using D_t, and add it to the ensemble.

- $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} h_t(\boldsymbol{x}))$
  - Combine models by voting for classification and by averaging for regression

Bagging on decision trees

- Decision trees are popular classifiers:
  - interpretable
  - can handle discrete and continuous features
  - robust to outliers
  - low bias

- However, they are high variance

- Trees are perfect candidates for ensembles
  - Consider averaging many (nearly) unbiased tree estimators
  - Bias remains similar, but variance is reduced
  - This is called bagging
    - Train many trees on bootstrapped data, then average outputs

Bagging

- Sampling with replacement
- Build classifier on each bootstrap sample

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Algorithm 5.6 Bagging algorithm.

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size N, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \underset{y}{\operatorname{argmax}} \sum_i \delta\big(C_i(x) = y\big)$.
$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise$\}$.

Random Forest

- Bagging on decision trees

- Reduce correlation between trees, by introducing randomness
  - For b = 1, …, B,
    - Draw a bootstrap dataset
    - Learn a tree on this dataset
      - Select m features randomly out of d features as candidates before splitting
  - Output:
    - Regression: average of outputs
    - Classification: majority vote

Usually: $m \le \sqrt{d}$

Random Forest

- ✓ ensemble methods specifically designed for decision tree classifiers
- ✓ multiple decision trees where each tree is generated based on the values of an independent set of random vectors

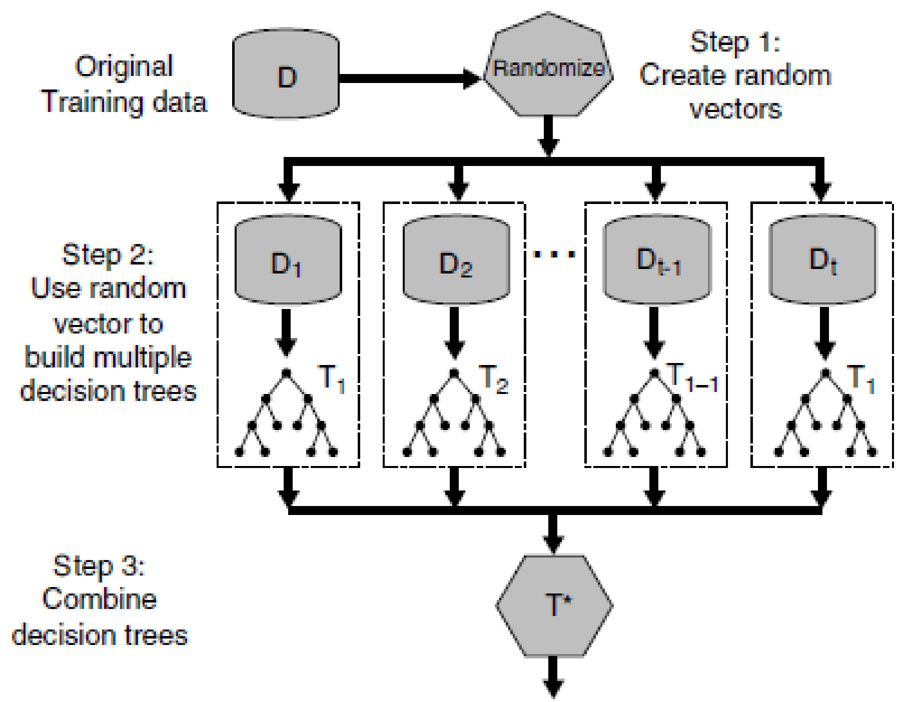


Figure 5.40. Random forests.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of boosting round

1. how the weights of the training examples are updated at the end of each boosting round?
2. how the predictions made by each classifier are combined?

Boosting idea

- We can select simple "weak" classification or regression methods and combine them into a single "strong" method

- Examples of weak classifiers: Naïve bayes, logistic regression, decision stumps or shallow decision trees

- Learn many weak classifiers that are good at different parts of the input space.

- To find the output class, find weighted vote of classifiers

Boosting: AdaBoost

Let $\{(\mathbf{x}j, yj) \mid j = 1, 2, \ldots, N\}$ denote a set of N training examples.
Unlike bagging, importance of a base classifier Ci depends on its error rate

$$\epsilon_i = \frac{1}{N}\left[\sum_{j=1}^{N} w_j\, I\Big(C_i(\mathbf{x}_j) \neq y_j\Big)\right], \qquad \alpha_i = \frac{1}{2}\ln\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$$

weight assigned to example ($\mathbf{x}i, yi$)
during the *jth* boosting round

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(\mathrm{x_i}) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(\mathrm{x_i}) \neq y_i \end{cases}$$

Normalization factor

AdaBoost

Algorithm 5.7 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all examples in the original training set, D.
7: $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \, \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error.}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each example according to Equation 5.69.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

Example

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

Boosting

- Try to combine many simple "weak" classifiers (in sequence) to find a single "strong" classifier
  - Each component is a simple binary ±1 classifier
  - Voted combinations of component classifiers

$$H_m(\boldsymbol{x}) = \alpha_1 h(\boldsymbol{x}; \boldsymbol{\theta}_1) + \ldots + \alpha_m h(\boldsymbol{x}; \boldsymbol{\theta}_m)$$

  - To simplify notation: $h(\boldsymbol{x}; \boldsymbol{\theta}_i) = h_i(\boldsymbol{x})$

$\alpha_i \geq 0$ are higher for more reliable classifiers

$$H_m(\boldsymbol{x}) = \alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_m h_m(\boldsymbol{x})$$

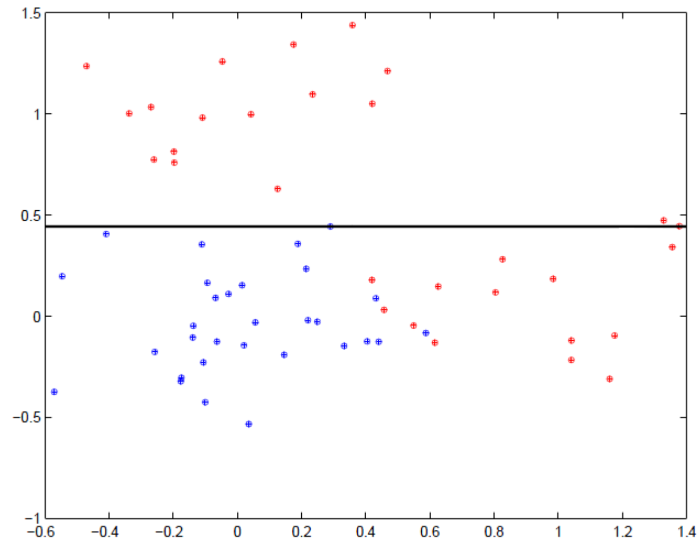
- Prediction: $\hat{y} = sign(H_t(\boldsymbol{x}))$

Simple component classifiers

- Simple family of component classifiers (called decision stumps):

$$h(\boldsymbol{x};\boldsymbol{\theta}) = sign(w_1 x_k - w_0) \qquad \boldsymbol{\theta} = \{k, w_1, w_0\}$$

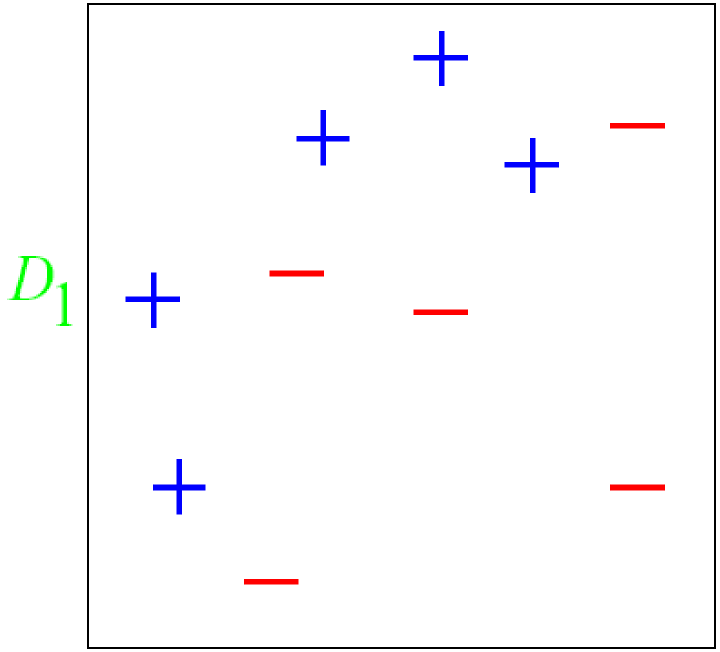
- Each classifier is based on only a single feature of $\boldsymbol{x}$ (e.g., x_k): decision tree of depth one



AdaBoost: basic ideas

- Sequential production of classifiers
  - choose classifier whose addition will be most helpful.
- Each classifier is dependent on the previous ones
  - focuses on the previous ones' errors
- Incorrectly predicted samples in previous classifiers are weighted more heavily

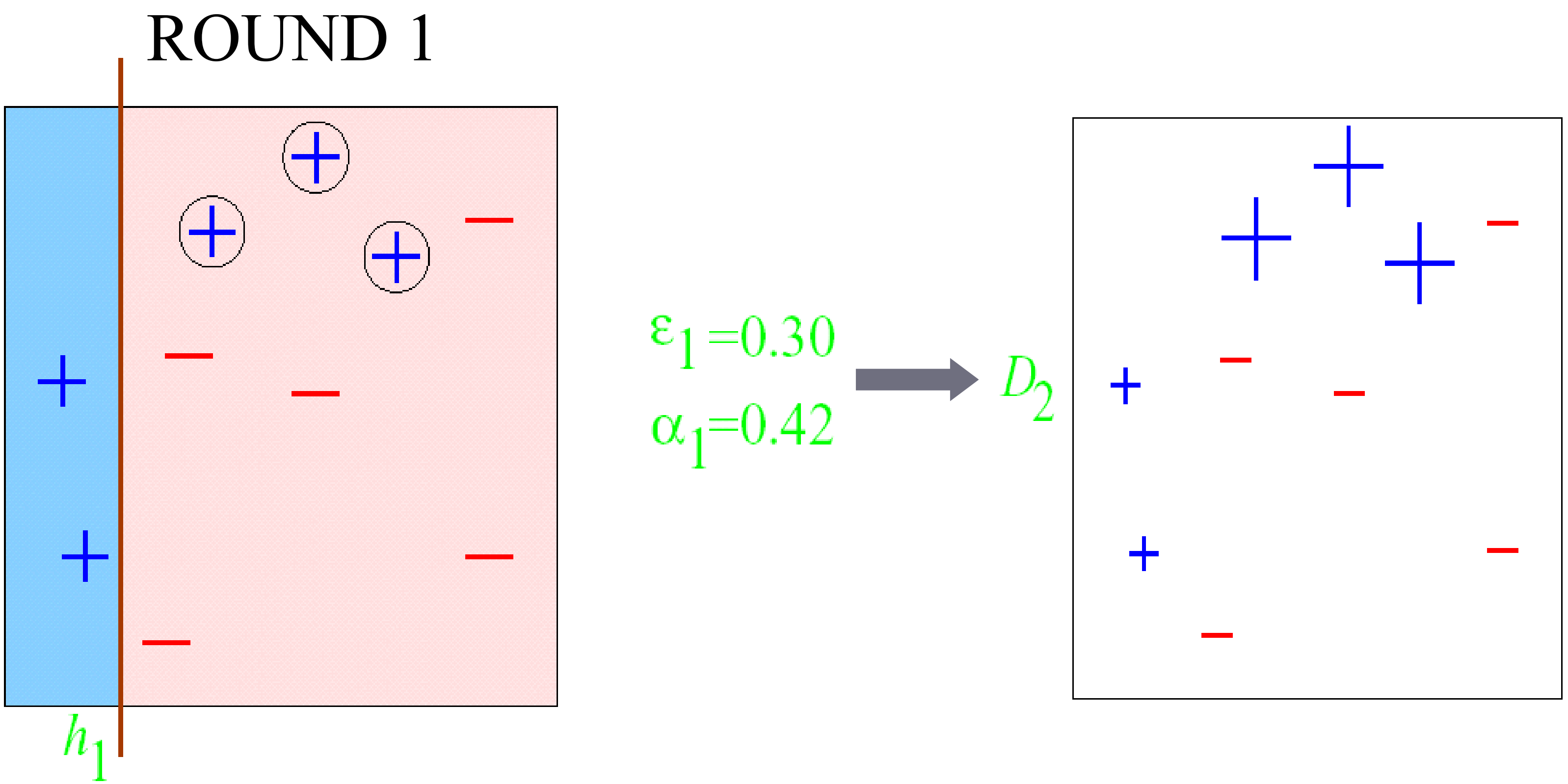
AdaBoost: example



Equal Weights to all training samples

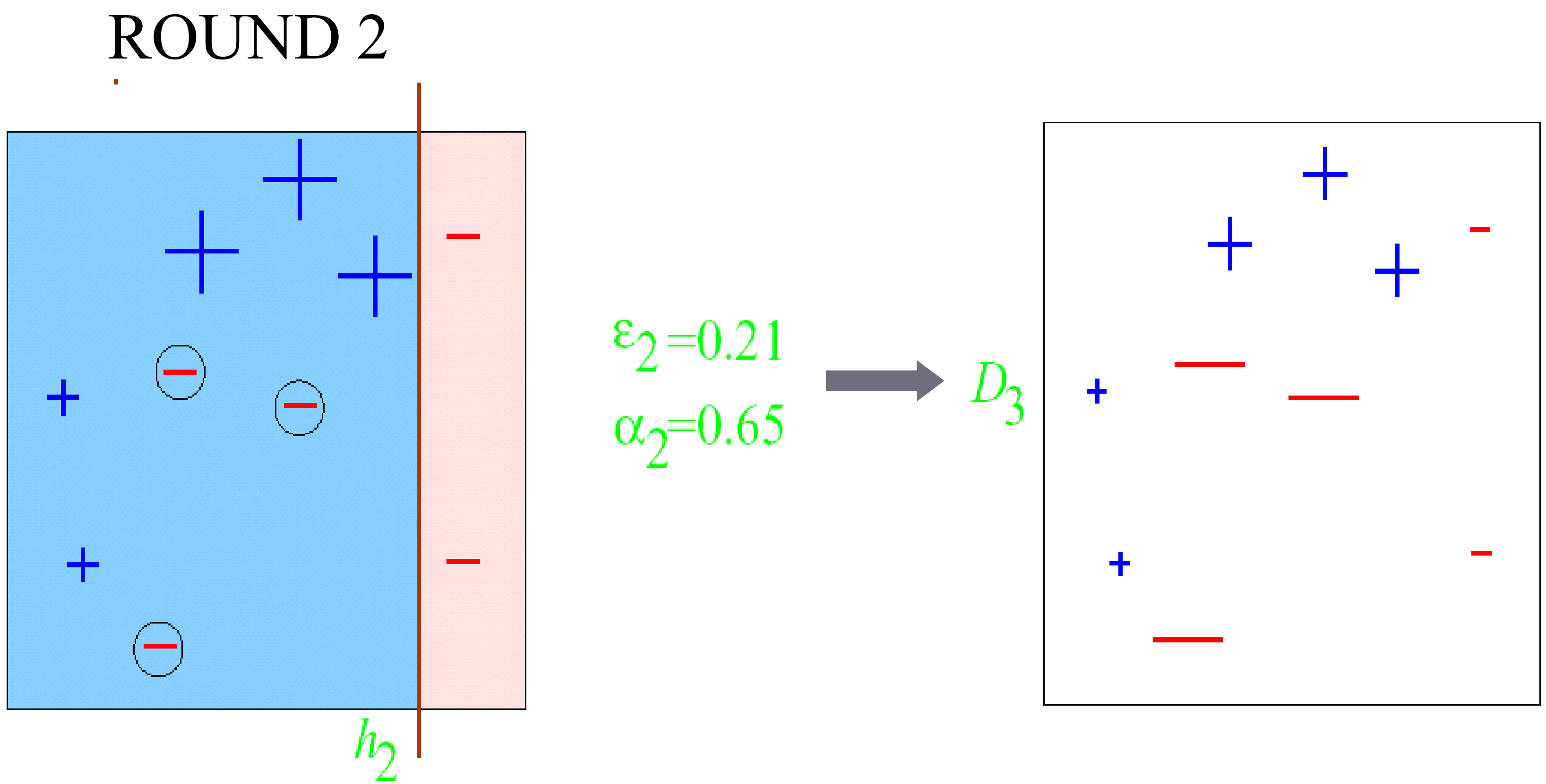
[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

AdaBoost: example



[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

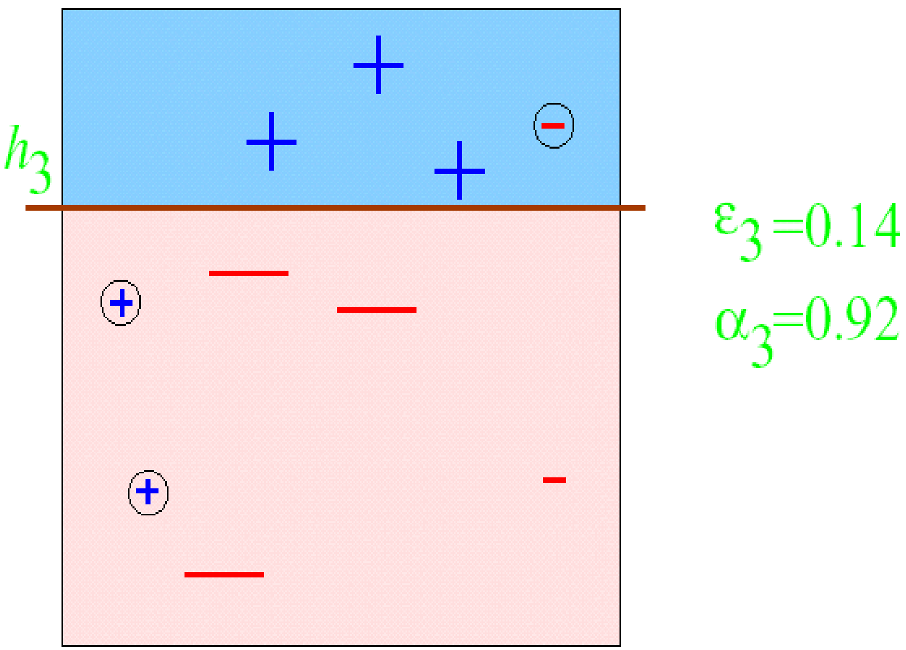
AdaBoost: example



[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

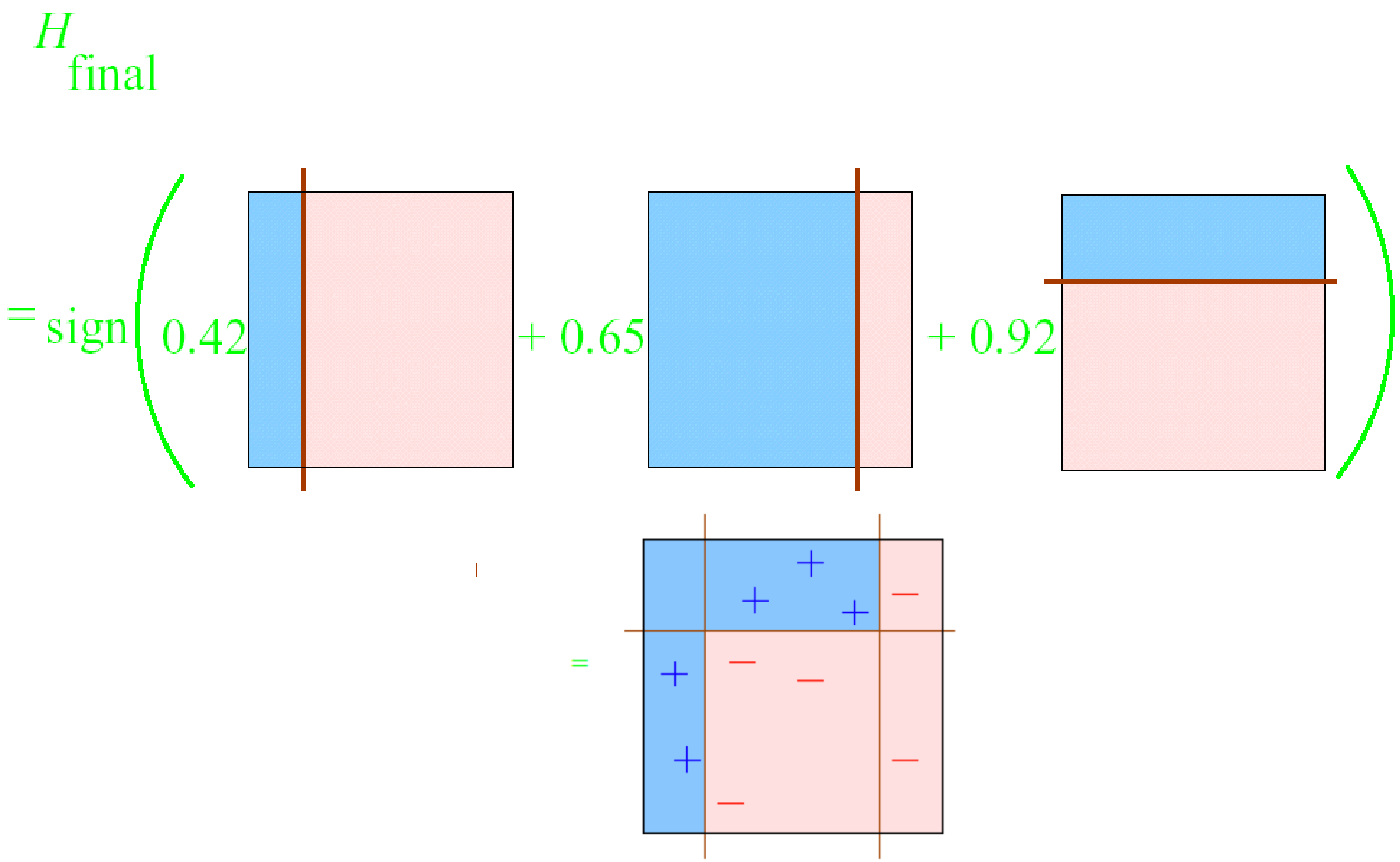
AdaBoost: example

ROUND 3



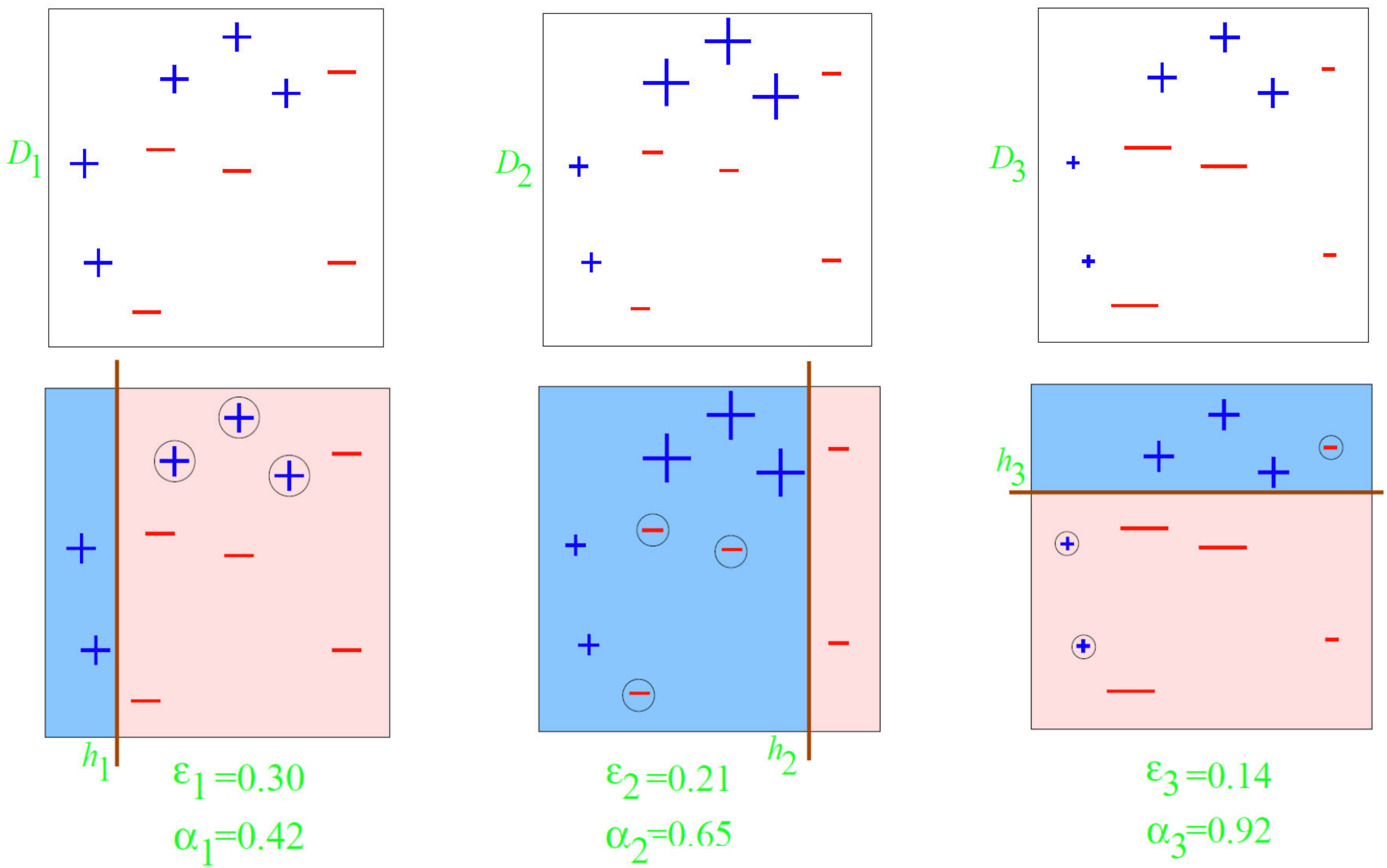
[Taken from "**A Tutorial on Boosting**" **by** Yoav Freund and Rob Schapire]

AdaBoost: example



[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

AdaBoost: example



[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

AdaBoost algorithm

Only when $h_t(\boldsymbol{x})$ with $\epsilon_t < 0.5$ (better than chance) is found, boosting continues.

1) For $i = 1$ to N Initialize the data weight $w_1^{(i)} = \frac{1}{N}$
2) For $t = 1$ to T
   a) Find a classifier $h_t(\boldsymbol{x})$ by minimizing the weighted error function:

$$J_t = \sum_{i=1}^{N} w_t^{(i)} \times I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)$$

   b) Find the weighted error of $h_t(\boldsymbol{x})$:

$$\epsilon_t = \frac{\sum_{i=1}^{N} w_t^{(i)} \times I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)}{\sum_{i=1}^{N} w_t^{(i)}}$$

   and the new component is assigned votes based on its error:

$$\alpha_t = \ln((1 - \epsilon_t)/\epsilon_t)$$

   c) The normalized weights are updated:

$$w_{t+1}^{(i)} = w_t^{(i)} \mathrm{e}^{\alpha_t I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)}$$

3) Combined classifier $\hat{y} = \operatorname{sign}\left(H_T(\boldsymbol{x})\right)$ where $H_T(\boldsymbol{x}) = \sum_{t=1}^{M} \alpha_t h_t(\boldsymbol{x})$

Notation explanation

- $w_t^{(i)}$: Weighting coefficient of data point i in iteration t
- α_t: weighting coefficient of t-th base classifier in the final ensemble
  - ϵ_t: weighted error rate of t-th base classifier

Boosting: main ideas

- Boosting algorithms maintain weights on training data:
  - Initially, all weights are equal, $w_1^{(i)} = 1/N$.
  - In t-th iteration, the weights are updated:
    - If $\boldsymbol{x}^{(i)}$ is misclassified by h_t, $w_t^{(i)}$ goes up;

- Fitting of h_{t+1} is guided by weights of samples
  - Force h_{t+1} to focus on already misclassified examples.

Adaptive boosting (AdaBoost): intuition

- First iteration: a usual procedure for training a single (weak) classifier

- Subsequent iterations:
  - $w_t^{(i)}$ is increased for misclassified data points
  - Then, successive classifiers are forced to place greater emphasis on points misclassified by previous classifiers

Boosting: loss function

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

$$H_t(\boldsymbol{x}) = \frac{1}{2}\left(\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_t h_t(\boldsymbol{x})\right)$$

- Many options for the loss function
  - AdaBoost is equivalent to using the following exponential loss

$$Loss(y, H_t(\boldsymbol{x})) = e^{-y \times H_t(\boldsymbol{x})}$$

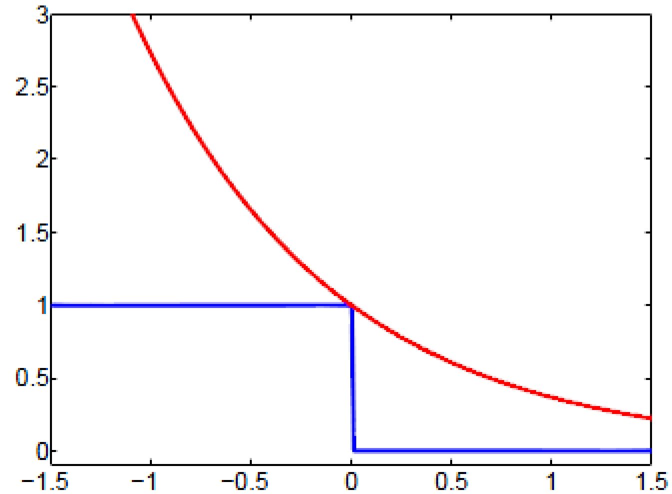
$$\hat{y} = sign(H_t(\boldsymbol{x}))$$

  - A simple interpretation of boosting in terms of the sequential minimization of the exponential loss function [Friedman *et al.*, 2000].

Boosting: exponential loss function

- Differentiable approximation (bound) of 0/1 loss
  - Easy to optimize
  - Optimizing an upper bound on classification error.

- Other options are possible.



AdaBoost: loss function

- Consider adding the t-th component:

$$E = \sum_{i=1}^{N} e^{-y^{(i)} H_t(\boldsymbol{x}^{(i)})} = \sum_{i=1}^{N} e^{-y^{(i)}[H_{t-1}(\boldsymbol{x}^{(i)}) + \frac{1}{2}\alpha_t h_t(\boldsymbol{x}^{(i)})]}$$

$$= \sum_{i=1}^{N} \underbrace{\mathrm{e}^{-y^{(i)} H_{t-1}(\boldsymbol{x}^{(i)})}}_{\text{Suppose it is fixed at stage t}} \, e^{-\frac{1}{2}\alpha_t y^{(i)} h_t(\boldsymbol{x}^{(i)})}$$

$$H_t(\boldsymbol{x}) = \frac{1}{2}[\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_t h_t(\boldsymbol{x})]$$

$$= \sum_{i=1}^{N} w_t^{(i)} \underbrace{\mathrm{e}^{-\frac{1}{2}\alpha_t y^{(i)} h_t(\boldsymbol{x}^{(i)})}}_{\text{Need to be optimized at stage } t \text{ by seeking } h_t(\boldsymbol{x}) \text{ and } \alpha_t}$$

$$w_t^{(i)} = \mathrm{e}^{-y^{(i)} H_{t-1}(\boldsymbol{x}^{(i)})}$$

Weighted exponential loss

$$E = \sum_{i=1}^{N} w_t^{(i)} e^{-\alpha_t y^{(i)} h_t(\boldsymbol{x}^{(i)})}$$

- sequentially adds a new component trained on reweighted training samples
- $w_t^{(i)}$: history of classification of $\boldsymbol{x}^{(i)}$ by H_{t-1}.
  - $\Rightarrow$ Loss weighted towards mistakes
- Iteration t optimization:
  - choose the new component $h_t = h(\boldsymbol{x}; \boldsymbol{\theta}_t)$
  - and the vote α_t that optimizes the weighted exponential loss.

Minimizing loss: finding h_t

$$E = \sum_{i=1}^{N} w_t^{(i)} e^{-\frac{1}{2}\alpha_t y^{(i)} h_t(\boldsymbol{x}^{(i)})}$$

$$= e^{\frac{-\alpha_t}{2}} \sum_{y^{(i)}=h_t(\boldsymbol{x}^{(i)})} w_t^{(i)} + e^{\frac{\alpha_t}{2}} \sum_{y^{(i)}\neq h_t(\boldsymbol{x}^{(i)})} w_t^{(i)}$$

$$= \left(e^{\frac{\alpha_t}{2}} - e^{\frac{-\alpha_t}{2}}\right) \underbrace{\sum_{y^{(i)}\neq h_t(\boldsymbol{x}^{(i)})} w_t^{(i)}}_{J_t = \sum_{i=1}^{N} w_t^{(i)} \times I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)} + e^{\frac{-\alpha_t}{2}} \sum_{i=1}^{N} w_t^{(i)}$$

Find $h_t(\boldsymbol{x})$ that minimizes J_t

Minimizing loss: finding α_m

$$\frac{\partial E}{\partial \alpha_t} = 0$$

$$\Rightarrow \frac{1}{2}\left(e^{\frac{\alpha_t}{2}} + e^{\frac{-\alpha_t}{2}}\right) \sum_{y^{(i)} \neq h_t(\boldsymbol{x}^{(i)})} w_t^{(i)} - \frac{1}{2} e^{\frac{-\alpha_t}{2}} \sum_{i=1}^{N} w_t^{(i)} = 0$$

$$\Rightarrow \frac{e^{\frac{-\alpha_t}{2}}}{\left(e^{\frac{\alpha_t}{2}} + e^{\frac{-\alpha_t}{2}}\right)} = \frac{\sum_{y^{(i)} \neq h_t(\boldsymbol{x}^{(i)})} w_t^{(i)}}{\sum_{i=1}^{N} w_t^{(i)}}$$

$$\alpha_t = \ln((1 - \epsilon_t)/\epsilon_t)$$

$$\epsilon_t = \frac{\sum_{i=1}^{N} w_t^{(i)} I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)}{\sum_{i=1}^{N} w_t^{(i)}}$$

Updating weights

- Updating weights in AdaBoost algorithm:

$$w_{t+1}^{(i)} = w_t^{(i)} \mathrm{e}^{-\frac{1}{2}\alpha_t y^{(i)} h_t\left(\boldsymbol{x}^{(i)}\right)}$$

$$= w_t^{(i)} \underbrace{\mathrm{e}^{-\frac{1}{2}\alpha_t}}_{\text{Independent of } i \text{ and can be ignored}} \mathrm{e}^{\alpha_t I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)}$$

$$y^{(i)} h_t\left(\boldsymbol{x}^{(i)}\right) = 1 - 2I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)$$

$$\Rightarrow w_{t+1}^{(i)} = w_t^{(i)} \mathrm{e}^{\alpha_t I\left(y^{(i)} \neq h_t\left(\boldsymbol{x}^{(i)}\right)\right)}$$

AdaBoost algorithm: summary

1) For $i = 1$ to N Initialize the data weight $w_1^{(i)} = \frac{1}{N}$
2) For $t = 1$ to T
   a) Find a classifier $h_t(\boldsymbol{x})$ by minimizing the weighted error function
   b) Find the normalized weighted error of $h_t(\boldsymbol{x})$ as ϵ_t
   c) Compute the new component weight as α_t
   d) Update example weights for the next iteration $w_{t+1}^{(i)}$
3) Combined classifier $\hat{y} = \text{sign}\big(H_T(\boldsymbol{x})\big)$ where $H_T(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$

Bias-variance trade-off

- Weak or simple learners
  - Low variance: they don't usually overfit
  - High bias: they can't usually learn complex functions

- Boosting to decrease the bias
  - boost weak learners to enhance their capabilities

- Bagging to decrease the variance

How to train base learners

- Base learners used in practice:
  - Decision stumps
  - Decision trees
  - Multi-layer neural networks

- Can base learners operate on weighted examples?
  - In many cases they can be modified to accept weights along with the examples
  - In general, we can sample the examples (with replacement) according to the distribution defined by the weights

AdaBoost: typical behavior

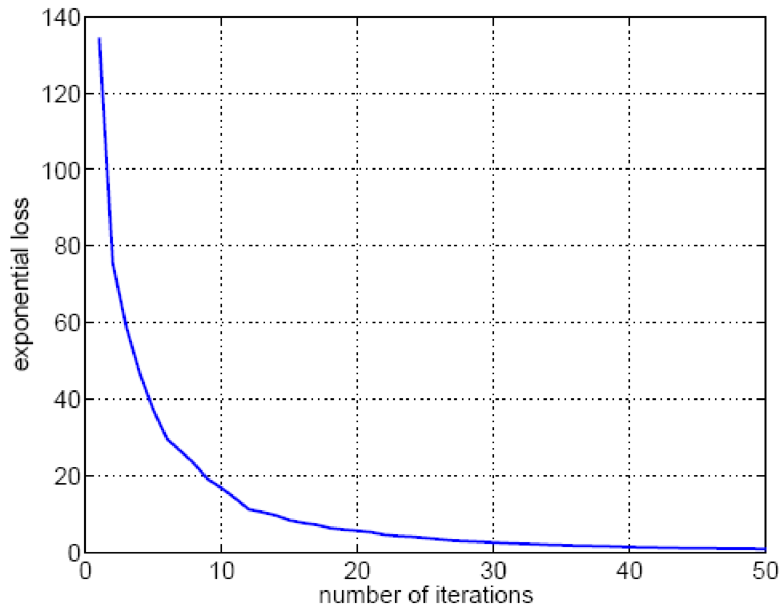
- Exponential loss goes strictly down.
- Training error of H goes down
- Weighted error ϵ_t goes up $\Rightarrow$ votes α_t go down.

AdaBoost properties: exponential loss

- In each boosting iteration, assuming we can find $h(\boldsymbol{x};\widehat{\boldsymbol{\theta}}_t)$ whose weighted error is better than chance.

$$H_t(\boldsymbol{x}) = \frac{1}{2}\left[\hat{\alpha}_1 h(\boldsymbol{x};\widehat{\boldsymbol{\theta}}_1) + \ldots + \hat{\alpha}_t h(\boldsymbol{x};\widehat{\boldsymbol{\theta}}_t)\right]$$

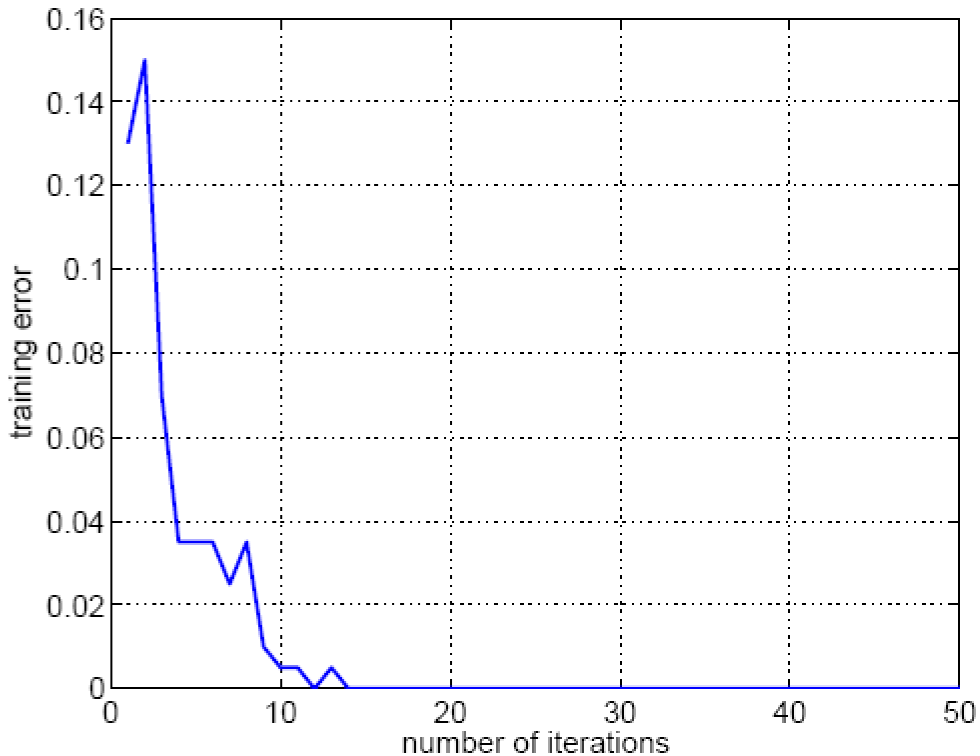
  - Thus, lower exponential loss over training data is guaranteed.



$$E = \sum_{i=1}^{N} e^{-y^{(i)} H_t(\boldsymbol{x}^{(i)})}$$

AdaBoost properties: training error

- Boosting iterations typically decrease the classification error of $H_t(\boldsymbol{x})$ over training examples.

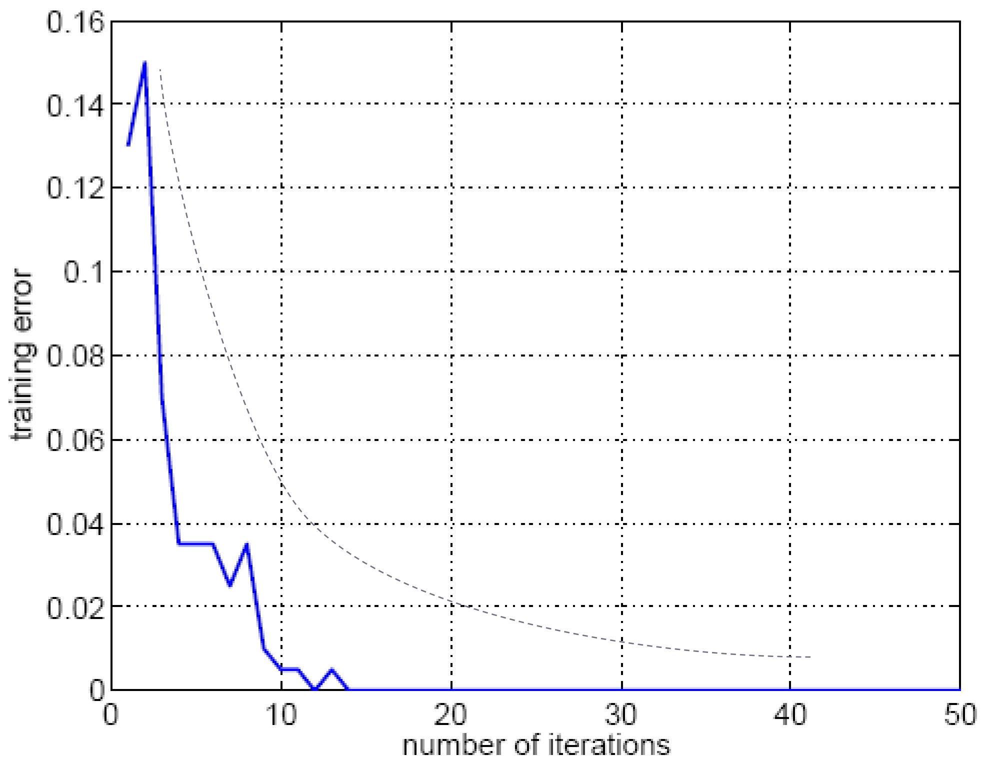


adopted from Prof. Jaakolla's slides, Machine Learning course, MIT

AdaBoost properties: training error

- Training error has to go down exponentially fast if the weighted error of each h_t is strictly better than chance (i.e., $\epsilon_t < 0.5$)

Training error in t-th iteration is bounded by an exponential function a^t ($0 < a < 1$)

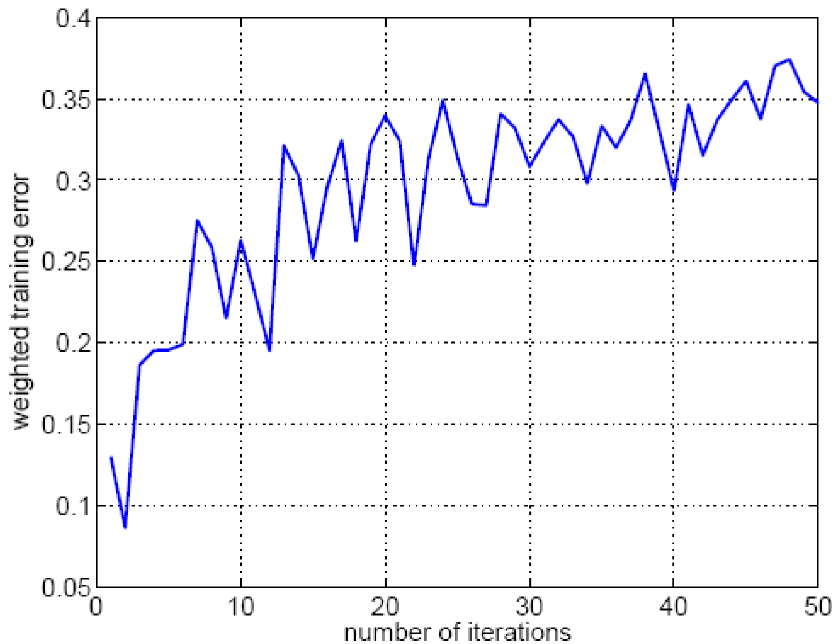


AdaBoost properties: weighted error

- Weighted error of each new component classifier

$$\epsilon_m = \frac{\sum_{i=1}^{n} w_m^{(i)} I\left(y^{(i)} \neq h_m\left(\boldsymbol{x}^{(i)}\right)\right)}{\sum_{i=1}^{n} w_m^{(i)}}$$

tends to increase as a function of boosting iterations.



adopted from Prof. Jaakolla's slides, Machine Learning course, MIT

Bound on training error:

Theorem: Error of h_t over D_t: $\epsilon_t = \frac{1}{2} - \gamma_t$

$$E_{train}(H_T) \leq e^{-2\sum_{t=1}^{T} \gamma_t^2}$$

Thus, if $\forall t, \gamma_t \geq \gamma > 0$ then $E_{train}(H_T) \leq e^{-2\gamma^2 T}$

- Training error decreases exponentially in T

Updates & Normalization

- Claim: D_{t+1} puts half of the weight on samples on which h_t was incorrect and other half on samples on which h_t was correct

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\frac{1}{2}\alpha_t y^{(i)} h_t(x^{(i)})}$$

$$\Pr_{D_{t+1}}\left[y^{(i)} \neq h_t\left(x^{(i)}\right)\right] = \sum_{i:y^{(i)} \neq h_t(x^{(i)})} \frac{D_t(i)}{Z_t} e^{\frac{1}{2}\alpha_t} = \frac{\epsilon_t}{Z_t} e^{\frac{1}{2}\alpha_t} = \frac{\epsilon_t}{Z_t}\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \frac{1}{Z_t}\sqrt{\epsilon_t(1-\epsilon_t)}$$

$$\Pr_{D_{t+1}}\left[y^{(i)} = h_t\left(x^{(i)}\right)\right] = \sum_{i:y^{(i)} = h_t(x^{(i)})} \frac{D_t(i)}{Z_t} e^{-\frac{1}{2}\alpha_t} = \frac{1-\epsilon_t}{Z_t} e^{-\frac{1}{2}\alpha_t} = \frac{1-\epsilon_t}{Z_t}\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} = \frac{1}{Z_t}\sqrt{\epsilon_t(1-\epsilon_t)}$$

$$Z_t = \sum_i D_t(i) e^{-\frac{1}{2}\alpha_t y^{(i)} h_t(x^{(i)})} = \sum_{i:y^{(i)} \neq h_t(x^{(i)})} D_t(i) e^{\frac{1}{2}\alpha_t} + \sum_{i:y^{(i)} = h_t(x^{(i)})} D_t(i) e^{-\frac{1}{2}\alpha_t}$$

$$= (1-\epsilon_t) e^{-\frac{1}{2}\alpha_t} + \epsilon_t e^{\frac{1}{2}\alpha_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

Bound on Training error

- **Step 1:** $D_{T+1}(i) = \frac{1}{N}\left(\frac{e^{-y^{(i)}H_T\left(x^{(i)}\right)}}{\prod_{t=1}^{T} Z_t}\right)$

- **Step 2**: $E_{train}(H_T) \leq \prod_{t=1}^{T} Z_t$

- **Step 3**: $\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2}$
  $\leq e^{-2\sum_{t=1}^{T} \gamma_t^2}$

$1 - x \leq e^{-x}$
For $0 \leq x \leq 1$

Bound on Training error

- Step 1: $D_{T+1}(i) = \frac{1}{N}\left(\frac{e^{-y^{(i)}H_T\left(x^{(i)}\right)}}{\prod_{t=1}^{T} Z_t}\right)$ $\qquad H_t(\boldsymbol{x}) = \frac{1}{2}[\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_t h_t(\boldsymbol{x})]$
- Proof:

$$D_1(i) = \frac{1}{N}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\frac{1}{2}\alpha_t y^{(i)} h_t(x^{(i)})}$$

$$D_{T+1}(i) = \frac{e^{-\frac{1}{2}\alpha_T y^{(i)} h_T(x^{(i)})}}{Z_T} D_T(i)$$

$$= \frac{e^{-\frac{1}{2}\alpha_T y^{(i)} h_T(x^{(i)})}}{Z_T} \times \frac{e^{-\frac{1}{2}\alpha_{T-1} y^{(i)} h_{T-1}(x^{(i)})}}{Z_{T-1}} D_{T-1}(i)$$

$$= \frac{e^{-\frac{1}{2}\alpha_T y^{(i)} h_T(x^{(i)})}}{Z_T} \times \cdots \times \frac{e^{-\frac{1}{2}\alpha_1 y^{(i)} h_1(x^{(i)})}}{Z_1} D_1(i)$$

$$\frac{1}{N}\left(\frac{e^{-\frac{1}{2}\left(\alpha_1 y^{(i)} h_1\left(x^{(i)}\right) + \cdots + \alpha_T y^{(i)} h_T\left(x^{(i)}\right)\right)}}{\prod_{t=1}^{T} Z_t}\right) = \frac{1}{N}\left(\frac{e^{-y^{(i)}H_T\left(x^{(i)}\right)}}{\prod_{t=1}^{T} Z_t}\right)$$

Bound on Training error

- **Step 1**: $D_{T+1}(i) = \frac{1}{N}\left(\frac{e^{-y^{(i)}H_T\left(x^{(i)}\right)}}{\prod_{t=1}^{T} Z_t}\right)$
- **Step 2**: $E_{train}(H_T) \le \prod_{t=1}^{T} Z_t$

Proof:

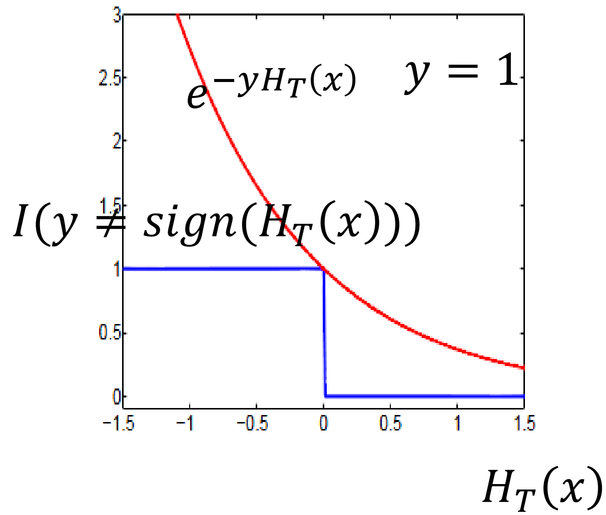
$$E_{train}(H_T) = \frac{1}{N}\sum_i I\left(y^{(i)} \neq sign\left(H_T\left(x^{(i)}\right)\right)\right)$$

$$= \frac{1}{N}\sum_i I\left(y^{(i)}H_T\left(x^{(i)}\right) \le 0\right)$$

$$\le \frac{1}{N}\sum_i e^{-y^{(i)}H_T\left(x^{(i)}\right)}$$

$$= \sum_i D_{T+1}\left(x^{(i)}\right)\prod_{t=1}^{T} Z_t$$

$$= \prod_{t=1}^{T} Z_t$$



Bound on Training error

- **Step 1**: $D_{T+1}(i) = \frac{1}{N}\left(\frac{e^{-y^{(i)}H_T\left(x^{(i)}\right)}}{\prod_{t=1}^{T} Z_t}\right)$
- **Step 2**: $E_{train}(H_T) \leq \prod_{t=1}^{T} Z_t$
- **Step 3:** $\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2}$
$\leq e^{-2\sum_{t=1}^{T} \gamma_t^2}$

Error of h_t over D_t

$\epsilon_t = \frac{1}{2} - \gamma_t$

Recall: $Z_t = (1-\epsilon_t)e^{-\frac{1}{2}\alpha_t} + \epsilon_t e^{\frac{1}{2}\alpha_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$

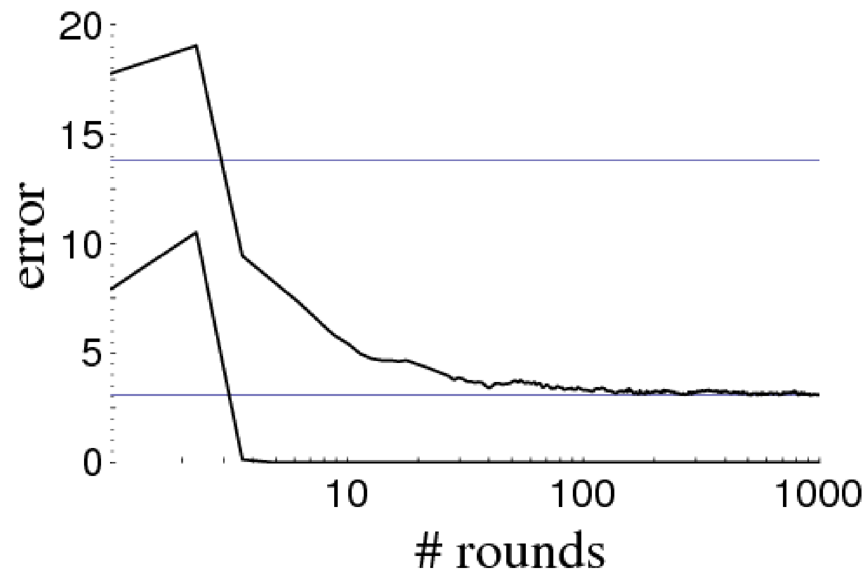
Boosting and overfitting

- Boosting is **often robust to overfitting**
  - But not always
    - may easily overfit in the presence of labeling noise or overlap of classes

- Test set error decreases even after training error is zero

Training and test error

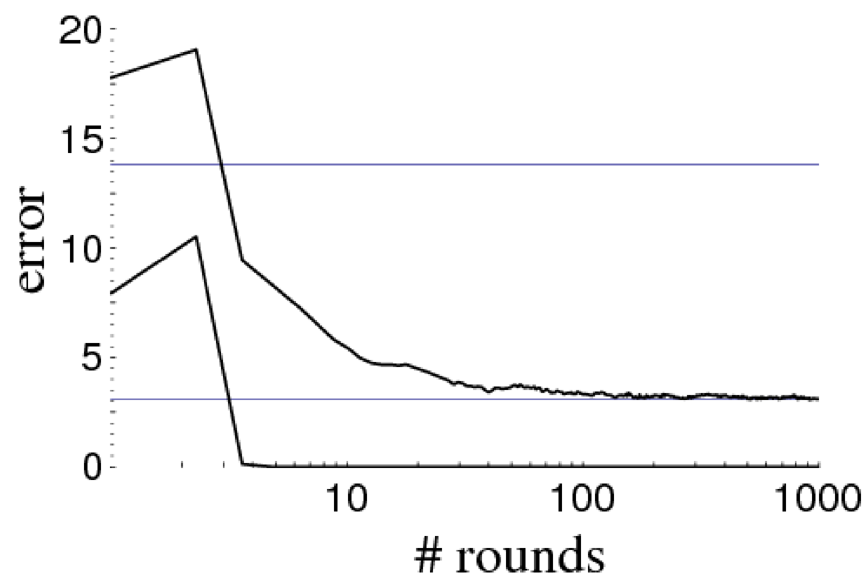
- Test error usually does not increase as the number of base classifiers becomes very large.



Robert E. Schapire, The Boosting Approach to Machine Learning, 2001.

AdaBoost: test error

- Continuing to add new weak learners after achieving zero training error could even decrease test error!



Robert E. Schapire, The Boosting Approach to Machine Learning, 2001.

Generalization Error Bounds

- $E_{true}(H) \leq E_{train}(H) + \tilde{O}\left(\sqrt{\frac{\boldsymbol{T}d}{N}}\right)$

With high probability

[Freund & Schapire'95]

T: number of boosting rounds
d: VC dimension of weak learner, measures complexity of classifier
N: number of training examples

- Is not consistent with experimental results
- The bound is too loose
- Margin-based bounds as better analysis

AdaBoost and margin

- Combined classifier in a more useful form:

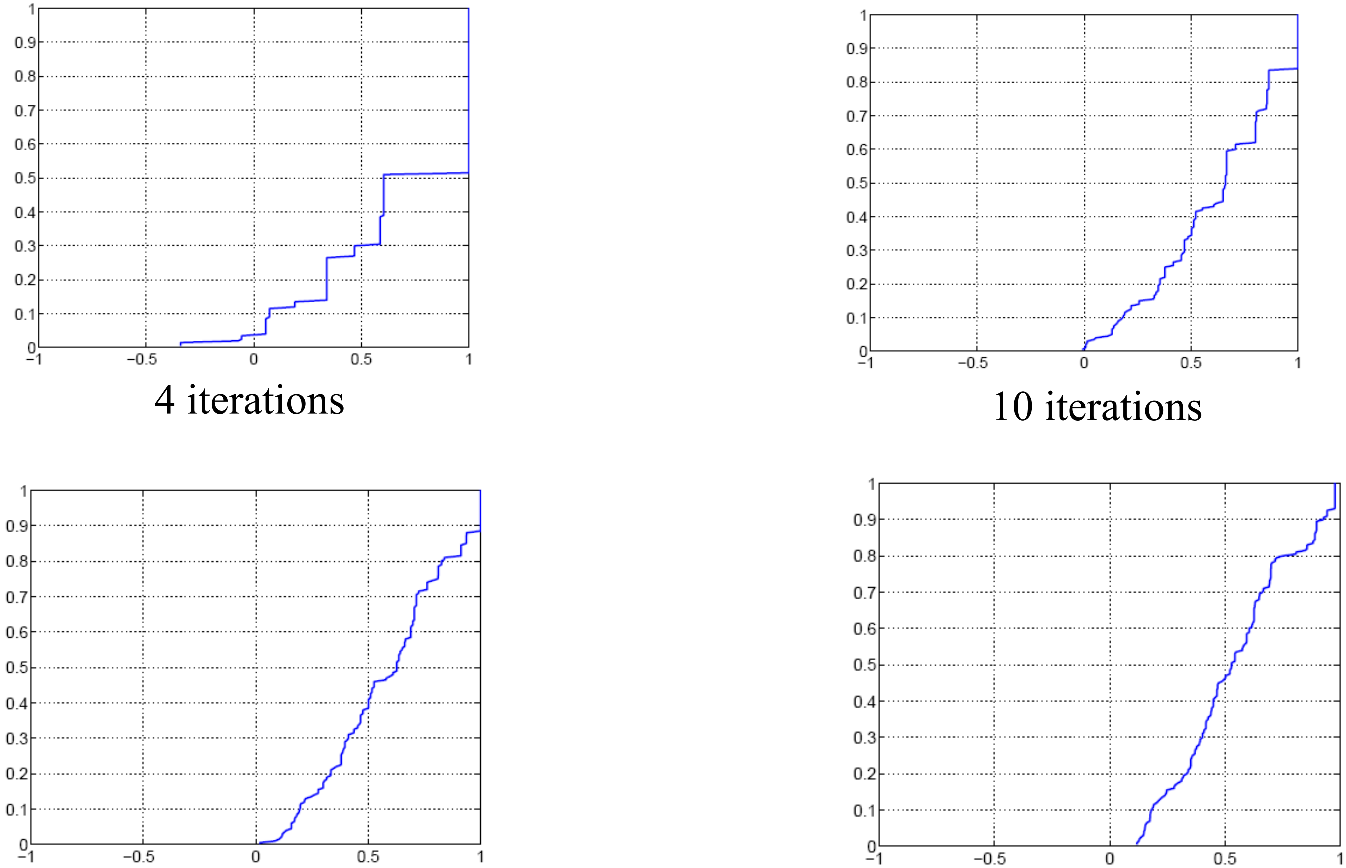
$$H_t(\boldsymbol{x}) = \frac{\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_t h_t(\boldsymbol{x})}{\alpha_1 + \ldots + \alpha_t}$$

- This allows us to define a margin:

$$\text{margin}(\boldsymbol{x}_i) = y^{(i)} H_t(\boldsymbol{x}^{(i)})$$

  - margin lies in $[-1,1]$ and is negative for misclassified examples.
    - a measure of confidence in the correct decision
- Margin of training examples is increased during iterations
  - Even for correct classification can further improve confidence.

Cumulative distributions of margin values



4 iterations

10 iterations

20 iterations

50 iterations

adopted from Prof. Jaakolla's slides, Machine Learning course, MIT

Adaboost and margin

- When a combined classifier is used, the more classifier **agreeing**, the more **confident** you are in your prediction.

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

A Margin Bound

- For any γ, the generalization error is less than:

$$P_{(\boldsymbol{x},y)\sim D}(yH_T(\boldsymbol{x}) \leq 0\,)$$
$$\leq P_{(x,y)\sim S}(yH_T(\boldsymbol{x}) \leq \gamma\,) + O\left(\sqrt{\frac{d}{N\gamma^2}}\right)$$

- It does not depend on *T*.

$$margin(\boldsymbol{x}, y) = yH_T(\boldsymbol{x})$$
$$H_t(\boldsymbol{x}) = \frac{\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_t h_t(\boldsymbol{x})}{\alpha_1 + \ldots + \alpha_t}$$

D: distribution on $(\boldsymbol{x}, y)$

S: a set of i.i.d. training samples from D

Robert E. Schapire et. al, Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651-1686, 1998.

Bagging & Boosting: Summary

- Bagging
  - Uses **bootstrap** sampling to construct several training sets from the original training set and then **aggregate the learners** trained on these datasets
  - Bagging **reduces the variance** of high variance learners (e.g. decision tree)

- Boosting
  - Combines many **"weak" classifiers** in sequence to find a single "strong" classifier
    - In each iteration, changes the **distribution** of data to emphasis the samples that have been **misclassified** by the previous learner

Resources

- C. Bishop, "Pattern Recognition and Machine Learning", Chapter 14.2-14.3.

- Robert E. Schapire, The Boosting Approach to Machine Learning, 2001.

- Robert E. Schapire et. al, Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651-1686, 1998.