Ensemble learning

CE-477: Machine Learning - CS-828: Theory of Machine Learning
Sharif University of Technology
Fall 2024

Fatemeh Seyyedsalehi

References of the lecture are mentioned in the last slide

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 learning

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

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

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 $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \dots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$

- **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.

- $H(\boldsymbol{x}) = \mathrm{sign}(\sum_{t=1}^{T} h_t(\boldsymbol{x}))$
  - 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

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

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

- 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

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

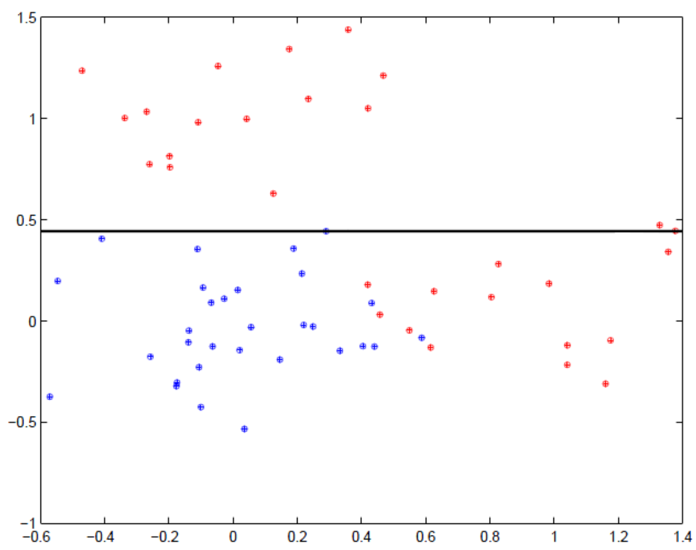
- Prediction: $\hat{y} = sign(H_t(\boldsymbol{x}))$

Simple component classifiers

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

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

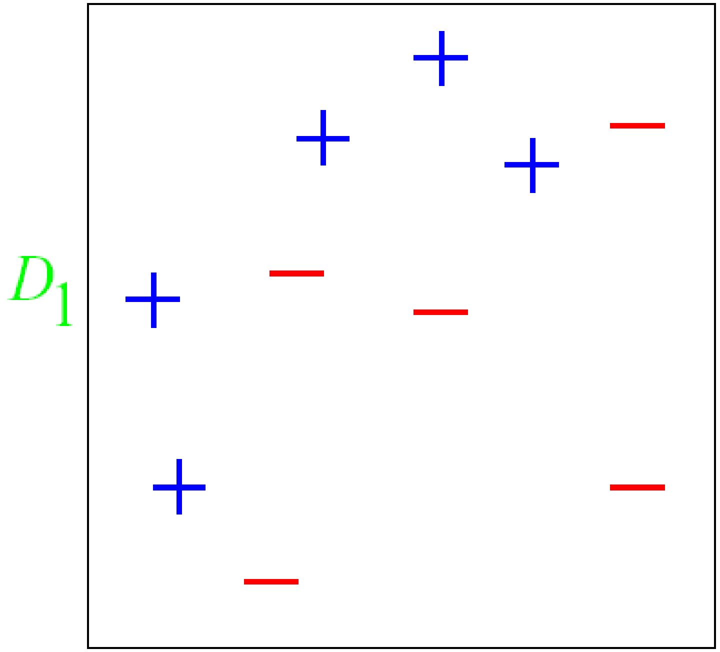
- Each classifier is based on only a single feature of $\boldsymbol{x}$ (e.g., x_k): decision tree of depth one



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

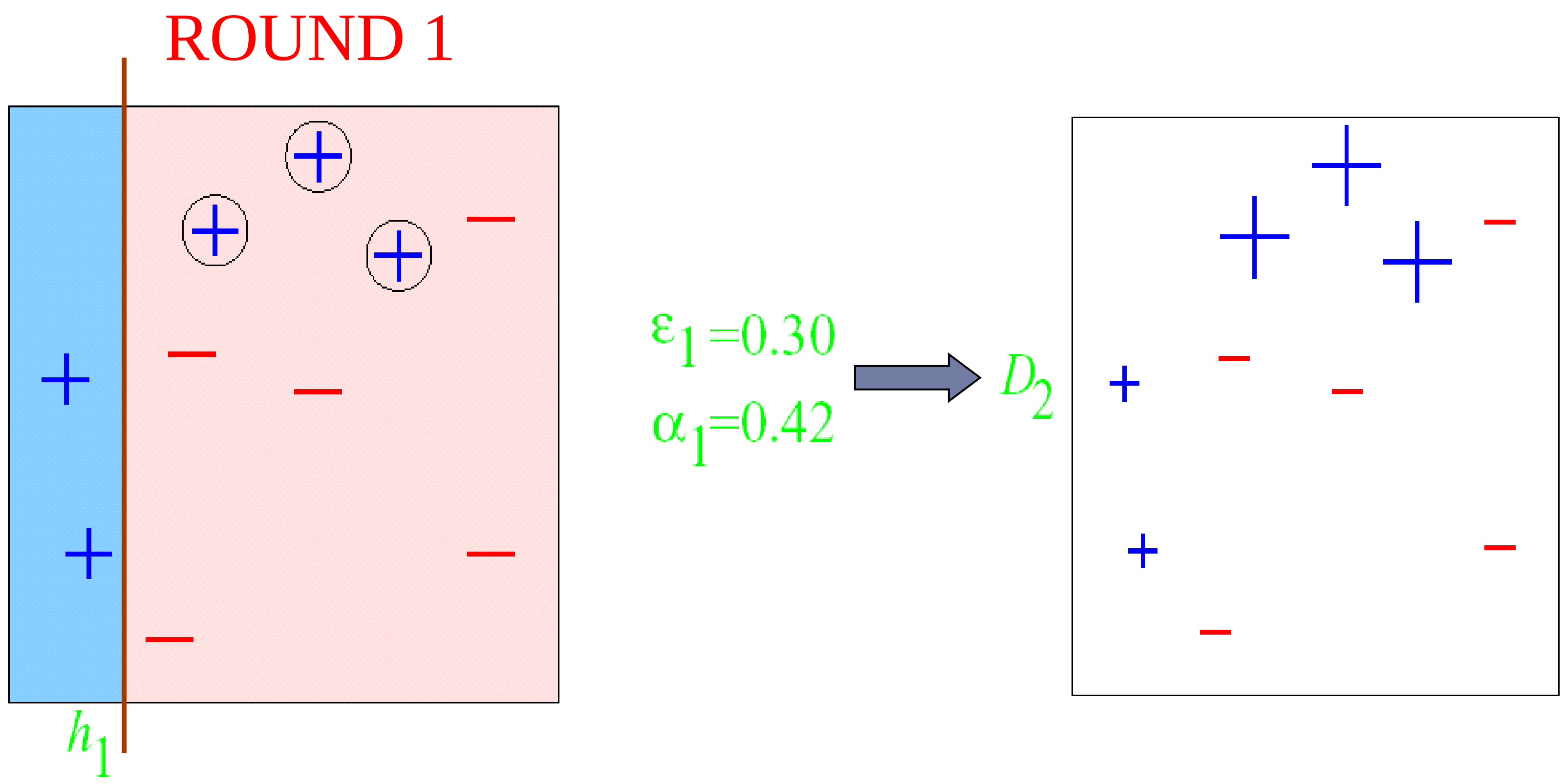
AdaBoost: example



Equal Weights to all training samples

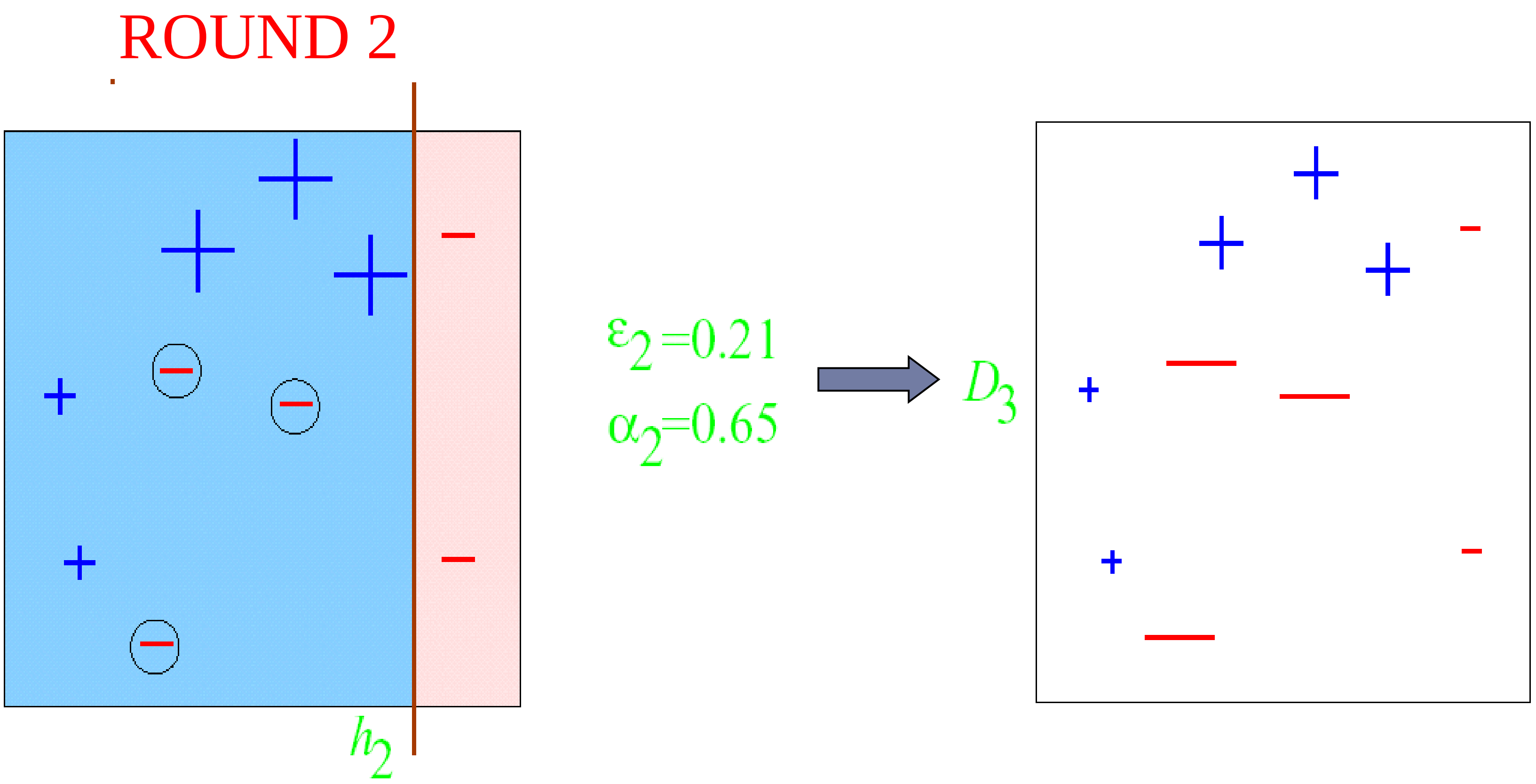
[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

AdaBoost: example



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

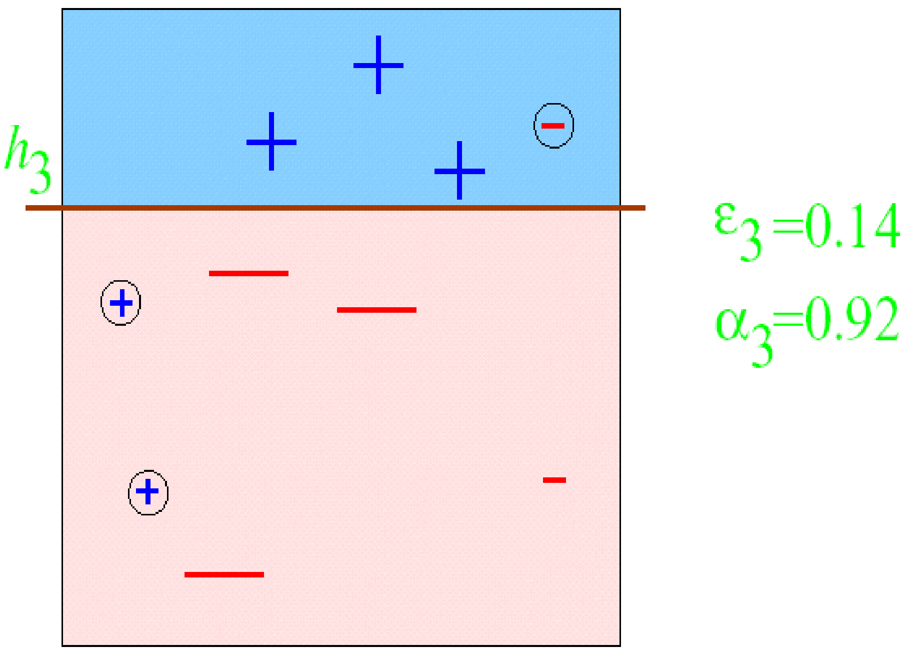
AdaBoost: example



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

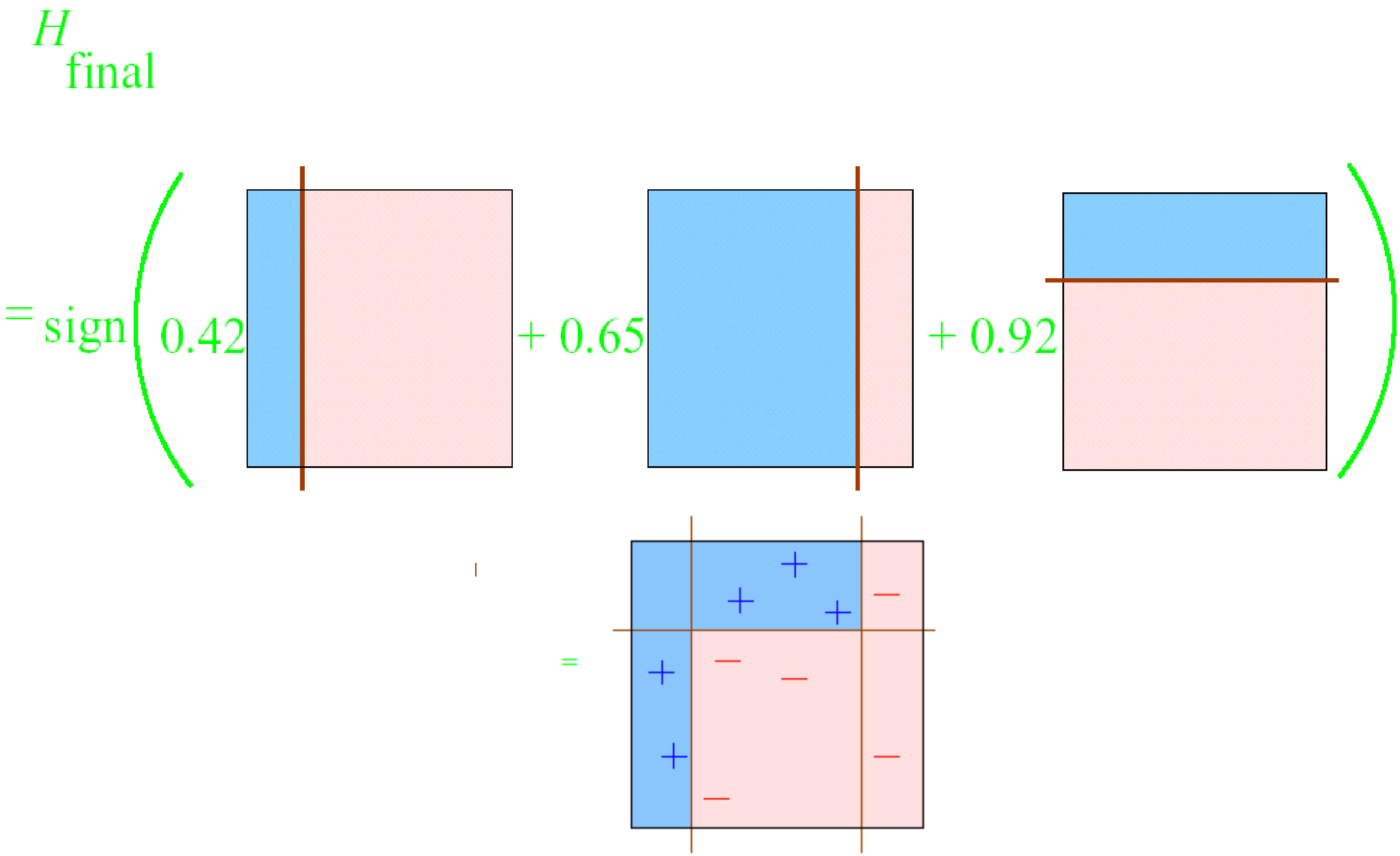
AdaBoost: example

ROUND 3



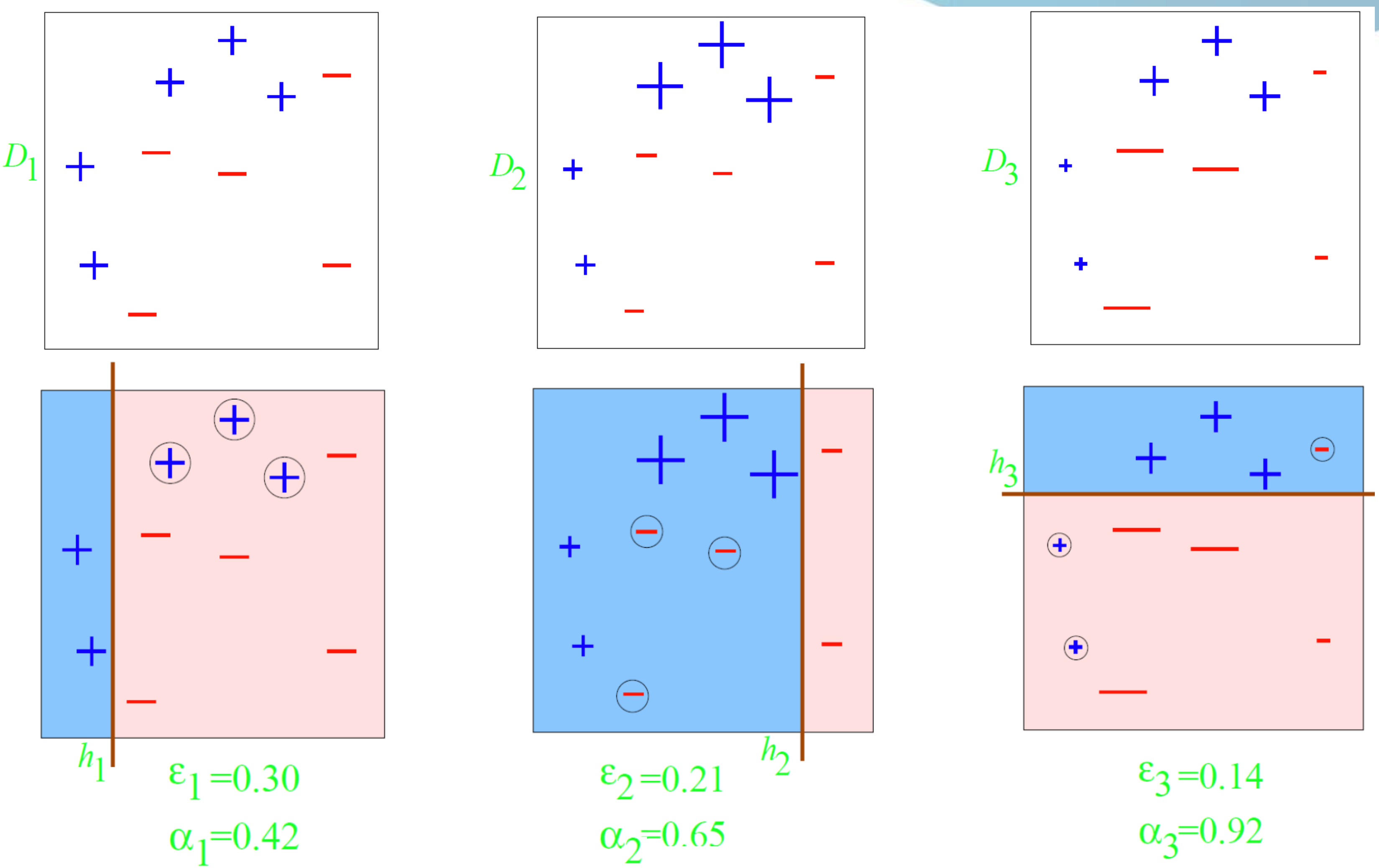
[Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire]

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

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:

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

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

   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 $\hat{y} = \mathrm{sign}\left(H_T(\boldsymbol{x})\right)$ 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

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

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

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

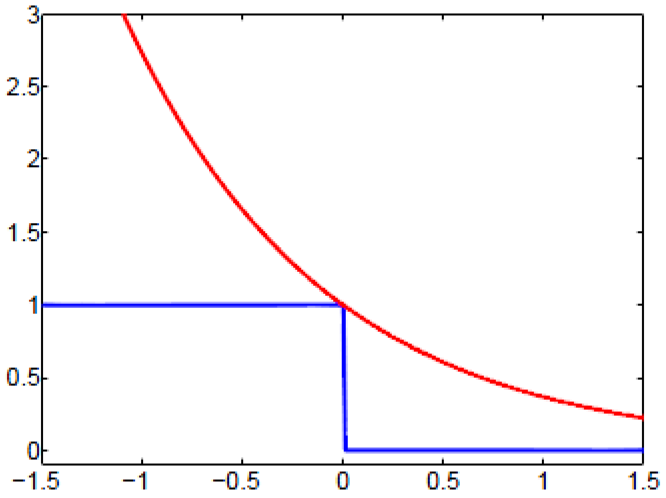
$$\hat{y} = sign(H_t(\boldsymbol{x}))$$

  - 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.



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

AdaBoost: loss function

- Consider adding the t-th component:

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

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

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

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

Weighted exponential loss

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

- 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

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

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

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

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

Minimizing loss: finding α_m

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

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

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

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

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

Updating weights

- Updating weights in AdaBoost algorithm:

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

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

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

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

Another perspective for AdaBoost

Define a uniform distribution $D_1(i)$ over elements of S.

for $t = 1$ to T **do**

Train a model h_t using distribution D_t.

Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$

If $\epsilon_t \geq 0.5$ break

Set $\alpha_t = \quad \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

end for

For a new testing point (x', y'),

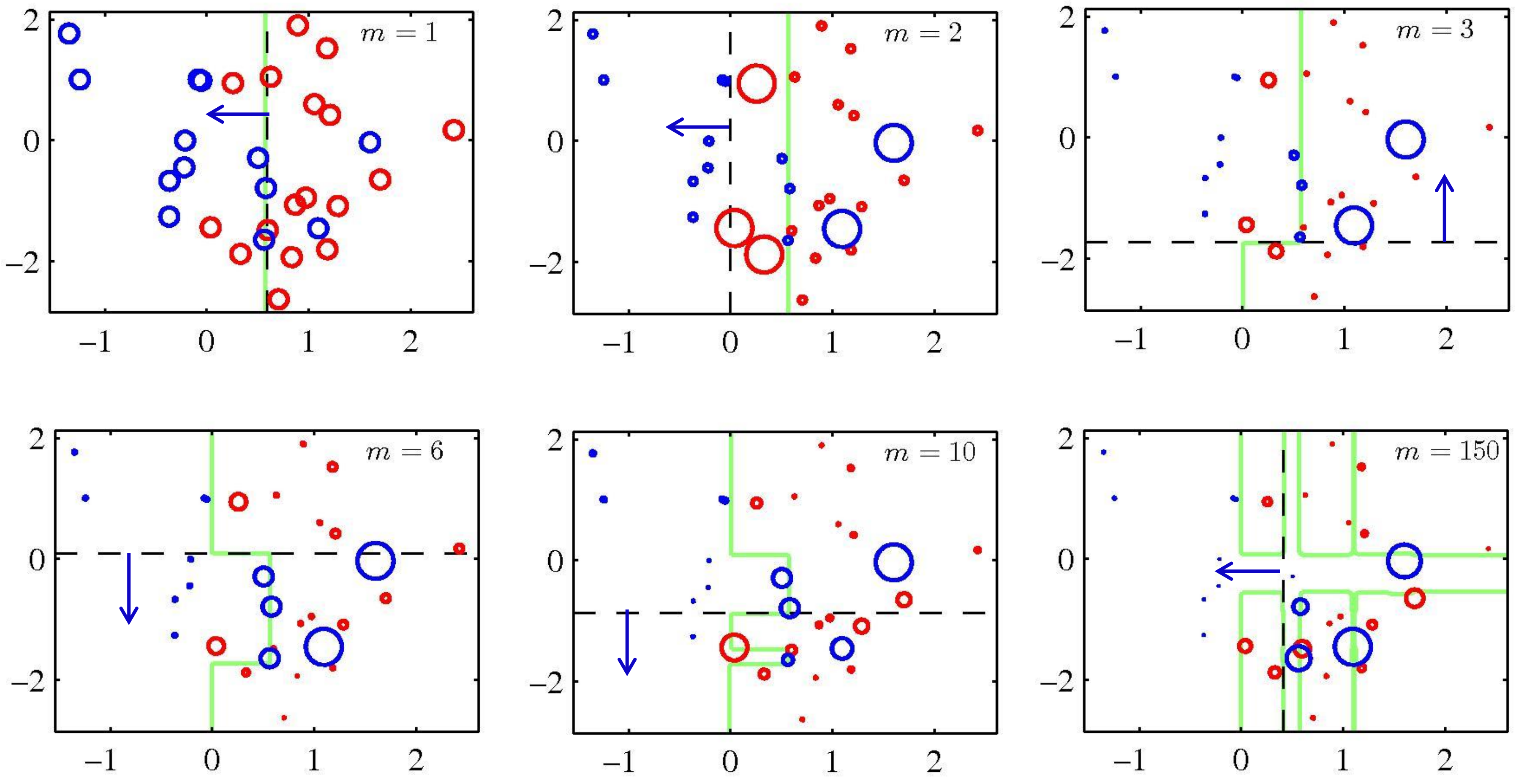
$H(x') = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x'))$

$$H_T(\boldsymbol{x}) = \frac{1}{2}[\alpha_1 h_1(\boldsymbol{x}) + \ldots + \alpha_T h_T(\boldsymbol{x})]$$

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 $\hat{y} = \text{sign}\left(H_T(\boldsymbol{x})\right)$ where $H_T(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$

AdaBoost: example 2



[Bishop]

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

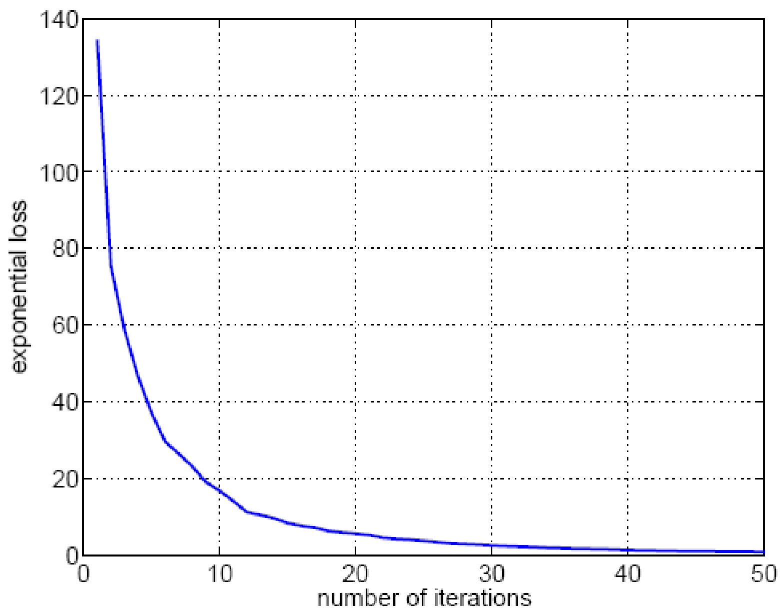
- Exponential loss goes strictly down.
- Training error of H goes down
- Weighted error ϵ_t goes up $\Rightarrow$ votes α_t go down.

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.

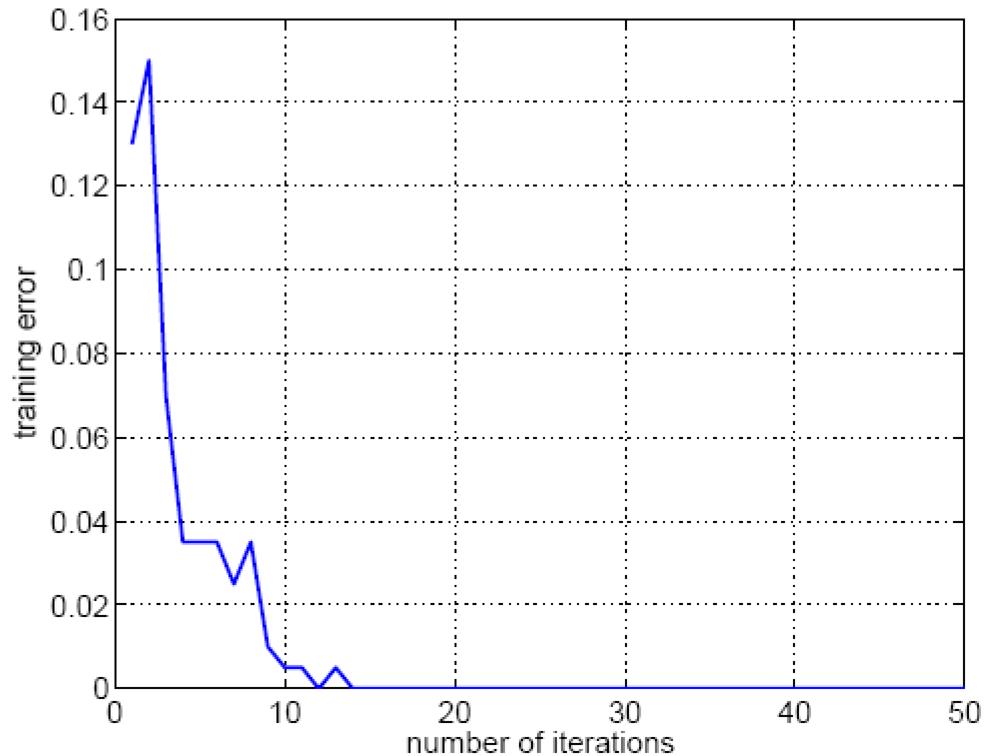


$$E = \sum_{i=1}^{N} e^{-y^{(i)} H_t(\boldsymbol{x}^{(i)})}$$

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

AdaBoost properties: training error

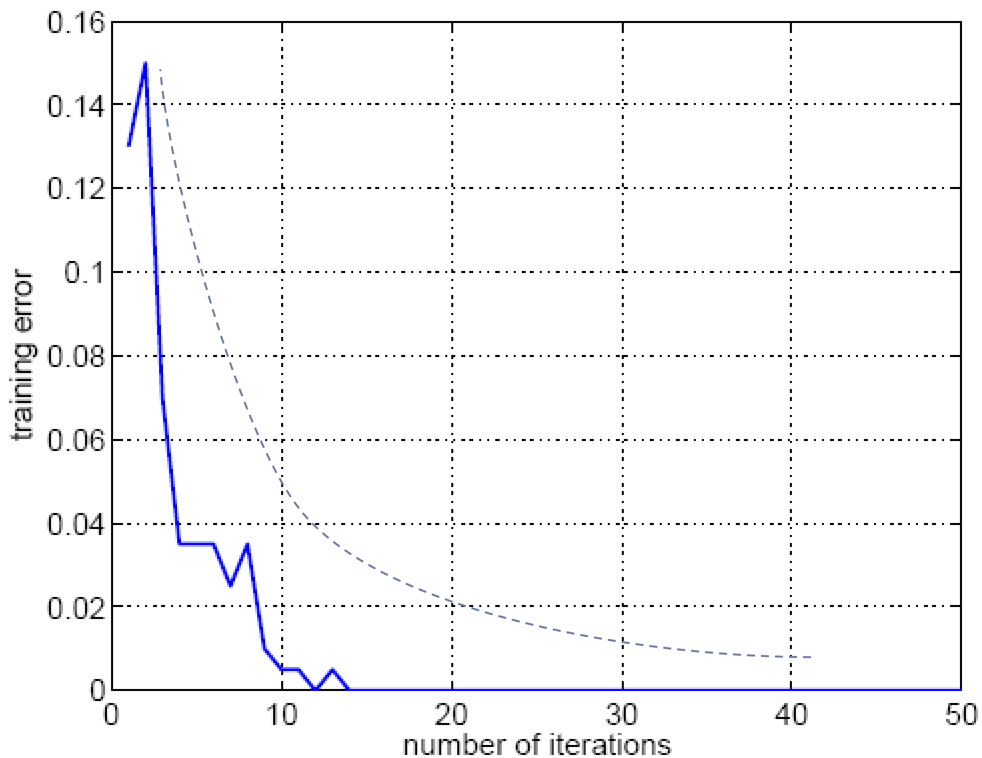
- Boosting iterations typically decrease the classification error of $H_t(\boldsymbol{x})$ over training examples.



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., $\epsilon_t < 0.5$)

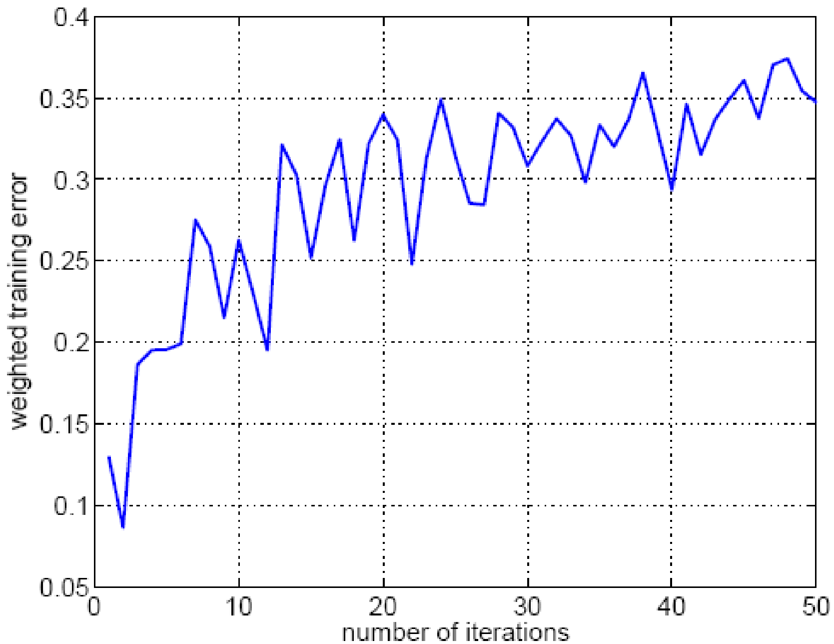


AdaBoost properties: weighted error

- Weighted error of each new component classifier

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

tends to increase as a function of boosting iterations.



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

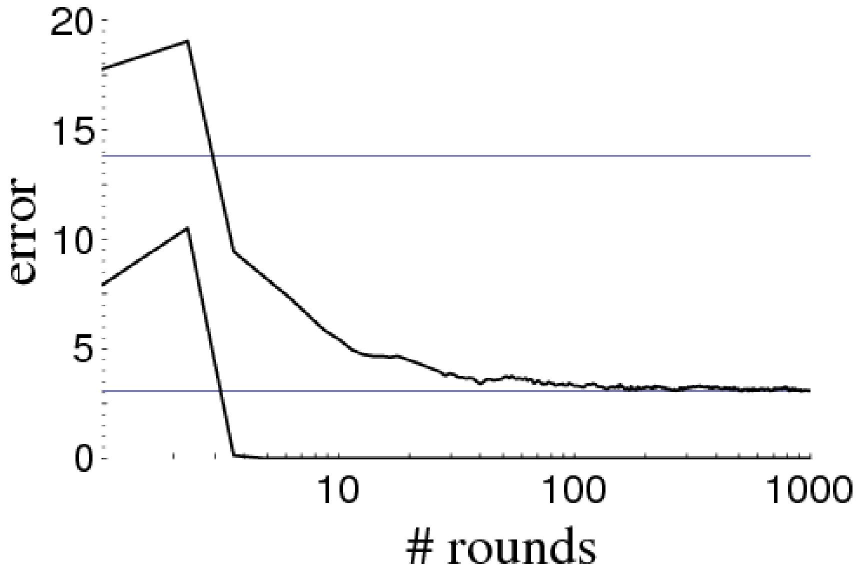
Boosting and overfitting

- However, 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

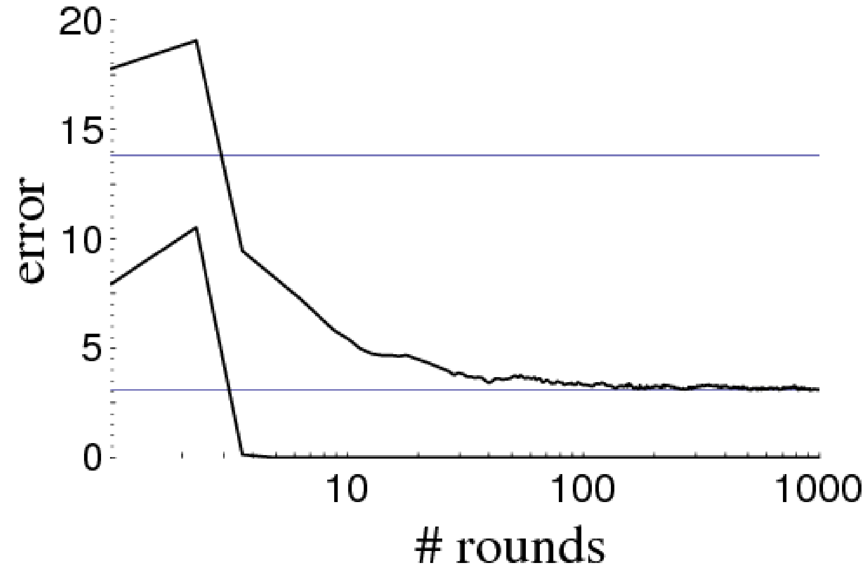
- Test error usually does not increase as the number of base classifiers becomes very large.



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.

AdaBoost and margin

- Combined classifier in a more useful form:

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

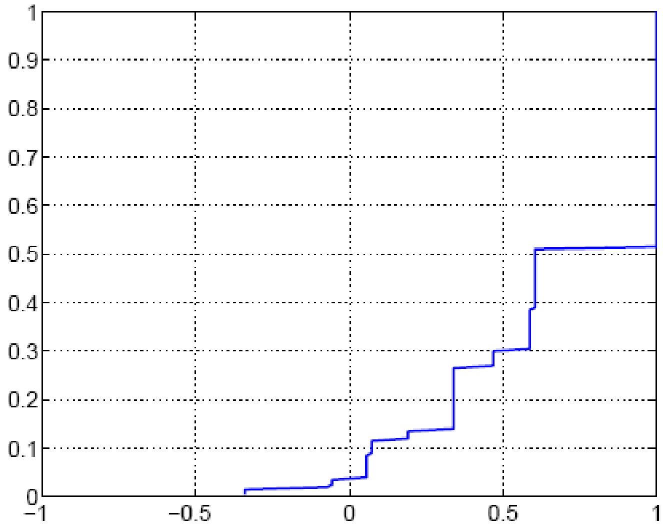
- This allows us to define a margin:

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

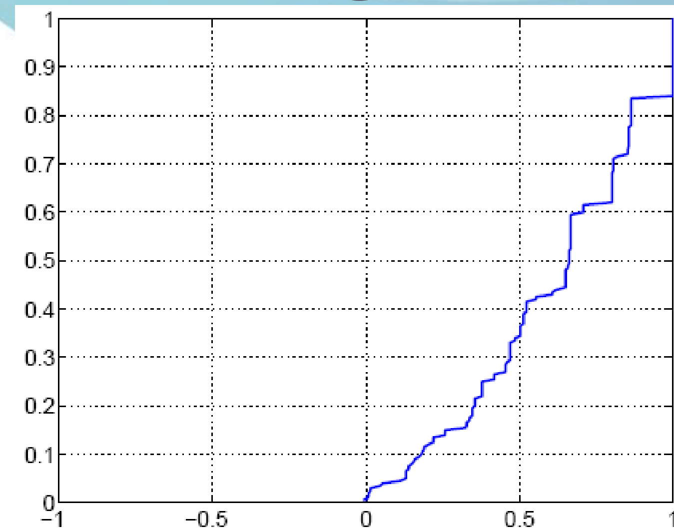
  - margin lies in $[-1,1]$ and is negative for misclassified examples.
    - a measure of confidence in the correct decision

- Margin of training examples is increased during iterations
  - Even for correct classification can further improve confidence.

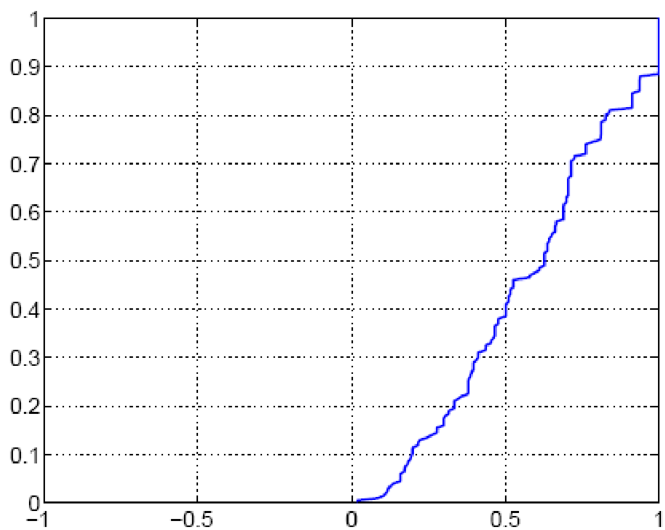
Cumulative distributions of margin values



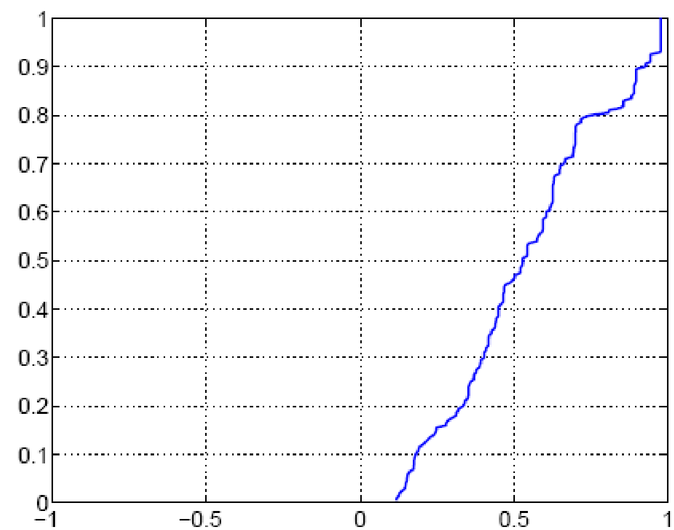
4 iterations



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

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

- Mahdieh Soleymani – Machine learning – Sharif university