University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chenhao Tan

Lecture 13: Boosting

Slides adapted from Jordan Boyd-Graber, Chris Ketelsen

Learning objectives

- Understand the general idea behind ensembling
- Learn about Adaboost
- Learn the math behind boosting

Ensemble methods

- We have learned
  - KNN
  - Naïve Bayes
  - Logistic regression
  - Neural networks
  - Support vector machines
- Why use a single model?

Ensemble methods

- Bagging
  - Train classifiers on subsets of data
  - Predict based on majority vote
- Stacking
  - Take multiple classifiers' outputs as inputs and train another classifier to make final prediction

Boosting intuition

- Boosting is an ensemble method, but with a different twist
- Idea:
  - Build a sequence of *dumb models*
  - Modify training data along the way to focus on difficult to classify examples
  - Predict based on weighted majority vote of all the models
- Challenges
  - What do we mean by dumb?
  - How do we promote difficult examples?
  - Which models get more say in vote?

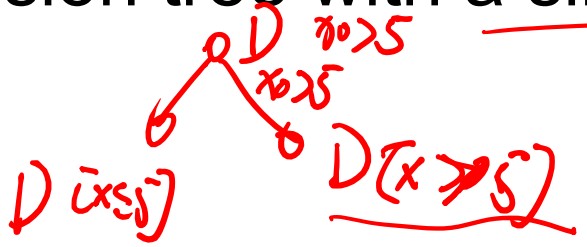
Boosting intuition

- What do we mean by dumb?
- Each model in our sequence will be a weak learner

A model $h(\mathbf{x})$ is a weak learner if its training error is just under 50

$$\text{err} = \frac{1}{m}\sum_{i=1}^{m} I(y_i \neq h(\mathbf{x}_i) = \frac{1}{2} - \gamma$$

- Most common weak learner in Boosting is a *decision stump* - a decision tree with a single split



Boosting intuition

- How do we promote difficult examples?
- After each iteration, we'll increase the importance of training examples that we got wrong on the previous iteration and decrease the importance of examples that we got right on the previous iteration
- Each example will carry around a weight w_i that will play into the decision stump and the error estimation Weights are normalized so they act like a probability distribution

$$\sum_{i=1}^{m} w_i = 1$$

Boosting intuition

- Which models get *more say* in vote?
- The models that performed better on training data get more say in the vote

For our sequence of weak learners: $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_K(\mathbf{x})$
Boosted classifier defined by

$$H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$$

Weight α_k is measure of accuracy of h_k on training data

The Plan

- Learn Adaboost
- Unpack it for intuition
- Come back later and show the math

The Plan

- Learn Adaboost
- Unpack it for intuition
- Come back later and show the match

Usual binary classification assumptions:

- Training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Feature vector $\mathbf{x} \in \mathbb{R}^d$
- Labels are $y_i \in \{-1, 1\}$

Adaboost

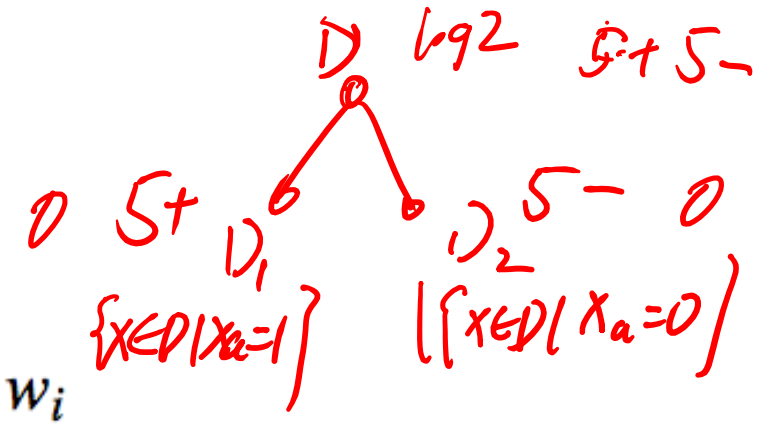
1. Initialize data weights to $w_i = \frac{1}{m}, i = 1, \ldots, m$
2. For $k = 1$ to K:
   (a) Fit classifier $h_k(\mathbf{x})$ to training data with weights w_i
   (b) Compute weighted error $\text{err}_k = \frac{\sum_{i=1}^{m} w_i I(y_i \neq h_k(\mathbf{x}_i))}{\sum_{i=1}^{m} w_i}$
   (c) Compute $\alpha_k = \frac{1}{2} \log((1 - \text{err}_k)/\text{err}_k)$
   (d) Set $w_i \leftarrow \frac{w_i}{Z_k} \cdot \exp[-\alpha_k y_i h_k(\mathbf{x}_i)], i = 1, \ldots, m$
3. Output $H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$

Adaboost

1 . Initialize data weights to $w_i = \frac{1}{m}, i = 1, \ldots, m$

Weights are initialized to uniform distribution. Every training example counts equally on first iteration.

Adaboost



2a. Fit classifier $h_k(\mathbf{x})$ to training data with weights w_i
Decide split based on info gain with weighted entropy:

$$H(D) = -p \log_2 p - (1-p) \log_2 (1-p)$$

$$I(D,a) = H(D) - \sum_{v \in vals(a)} \frac{|\{x \in D | x_a = v\}|}{|D|} H(\{x \in D | x_a = v\})$$

$$p = \frac{\sum_{i=1}^{m} w_i \cdot I(y_i = 1)}{\sum_{i=1}^{m} w_i \cdot I(y_i = \pm 1)}$$

Adaboost

2b. Compute weighted error

$$\text{err}_k = \frac{\sum_{i=1}^{m} w_i I(y_i \neq h_k(\mathbf{x}_i))}{\sum_{i=1}^{m} w_i}$$

Still gives error rate in $[0,\ 1]$

Mistakes on highly weighted examples hurt more

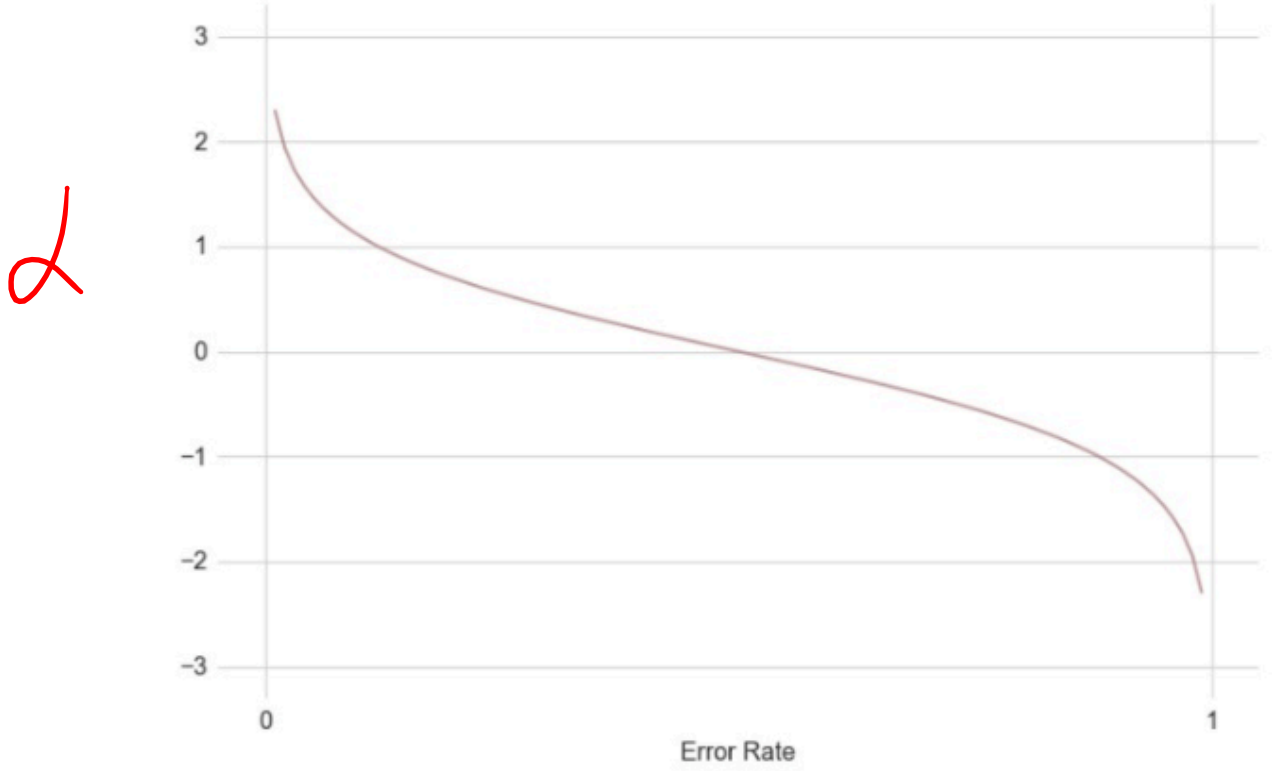
Mistakes on lowly weighted examples don't register too much

Adaboost

2c. Compute $\alpha_k = \frac{1}{2}\log((1-\text{err}_k)/\text{err}_k)$

Models with small err get promoted (exponentially)

Models with large err get demoted (exponentially)



Adaboost

$y_i \in \{-1, +1\}$

$h_k(x_i)$

2d. Set $w_i \leftarrow \frac{w_i}{Z_k} \cdot \exp[-\alpha_k y_i h_k(\mathbf{x}_i)], \; i = 1, \ldots, m$

$\alpha_k \geq 0$

- If example was misclassified weight goes up
- If example was classified correctly weight goes down
- How big of a jump depends on accuracy of model

$y_i h_k = -1$

1

- Do we need to compute Z_k?

No!

$w_i \leftarrow w_i \cdot \exp(-\alpha_k y_i h_k)$

$w_i \leftarrow \frac{w_i}{\text{sum}(w_i)}$

$Z_k = \sum_i w_i \cdot \exp(-\alpha_k y_i h_k(x_i))$

Adaboost

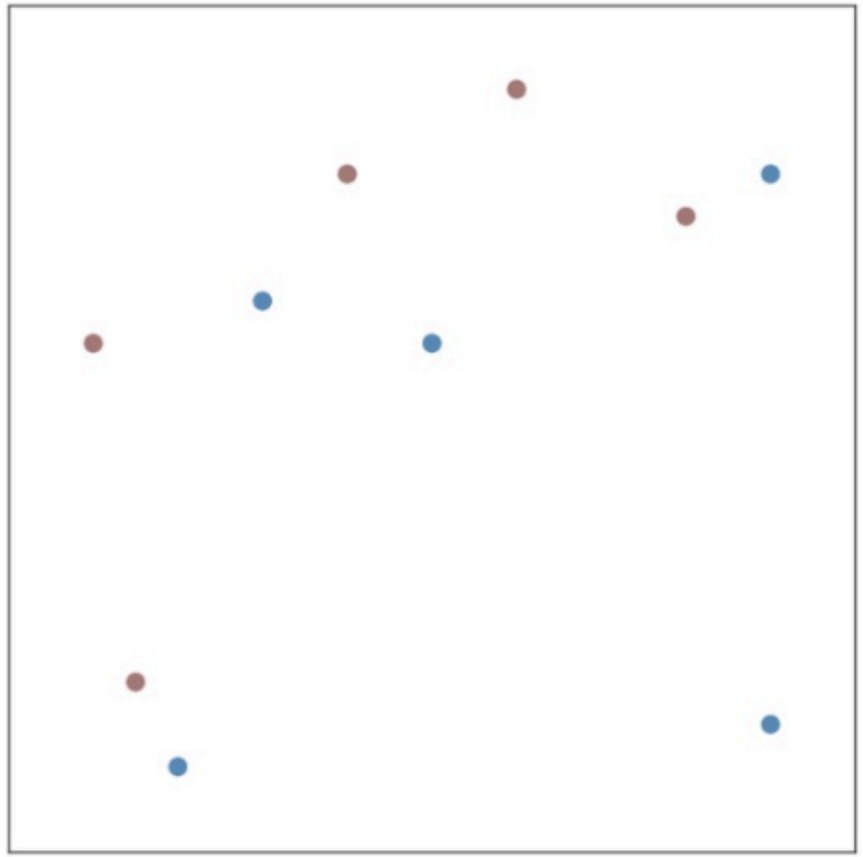
3 . Output $H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$

Sum up weighted votes from each model

Classify $y = 1$ if positive and $y = -1$ if negative

Adaboost Example

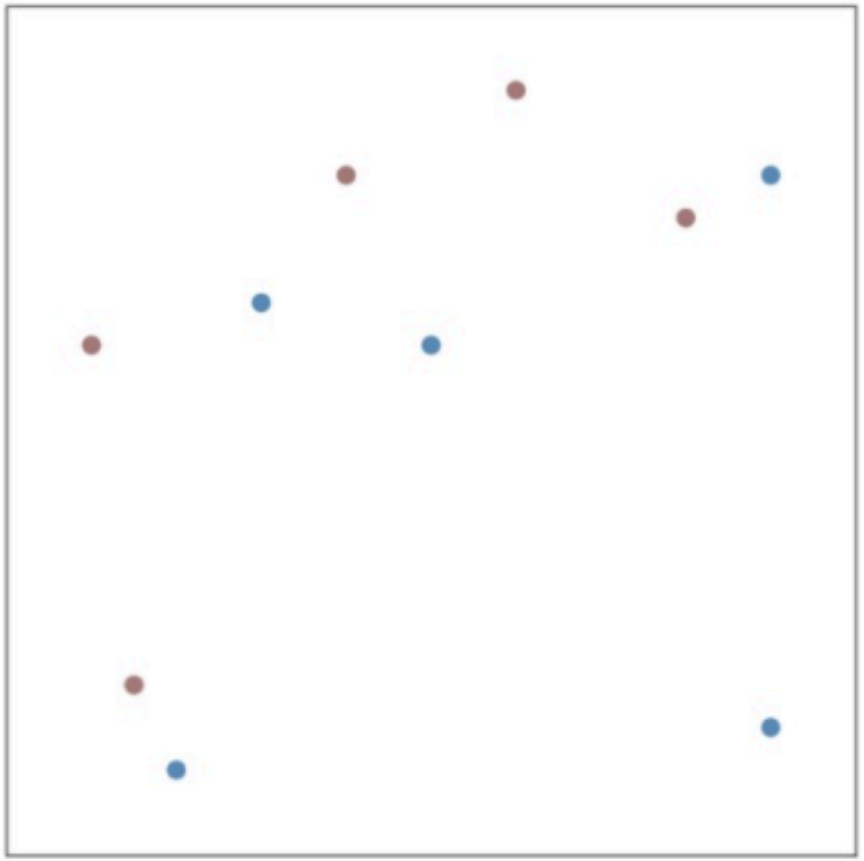
Suppose we have the following training data

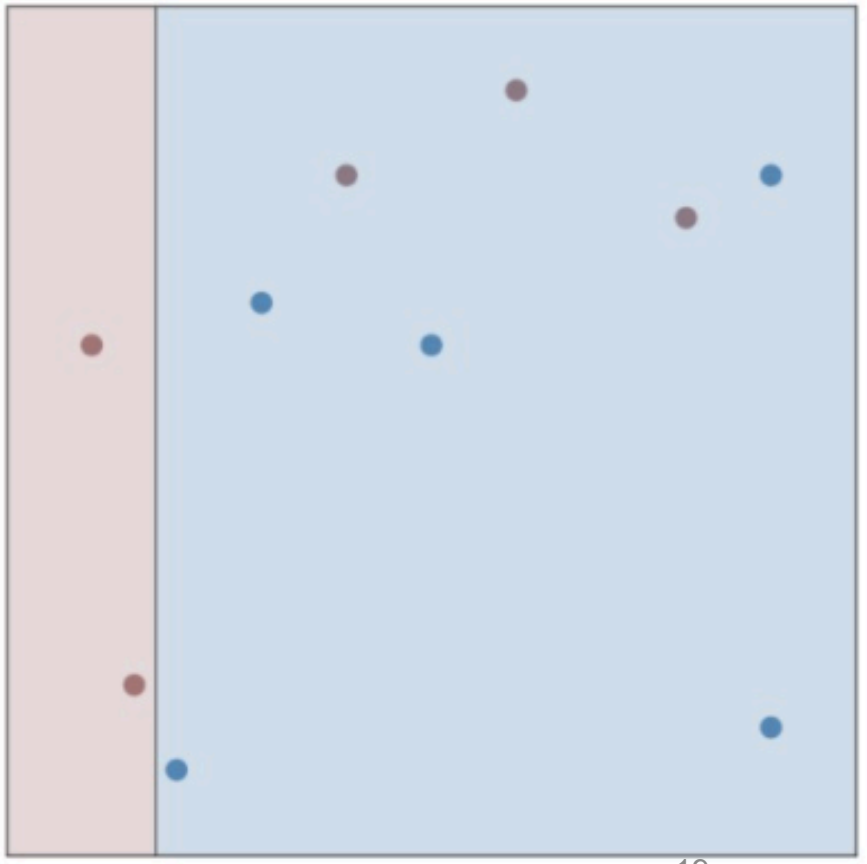


5+ 5−

Adaboost Example

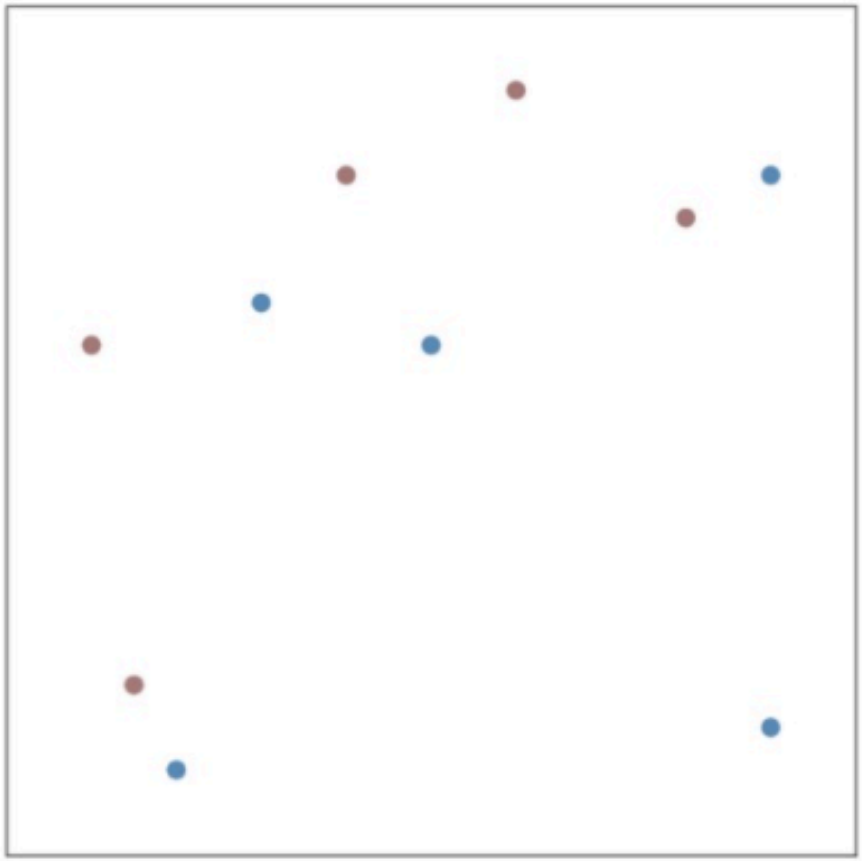
First decision stump





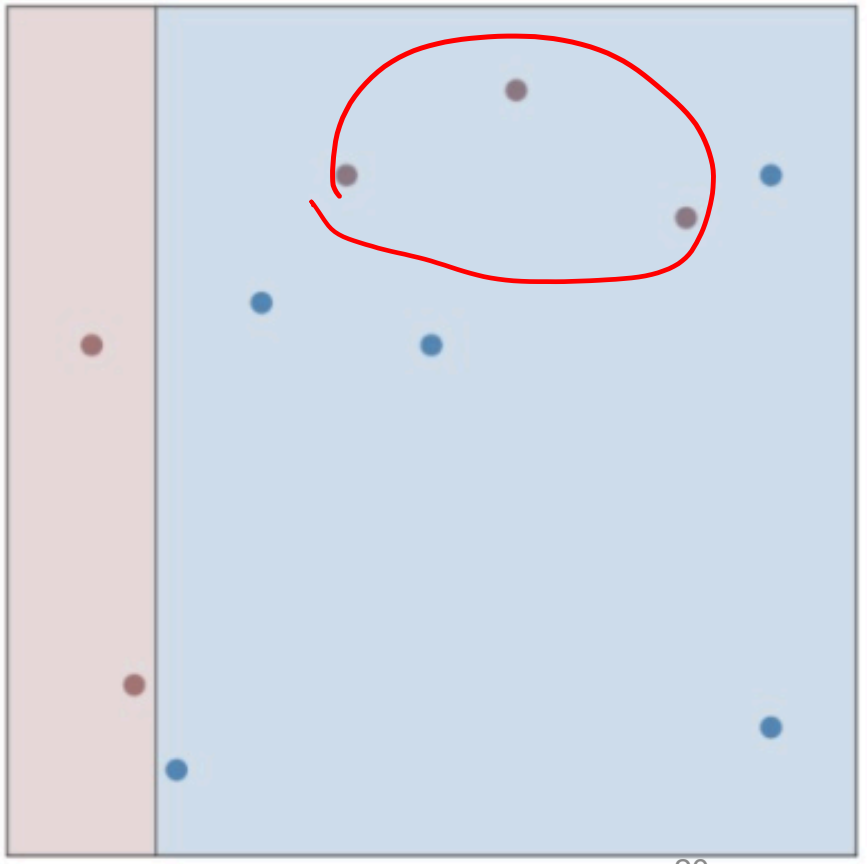
Adaboost Example

First decision stump



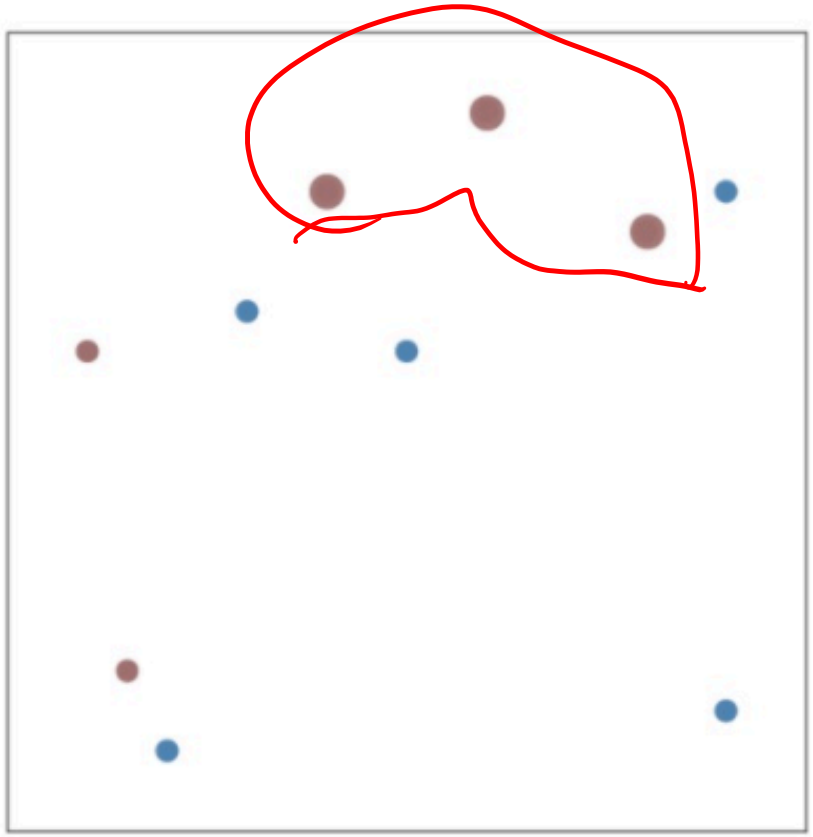
$$\text{err}_1 = 0.30$$

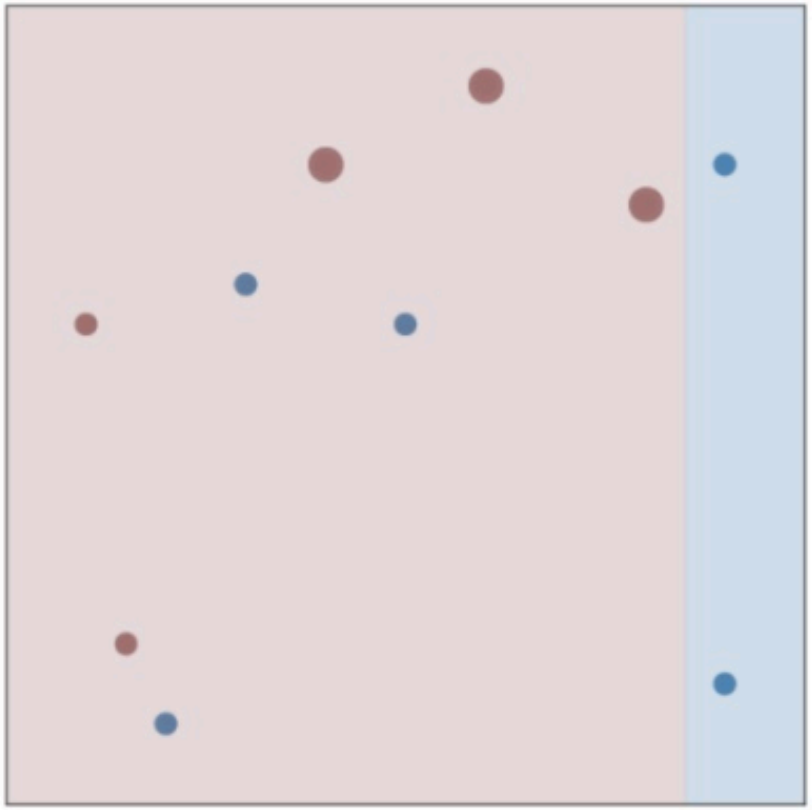
$$\alpha_1 = 0.42$$



Adaboost Example

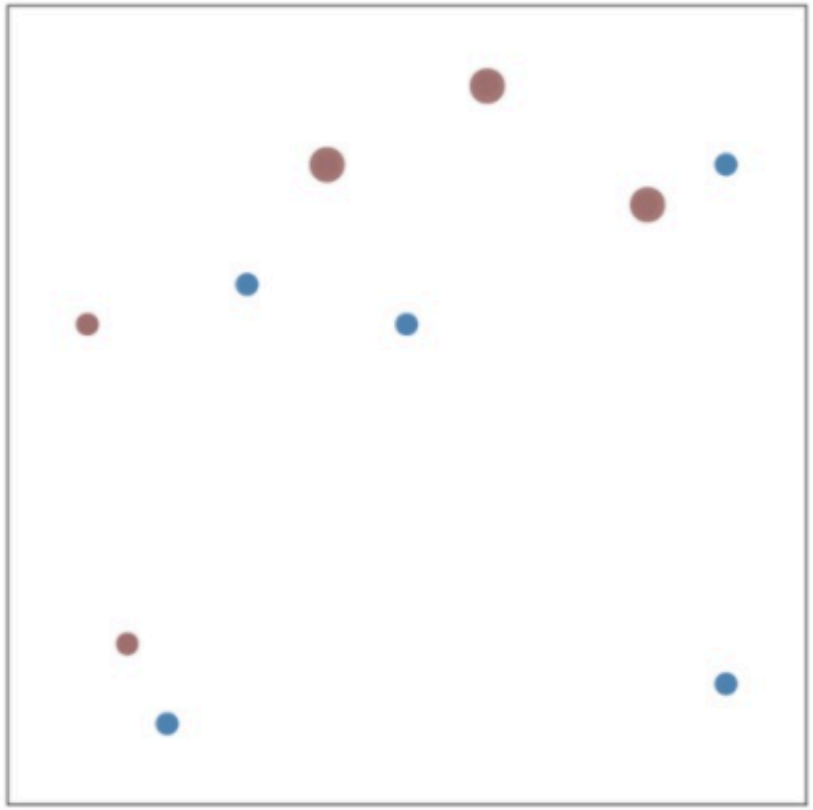
Second decision stump





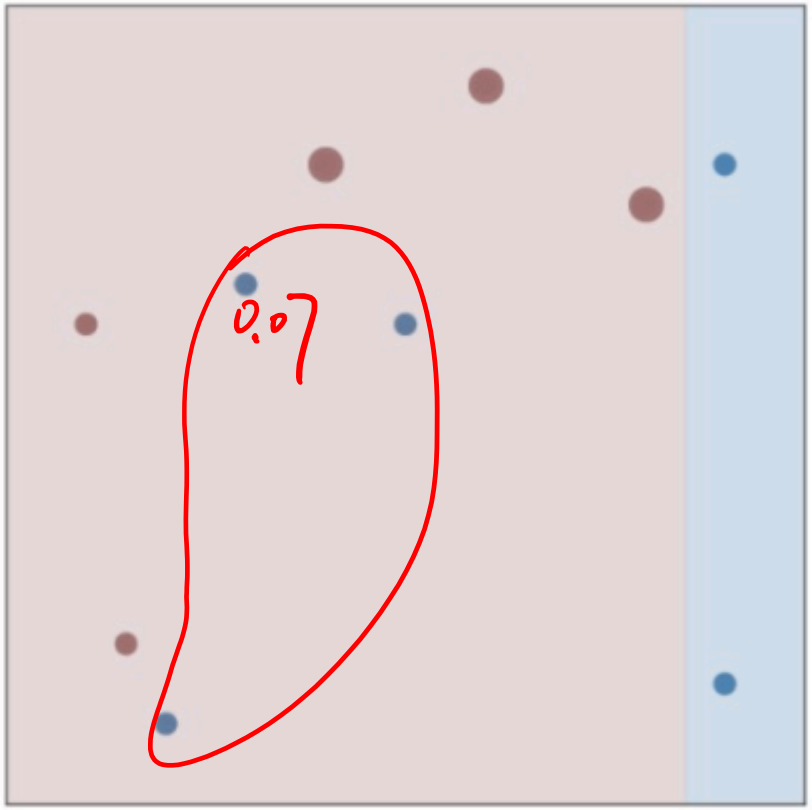
Adaboost Example

Second decision stump



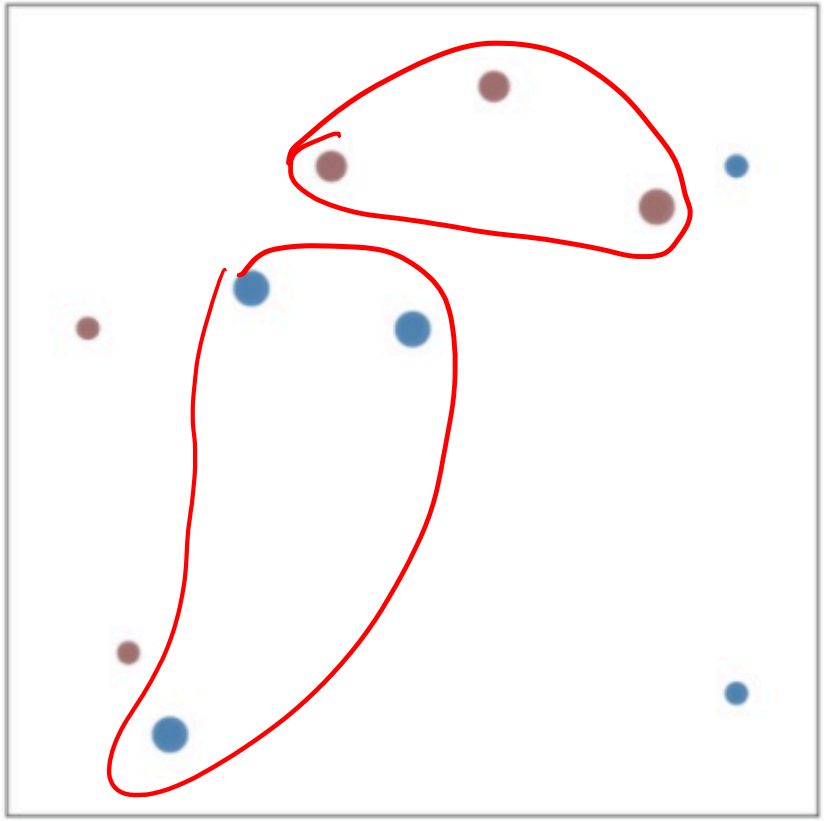
$$\text{err}_2 = 0.21$$

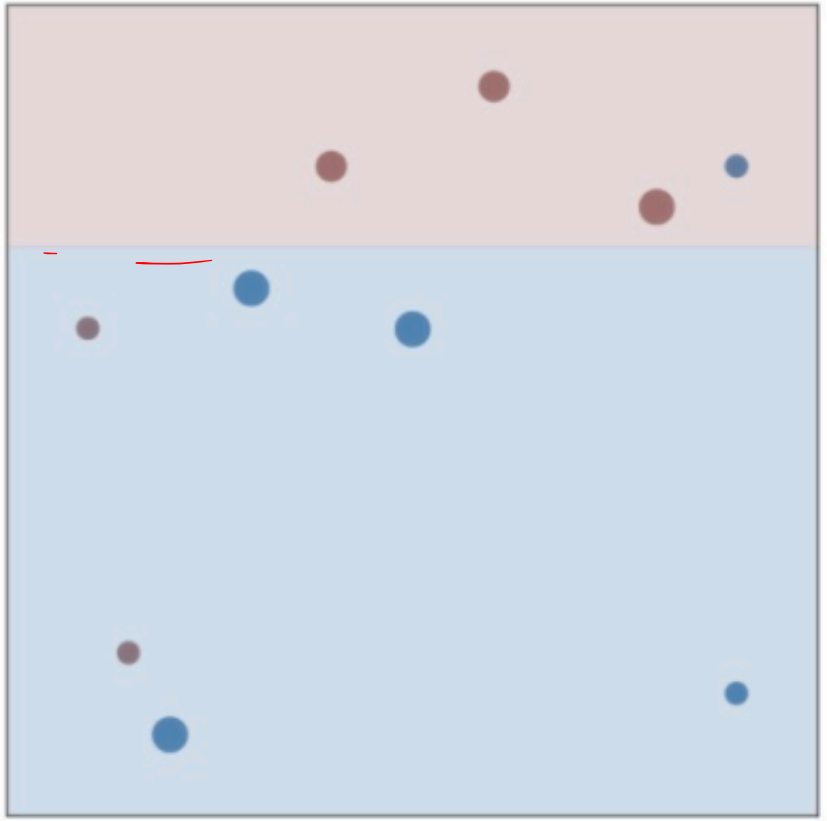
$$\alpha_2 = 0.65$$



Adaboost Example

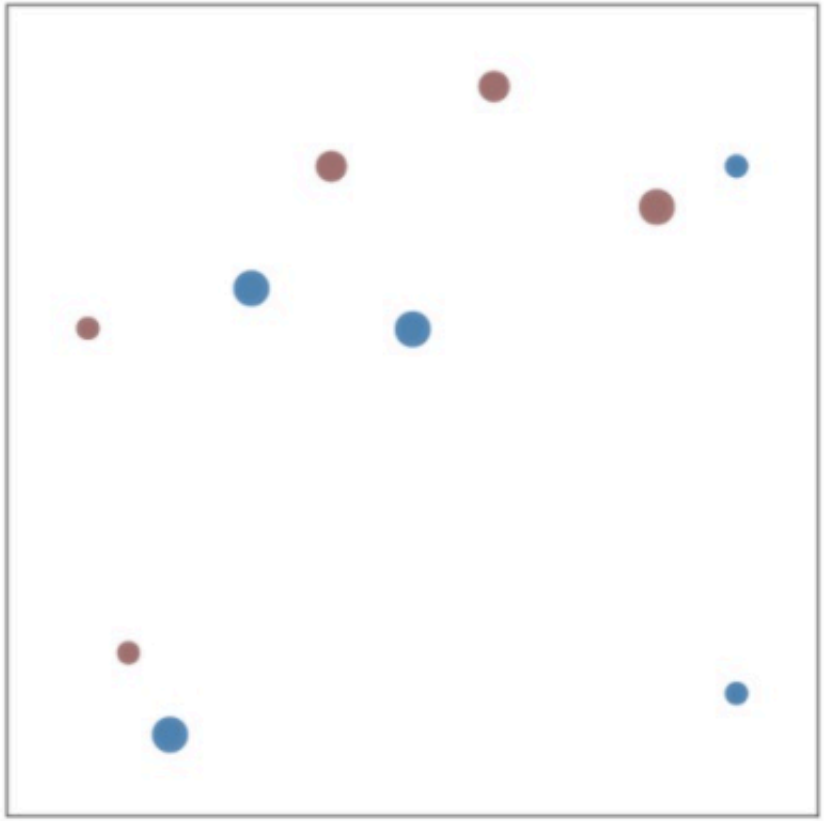
Third decision stump





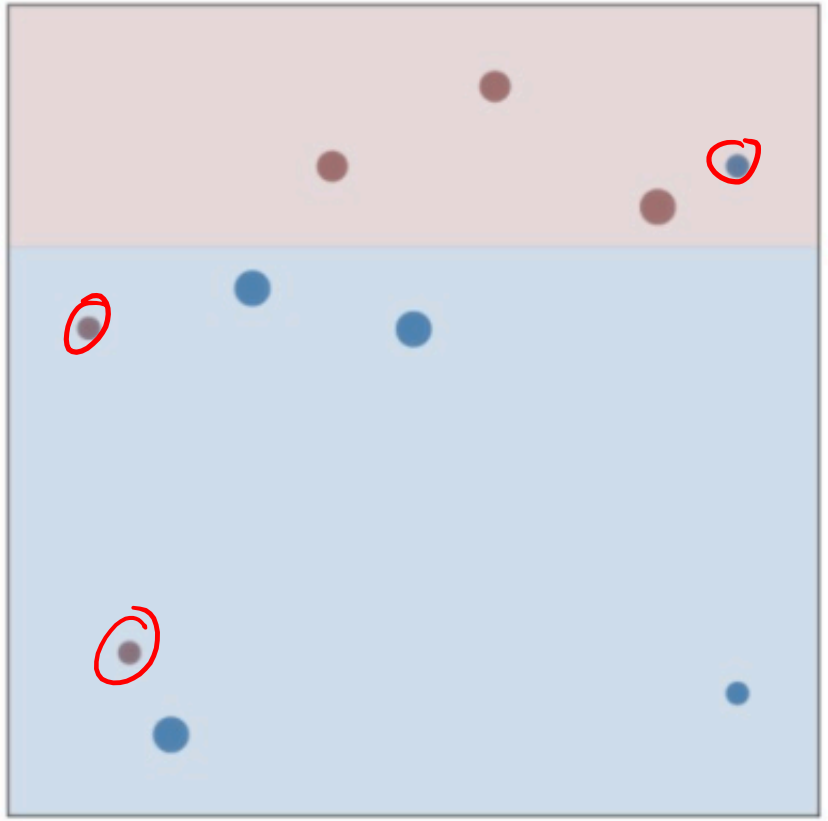
Adaboost Example

Third decision stump



$$\text{err}_3 = 0.14$$

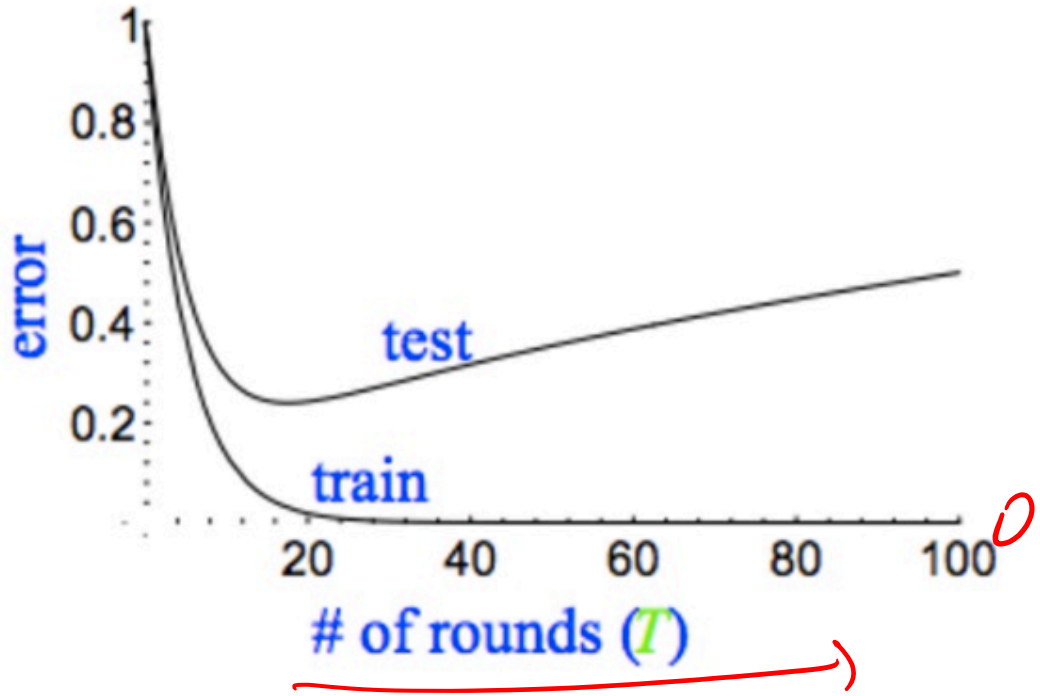
$$\alpha_3 = 0.92$$



Adaboost Example

Generalization performance

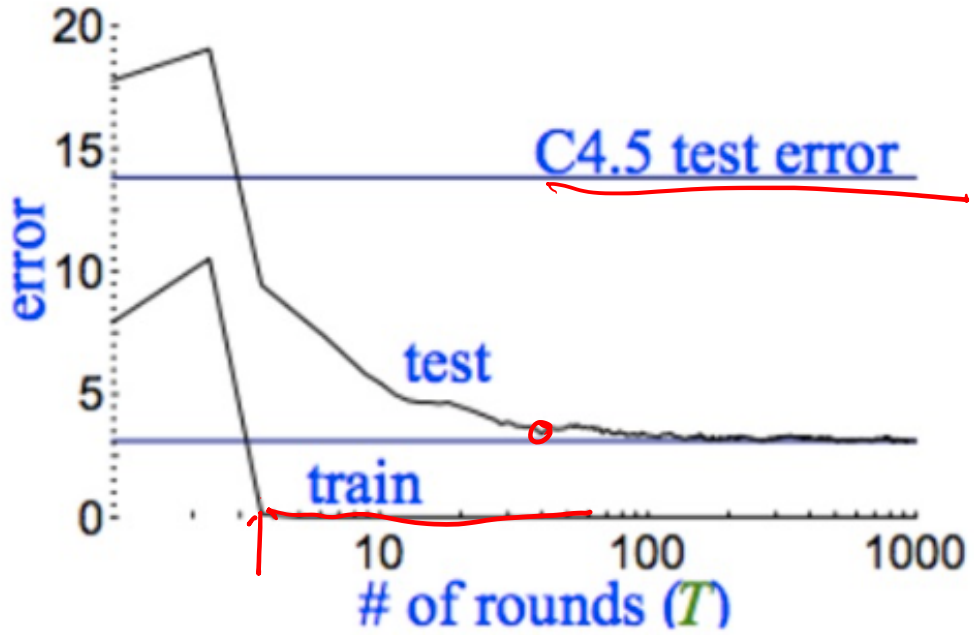
Recall the standard experiment of measuring test and training error vs. model complexity



Once overfitting begins, test error goes up

Generalization performance

Boosting has remarkably uncommon effect



Happens much slower with boosting

The Math

- So far this looks like a reasonable thing that just worked out
- But is there math behind it?

The Math

- Yep! It is minimization of a loss function, like always

Formulate the problem as finding a classifier $H(\mathbf{x})$ such that

$$H^* = \arg\min_H \sum_{i=1}^{m} L(y_i, H(\mathbf{x}_i))$$

where here we take the loss function L to be the following exponential function

$$L = \exp[-y\, H(\mathbf{x})]$$

Notice if $y \neq H(\mathbf{x})$ we get a positive exponent and a large loss.

The Math

Since we're doing this in an iterative way, we're going to assume a form of $H(\mathbf{x})$ that is amenable to iterative improvement. Specifically

$$H(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x})$$

So the problem becomes to choose the optimal weights, α, and optimal basis functions ϕ_k.

For everything I will assume that we've already computed a good H_{k-1} and attempt to build a better H_k.

The Math

At step k we have

$$L_k = \sum_{i=1}^{m} \exp\left[-y_i \left(H_{k-1}(\mathbf{x}_i) + \alpha\phi(\mathbf{x}_i)\right)\right]$$

Taking the things we know out of the exponent gives

$$L_k = \sum_{i=1}^{m} w_{i,k} \exp\left[-\alpha y_i \phi(\mathbf{x}_i)\right], \quad w_{i,k} = \exp\left[-y_i\, H_{k-1}(\mathbf{x}_i)\right]$$

Want to choose good α and ϕ to reduce loss

The Math

Can rewrite as

$$L_k = e^{\alpha} \sum_{y_i \neq \phi(\mathbf{x}_i)} w_{i,k} + e^{-\alpha} \sum_{y_i = \phi(\mathbf{x}_i)} w_{i,k}$$

The Math

Can rewrite as

$$L_k = e^{\alpha} \sum_{y_i \neq \phi(\mathbf{x}_i)} w_{i,k} + e^{-\alpha} \sum_{y_i = \phi(\mathbf{x}_i)} w_{i,k}$$

$$err = \frac{\sum_{i=1}^{m} w_{i,k} I(y_i \neq \phi(x_i))}{\sum_{i=1}^{m} w_{i,k}}$$

Add zero in a fancy way

$$L_k = (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{m} w_{i,k} I(y_i \neq \phi(\mathbf{x}_i)) + e^{-\alpha} \sum_{i=1}^{m} w_{i,k}$$

The Math

Choose ϕ and α separately.
A good ϕ would be one that minimizes weighted misclassifications

$$h_k = \arg\min_{\phi} w_{i,k} I(y_i \neq \phi(\mathbf{x}_i))$$

That's what our weak learner is for

The Math

Plugging that into our Loss function gives

$$L_k = (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{m} w_{i,k} I(y_i \neq h_k(\mathbf{x}_i)) + e^{-\alpha} \sum_{i=1}^{m} w_{i,k}$$

$$\frac{\partial L}{\partial \alpha} = (e^{\alpha} + e^{-\alpha}) \sum_{i=1}^{m} w_{i,k} I(y_i \neq h_k(x_i)) - e^{-\alpha} \sum_{i=1}^{m} w_{i,k} = 0$$

$$(e^{2\alpha} + 1) = \sum_{i=1}^{m} w_{i,k} \Big/ \sum_{i=1}^{m} w_{i,k} I(y_i \neq h_k(x_i)) = \frac{1}{err}$$

$$\alpha = \frac{1}{2} \log \frac{1 - err}{err}$$

The Math

Plugging that into our Loss function gives

$$L_k = (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{m} w_{i,k} I(y_i \neq h_k(\mathbf{x}_i)) + e^{-\alpha} \sum_{i=1}^{m} w_{i,k}$$

Now we want to minimize w.r.t. α. Take derivative, set equal to zero

$$0 = \frac{dL_k}{d\alpha} = (e^{\alpha} + e^{-\alpha}) \sum_{i=1}^{m} w_{i,k} I(y_i \neq h_k(\mathbf{x}_i)) - e^{-\alpha} \sum_{i=1}^{m} w_{i,k}$$

which gives

$$e^{2\alpha} = \frac{\sum_i w_{i,k}}{\sum_i w_{i,k} I(y_i \neq h_k(\mathbf{x}))} - 1 = \frac{1}{\text{err}_k} - 1 = \frac{1 - \text{err}_k}{\text{err}_k}$$

And finally $\alpha_k = \frac{1}{2} \log \left(\frac{1-\text{err}_k}{\text{err}_k} \right)$

The Math

What about the weight update?
Remember we got the weights by pulling the already computed function out of L.
For the new weights, we have

$$w_{i,k+1} = \exp[-y_i H_k(\mathbf{x}_i)] = \exp[-y_i H_{k-1}(\mathbf{x}_i) - \alpha_k y_i h_k(\mathbf{x}_i)] = w_{i,k} \exp[-\alpha_k y_i h_k(\mathbf{x}_i)]$$

which gives the update

$$w_i \leftarrow w_i \exp[-\alpha_k y_i h_k(\mathbf{x}_i)]$$

And finally, $H_K(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$
This is exactly Adaboost

Practical Advantages of Boosting

- It's fast!
- Simple and easy to program
- No parameters to tune (except *K*)
- Flexible. Can choose any weak learner
- Shift in mindset. Now can look for weak classifiers instead of strong classifiers
- Can be used in lots of settings

Caveats

- Performance depends on data and weak learner
- Adaboost can fail if
  - Weak classifier not weak enough (overfitting)
  - Weak classifier too weak (underfitting)