University of Colorado **Boulder**

Department of Computer Science
CSCI 5622: Machine Learning
Chris Ketelsen

Lecture 9: Boosting

Learning Objectives

- [ ] See some general ensemble classifiers
- [ ] Learn about the Adaboost algorithm
- [ ] Learn the math behind Boosting

Boosting Intuition

How is it that a committee of blockheads can somehow arrive at highly reasoned decisions, despite the weak judgment of the individual members? How can the shaky separate views of a panel of dolts be combined into a single opinion that is very likely to be correct?

Schapire and Freund, *Boosting: Foundations and Algorithms*

Ensemble Learning Intuition

Problem: Decision trees tend to overfit and are sensitive to noise in training data, leading to poor generalization error

Idea: Build a bunch of simple models

Make prediction based on majority vote of simple models

Bagging: Train bunch of decision trees on subsets of training data (with replacement). Predict based on majority vote.

Random Forests: Train bunch of decision trees on subsets of training data **and** subsets of features. Predict based on majority vote.

Ensemble Learning Intuition

Bagging: Train bunch of decision trees on subsets of training data (with replacement). Predict based on majority vote.

Random Forests: Train bunch of decision trees on subsets of training data **and** subsets of features. Predict based on majority vote.

A single tree is very sensitive to training data. An average over many trees (hundreds of thousands) is less sensitive.

Bagging can sometimes create correlated trees, especially if there are a handful of very strong features

Random forests help to de-correlate trees by leaving the strong features out in some instances

Boosting Intuition

Boosting is an ensemble method, but with a different twist

Idea: Build a sequence of **dumb models**

Change training data along the way to focus on difficult to classify examples

Predict based on **weighted majority vote** of all the models

- 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

- Look at example of popular Boosting method
- Unpack it for intuition
- Come back later and show the math

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

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

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

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.

2a. Fit classifier $h_k(\mathbf{x})$ to training data with weights w_i

Decide split based on info gain with weighted entropy:

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

Before $p =$ fraction in positive class

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

Adaboost

2b. Compute weighted error

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

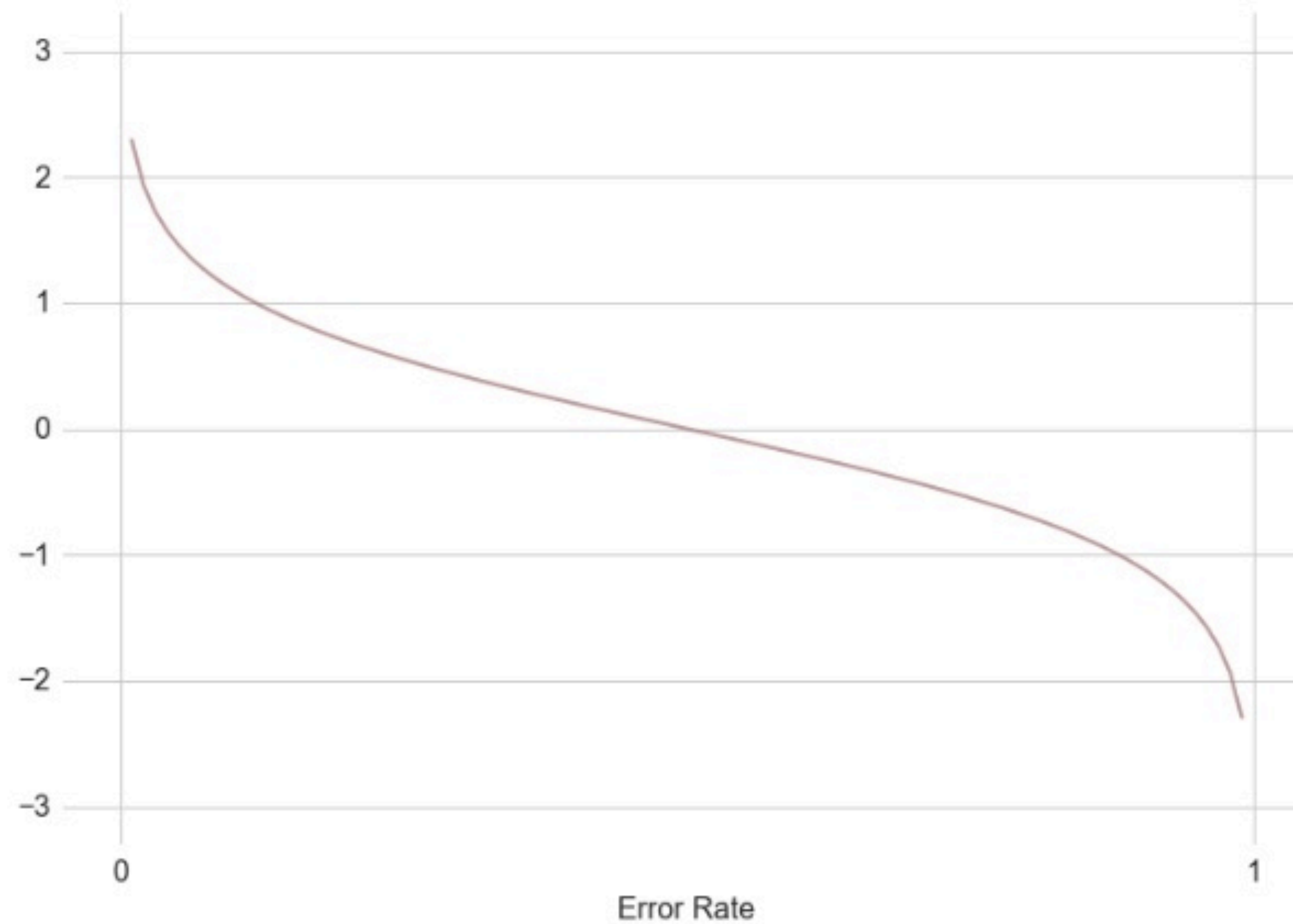
Still gives error rate in $[0,\ 1]$

Mistakes on highly weighted examples hurt more

Mistakes on lowly weighted examples don't register too much

Adaboost

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



Models with small err get promoted (exponentially)

Models with large err get demoted (exponentially)

Adaboost

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

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

Z_k is just a normalizing constant to ensure that the w's are a distribution

Adaboost

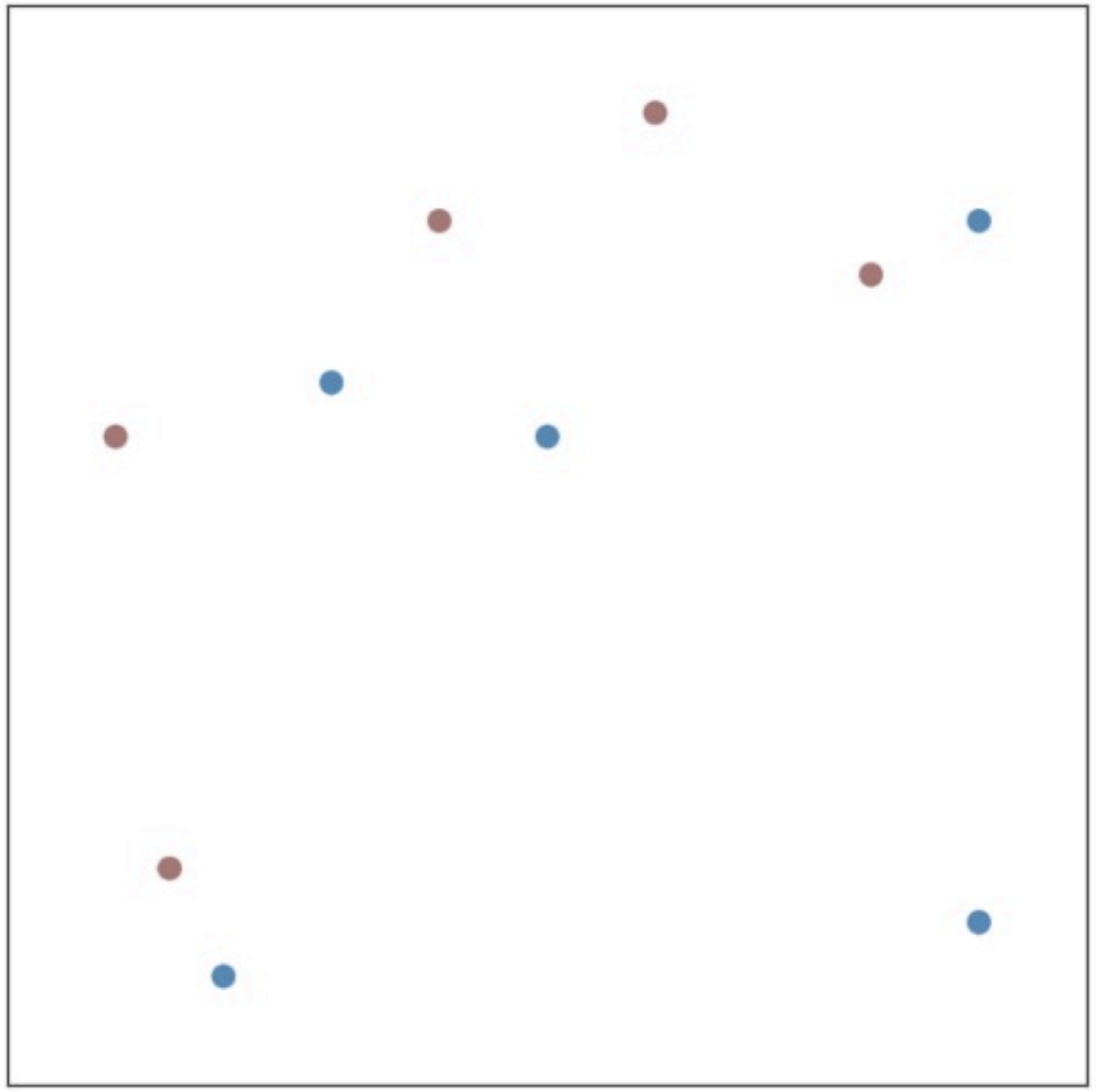
3 . Output $H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$

Sum up weighted votes from each model

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

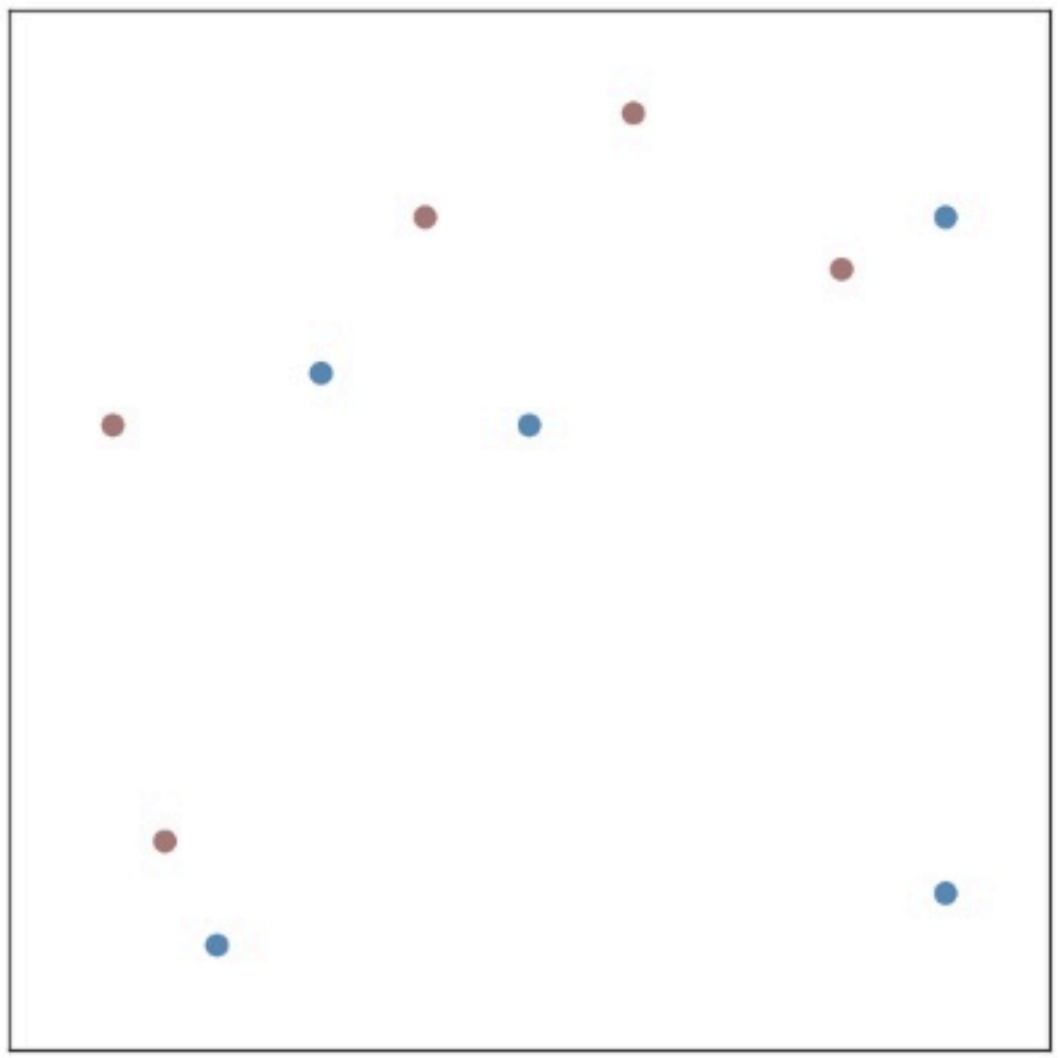
Example

Suppose you have the following training data



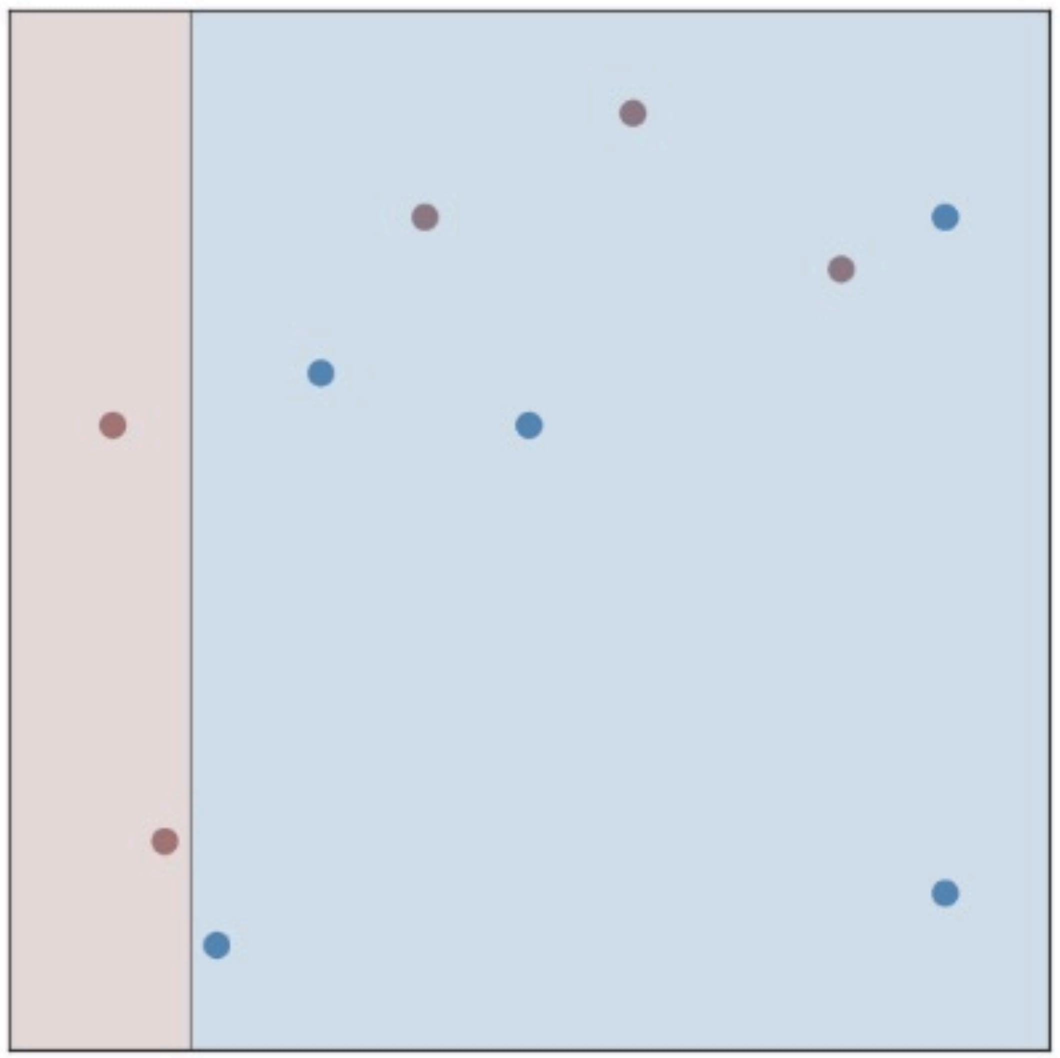
Example

First decision stump



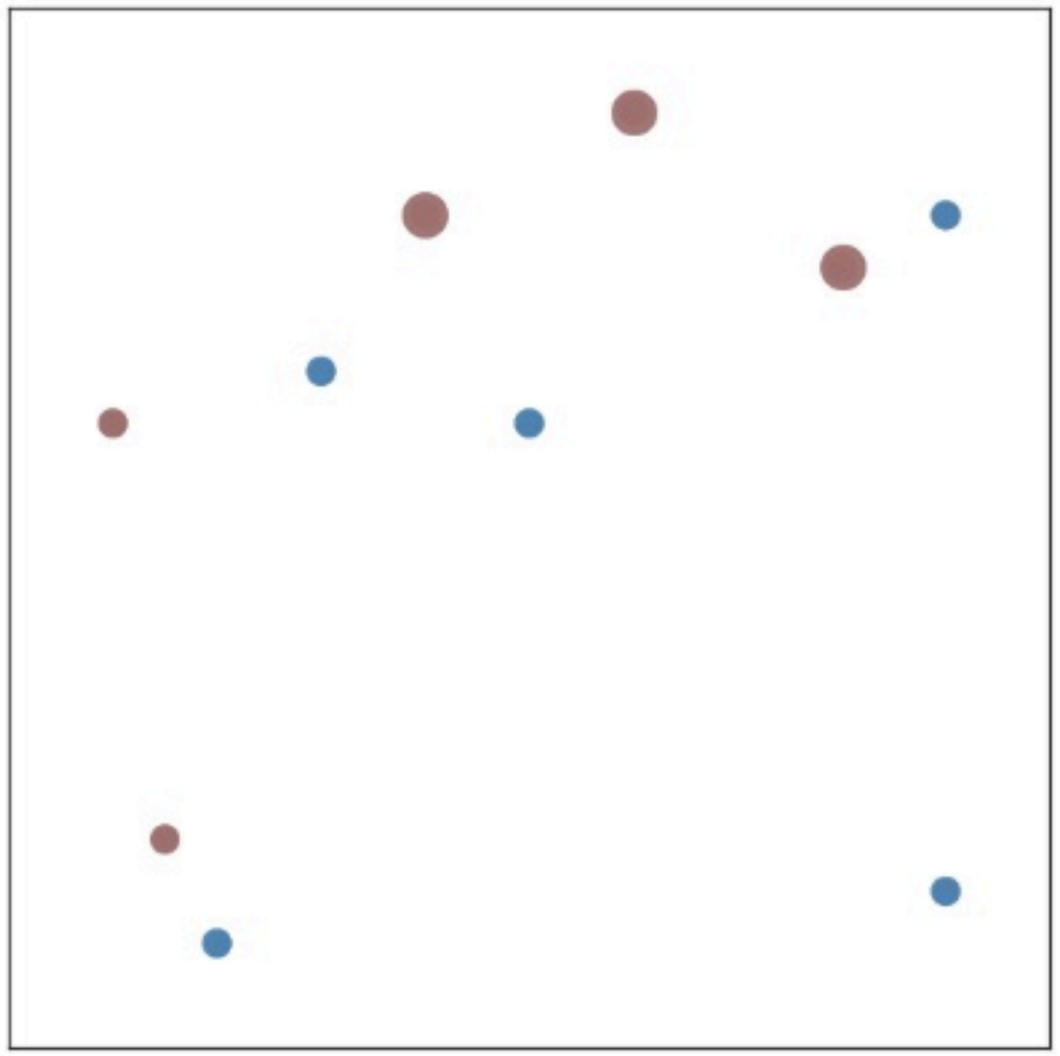
$$\mathrm{err}_1 = 0.30$$

$$\alpha_1 = 0.42$$



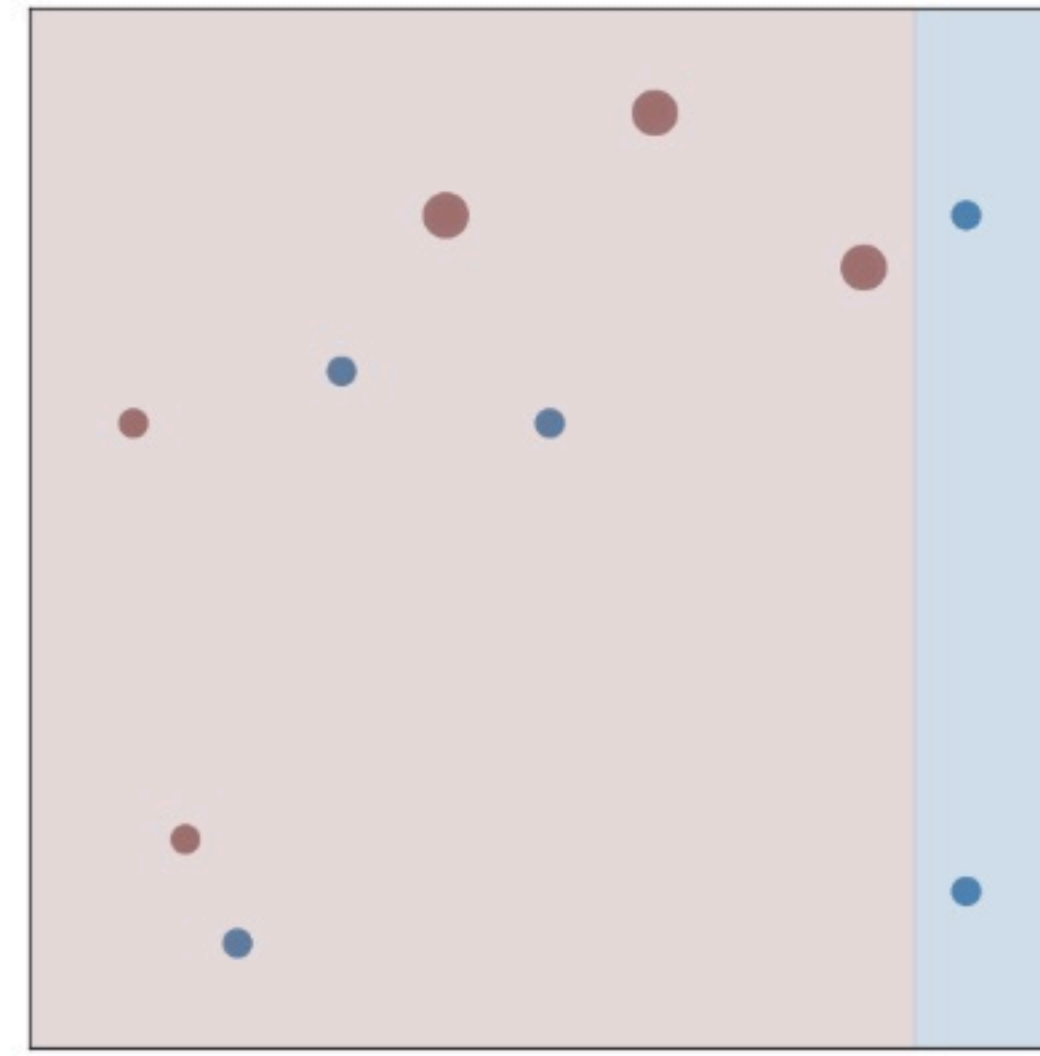
Example

Second decision stump



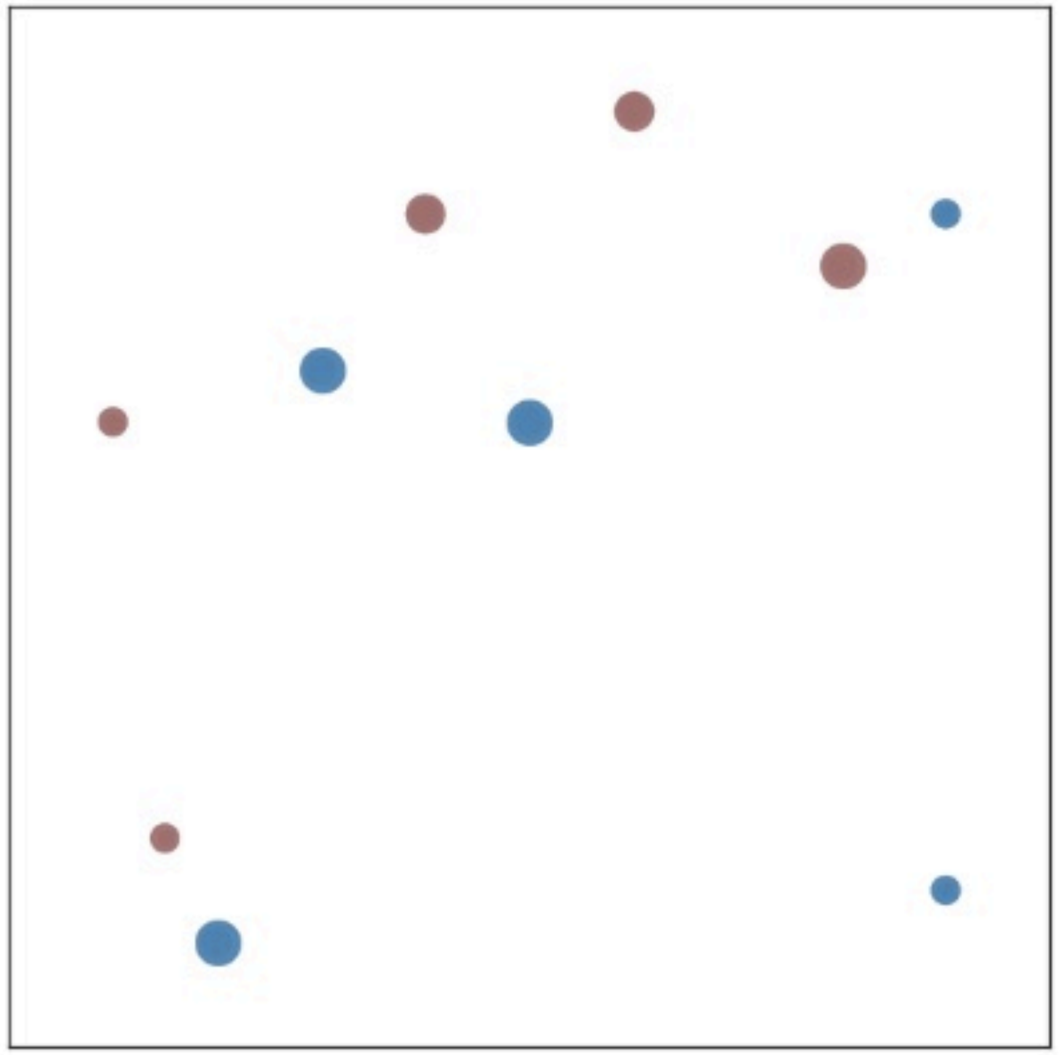
$$\mathrm{err}_2 = 0.21$$

$$\alpha_2 = 0.65$$



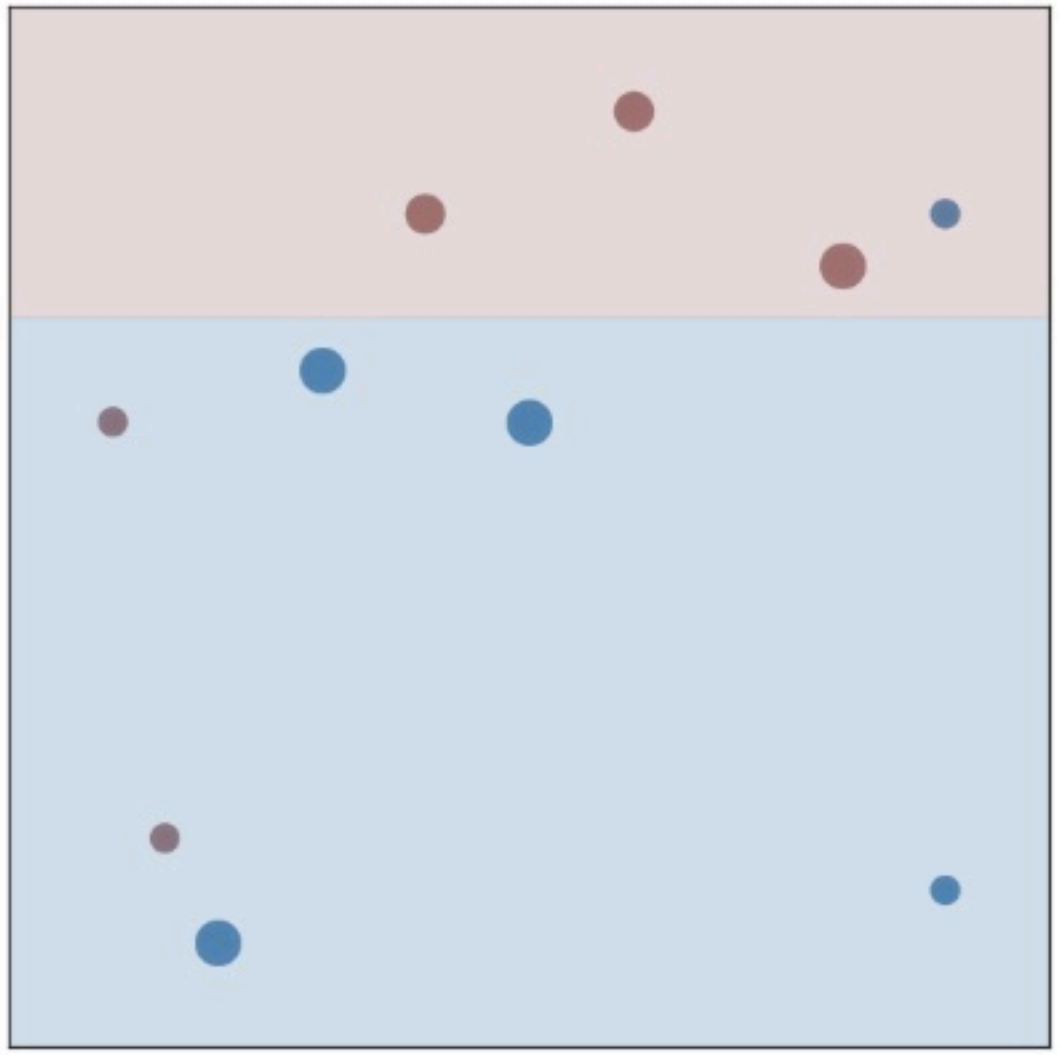
Example

Third decision stump

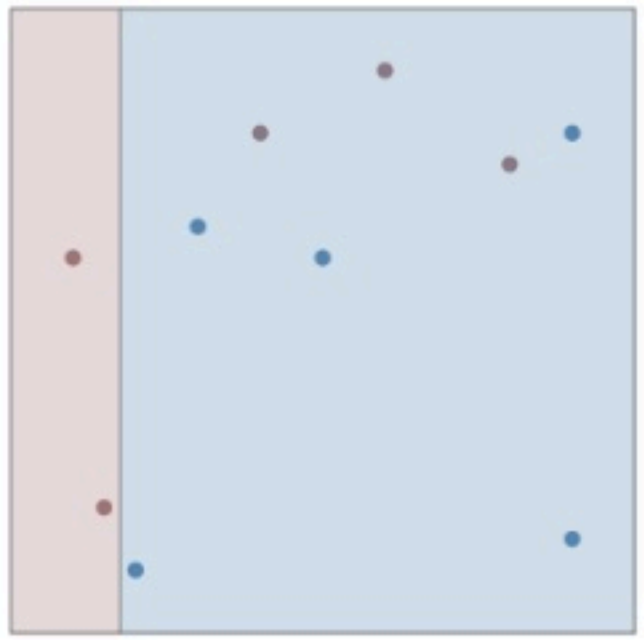
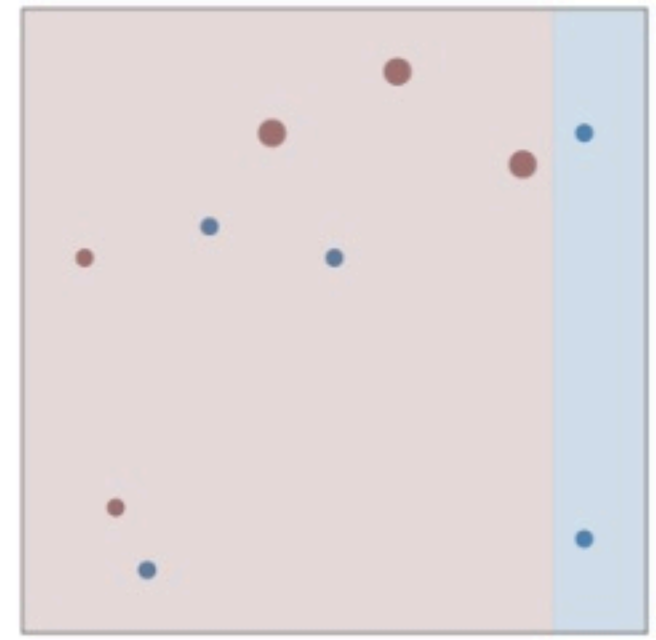


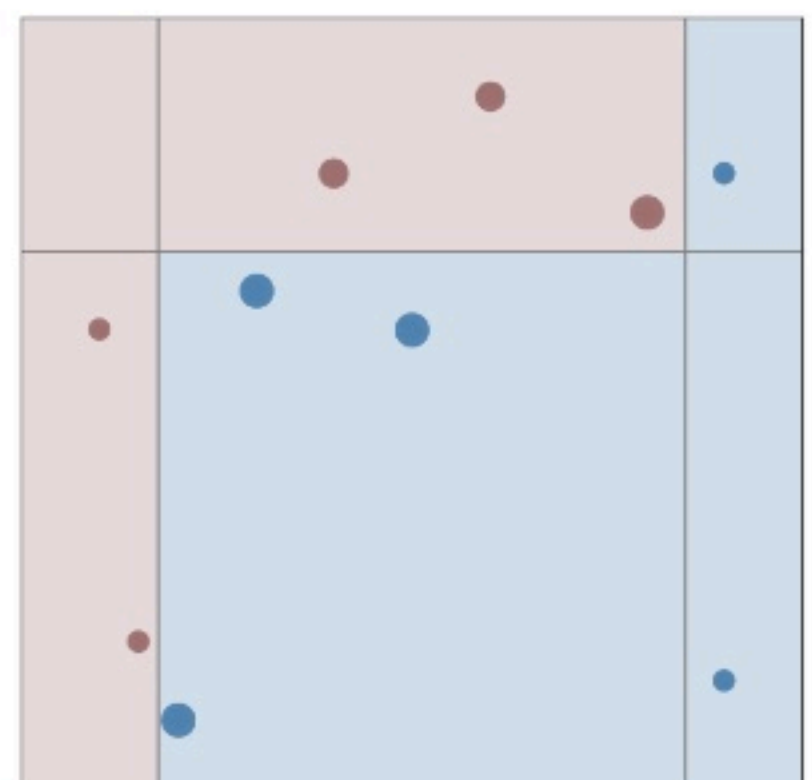
$$\mathrm{err}_3 = 0.14$$

$$\alpha_3 = 0.92$$



Example

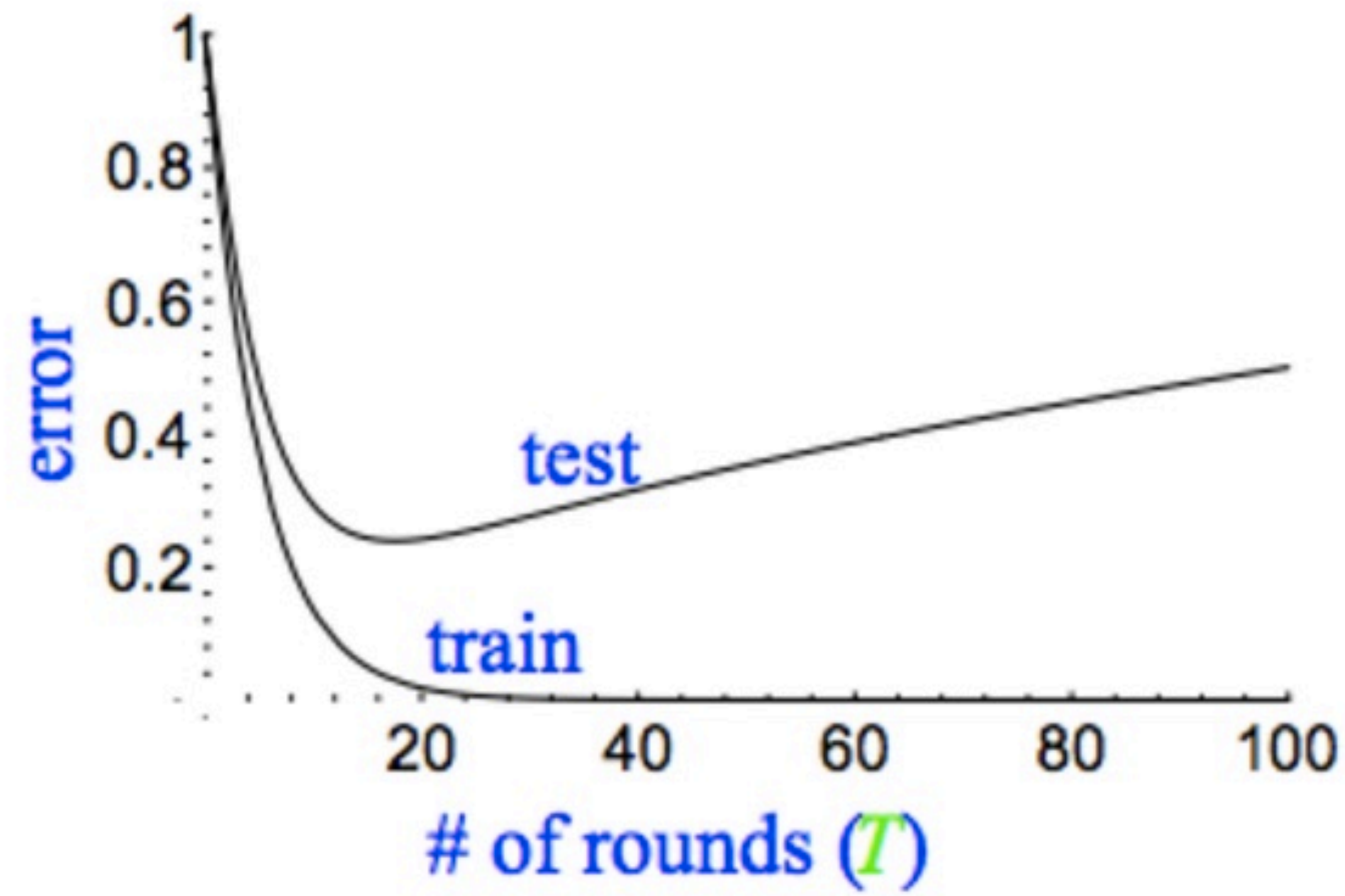
0.42		+0.65		+0.92		=



Performance

Boosting has a remarkably uncommon affect

If we plot errors vs model complexity

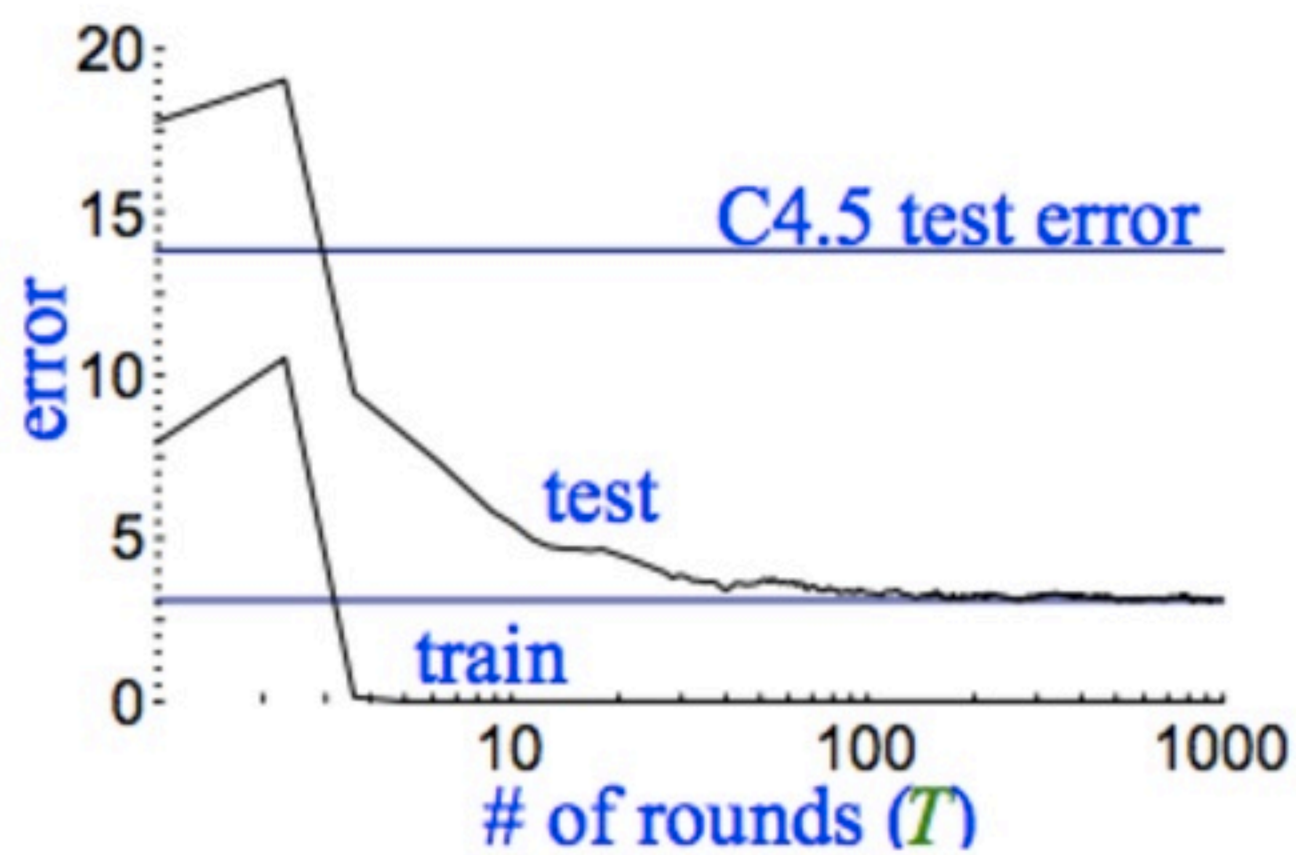


Once overfitting begins, test error goes up

Performance

Boosting has a remarkably uncommon affect

If we plot errors vs model complexity



(boosting C4.5 on "letter" dataset)

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! And it's just 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

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

Taking the things we know out of the exponent gives

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

Want to choose good α and ϕ to reduce loss

The Math

Can rewrite as

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

Add zero in a fancy way

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

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

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

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

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

which gives

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

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

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

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

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 (text learning, images, discrete, continuous)

Caveats

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

Next Time

- Regression
- More on regularization
- The Bias - Variance Trade-Off