Boosted Decision Trees

Previously on CSCI 4622

Ensemble Learning: Build a bunch of slightly different models. Predict by majority vote.

Bagging: Train a bunch of decision trees on bootstrapped resamples of training data.

Random Forests: Train a bunch of decision trees on bootstrapped resamples of training data. On each node, consider only a random subset of features to split on.

A single tree is very sensitive to training data. An average over many trees is less sensitive.

Bagging can sometimes create correlated trees, especially if there's a handful of strong features

Random Forests helps to de-correlate trees by leaving strong features out in some splits

Boosting Intuition

Boosting is another ensemble model, but with a slight twist

"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

Boosting Intuition

Boosting is another ensemble model, but with a slight twist

"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"

Idea: Build a sequence of dumb models

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

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

- What do we mean by **dumb**?
- How do we **promote** difficult examples?
- Which models get **more say** in the vote?

Dumb Models

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 is a **Decision Stump** - a Decision Tree with a **single split**

Other common weak learners include slightly deeper Decision Trees and perceptrons

Promoting Difficult Examples

$\{(\vec{x}_i, y_i)\}_{i=1}^{m}$

How do we **promote** difficult examples?

After each iteration:

- **Increase** importance of training examples we got **wrong** on previous iteration
- **Decrease** importance of training examples we got **right** on previous iteration

Each training example will carry around a weight w_i $\quad i = 1, \ldots, m$

Weights are nonnegative normalized so that they behave like a probability distribution

$$\sum_{i=1}^{m} w_i = 1, \quad w_i \geq 0$$

Weighted Majority Vote

Which models get **more say** in the weighted majority vote?

The models that performed better on the 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

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

The weight $h_k(\mathbf{x})$ controls vote contribution of α_k . Think of as measure of accuracy of $h_k(\mathbf{x})$

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 vectors $\mathbf{x}_i \in \mathbb{R}^p$
- Labels are $y_i \in \{+1, -1\}$

AdaBoost Algorithm

1. Initialize training example weights to uniform distribution $w_i = \frac{1}{m}$ for $i = 1, 2, \ldots, m$

2. For $k = 1, 2, \ldots, K$: (ADDING MORE WEAK LEARNERS)

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

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

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

   d) Update training example weights $w_i \leftarrow \frac{1}{Z_k} \cdot w_i \exp\left[-\alpha_k y_i h_k(\mathbf{x}_i)\right]$

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

Adaboost Algorithm

1. Initialize training example weights to uniform distribution $w_i = \frac{1}{m}$ for $i = 1, 2, \ldots, m$

Weights are initialized to uniform dist. At beginning, all training examples treated the same

Adaboost Algorithm

1. Initialize training example weights to uniform distribution $w_i = \frac{1}{m}$ for $i = 1, 2, \ldots, m$

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

Decide Decision Stump split based on Impurity and Information gain as usual

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

Before: p was fraction of examples with positive label $p = \frac{\sum_{i=1}^{m} I(y_i = +1)}{m}$

Now: p is fraction of **weights** in pos class $p = \frac{\sum_{i=1}^{m} w_i \cdot I(y_i = +1)}{\sum_{i=1}^{m} w_i}$

← pos pts

Adaboost Algorithm

2b) Compute weighted errors:

got i-th wrong

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

- Still gives an error rate in [0, 1]
- Mistakes on highly weighted examples hurt more
- Mistakes on lowly weighted examples don't hurt as much

Adaboost Algorithm

2c) Compute vote weight for the current weak learner $\alpha_k = \frac{1}{2}\log((1-\text{err}_k)/\text{err}_k)$

- Models with small weighted error get a larger vote
- Models with large weighted error get less vote (or, sometimes a negative vote!)

Adaboost Algorithm

2d) Update training example weights $w_i \leftarrow \frac{1}{Z_k} \cdot w_i \exp\left[-\alpha_k y_i h_k(\mathbf{x}_i)\right]$

- If example is misclassified then weight goes up
- If example is classified correctly then weight goes down
- How big of a jump up/down depends on how much vote the current learner has
- Z_k is just a normalizing constant to make it a probability distribution
- Don't really have to compute Z_k, just scale weights so they sum to 1

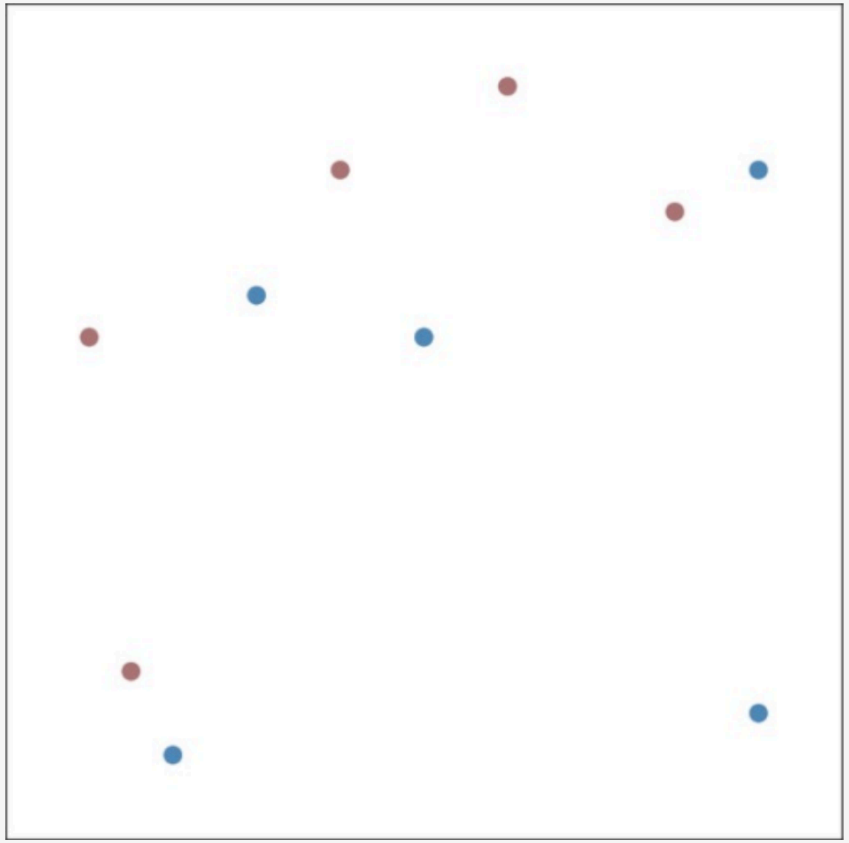
Adaboost Algorithm

3) Algorithm produces the final boosted classifier $H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right]$

- Sum up weighted votes from each model
- Apply sign function so that it produces predictions in $\{+1, -1\}$

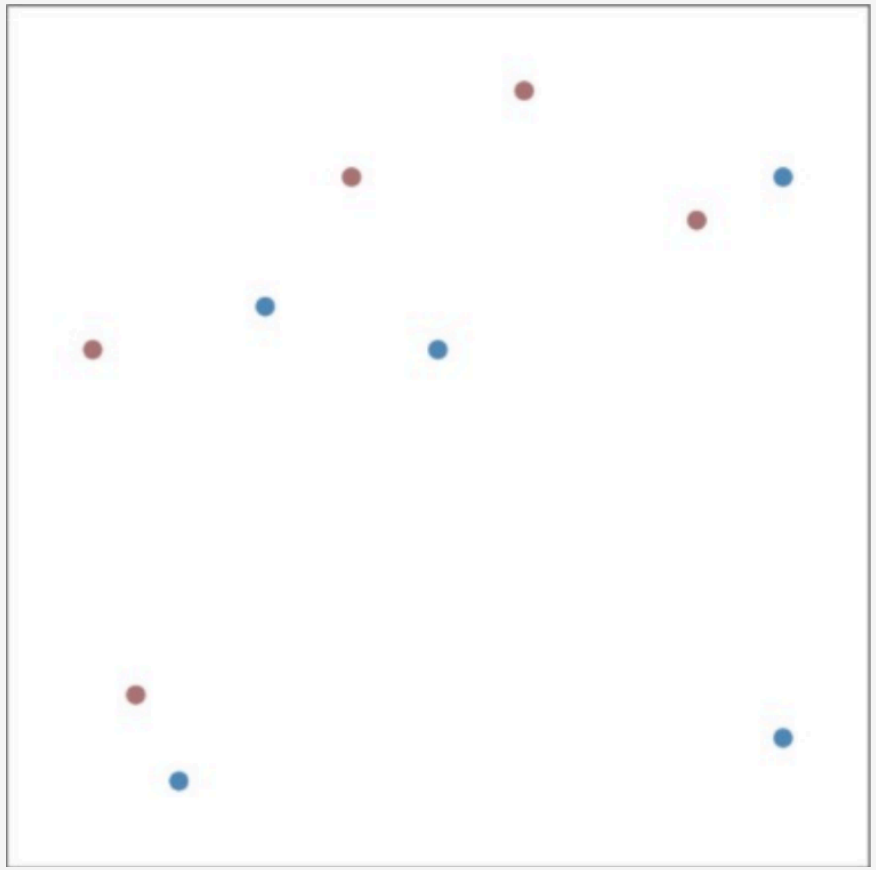
Adaboost Example

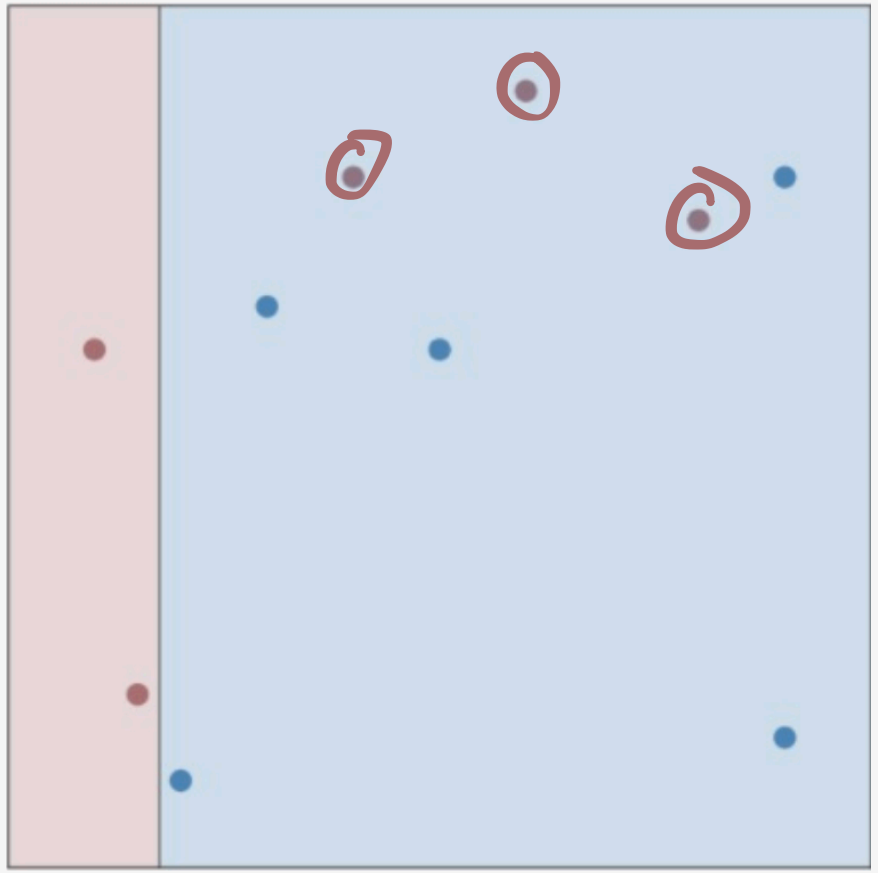
Example: Suppose you have the following training data



Adaboost Example

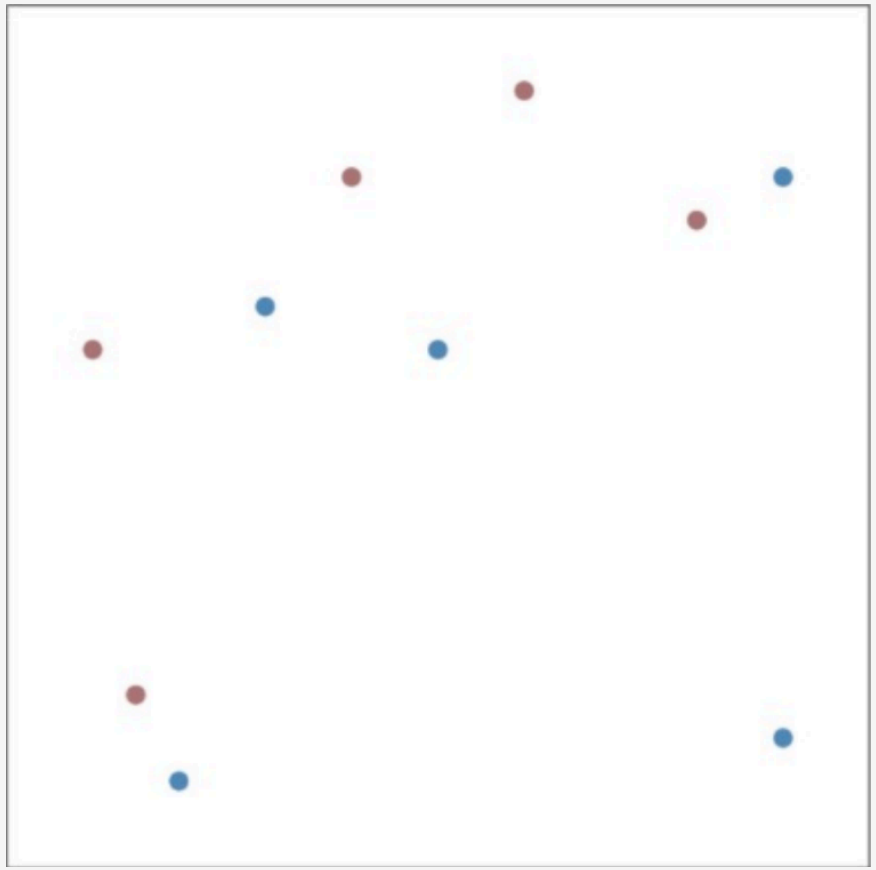
- **First Decision Stump:**



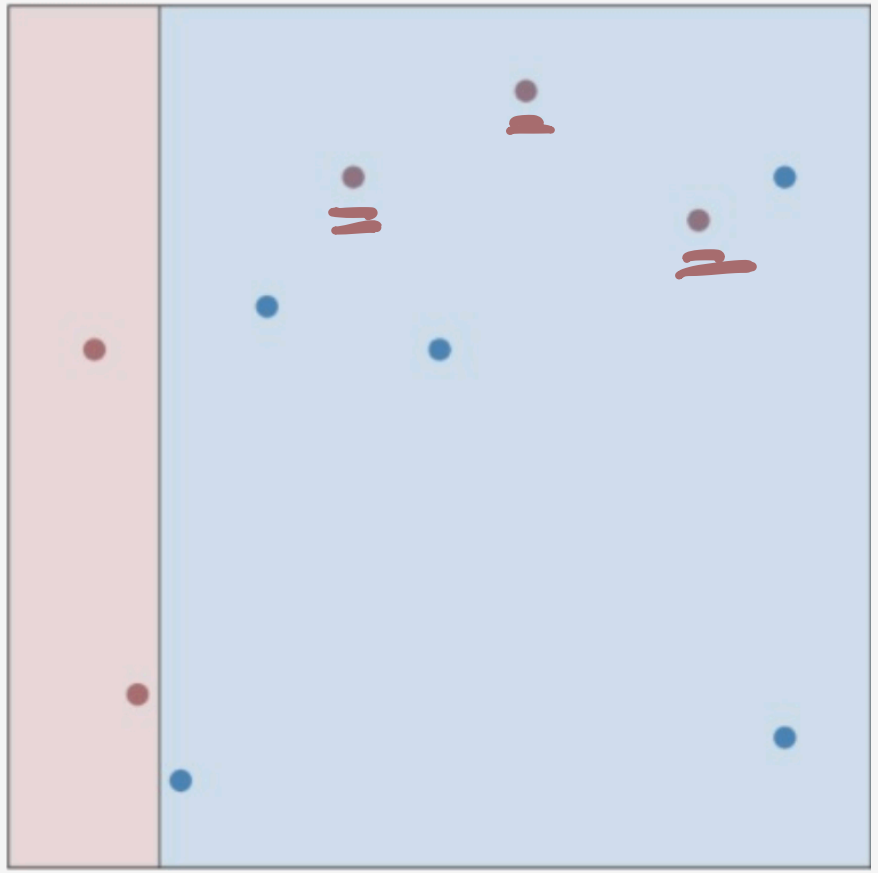


Adaboost Example

- **First Decision Stump:**

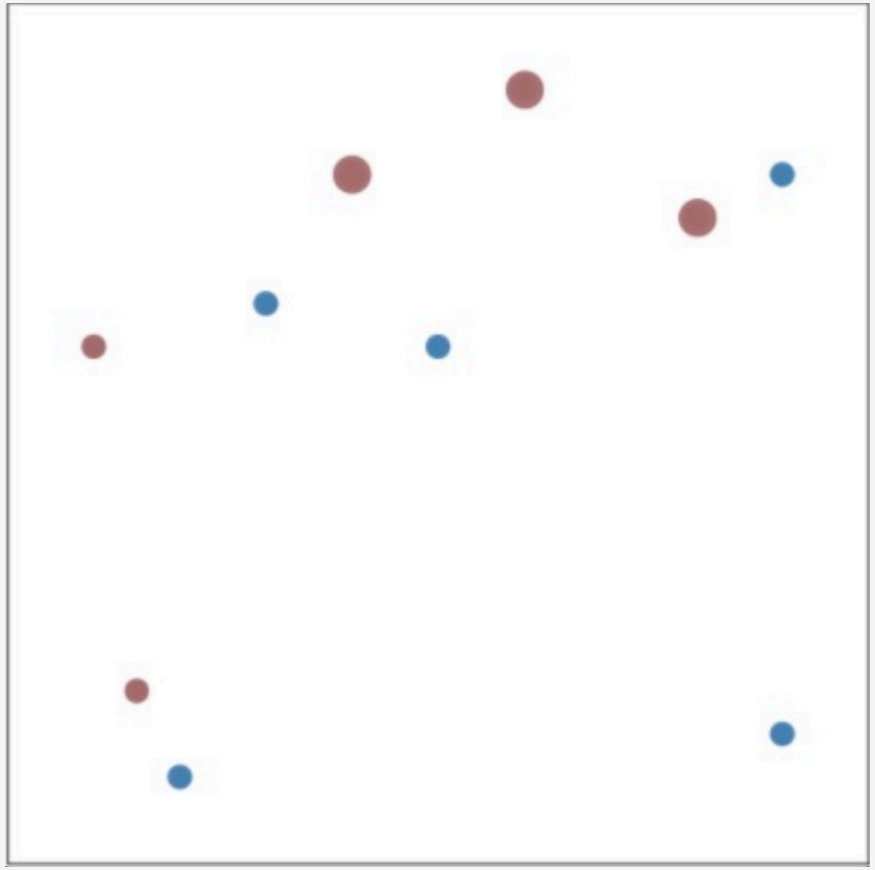


$$\begin{aligned} \text{err}_1 &= 0.30 \\ \alpha_1 &= 0.42 \end{aligned}$$

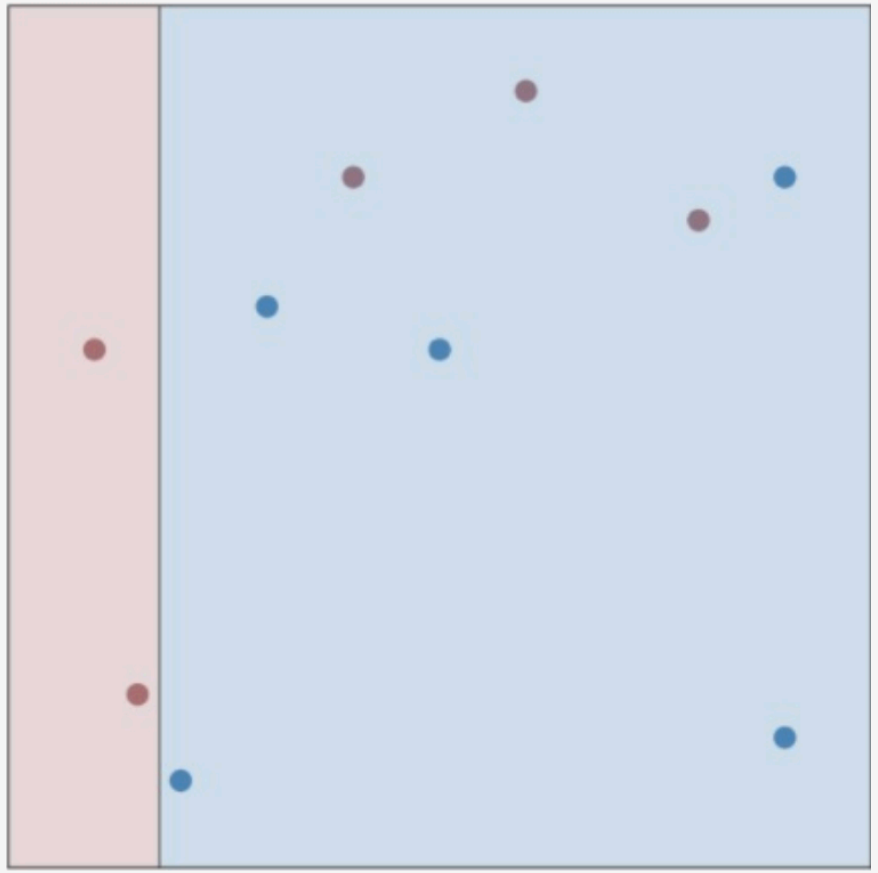


Adaboost Example

- **First Decision Stump:**

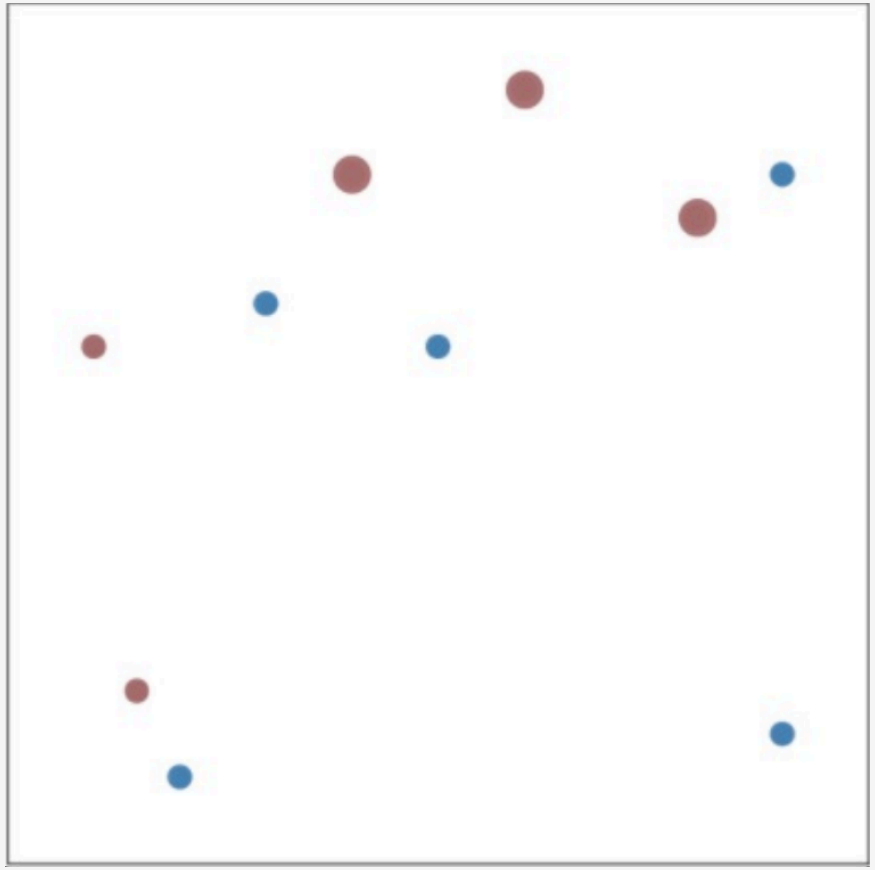


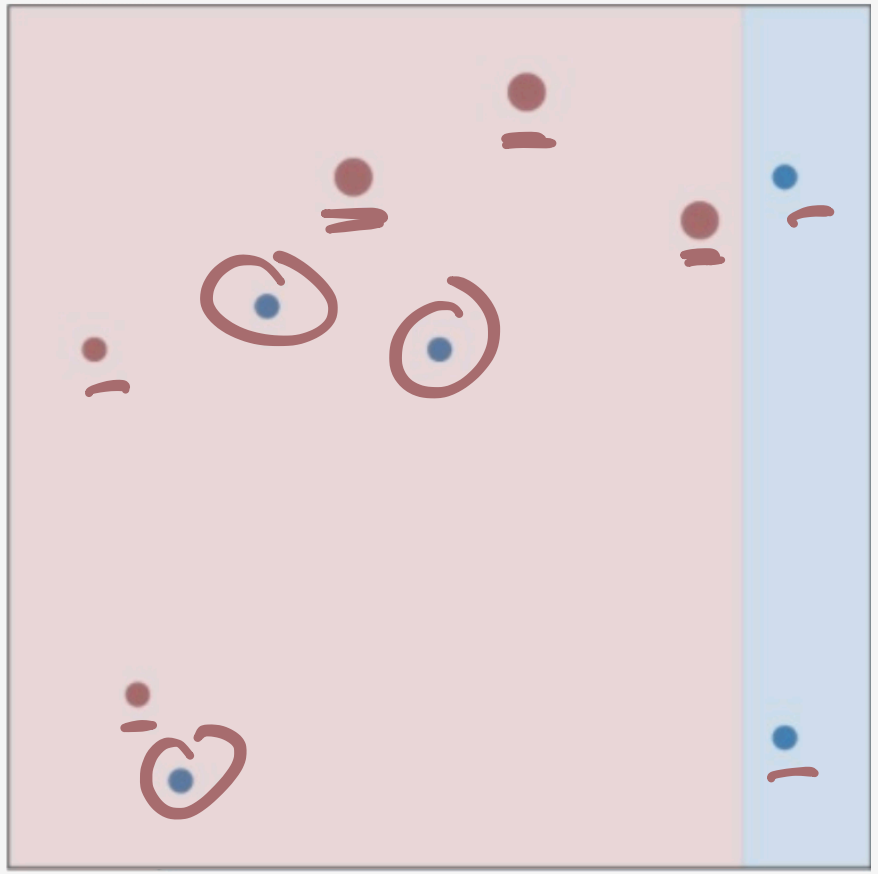
$$\begin{aligned} \text{err}_1 &= 0.30 \\ \alpha_1 &= 0.42 \end{aligned}$$



Adaboost Example

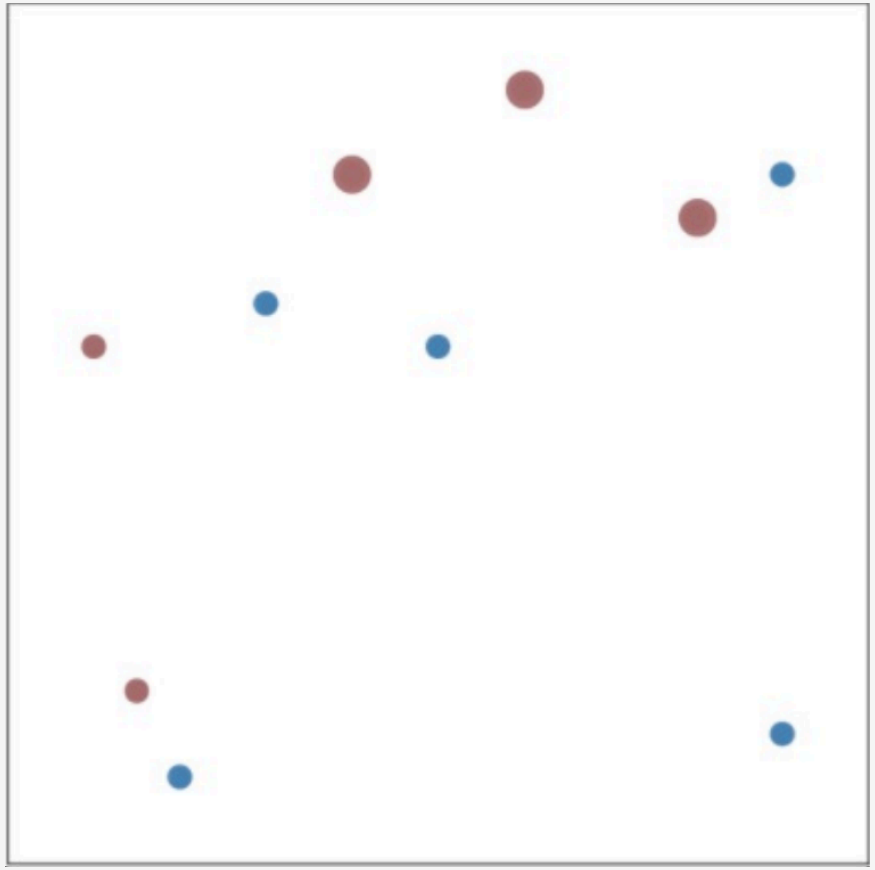
- **Second Decision Stump:**



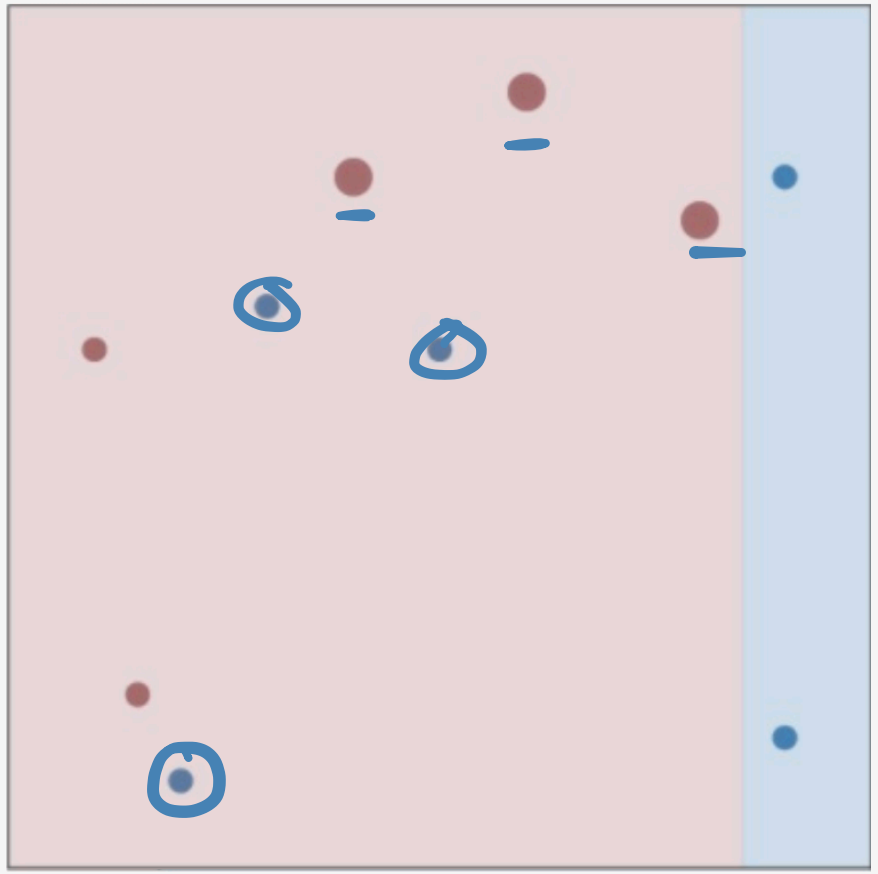


Adaboost Example

- **Second Decision Stump:**

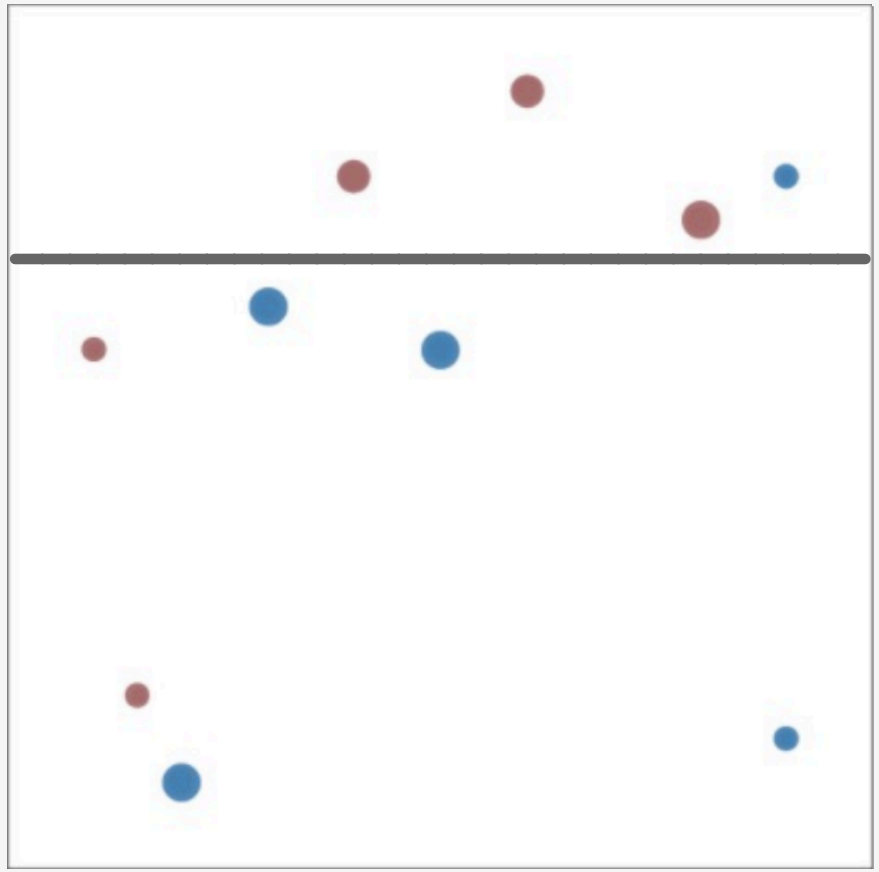


$$\begin{aligned} \text{err}_2 &= 0.21 \\ \alpha_2 &= 0.65 \end{aligned}$$

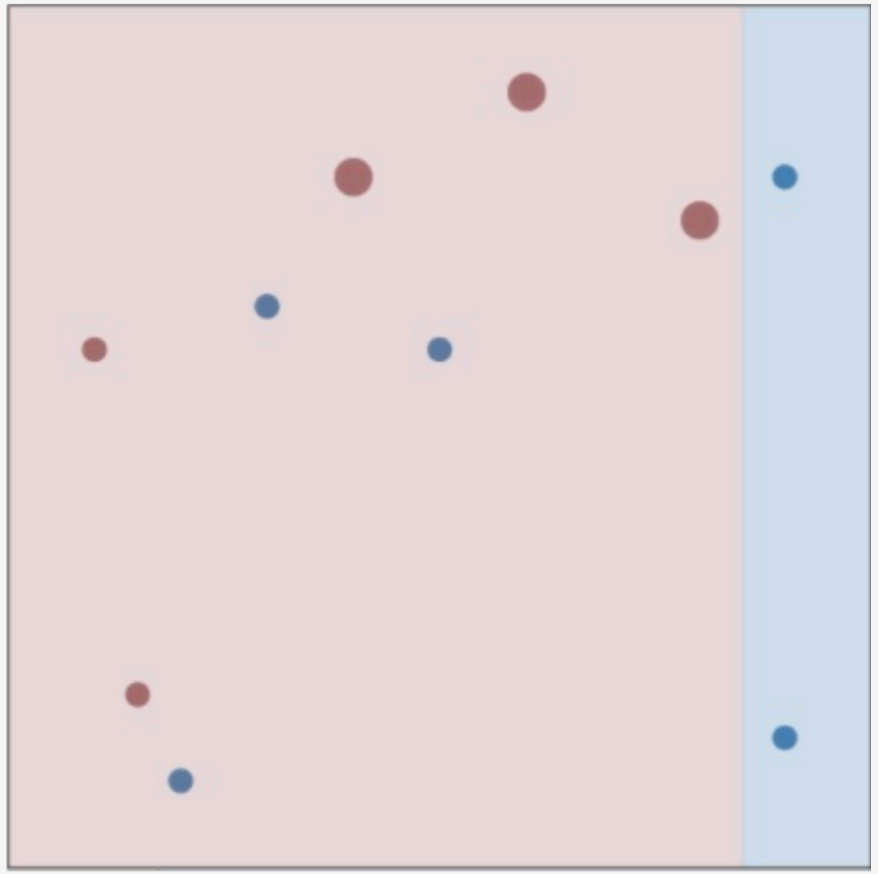


Adaboost Example

- **Second Decision Stump:**

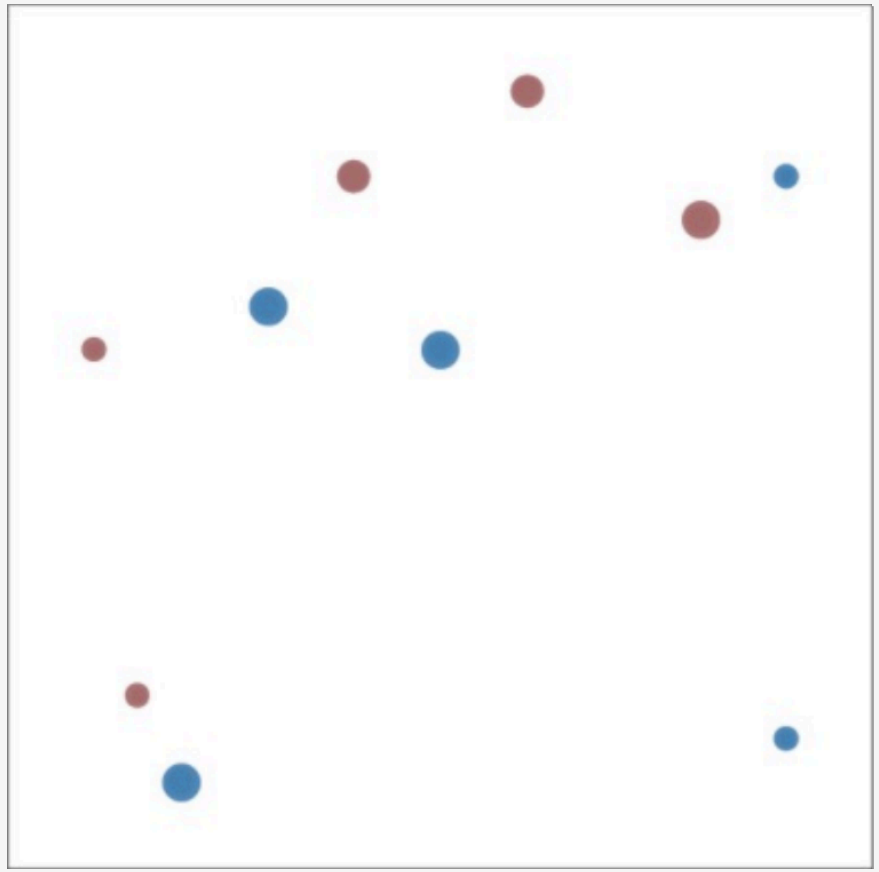


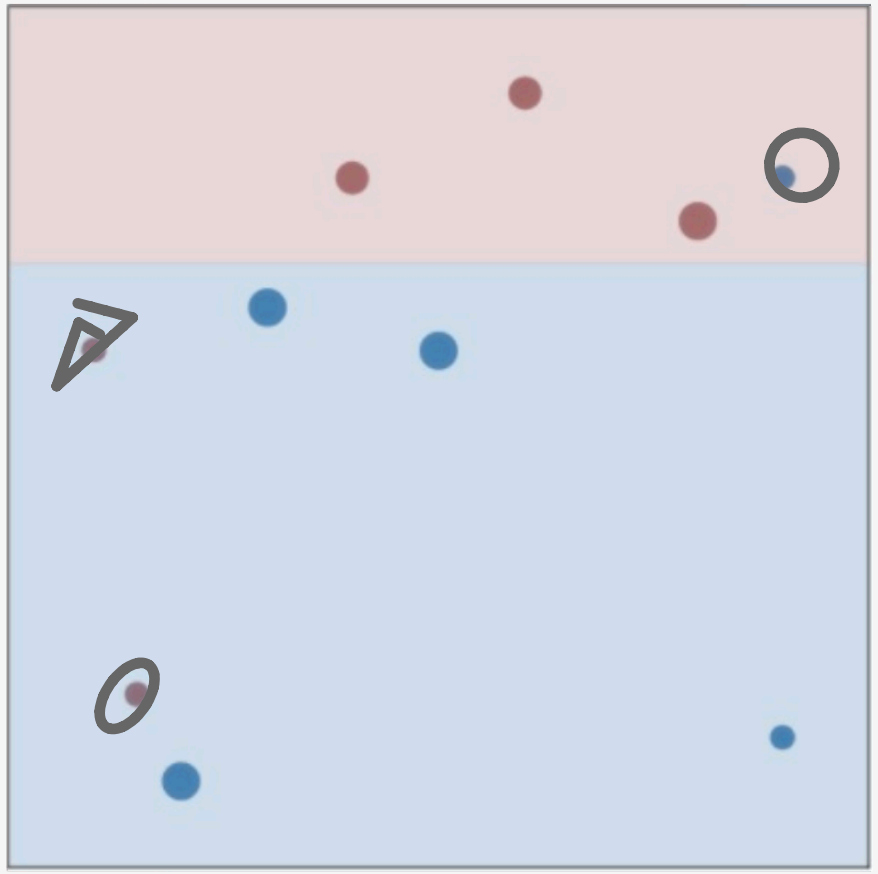
$$\begin{aligned} \text{err}_2 &= 0.21 \\ \alpha_2 &= 0.65 \end{aligned}$$



Adaboost Example

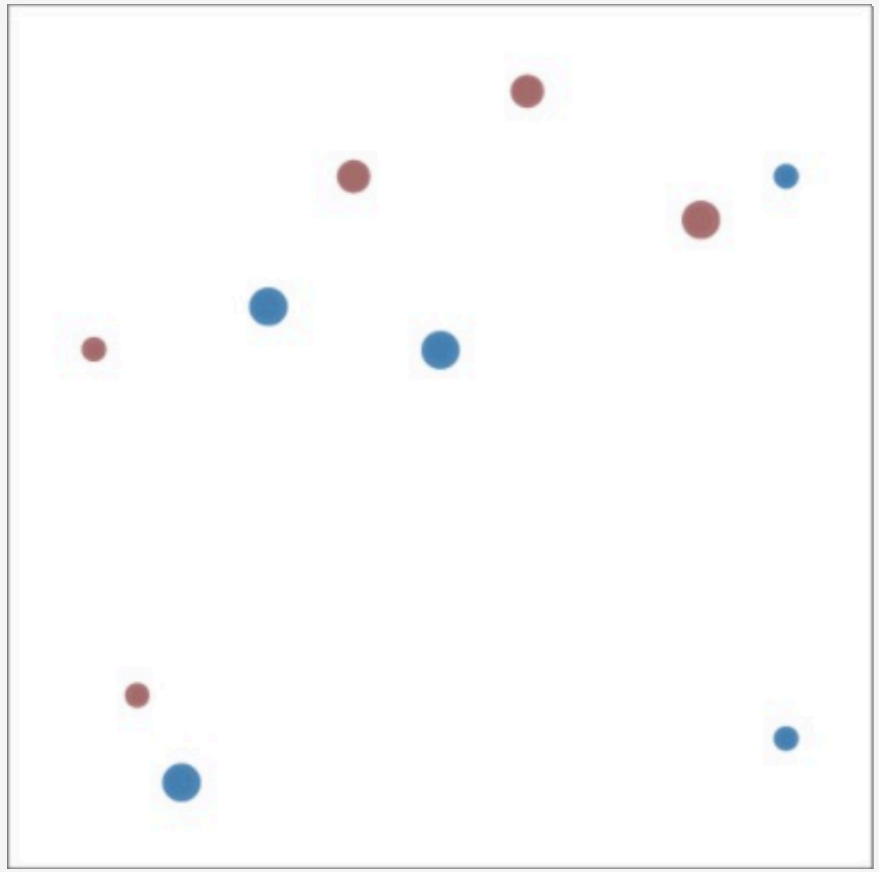
- **Third Decision Stump:**



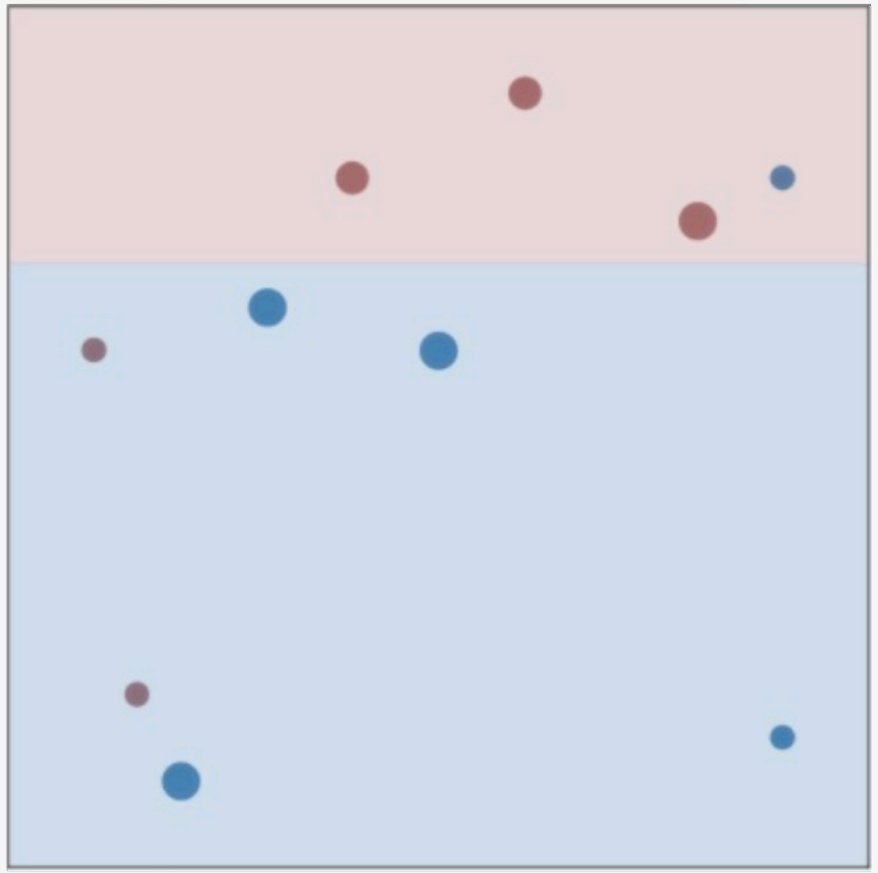


Adaboost Example

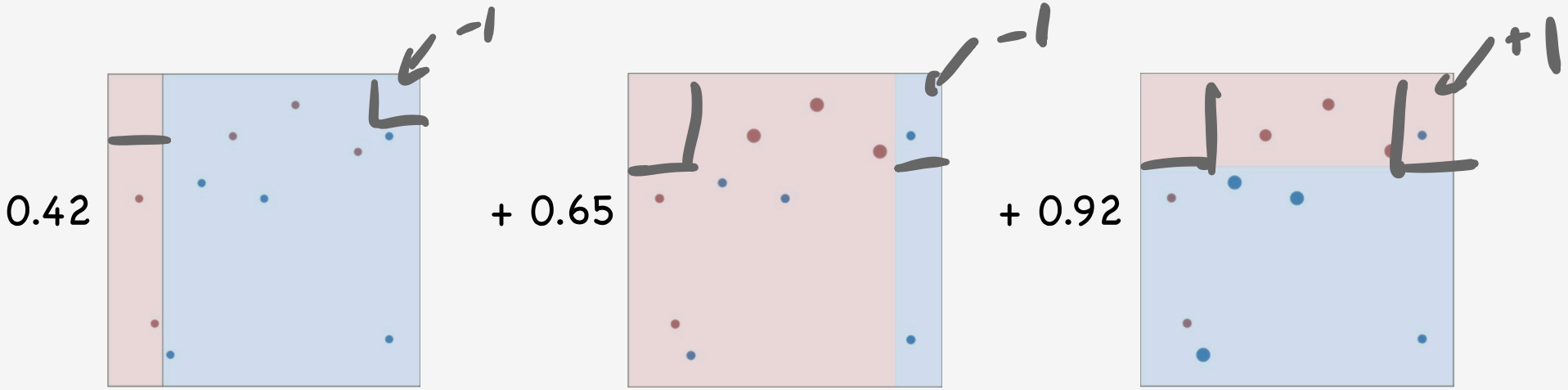
- **Third Decision Stump:**



$$\begin{aligned} \mathrm{err}_3 &= 0.14 \\ \alpha_3 &= 0.92 \end{aligned}$$

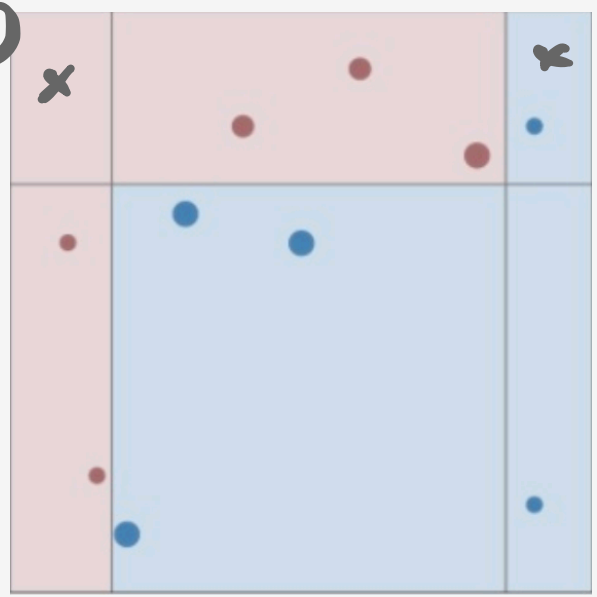


Example: Boosted Classifier



$.42(1) + .65(1) + .92(1)$
$= > 0$

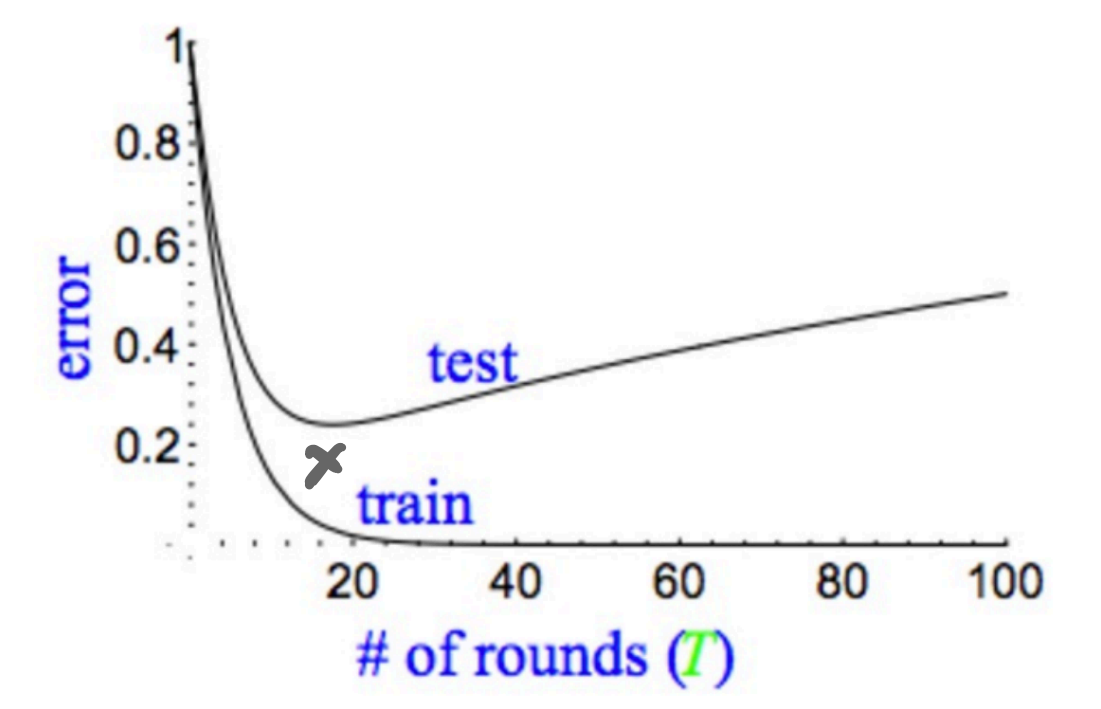
=



$.42(-1) + .65(-1) + .92(1)$
$= -1.07 + 0.92$
$= -0.15 < 0$
$\rightarrow -1$

Performance

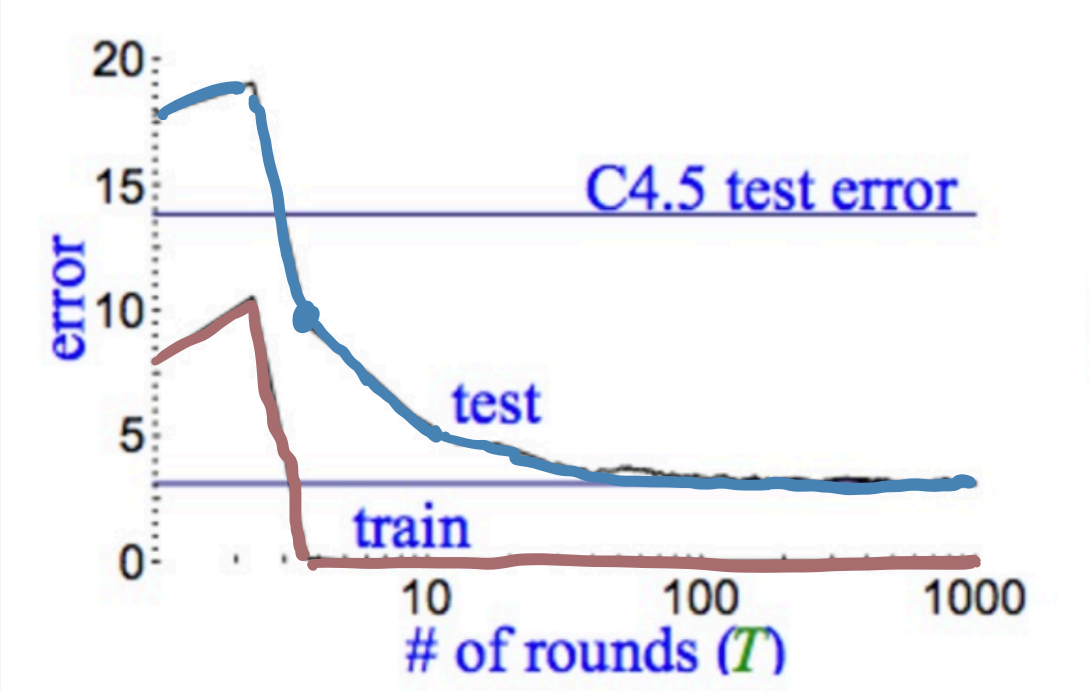
Recall the standard experiment of measuring train and validation error vs model complexity



Once overfitting begins, validation error goes up

Performance

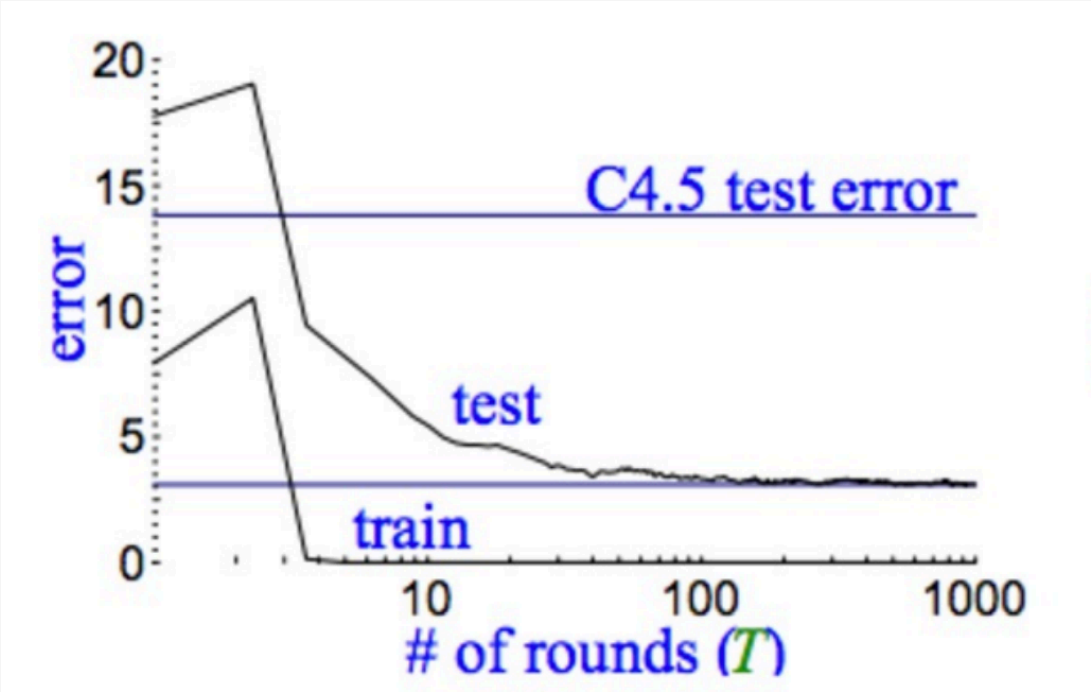
Boosting has a remarkable affect. Think of number of weak learners as model complexity



Even after training error becomes zero, validation error continues to improve!

Performance

HOW CAN THIS BE?!



Boosting as Margin Maximizing Algo

Adaboost actually induces a margin, and tries to maximize it

Despite the fact that we make the model more complex with each weak learner, the model actually becomes **more confident** with each boosting iteration

This confidence can be represented as a margin

Our learn classifier with K weak learners is $H(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x}) \right]$

Define an intermediate classifier by $H_\ell(\mathbf{x}) = \text{sign}\left[\sum_{k=1}^{\ell} \alpha_k h_k(\mathbf{x}) \right]$ where $\ell \leq K$

Boosting as Margin Maximizing Algo

Note that the classifier only returns +1 or -1, which doesn't give us any measure of confidence

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

Define the normalized voting coefficients as

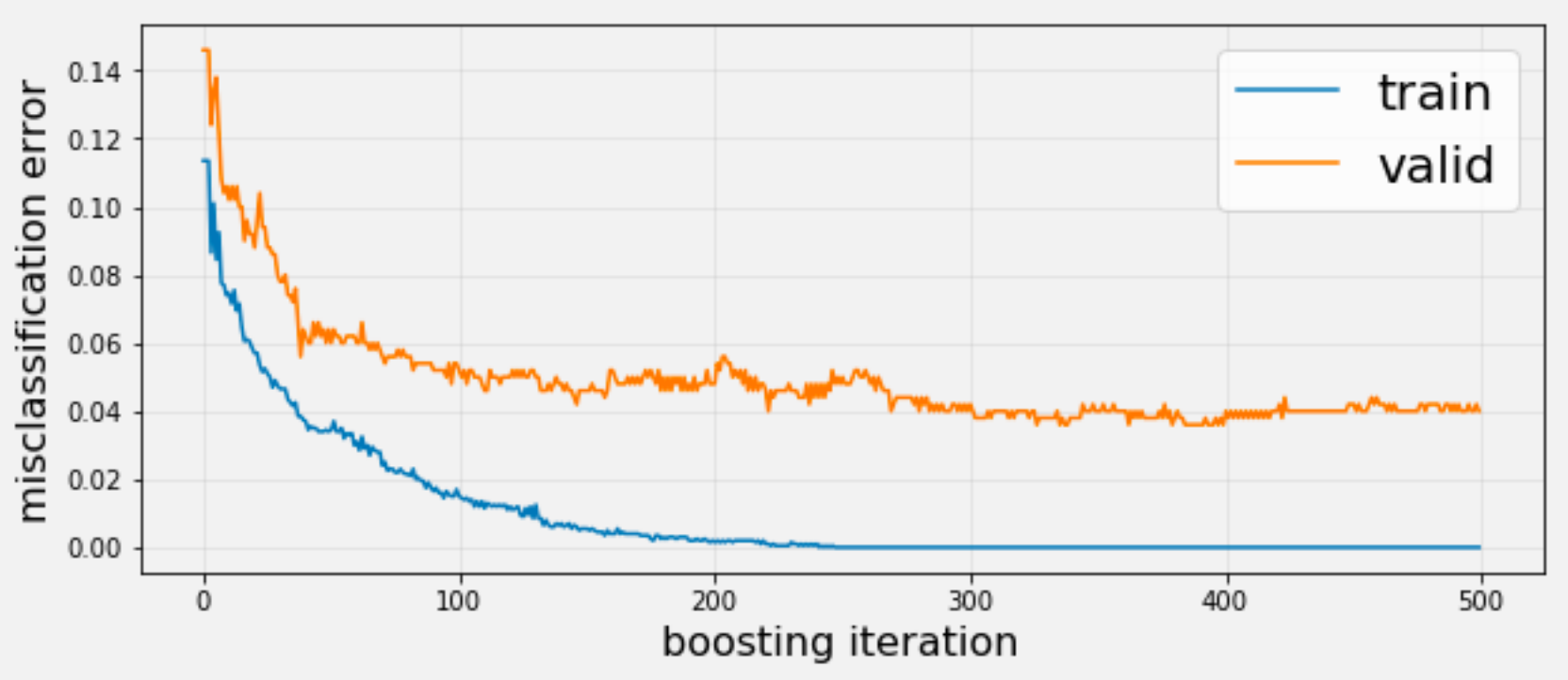
$$\hat{\alpha}_k = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j}$$

Define the margin of a training example after ℓ boosting iterations as

$$\text{margin}_\ell(\mathbf{x}) = \sum_{k=1:\ h_k(\mathbf{x})=y}^{\ell} \hat{\alpha}_k - \sum_{k=1:\ h_k(\mathbf{x})\neq y}^{\ell} \hat{\alpha}_k$$

Boosting as Margin Maximizing Algo

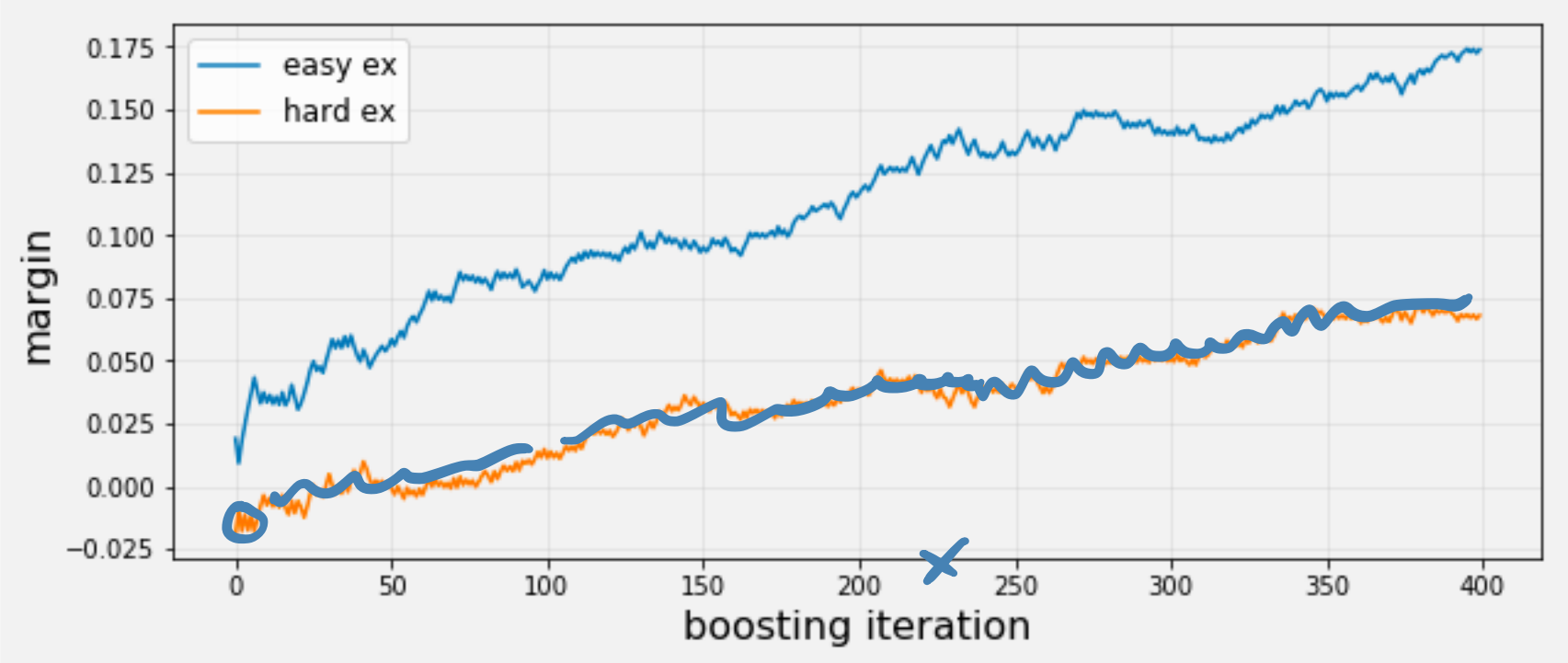
Handwritten Digit Recognition with Adaboost



Note that Adaboost achieves zero training error after roughly 250 iterations

Boosting as Margin Maximizing Algo

Handwritten Digit Recognition with Adaboost. Two easy and two hard training examples.



$$\text{margin}_\ell(\mathbf{x}) = \sum_{k=1:\ h_k(\mathbf{x})=y}^{\ell} \hat{\alpha}_k - \sum_{k=1:\ h_k(\mathbf{x})\neq y}^{\ell} \hat{\alpha}_k$$

Practical Advantages of Boosting

- It's fast!
- Simple and easy to program (you'll find out on Monday cuz we're gonna do it!)
- No parameters to really 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 features, continuous features)