

# DS 4400

## Machine Learning and Data Mining I

Alina Oprea  
Associate Professor  
Khoury College of Computer Science  
Northeastern University

November 3 2020

# Announcements

- Projects
  - We have received 21 projects
  - You will get an assigned TA
  - Project milestone will be due on November 24
  - Report of 3-4 pages on progress and challenges
- HW4 will be out soon
  - Due on Nov 17
- Exam is on Thursday, November 19
  - Work on assignments and review class lectures
  - Will share a list of topics

# Outline

- Ensemble learning
  - Review
- Boosting
  - General method
  - AdaBoost algorithm
  - Boosting vs Bagging
- Introduction to Deep Learning

# Ensemble Learning

Consider a set of classifiers  $h_1, \dots, h_L$

**Idea:** construct a classifier  $H(\mathbf{x})$  that combines the individual decisions of  $h_1, \dots, h_L$

- e.g., could have the member classifiers vote, or
- e.g., could use different members for different regions of the instance space

Successful ensembles require **diversity**

- Classifiers should make different mistakes
- Can have different types of base learners

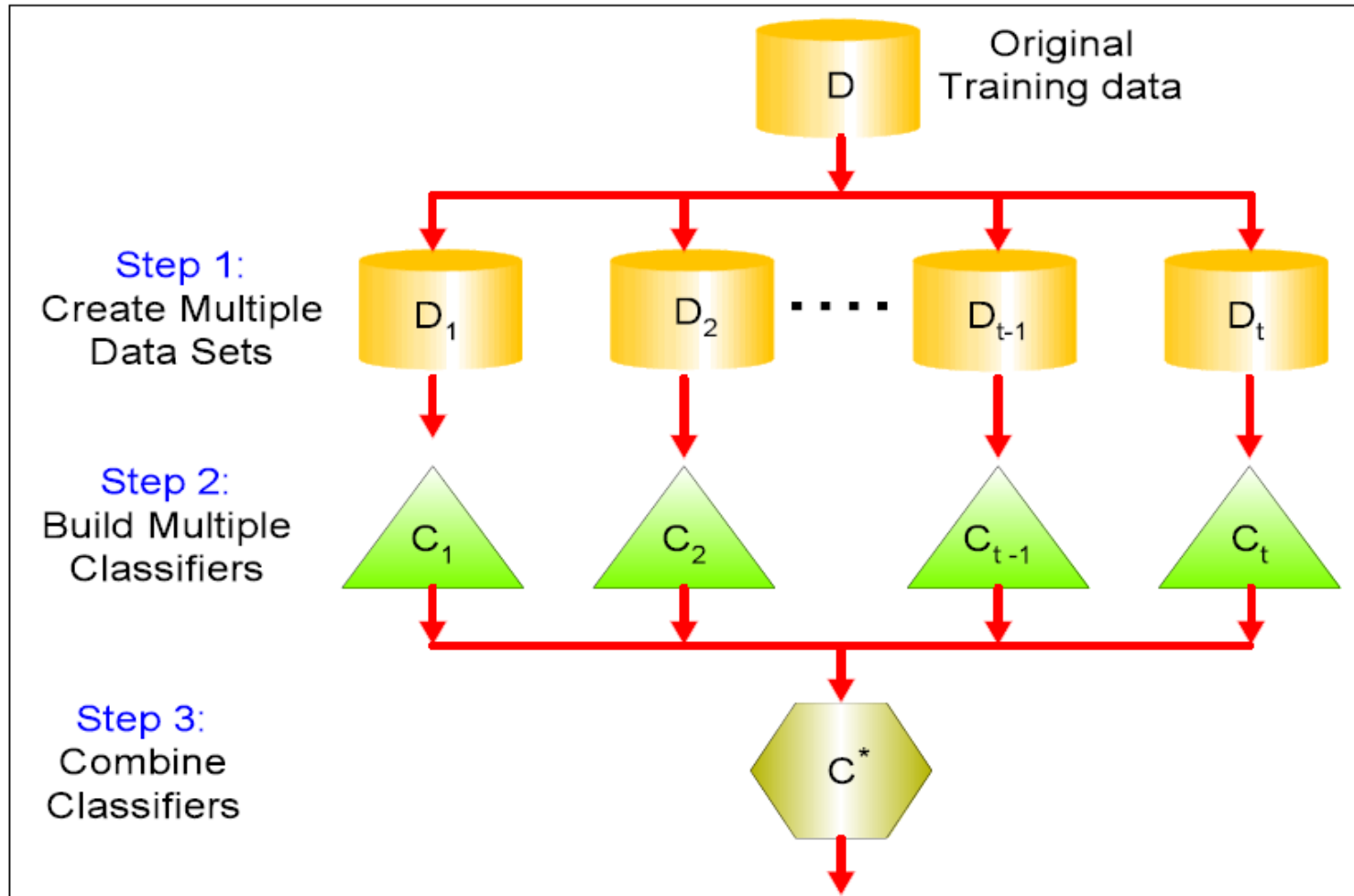
# How to Achieve Diversity

- Avoid overfitting
  - Vary the training data
- Features are noisy
  - Vary the set of features

Two main ensemble learning methods

- **Bagging** (e.g., Random Forests)
- **Boosting** (e.g., AdaBoost)

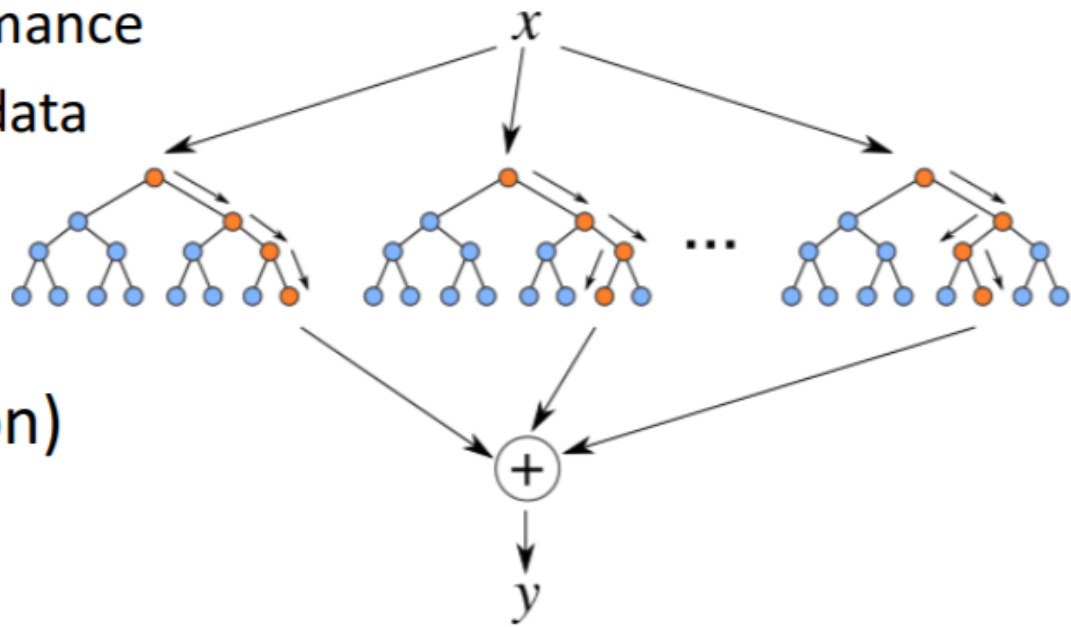
# Bagging



Majority Votes

# Random Forests

- Construct decision trees on bootstrap replicas
  - Restrict the node decisions to a small subset of features picked randomly for each node
- Do not prune the trees
  - Estimate tree performance on out-of-bootstrap data
- Average the output of all trees (or choose mode decision)



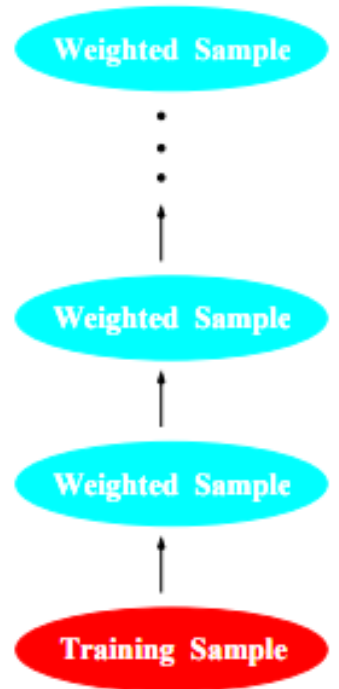
# AdaBoost

- A meta-learning algorithm with great theoretical and empirical performance
- Turns a base learner (i.e., a “weak hypothesis”) into a high performance classifier
- Creates an ensemble of weak hypotheses by repeatedly emphasizing mispredicted instances

Adaptive Boosting  
Freund and Schapire 1997



# Overview of AdaBoost



**FIGURE 10.1.** *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

# Boosting [Shapire '89]

- **Idea:** given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration  $t$ :
  - weight each training example by how incorrectly it was classified
  - Learn a weak hypothesis –  $h_t$
  - A strength for this hypothesis –  $\beta_t$
- Final classifier:  $H(x) = \text{sign}(\sum \beta_t h_t(x))$

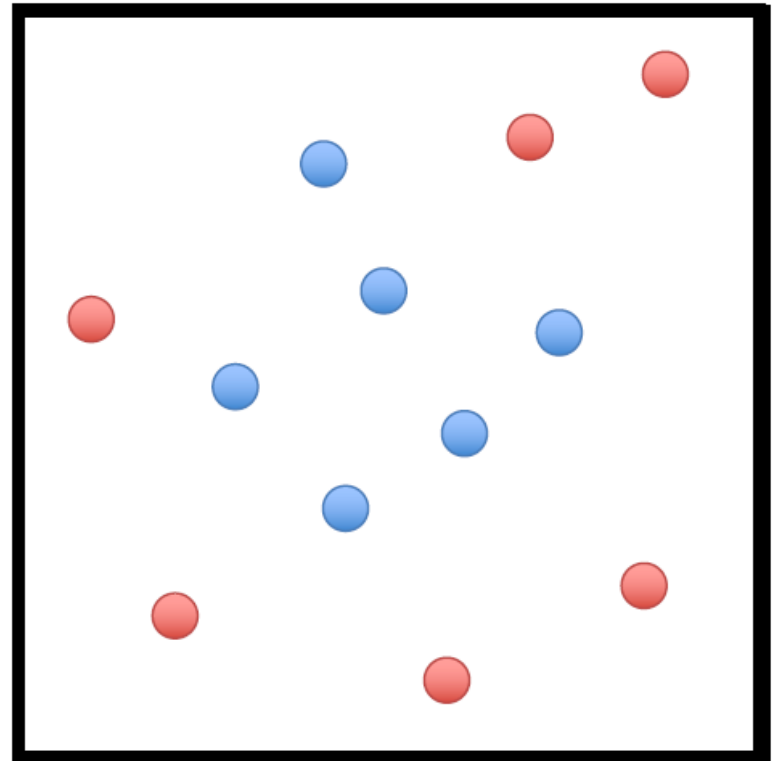
## Convergence bounds with minimal assumptions on weak learner

If each weak learner  $h_t$  is slightly better than random guessing ( $\epsilon_t < 0.5$ ), then training error of AdaBoost decays exponentially fast in number of rounds  $T$ .

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

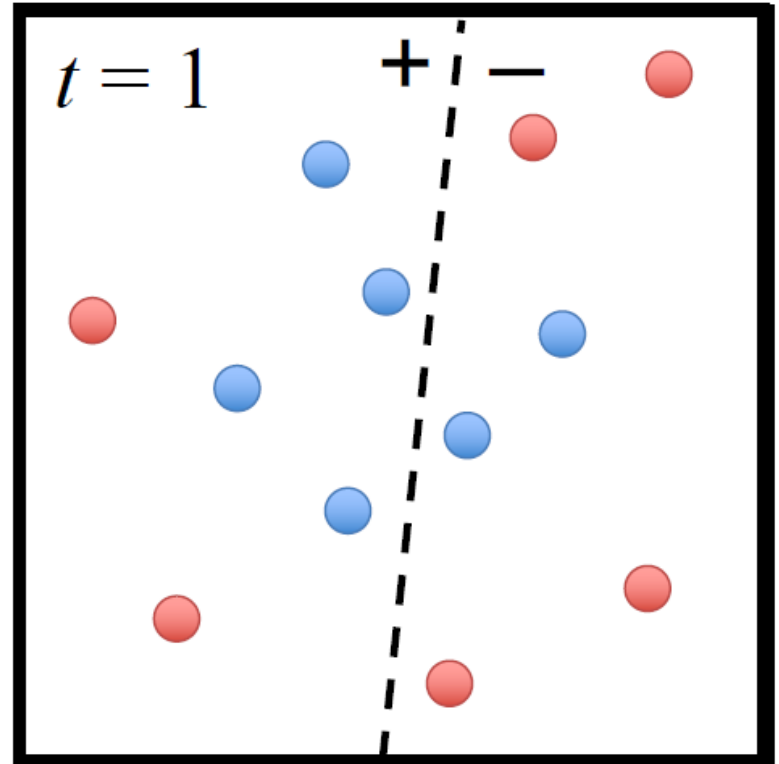


- Size of point represents the instance's weight

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

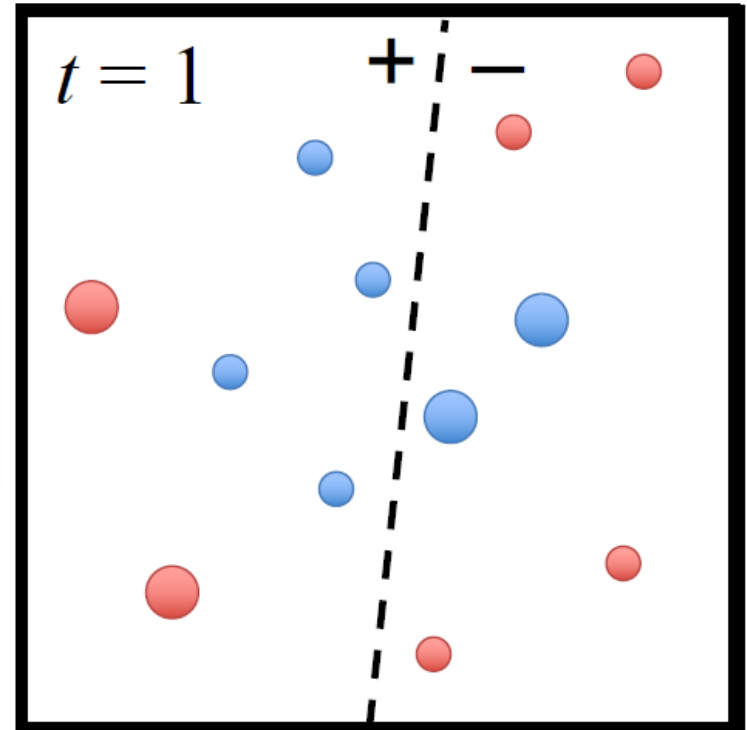


- $\beta_t$  measures the importance of  $h_t$
- If  $\epsilon_t \leq 0.5$ , then  $\beta_t \geq 0$  (can trivially guarantee)

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

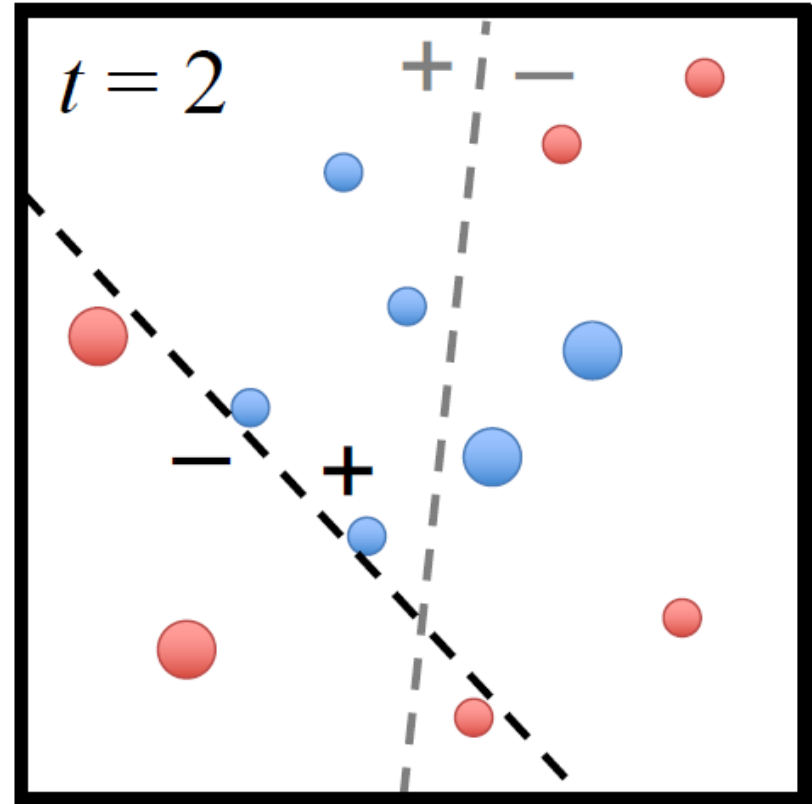


- Weights of correct predictions are multiplied by  $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by  $e^{\beta_t} \geq 1$

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



- Compute importance of hypothesis  $\beta_t$
- Update weights  $w_t$

# AdaBoost

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$

2: **for**  $t = 1, \dots, T$

3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$

4:   Compute the weighted training error of  $h_t$

5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$

6:   Update all instance weights:

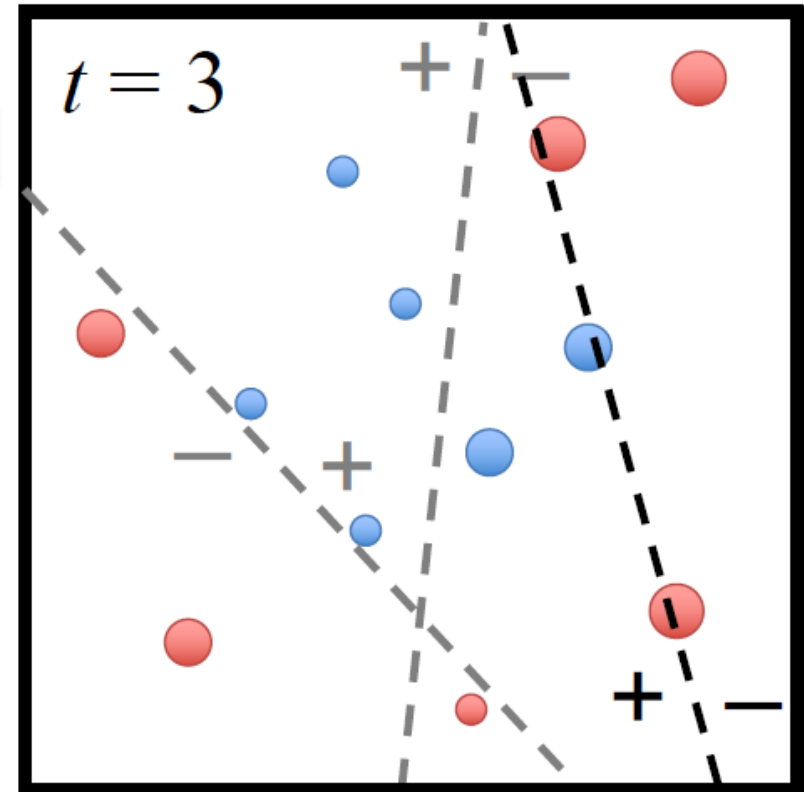
$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$$

7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



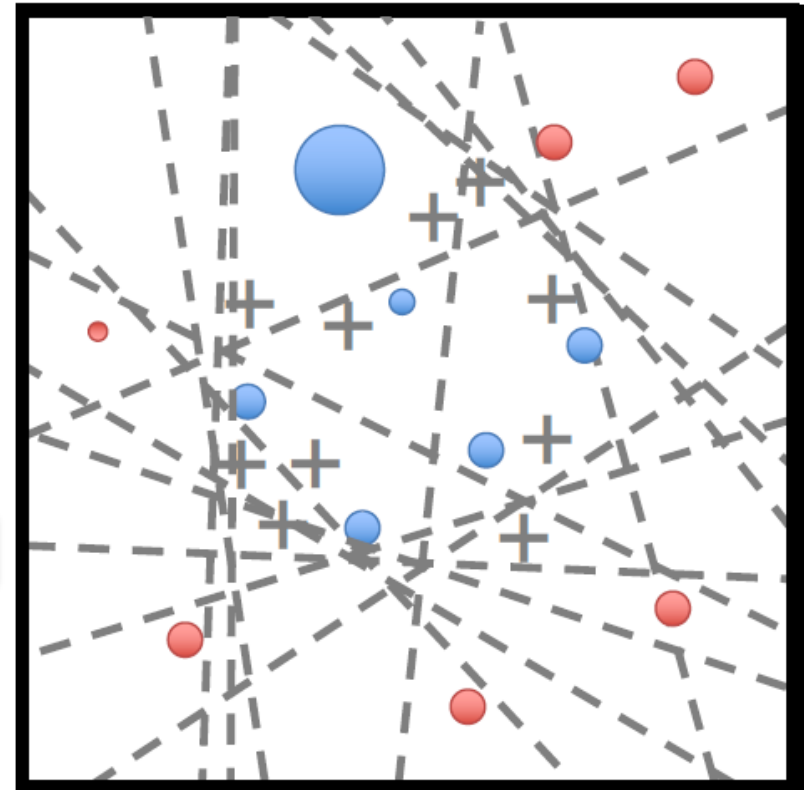
- Compute importance of hypothesis  $\beta_t$
- Update weights  $w_t$

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

$t = T$



- Final model is a weighted combination of members
  - Each member weighted by its importance



# AdaBoost

**INPUT:** training data  $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ ,  
the number of iterations  $T$

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$

2: **for**  $t = 1, \dots, T$

3:   Train model  $h_t$  on  $X, y$  with instance weights  $\mathbf{w}_t$

4:   Compute the weighted training error rate of  $h_t$ :

$$\epsilon_t = \sum_{i: y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$

6:   Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i)) \quad \forall i = 1, \dots, n$$

7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^n w_{t+1,j}} \quad \forall i = 1, \dots, n$$

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

# Train with Weighted Instances

- For algorithms like logistic regression, can simply incorporate weights  $w$  into the cost function
  - Essentially, weigh the cost of misclassification differently for each instance

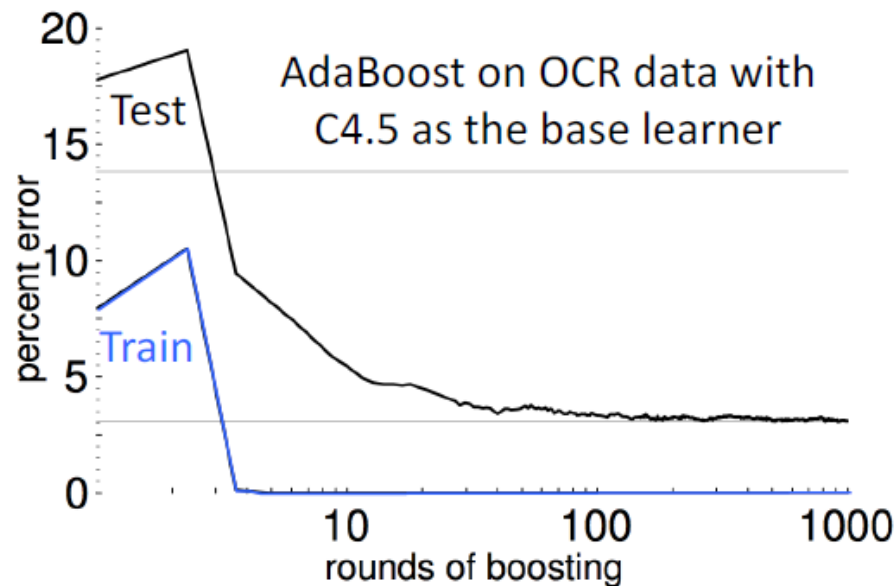
$$J_{\text{reg}}(\boldsymbol{\theta}) = - \sum_{i=1}^n w_i [y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

- For algorithms that don't directly support instance weights (e.g., ID3 decision trees, etc.), use weighted bootstrap sampling
  - Form training set by resampling instances with replacement according to  $w$

# Properties

- If a point is repeatedly misclassified
  - Its weight is increased every time
  - Eventually it will generate a hypothesis that correctly predicts it
- In practice AdaBoost does not typically overfit
- Does not use explicitly regularization

# Resilience to overfitting



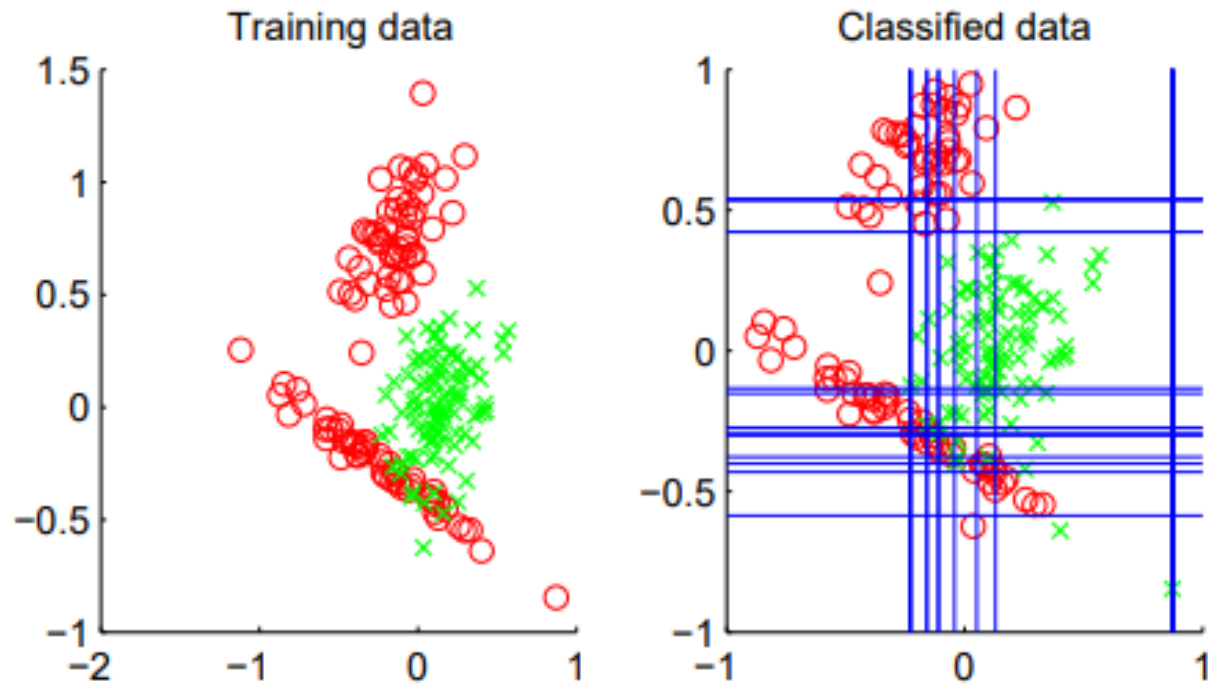
- Empirically, boosting resists overfitting
- Note that it continues to drive down the test error even AFTER the training error reaches zero

Increases confidence in prediction when adding more rounds

# Base Learner Requirements

- AdaBoost works best with “weak” learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in  $O(\log n)$  iterations
- Examples:
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

# AdaBoost with Decision Stumps



# AdaBoost in Practice

## Strengths:

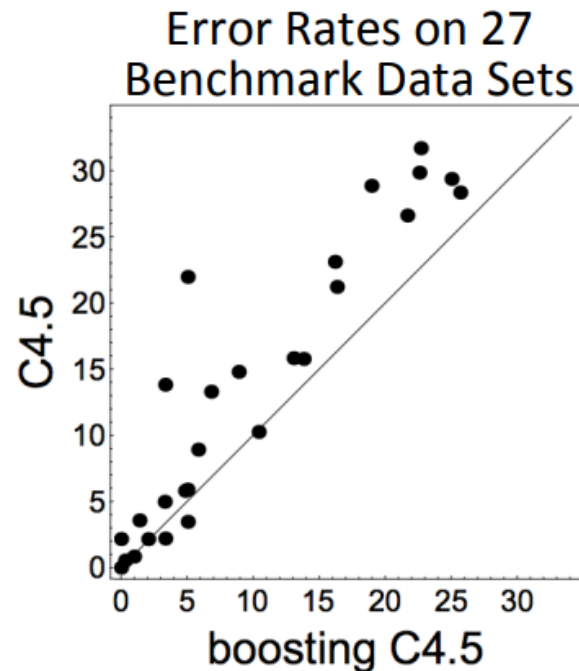
- Fast and simple to program
- No parameters to tune (besides T) **Learn with Cross-Validation**
- No assumptions on weak learner **Error less than  $\frac{1}{2}$**

## When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

# Boosted Decision Trees

- Boosted decision trees are one of the best “off-the-shelf” classifiers
  - i.e., no parameter tuning
- Limit member hypothesis complexity by limiting tree depth
- Gradient boosting methods are typically used with trees in practice



“AdaBoost with trees is the best off-the-shelf classifier in the world” -Breiman, 1996  
(Also, see results by Caruana & Niculescu-Mizil, ICML 2006)



# Bagging vs Boosting

## Bagging

vs.

## Boosting

Resamples data points

Weight of each classifier is the same

Only variance reduction

Applicable to complex models with low bias, high variance

Reweights data points (modifies their distribution)

Weight is dependent on classifier's accuracy

Both bias and variance reduced – learning rule becomes more complex with iterations

Applicable to weak models with high bias, low variance

# Review Ensembles

- Ensemble learning are powerful learning methods
  - Better accuracy than standard classifiers
- Bagging uses bootstrapping (with replacement), trains  $T$  models, and averages their prediction
  - Random forests vary training data and feature set at each split
- Boosting is an ensemble of  $T$  weak learners that emphasizes mis-predicted examples
  - AdaBoost has great theoretical and experimental performance
  - Can be used with linear models or simple decision trees (stumps, fixed-depth decision trees)

# Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Thanks!