DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

February 19 2019

Logistics

- HW3 is due on Friday, February 22
- Project proposal due on Tuesday 02/26
  - 1 page description of your project, including problem statement, dataset, and ML algorithms
- Week of February 25
  - Lecture on 02/26 taught by Lisa Friedland
  - Lecture on 02/28 canceled

Outline

- Ensemble learning review
  - Bagging and Random Forests
- Boosting
  - AdaBoost
  - Comparing Boosting and Bagging
- Density Estimation

Ensemble Learning

Consider a set of classifiers h_1, ..., h_L

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

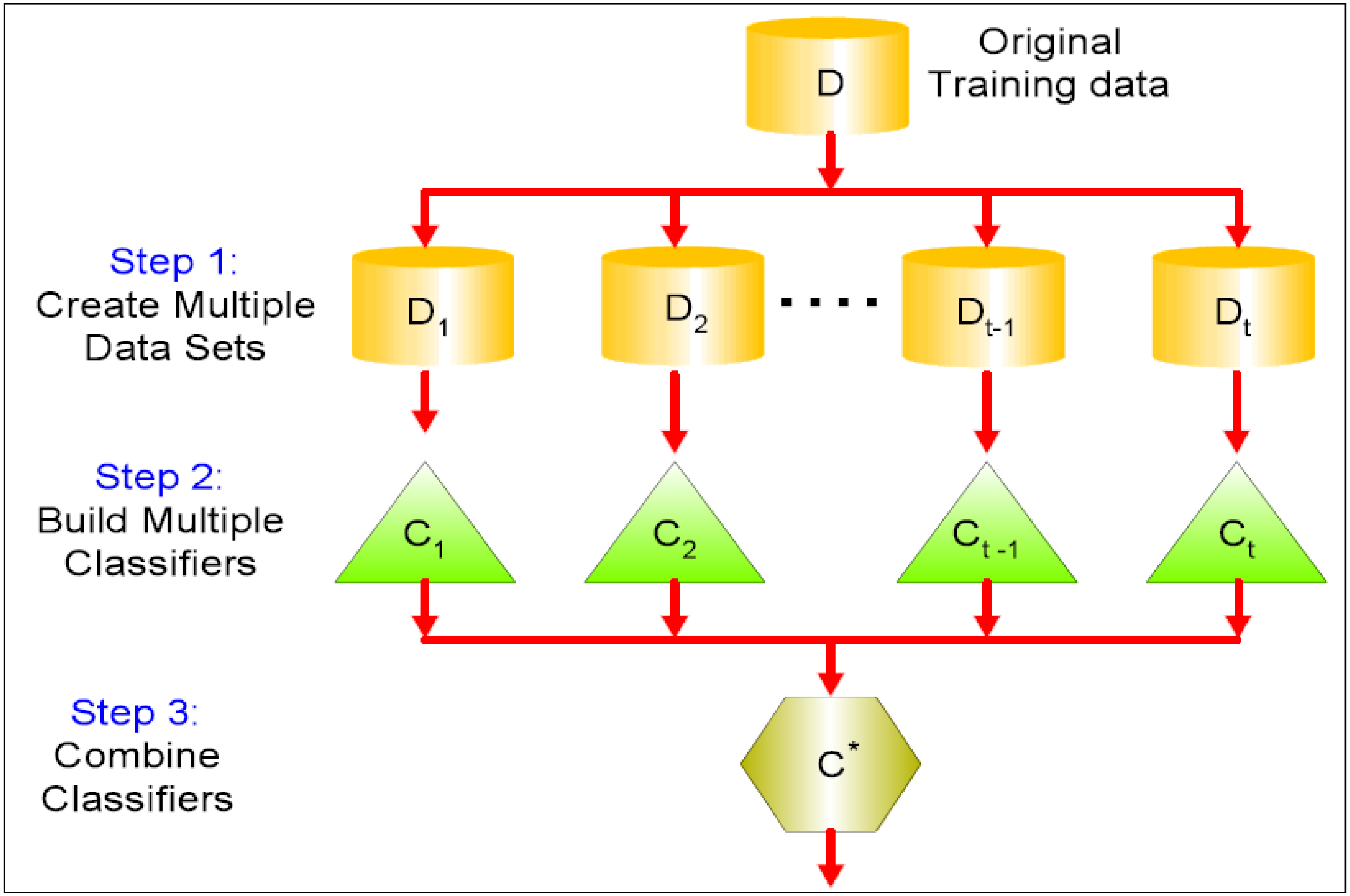
- Bagging (e.g., Random Forests) Parallel
- Boosting (e.g., AdaBoost) Sequential

Bagging

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set D
- *Bootstrap sampling*: Given set D containing N training examples, create D' by drawing N examples at random with replacement from D.

- Bagging:
  - Create k bootstrap samples $D_1 \ldots D_k$.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

General Idea



Majority Votes

Random Forest Algorithm

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

If $m = p$, this is equivalent to Bagging

Random Forest uses $m = \sqrt{p}$

Lab

```
>
> library(randomForest)
> rf.carseats=randomForest(High~.-Sales,Carseats,subset=train,importance=TRUE)
> rf.carseats

Call:
 randomForest(formula = High ~ . - Sales, data = Carseats, importance = TRUE,      subset = train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 18.5%
Confusion matrix:
     No Yes class.error
No  104  14   0.1186441
Yes  23  59   0.2804878
```

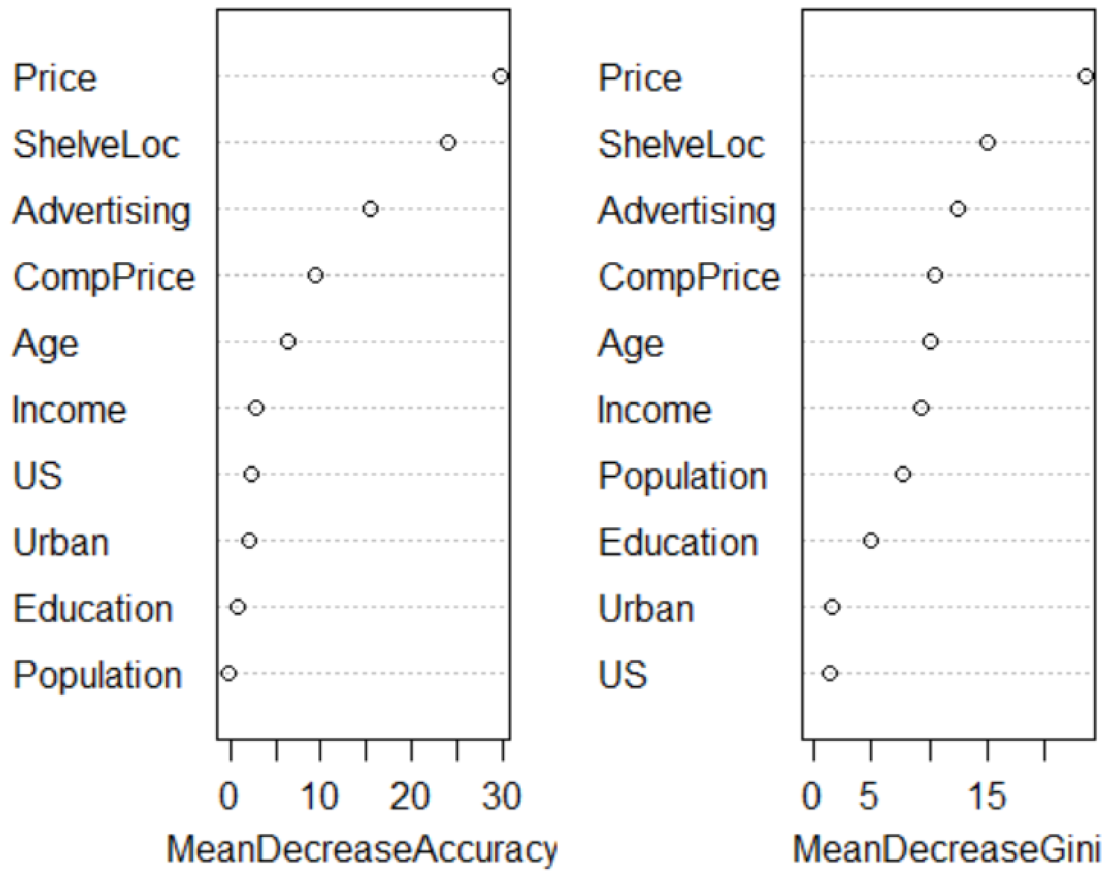
```
>
> rf.pred=predict(rf.carseats,Carseats.test,type="class")
> table(rf.pred,High.test)
       High.test
rf.pred  No Yes
    No  105  25
    Yes  13  57
> mean(rf.pred==High.test)
[1] 0.81
```

Lab

```
> importance(rf.carseats,type=2)
            MeanDecreaseGini
CompPrice          10.444114
Income              9.204883
Advertising        12.367002
Population          7.722053
Price              23.437998
ShelveLoc          15.053694
Age                10.135102
Education           4.879102
Urban               1.585268
US                  1.369725
```

Lab

```
>
> varImpPlot(rf.carseats)
>
```



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)

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

- Mis-classified examples get higher weights
- Correct examples get lower weights

Better classifiers will get higher weights

Uniform weights

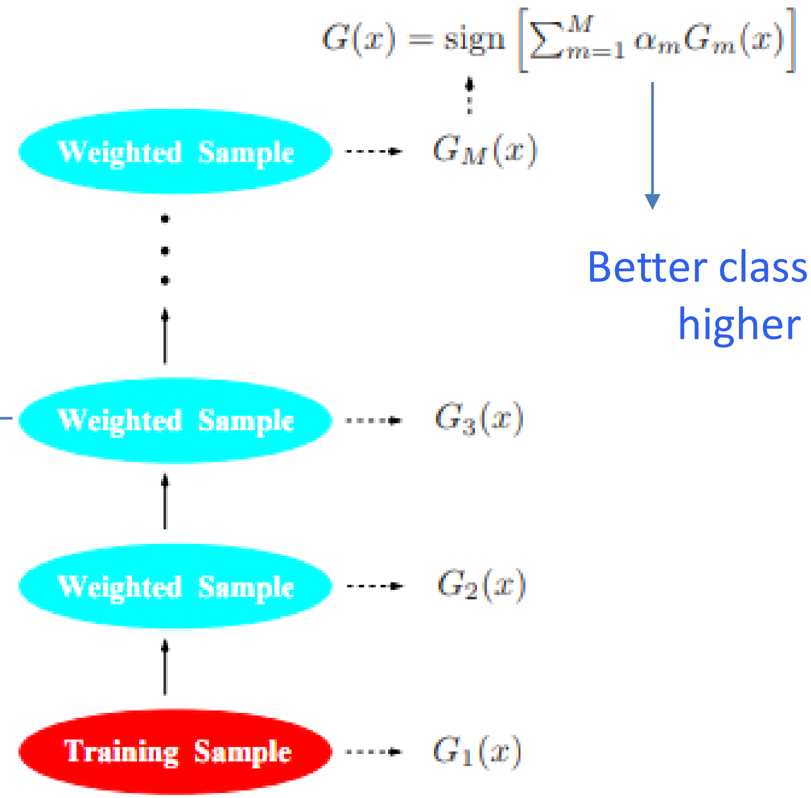


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

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 – α_t

- Final classifier: $H(X) = \text{sign}(\sum \alpha_t h_t(X))$

Convergence bounds with minimal assumptions on weak learner

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

Power of Boosting

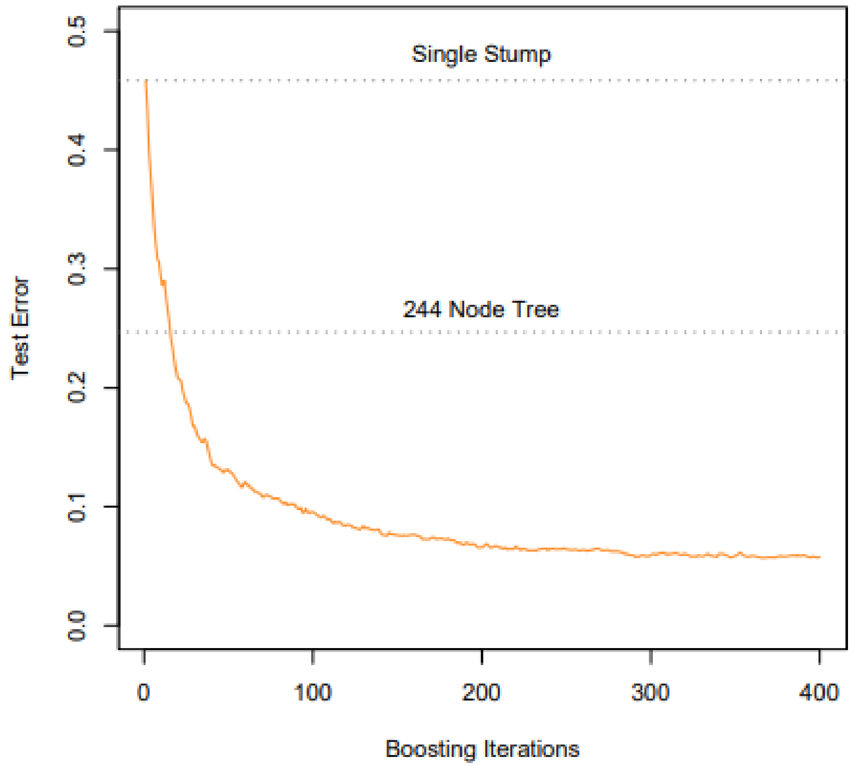


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

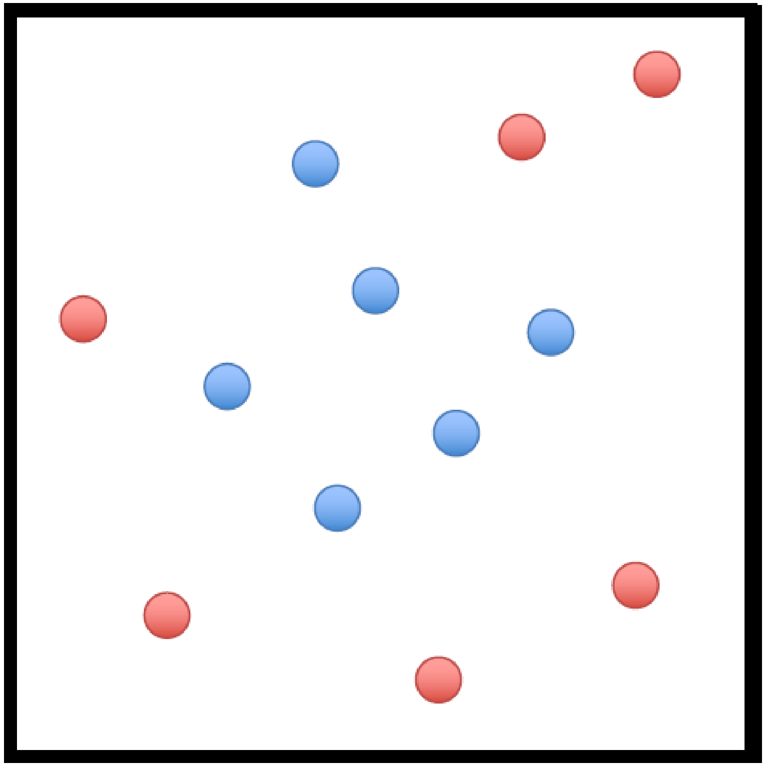
AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
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:

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

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

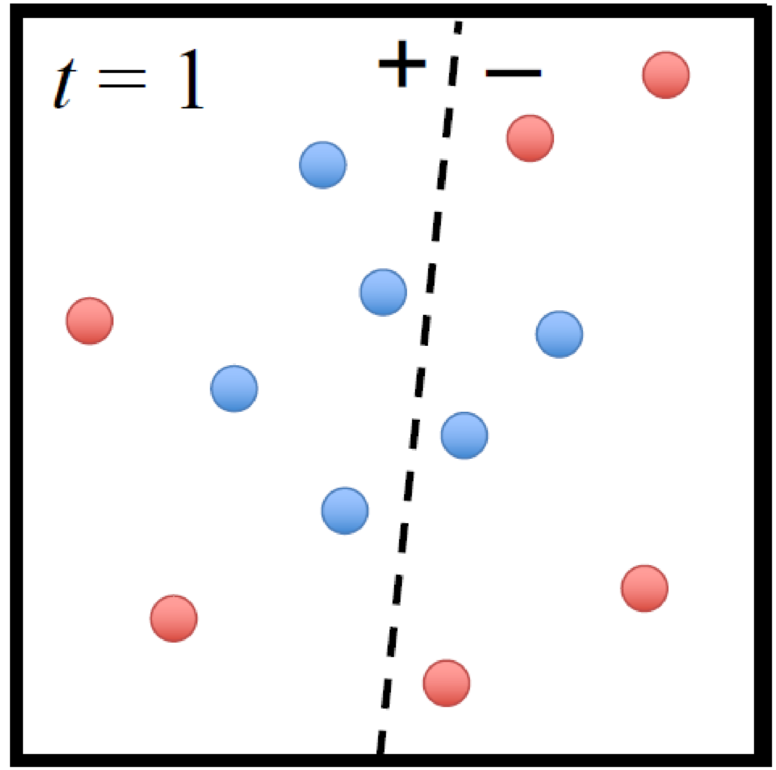
AdaBoost

Initialize a vector of n uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model h_t on X, y with weights $\mathbf{w}_t$
    Compute the weighted training error of h_t
    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
    Update all instance weights:

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

    Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis

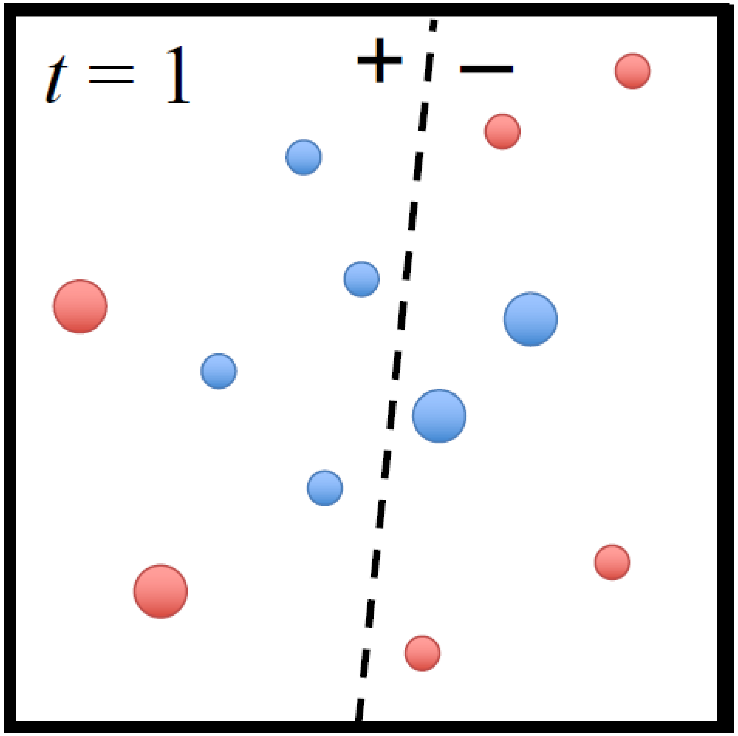
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



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

AdaBoost

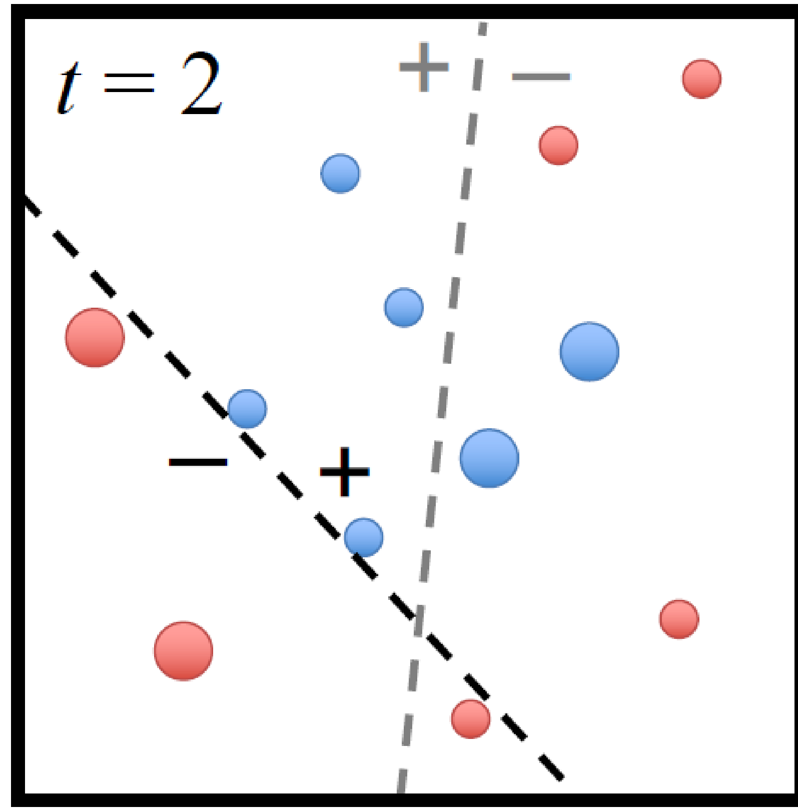
1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights: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

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



- Compute importance of hypothesis β_t
- Update weights w_t

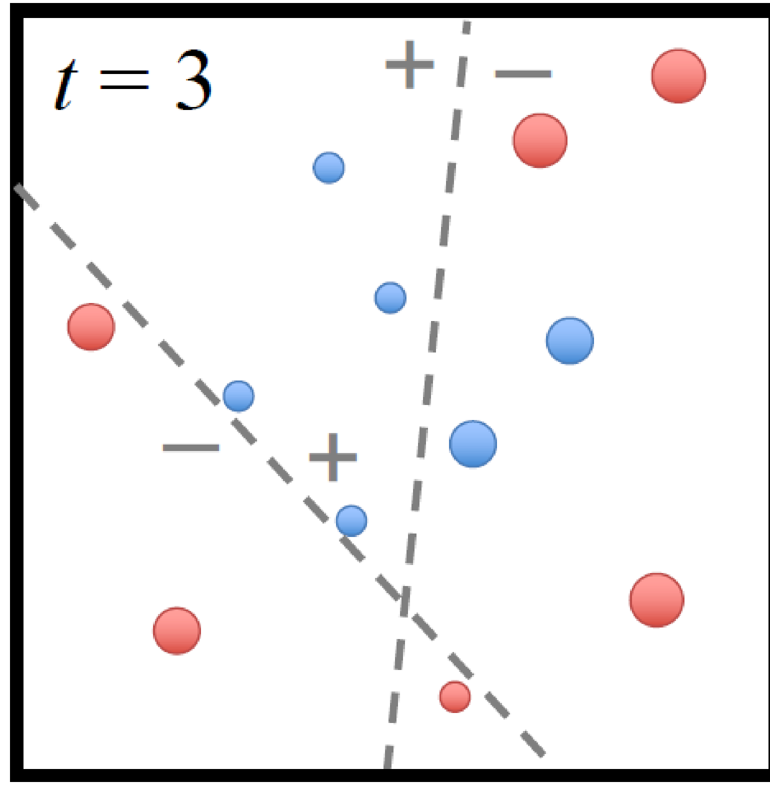
AdaBoost

Initialize a vector of n uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model h_t on X, y with weights $\mathbf{w}_t$
    Compute the weighted training error of h_t
    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance weights:

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

    Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis

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



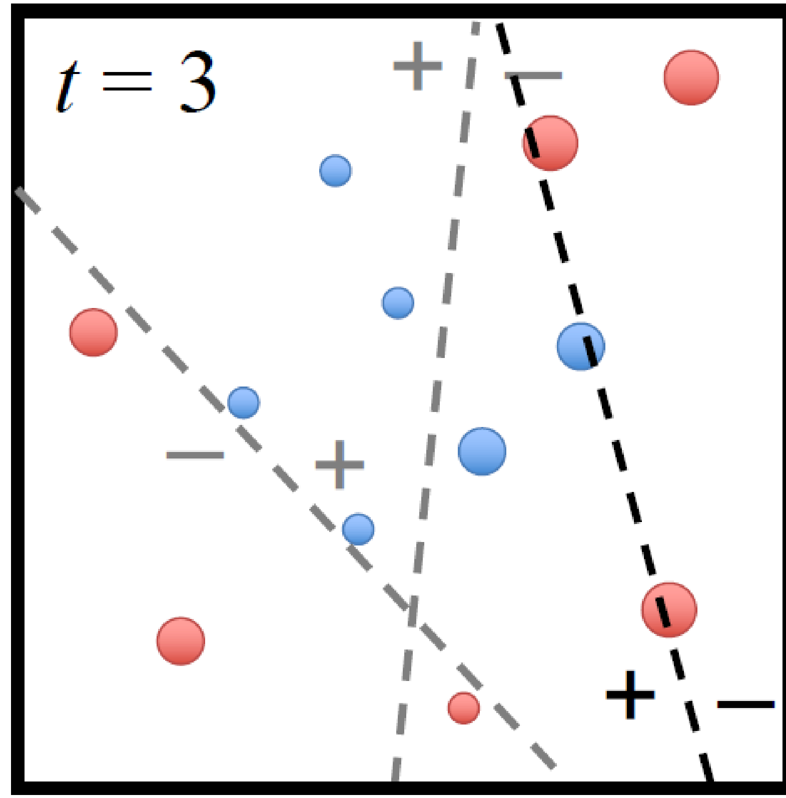
AdaBoost

Initialize a vector of n uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
  Train model h_t on X, y with weights $\mathbf{w}_t$
  Compute the weighted training error of h_t
  Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
  Update all instance weights:

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

  Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis

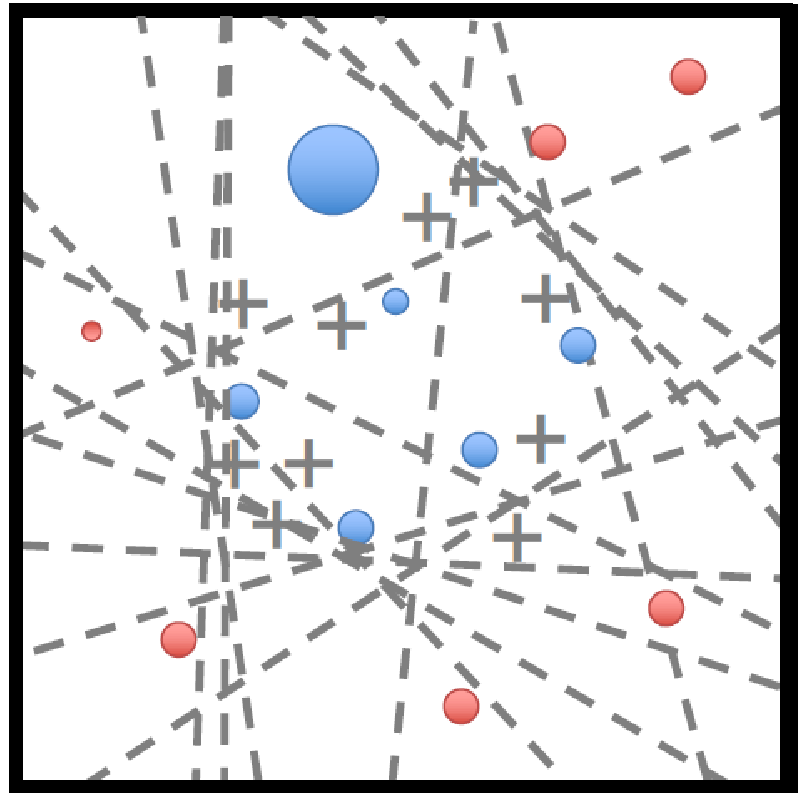
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



- Compute importance of hypothesis β_t
- Update weights w_t

AdaBoost

$t = \mathrm{T}$



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

for $t = 1, \ldots, T$

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

    Compute the weighted training error of h_t

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

    Update all instance weights:

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

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

end for

Return the hypothesis

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

- 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 $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
2: **for** $t = 1, \ldots, T$
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:

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

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

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

8: **end for**
9: **Return** the hypothesis

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

Greedy Algorithm

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

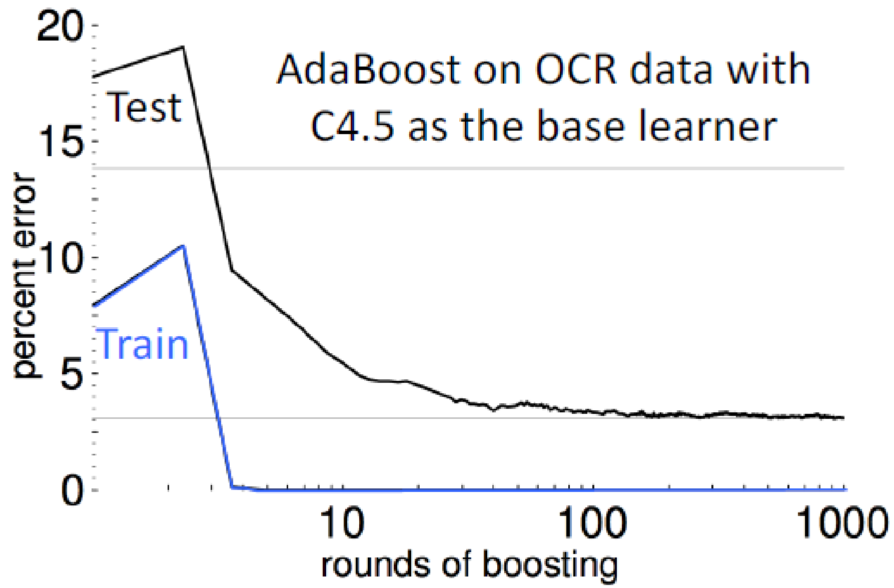
$$J_{\text{reg}}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} w_i \left[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log \left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)\right] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

- 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

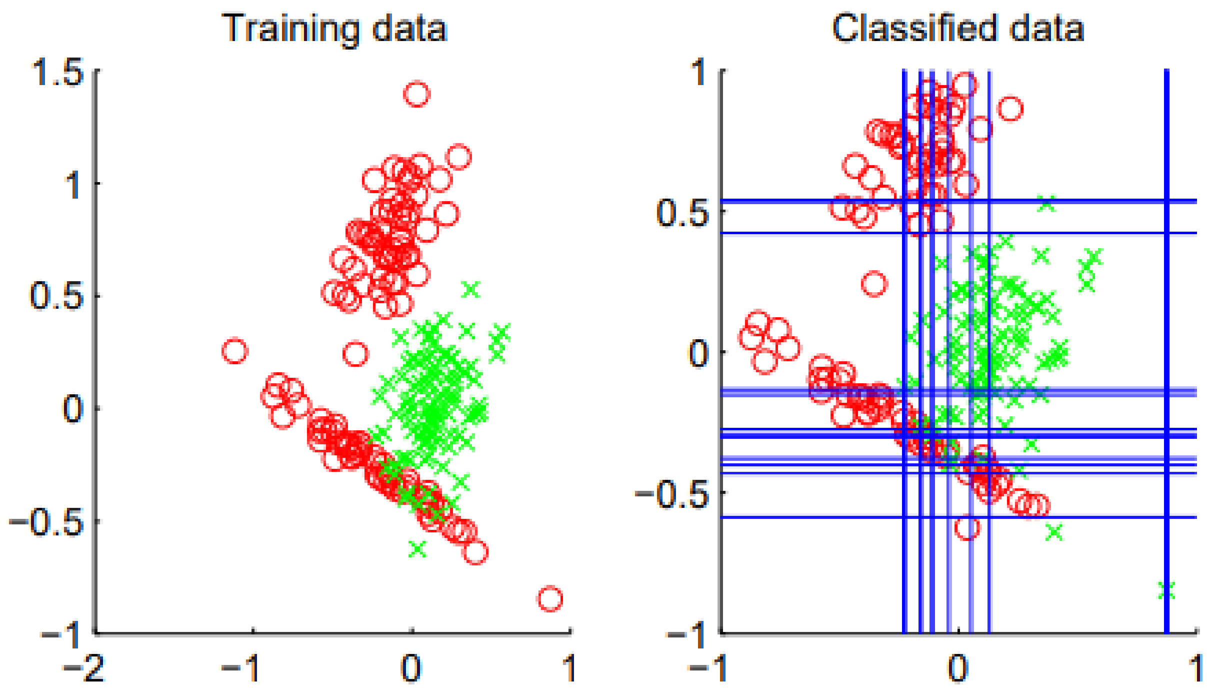
Increases confidence in prediction when adding more rounds

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 ½

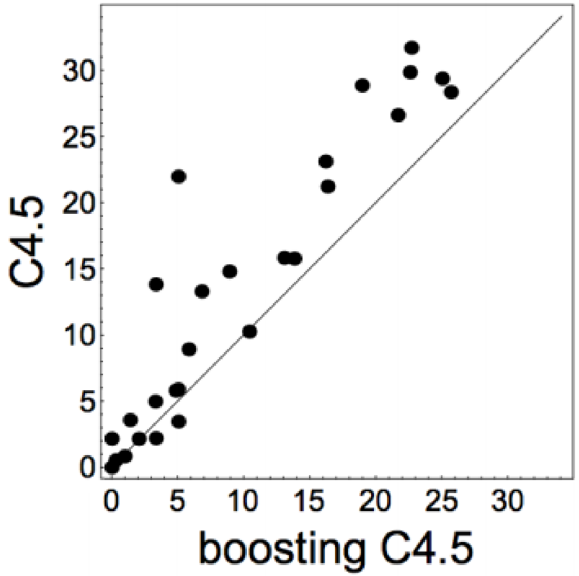
When boosting can fail:

- 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

Error Rates on 27 Benchmark Data Sets



"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		Reweights data points (modifies their distribution)
Weight of each classifier is the same		Weight is dependent on classifier's accuracy
Only variance reduction		Both bias and variance reduced – learning rule becomes more complex with iterations

Review

- 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
  - Andrew Moore
- Thanks!