DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

October 11 2018

Review

- Decision trees are interpretable, non-linear models
  – Greedy algorithm to train Decision Trees (ID3)
  – Works on categorical and numerical data
  – Node splitting done by feature with max Information Gain
- Ensemble learning
  – Combines multiple ML models for better accuracy
  – Reduces variance of individual model

Outline

- Ensemble learning
  - How to combine classifiers
  - Variance reduction
  - Methods to create diversity
- Bagging method
  - Random Forest model
  - Variable importance
- Boosting method
  - AdaBoost

Ensemble Learning

Consider a set of classifiers $h_1, \ldots, \mathrm{h}_\mathrm{L}$

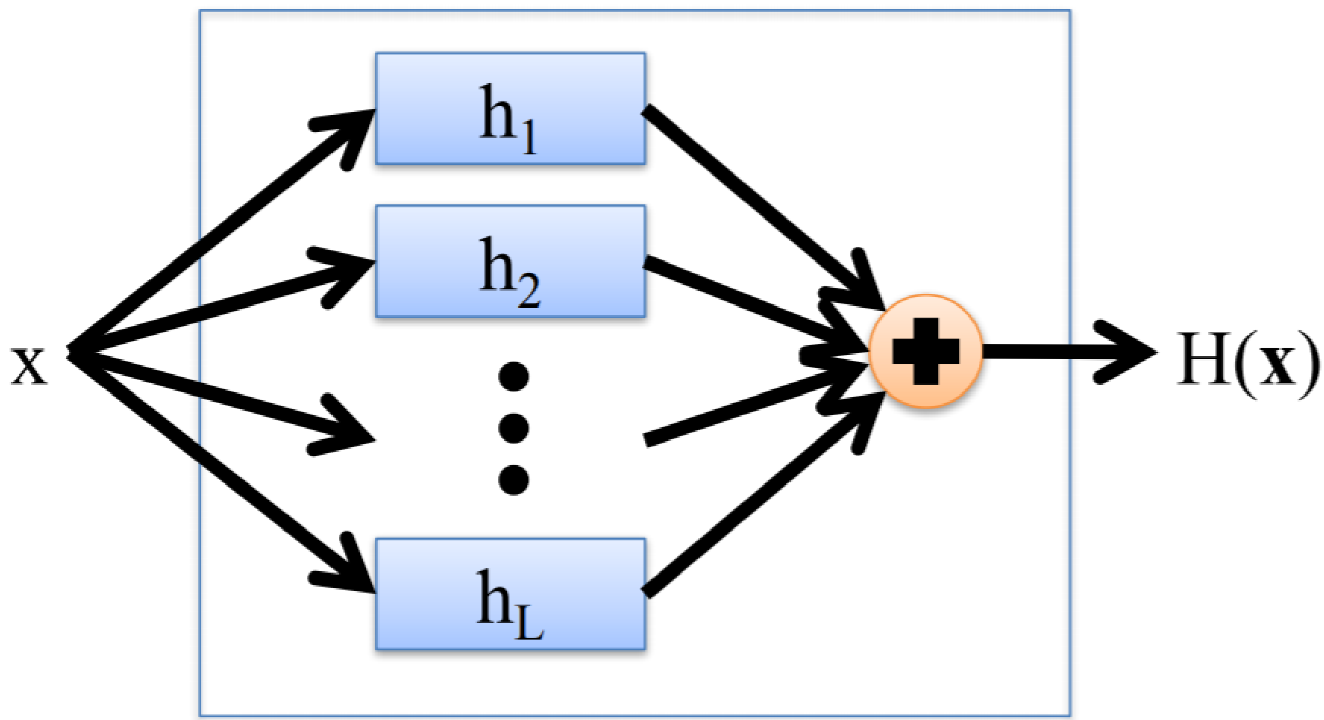
Idea: construct a classifier $H(\mathbf{x})$ that combines the individual decisions of $h_1, \ldots, \mathrm{h}_\mathrm{L}$

- 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**

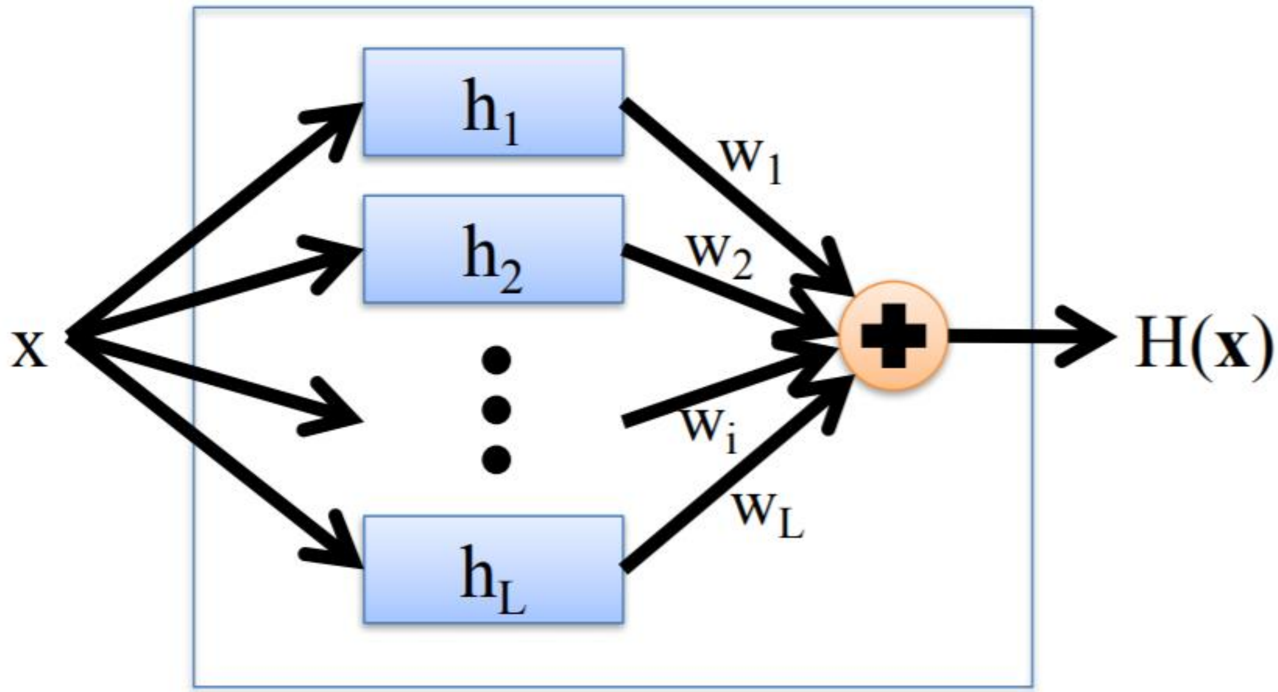
- Classifiers should make different mistakes
- Can have different types of base learners

Combining Classifiers: Averaging



- Final hypothesis is a simple vote of the members

Combining Classifiers: Weighted Averaging



- Coefficients of individual members are trained using a validation set

Reduce Variance

- Averaging reduces variance:

$$Var(\bar{X}) = \frac{Var(X)}{N}$$

(when predictions are **independent**)

Average models to reduce model variance

One problem:

only one training set

where do multiple models come from?

How to Achieve Diversity

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

Two main ensemble learning methods

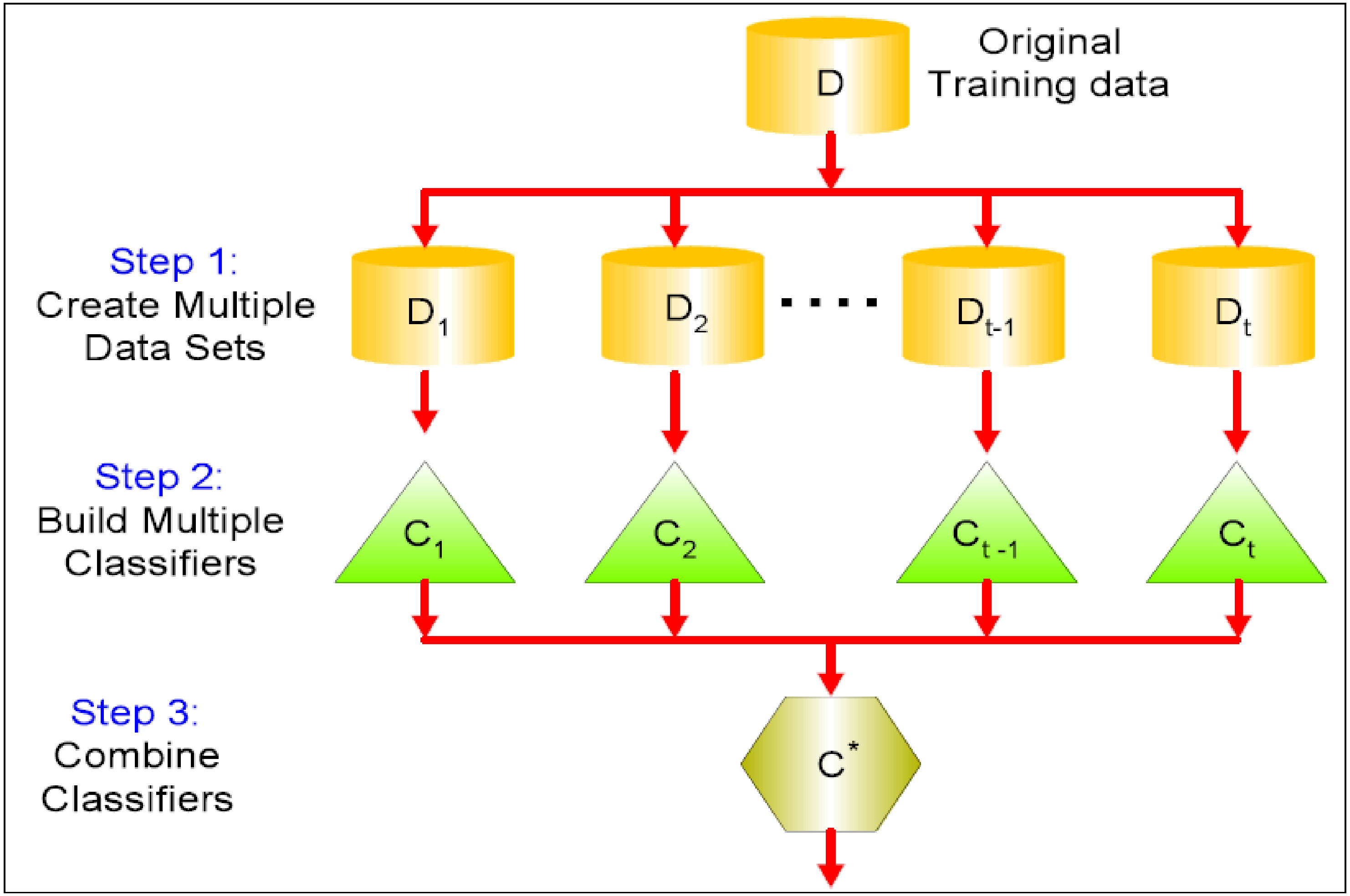
- Bagging (e.g., Random Forests)
- Boosting (e.g., AdaBoost)

Bagging

- 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 ... D_k.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

General Idea



Majority Votes

Example of Bagging

- Sampling with replacement

Training Data

Data ID

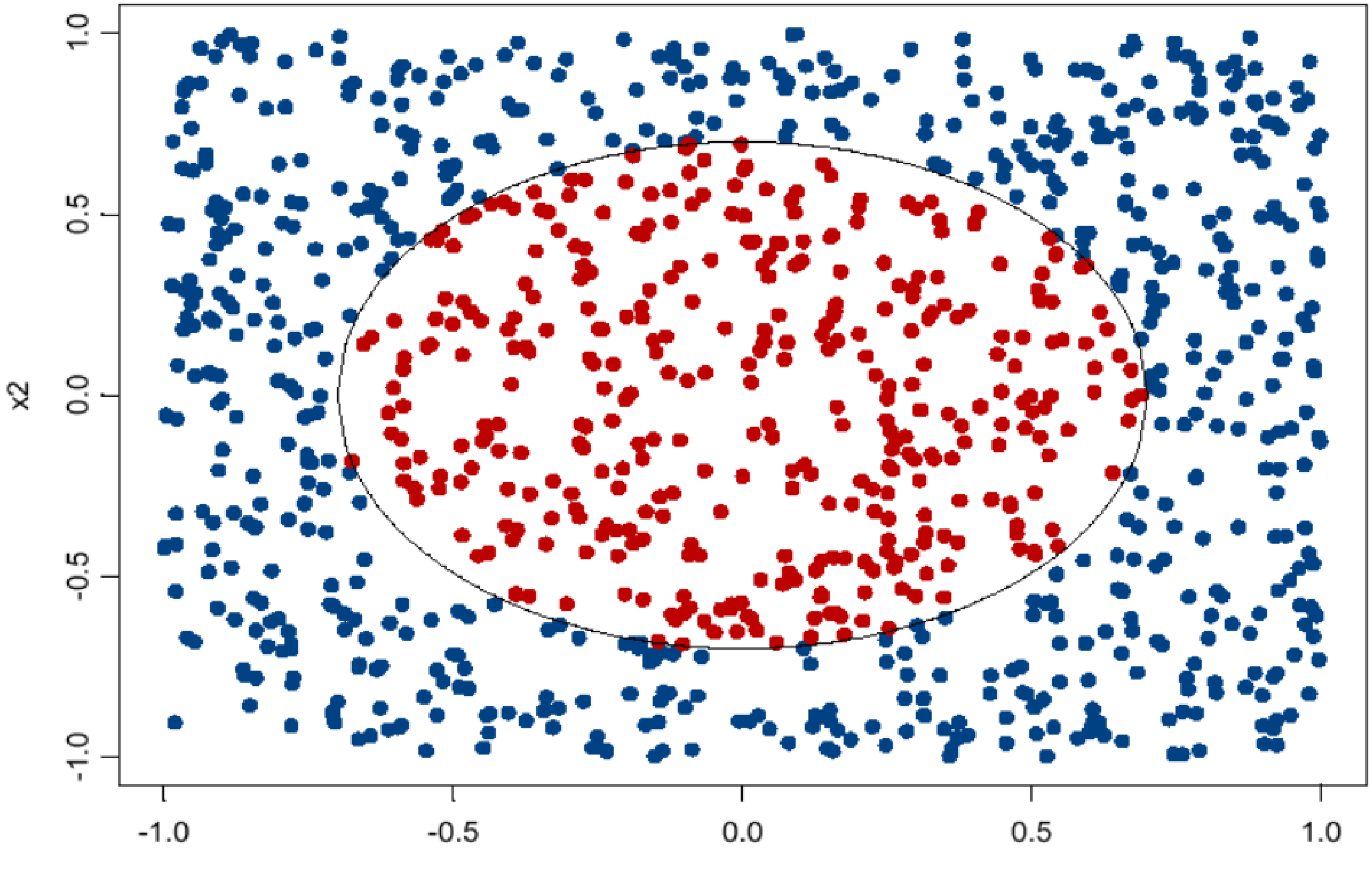
Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Sample each training point with probability 1/n
- Out-Of-Bag (OOB) observation: point not in sample
  - For each point: prob $(1-1/n)^n$
  - About 1/3 of data
  - OOB error: error on OOB samples
- OOB average error
  - Compute across all models in Ensemble
  - Use instead of Cross-Validation error

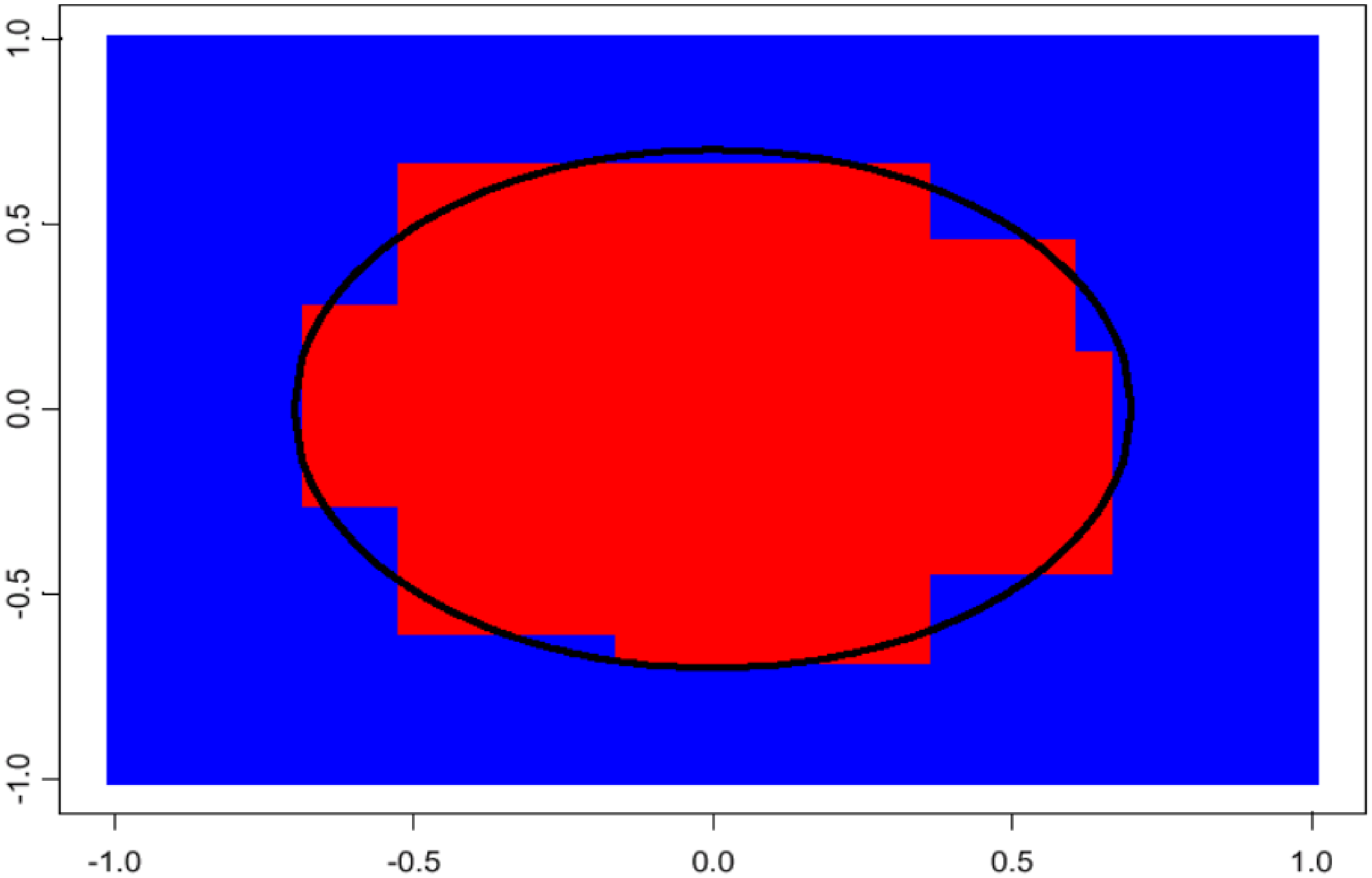
Bagging

- Can be applied to multiple classification models
- Very successful for decision trees
  - Decision trees have high variance
  - Don’t prune the individual trees, but grow trees to full extent
  - Precision accuracy of decision trees improved substantially
- OOB average error used instead of Cross Validation

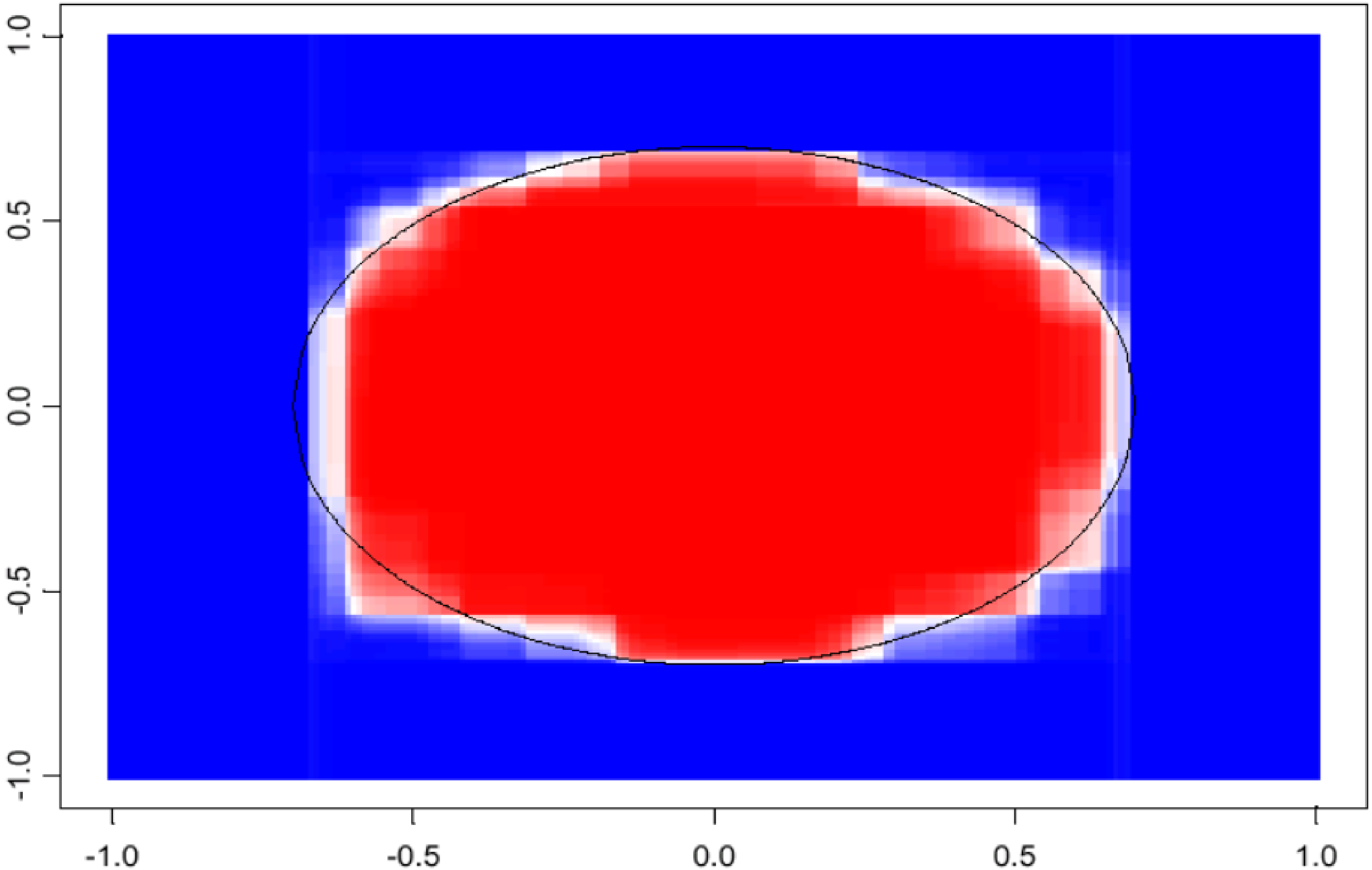
Example Distribution



Decision Tree Decision Boundary



100 Bagged Trees



shades of blue/red indicate strength of vote for particular classification

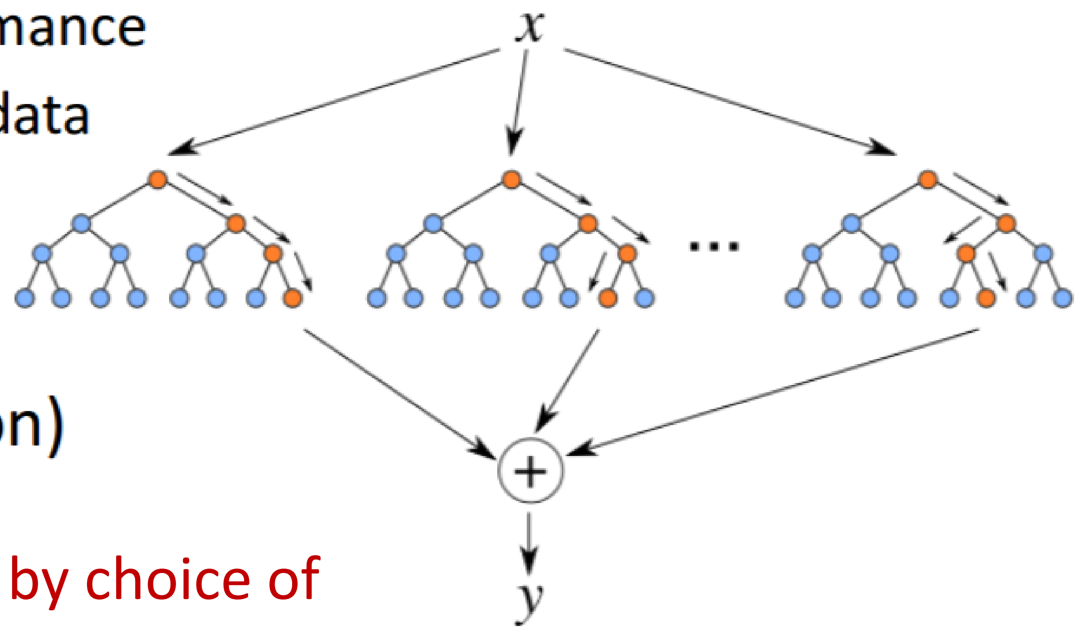
Random Forests

- Ensemble method specifically designed for decision tree classifiers

- Introduce two sources of randomness: "Bagging" and "Random input vectors"
  - Bagging method: each tree is grown using a bootstrap sample of training data
  - Random vector method: At each node, best split is chosen from a random sample of *m* attributes instead of all attributes

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)



Trees are de-correlated by choice of random subset of features

Random Forest Algorithm

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample **S** 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.

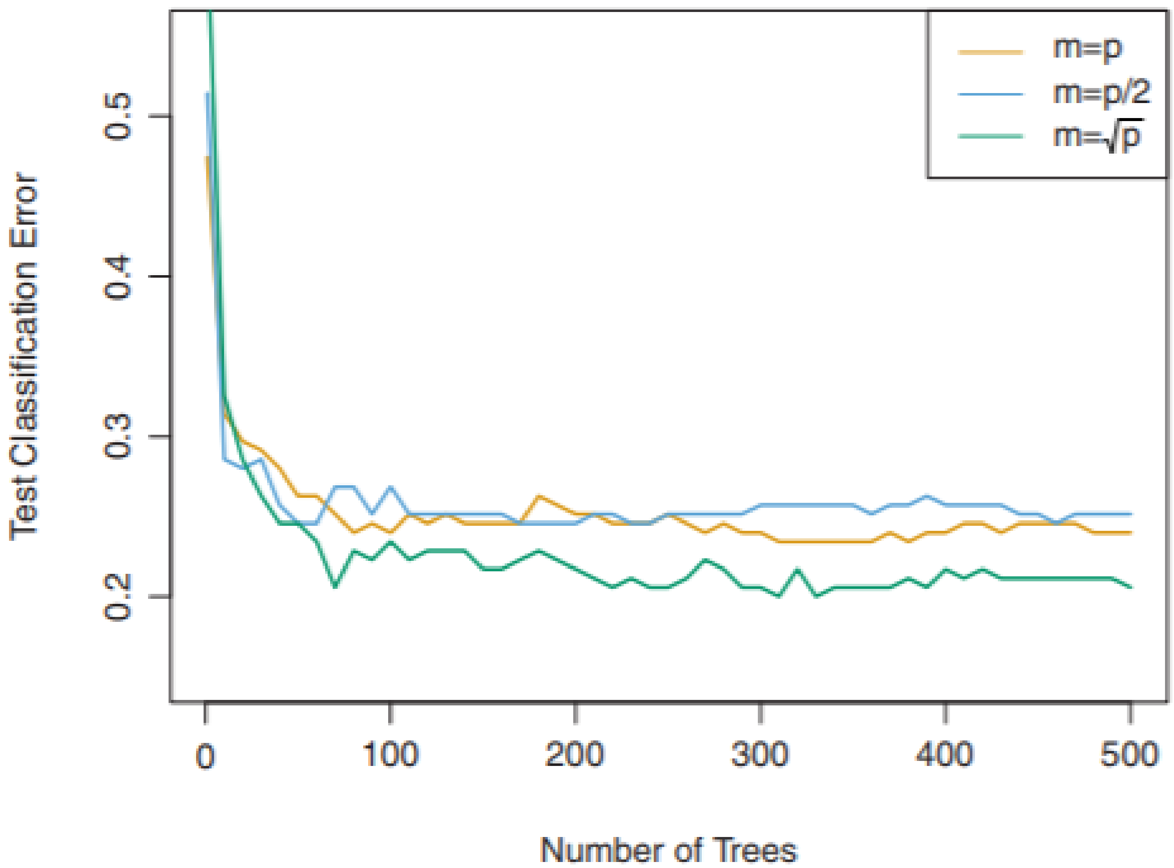
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

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

Effect of Number of Predictors



- p = total number of predictors; m = predictors chosen in each split
- Random Forests uses $m = \sqrt{p}$

Variable Importance

- Ensemble of trees looses somewhat interpretability of decision trees
- Which variables contribute mostly to prediction?
- Random Forest computes a Variable Importance metric

Gini index

- Take a node of decision tree
- Let p_i be the fraction of examples from class i
- Measures the “purity” of the node
  - If node has most examples from one class, Gini index is low
- What is the probability that a random example is mis-classified at that node?
  - $\sum_{i=1}^{k} p_i(1 - p_i)$
- Variable importance of a feature measures decrease in Gini

Variable Importance Plots

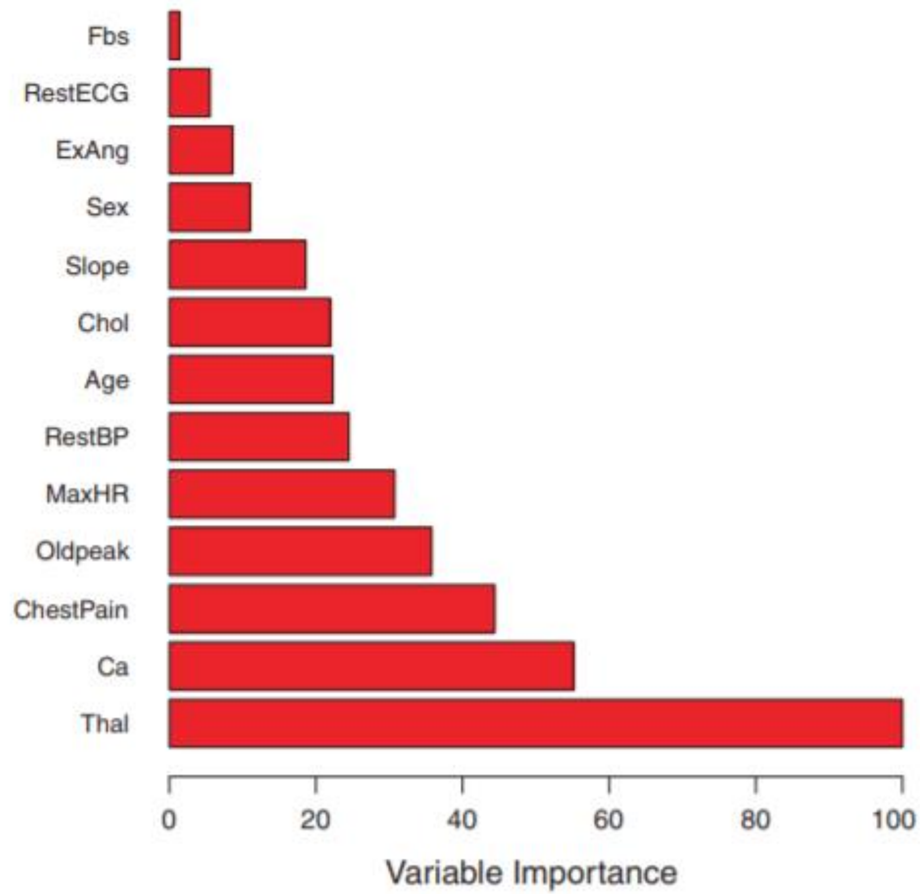


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

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
```

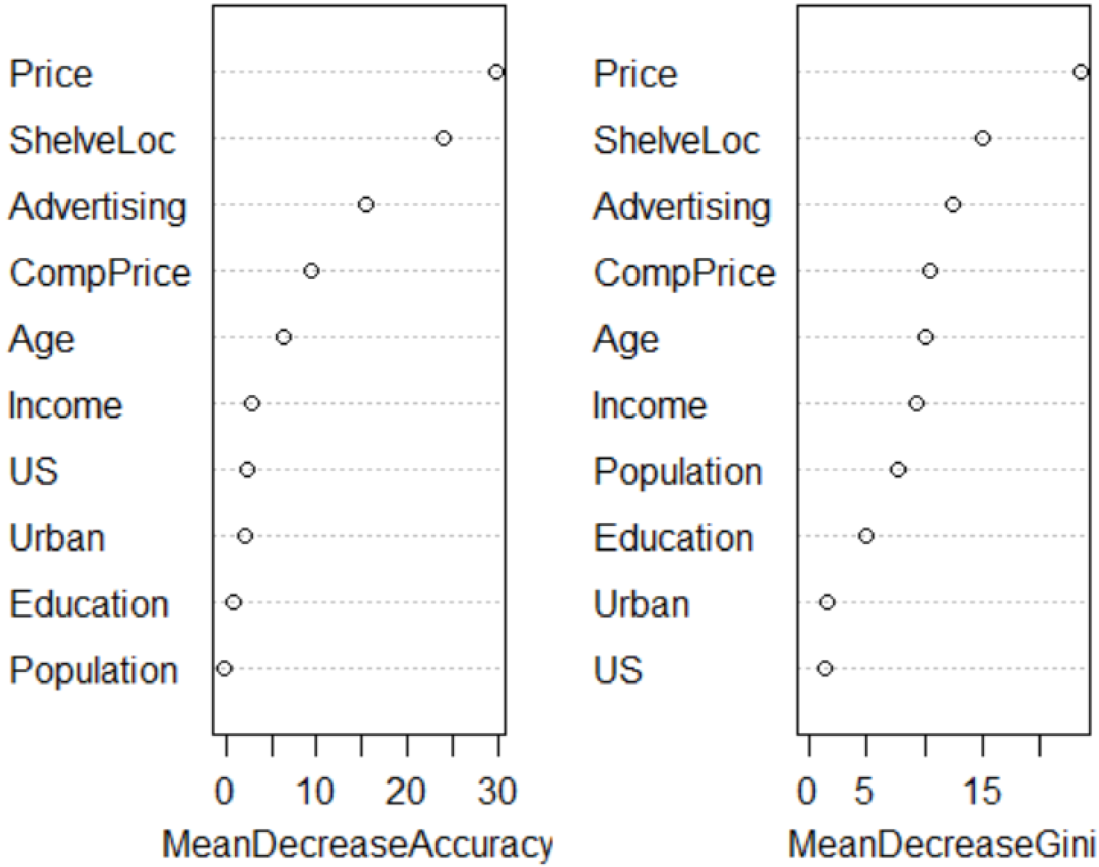
```
>
> rf.pred=predict(rf.carseats,Carseats.test,type="class")
> table(rf.pred,High.test)
       High.test
rf.pred  No Yes
    No  105  25
    Yes  13  57
> mean(rf.pred==High.test)
[1] 0.81
```

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)

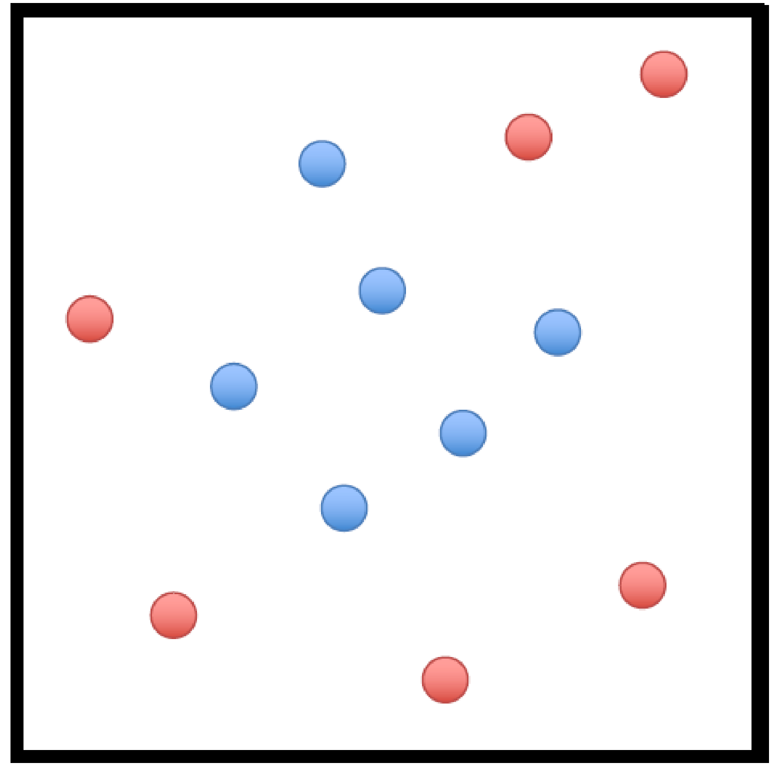
AdaBoost

- 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

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:
  $$w_{t+1,i} = w_{t,i} \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$
  Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



- Size of point represents the instance's weight

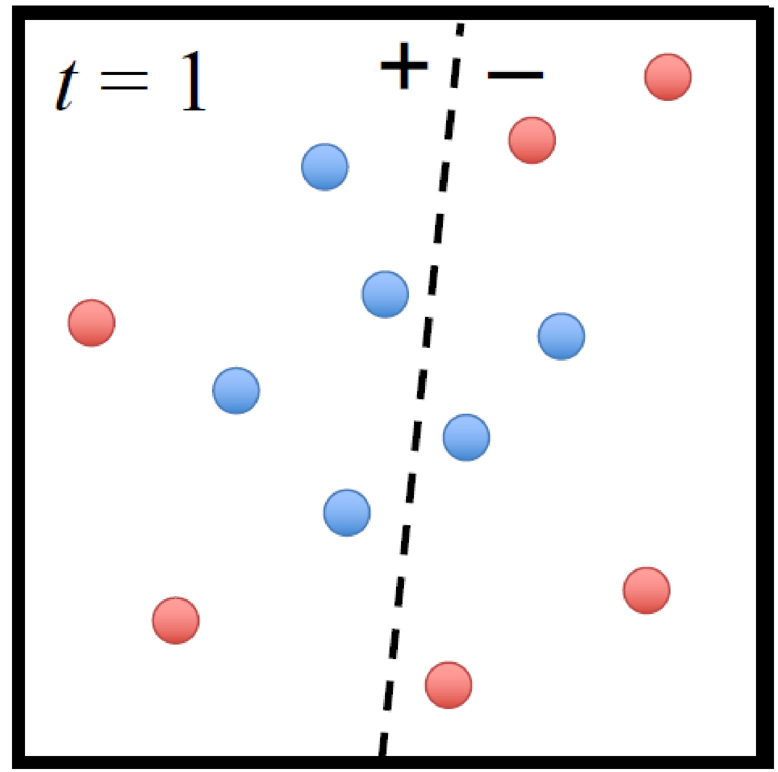
AdaBoost

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:

$$w_{t+1,i} = w_{t,i} \ \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

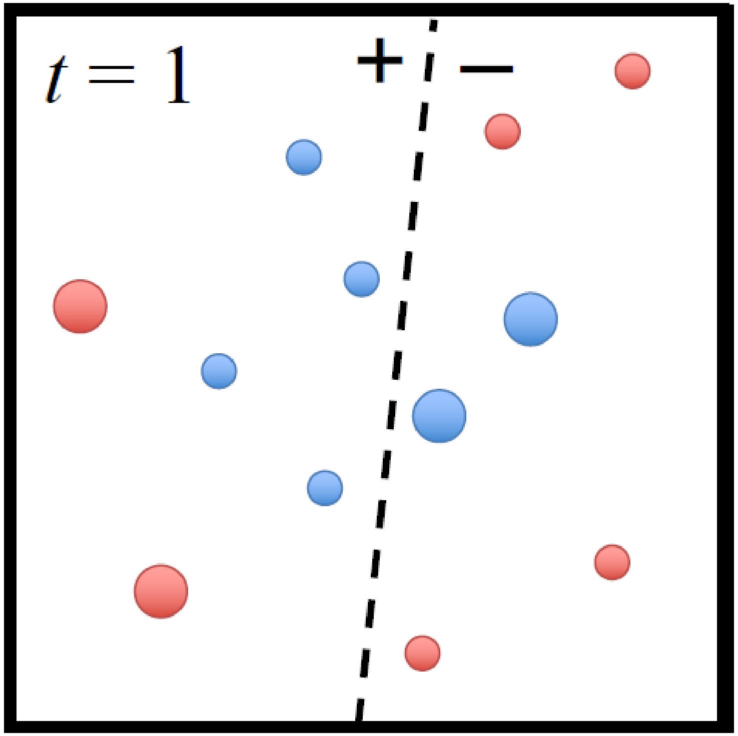
    Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis

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



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:
$$w_{t+1,i} = w_{t,i} \; \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$
    Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis
$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



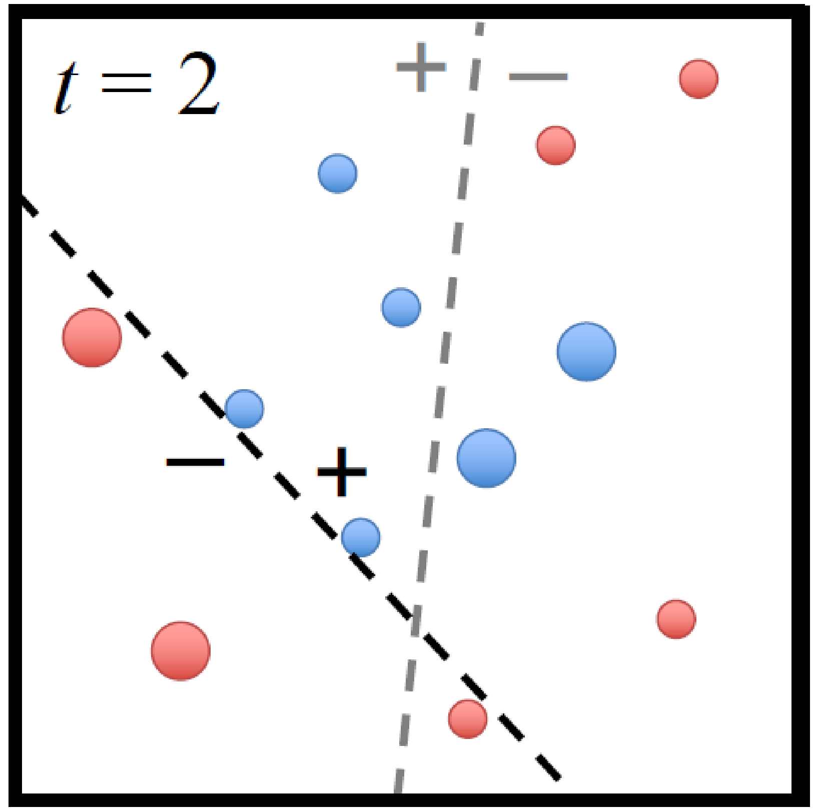
AdaBoost

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:

$$w_{t+1,i} = w_{t,i} \ \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

  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

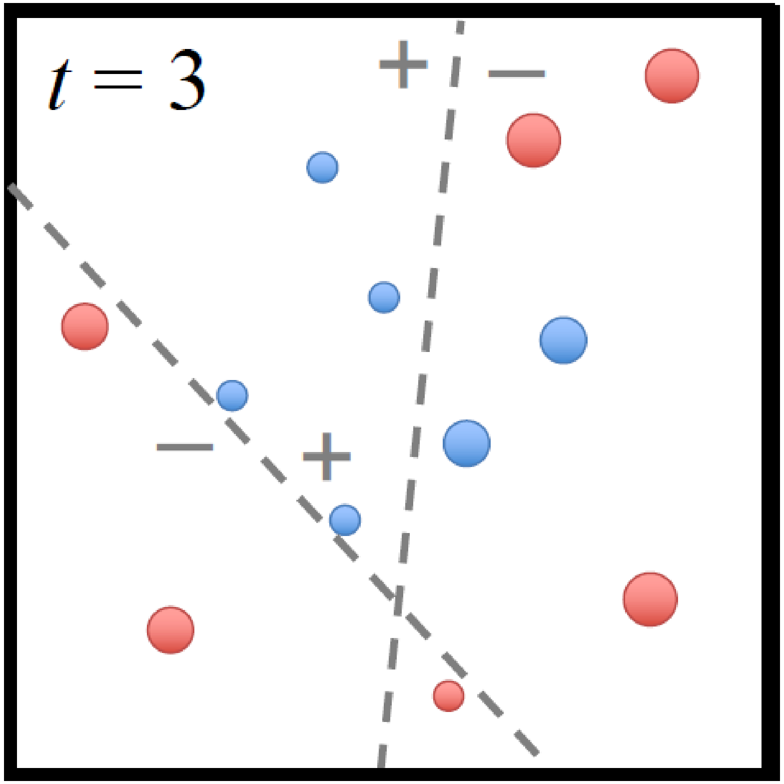
AdaBoost

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:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

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

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



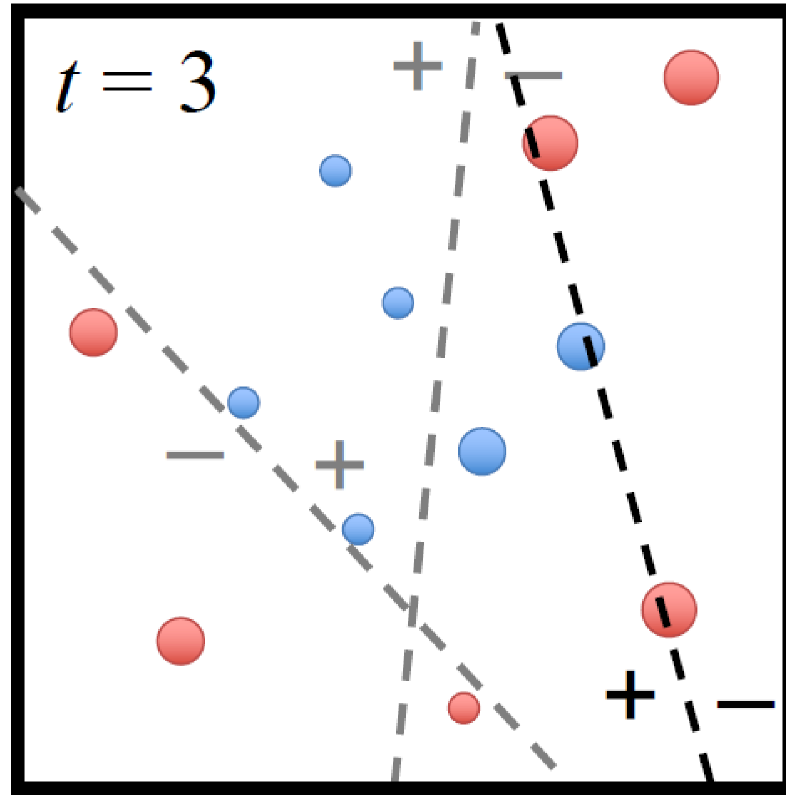
AdaBoost

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:

$$w_{t+1,i} = w_{t,i} \cdot \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

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

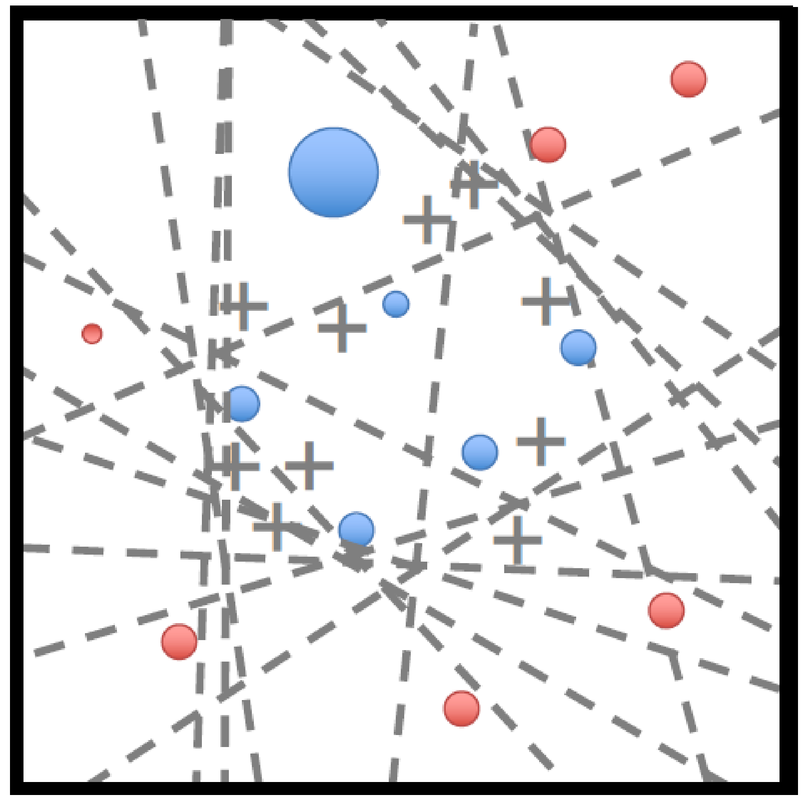
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- 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:

$$w_{t+1,i} = w_{t,i} \cdot \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

    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 $X, y = \{(x^{(i)}, y^{(i)})\}, \mathrm{i} = 1 \ldots \mathrm{n}$

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:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y^{(i)} h_t(x^{(i)})), i = 1, \ldots, n$$

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

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

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

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

- 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

Base Learner Requirements

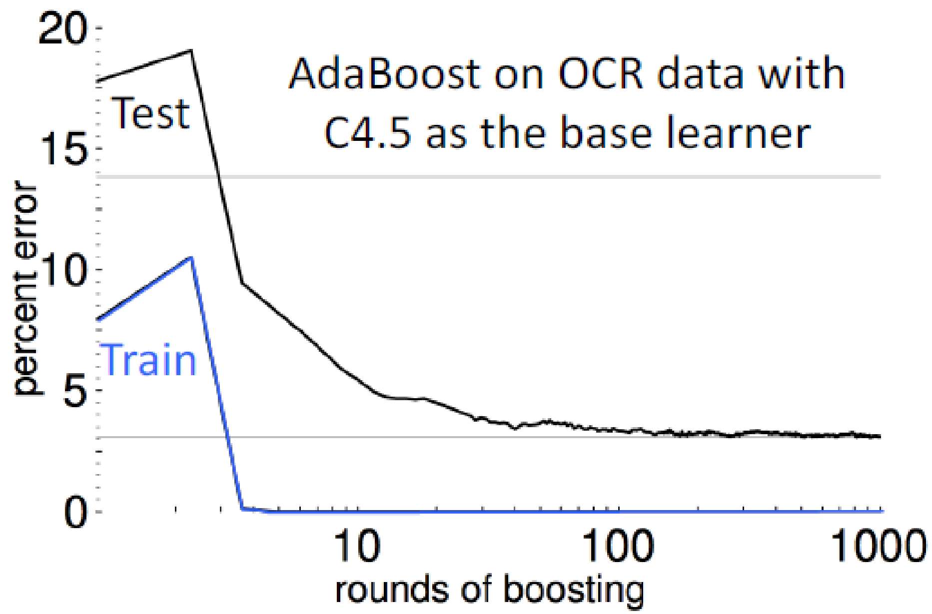
- AdaBoost works best with "weak" learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in $O(\log n)$ iterations

- Examples:
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

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 overfit
- Does not use explicitly regularization

No overfitting



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

AdaBoost in Practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

When boosting can fail:

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

Review

- Ensemble learning are powerful learning methods
- 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 weak learners that emphasizes mis-predicted examples
  - AdaBoost has great theoretical and experimental performance
  - Can be used with linear models or simple decision trees

Acknowledgements

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