DS 4400

Machine Learning and Data Mining I
Spring 2024

David Liu
Khoury College of Computer Science
Northeastern University

March 12 2024

Outline

- Ensemble models
  - Reduction in error and variance
- Bagging
  - Random forest
  - Variable importance
- Boosting
  - AdaBoost
  - Properties of boosting
  - Bagging vs Boosting

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

Reduce error

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

Reduce error

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{\mathrm{i}} (1-\varepsilon)^{25-i} = 0.06$$

Reduce Variance

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

How to Achieve Diversity

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

Two main ensemble learning methods

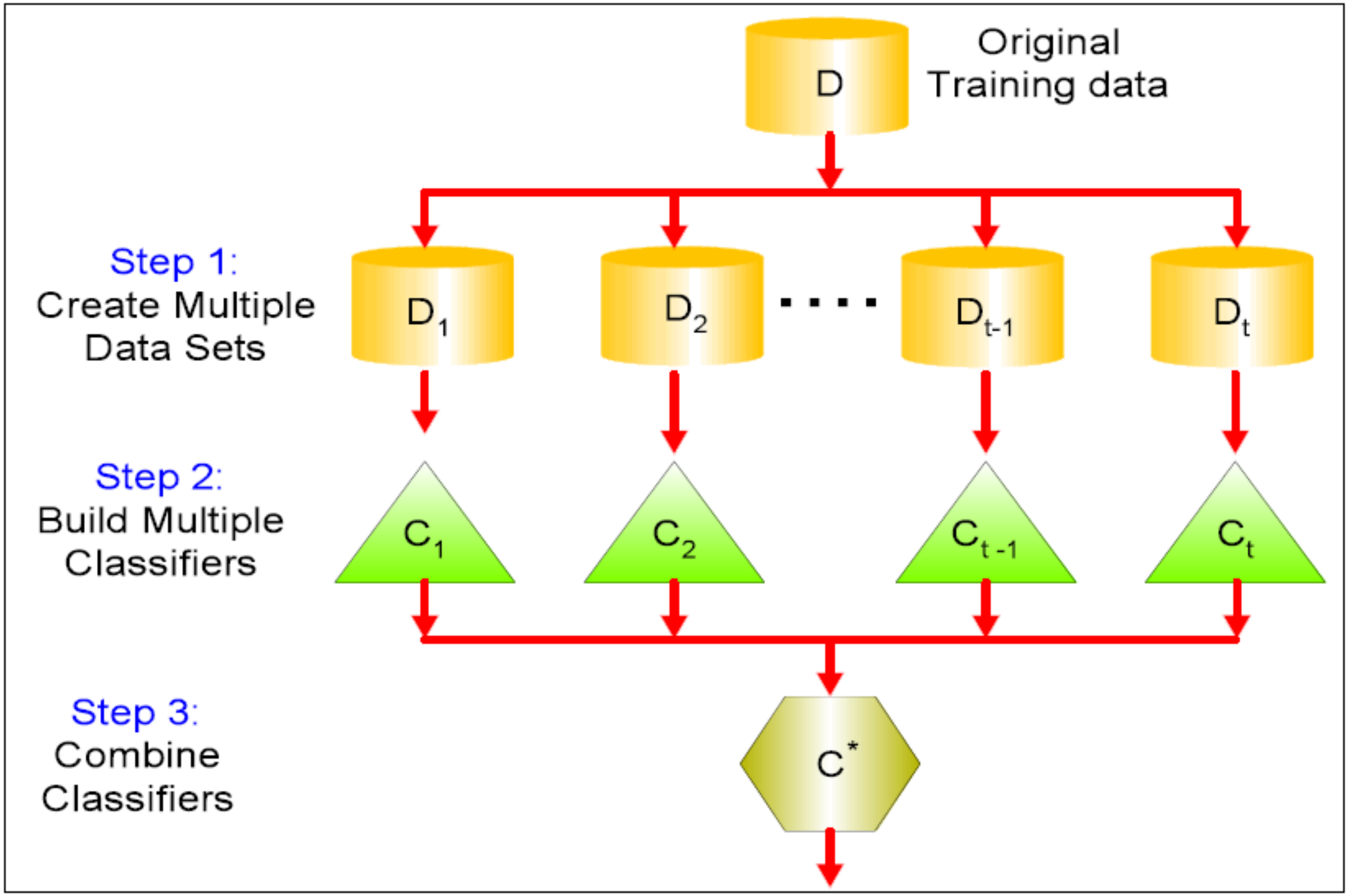
- Bagging (e.g., Random Forests)
- Boosting (e.g., AdaBoost)

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 $D_1 \ldots D_k$.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

General Idea



Majority Votes

Example of Bagging

- Sampling with replacement

Data ID   Training Data

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

Example of Bagging

- Sampling with replacement

Training Data

Data ID

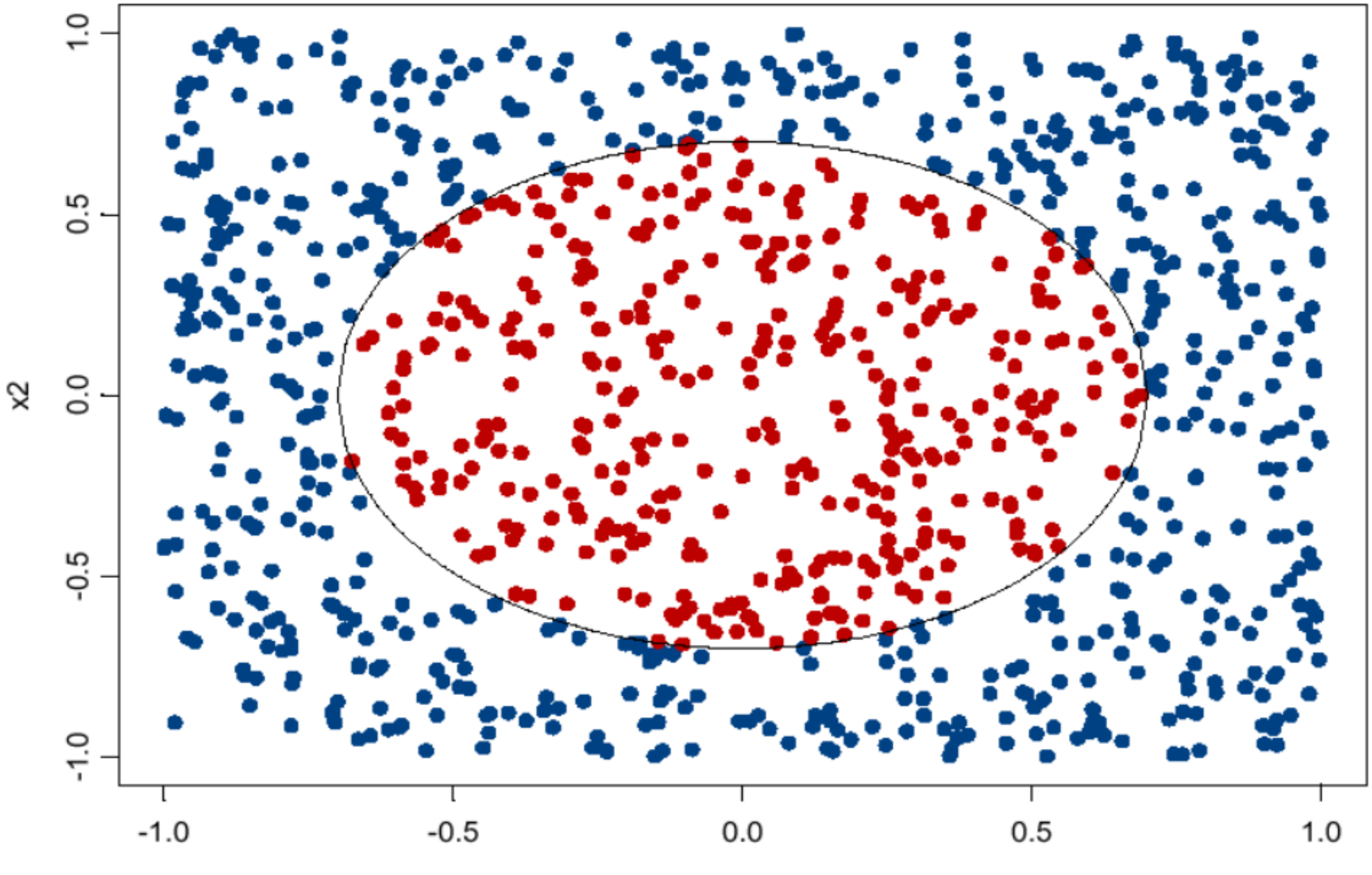
Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- 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

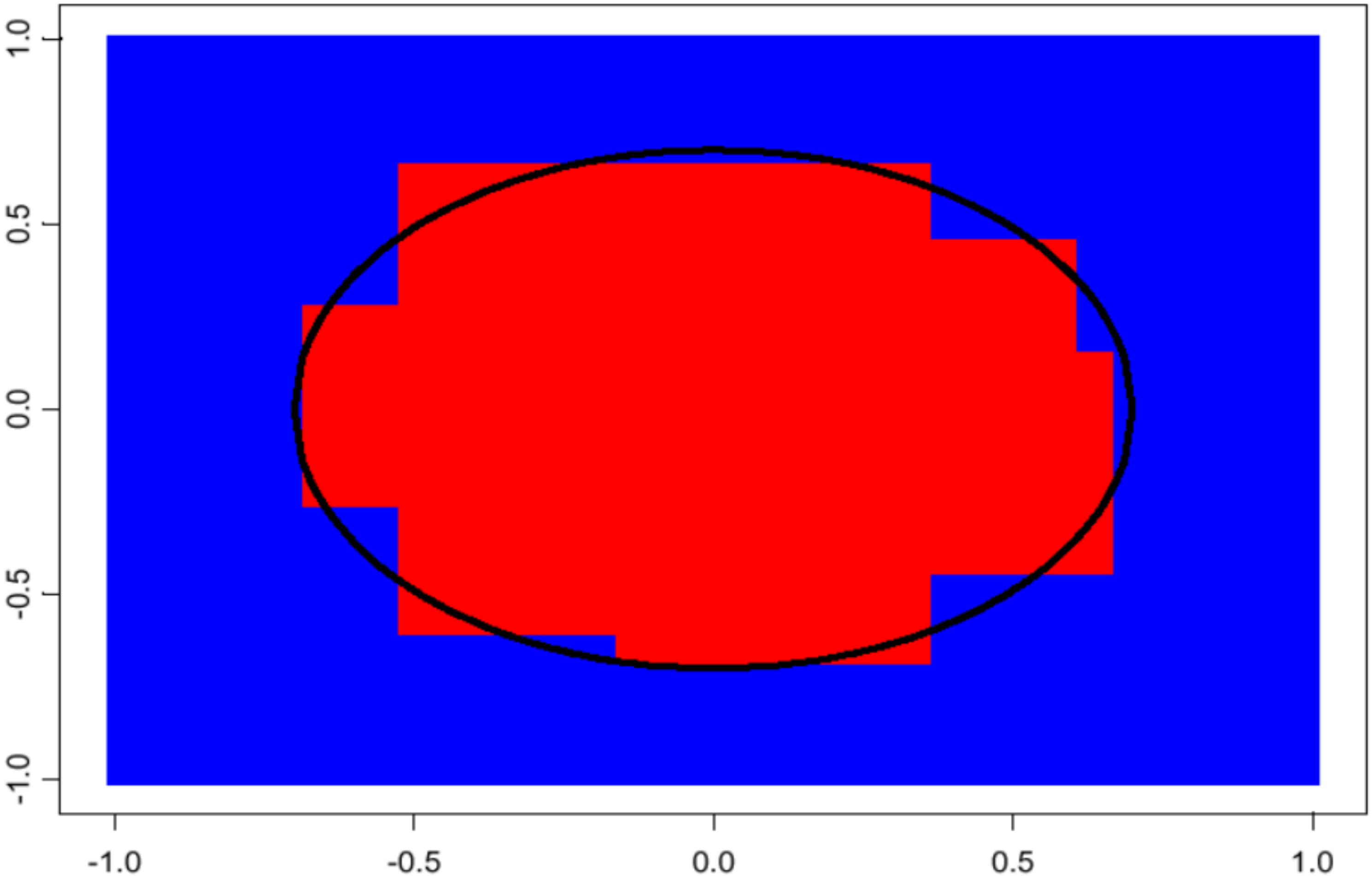
Bagging

- 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

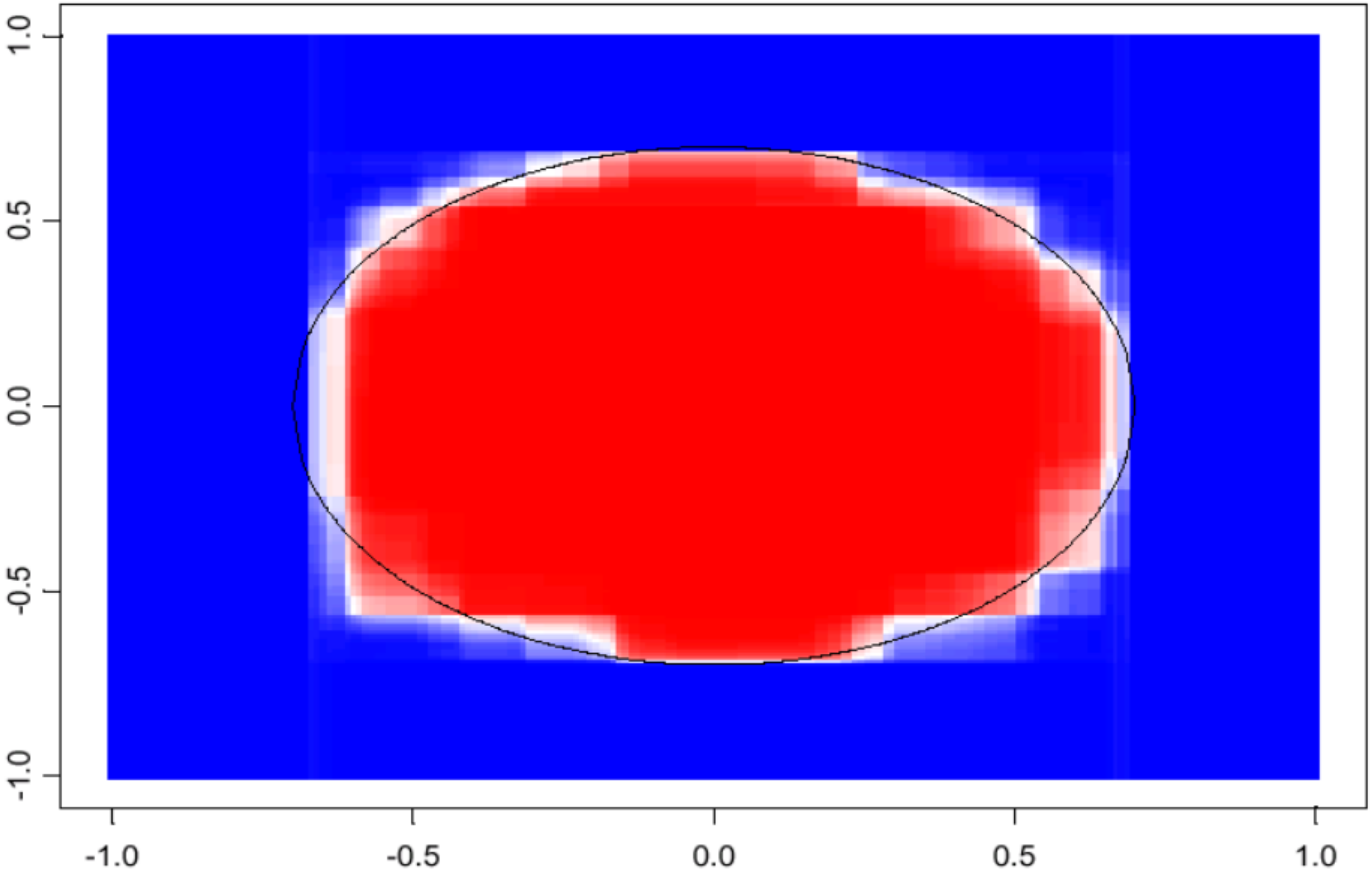
Example Distribution



Decision Tree Decision Boundary



100 Bagged Trees



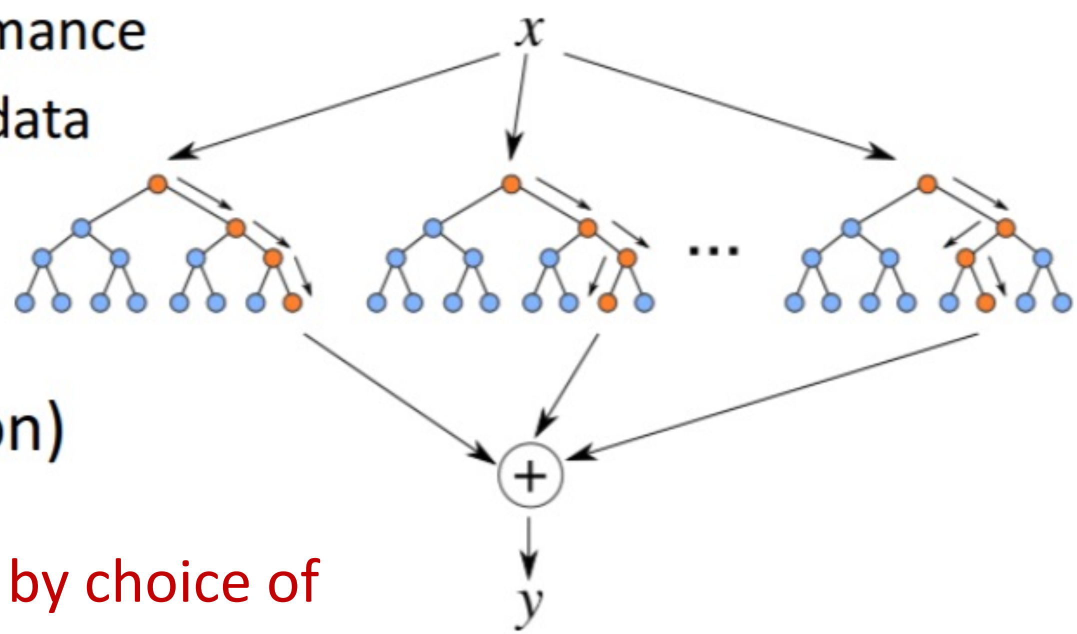
shades of blue/red indicate strength of vote for particular classification

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 $\boxed{\text{bootstrap sample}}$ $\mathbf{Z}^*$ 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 $\boxed{m \text{ variables at random}}$ 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 $\{T_b\}_1^B$.

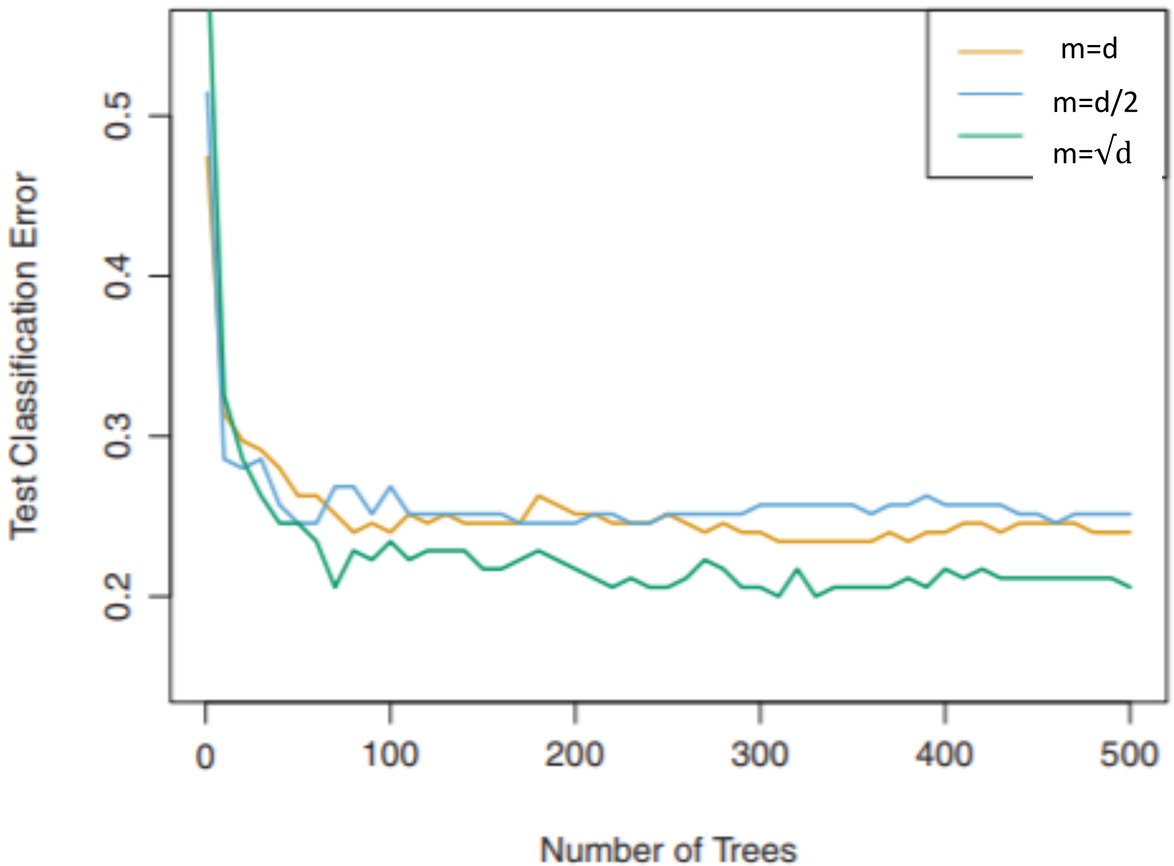
To make a prediction at a new point x:

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 with Decision Trees as base learner

Effect of Number of Predictors



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

Variable Importance

- Ensemble of trees looses somewhat interpretability of decision trees
- Which variables contribute mostly to prediction?
- Random Forests computes a Variable Importance metric per feature
  – For each tree in the ensemble, consider the split by the particular feature
  – How much information gain / Gini index decreases after the split
  – Average over all trees

Variable Importance Plots

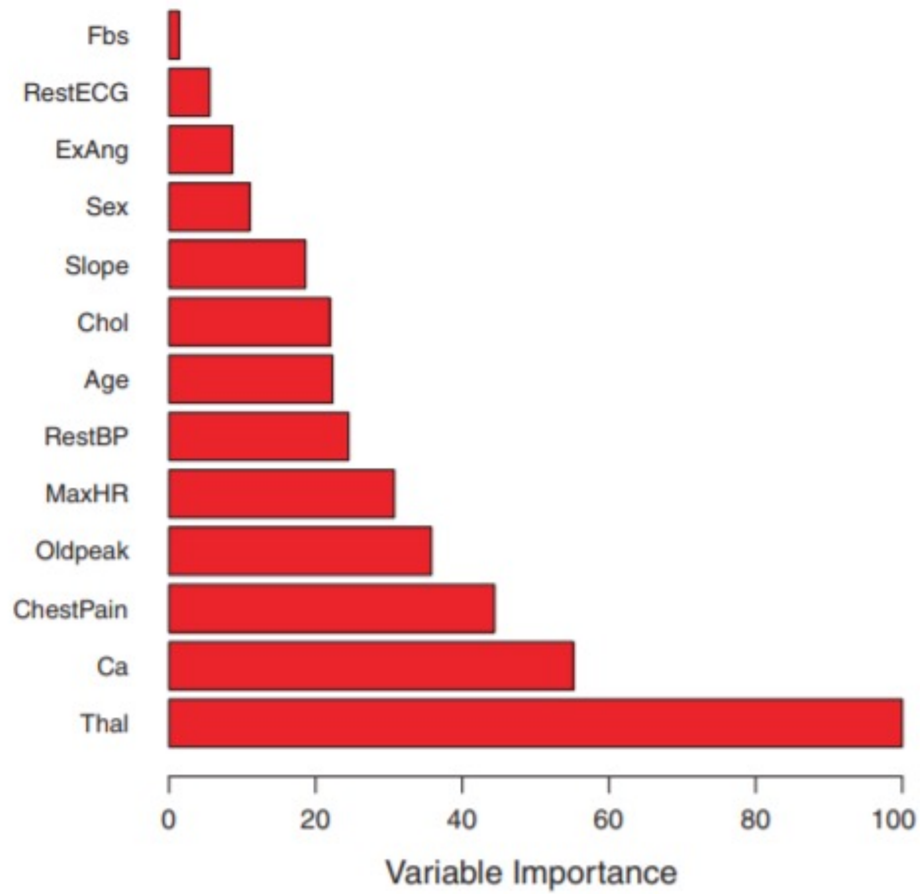


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Variable Importance

- Ensembles of trees loose in interpretability
  - Variable importance helps with determining important features
- Can be used for feature selection
  - Train Random Forest model
  - Compute variable importance
  - Select top k features by highest important
  - Train other models with the k features

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

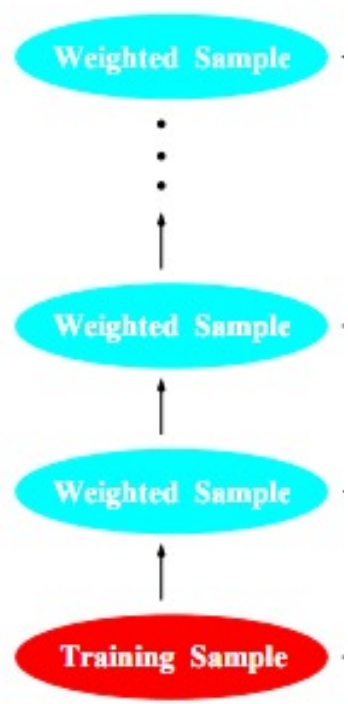
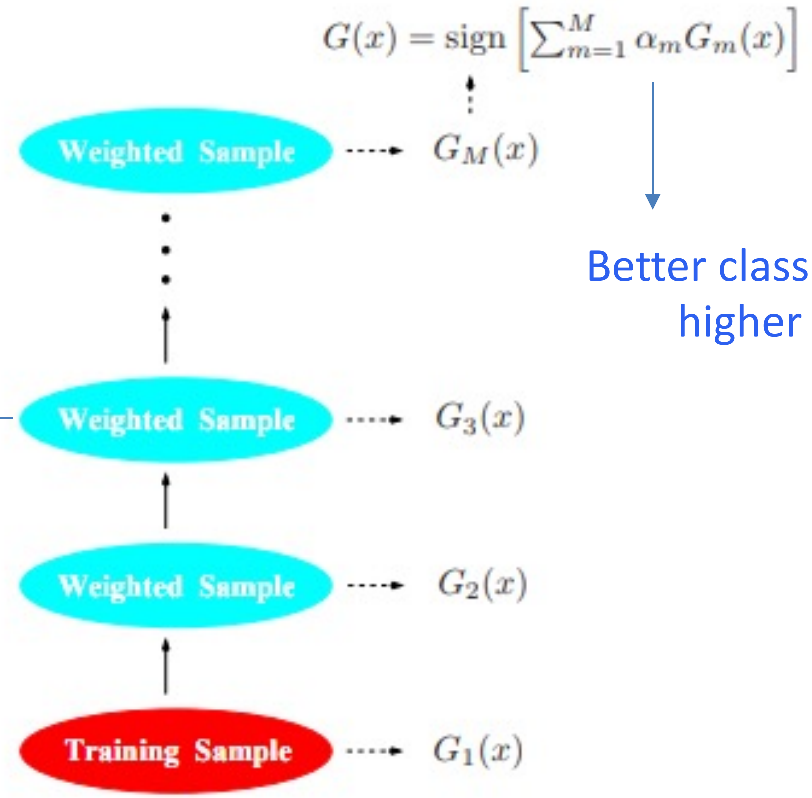


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Overview of AdaBoost



Better classifiers will get higher weights

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

- Final classifier: $\boxed{H(x) = sign(\sum \beta_t h_t(x))}$

Convergence bounds with minimal assumptions on weak learner

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

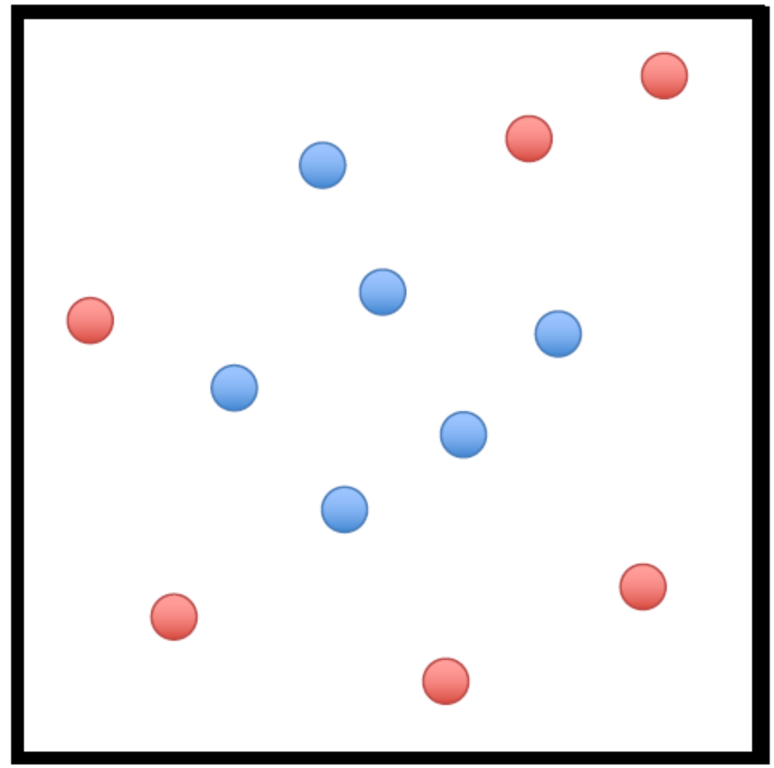
AdaBoost

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

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



- Size of point represents the instance's weight

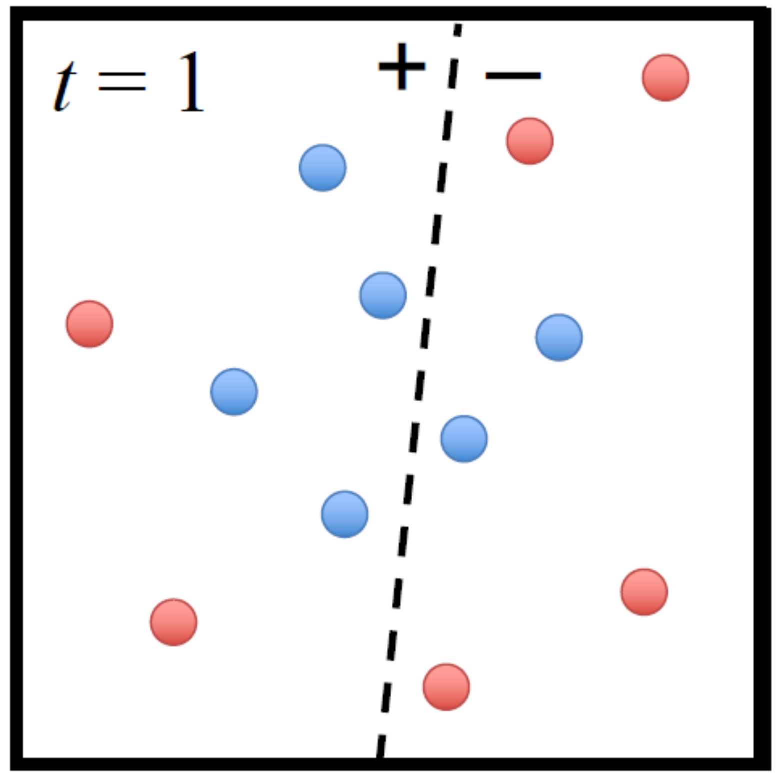
AdaBoost

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

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



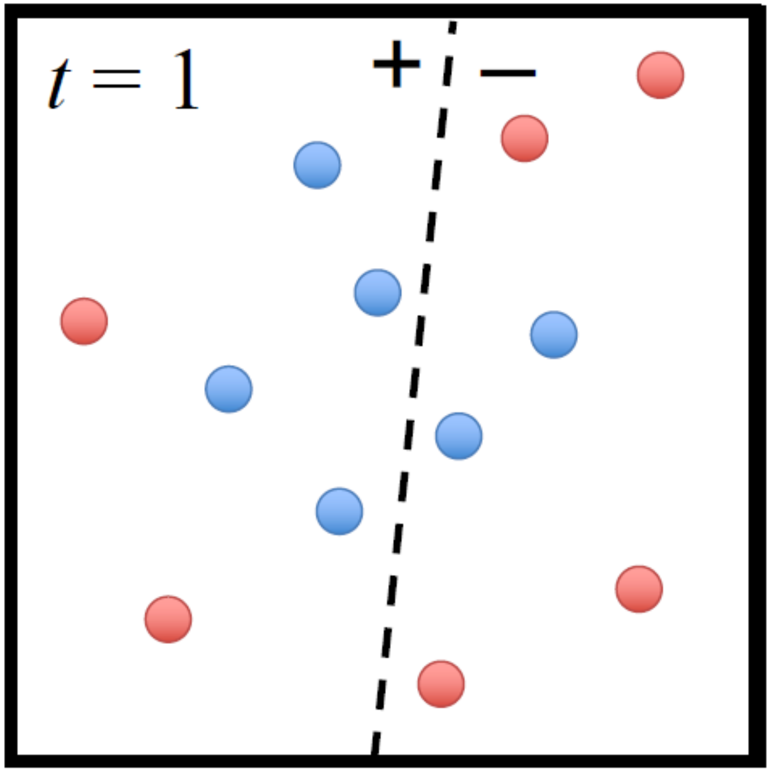
AdaBoost

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

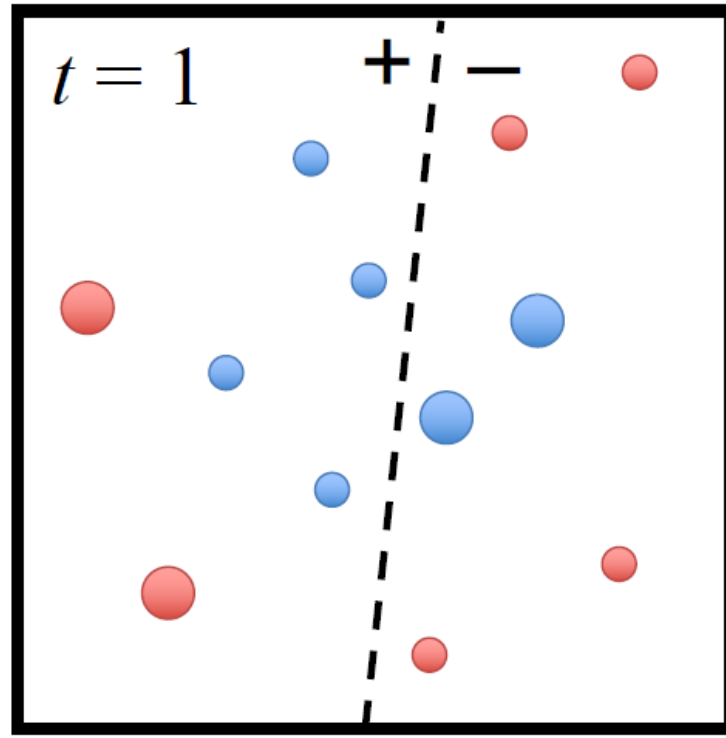
$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



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

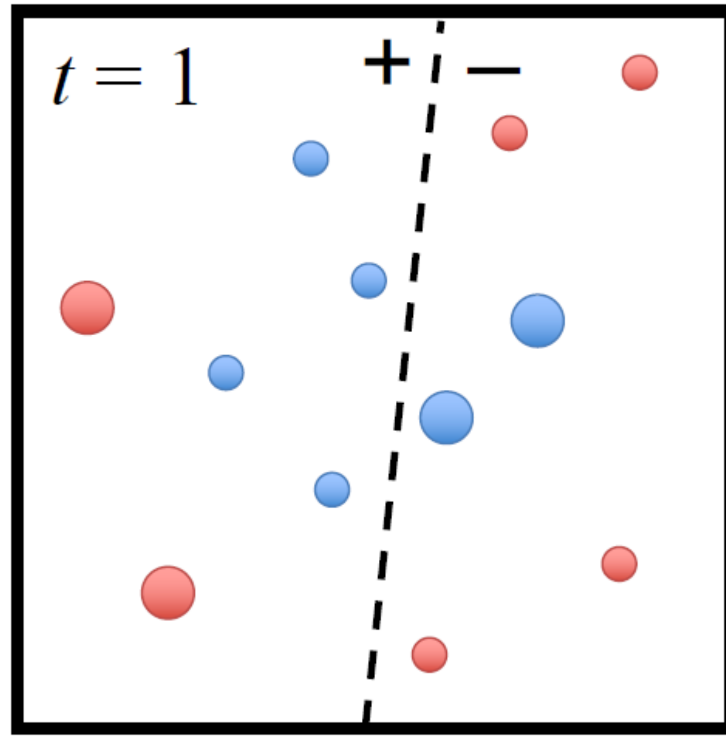
AdaBoost

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



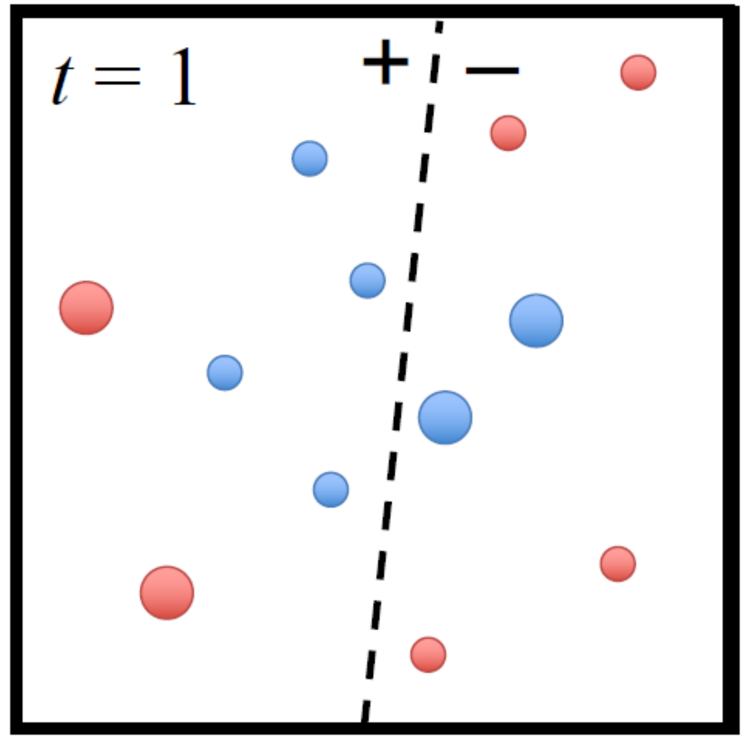
AdaBoost

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



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



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

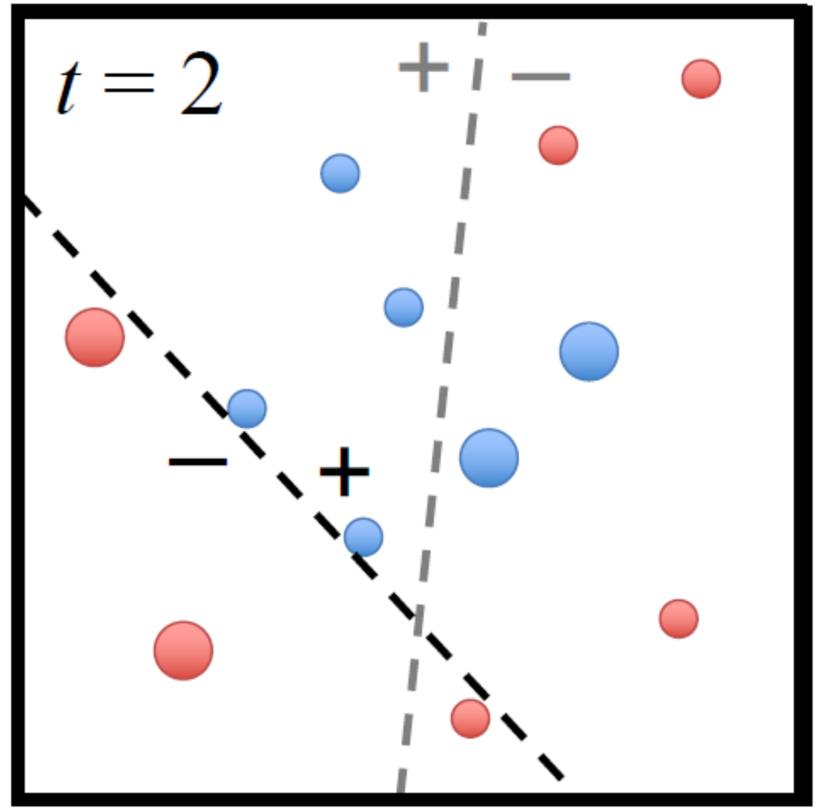
AdaBoost

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

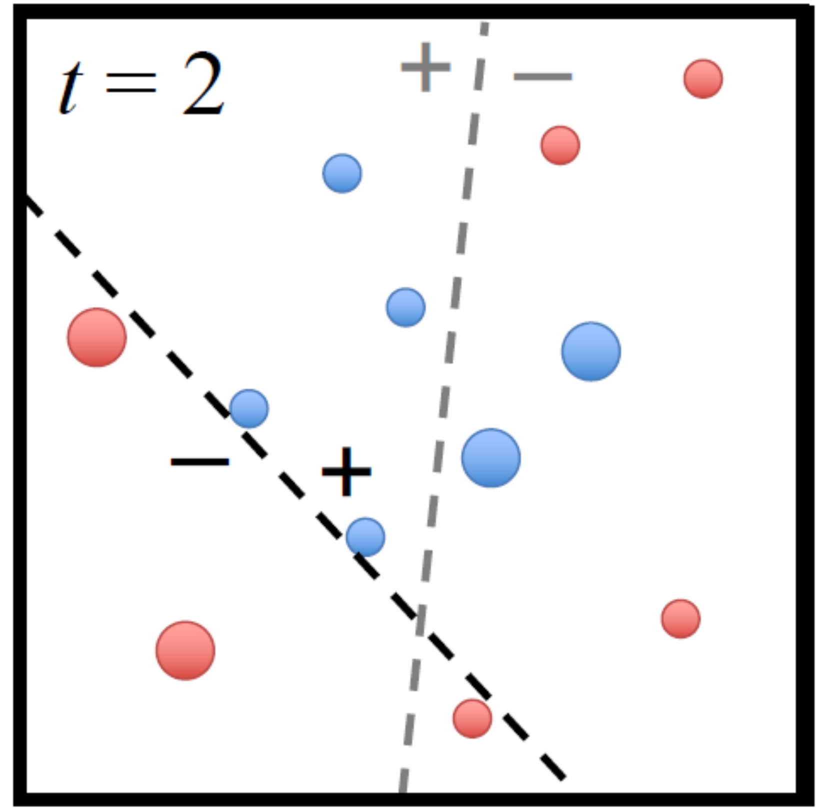
$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

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

end for

Return the hypothesis

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



- Compute importance of hypothesis β_t
- Update weights w_t

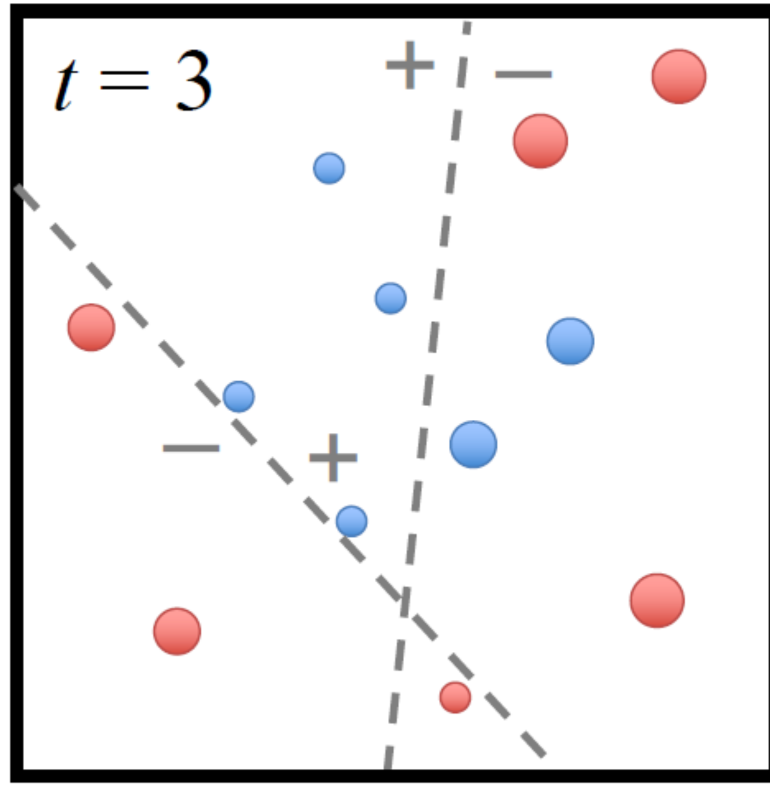
AdaBoost

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

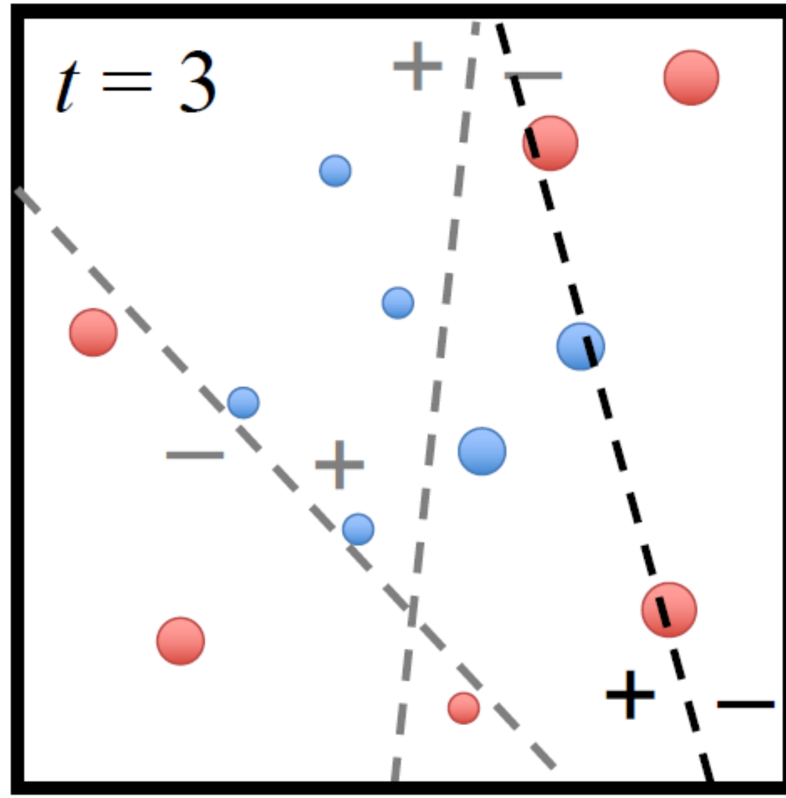
$$H(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost

```
Initialize a vector of n uniform weights w_1
for t = 1, ..., T
    Train model h_t on X, y with weights w_t
    Compute the weighted training error of h_t
    Choose β_t = 1/2 ln((1 − ε_t) / ε_t)
    Update all instance weights:
        w_{t+1,i} = w_{t,i} exp(−β_t y_i h_t(x_i))
    Normalize w_{t+1} to be a distribution
end for
Return the hypothesis
```

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



- Compute importance of hypothesis β_t
- Update weights w_t

AdaBoost

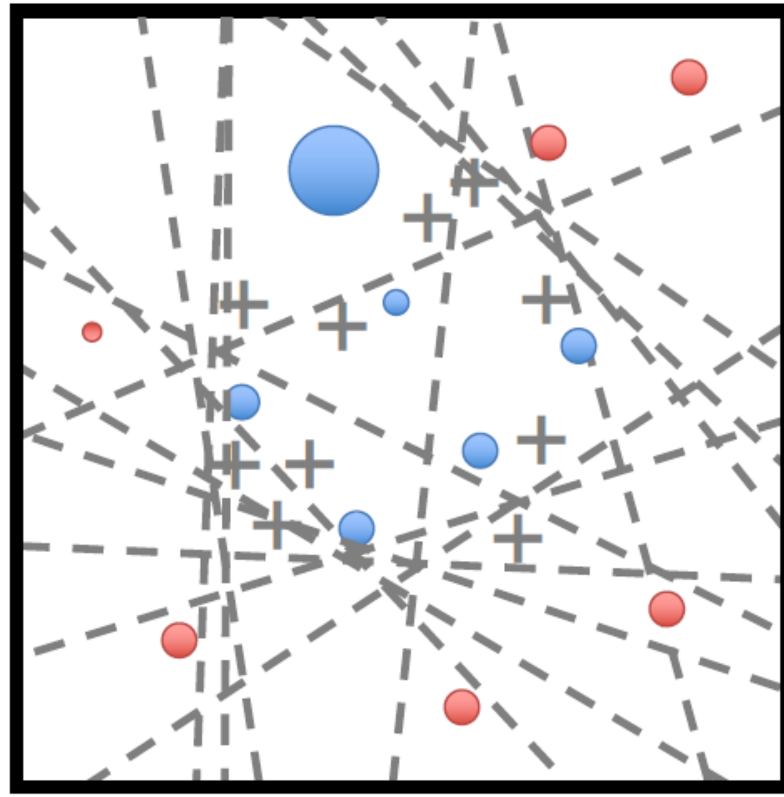
Initialize a vector of n uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model h_t on X, y with weights $\mathbf{w}_t$
    Compute the weighted training error of h_t
    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance weights:

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

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

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

$t = \mathrm{T}$



- 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 $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$
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

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

Train with Weighted Instances

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

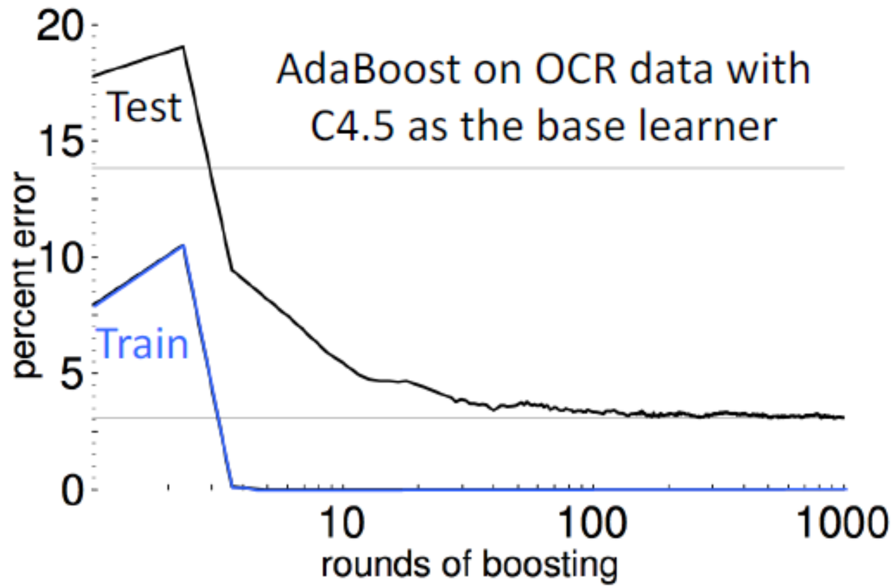
$$J_{\text{reg}}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} w_i \left[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log \left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)\right] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

- 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



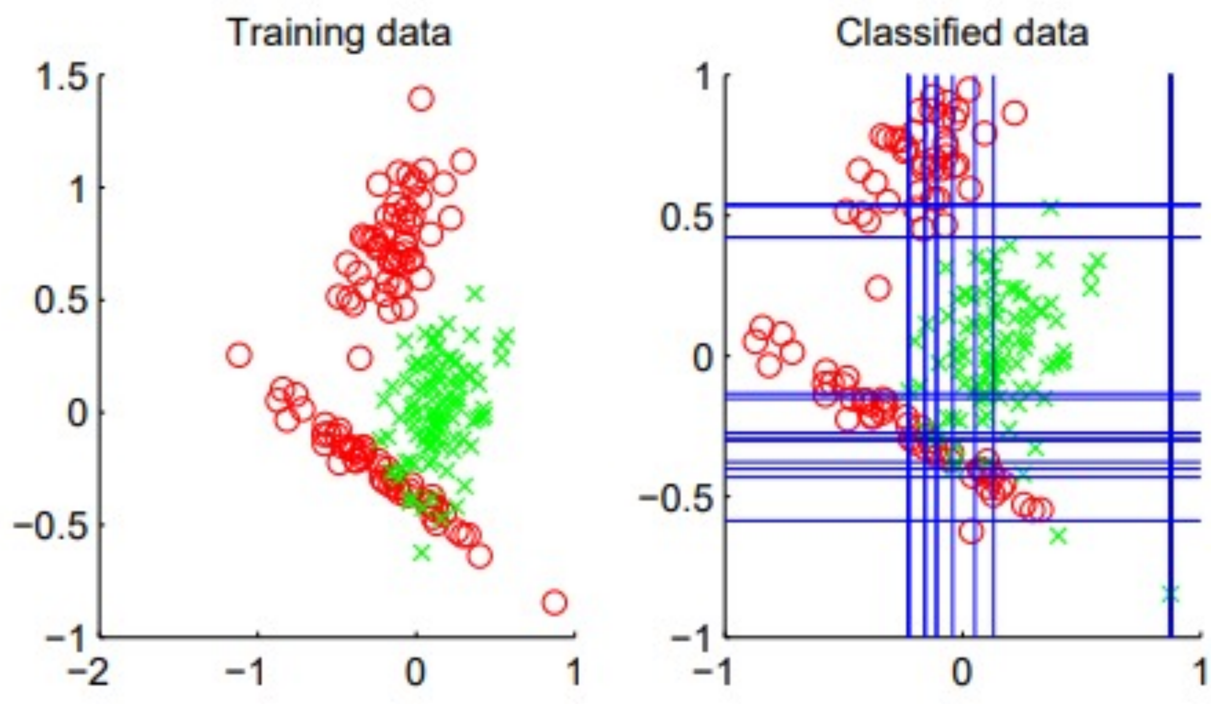
- Empirically, boosting resists overfitting
- Note that it continues to drive down the test error even AFTER the training error reaches zero

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:

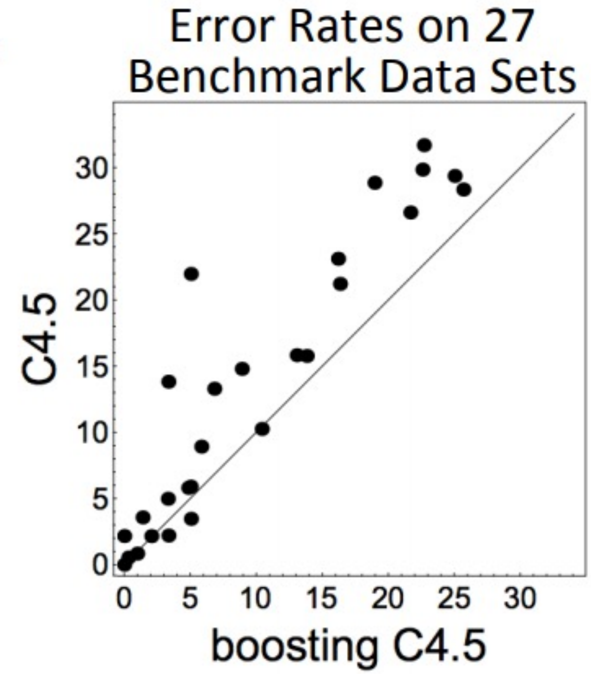
- Fast and simple to program
- No parameters to tune (besides T) Learn with Cross-Validation
- No assumptions on weak learner Error less than ½

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



"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

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

Applicable to complex models with low bias, high variance

Applicable to weak models with high bias, low variance

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)