



# Ensemble Techniques

# Ensemble Methods

---

- Improve classification accuracy by aggregating the predictions of multiple classifiers
- Construct a set of base classifiers learned from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers (e.g., by taking majority vote)

# Example: Why Do Ensemble Methods Work?

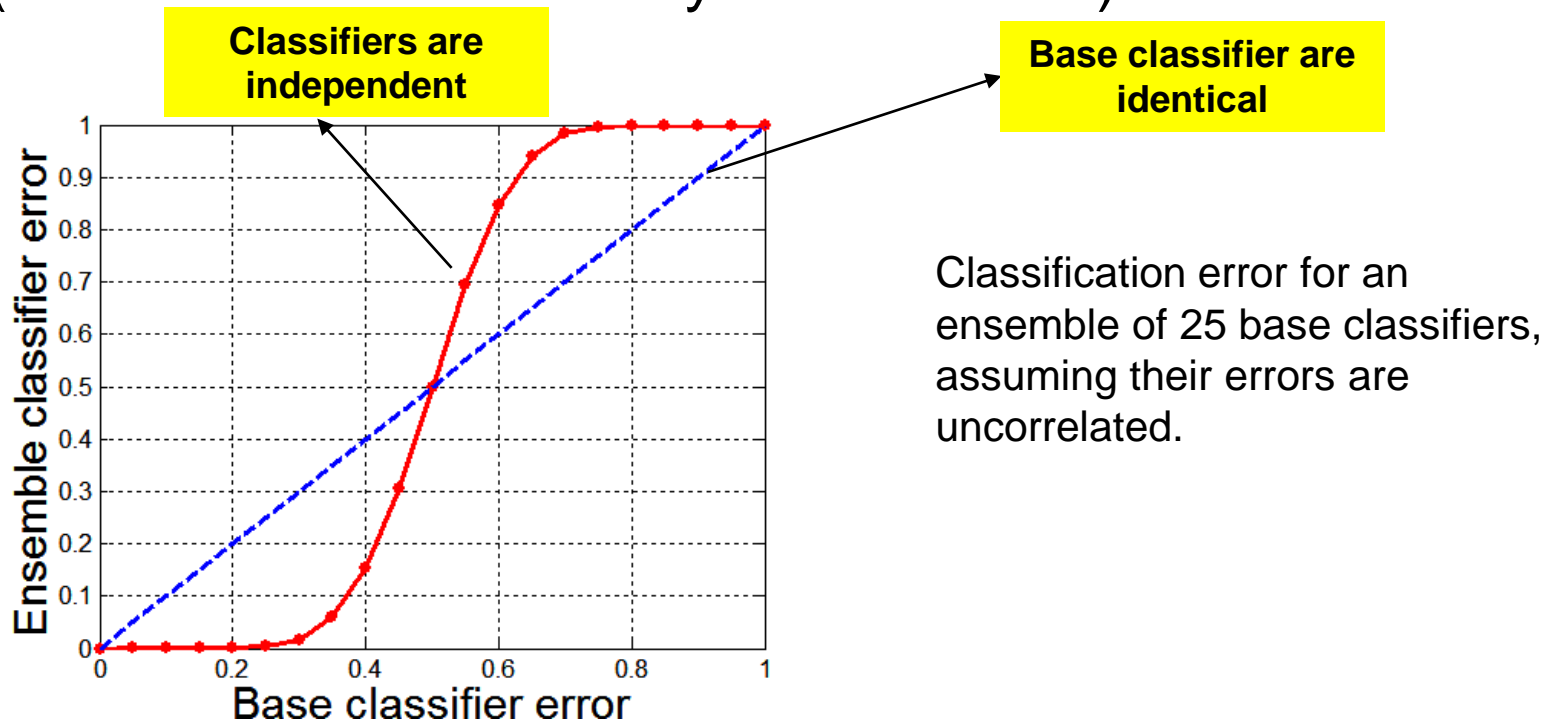
- Suppose there are 25 base classifiers
  - Each classifier has error rate,  $\epsilon = 0.35$
  - Majority vote of classifiers used for classification
  - If all classifiers are identical:
    - ◆ Error rate of ensemble =  $\epsilon$  (0.35)
  - If all classifiers are independent (errors are uncorrelated):
    - ◆ Error rate of ensemble = probability of having more than half of base classifiers being wrong

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1 - \epsilon)^{25-i} = 0.06$$

**Lower than the error rate of base classifiers**

# Necessary Conditions for Ensemble Methods

- Ensemble Methods work better than a single base classifier if:
  1. All base classifiers are independent of each other
  2. All base classifiers perform better than random guessing (error rate  $< 0.5$  for binary classification)



# Procedure for Ensemble Method

---

1. Let  $D$  denote the original training data,  $k$  is the number of base classifiers, and  $T$  be the test data
2. for  $i = 1$  to  $k$  do
  1. Create training set  $D_i$  from  $D$
  2. Build a base classifier  $C_i$  from  $D_i$
3. end for
4. for each test record  $x \in T$  do
  1.  $C^*(x) = \text{Vote}(C_1(x), C_2(x), \dots, C_k(x))$
5. end for

# Rationale for Ensemble Learning

---

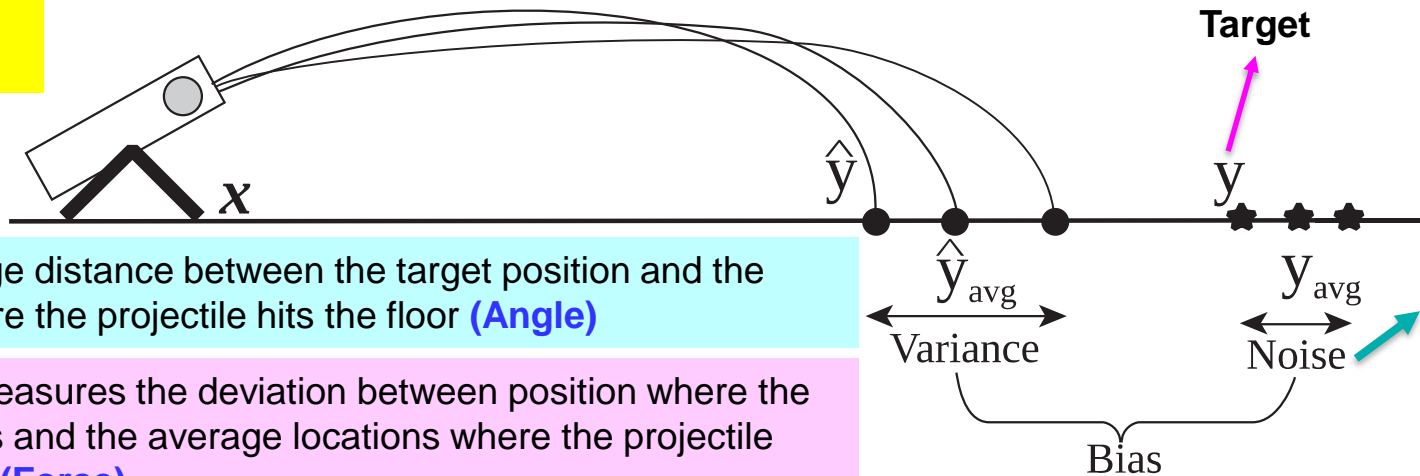
- Ensemble Methods work best with **unstable base classifiers**
  - Classifiers that are sensitive to minor perturbations in training set, due to *high model complexity*
    - ◆ Examples: Unpruned decision trees, ANNs, ...
  - **Low Bias** in finding optimal decision boundary
  - **High Variance** for minor changes in training set or model selection procedure

# Bias-Variance Decomposition

- Analogous problem of reaching a target  $y$  by firing projectiles from  $x$  (regression problem)

Depends on:

- (1) Angle
- (2) force



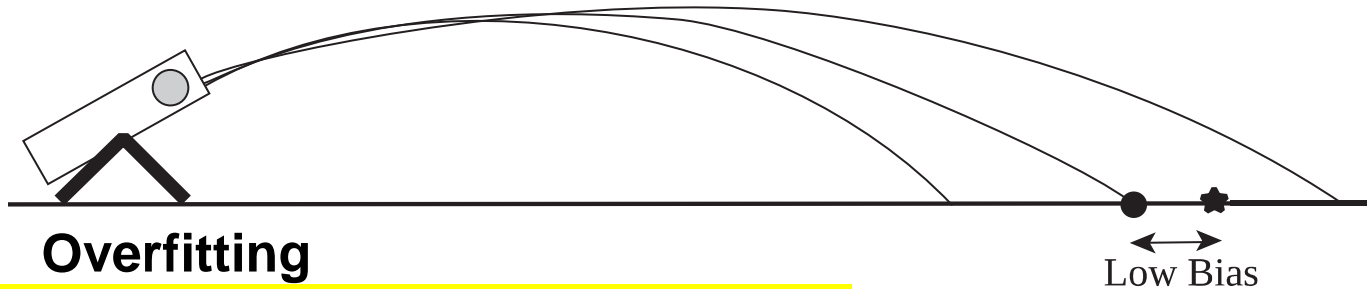
**Bias:** Average distance between the target position and the location where the projectile hits the floor (**Angle**)

**Variance:** measures the deviation between position where the projectile hits and the average locations where the projectile hits the floor (**Force**)

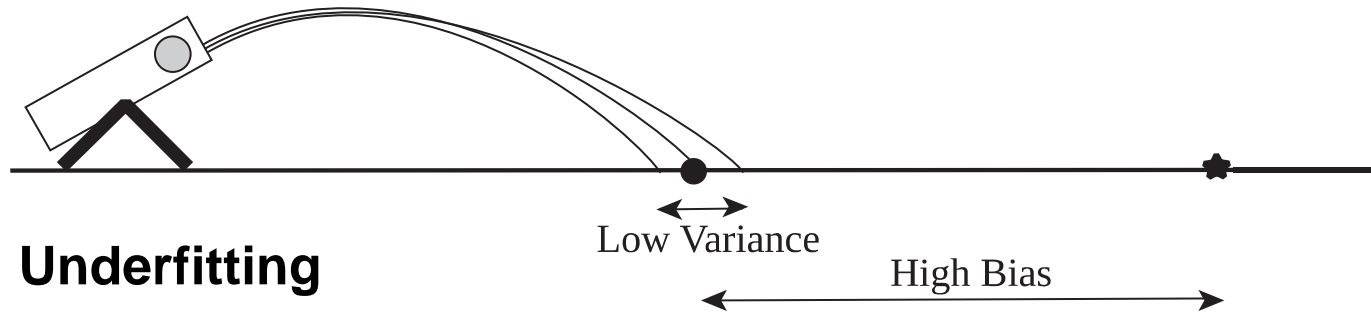
- For classification, gen. error or model  $m$  can be given by:

$$gen.error(m) = c_1 \times noise + bias(m) + c_2 \times variance(m)$$

# Bias-Variance Trade-off and Overfitting



Stronger assumption is made by the classifier about the nature of decision tree

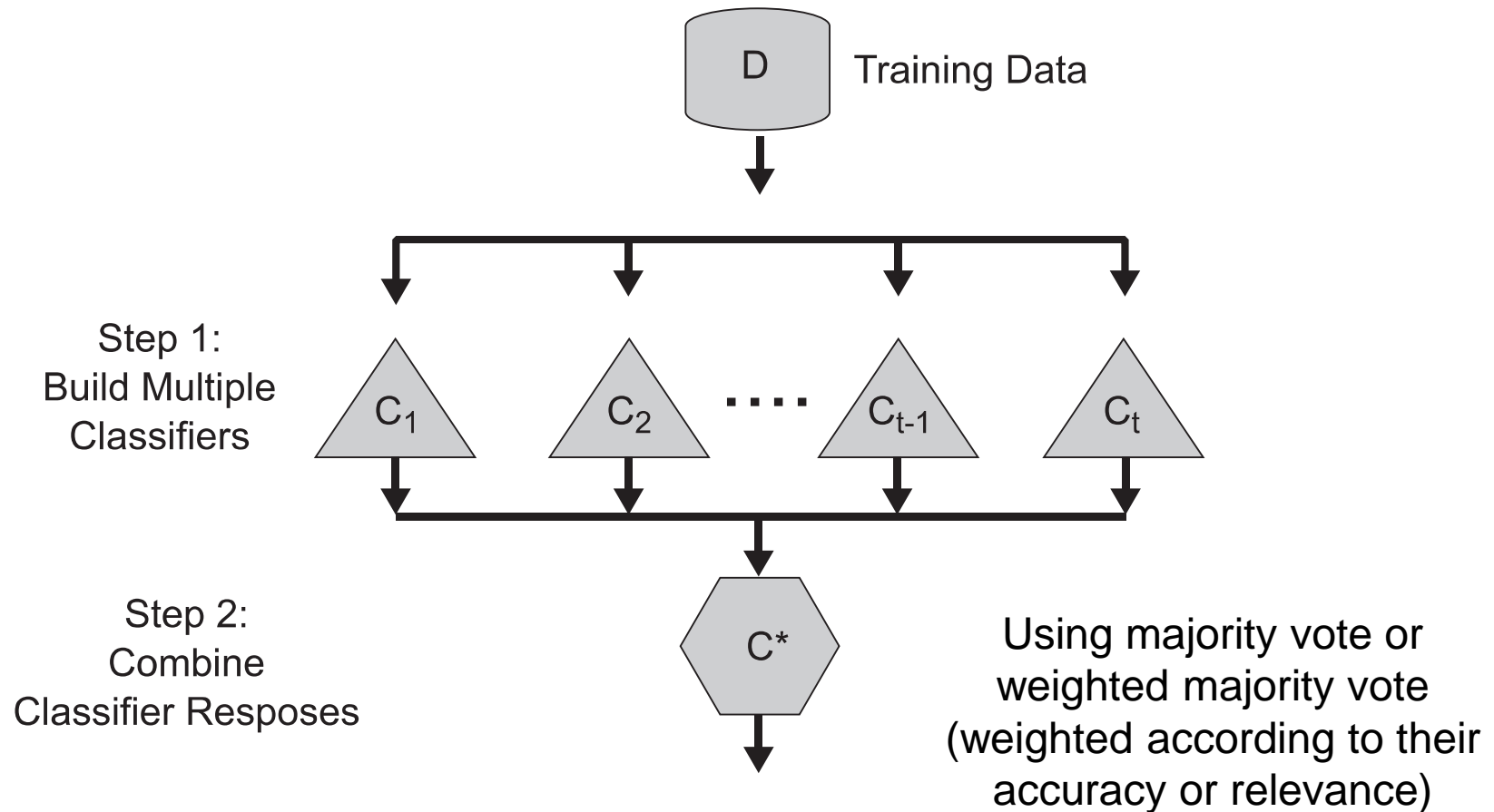


Variability in the training data

- Ensemble methods try to reduce the variance of complex models (with low bias) by *aggregating* responses of multiple base classifiers



# General Approach of Ensemble Learning



# Constructing Ensemble Classifiers

---

- By manipulating training set-
  - resampling training data
  - Example: bagging, boosting
- By manipulating input features
  - Subset of features are chosen at random
  - Example: random forests
- By manipulating class labels
  - Example: error-correcting output coding
- By manipulating learning algorithm
  - Example: injecting randomness in ANN or decision tree

# Bagging (Bootstrap AGGREGatING)

- Bootstrap sampling: sampling with replacement
- Bootstrap sample has same size as training set

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

- Build classifier on each bootstrap sample
- Probability of a training instance being selected in a bootstrap sample is:
  - $1 - (1 - 1/n)^n$  (n: number of training instances)
  - $\sim 0.632$  when n is large

# Bagging Algorithm

---

---

**Algorithm**      Bagging algorithm.

---

- 1: Let  $k$  be the number of bootstrap samples.
  - 2: **for**  $i = 1$  to  $k$  **do**
  - 3:     Create a bootstrap sample of size  $N$ ,  $D_i$ .
  - 4:     Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .
  - 5: **end for**
  - 6:  $C^*(x) = \operatorname{argmax}_y \sum_i \delta(C_i(x) = y)$ .  
     $\{\delta(\cdot) = 1 \text{ if its argument is true and } 0 \text{ otherwise.}\}$
-

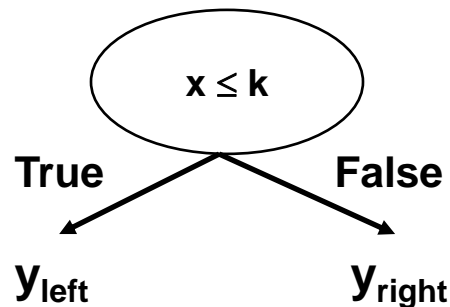
# Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump (decision tree of size 1)
  - Decision rule:  $x \leq k$  versus  $x > k$
  - Split point  $k$  is chosen based on entropy



# Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

# Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

$x \leq 0.7 \rightarrow y = 1$

$x > 0.7 \rightarrow y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \rightarrow y = 1$

$x > 0.3 \rightarrow y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

# Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \rightarrow y = -1$   
 $x > 0.75 \rightarrow y = 1$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.05 \rightarrow y = 1$   
 $x > 0.05 \rightarrow y = 1$



# Bagging Example

---

## □ Summary of Trained Decision Stumps:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

# Bagging Example

- Use majority vote (sign of sum of predictions) to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Predicted Class Sign	1	1	1	-1	-1	-1	-1	1	1	1

- Bagging can also increase the complexity (representation capacity) of simple classifiers such as decision stumps

# Boosting

---

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all  $N$  records are assigned equal weights (for being selected for training)
  - Unlike bagging, weights may change at the end of each boosting round

# Boosting

- Records that are wrongly classified will have their weights increased in the next round
- Records that are classified correctly will have their weights decreased in the next round

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

- Suppose example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

# AdaBoost

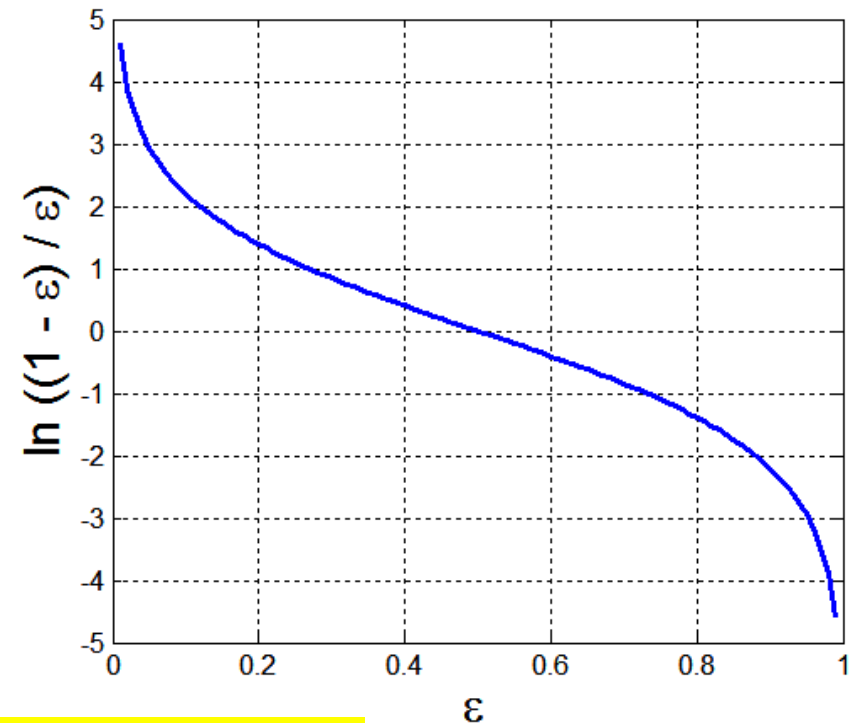
□ Base classifiers:  $C_1, C_2, \dots, C_T$

□ Error rate of a base classifier:

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^N w_j^{(i)} \delta(C_i(x_j) \neq y_j)$$

□ Importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \epsilon_i}{\epsilon_i} \right)$$



Large negative value if error rate is close to 1  
 $\alpha$  has large positive value if the error rate is close to 0 and

# AdaBoost Algorithm

---

- Weight update:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

Where  $Z_i$  is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to  $1/n$  and the resampling procedure is repeated
- Classification:

$$C^*(x) = \arg \max_y \sum_{i=1}^T \alpha_i \delta(C_i(x) = y)$$

# AdaBoost Algorithm

---

---

## Algorithm 4.6 AdaBoost algorithm.

---

- 1:  $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$ .    {Initialize the weights for all  $N$  examples.}
  - 2: Let  $k$  be the number of boosting rounds.
  - 3: **for**  $i = 1$  to  $k$  **do**
  - 4:    Create training set  $D_i$  by sampling (with replacement) from  $D$  according to  $\mathbf{w}$ .
  - 5:    Train a base classifier  $C_i$  on  $D_i$ .
  - 6:    Apply  $C_i$  to all examples in the original training set,  $D$ .
  - 7:     $\epsilon_i = \frac{1}{N} \left[ \sum_j w_j \delta(C_i(x_j) \neq y_j) \right]$     {Calculate the weighted error.}
  - 8:    **if**  $\epsilon_i > 0.5$  **then**
  - 9:      $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$ .    {Reset the weights for all  $N$  examples.}
  - 10:    Go back to Step 4.
  - 11:    **end if**
  - 12:     $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$ .
  - 13:    Update the weight of each example
  - 14: **end for**
  - 15:  $C^*(\mathbf{x}) = \operatorname{argmax}_y \sum_{j=1}^T \alpha_j \delta(C_j(\mathbf{x}) = y)$ .
-

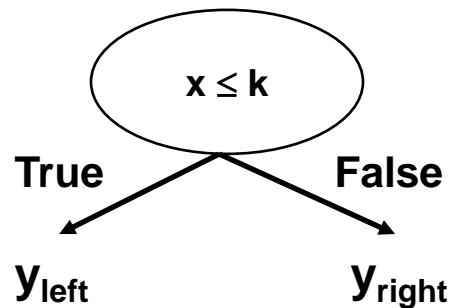
# AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule:  $x \leq k$  versus  $x > k$
  - Split point  $k$  is chosen based on entropy





# AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

# AdaBoost Example

## □ Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

## □ Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted  
Class

# Random Forest Algorithm

---

- Construct an ensemble of decision trees by manipulating training set as well as features
  - Use bootstrap sample to train every decision tree (similar to Bagging)
  - Use the following tree induction algorithm:
    - ◆ At every internal node of decision tree, randomly sample  $p$  attributes for selecting split criterion
    - ◆ Repeat this procedure until all leaves are pure (unpruned tree)



# Random forest

---

*Random Forest for Regression or Classification.*

---

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  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

# Characteristics of Random Forest

---

- Base classifiers are unpruned trees and hence are *unstable classifiers*
- Base classifiers are *decorrelated* (due to randomization in training set as well as features)
- Random forests reduce variance of unstable classifiers without negatively impacting the bias
- Selection of hyper-parameter  $p$ 
  - Small value ensures lack of correlation
  - High value promotes strong base classifiers
  - Common default choices:  $\sqrt{d}$ ,  $\log_2(d + 1)$

**Thank You**