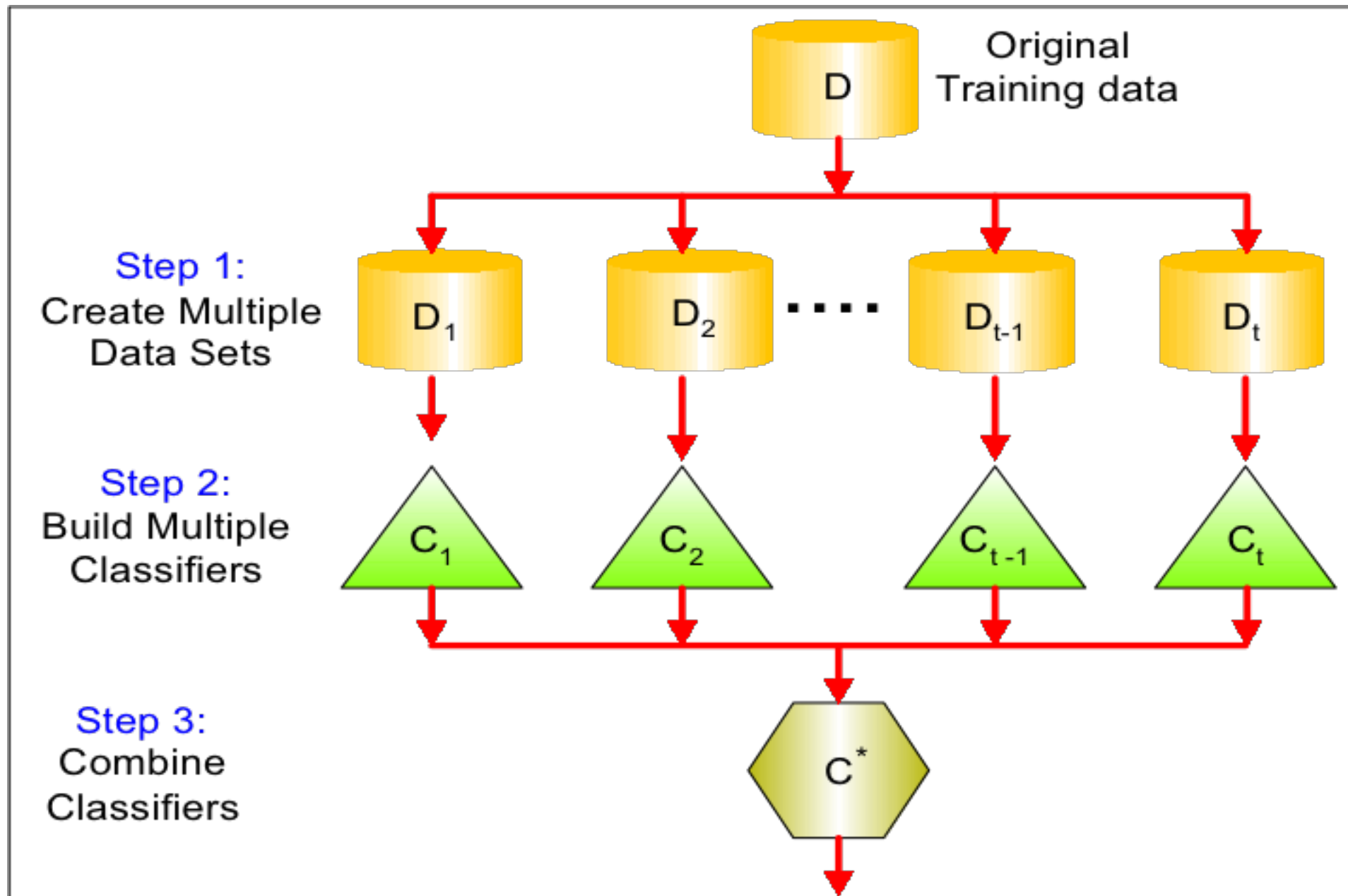


Ensemble Learning: An Introduction

Adapted from Slides by Tan,
Steinbach, Kumar

General Idea



Why does it work?

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

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

Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
 - Bagging
 - Boosting

Bagging

- Sampling with replacement

Training Data
↙

Data ID	1	2	3	4	5	6	7	8	9	10
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
- Each sample has probability $(1 - 1/n)^n$ of being selected as test data
- Training data = $1 - (1 - 1/n)^n$ of the original data

The 0.632 bootstrap

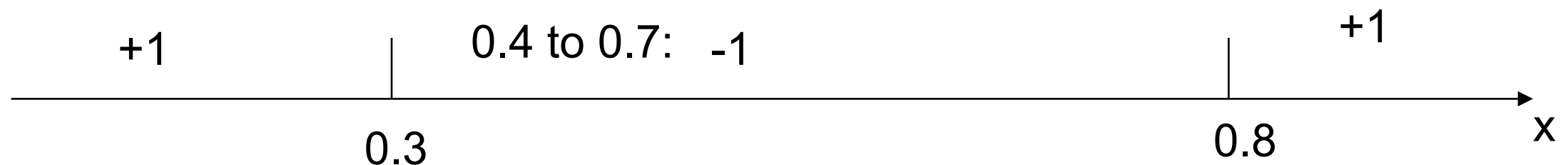
- This method is also called the *0.632 bootstrap*
 - A particular training data has a probability of $1-1/n$ of *not* being picked
 - Thus its probability of ending up in the test data (not selected) is:

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

- This means the training data will contain approximately 63.2% of the instances

Example of Bagging

Assume that the training data is:



Goal: find a collection of 10 simple thresholding classifiers that collectively can classify correctly.

-Each simple (or weak) classifier is:

($x \leq K \rightarrow \text{class} = +1 \text{ or } -1$ depending on which value yields the lowest error; where K is determined by entropy minimization)

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

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

Bagging Round 2:

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

$x \leq 0.65 \implies y = 1$

$x > 0.65 \implies y = 1$

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

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

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

$x \leq 0.3 \implies y = 1$

$x > 0.3 \implies y = -1$

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

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

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

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

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

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

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

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

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

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

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

$x \leq 0.05 \implies y = -1$

$x > 0.05 \implies y = 1$

Figure 5.35. Example of bagging.

Bagging (applied to training data)

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
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy of ensemble classifier: 100% ☺

Bagging- Summary

- Works well if the base classifiers are unstable (complement each other)
- Increased accuracy because it ***reduces the variance*** of the individual classifier
- Does not focus on any particular instance of the training data
 - Therefore, less susceptible to model over-fitting when applied to noisy data
- What if we want to focus on a particular instances of training data?

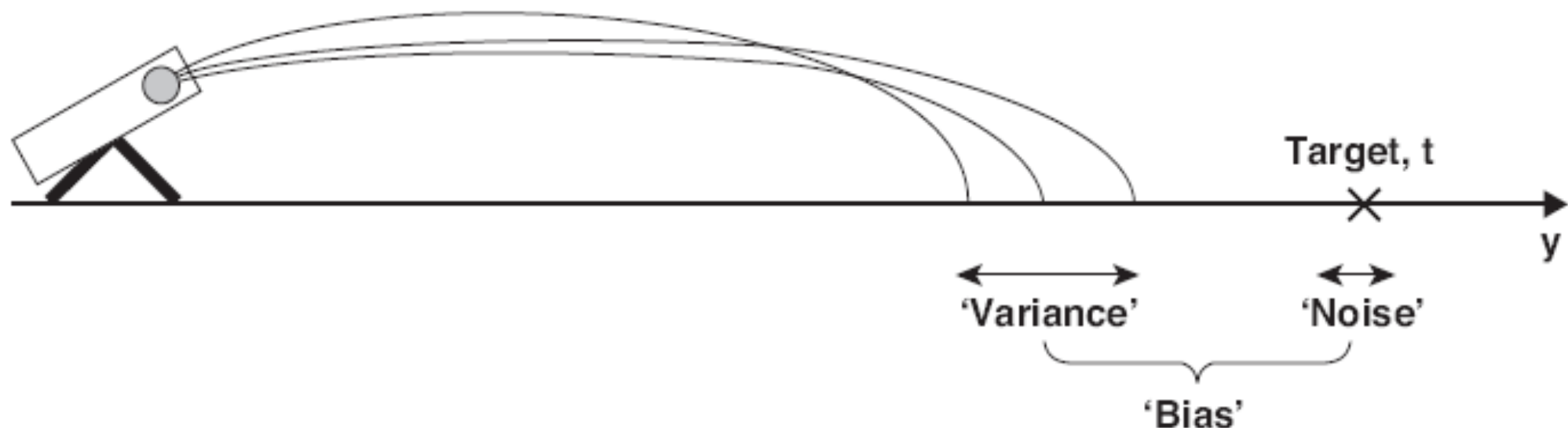
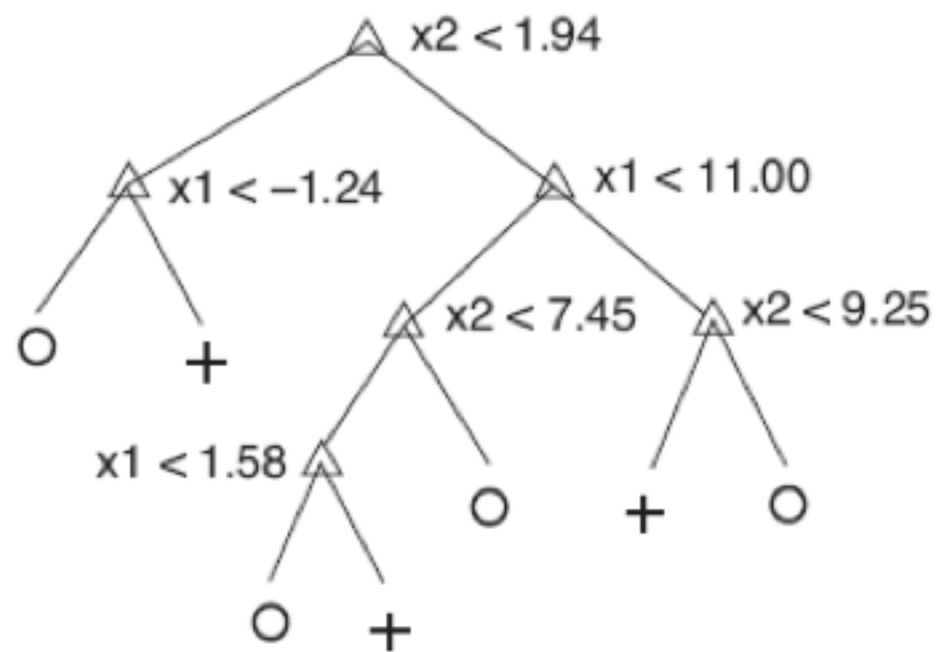


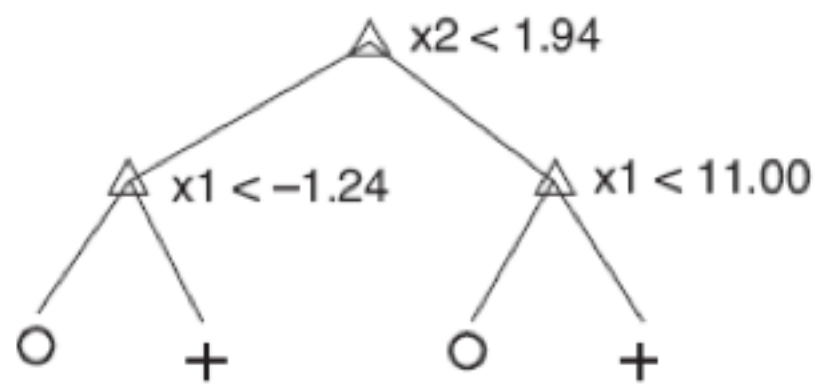
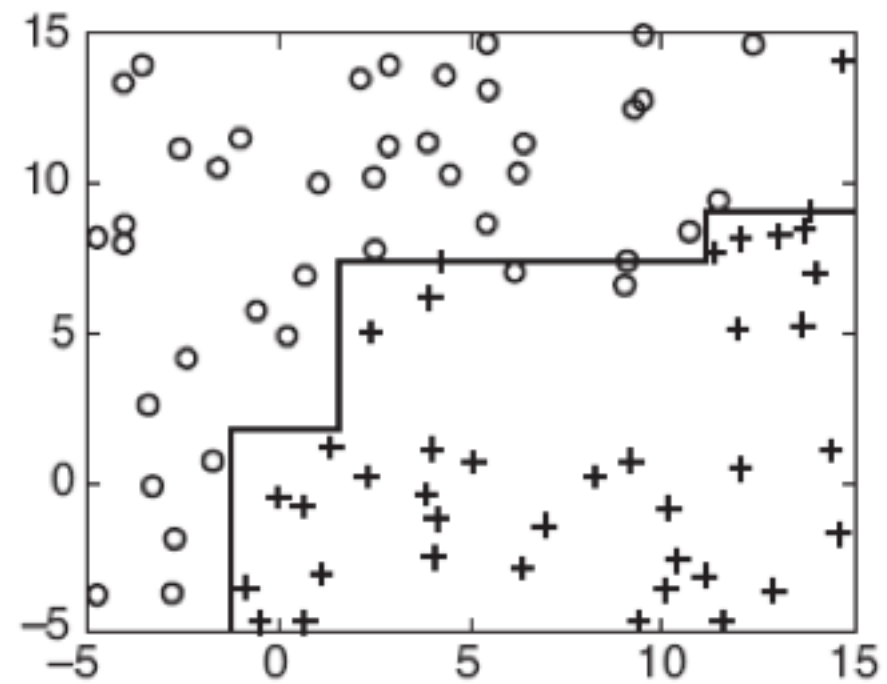
Figure 5.32. Bias-variance decomposition.

In general,

- **Bias** is contributed to by the training error; a complex model has low bias.
- **Variance** is caused by future error; a complex model has High variance.
- Bagging reduces the variance in the base classifiers.



(a) Decision tree T_1



(b) Decision tree T_2

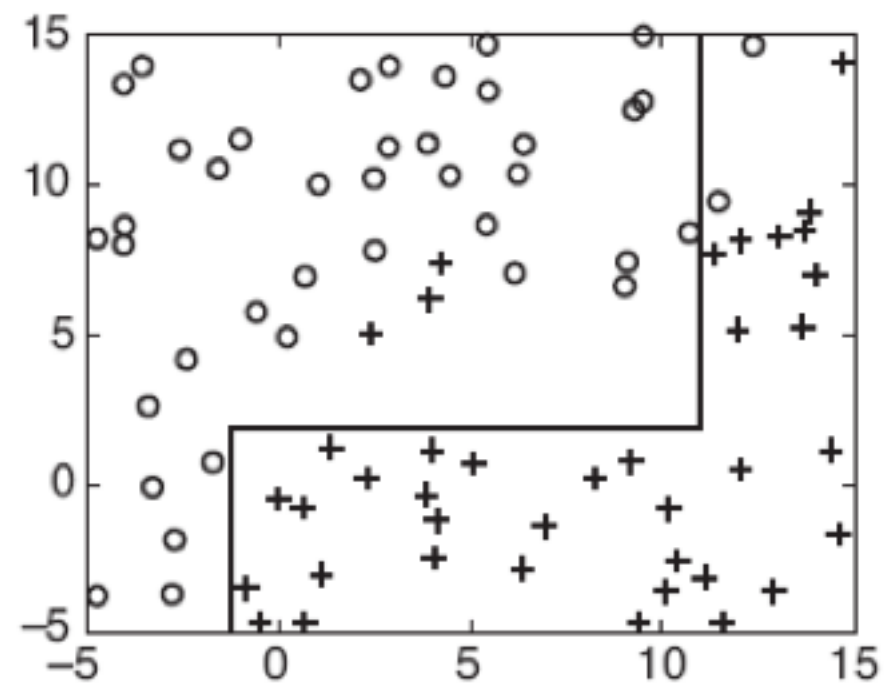
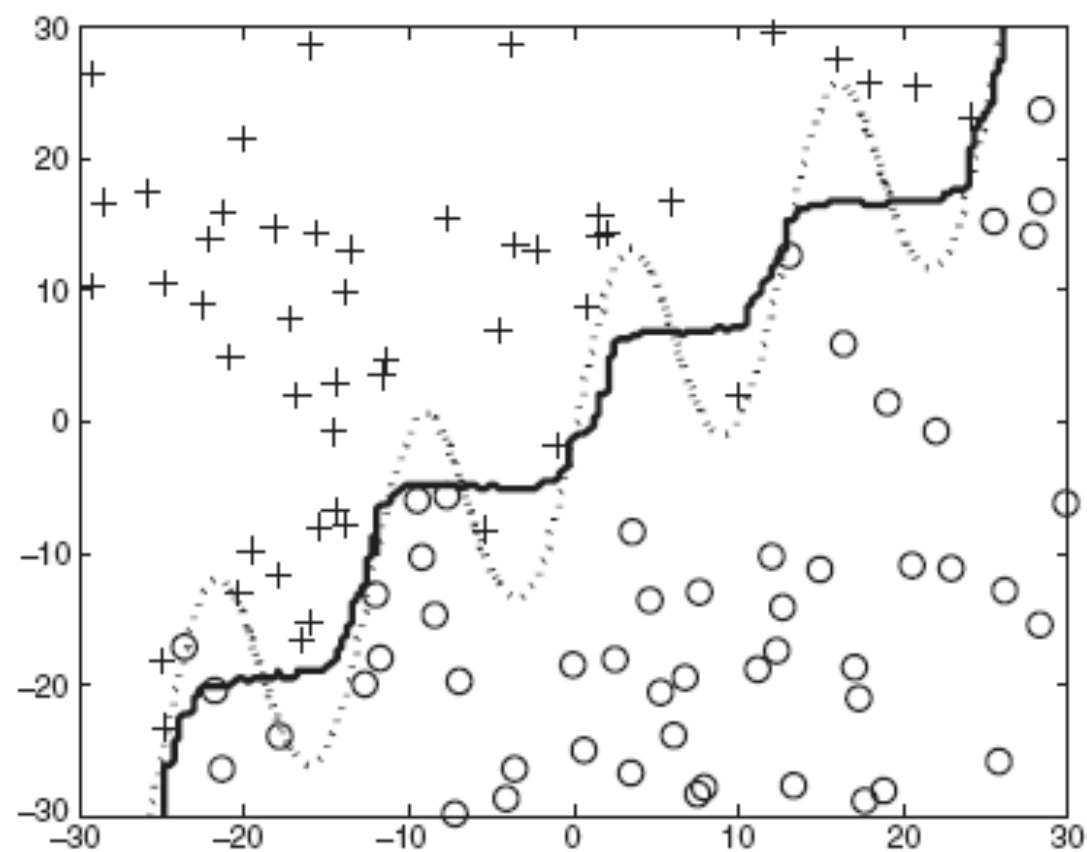
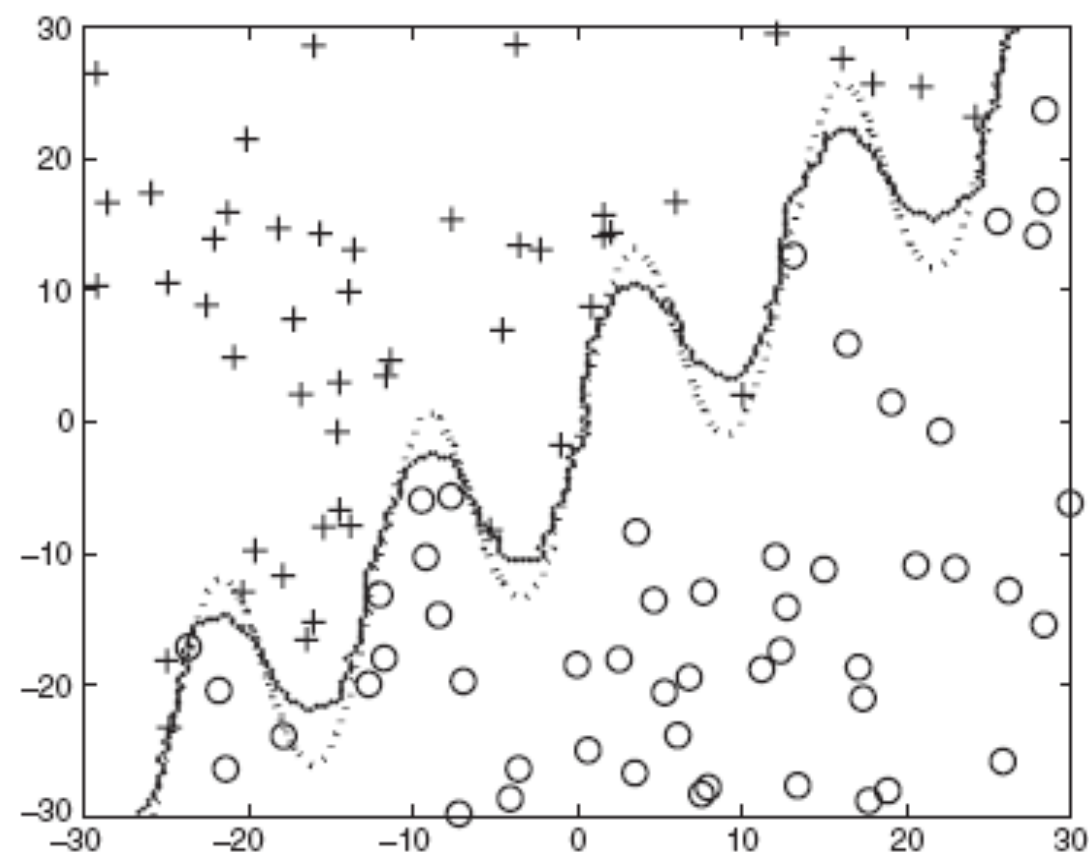


Figure 5.33. Two decision trees with different complexities induced from the same training data.



(a) Decision boundary for decision tree.



(b) Decision boundary for 1-nearest neighbor.

Figure 5.34. Bias of decision tree and 1-nearest neighbor classifiers.

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
 - Unlike bagging, weights may change at the end of a boosting round

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

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

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

Boosting

- Equal weights are assigned to each training instance ($1/d$ for round 1) at first
- After a classifier C_i is learned, the weights are adjusted to allow the subsequent classifier C_{i+1} to “pay more attention” to data that were misclassified by C_i .
- Final boosted classifier C^* combines the votes of each individual classifier
 - Weight of each classifier’s vote is a function of its accuracy
- Adaboost – popular boosting algorithm

Adaboost (Adaptive Boost)

- Input:
 - Training set D containing N instances
 - T rounds
 - A classification learning scheme
- Output:
 - A composite model

Adaboost: Training Phase

- Training data D contain N labeled data $(X_1, y_1), (X_2, y_2), (X_3, y_3), \dots, (X_N, y_N)$
- Initially assign equal weight $1/d$ to each data
- To generate T base classifiers, we need T rounds or iterations
- Round i , data from D are sampled with replacement, to form D_i (size N)
- Each data's chance of being selected in the next rounds depends on its weight
 - Each time the new sample is generated directly from the training data D with different sampling probability according to the weights; these weights are not zero

Adaboost: Training Phase

- Base classifier C_i , is derived from training data of D_i
- Error of C_i is tested using D_i
- Weights of training data are adjusted depending on how they were classified
 - Correctly classified: Decrease weight
 - Incorrectly classified: Increase weight
- Weight of a data indicates how hard it is to classify it (directly proportional)

Adaboost: Testing Phase

- The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be
- Weight of a classifier C_i 's vote is

$$a_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

- Testing:
 - For each class c , sum the weights of each classifier that assigned class c to X (unseen data)
 - The class with the highest sum is the WINNER!

$$C^*(x_{test}) = \arg \max_y \sum_{i=1}^T a_i \delta(C_i(x_{test}) = y)$$

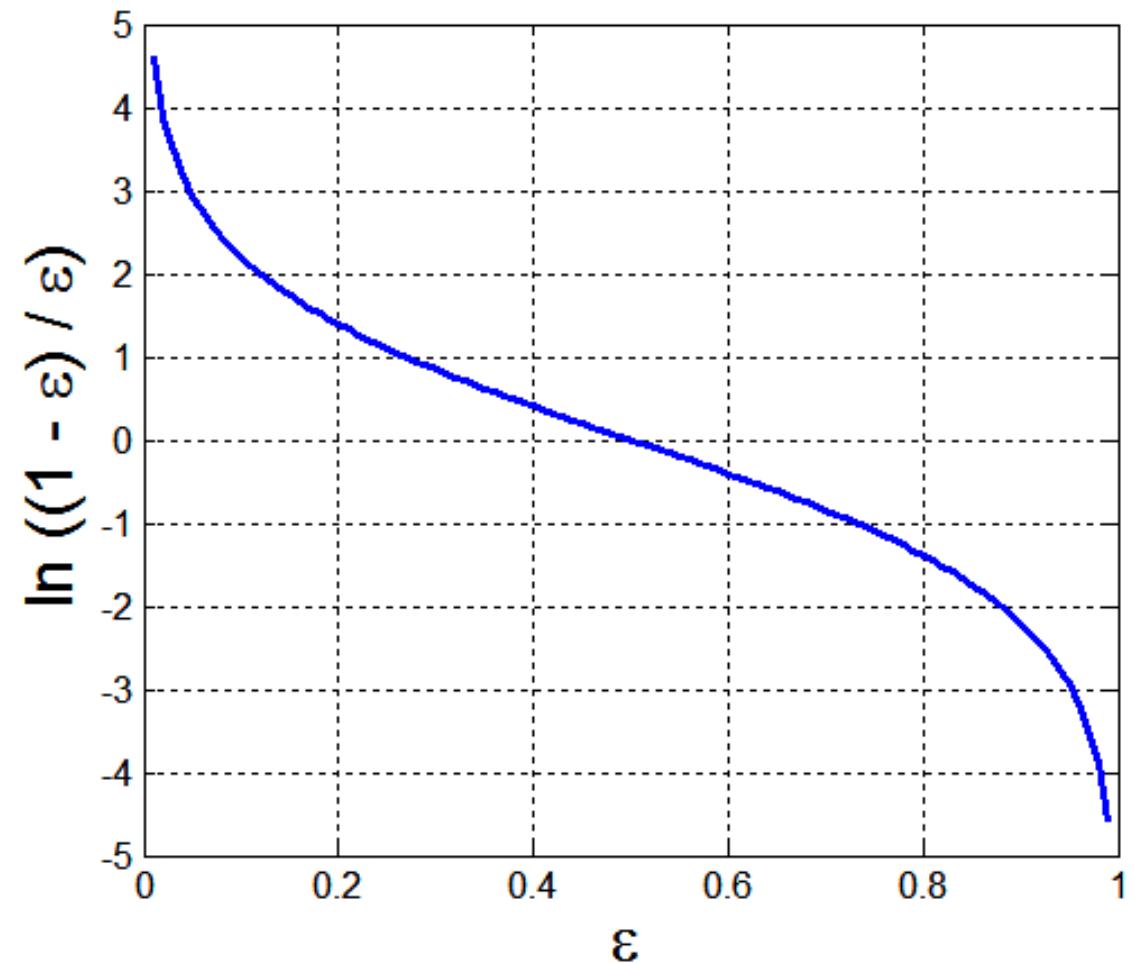
Example: AdaBoost

- Base classifiers: C_1, C_2, \dots, C_T
- Error rate: (i = index of classifier, j =index of instance)

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

- Importance of a classifier:

$$a_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$



Example: AdaBoost

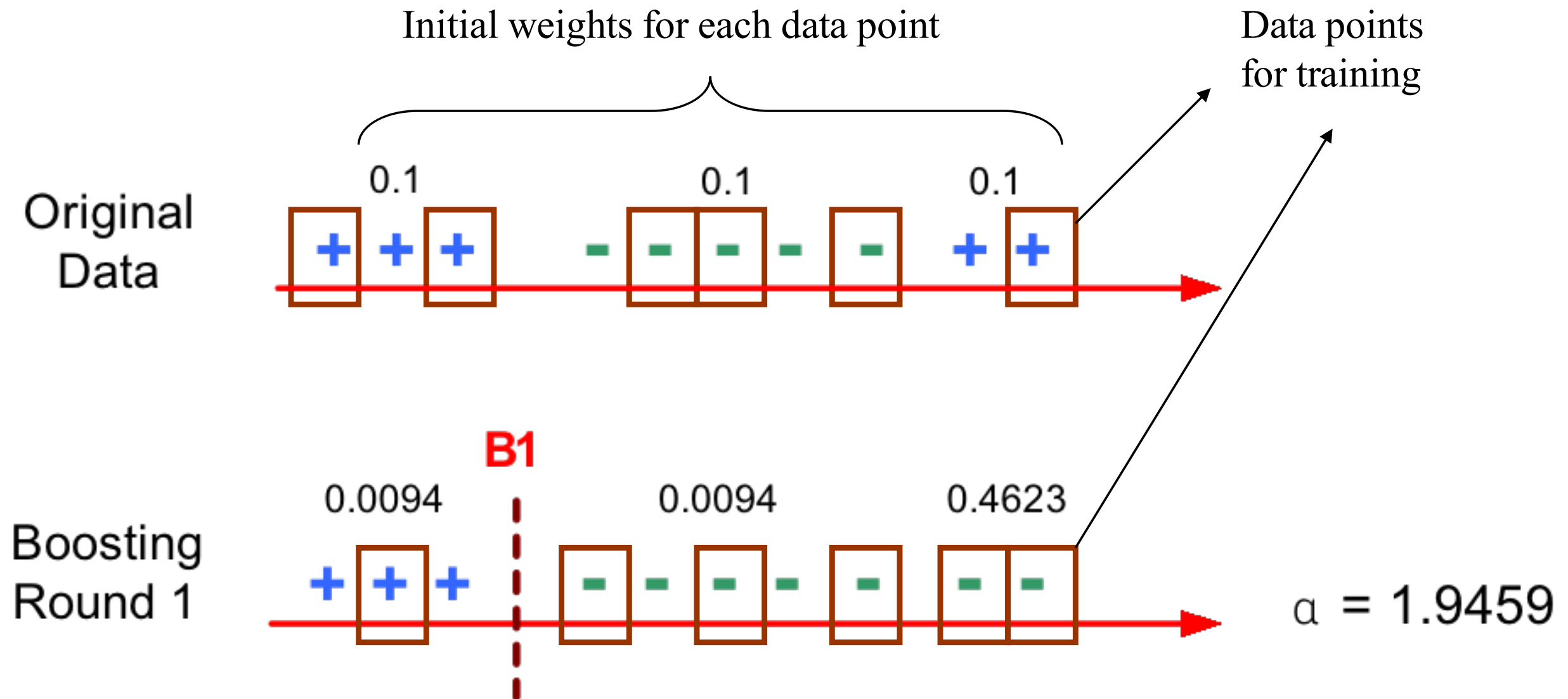
- Assume: N training data in D , T rounds, (x_j, y_j) are the training data, C_i , a_i are the classifier and weight of the i^{th} round, respectively.
- Weight update on all training data in D :

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

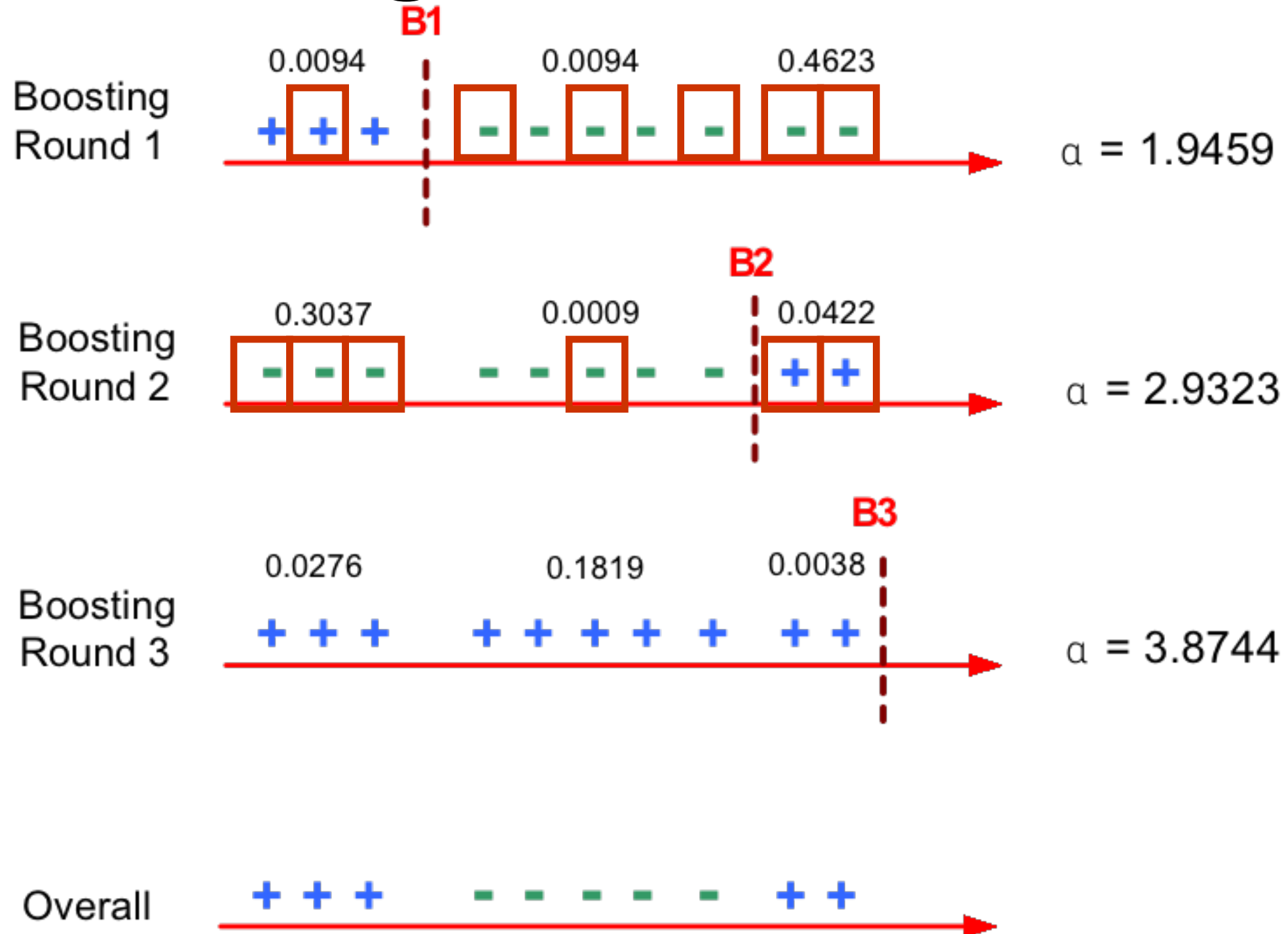
where Z_i is the normalization factor

$$C^*(x_{\text{test}}) = \arg \max_y \sum_{i=1}^T a_i \delta(C_i(x_{\text{test}}) = y)$$

Illustrating AdaBoost



Illustrating AdaBoost



Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Random Forests grows many trees
 - Ensemble of unpruned decision trees
 - Each base classifier classifies a “new” vector of attributes from the original data
 - Final result on classifying a new instance: voting. Forest chooses the classification result having the most votes (over all the trees in the forest)

Random Forests

- 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

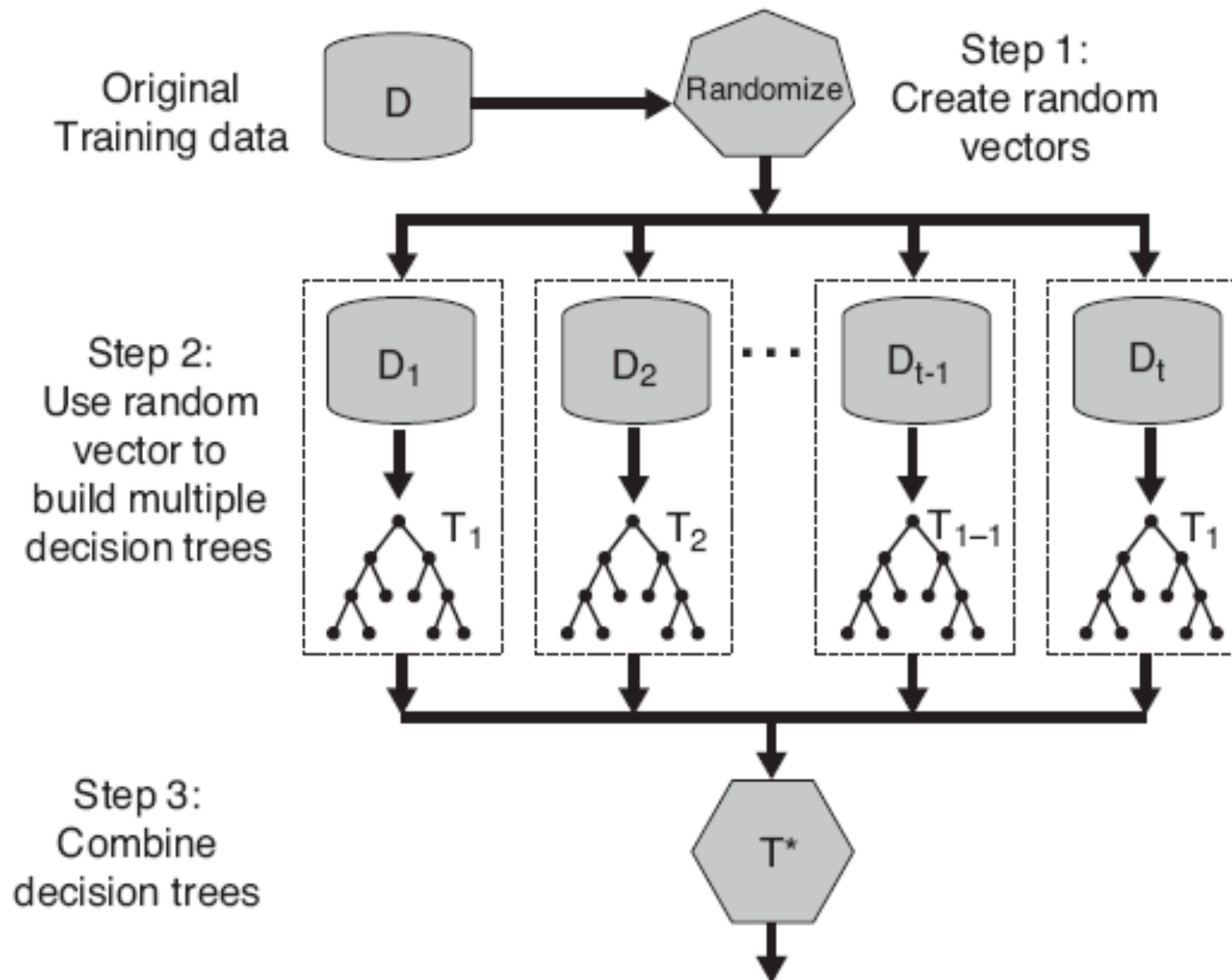


Figure 5.40. Random forests.

Methods for Growing the Trees

- Fix a $m \leq M$. At each node
 - Method 1:
 - Choose m attributes randomly, compute their information gains, and choose the attribute with the largest gain to split
 - Method 2:
 - (When M is not very large): select L of the attributes randomly. Compute a linear combination of the L attributes using weights generated from $[-1, +1]$ randomly. That is, new $A = \text{Sum}(W_i * A_i)$, $i=1..L$.
 - Method 3:
 - Compute the information gain of all M attributes. Select the top m attributes by information gain. Randomly select one of the m attributes as the splitting node.

Random Forest Algorithm:

method 1 in previous slide

- M input features in training data, a number $m \ll M$ is specified such that **at each node**, m features are selected at random out of the M and the best split on these m features is used to split the node. (In weather data, $M=4$, and m is between 1 and 4)
- m is held constant during the forest growing
- Each tree is grown to the largest extent possible (deep tree, overfit easily), and there is no pruning

Generalization Error of Random Forests (page 291 of Tan book)

- It can be proven that the generalization Error $\leq \frac{\rho(1-s^2)}{s^2}$,
 - ρ is the average correlation among the trees
 - s is the strength of the tree classifiers
 - Strength is defined as *how certain* the classification results are on the training data on average
 - How certain is measured $\Pr(C1|X) - \Pr(C2|X)$, where $C1, C2$ are class values of two highest probability in decreasing order for input instance X .
- Thus, higher diversity and accuracy is good for performance