Ensemble Methods

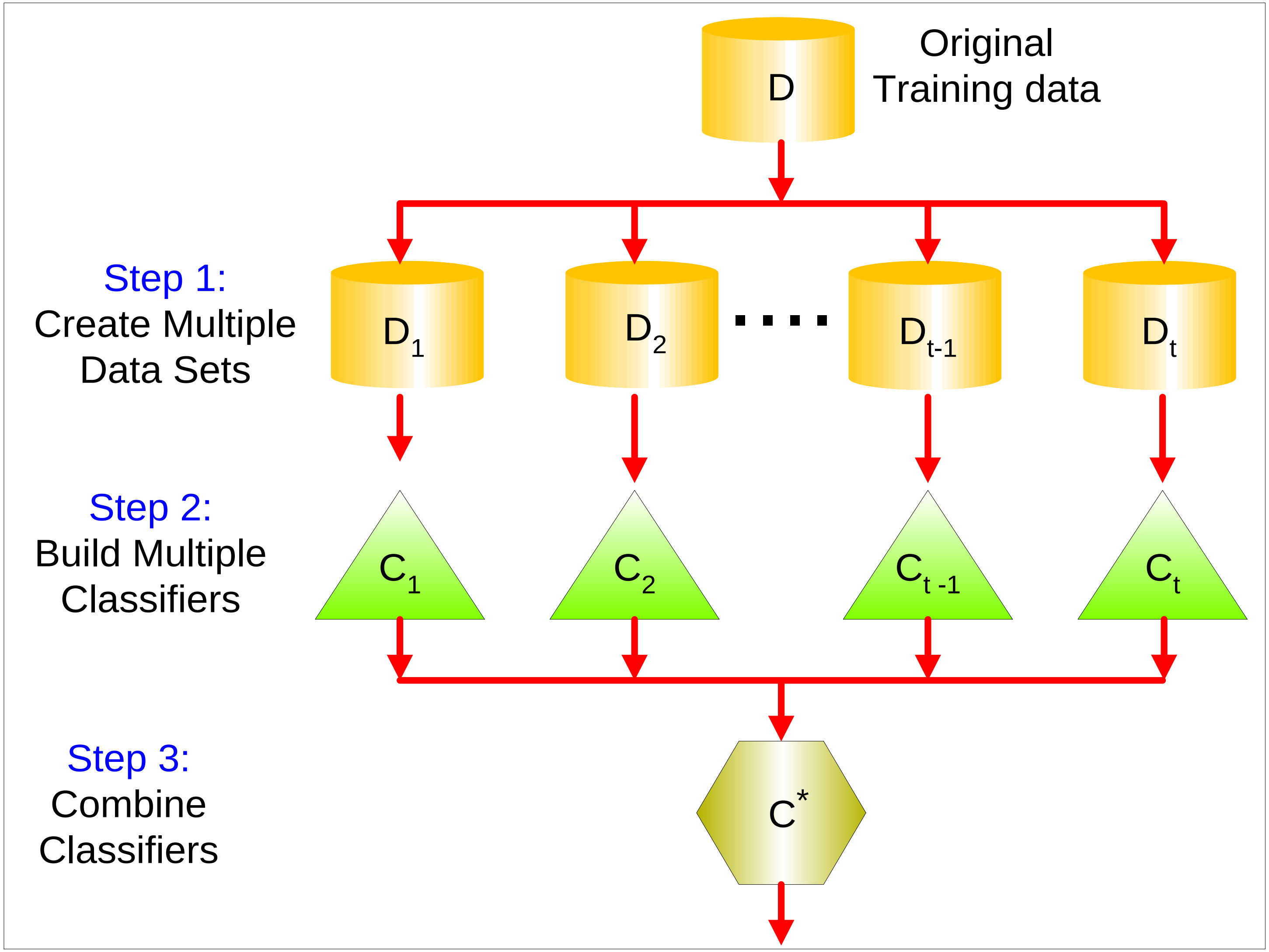
- Construct a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

Ensemble Methods (Section 5.6, page 276)

- **Ensemble methods aim at "improving classification accuracy by aggregating the predictions from multiple classifiers" (page 276)**

- **One of the most obvious ways of doing this is simply by averaging classifiers which make errors somewhat independently of each other**

General Idea



Why does it work?

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

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

Suppose I have 5 classifiers which each classify a point correctly 70% of the time. If these 5 classifiers are completely independent and I take the majority vote, how often is the majority vote correct for that point?

Suppose I have 5 classifiers which each classify a point correctly 70% of the time. If these 5 classifiers are completely independent and I take the majority vote, how often is the majority vote correct for that point?

Solution (continued):

```
10*.7^3*.3^2 + 5*.7^4*.3^1 + .7^5
```

or

```
1-pbinom(2, 5, .7)
```

Suppose I have 101 classifiers which each classify a point correctly 70% of the time. If these 101 classifiers are completely independent and I take the majority vote, how often is the majority vote correct for that point?

Suppose I have 101 classifiers which each classify a point correctly 70% of the time. If these 101 classifiers are completely independent and I take the majority vote, how often is the majority vote correct for that point?

Solution (continued):

```
1-pbinom(50, 101, .7)
```

Examples of Ensemble Methods

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

Ensemble Methods (Section 5.6, page 276)

- Ensemble methods include

-Bagging (page 283)

-Boosting (page 285)

-Random Forests (page 290)

- Bagging builds different classifiers by training on repeated samples (with replacement) from the data

- Boosting combines simple base classifiers by upweighting data points which are classified incorrectly

- Random Forests averages many trees which are constructed with some amount of randomness

Bagging

- Sampling with replacement

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

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 <= 0.35 ==> y = 1
x > 0.35 ==> 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 <= 0.65 ==> y = 1
x > 0.65 ==> 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 <= 0.35 ==> y = 1
x > 0.35 ==> 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 <= 0.3 ==> y = 1
x > 0.3 ==> 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 <= 0.35 ==> y = 1
x > 0.35 ==> 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 <= 0.75 ==> y = -1
x > 0.75 ==> 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 <= 0.75 ==> y = -1
x > 0.75 ==> 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 <= 0.75 ==> y = -1
x > 0.75 ==> 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 <= 0.75 ==> y = -1
x > 0.75 ==> 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 <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

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.

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 boosting round

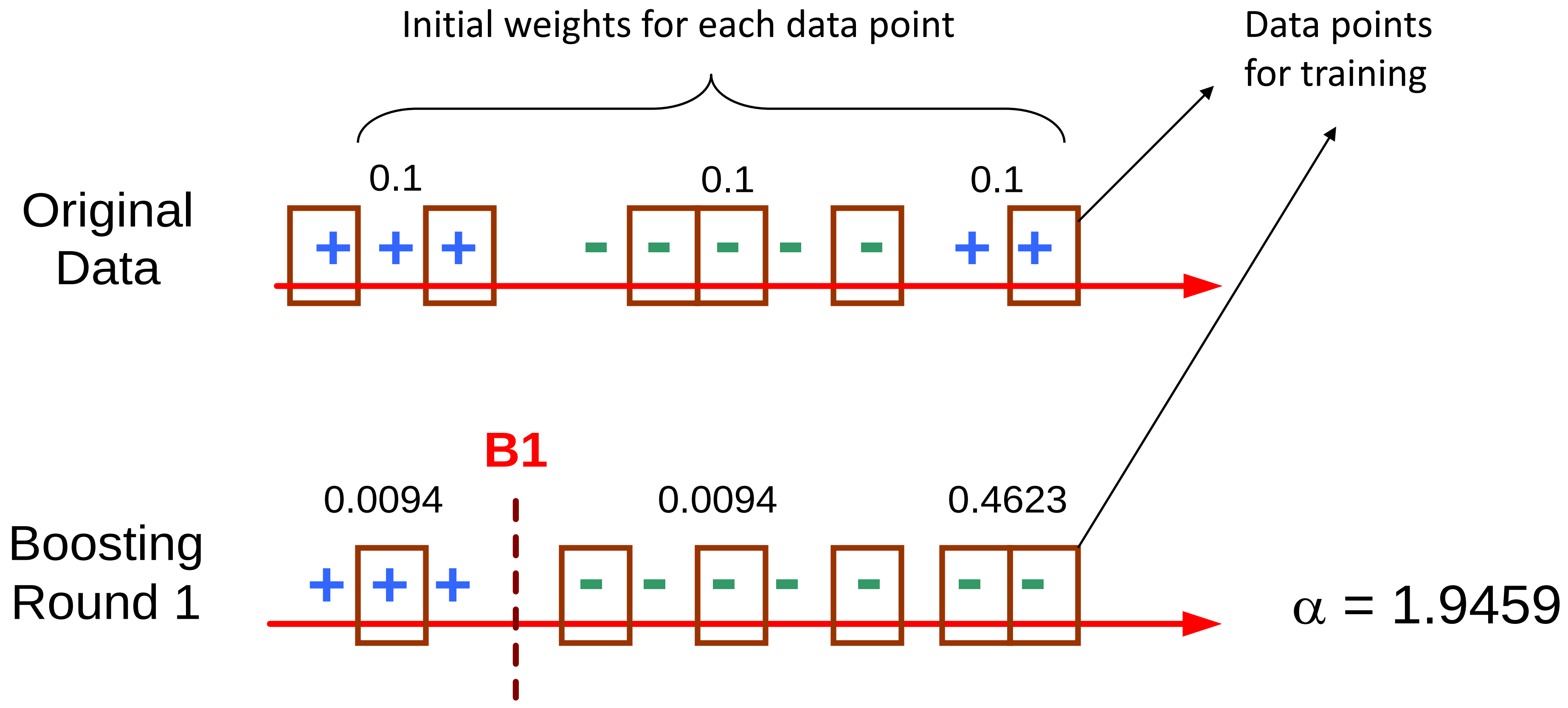
Boosting

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

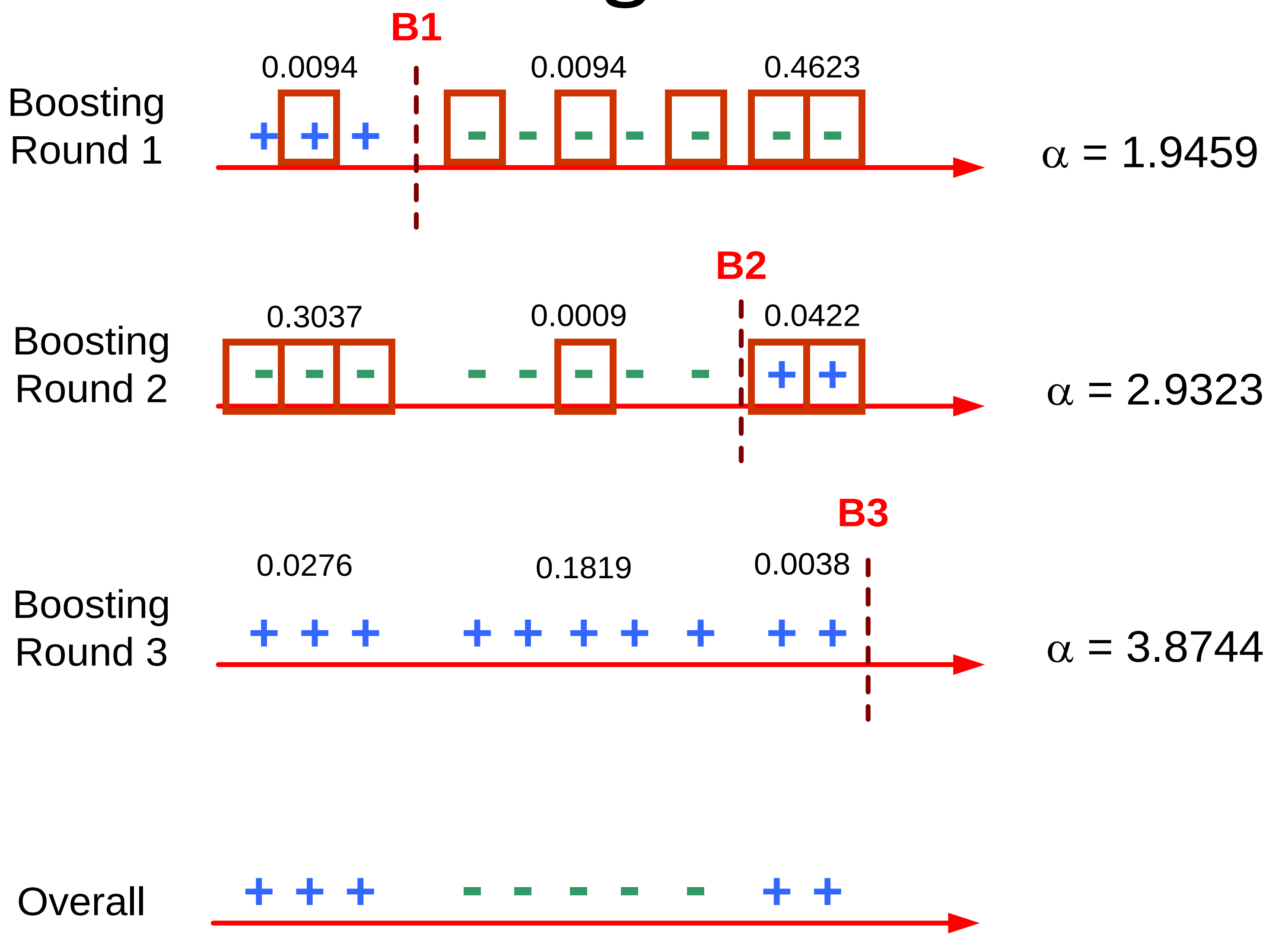
Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

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

Illustrating AdaBoost



Illustrating AdaBoost



Example: AdaBoost

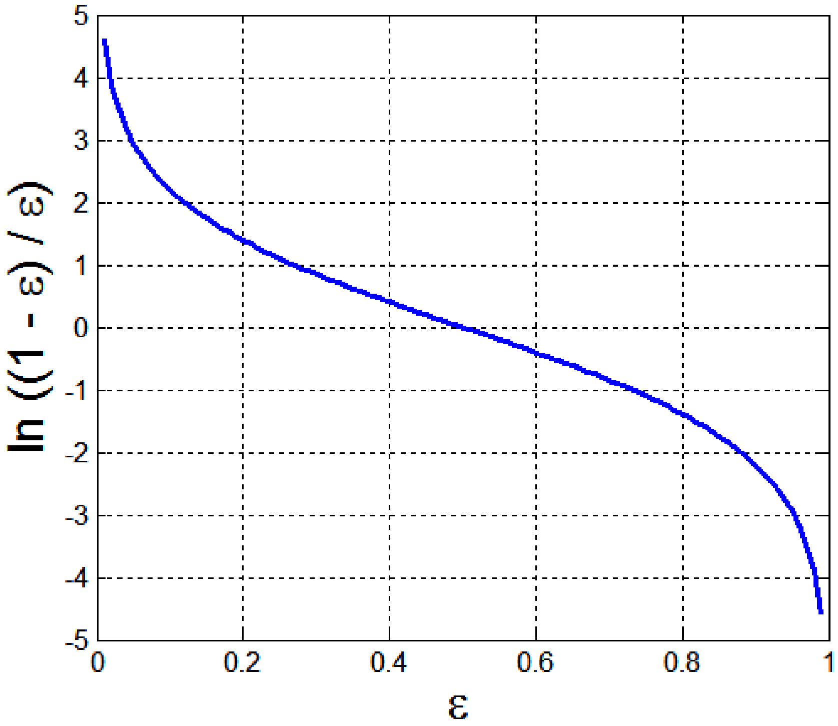
- Base classifiers: $C_1, C_2, \ldots, C_T$

- Error rate:

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

- Importance of a classifier:

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



Example: AdaBoost

- Weight update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j 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:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

AdaBoost

- **Here is a version of the AdaBoost algorithm**

First let $F_0(x_i) = 0$ for all x_i and initialize weights $w_i = 1/n$ for $i = 1, ..., n$. Then repeat the following for m from 1 to M:

- Fit the classifier g_m to the training data using weights w_i where g_m maps each x_i to -1 or 1.
- Compute the weighted error rate $\epsilon_m \equiv \sum_{i=1}^{n} w_i \mathrm{I}[y_i \neq g_m(x_i)]$ and half its log-odds, $\alpha_m \equiv \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$.
- Let $F_m = F_{m-1} + \alpha_m g_m$.
- Replace the weights w_i with $w_i \equiv w_i e^{-\alpha_m g_m(x_i) y_i}$ and then renormalize by replacing each w_i by $w_i / (\sum w_i)$.

The final classifier is 1 if $F_M > 0$ and -1 otherwise.

- **The algorithm repeats until a chosen stopping time**

- **The final classifier is based on the sign of F_m**

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

(a) Training records chosen during boosting

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

(b) Weights of training records

Figure 5.38. Example of boosting.

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

(a)

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

(b)

Figure 5.39. Example of combining classifiers constructed using the AdaBoost approach.

Random Forests (Section 5.6.6, page 290)

- **One way to create random forests is to grow decision trees top down but at each node consider only a random subset of attributes for splitting instead of all the attributes**

- **Random Forests are a very effective technique**

- **They are based on the paper**

  L. Breiman. Random forests. Machine Learning, 45:5-32, 2001

- **They can be fit in R using the function randomForest() in the library randomForest**

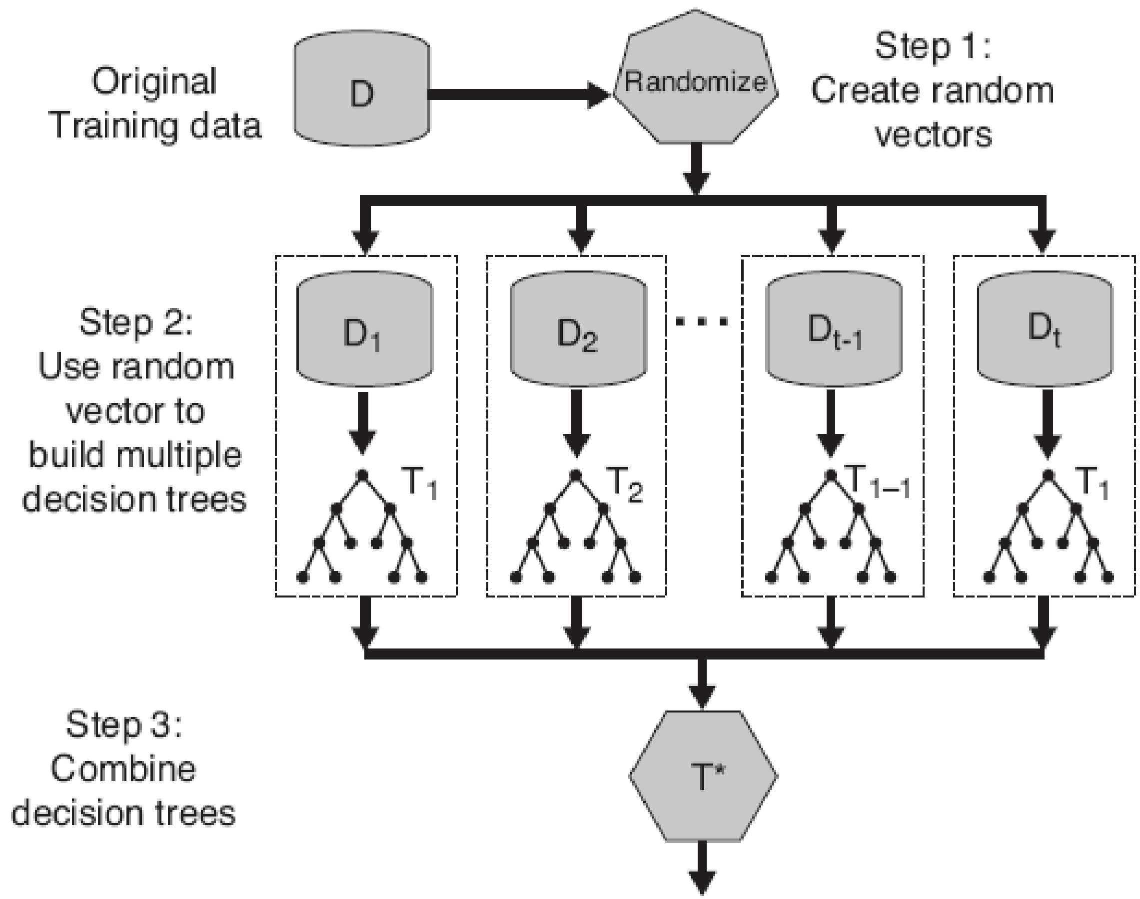


Figure 5.40. Random forests.

Use randomForest() in R to fit the default Random Forest to the last column of the sonar training data at
http://sites.google.com/site/stats202/data/sonar_train.csv

Compute the misclassification error for the test data at
http://sites.google.com/site/stats202/data/sonar_test.csv

Use randomForest() in R to fit the default Random Forest to the last column of the sonar training data at
http://sites.google.com/site/stats202/data/sonar_train.csv
Compute the misclassification error for the test data at
http://sites.google.com/site/stats202/data/sonar_test.csv

Solution:

```
install.packages("randomForest")
library(randomForest)
train<-read.csv("sonar_train.csv",header=FALSE)
test<-read.csv("sonar_test.csv",header=FALSE)
y<-as.factor(train[,61])
x<-train[,1:60]
y_test<-as.factor(test[,61])
x_test<-test[,1:60]
fit<-randomForest(x,y)
1-sum(y_test==predict(fit,x_test))/length(y_test)
```

Table 5.5. Comparing the accuracy of a decision tree classifier against three ensemble methods

Data Set	Number of (Attributes, Classes, Records)	Decision Tree (%)	Bagging (%)	Boosting (%)	RF (%)
Anneal	(39, 6, 898)	92.09	94.43	95.43	95.43
Australia	(15, 2, 690)	85.51	87.10	85.22	85.80
Auto	(26, 7, 205)	81.95	85.37	85.37	84.39
Breast	(11, 2, 699)	95.14	96.42	97.28	96.14
Cleve	(14, 2, 303)	76.24	81.52	82.18	82.18
Credit	(16, 2, 690)	85.8	86.23	86.09	85.8
Diabetes	(9, 2, 768)	72.40	76.30	73.18	75.13
German	(21, 2, 1000)	70.90	73.40	73.00	74.5
Glass	(10, 7, 214)	67.29	76.17	77.57	78.04
Heart	(14, 2, 270)	80.00	81.48	80.74	83.33
Hepatitis	(20, 2, 155)	81.94	81.29	83.87	83.23
Horse	(23, 2, 368)	85.33	85.87	81.25	85.33
Ionosphere	(35, 2, 351)	89.17	92.02	93.73	93.45
Iris	(5, 3, 150)	94.67	94.67	94.00	93.33
Labor	(17, 2, 57)	78.95	84.21	89.47	84.21
Led7	(8, 10, 3200)	73.34	73.66	73.34	73.06
Lymphography	(19, 4, 148)	77.03	79.05	85.14	82.43
Pima	(9, 2, 768)	74.35	76.69	73.44	77.60
Sonar	(61, 2, 208)	78.85	78.85	84.62	85.58
Tic-tac-toe	(10, 2, 958)	83.72	93.84	98.54	95.82
Vehicle	(19, 4, 846)	71.04	74.11	78.25	74.94
Waveform	(22, 3, 5000)	76.44	83.30	83.90	84.04
Wine	(14, 3, 178)	94.38	96.07	97.75	97.75
Zoo	(17, 7, 101)	93.07	93.07	95.05	97.03