

START

# 머신러닝과 딥러닝

Machine Learning & deep Learning

# Chapter 7. 비선형분류모형 II

Machine Learning & Deep Learning

손영두

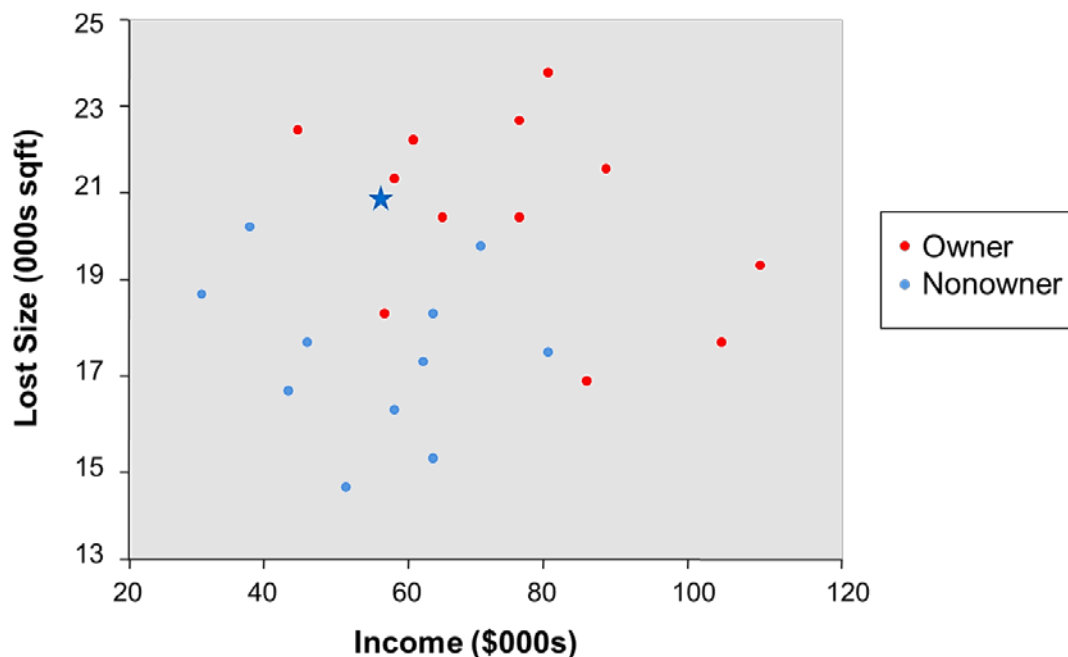
e-mail: [youngdoo@dongguk.edu](mailto:youngdoo@dongguk.edu)



## k-근접 이웃 (nearest neighbor) 분류

### ✓ k-근접 이웃 (k-nearest neighbor, k-NN) 분류

- 분류하고자하는 새로운 데이터와 가장 가까운 k개의 학습 데이터를 이용하여 새로운 데이터의 클래스를 분류
- 가장 가까운 k개의 학습 데이터들이 주로 속한 클래스로 새로운 데이터를 분류: majority voting
- “가까움”은 데이터들의 입력변수를 이용하여 결정





## k-근접 이웃 (nearest neighbor) 분류

### ✓ “가까움”의 정의: 연속 값의 경우

- 유클리디안 거리: 가장 일반적인 형태

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

- 마할라노비스 거리

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^N \frac{(x_i - y_i)^2}{s_i^2}}$$

- 코사인 유사도

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}.$$



### k-근접 이웃 (nearest neighbor) 분류

✓ “가까움”의 정의: 연속 값의 경우

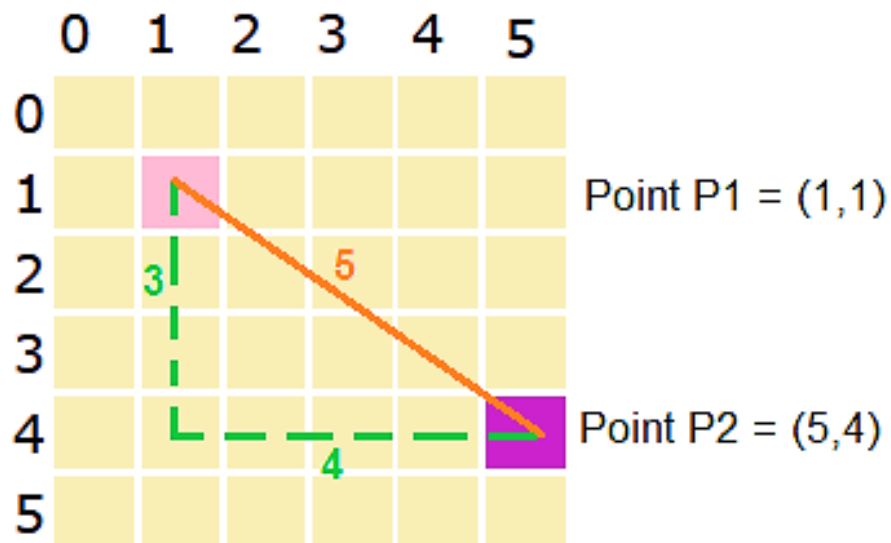
#### ■ 민코프스키 ( $L_p$ ) 거리

$$D(X, Y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- $p=1$  일 경우, Manhattan distance로 불림
- $p=\infty$  ?  $p=0$ ?



## k-근접 이웃 (nearest neighbor) 분류

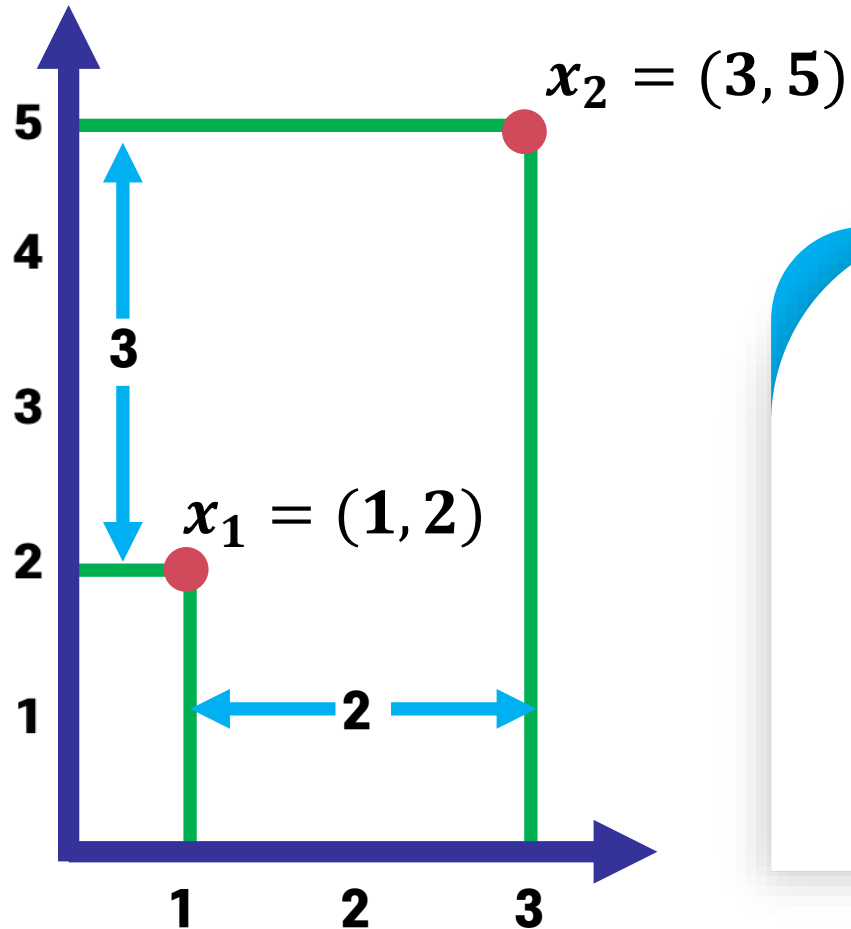


$$\text{Euclidean distance} = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$\text{Manhattan distance} = |5-1| + |4-1| = 7$$



## k-근접 이웃 (nearest neighbor) 분류



**Euclidean distance**  
 $= (2^2 + 3^2)^{1/2} = 3.61$

**Manhattan distance**  
 $= 2 + 3 = 5$

**Supremum distance**  
 $= 5 - 2 = 3$



### k-근접 이웃 (nearest neighbor) 분류

✓ “가까움”의 정의: 이산 값의 경우

■ 해밍 거리: 서로 다른 성분의 수

- "karolin" and "kathrin" is 3.
- "karolin" and "kerstin" is 3.
- 1011101 and 1001001 is 2.
- 2173896 and 2233796 is 3.





## k-근접 이웃 (nearest neighbor) 분류

### ✓ “가까움”의 정의: issues

- 입력 변수 간의 scale이 다른 경우 거리 계산이 하나의 입력 변수에 의해 지배될 수 있기 때문에, 필요에 따라 scaling을 수행
- 고차원 데이터의 경우 차원의 저주로 인하여 유클리디안 거리 계산이 무의미해질 수 있음
- 또한 유클리디언 거리는 비직관적인 결과를 가져다줄 수 있음

1 1 1 1 1 1 1 1 1 1 0

0 1 1 1 1 1 1 1 1 1 1

$$d = 1.4142$$

VS

1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1

$$d = 1.4142$$

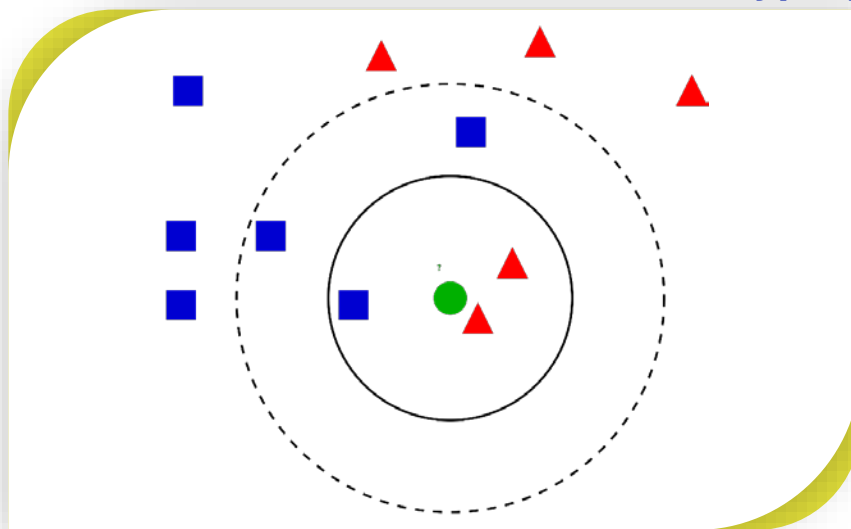
- Scaling 이후?



## k-근접 이웃 (nearest neighbor) 분류

### ✓ k 값의 결정

- k 값은 새로운 클래스를 분류하기 위하여 사용되는 학습 데이터의 수
  - k=1은 분류를 하나의 데이터에 의존하여 수행함을 의미
  - k=5는 5개의 데이터를 참조하여 분류
- 일반적으로 k 값은 validation set의 error를 통하여 결정 (hyperparameter)



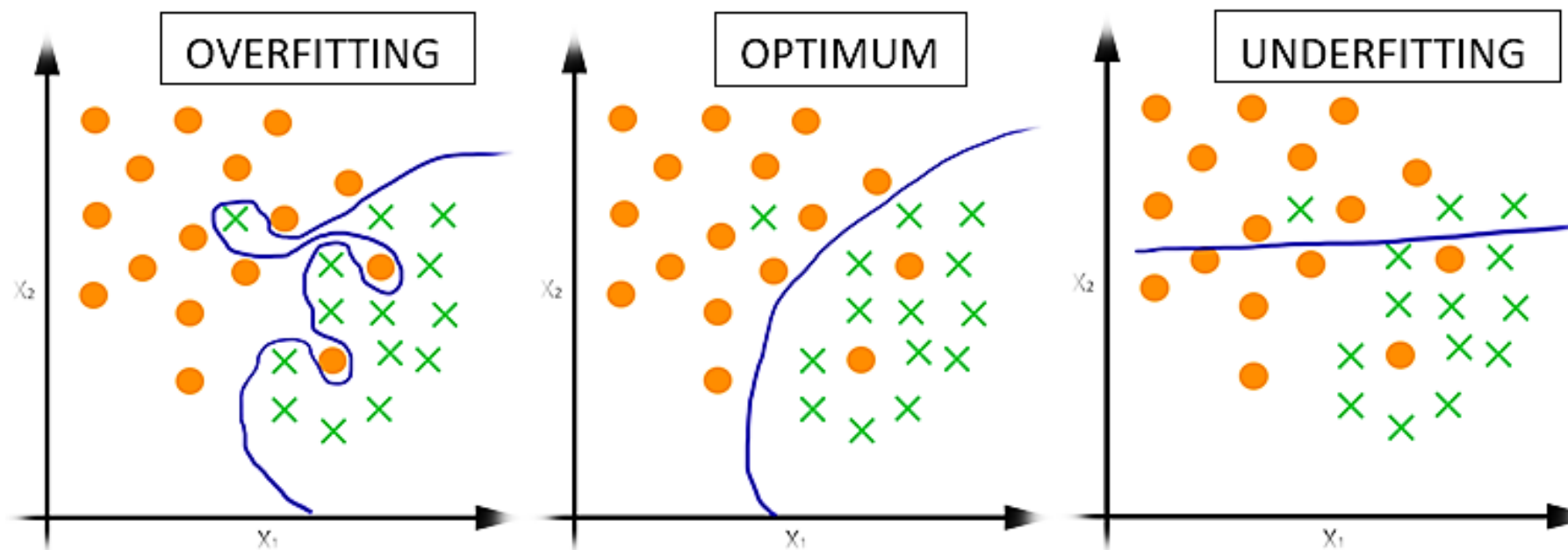
- 일정 수의 근접 이웃이 아닌 일정 거리 안의 근접 이웃을 활용하는 방식 또한 사용 ( $\epsilon$ -nearest neighbor)



## k-근접 이웃 (nearest neighbor) 분류

### ✓ k 값의 trade-off

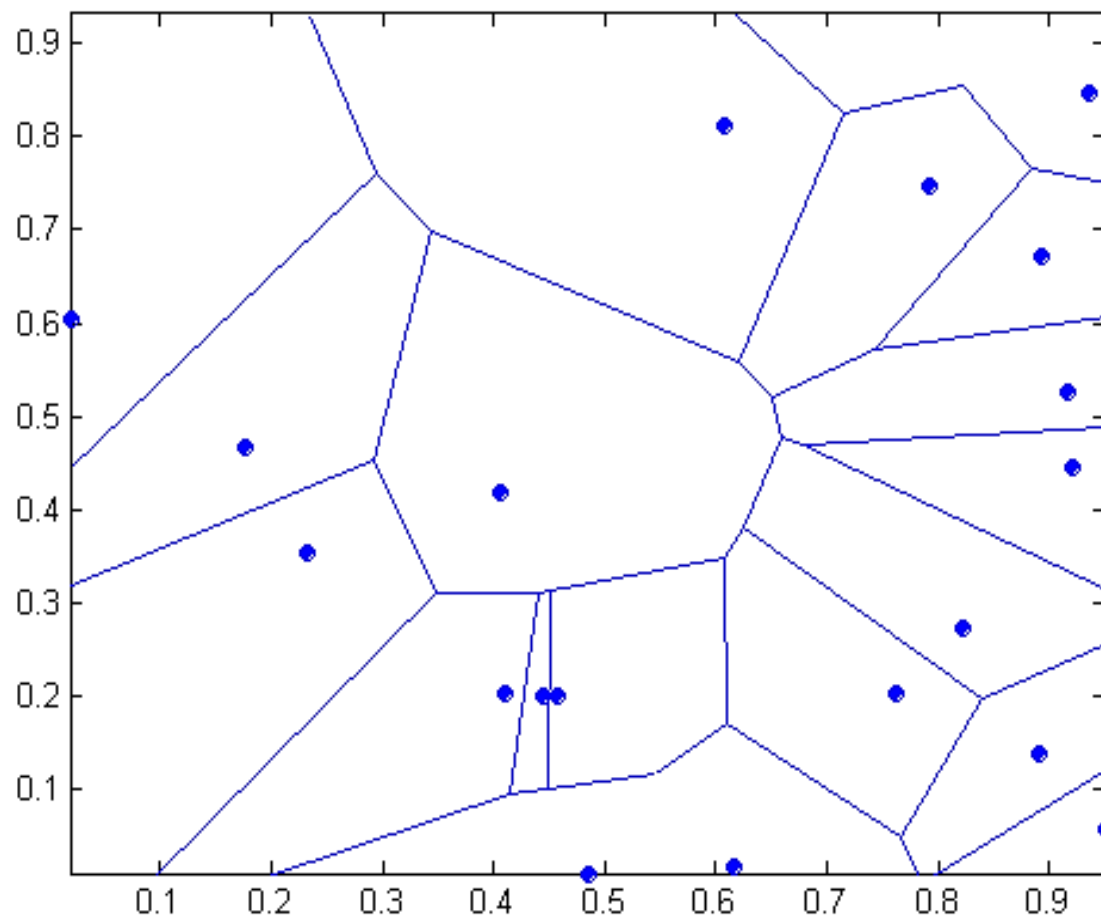
- 지나치게 작은 k 값은 overfitting, 지나치게 큰 k 값은 underfitting을 유발





## k-근접 이웃 (nearest neighbor) 분류

### ✓ 1-NN: Voronoi diagram





### k-근접 이웃 (nearest neighbor) 분류

#### ✓ k-근접 이웃 회귀

- k-근접 이웃 방법을 이용하여 연속 값을 예측하는 회귀분석을 수행할 경우, 근접한 데이터의 값들을 평균으로 사용
- 이 때, 거리에 반비례하는 가중치를 이용한 가중 평균 방법 또한 널리 사용



## k-근접 이웃 (nearest neighbor) 분류

### ✓ 예시

- 다음 데이터를 이용하여 간단히 k-근접 이웃 분류의 원리를 알아보자.

#.	Width	Length	Height	Class
1	3	4	0	a
2	5	2	2	b
3	4	3	3	a
4	6	3	2	b
5	1	2	2	a
6	2	2	3	a
7	3	5	3	b
8	5	3	2	b
9	4	2	3	b
10	3	4	2	a
11	3	4	1	a
12	2	2	5	a
13	5	4	0	b



## k-근접 이웃 (nearest neighbor) 분류

### 예시

- 다음과 같이 train set, validation set, test set 으로 나눈 후 k-근접 이웃 알고리즘으로 분류 예측을 해보겠다.

	#.	Width	Length	Height	Class
Train set	1	3	4	0	a
	2	5	2	2	b
	3	4	3	3	a
	4	6	3	2	b
	5	1	2	2	a
	6	2	2	3	a
	7	3	5	3	b
	8	5	3	2	b
Validation set	9	4	2	3	b
	10	3	4	2	a
Test set	11	3	4	1	a
	12	2	2	5	a
	13	5	4	0	b



## k-근접 이웃 (nearest neighbor) 분류

### 예시

- 먼저 K=1일 때, #.9, #.10 데이터로부터의 유클리디안 거리를 각각 구한 후 가장 가까운 데이터를 확인해보면 공교롭게도 두 데이터 모두 잘 못 예측된다.

#.	Width	Length	Height	Class	Distance to #.9	Distance to #.10
1	3	4	0	a	3.741657	2
2	5	2	2	b	1.414214	2.828427
3	4	3	3	a	1	1.732051
4	6	3	2	b	2.44949	3.162278
5	1	2	2	a	3.162278	2.828427
6	2	2	3	a	2	2.44949
7	3	5	3	b	3.162278	1.414214
8	5	3	2	b	1.732051	2.236068
9	4	2	3	b	-	-
10	3	4	2	a	-	-
11	3	4	1	a		
12	2	2	5	a		
13	5	4	0	b		





## k-근접 이웃 (nearest neighbor) 분류

### 예시

- K=3일 때, 두 데이터 모두 majority voting에 의해 올바르게 예측된다

#.	Width	Length	Height	Class	Distance to #.9	Distance to #.10
1	3	4	0	a	3.741657	2
2	5	2	2	b	1.414214	2.828427
3	4	3	3	a	1	1.732051
4	6	3	2	b	2.44949	3.162278
5	1	2	2	a	3.162278	2.828427
6	2	2	3	a	2	2.44949
7	3	5	3	b	3.162278	1.414214
8	5	3	2	b	1.732051	2.236068
9	4	2	3	b	-	-
10	3	4	2	a	-	-
11	3	4	1	a		
12	2	2	5	a		
13	5	4	0	b		



## k-근접 이웃 (nearest neighbor) 분류

### 예시

- K=3일 때, 두 데이터 모두 majority voting에 의해 올바르게 예측된다.

#.	Width	Length	Height	Class	Distance to #.9	Distance to #.10
1	3	4	0	a	3.741657	2
2	5	2	2	b	1.414214	2.828427
3	4	3	3	a	1	1.732051
4	6	3	2	b	2.44949	3.162278
5	1	2	2	a	3.162278	2.828427
6	2	2	3	a	2	2.44949
7	3	5	3	b	3.162278	1.414214
8	5	3	2	b	1.732051	2.236068
9	4	2	3	b	-	-
10	3	4	2	a	-	-
11	3	4	1	a		
12	2	2	5	a		
13	5	4	0	b		



## k-근접 이웃 (nearest neighbor) 분류

### 예시

- K값을 3으로 정한 후 test set을 분류하면 테스트 데이터 모두 majority voting에 의해 올바르게 예측되는 것을 확인할 수 있다.

#.	Width	Length	Height	Class	Distance to #.11	Distance to #.12	Distance to #.13
1	3	4	0	a	1	5.477226	2
2	5	2	2	b	3	4.242641	2.828427
3	4	3	3	a	2.44949	3	3.316625
4	6	3	2	b	3.316625	5.09902	2.44949
5	1	2	2	a	3	3.162278	4.898979
6	2	2	3	a	3	2	4.690416
7	3	5	3	b	2.236068	3.741657	3.741657
8	5	3	2	b	2.44949	4.358899	2.236068
9	4	2	3	b	3	2.828427	3.741657
10	3	4	2	a	1	3.741657	2.828427
11	3	4	1	a			
12	2	2	5	a			
13	5	4	0	b			



## k-근접 이웃 (nearest neighbor) 분류



### 예시

- K값을 3으로 정한 후 test set을 분류하면 테스트 데이터 모두 majority voting에 의해 올바르게 예측되는 것을 확인할 수 있다.

#.	Width	Length	Height	Class	Distance to #.11	Distance to #.12	Distance to #.13
1	3	4	0	a	1	5.477226	2
2	5	2	2	b	3	4.242641	2.828427
3	4	3	3	a	2.44949	3	3.316625
4	6	3	2	b	3.316625	5.09902	2.44949
5	1	2	2	a	3	3.162278	4.898979
6	2	2	3	a	3	2	4.690416
7	3	5	3	b	2.236068	3.741657	3.741657
8	5	3	2	b	2.44949	4.358899	2.236068
9	4	2	3	b	3	2.828427	3.741657
10	3	4	2	a	1	3.741657	2.828427
11	3	4	1	a			
12	2	2	5	a			
13	5	4	0	b			



## k-근접 이웃 (nearest neighbor) 분류



### 예시

- K값을 3으로 정한 후 test set을 분류하면 테스트 데이터 모두 majority voting에 의해 올바르게 예측되는 것을 확인할 수 있다.

#.	Width	Length	Height	Class	Distance to #.11	Distance to #.12	Distance to #.13
1	3	4	0	a	1	5.477226	2
2	5	2	2	b	3	4.242641	2.828427
3	4	3	3	a	2.44949	3	3.316625
4	6	3	2	b	3.316625	5.09902	2.44949
5	1	2	2	a	3	3.162278	4.898979
6	2	2	3	a	3	2	4.690416
7	3	5	3	b	2.236068	3.741657	3.741657
8	5	3	2	b	2.44949	4.358899	2.236068
9	4	2	3	b	3	2.828427	3.741657
10	3	4	2	a	1	3.741657	2.828427
11	3	4	1	a			
12	2	2	5	a			
13	5	4	0	b			



### k-근접 이웃 (nearest neighbor) 분류

#### ✓ 장점

- 단순하고 직관적임
- 확률 분포 등 특별한 과정 불필요
- 복잡한 형태의 데이터에서 잘 작동

#### ✓ 단점

- 테스트 데이터마다 거리 계산이 필요하여 분류 속도가 느림: lazy learner
- Scaling, distance measure 등 사용자의 선택이 필요



### 앙상블 (Ensemble Learning)

- ✓ 현재까지 하나의 가설을 이용하여 데이터를 분류하는 방법에 대하여 알아봄  
(LDA, Naïve Bayes, k-NN, Decision Tree)
- ✓ 앙상블 학습 -> 여러 가설의 집합을 선택하고, 그들의 예측 결과를 결합하는 방법
- ✓ Example 1 – 같은 혹은 서로 다른 학습 데이터에 대하여 100 개의 decision tree를 만들고 그들의 결과를 투표하여 새로운 데이터에 대한 분류를 수행하는 방식  
(Random forest)
- ✓ Key motivation: 분류 오차를 감소시킬 수 있음. 앙상블 모델은 여러 분류기를 동시에 사용함에 따라 오분류를 수행하지 않을 것으로 기대.



### 앙상블 (Ensemble Learning)

✓ **다음과 같은 방법을 이용하여 서로 다른 분류기(가설)을 학습**

1. 서로 다른 학습 데이터를 사용 (다른 instance, 다른 feature)
2. 서로 다른 학습 알고리즘을 사용

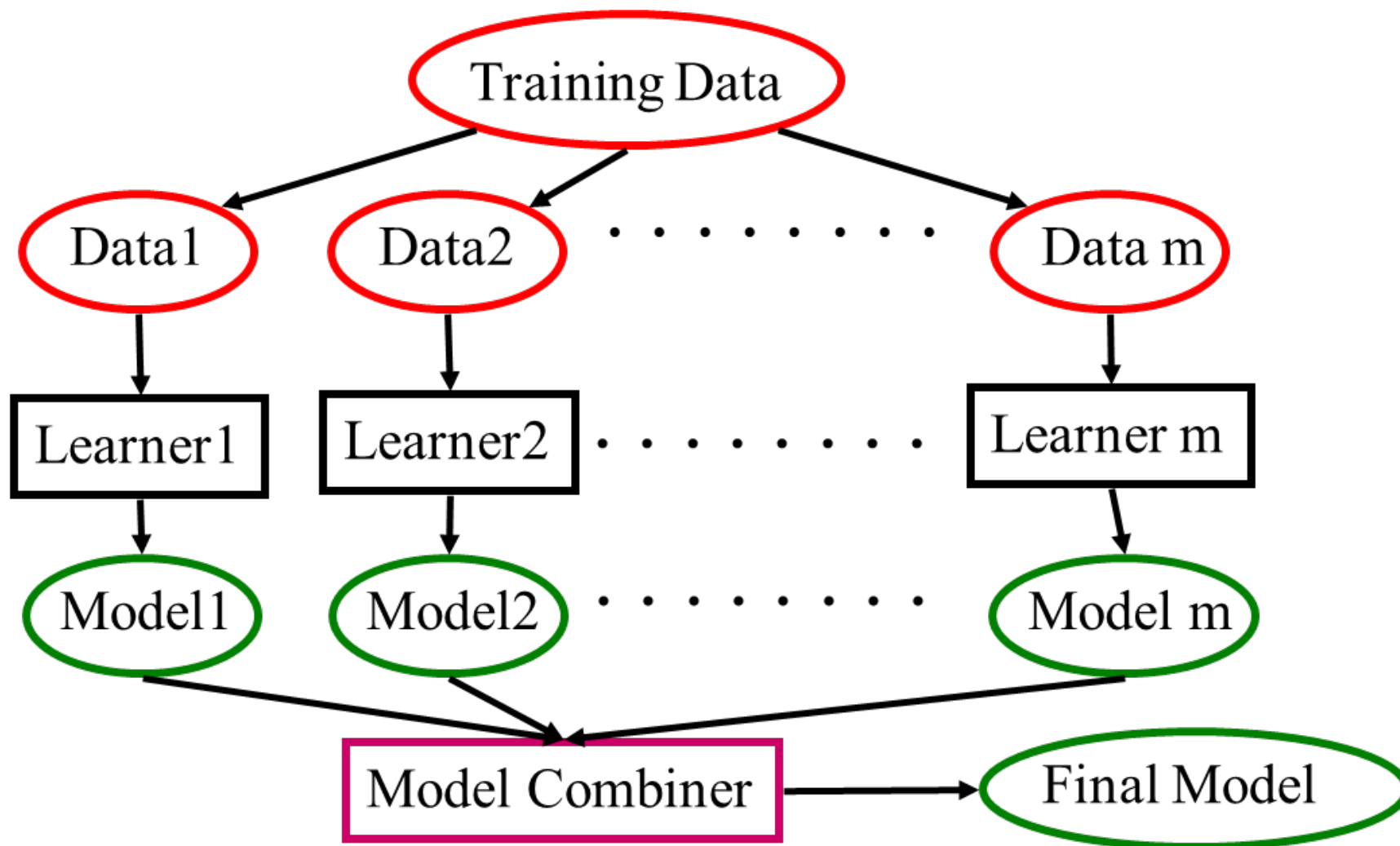
✓ **서로 다른 분류기로부터 발생한 결과들을 조합**

e.g. majority voting, weighted voting





## 앙상블 (Ensemble Learning)





### 앙상블 (Ensemble Learning)

#### ✓ No Free Lunch” Theorem

- 어떤 하나의 알고리즘도 모든 경우에 좋을 수 없다!


















































- ✓ 여러 독립적이고 다양한 결과들을 조합할 경우, 각 개별 결과들이 최소한 random guessing보다 우수하다면, 무작위 오차들이 서로 상쇄되어 좋은 결과를 가져올 수 있음

#### ✓ Examples: Human ensembles are demonstrably better

- How many jelly beans in the jar?: Individual estimates vs. group average.
- Who Wants to be a Millionaire: Audience vote.



## 앙상블 (Ensemble Learning): 예시

Reality							
1							
2							
3							
4							
5							
Combine							



### 앙상블 (Ensemble Learning)

#### ✓ Majority vote

#### ✓ 5개의 완전히 독립적인 분류기가 있다고 가정하면,

##### ■ 만일 각각의 정확도가 70%라면,

- $(.75)+5(.74)(.3)+ 10 (.73)(.32)$
- 83.7% majority vote accuracy

##### ■ 이러한 분류기 101개를 조합하면,

- 99.9% majority vote accuracy



### 앙상블 (Ensemble Learning): Homogeneous Ensemble

✓ 한 종류의 학습 방법을 사용하지만, 학습 데이터를 조작하여 다양한 모델을 학습

- $\text{Data1} \neq \text{Data2} \neq \dots \neq \text{Data } m$
- $\text{Learner1} = \text{Learner2} = \dots = \text{Learner } m$

✓ 학습 데이터의 변형에는 여러 방법이 사용될 수 있음

- Bagging: Resample training data
- Boosting: Reweight training data



### 앙상블 (Ensemble Learning): Homogeneous Ensemble

✓ 서로 다른 학습  
알고리즘을 사용

✓ 같은  
알고리즘이지만  
서로 다른  
모수를 선택

✓ 서로 다른 종류의  
입력 변수를 사용



## Bagging

✓ “bootstrap aggregation”을 이용하여 앙상블 모델을 학습

- “bootstrap aggregation”: repeatedly randomly resampling the training data (Brieman, 1996).

✓ Bootstrap: 학습 데이터 X로부터 N 개의 sample을 복원 추출

✓ Bagging

- M개의 모델을 M 개의 bootstrap sample로부터 학습
- Combine outputs by voting (e.g., majority vote)

✓ Unstable learner들의 분산을 감소시킴으로 인하여 전체적인 오차를 감소

- Unstable learner: 학습 데이터에서의 작은 변화가 결과에 큰 변화를 야기하는 학습 알고리즘 (decision tree, artificial neural networks, etc.)

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



### Bagging - Algorithm

✓ Given a standard training set  $D$  of size  $n$

✓ For  $i = 1 \dots M$

- Draw a sample of size  $n^* < n$  from  $D$  uniformly and with replacement
- Learn classifier  $C_i$

✓ Final classifier is a vote of  $C_1 \dots C_M$

✓ Increases classifier stability/reduces variance





## Boosting

### ✓ Strong Learner -> Objective of machine learning

- 레이블이 있는 데이터를 이용하여 학습
- 원하는 수준의 정확도를 가지는 모델을 생성

### ✓ Weak Learner

- 레이블이 있는 데이터를 이용하여 학습
- 무작위 선택보다 더 우수한 정확도의 모델을 생성



### Boosting

✓ Weak Learner: 학습 정확도가 0.5 이상인 분류 가설만 생성해내면 달성 가능,  
i.e., <50% error over any distribution

✓ Learners

- Strong learner는 구축이 매우 어려움
- Weak learner의 구축은 상대적으로 쉬움

✓ Question: Weak learner의 집합을 이용하여 strong learner를 만들 수 있을까?



✓ Weak learner의 학습 성능을 개선을 보장하기 위한 학습 이론적인 연구로부터 출발  
(Schapire, 1990)

✓ 후에 앙상블 모델을 이용하여 일반화 성능을 개선하는 현실적인 알고리즘을 개선됨  
(AdaBoost, Freund & Shapire, 1996)

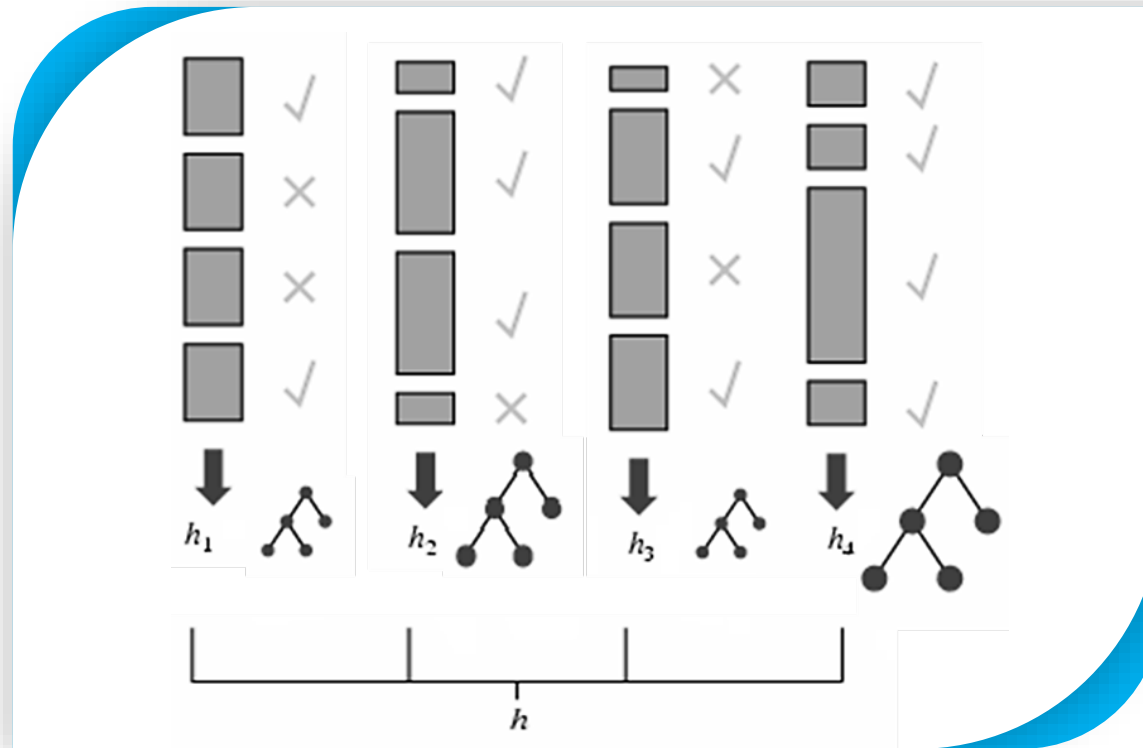
### ✓ Key Insights

- 샘플링 대신 (as in bagging) 데이터들에 가중치를 수정!
- 데이터들은 각각 가중치를 가지고 있으며, 매 iteration 마다 새로운 가설이 해당 가중치를 이용하여 학습됨. 그리고 현재 가설에서 오분류된 데이터가 다음 가설에서는 많이 반영될 수 있도록, 데이터들에 새로운 가중치를 부여.
- 최종 분류기는 weak classifier들의 weighted vote



## Adaptive Boosting

- ✓ 각 사각형은 데이터를 나타내며, 각 데이터의 가중치는 높이에 비례
- ✓ X 표시는 오분류 된 데이터를 의미
- ✓ 분류기의 크기는 최종 모델에 반영되는 가중치를 의미





# Boosting – Construct Weak Learners

### ☑ 서로 다른 데이터의 분포를 사용

- 동일한 가중치로 시작
- 학습 단계 동안:
  - Weak learner에 의해 오분류된 데이터의 가중치를 증가
  - Weak learner에 의해 정분류된 데이터의 가중치를 감소

### ☑ Idea

- 이전 단계에서 제대로 분류하지 못한 “어려운” 데이터들에게 집중



# Boosting – Construct Weak Learners

## ✓ Weighted Voting

- Weak classifier의 weighted voting을 통하여 strong classifier를 학습

## ✓ Idea

- 좋은 weak classifier가 높은 가중치를 부여받음
- 반복적으로 weak classifier를 추가
  - 조합된 분류기의 성능을 지속적으로 향상



## Adaptive Boosting - Algorithm

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ .

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

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

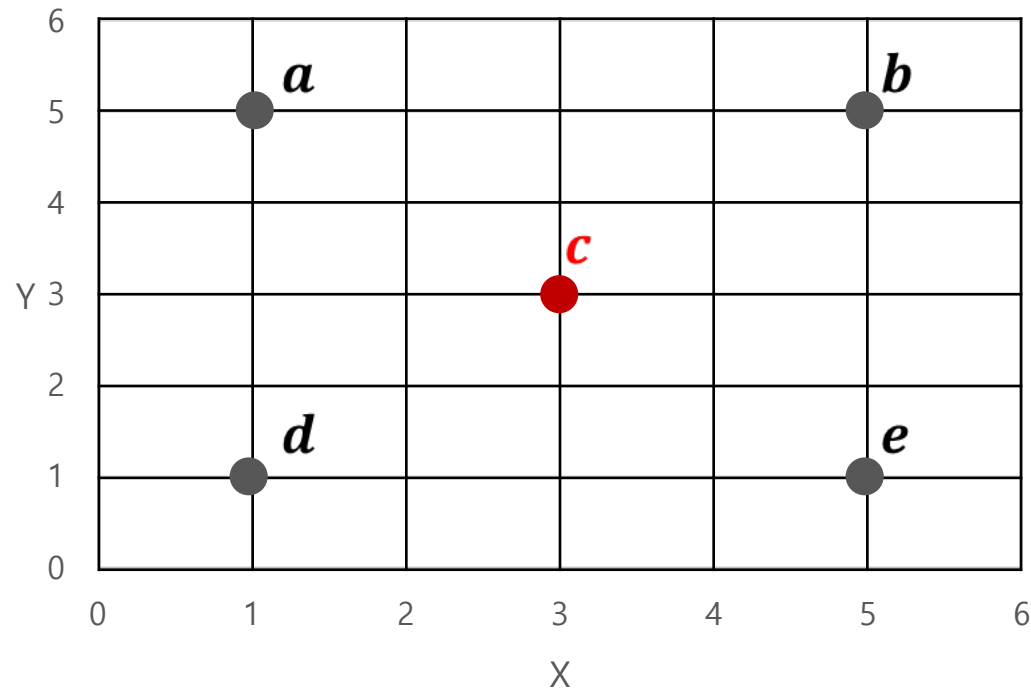
**Fig. 1** The boosting algorithm AdaBoost.



## Adaptive Boosting - Example

☑ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자.

■ 동일한 가중치로 시작



$w_a$	1/5
$w_b$	1/5
$w_c$	1/5
$w_d$	1/5
$w_e$	1/5

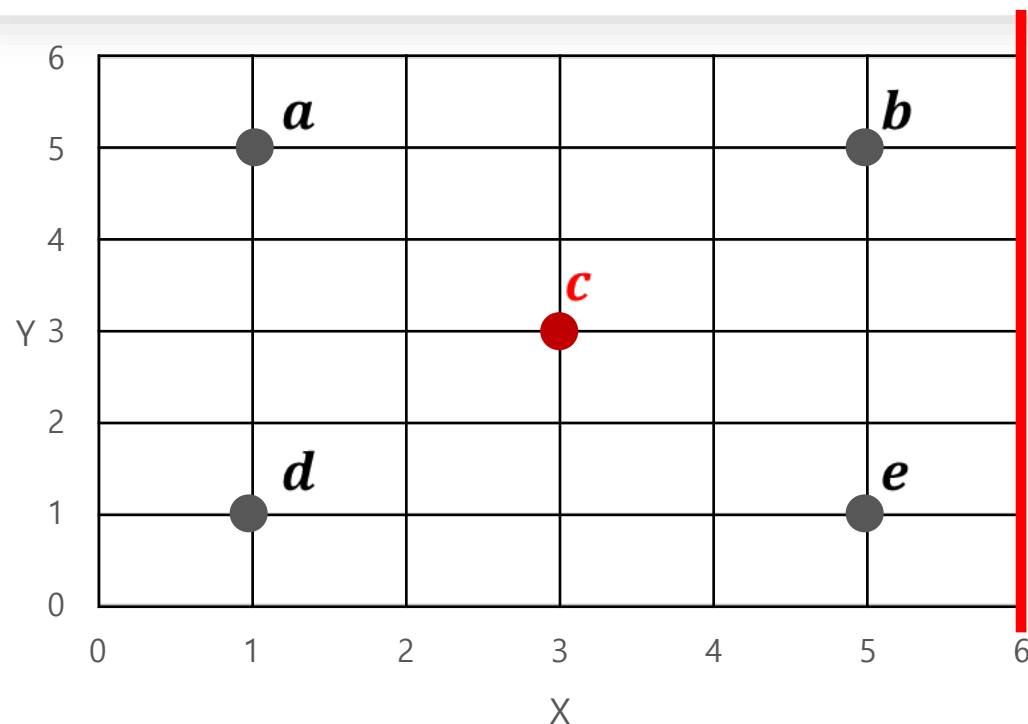




## Adaptive Boosting - Example

✓ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

- 학습 단계 동안:
  - Weak learner에 의해 오분류 된 데이터의 가중치를 증가
  - Weak learner에 의해 정분류 된 데이터의 가중치를 감소



$$h_1(X): x = 6$$

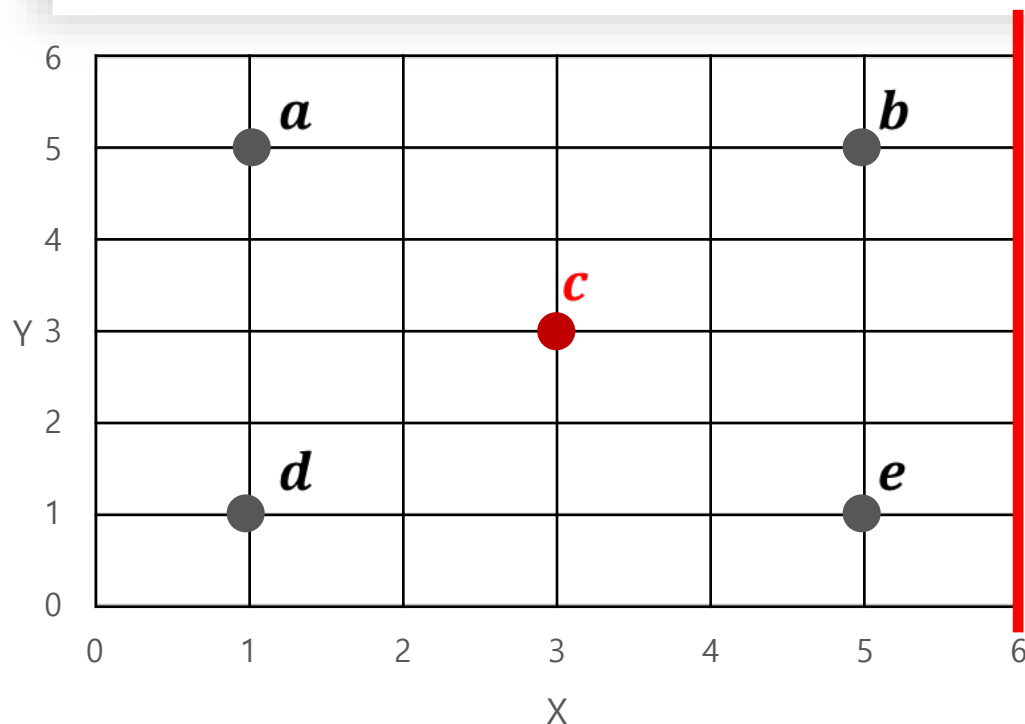
$w_a$	1/5
$w_b$	1/5
$w_c$	1/5
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

✓ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

- 첫 기준선이  $x=6$ 라고 하면, c만 틀렸으므로 ( $x < 6$  인 점을 모두 같은 색의 점으로 판단) c의 가중치가 증가하고 나머지 가중치는 감소한다.



$$h_1(X): x = 6$$

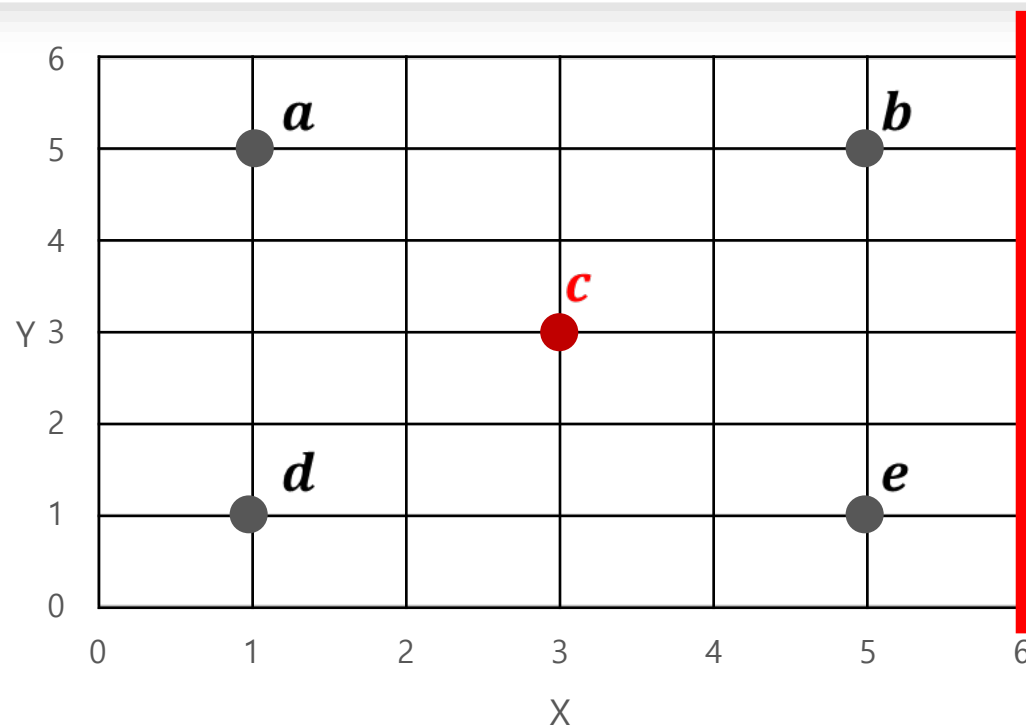
$w_a$	1/5
$w_b$	1/5
$w_c$	1/2
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

☑ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

■  $h_1(X)$ 의 error, 즉  $\varepsilon_1 = \frac{1}{5}$ 이었으므로  $h_1(X)$ 가 가지는 voting power는  $\alpha_1 = \frac{1}{2} \ln \frac{1-\varepsilon_1}{\varepsilon_1} = \frac{1}{2} \ln 4$  이다.



$h_1(X): x = 6$

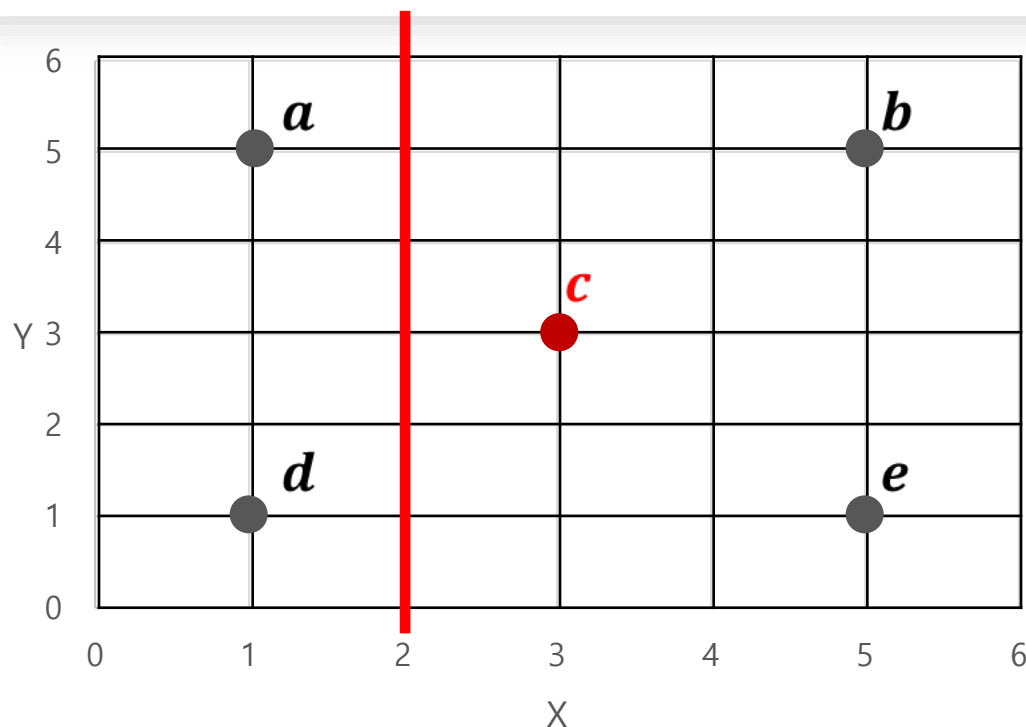
$w_a$	1/5
$w_b$	1/5
$w_c$	1/2
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

✓ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

- 이제 c점에 많은 비중을 두어 확실히 나눌 수 있는 기준선을 찾는다. 여기서는 c를 검은색 점과 확실히 나누는 선 중  $x=2$ 를  $h_2(X)$ 로 결정했다. 그러나 b와e에서 에러가 발생한다. 다시 가중치가 업데이트된다.



$$h_2(X): x = 2$$

$w_a$	1/5
$w_b$	1/5
$w_c$	1/3
$w_d$	1/5
$w_e$	1/5

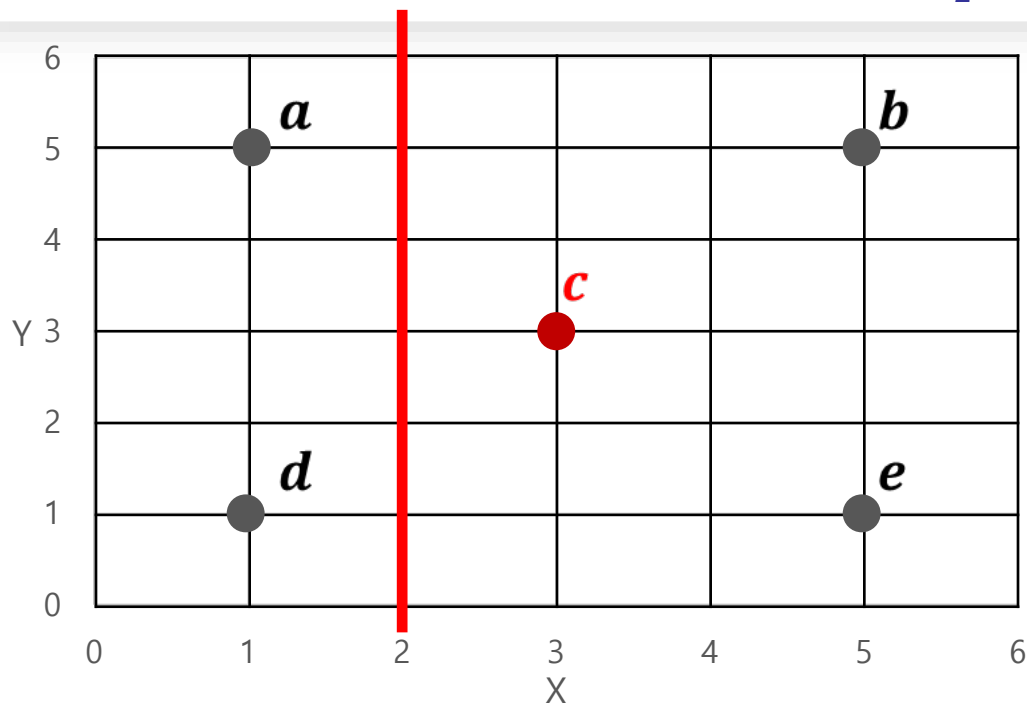


## Adaptive Boosting - Example

- ☑️ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

■  $h_2(X)$ 의 error, 즉  $\varepsilon_2 = \frac{2}{8}$ 이었으므로 (여기서 틀린 b와e의 기존 가중치는 둘 다  $\frac{1}{8}$ ),

$h_2(X)$ 가 가지는 voting power는  $\alpha_2 = \frac{1}{2} \ln \frac{1-\varepsilon_2}{\varepsilon_2} = \frac{1}{2} \ln 3$  이다.



$h_2(X): x = 2$

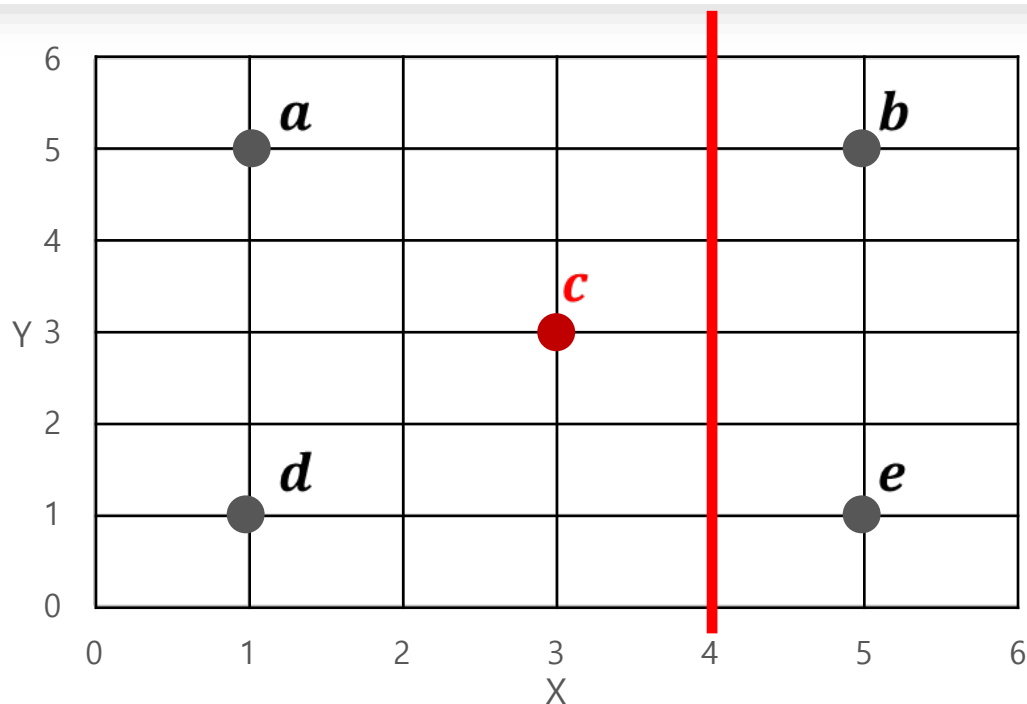
$w_a$	1/5
$w_b$	1/5
$w_c$	1/3
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

☑️ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

- 여전히 c점에 많은 비중을 두어 확실히 나눌 수 있는 기준선을 찾는다. 여기서는 c를 검은색 점과 확실히 나누는 선 중  $x=4$ 를  $h_3(X)$ 로 결정했다. 그러나 a와d에서 에러가 발생한다. 다시 가중치가 업데이트된다.



$$h_3(X): x = 4$$

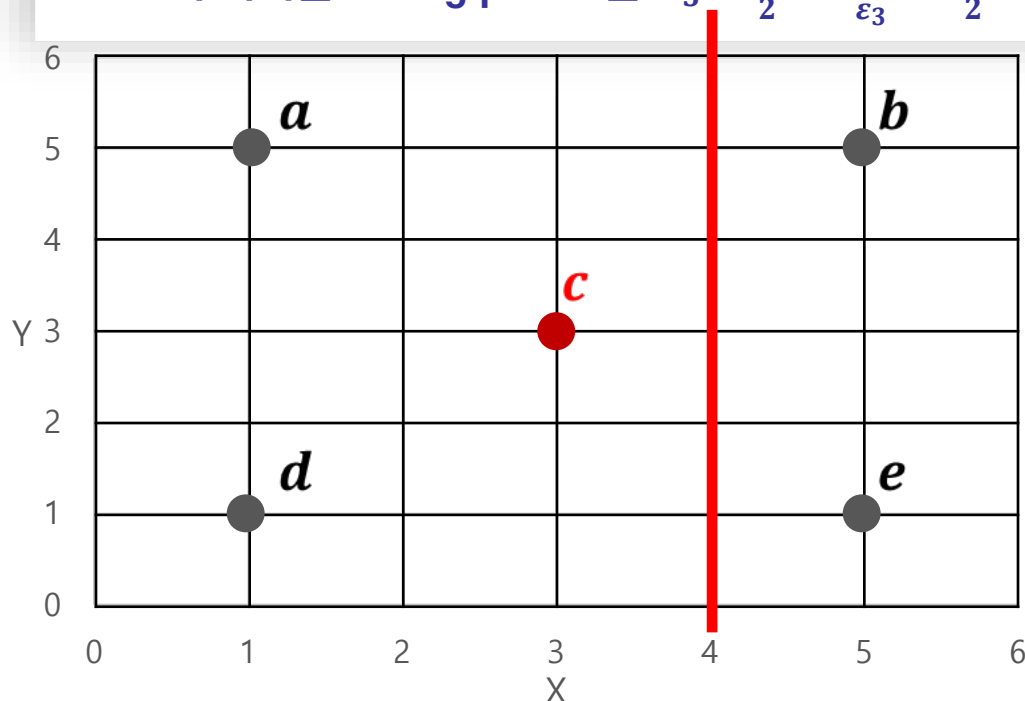
$w_a$	1/5
$w_b$	1/5
$w_c$	1/5
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

☑ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

- $h_3(X)$ 의 error, 즉  $\varepsilon_3 = \frac{2}{12}$  이었으므로 (여기서 틀린 a와d의 기존 가중치는 둘 다  $\frac{1}{12}$ ),  $h_3(X)$ 가 가지는 voting power는  $\alpha_3 = \frac{1}{2} \ln \frac{1-\varepsilon_3}{\varepsilon_3} = \frac{1}{2} \ln 5$  이다.



$h_3(X): x = 4$

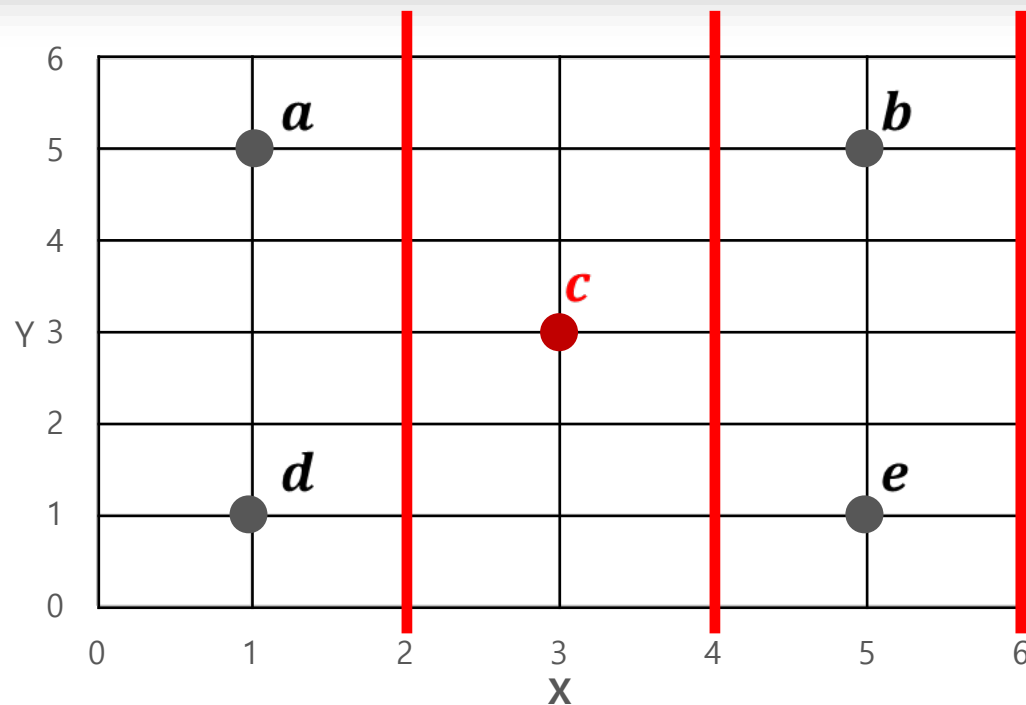
$w_a$	1/5
$w_b$	1/5
$w_c$	1/5
$w_d$	1/5
$w_e$	1/5



## Adaptive Boosting - Example

✓ 다음의 간단한 데이터에서 검은색 점과 빨간색 점을 분류하는 AdaBoost 모델의 원리를 알아보자. (계속)

■ 종합하면,  $H(X) = \text{sign}\left(\frac{1}{2} \ln 4 h_1(X) + \frac{1}{2} \ln 3 h_2(X) + \frac{1}{2} \ln 5 h_3(X)\right)$







## Random Forests

### ✓ Bagging of Decision Trees

### ✓ Algorithm

#### ■ Repeat k times

- 1) Draw with replacement  $N$  examples, put in train set (Bagging)
- 2) Build d-tree, but in each recursive call
  - Choose (w/o replacement)  $m$  features
  - Choose best of these  $m$  as the root of this (sub)tree
- 3) Do NOT prune

Let  $N = \#$  of examples  
 $M = \#$  of features  
 $m = \text{some number} \ll M$



## Random Forests

### ✓ Training Data

N examples

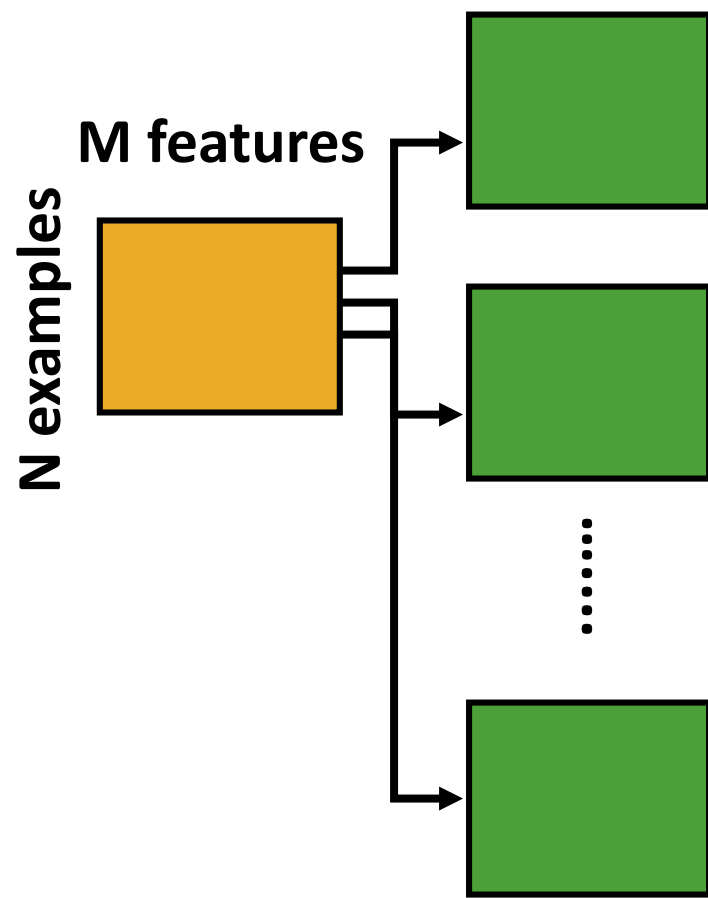
M features





## Random Forests

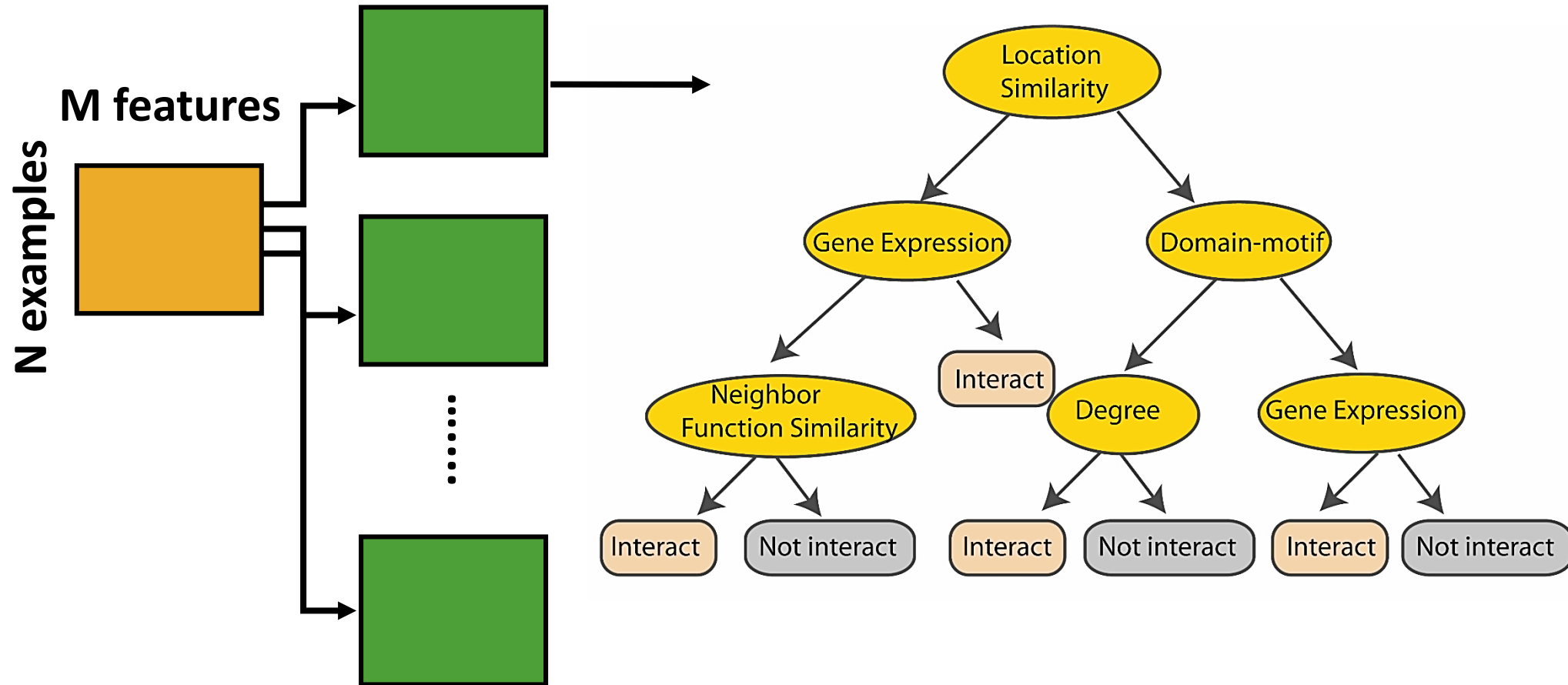
- ✓ Create bootstrap samples from the training data





## Random Forests

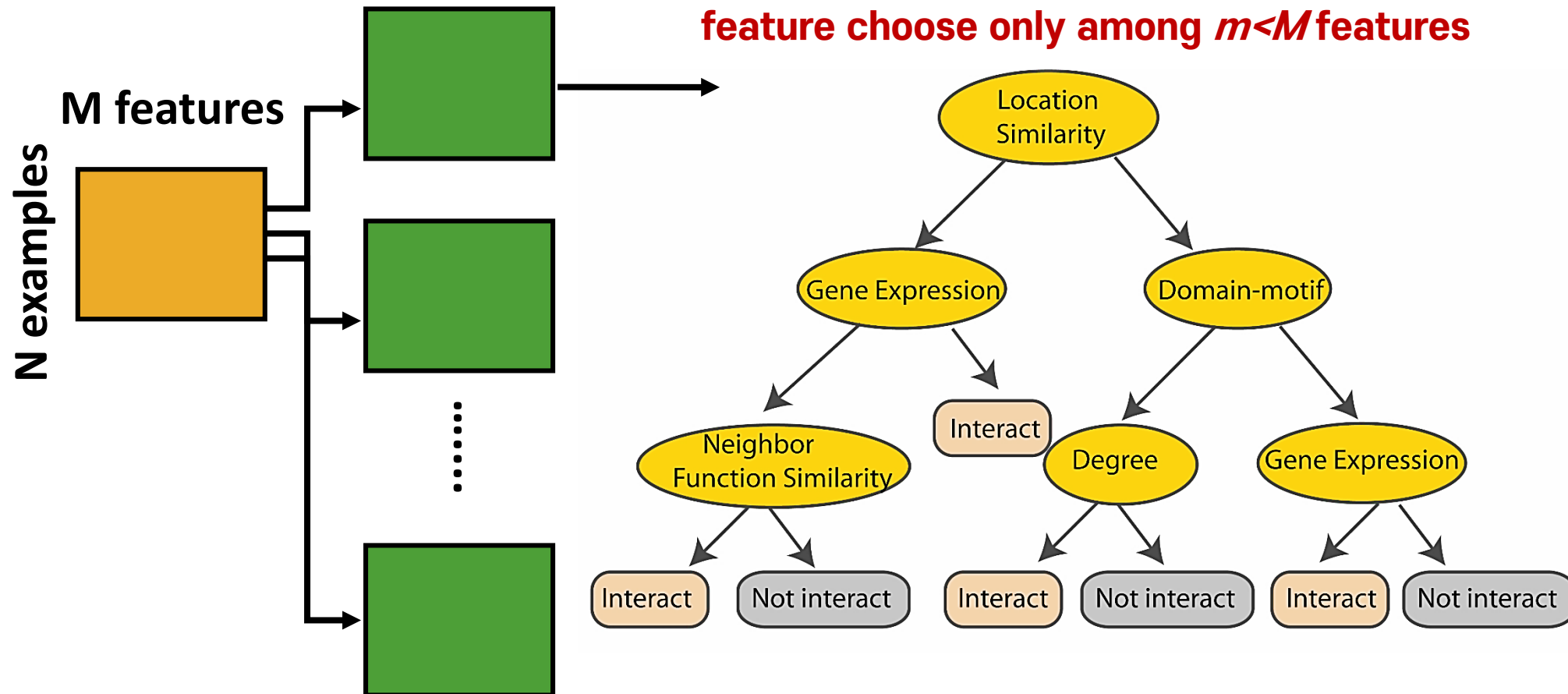
✓ Construct a decision tree





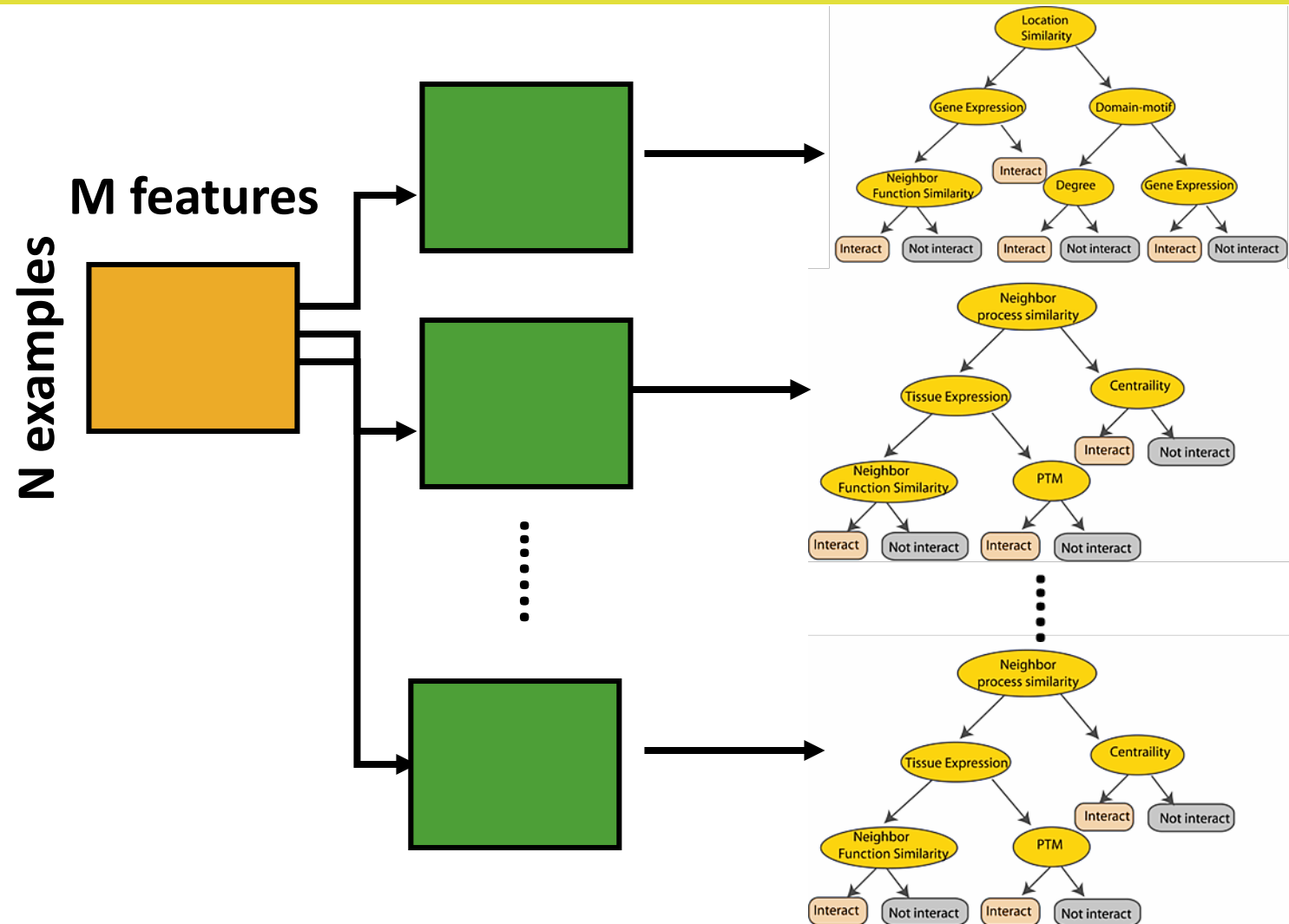
## Random Forests

✓ At each node in choosing the split  
feature choose only among  $m < M$  features



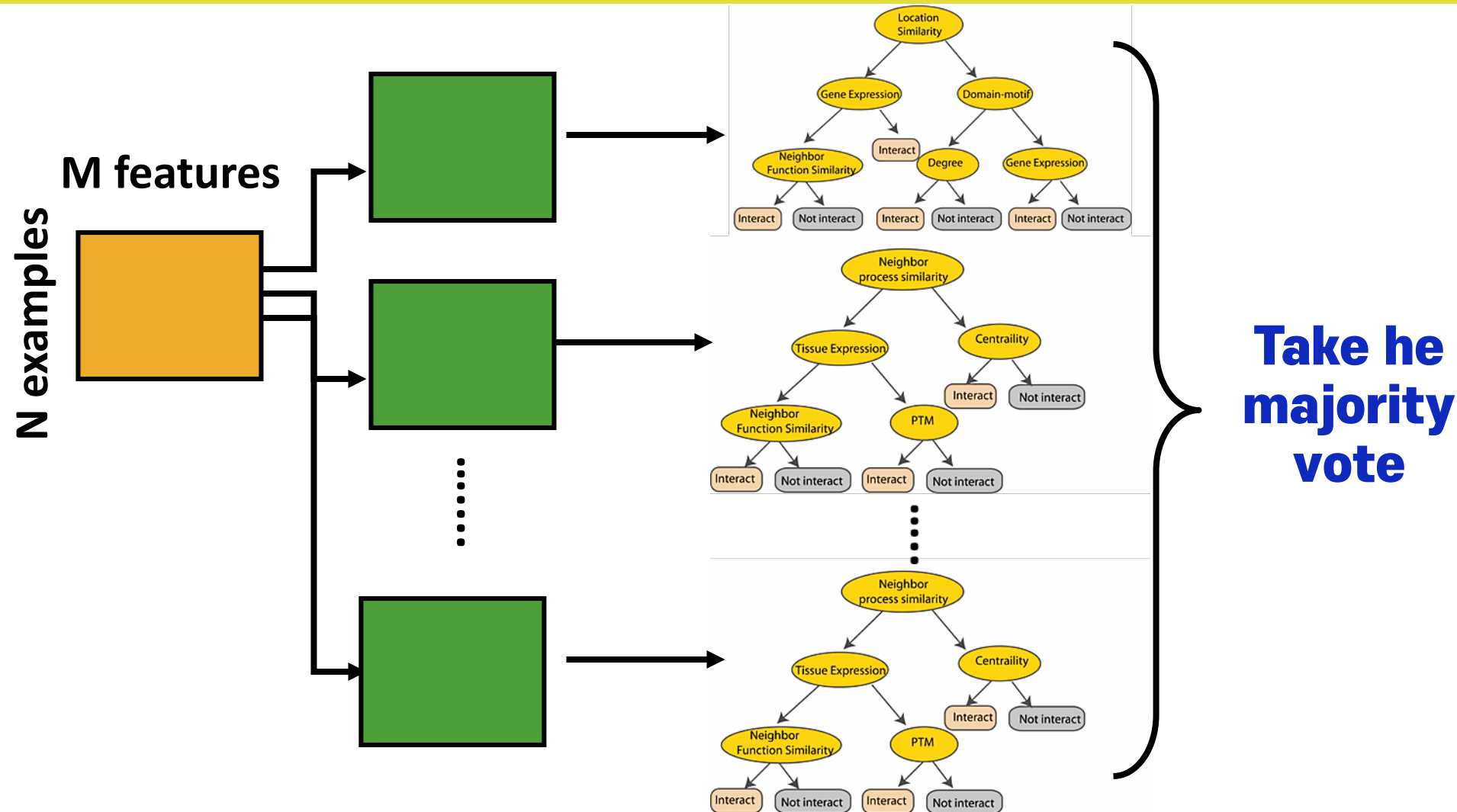


## Random Forests





## Random Forests





# Random Forests

### ✓ Increasing $m$

- Decision tree들 간의 correlation 증가 (BAD)
- 각 tree의 정확도 증가 (GOOD)

### ✓ Tuning set을 이용하여 적절한 $m$ 을 결정 가능

### ✓ Random forest는

- 빠른 속도
- 많은 수의 feature를 처리 가능
- Overfitting을 방지
- 실제 문제에서 잘 작동





## Ensemble Learning: Netflix Example



Users rate movies (1,2,3,4,5 stars);  
Netflix makes suggestions to users based on previous rated movies.



## Ensemble Learning: Netflix Example



*“The Netflix Prize seeks to substantially improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences. Improve it enough and you win one (or more) Prizes. Winning the Netflix Prize improves our ability to connect people to the movies they love.”*



# Ensemble Learning: Netflix Example

## ✓ Supervised learning task

- Training data is a set of users and ratings (1,2,3,4,5 stars) those users have given to movies.
- Construct a classifier that given a user and an unrated movie, correctly classifies that movie as either 1, 2, 3, 4, or 5 stars

✓ **\$1 million prize for a 10% improvement over Netflix's current movie recommender/classifier (MSE = 0.9514)**



## Ensemble Learning: Netflix Example

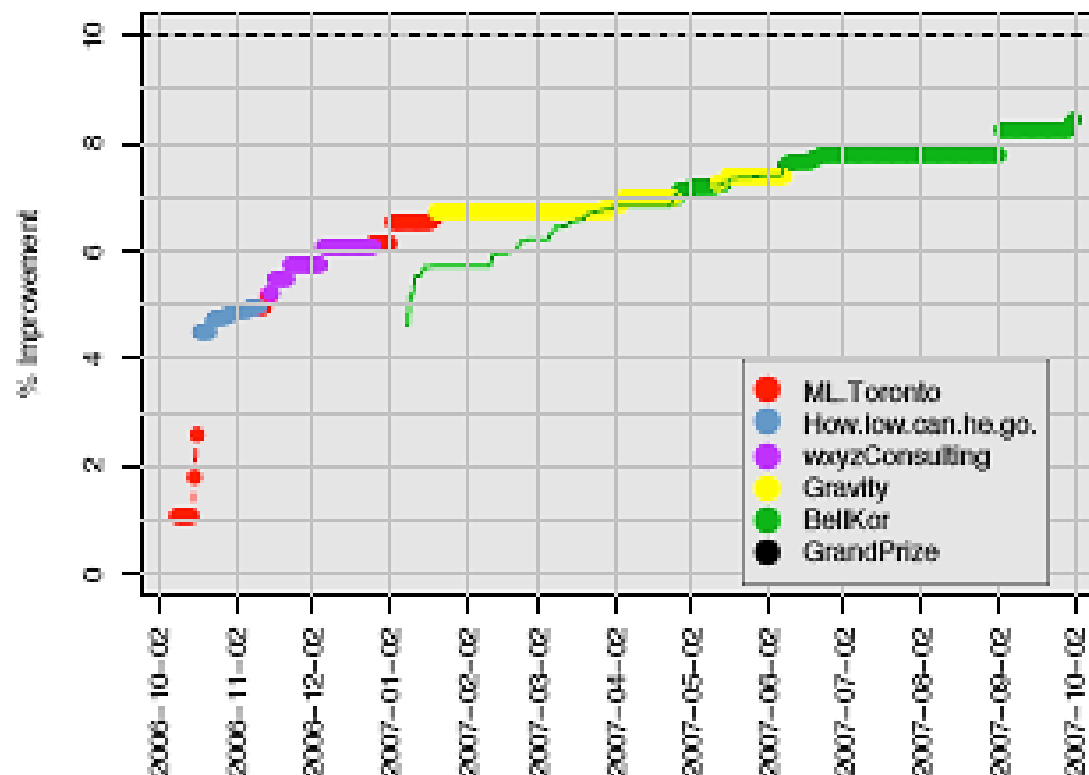
### ✓ BellKor / KorBell

- Scores of the leading team for the first 12 months of the Netflix Prize.
- Colors indicate when a given team had the lead. The % improvement is over Netflix' Cinematch algorithm.

### ✓ The million dollar

- Grand Prize level is shown as a dotted line at 10% improvement.

### ■ Top contenders for Progress Prize 2007





# Ensemble Learning: Netflix Example

## The BellKor solution to the Netflix Prize

Robert M. Bell, Yehuda Koren and Chris Volinsky  
AT&T Labs – Research  
BellKor@research.att.com

**“Our final solution  
(RMSE=0.8712)  
consists of  
blending  
107 individual results.”**

Our final solution (RMSE=0.8712) consists of blending 107 individual results. Since many of these results are close variants, we first describe the main approaches behind them. Then, we will move to describing each individual result.

The core components of the solution are published in our ICDM'2007 paper [1] (or, KDD-Cup'2007 paper [2]), and also in the earlier KDD'2007 paper [3]. We assume that the reader is familiar with these works and our terminology there.

### Neighborhood-based model (k-NN)

A movie-oriented k-NN approach was thoroughly described in our KDD-Cup'2007 paper [kNN]. We apply it as a post-processor for most other models. Interestingly, it was most effective when applied on residuals of RBMs [5], thereby driving the Quiz RMSE from 0.9093 to 0.8888.

An earlier k-NN approach was described in the KDD'2007 paper ([3], Sec. 3) [Slow-kNN]. It appears that this earlier approach can achieve slightly more accurate results than the newer one, at the expense of a significant increase in running time. Consequently, we dropped the older approach, though some results involving it survive within the final blend.

We also tried more naïve k-NN models, where interpolation weights are based on pairwise similarities between movies (see [2], Sec. 2.2). Specifically, we based weights on  $corr^2/(1-corr^2)$  [Corr-kNN], or on  $mse^{-10}$  [MSE-kNN]. Here,  $corr$  is the Pearson correlation coefficient between the two respective movies, and  $mse$  is the mean squared distance between two movies (see definition of  $s_{ij}$  in Sec. 4.1 of [2]). We also tried taking the interpolation weights as the “support-based similarities”, which will be defined shortly [Supp-kNN].



## 실습 – MNIST dataset 로드

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
mnist

{'COL_NAMES': ['label', 'data'],
 'DESCR': 'mldata.org dataset: mnist-original',
 'data': array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
 'target': array([0., 0., 0., ..., 9., 9., 9.])}
```

```
X, y = mnist["data"], mnist["target"]
X.shape
```

(70000, 784)

```
y.shape
```

(70000,)



## 실습 – MNIST dataset 로드

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

```
X_train.shape
```

```
(60000, 784)
```

```
X_test.shape
```

```
(10000, 784)
```

```
import numpy as np
```

```
shuffle_index = np.random.permutation(60000)
```

```
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

```
shuffle_index
```

```
array([12628, 37730, 39991, ..., 860, 15795, 56422])
```

**Training set 순서 섞기 (shuffling)**





### 실습 – KNN classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=4)
knn_clf.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=-1, n_neighbors=4, p=2,
                    weights='distance')
```

```
y_knn_pred = knn_clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```

```
0.9714
```





### 실습 – make\_moons 함수를 활용한 데이터 생성

- ✓ **Make\_moons 함수는 두 개의 초승달 모양의 클러스터를 이루는 데이터를 생성한다. 이렇게 만들어진 데이터는 직선을 사용하여 분류할 수 없다.**

- **해당 함수의 인수는 다음과 같다.**

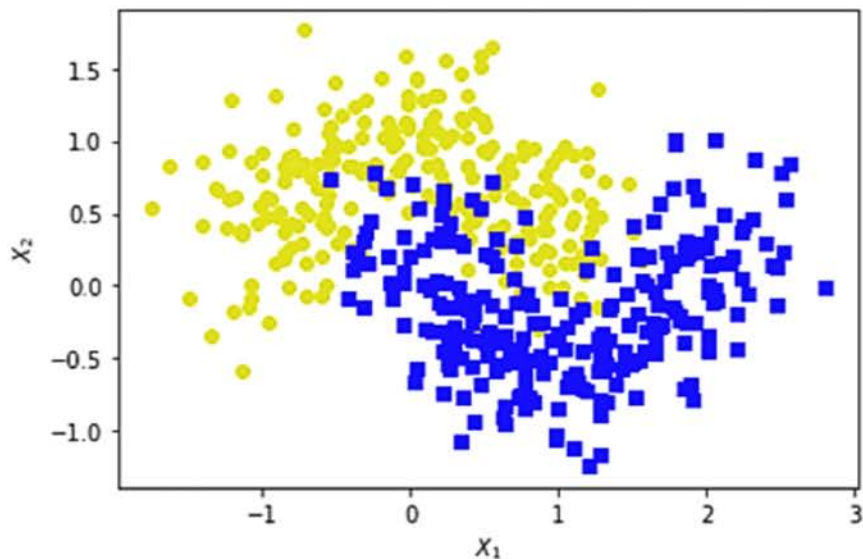
- **n\_samples:** 표본 데이터의 수, default 100
- **shuffle:** 샘플의 셔플 여부, default true
- **noise:** 잡음의 크기. (0이면 정확한 반원을 이룬다.)
- **random\_state:** 데이터셋의 랜덤 생성 수



## 실습 – make\_moons 함수를 활용한 데이터 생성

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.show()
```



데이터의 형태 출력



## 실습 – Bagging Ensemble

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

0.904

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

0.856

✓ Bagging Ensemble을 이용할 경우가 약 90.4%로, 단일 Decision Tree의 경우 (약 85.6%)보다 더 잘 분류했다.



## 실습 – Bagging Ensemble

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.5, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
```

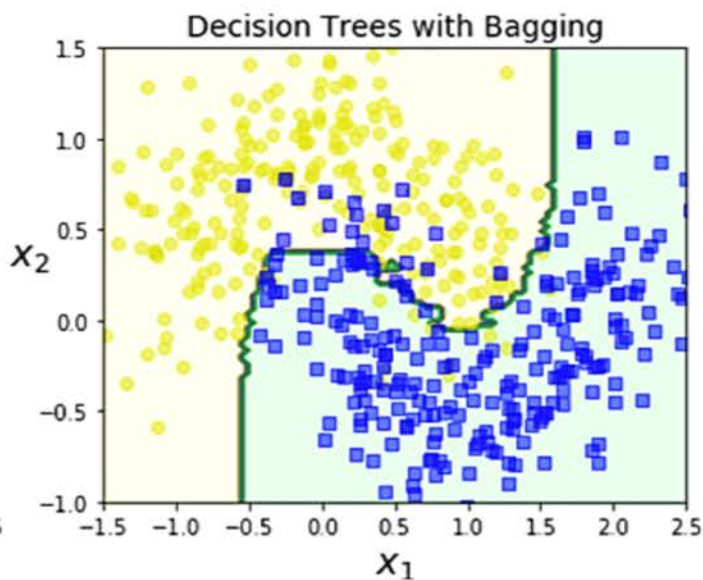
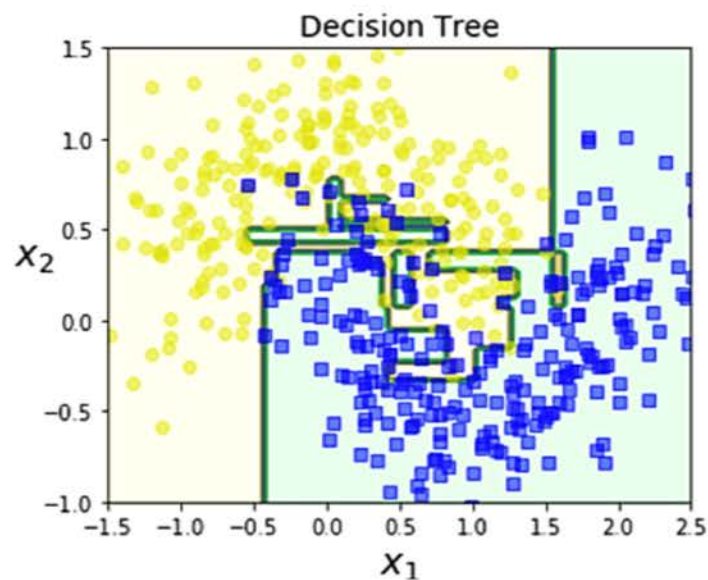
- ☑ 단일 Decision Tree와 Bagging Ensemble의 비교를 위하여 Decision Boundary를 표시하는 새로운 함수를 정의



## 실습 – Bagging Ensemble

```
plt.figure(figsize=(11,4))
plt.subplot(121)
plot_decision_boundary(tree_clf, X, y)
plt.title("Decision Tree", fontsize=14)
plt.subplot(122)
plot_decision_boundary(bag_clf, X, y)
plt.title("Decision Trees with Bagging", fontsize=14)
plt.show()
```

✓ 단일 Decision Tree보다 Bagging Ensemble이 과적합을 방지하고 안정적인 결과를 보이는 것을 알 수 있다.







## 실습 – AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

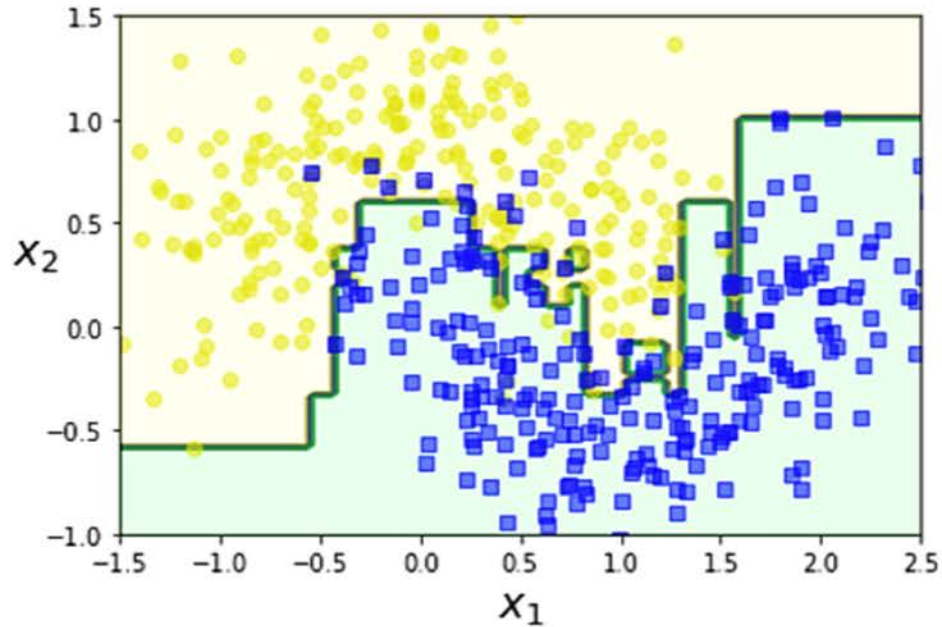
ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)

AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                            class_weight=None,
                                                            criterion='gini',
                                                            max_depth=1,
                                                            max_features=None,
                                                            max_leaf_nodes=None,
                                                            min_impurity_decrease=0.0,
                                                            min_impurity_split=None,
                                                            min_samples_leaf=1,
                                                            min_samples_split=2,
                                                            min_weight_fraction_leaf=0.0,
                                                            presort='deprecated',
                                                            random_state=None,
                                                            splitter='best'),
                    learning_rate=0.5, n_estimators=200, random_state=42)
```



## 실습 – AdaBoost

```
plot_decision_boundary(ada_clf, X, y)
```



✓ AdaBoost의 Decision Boundary



## 실습 – Random Forests

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),  
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1, random_state=42)  
  
bag_clf.fit(X_train, y_train)  
y_pred = bag_clf.predict(X_test)  
  
from sklearn.ensemble import RandomForestClassifier  
  
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)  
rnd_clf.fit(X_train, y_train)  
  
y_pred_rf = rnd_clf.predict(X_test)  
  
print(accuracy_score(y_test, y_pred))  
  
0.92  
  
print(accuracy_score(y_test, y_pred_rf))  
  
0.912  
  
np.sum(y_pred == y_pred_rf) / len(y_pred)  
  
0.976
```





### 실습 – Random Forests

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
```

✓ Random Forest는 변수 중요도까지 도출이 가능하다. (Iris dataset에 적용)