

Machine Learning Algorithm

2019. 04.23





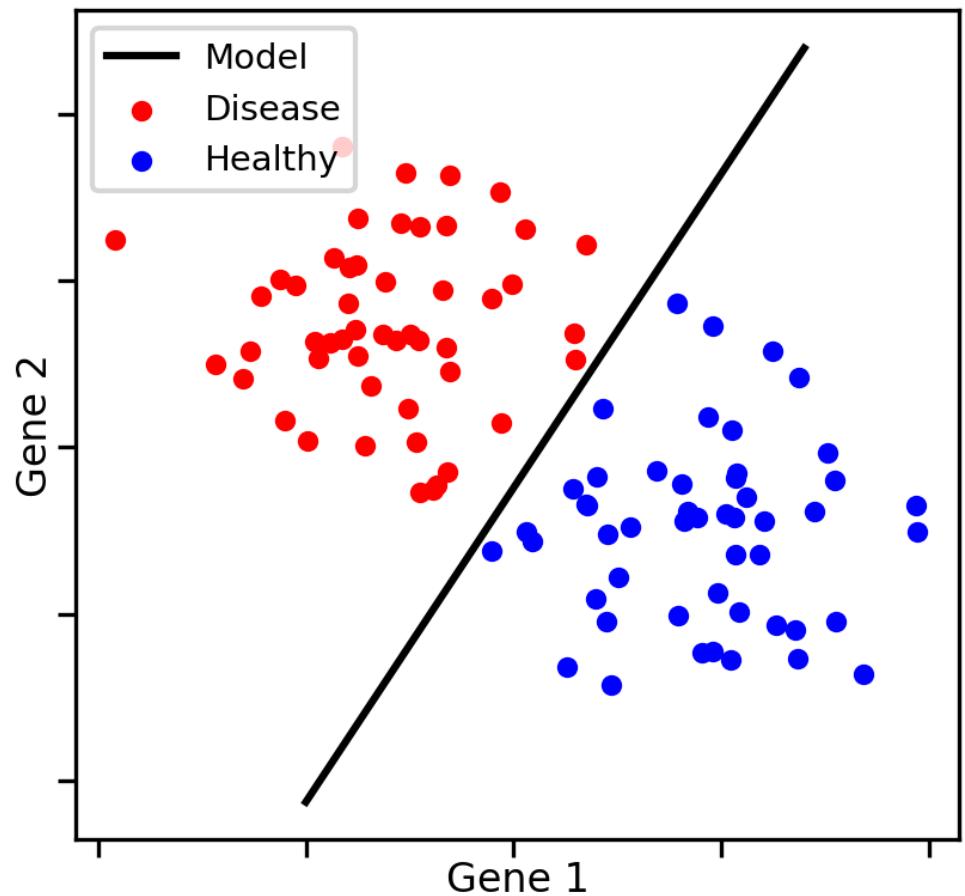
- 1. Regression과 Classification**
- 2. Logistic Regression**
- 3. kNN(k-Nearest Neighbors)**
- 4. Decision Tree**
- 5. Bagging and Random Forrest**
- 6. Boosting and XGBoost**
- 7. Support Vector Machine (SVM) / Classifier (SVC)**
- 8. Voting Classifier**
- 9. ExtraTree**
- 10. AdaBoost**



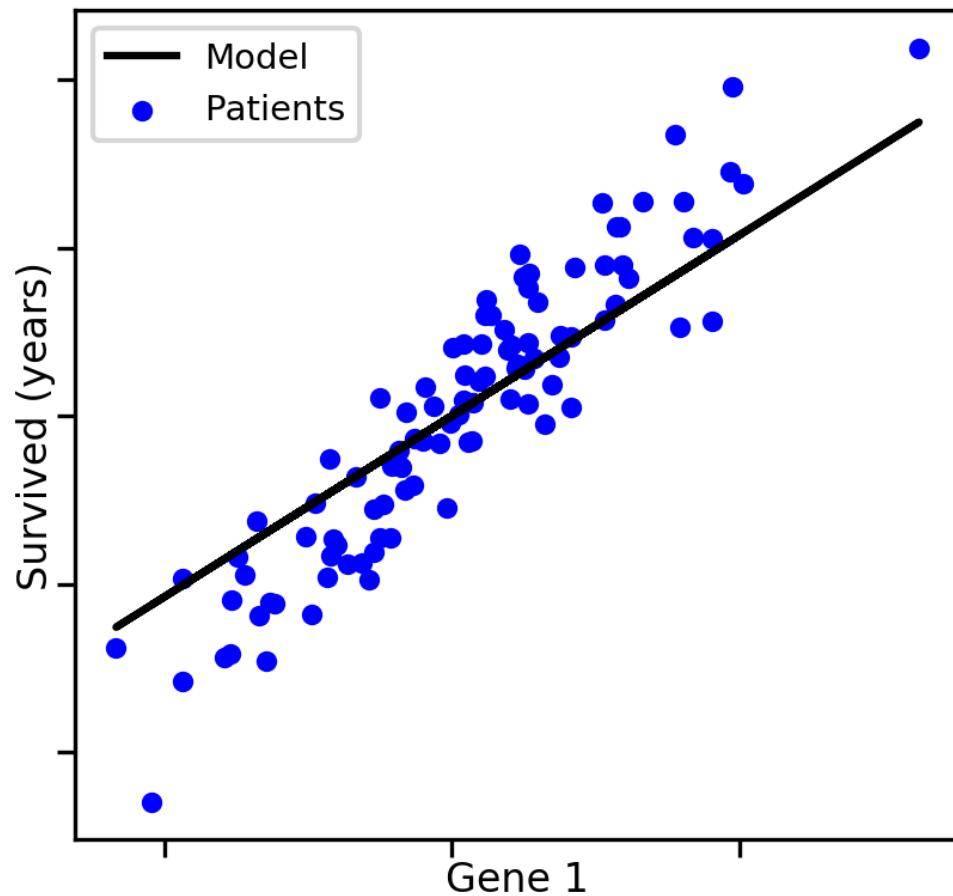
Classification

Regression과 Classification

Classification

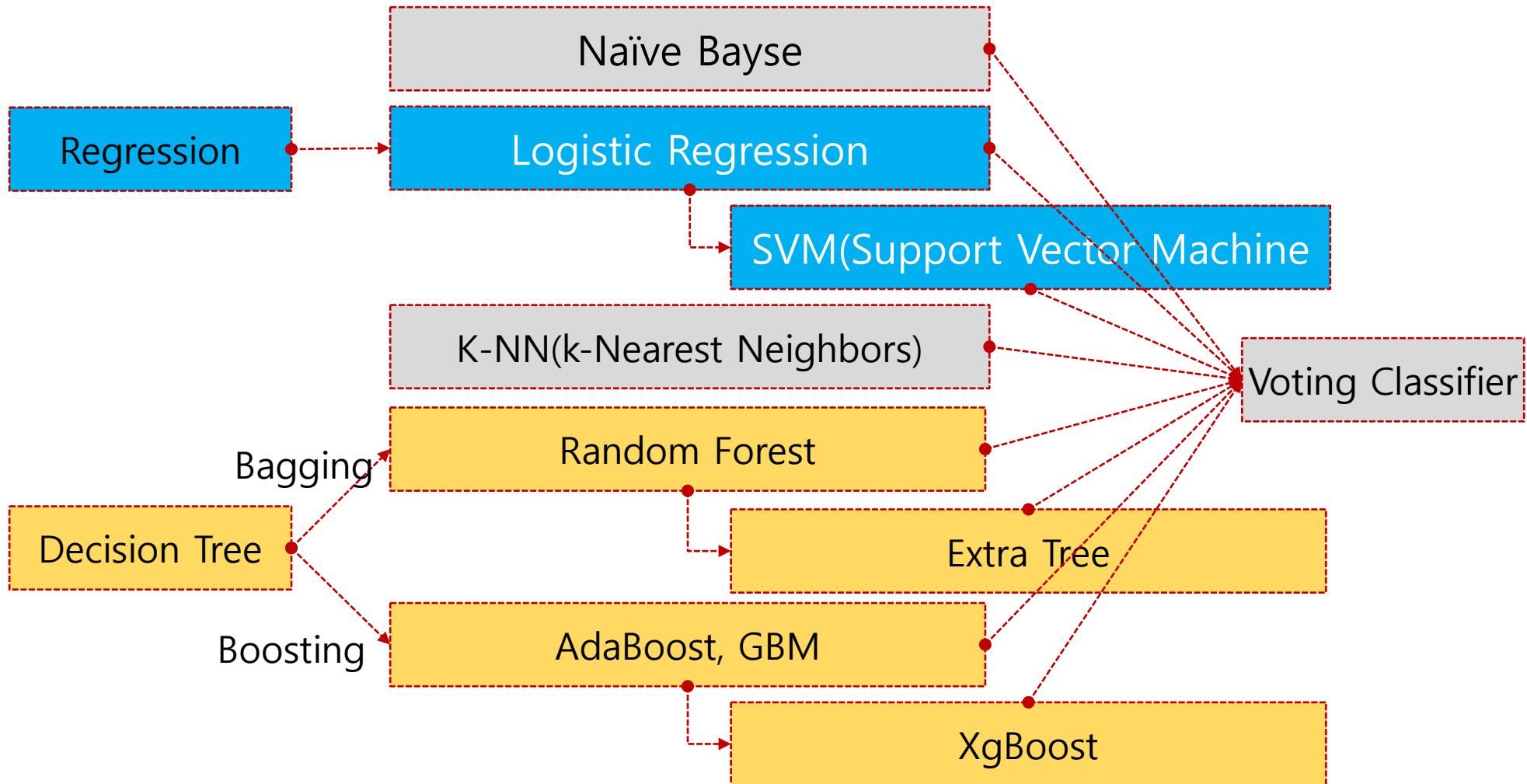


Regression



Classification

□ 알고리즘간의 관계



Logistic Regression

□ 알고리즘의 이해

- 출력변수가 '범주형'으로 되어있는
데이터 포인트를 분류(Classification)

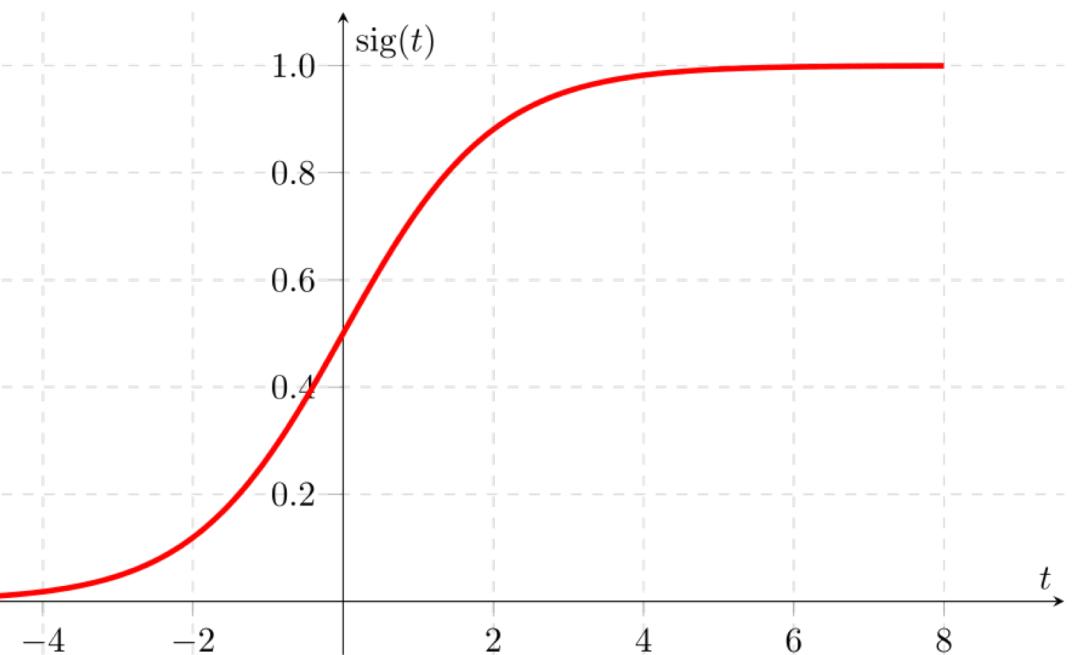
하기 위한 모델

$$\text{sig}(t) = \frac{1}{1+e^{-t}}$$

- 입력값 x 에 대한 y 의 확률로 범주를
구분함

- 수식 $f(x) = \frac{1}{1+e^{-x}}$

- $Y > 0.5$ 이면 $y=1(\사망)$ 으로 예측하고,
 $y < 0.5$ 이면 $y=0(\생존)$ 으로 예측

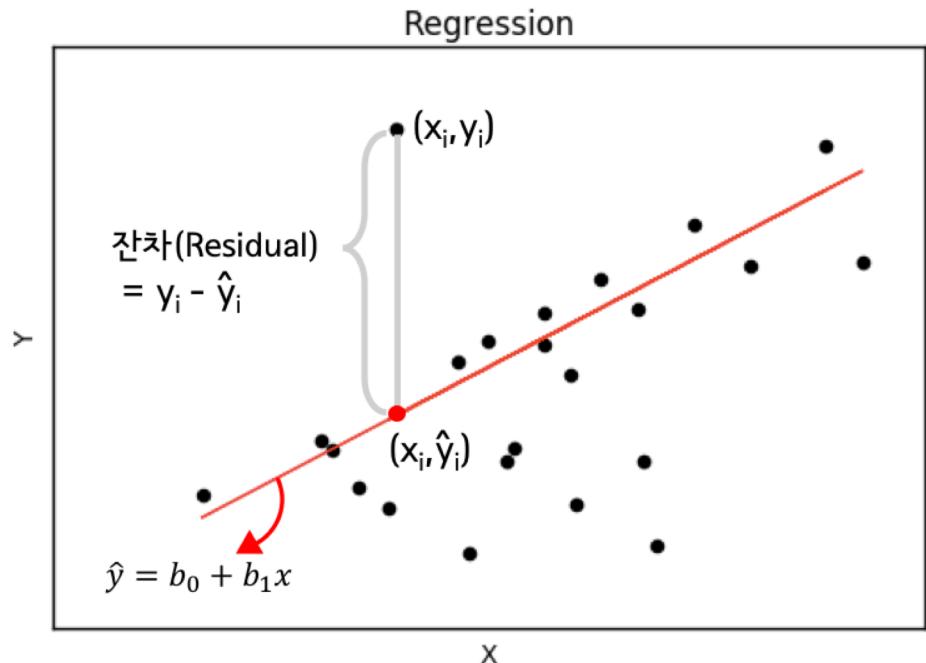


Logistic Regression

□ 선형 회귀모델에서 도출

- 선형회귀 모델은 주어진 데이터들의 추세선을 도출하여 특정 x에 대한 값을 예측
- 실제 y와 예측한 y간의 차이의 제곱을 최소화하는 추세선을 도출
- 선형 회귀 모델은 출력변수의 값이 $(-\infty, \infty)$ 에서 정의됨

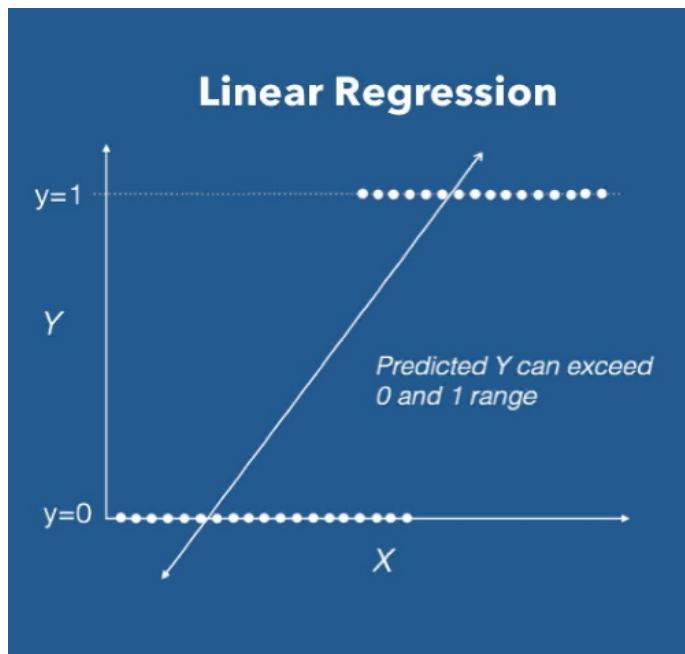
$$Y = \beta_0 + \beta_1 x + \varepsilon$$



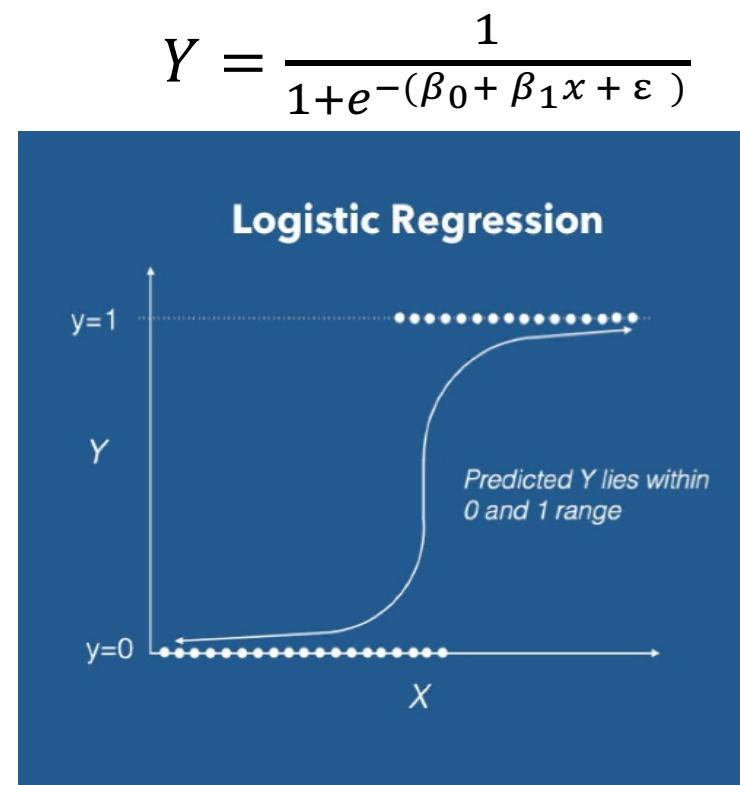
Logistic Regression

□ 선형 회귀모델에서 도출

- 선형회귀모델의 결과를 Logistic 함수의 x 에 삽입하여 $(0,1)$ 의 확률분포로 변환시킴으로써 0과 1의 두 클래스로 분류할 수 있도록 함
- $Y = \beta_0 + \beta_1 x + \varepsilon$



$$f(x) = \frac{1}{1+e^{-x}}$$



Logistic Regression

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.linear_model import LogisticRegression
```



Logistic Regression

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.linear_model import LogisticRegression
```

regularization
parameters

- Create an instance of the class

```
LR = LogisticRegression(penalty='l2', C=10.0)
```



Logistic Regression

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.linear_model import LogisticRegression
```

- Create an instance of the class

```
LR = LogisticRegression(penalty='l2', C=10.0)
```



regularization
parameters

- Fit the instance on the data and then predict the expected value

```
LR = LR.fit(X_train, y_train)
```

```
y_predict = LR.predict(X_test)
```



Logistic Regression

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.linear_model import LogisticRegression
```

- Create an instance of the class

```
LR = LogisticRegression(penalty='l2', c=10.0)
```



regularization
parameters

- Fit the instance on the data and then predict the expected value

```
LR = LR.fit(X_train, y_train)
```

```
y_predict = LR.predict(X_test)
```

- Tune regularization parameters with cross-validation:

LogisticRegressionCV or **GridSearchCV**



Logistic Regression

□ 주요 파라미터

- **Penalty**

L1 regularization $\sum_i^n (y_i - \hat{y}_i)^2 + \lambda \sum_j^p |\beta_j|$ 을 최소화하는 β_j 를 찾음

L2 regularization $\sum_i^n (y_i - \hat{y}_i)^2 + \lambda \sum_j^p \beta_j^2$ 을 최소화하는 β_j 를 찾음

Where, $\hat{y}_i = \frac{1}{1+e^{-(\beta_0+\beta_1x_{i1}+\cdots+\beta_px_{ip})}}$

- **C = 1/ λ**

C가 작으면 λ가 커지므로, β가 커지는 것을 제약함



Quiz

두 그룹이 아니라 여러 그룹으로 분류하
려면 어떻게 해야 할까요?



Quiz

□ 두 그룹이 아니라 여러 그룹으로 분류하려면 어떻게 해야 할까요?

- **One-versus-rest**

- 세개의 클래스 A,B,C가 있을 때
- A vs. ~A, B vs. ~B, C vs ~C의 세개의 Logistic Regression을 생성
- 확률이 제일 높은 것을 해당 클래스로 분류

- **Softmax 함수를 사용**

- 클래스의 개수만큼 확률을 계산, 제일 높은 확률치를 해당 클래스로 분류

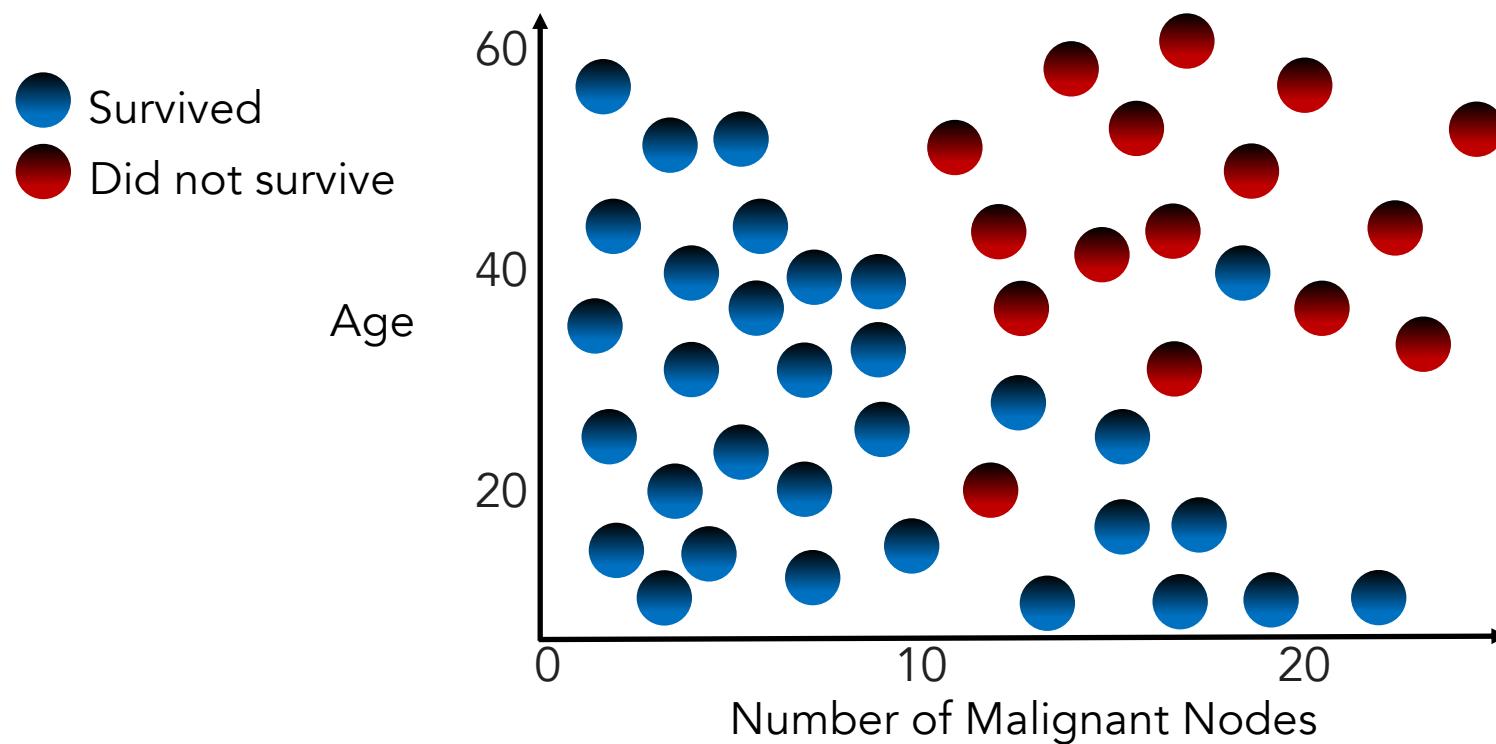
- **다른 알고리즘(모델)을 사용**

- K-NN, SVM, Decision Tree 등



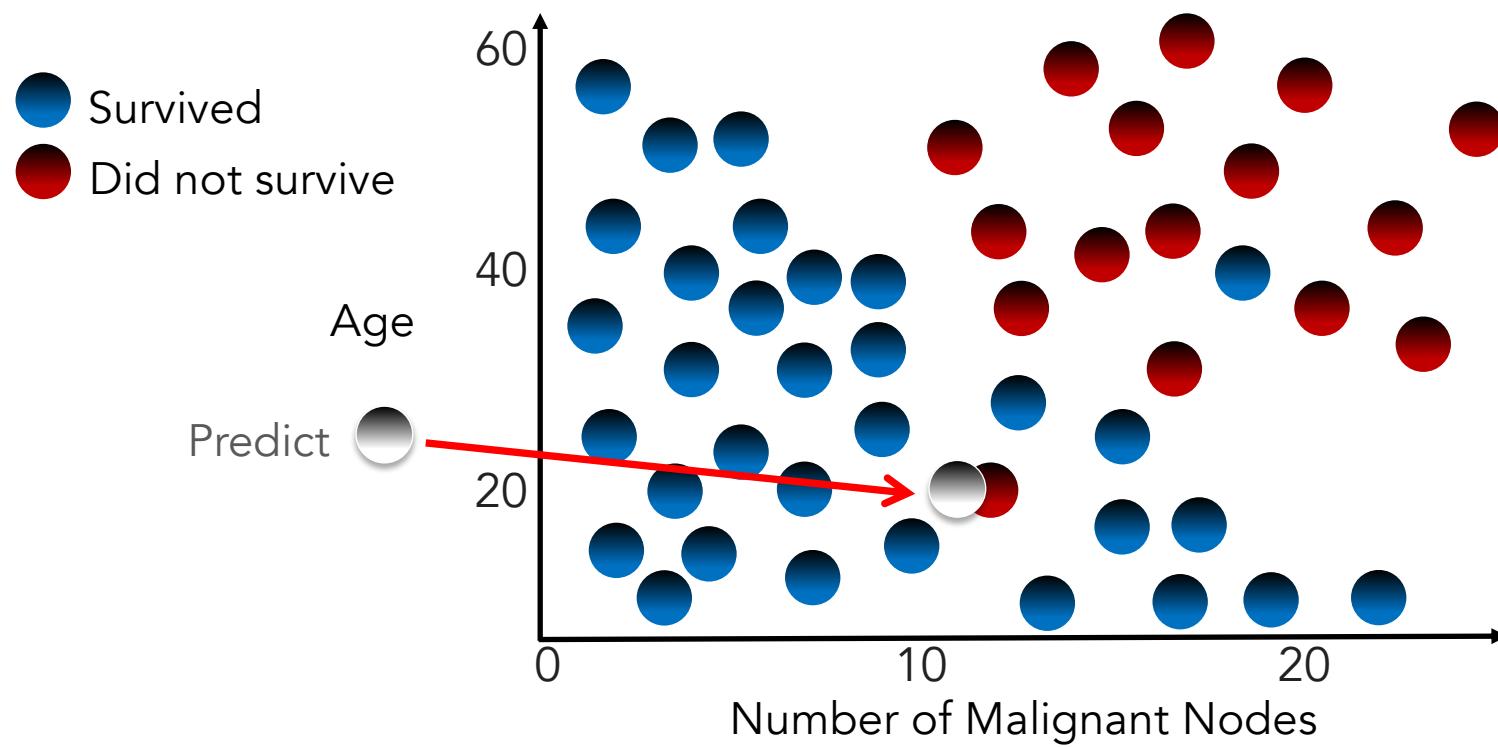
k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해



k-NN(k-Nearest Neighbors)

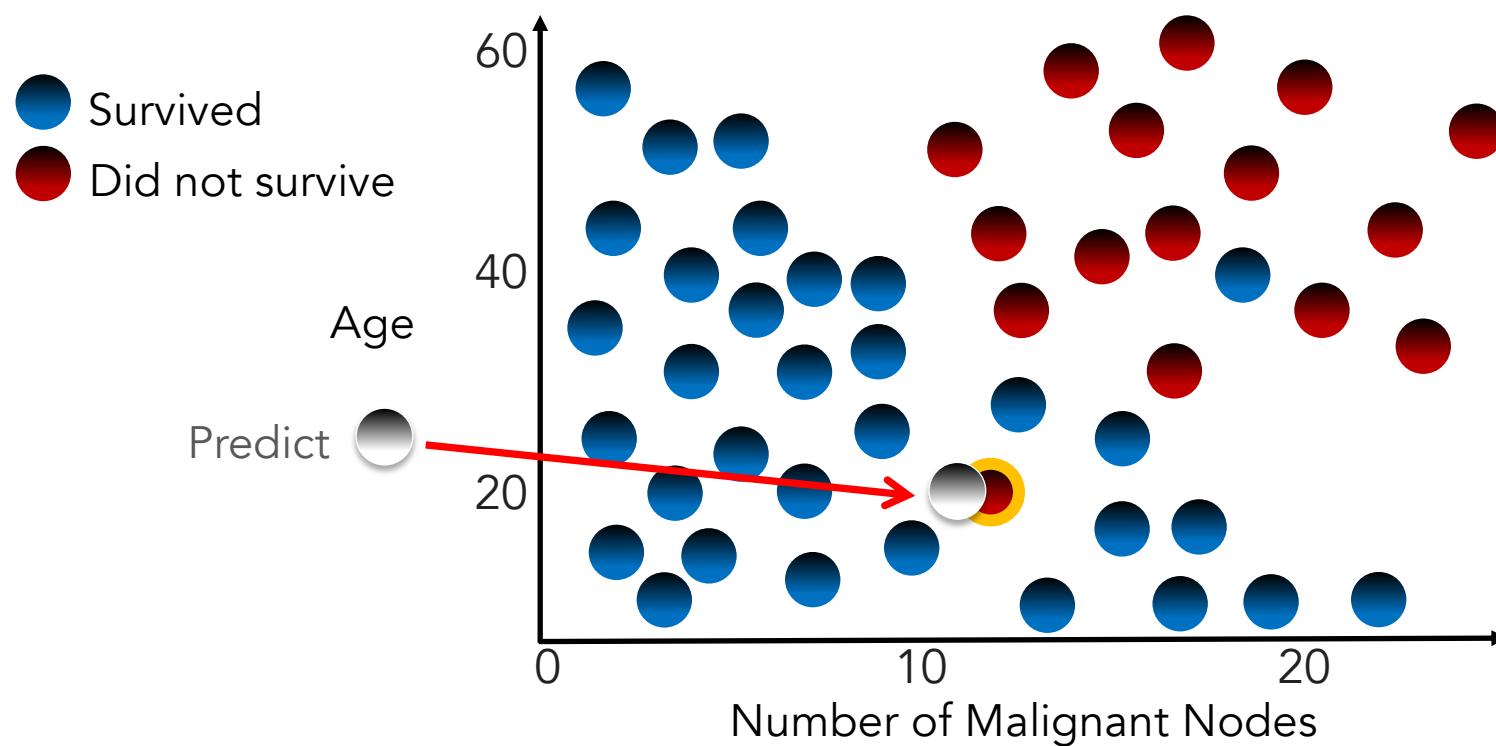
□ 알고리즘의 이해



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

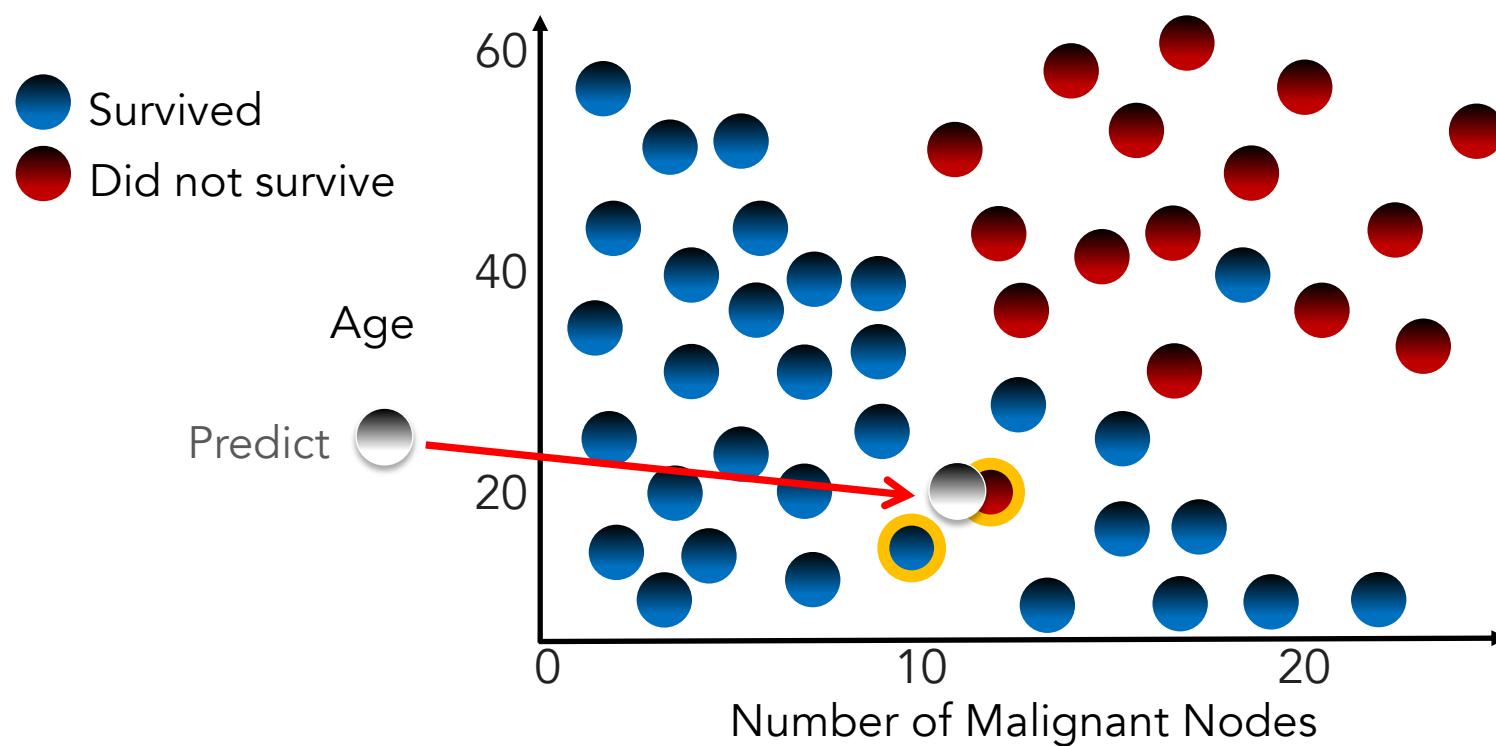
Neighbor Count (K = 1): 0 1



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

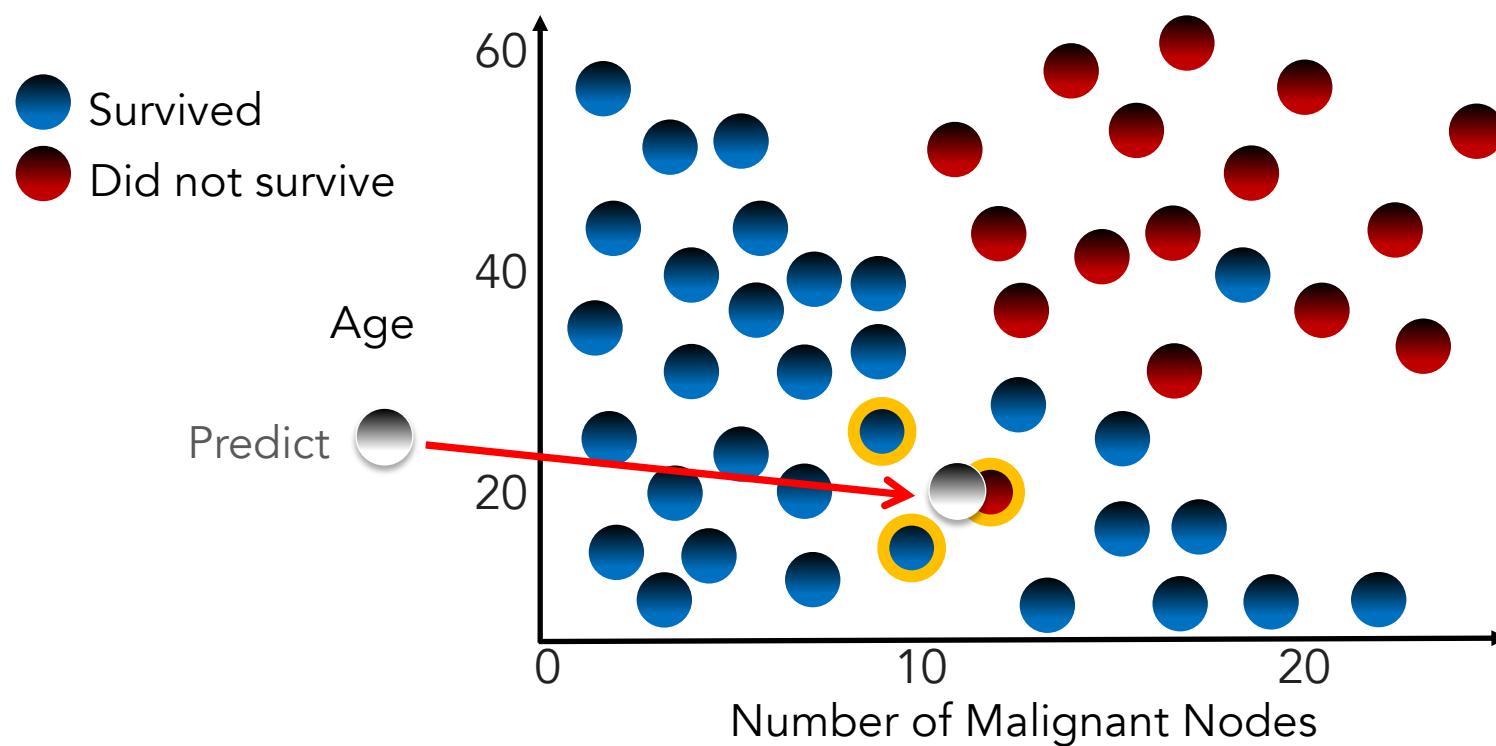
Neighbor Count (K = 2): ● 1 ● 1



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

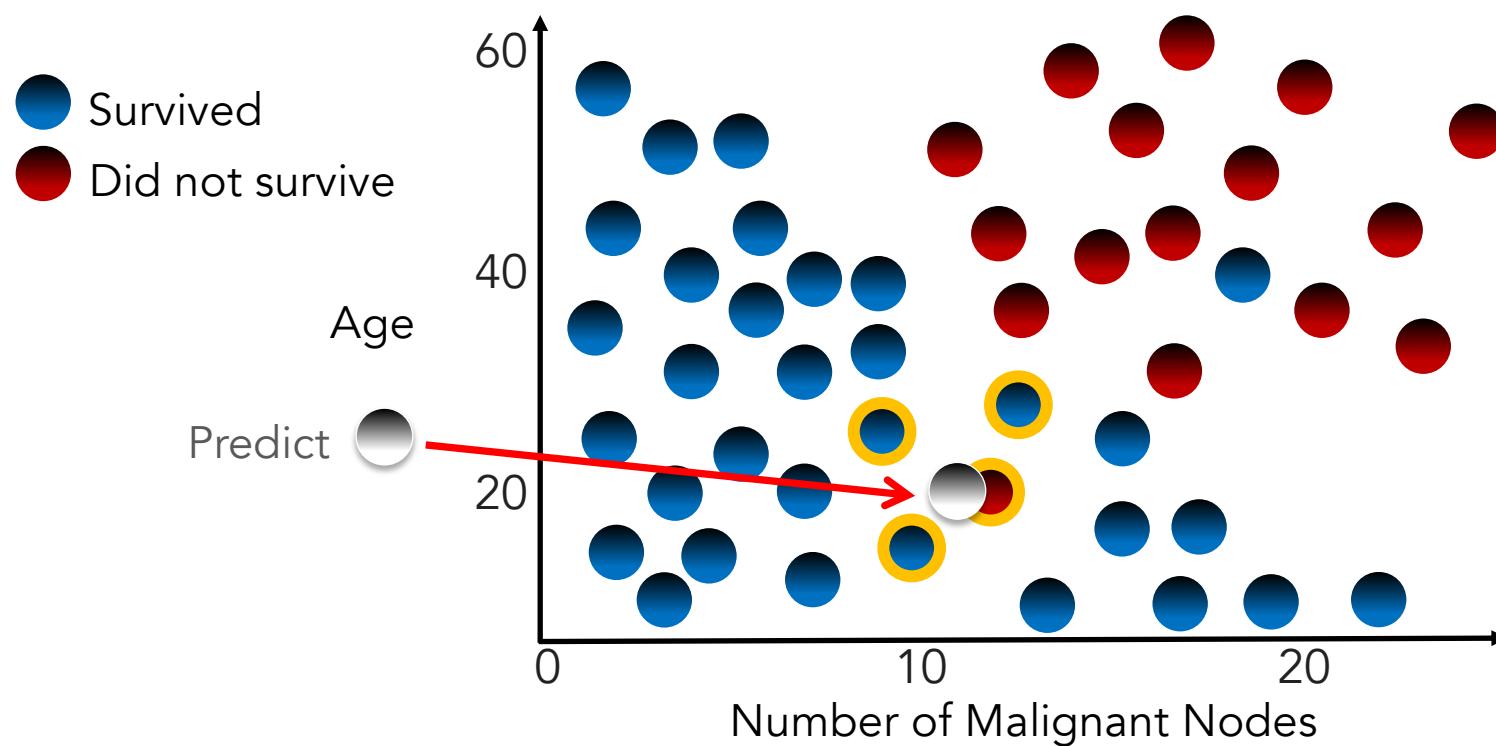
Neighbor Count (K = 3): ● 2 ● 1



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

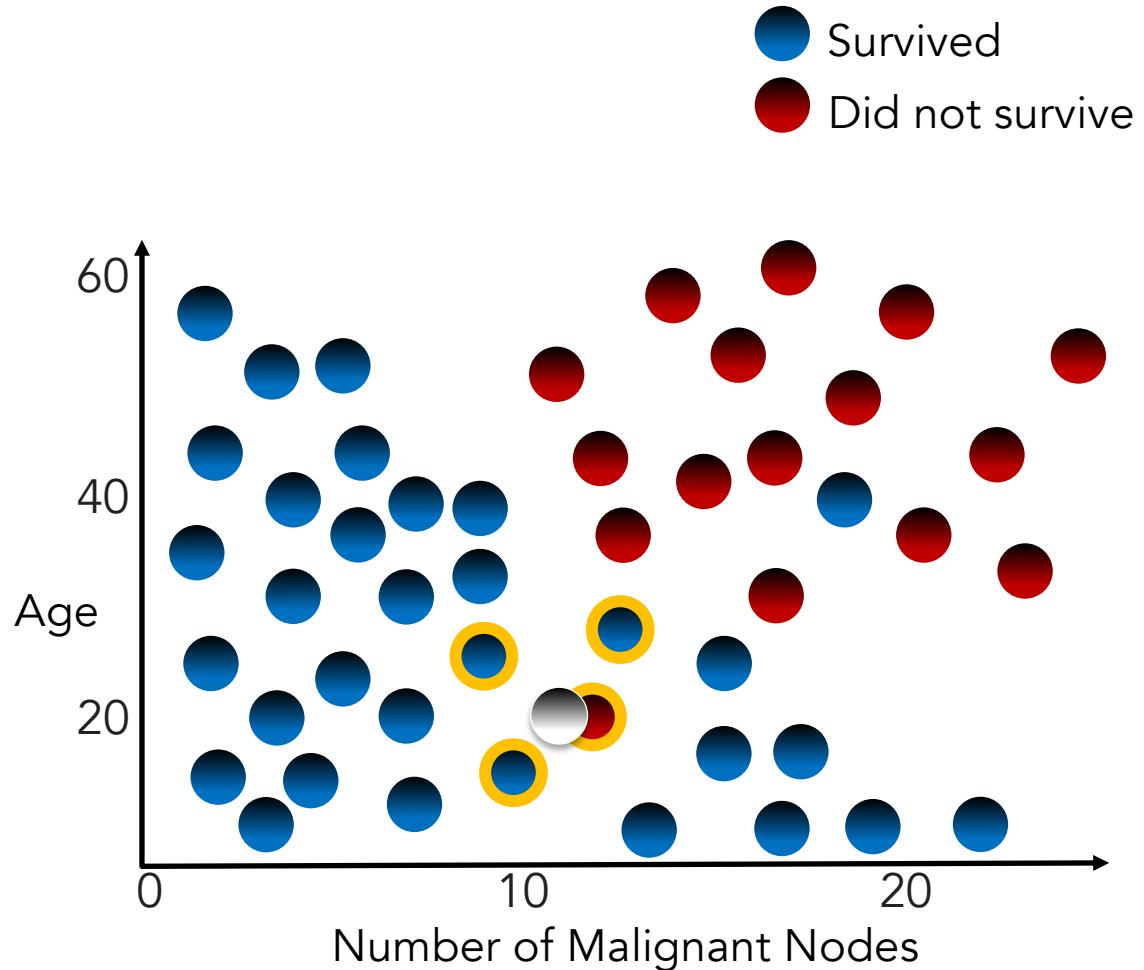
Neighbor Count (K = 4): ● 3 ● 1



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

- K의 결정
- 거리 계산
- 가중치 정의



k-NN(k-Nearest Neighbors)

□ 알고리즘의 이해

■ 주어진 학습 데이터 X, Y

- X : Input Variables
- Y : Output Variables

■ 새로운 포인트 Q가 등장

1. Q와 모든 X와의 거리 $D_i(Q, X_i)$ 를 계산한다.
2. D_i 중 가장 작은 k개를 찾는다.
3. K개의 이웃 포인트들의 y값을 확인하여 \hat{y} 을 결정한다. 이 때 거리에 대한 가중치를 적용할 수도 있다.



k-NN(k-Nearest Neighbors)

□ Code syntax (코드 문법)

- Import the class containing the classification method
from sklearn.neighbors import KNeighborsClassifier



k-NN(k-Nearest Neighbors)

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.neighbors import KNeighborsClassifier
```

- Create an instance of the class

```
KNN = KNeighborsClassifier(n_neighbors=3,  
                           weights='uniform')
```



k-NN(k-Nearest Neighbors)

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.neighbors import KNeighborsClassifier
```

- Create an instance of the class

```
KNN = KNeighborsClassifier(n_neighbors=3, weights='uniform')
```

- Fit the instance on the data and predict the expected value

```
KNN = KNN.fit(X_data, y_data)
```

```
y_predict = KNN.predict(X_data)
```

- Regression can be done with KNeighborsRegressor.



k-NN(k-Nearest Neighbors)

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.neighbors import KNeighborsClassifier
```

- Create an instance of the class

```
KNN = KNeighborsClassifier(n_neighbors=3, weights='uniform')
```

- Fit the instance on the data and predict the expected value

```
KNN = KNN.fit(X_data, y_data)
```

```
y_predict = KNN.predict(X_data)
```

- Regression can be done with **KNeighborsRegressor**.

- Tune regularization parameters with cross-validation:

GridSearchCV



k-NN(k-Nearest Neighbors)

□ 주요 파라미터

▪ n_neighbors=3은 k값을 의미

- n_neighbors가 너무 작으면 과적합할 수 있으며, 지역적인 노이즈에 민감
- n_neighbors가 너무 크면, 지역적인 데이터 구조를 잘 반영하지 못할 수 있음
- [GridSearchCV](#)를 이용하여 여러 가지 n_neighbors에 대한 예측 성능을 시뮬레이션하여 결정, 이때 검증데이터를 사용



k-NN(k-Nearest Neighbors)

□ 주요 파라미터

▪ **weights='uniform'**

- uniform : 가중치를 적용하지 않음
- distance : 거리의 역수로 가중치를 적용. 가까운 것이 큰 영향을 미침
- [callable] : 사용자가 정의한 함수를 적용



Quiz

□ 다음 중 k-NN에 대한 설명 중 맞는 것을 모두 고르시오.

1. 학습과정이 없다.
2. 데이터 포인트 수에 따라 예측 속도의 차이는 별로 없다.
3. 가중치의 적용에 따라 분류 결과가 달라질 수 있다.
4. 회귀에는 적용할 수 없다.



Decision Tree

□ 포유류 여부에 대한 분류

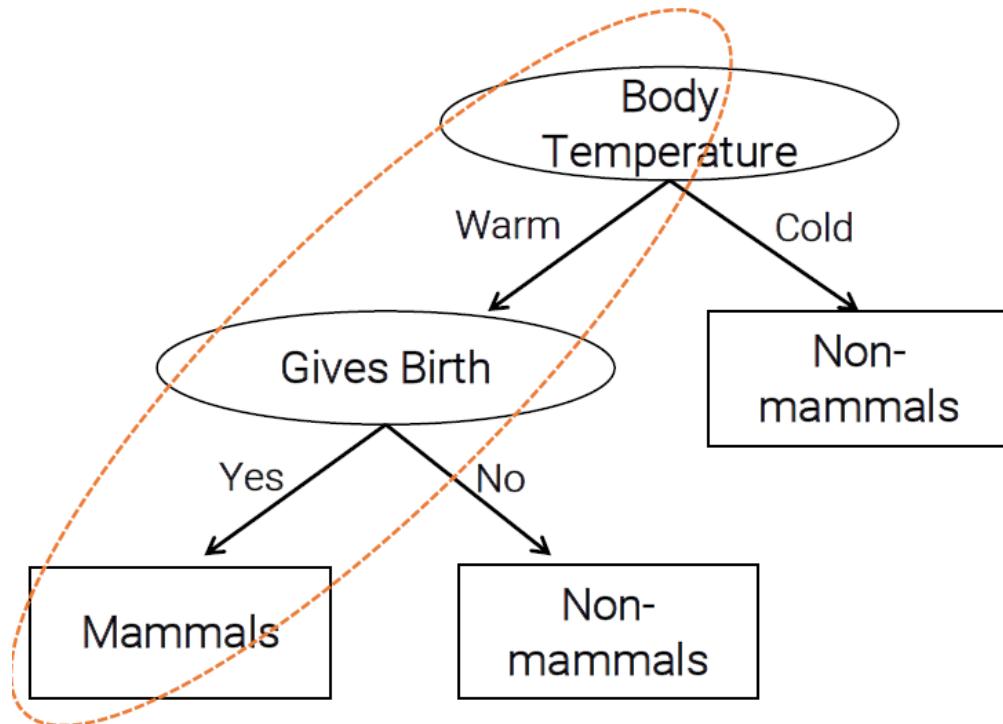
| ID | Body Temperature | Gives Birth? | Weight(kg) | ... | Mammal? |
|-----|------------------|--------------|------------|-----|------------|
| 1 | Warm | Yes | 65 | ... | Mammal |
| 2 | Warm | No | 0.32 | ... | Non-mammal |
| 3 | Cold | No | 0.14 | ... | Non-mammal |
| 4 | Warm | Yes | 3,000 | ... | Mammal |
| 5 | Cold | No | 127 | ... | Non-mammal |
| 6 | Cold | No | 0.35 | ... | Non-mammal |
| 7 | Warm | Yes | 1.5 | ... | Mammal |
| ... | ... | ... | ... | ... | ... |

Body Temperature = Warm → 온혈동물(혹은 정온동물)
Body Temperature = Cold → 냉혈동물(혹은 변온동물)

Gives Birth = Yes → 태생
Gives Birth = No → 난생

Decision Tree

▣ 도출된 Decision Tree



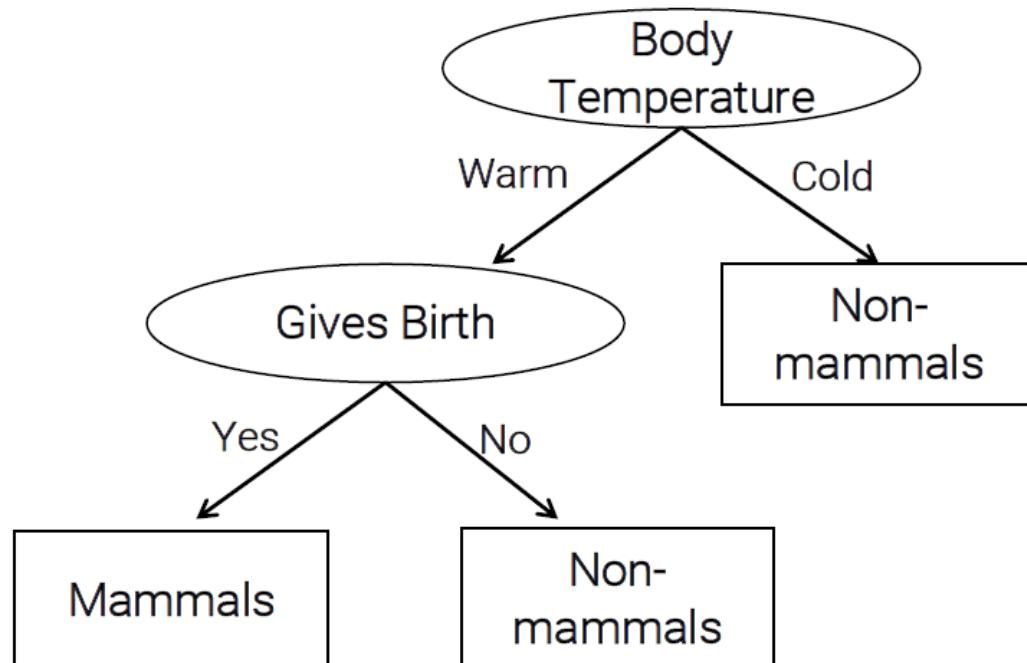
▶ “체온”과 “새끼를 낳는 방법”이 포유류를 분류하는 데에 있어 중요한 변수라는 것을 알 수 있다.

▶ 의사결정나무를 통해 포유류으로 분류하는 규칙을 구할 수 있다.

- 온혈동물이고 태생이면 포유류이다.
- IF Body Temperature = Warm and Gives Birth = Yes, then Mammals.

Decision Tree

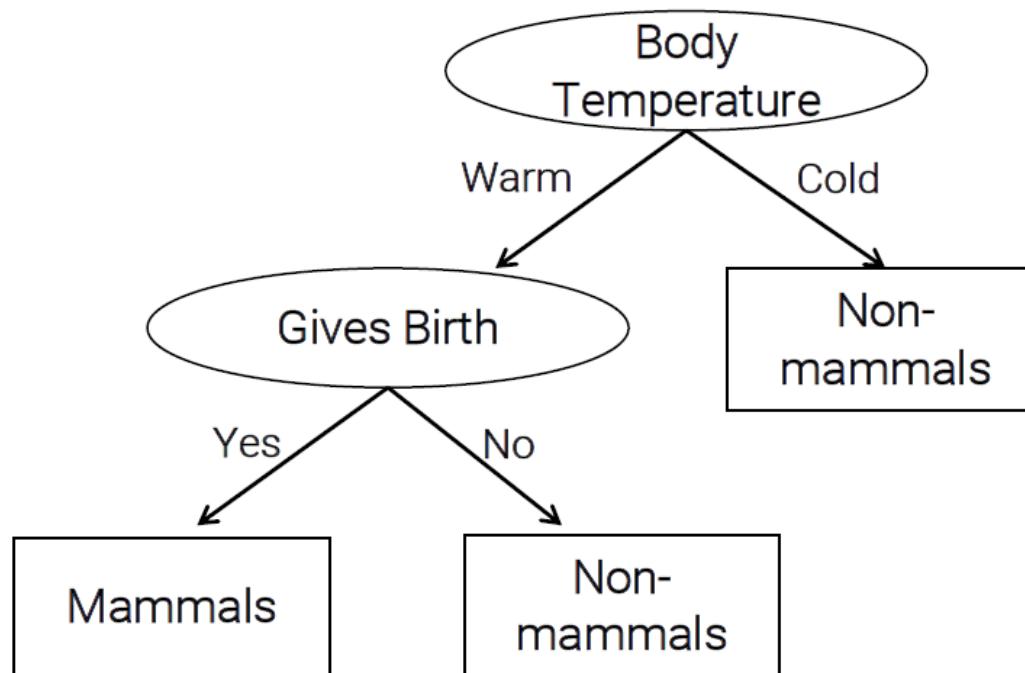
□ 새로운 데이터의 분류



새로운 데이터

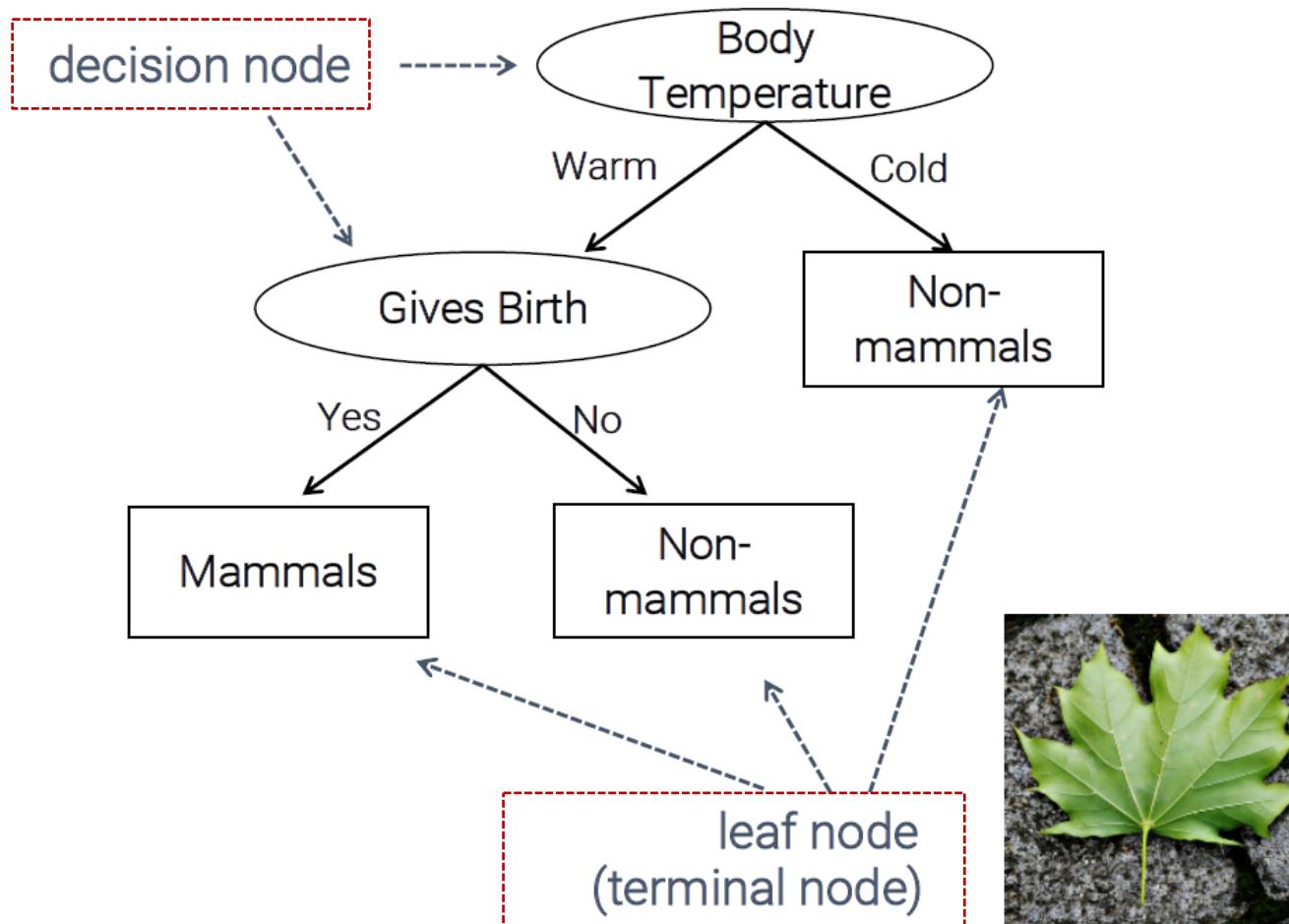
Decision Tree

□ 분류 결과



Decision Tree

□ Decision Tree의 구조



Decision Tree

□ 분류 예시(채무 이행 여부 예측)

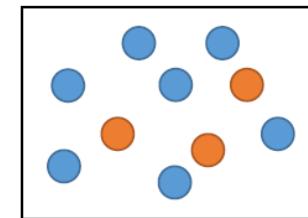
| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 |
|----|------|----|--------|--------|
| 1 | Yes | 미혼 | 125 | No |
| 2 | No | 기혼 | 100 | No |
| 3 | No | 미혼 | 70 | No |
| 4 | Yes | 기혼 | 120 | No |
| 5 | No | 이혼 | 95 | Yes |
| 6 | No | 기혼 | 60 | No |
| 7 | Yes | 이혼 | 220 | No |
| 8 | No | 미혼 | 85 | Yes |
| 9 | No | 기혼 | 75 | No |
| 10 | No | 미혼 | 90 | Yes |

Decision Tree

□ 채무 불이행자 선택 과정

| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 |
|----|------|----|--------|--------|
| 1 | Yes | 미혼 | 125 | No |
| 2 | No | 기혼 | 100 | No |
| 3 | No | 미혼 | 70 | No |
| 4 | Yes | 기혼 | 120 | No |
| 5 | No | 이혼 | 95 | Yes |
| 6 | No | 기혼 | 60 | No |
| 7 | Yes | 이혼 | 220 | No |
| 8 | No | 미혼 | 85 | Yes |
| 9 | No | 기혼 | 75 | No |
| 10 | No | 미혼 | 90 | Yes |

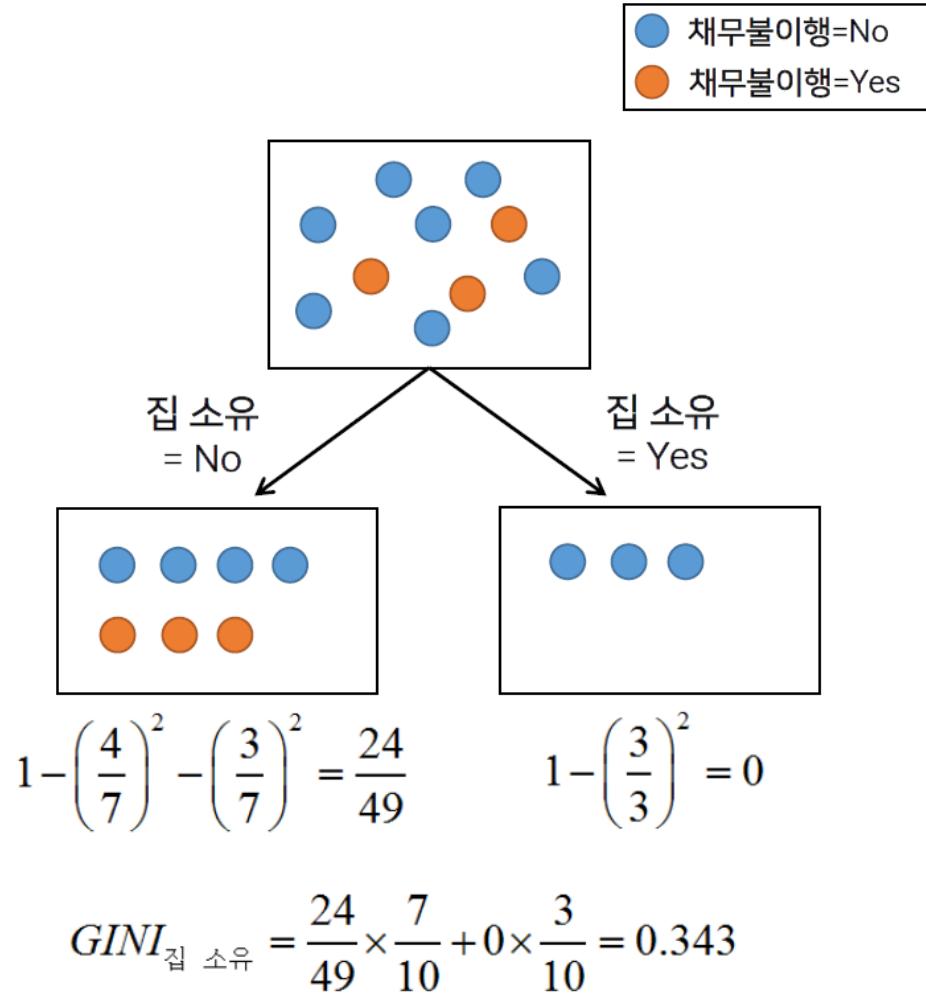
- 채무불이행=No
- 채무불이행=Yes



Decision Tree

□ “집소유” 여부에 의한 분기 시 Gini Index에 계산

| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 |
|----|------|----|--------|-----------|
| 1 | Yes | 미혼 | 125 | No |
| 2 | No | 기혼 | 100 | No |
| 3 | No | 미혼 | 70 | No |
| 4 | Yes | 기혼 | 120 | No |
| 5 | No | 이혼 | 95 | Yes |
| 6 | No | 기혼 | 60 | No |
| 7 | Yes | 이혼 | 220 | No |
| 8 | No | 미혼 | 85 | Yes |
| 9 | No | 기혼 | 75 | No |
| 10 | No | 미혼 | 90 | Yes |

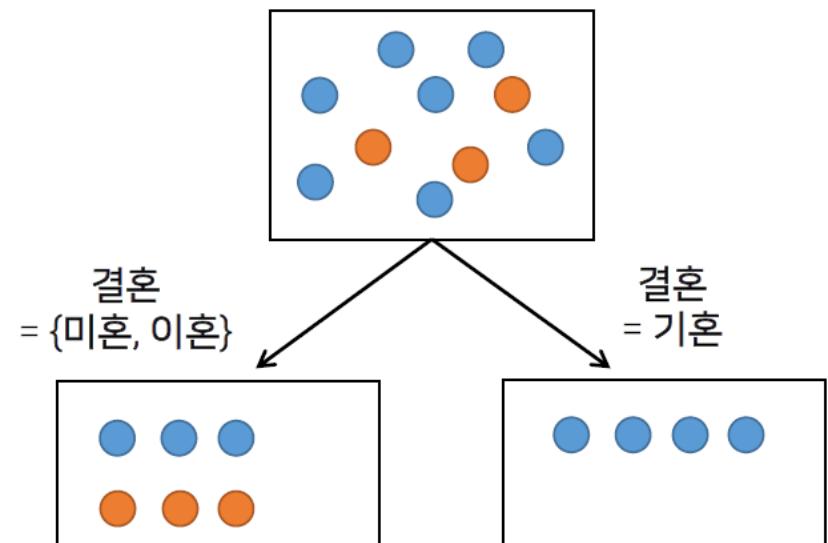


Decision Tree

□ “결혼” 여부에 의한 분기 시 Gini Index에 계산

| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 |
|----|------|----|--------|-----------|
| 1 | Yes | 미혼 | 125 | No |
| 2 | No | 기혼 | 100 | No |
| 3 | No | 미혼 | 70 | No |
| 4 | Yes | 기혼 | 120 | No |
| 5 | No | 이혼 | 95 | Yes |
| 6 | No | 기혼 | 60 | No |
| 7 | Yes | 이혼 | 220 | No |
| 8 | No | 미혼 | 85 | Yes |
| 9 | No | 기혼 | 75 | No |
| 10 | No | 미혼 | 90 | Yes |

- 채무불이행=No
- 채무불이행=Yes



$$1 - \left(\frac{3}{6} \right)^2 - \left(\frac{3}{6} \right)^2 = \frac{1}{2}$$

$$1 - \left(\frac{4}{4} \right)^2 = 0$$

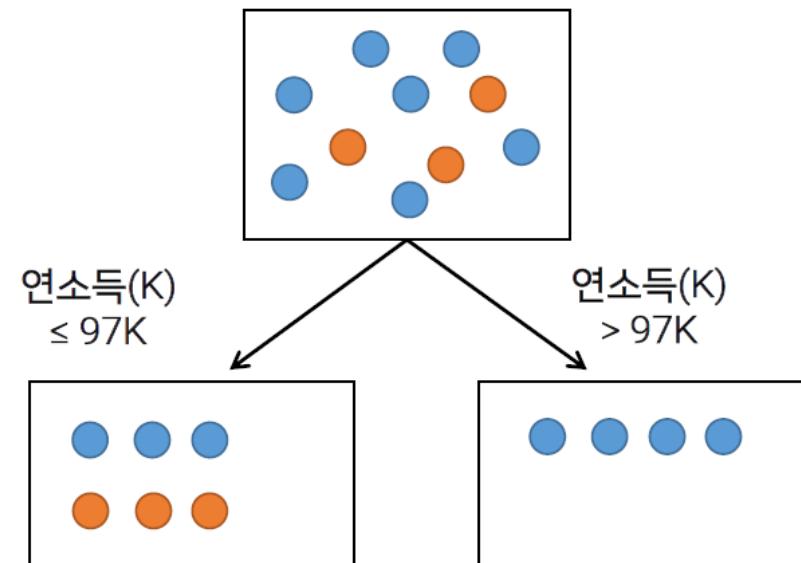
$$GINI_{\text{결혼}} = \frac{1}{2} \times \frac{6}{10} + 0 \times \frac{4}{10} = 0.3$$

Decision Tree

□ “연소득”에 의한 분기 시 Gini Index에 계산

| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 |
|----|------|----|--------|-----------|
| 1 | Yes | 미혼 | 125 | No |
| 2 | No | 기혼 | 100 | No |
| 3 | No | 미혼 | 70 | No |
| 4 | Yes | 기혼 | 120 | No |
| 5 | No | 이혼 | 95 | Yes |
| 6 | No | 기혼 | 60 | No |
| 7 | Yes | 이혼 | 220 | No |
| 8 | No | 미혼 | 85 | Yes |
| 9 | No | 기혼 | 75 | No |
| 10 | No | 미혼 | 90 | Yes |

- 채무불이행=No
- 채무불이행=Yes



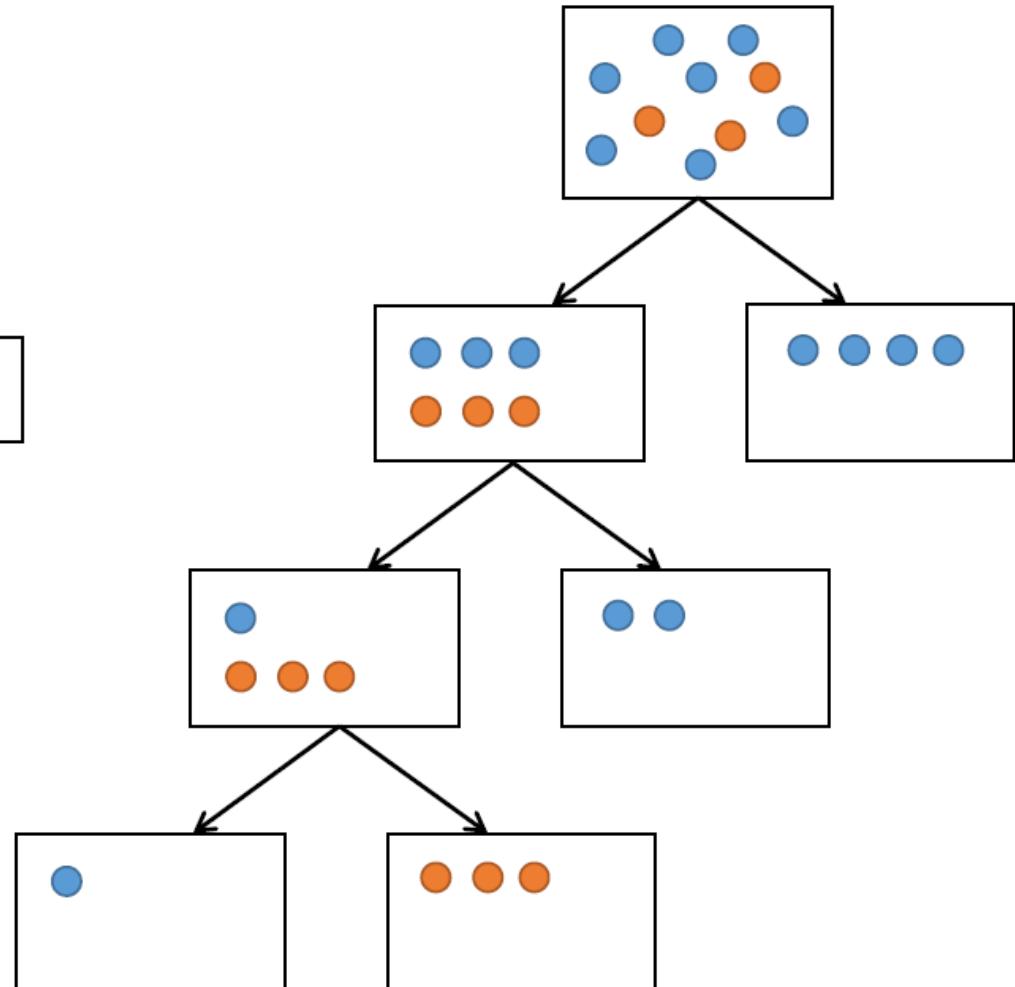
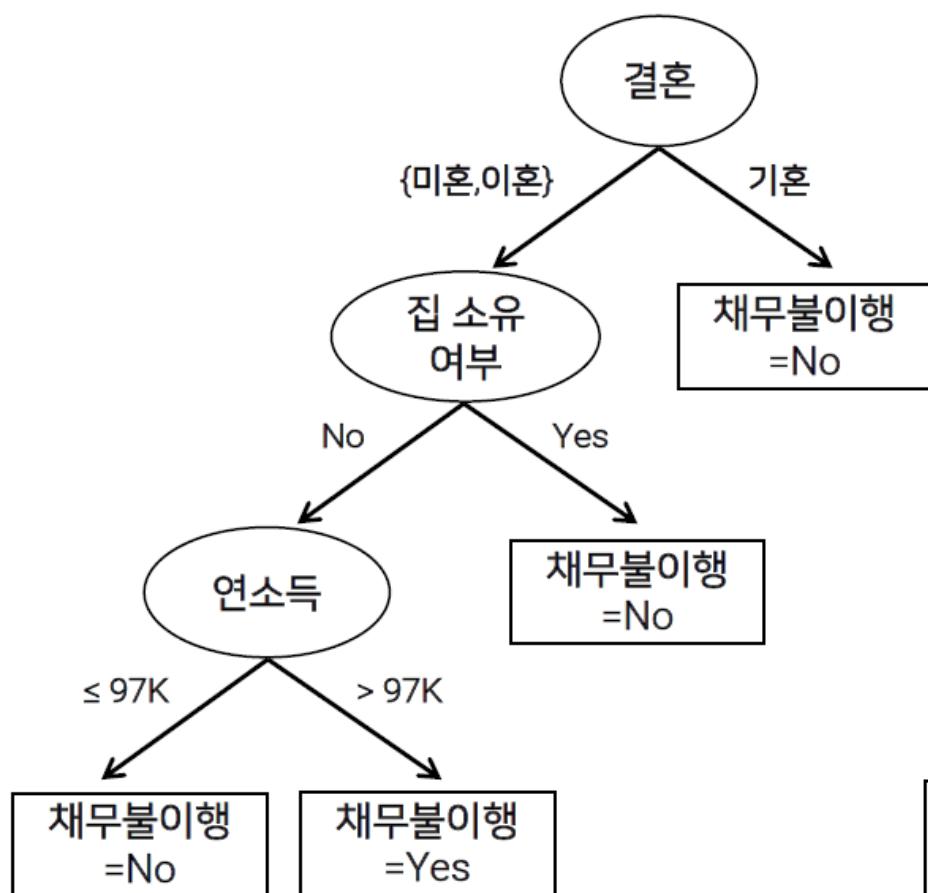
$$1 - \left(\frac{3}{6} \right)^2 - \left(\frac{3}{6} \right)^2 = \frac{1}{2}$$

$$1 - \left(\frac{4}{4} \right)^2 = 0$$

$$GINI_{연소득} = \frac{1}{2} \times \frac{6}{10} + 0 \times \frac{4}{10} = 0.3$$

Decision Tree

□ 최종 Decision Tree



□ 알고리즘

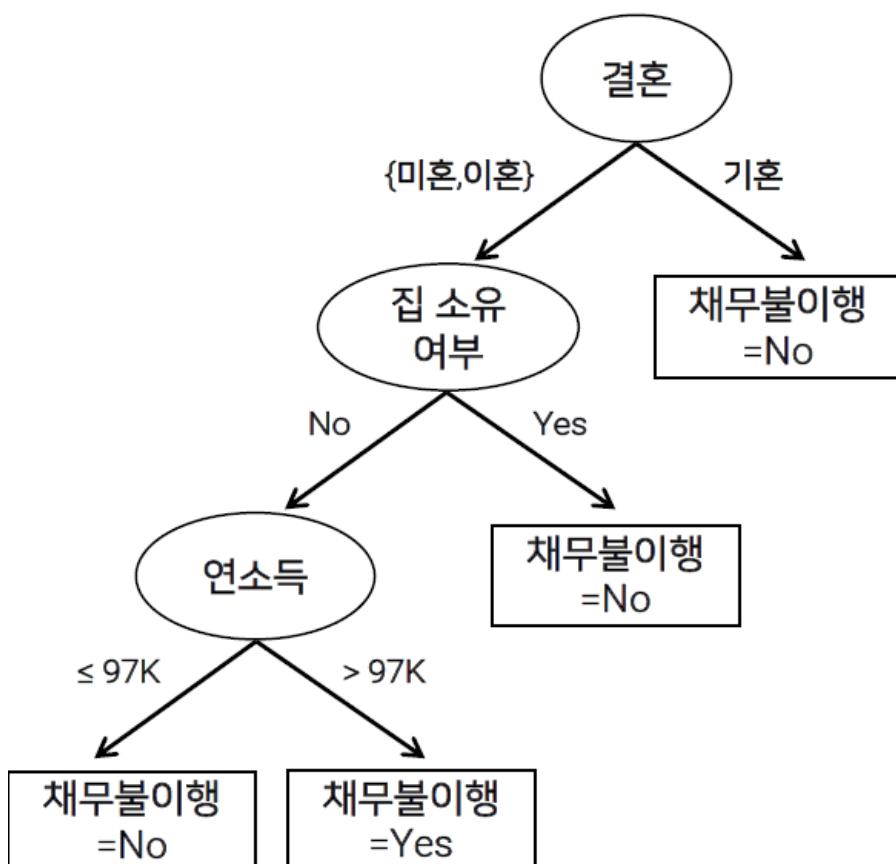
■ 모든 입력변수에 대해

- 하나의 입력변수를 선택
 - 선택한 후보변수에 데이터를 두 공간으로 나눌 수 있는 후보값 정의
 - 각 후보값별로 데이터를 두 공간으로 분할한 후 데이터 불순도(gini index or entropy)와 정보 획득량을 계산
 - 가장 불순도가 낮은 후보값을 그 입력변수의 분기 기준값으로 선정
- ### ■ 여러 입력 변수와 분기 기준값 중 가장 데이터 분순도가 낮아 지는 방법으로 데이터를 분할



Decision Tree

□ 새로운 데이터의 클래스 예측하기

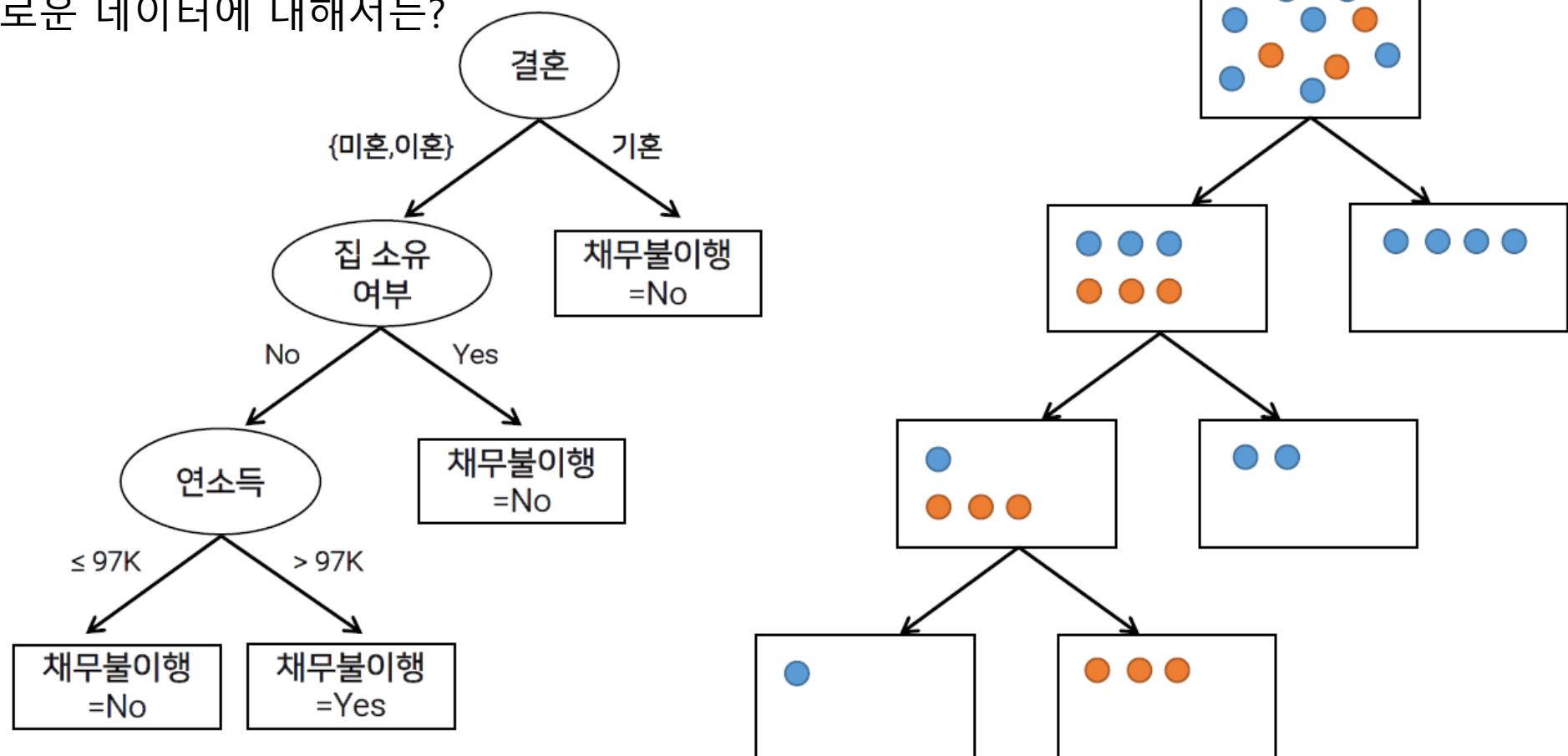


| ID | 집 소유 | 결혼 | 연소득(K) | 채무 불이행 | 예측 클래스 |
|----|------|----|--------|--------|--------|
| 11 | No | 미혼 | 55 | ? | No |
| 12 | Yes | 기혼 | 80 | ? | No |
| 13 | Yes | 미혼 | 110 | ? | No |
| 14 | No | 기혼 | 95 | ? | No |
| 15 | No | 이혼 | 300 | ? | Yes |

Decision Tree

□ 좋은 모델일까?

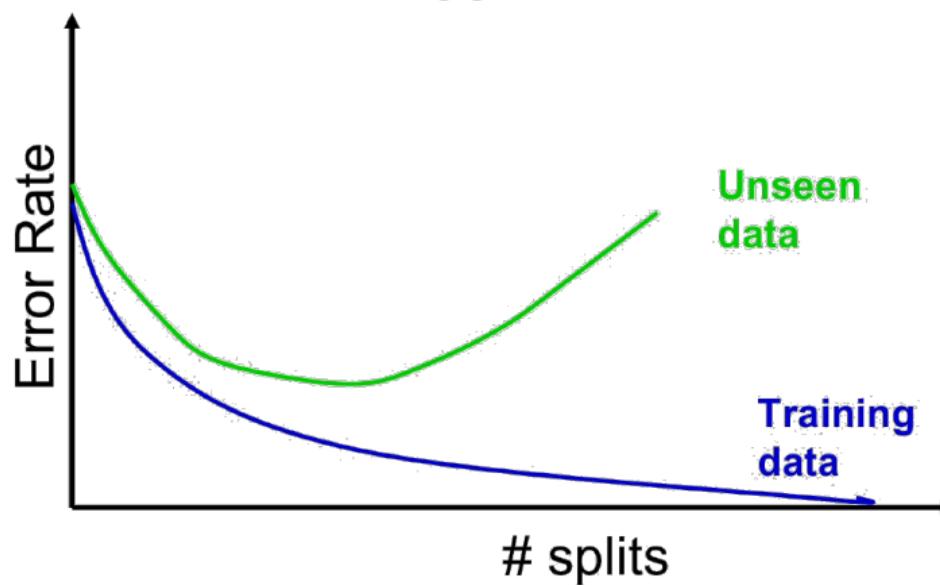
학습데이터를 완벽하게 분류
새로운 데이터에 대해서는?



Decision Tree

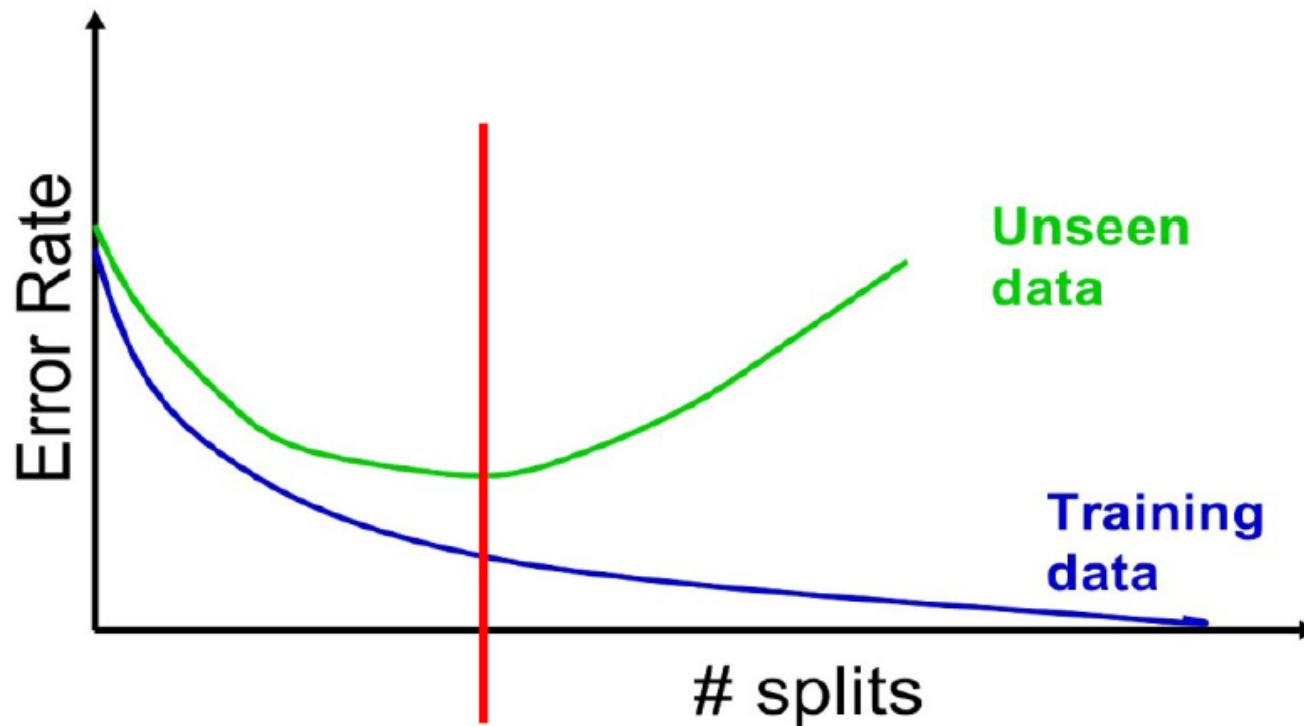
□ 과적합 문제

- 앞의 모델을 full-grown tree라고 한다.
 - CART 알고리즘은 모든 잎 노드에서 (학습 데이터에 대해) 0%의 불순도가 나올 수 있음.
 - 즉, 학습 데이터에 대한 분류 정확도가 100%가 나옴.
- 이는 과적합 문제로 이어짐.
 - 학습 데이터의 노이즈까지 모델에 반영되었을 것
 - 학습 데이터가 아닌 새로운 데이터에 대한 성능이 떨어질 것



Decision Tree

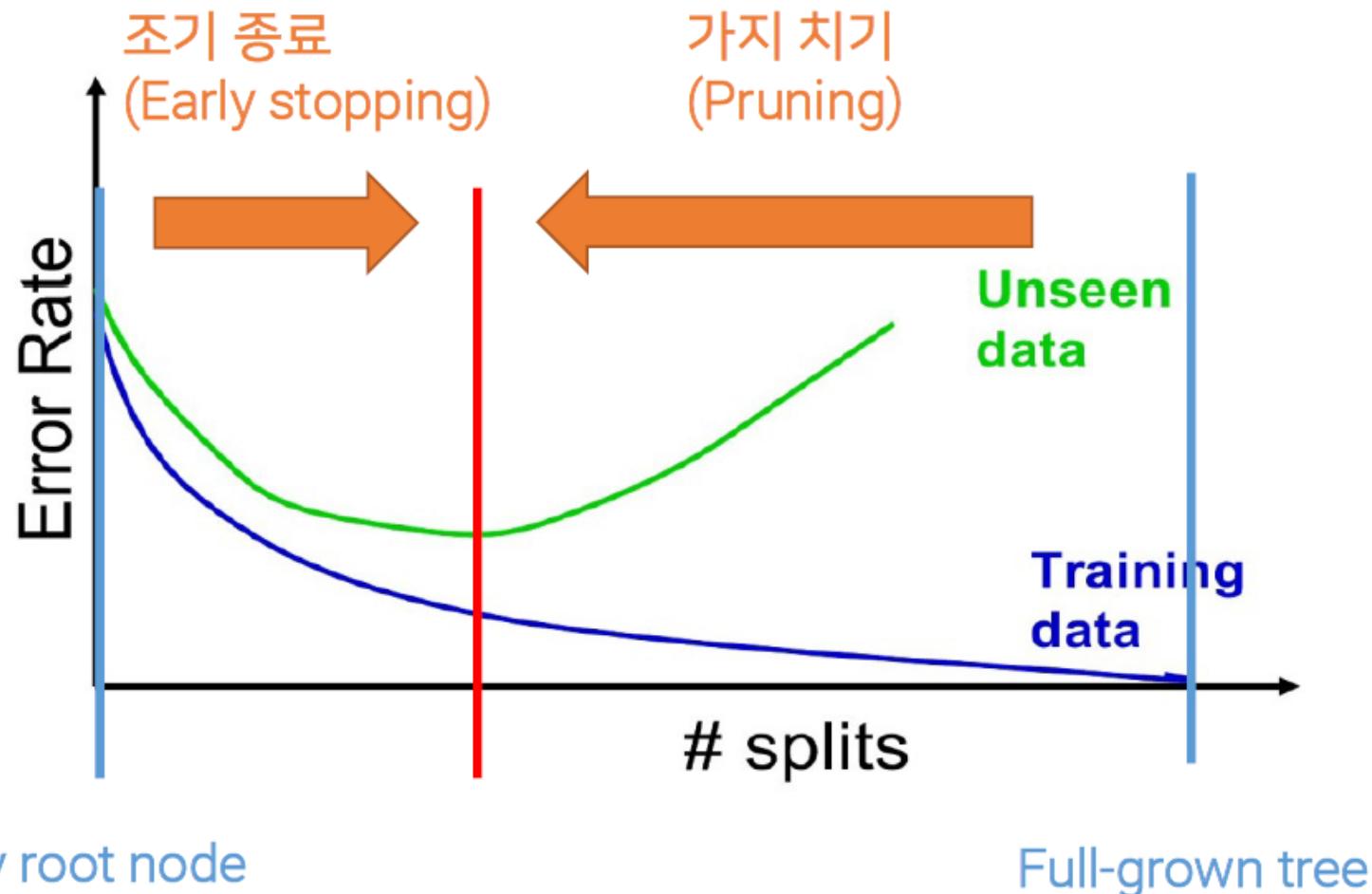
□ 최적의 Decision Tree는?



모델 학습에 쓰이지 않은
데이터에 대해 에러가 작은 때가
“최적”일 것

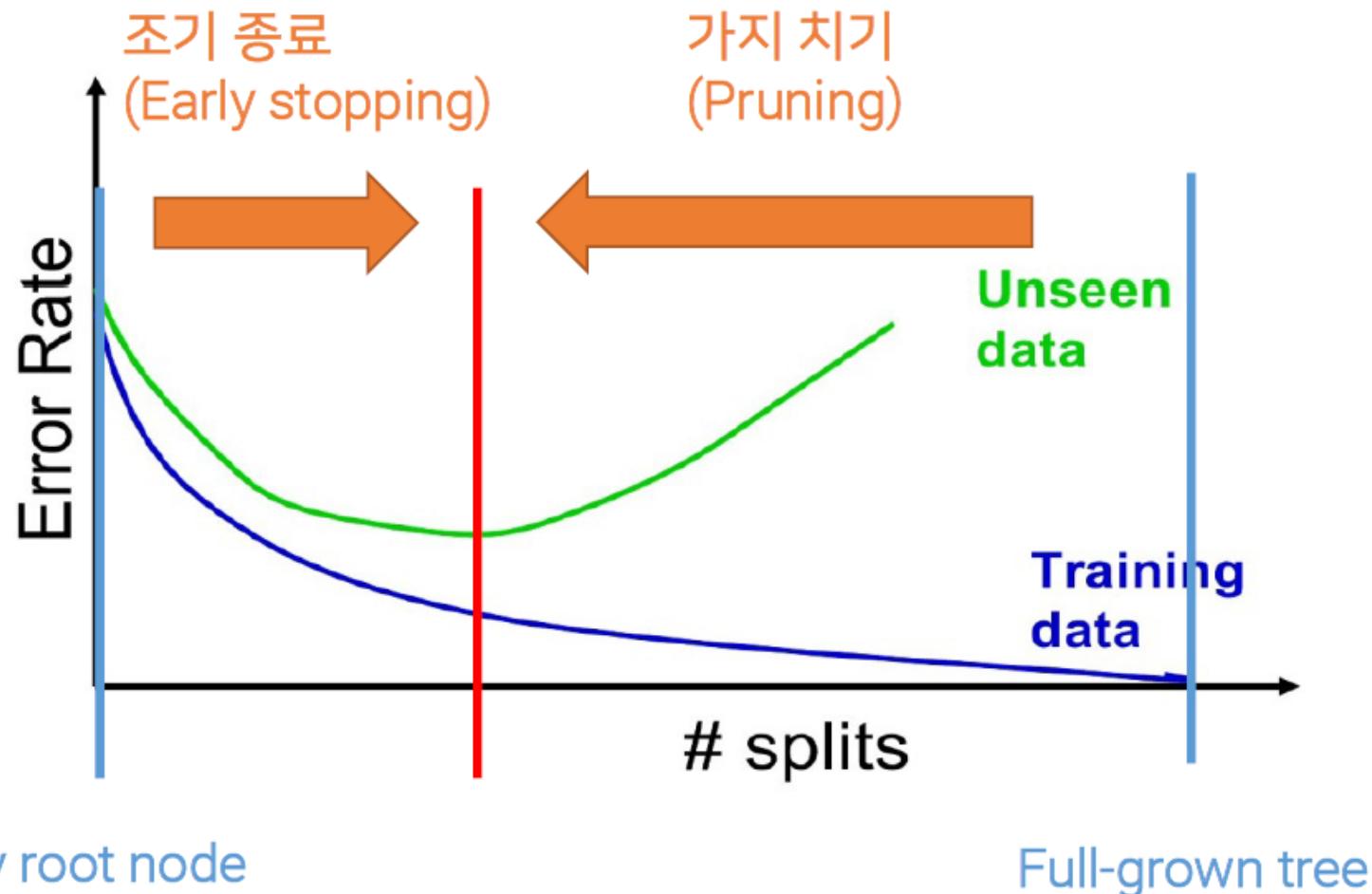
Decision Tree

□ 최적의 Decision Tree는?



Decision Tree

□ 최적의 Decision Tree는?



□ 과적합 방지: 조기종료

■ 조기 종료

- 특정 종료 기준을 만들고, 이를 학습에 사용하지 않은 데이터로 평가하여 최적의 나무 구조를 결정

■ 가능한 종료 기준

- 나무 깊이의 제한(*max_depth*)
- 잎 노드 수의 제한(*max_leafnodes*)
- 분기를 발생시키기 위한 최소한의 데이터 수(*min_samples_split*)
- 잎 노드가 되기 위한 최소한의 데이터 수(*min_samples_leaf*)
- 잎 노드의 최대 개수(*max_leaf_nodes*)

Decision Tree

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.tree import DecisionTreeClassifier
```



Decision Tree

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.tree import DecisionTreeClassifier
```

- Create an instance of the class

```
DTC = DecisionTreeClassifier(criterion='gini',  
                             max_features=None, max_depth=None,  
                             min_samples_split=2,  
                             min_samples_leaf=1)
```



Decision Tree

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.tree import DecisionTreeClassifier
```

Create an instance of the class

```
DTC = DecisionTreeClassifier(criterion='gini',  
                             max_features=None, max_depth=None,  
                             min_samples_split=2,  
                             min_samples_leaf=1)
```

- Fit the instance on the data and predict the expected value

```
DTC = DTC.fit(X_train, y_train)
```

```
y_predict = DTC.predict(X_test)
```

Tune parameters with cross-validation. Use **DecisionTreeRegressor** for regression.



Decision Tree

□ 주요 파라미터

▪ Criterion='gini'

- Gini : 경제학의 불평등 지수, 다양한 값을 가질 수록 평등하고 특정 값에 쏠릴 수록 불평등. 1에 가까울 수록 균일도가 높으므로 **지니계수가 높은 속성을 기준으로 분할**

$$G(t) = 1 - \sum_{i=1}^n p(i|t)^2$$

- Entropy : 주어진 데이터의 혼잡도. 서로 다른 값이 섞여 있으면 엔트로피가 높음. 정보이득지수는 1-entropy이며, 이 **정보이득지수가 높은 속성을 기준으로 분할**

$$H(t) = - \sum_{i=1}^n p(i|t) \log_2[p(i|t)]$$



Decision Tree

□ 주요 파라미터

▪ **min_samples_split**

- 노드를 분할하기 위한 최소한의 샘플 데이터 수로 과적합을 제어
- 디폴트는 2이고 작게 설정할수록 분할되는 노드가 많아져서 과적합 가능성 증가

▪ **min_samples_leaf**

- 잎 노드가 되기 위한 최소한의 샘플 데이터 수로 과적합을 제어
- 비대칭적 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우에는 작게 설정 필요



□ 주요 파라미터

■ max_features

- 최적의 분할을 위해 고려할 최대 피처 개수, 디폴트는 None으로 모든 피처 사용
- Int형이면 피처의 개수, float형이면 비율, 'sqrt'는 $\text{sqrt}(\text{전체피처수})$, 'log'는 $\text{log2}(\text{전체피처수})$

■ max_depth

- 트리의 최대 깊이를 규정, 디폴트는 None으로 완벽하게 분류될 때 까지 깊이를 min_samples_split보다 작아질 때까지 계속 증가시킴.



Quiz

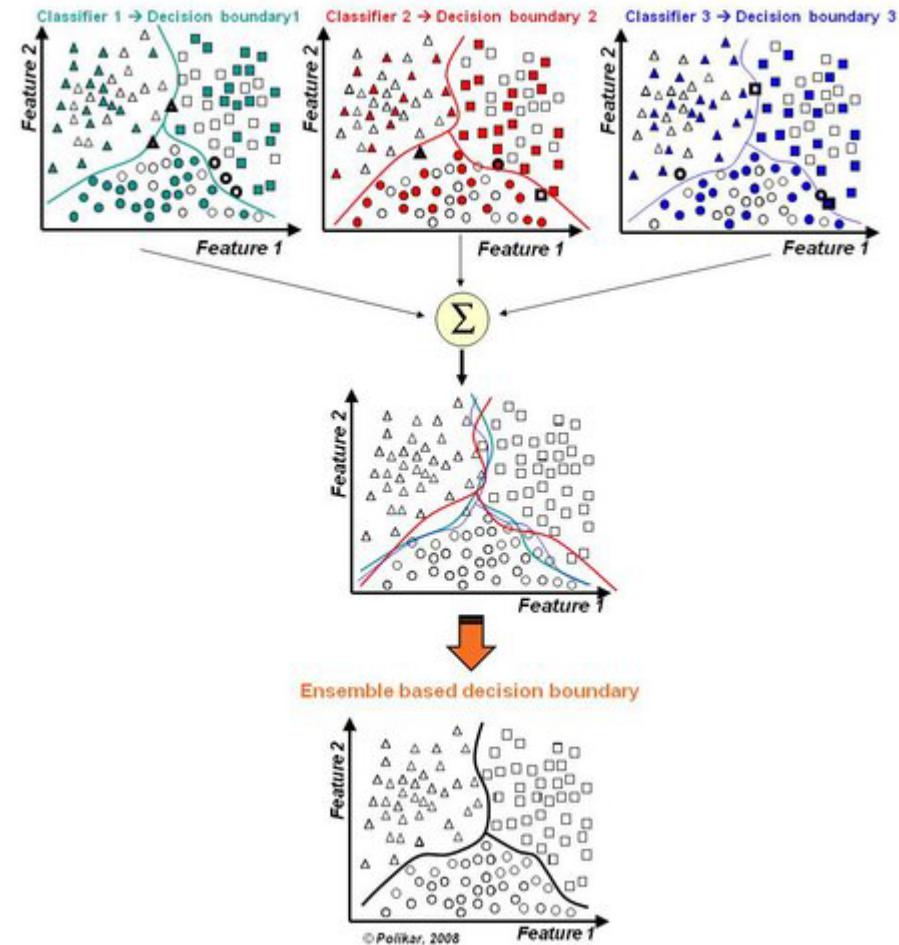
- Training 데이터에는 잘 맞으나 새로운 데이터에는 잘 안맞는 경우를 무엇이라고 하는가?
- 위의 경우를 Decision Tree에서 해결하는 방법이 아닌 것은?
 1. 학습을 조기 종료시킨다.
 2. `max_depth`를 작게 설정한다.
 3. `min_samples_split`을 작게 설정한다.
 4. `min_samples_leaf`를 크게 설정한다.



Bagging and Random Forrest

□ 양상블

- 단일모델이 아니라 여러 모델을 혼합하여 의사결정을 내리는 방법



Bagging and Random Forrest

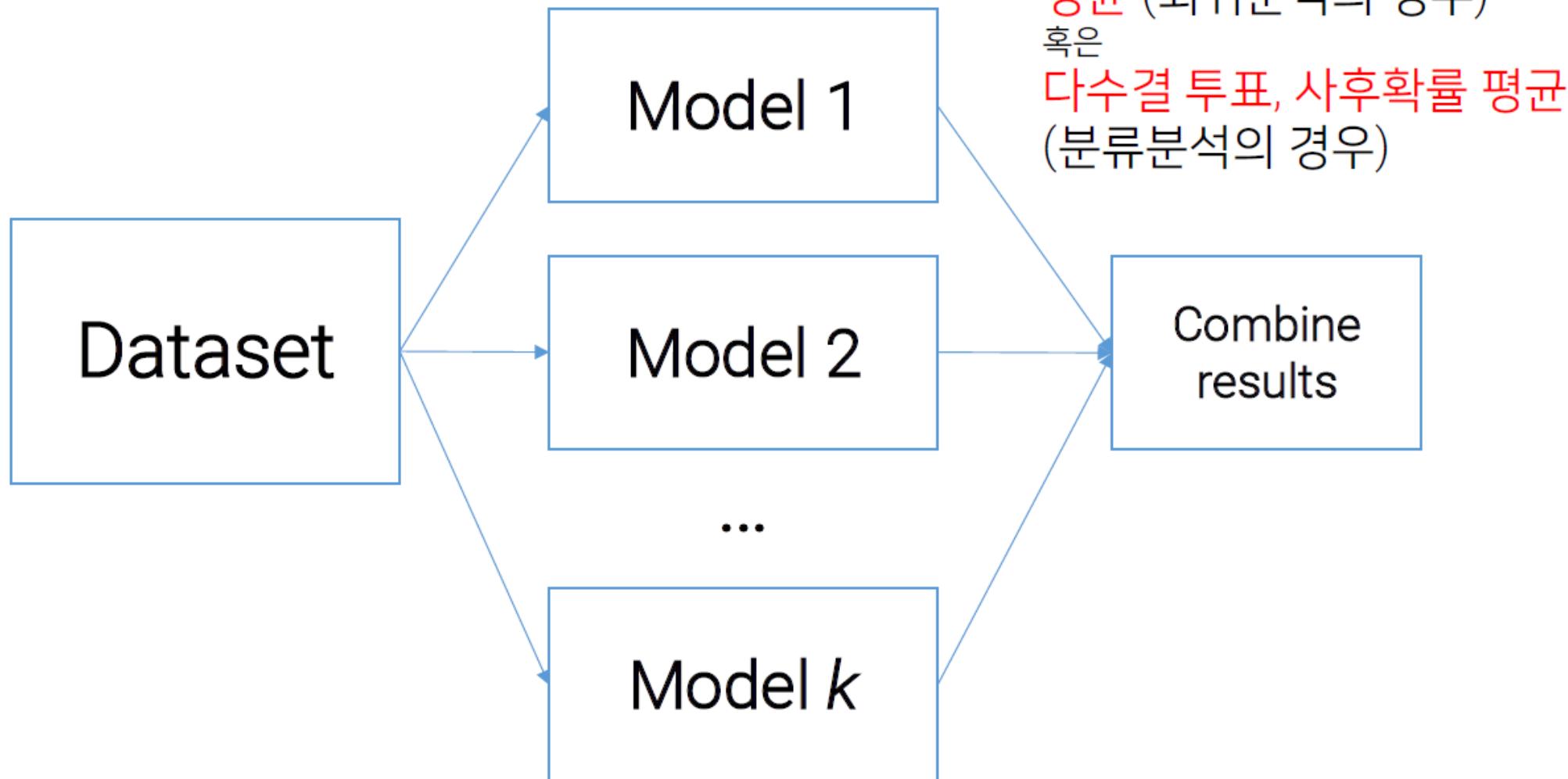
□ 양상블의 종류

- 단순/가중 평균(Simple/Weighted Average)
- 배깅(Bagging : Bootstrap Aggregation)
- 부스팅(Boosting)
- 스태킹(Stacking)



Bagging and Random Forrest

□ 단순/가중 평균



Bagging and Random Forrest

□ 여러 모델을 평균함으로써 분산을 줄임

▪ 표본 평균의 분산

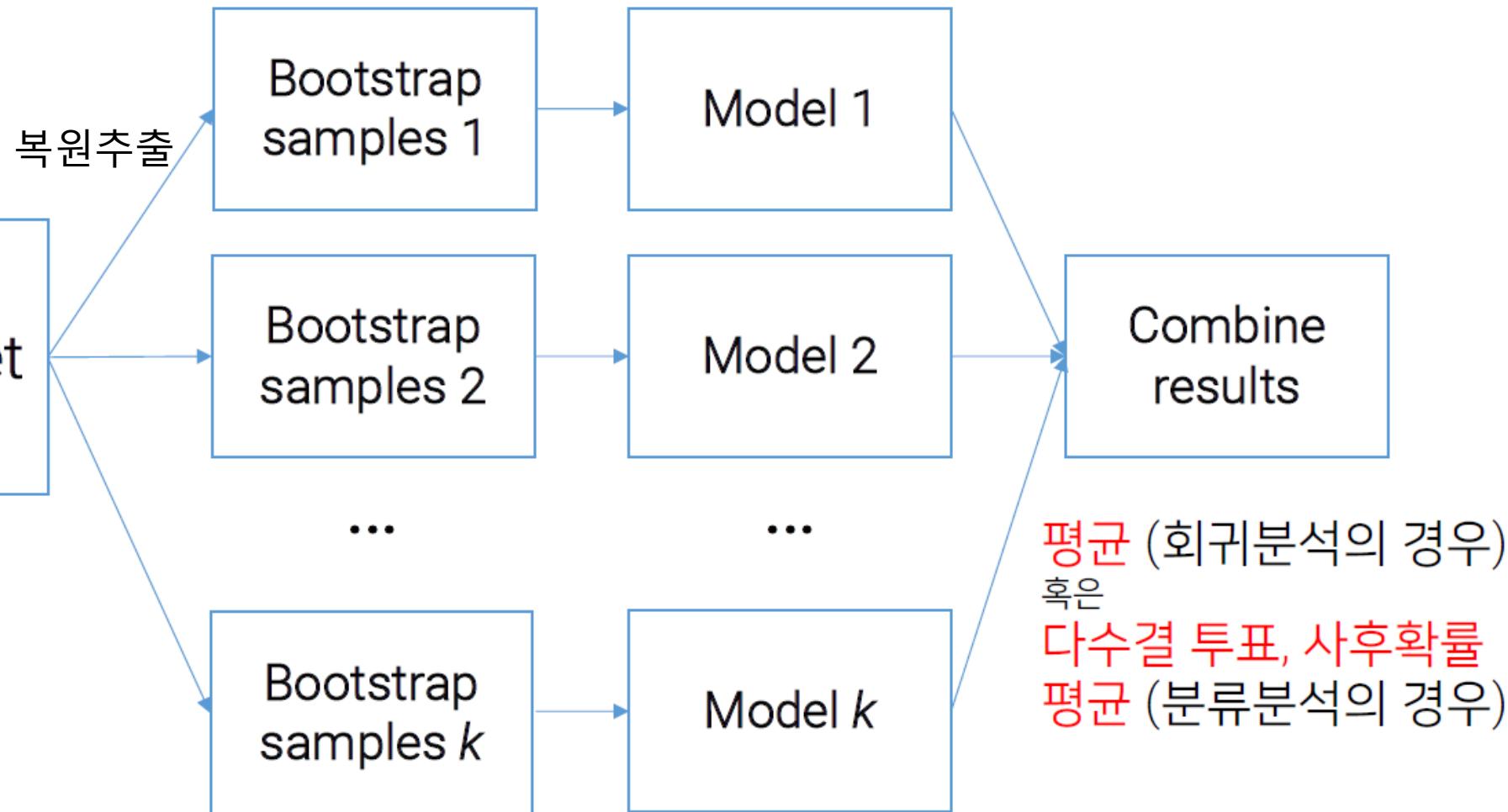
- 모집단에서 N만큼 표본을 추출하고 그 표본의 평균을 표본평균이라 할때 표본평균의 평균은 모평균과 동일하고, 표본평균의 분산을 모분산/N

$$\text{Var}(\bar{X}) = \frac{\text{Var}(X)}{N}$$

- 양상불된 모델의 분산이 각 개별 모델의 분산보다 낮을 것이라 기대
- 문제점 : 각 개별 모델을 만드는 데이터 셋이 동일하면 개별 모델이 비슷한 결과를 도출하게 됨
- 예)자문위원회 구성 시, 자문위원을 다양한 분야에서 뽑는 것이 좋은가?
한 분야에서만 뽑는 것이 좋은가?

Bagging and Random Forrest

□ Bagging



□ Bagging

▪ Bagging은 전문가 시스템과 유사

- 하나의 의안을 해결하기 위해 여러 분야의 전문가에게 의견을 구함

▪ 사용 가능한 알고리즘

- 이론상 모든 알고리즘이 사용 가능함
- Decision Tree 가 자연스럽게 많이 사용
- Full-grown tree 는 Bias는 매우 낮고, Variance는 매우 높기 때문에 여러 개의 Full-grown tree로 Bagging을 적용하여 Variance를 낮춤
- 그런데 큰 영향을 끼치는 Feature가 있으면 Decision Tree간 의존도, 유사성이 높음. → Random Forrest

Bagging and Random Forrest

□ Random Forrest

■ 2단계의 Randomization을 적용

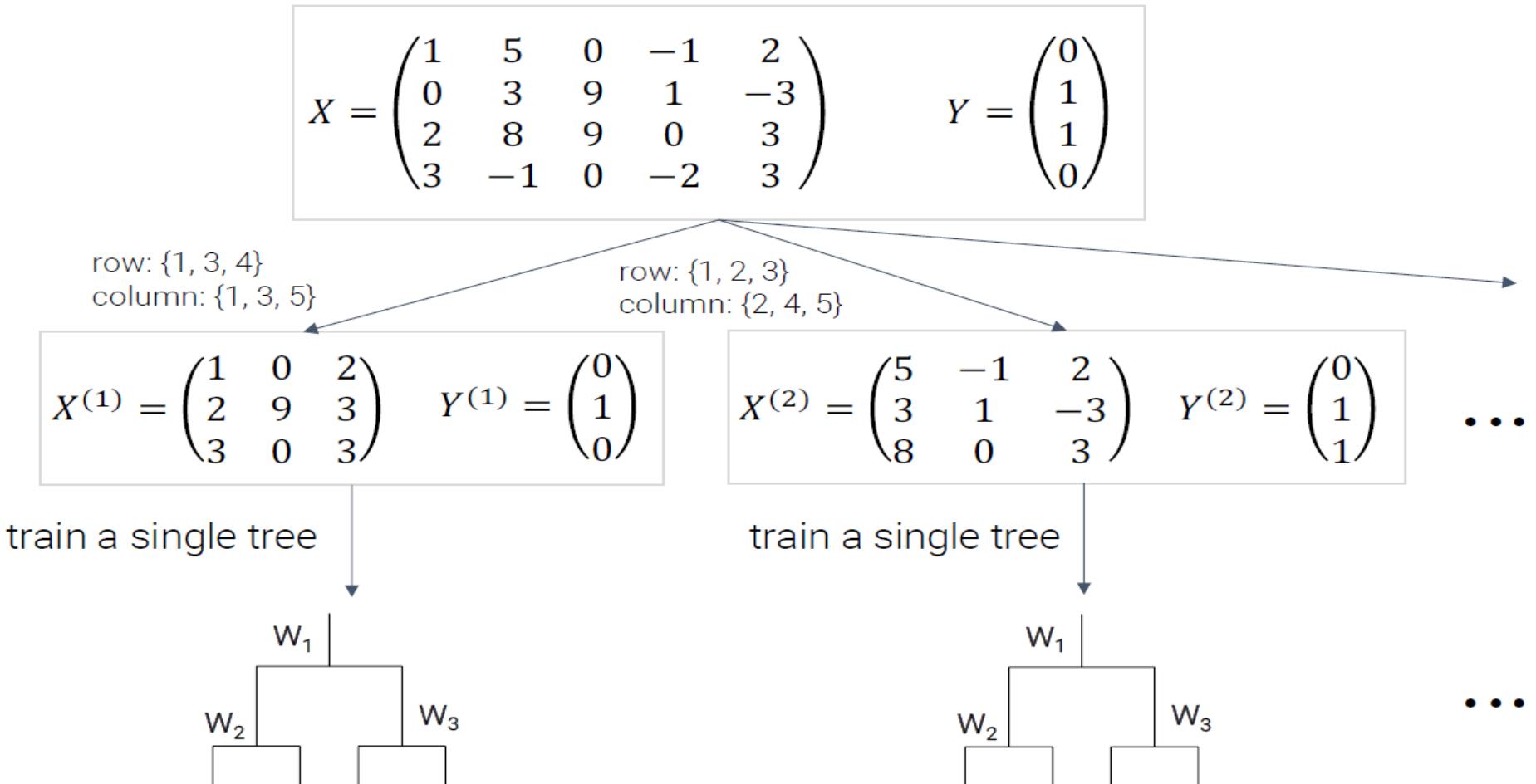
- 1단계 : Bagging, 데이터의 Random Sampling
- 2단계 : Feature(Classifier)의 Random Sampling
- → Classifier의 부분집합을 사용하여 Decision Tree를 생성하므로 Decision Tree가 서로 다른 특성을 갖게 됨

■ Bootstrapping training set + Bootstrapping features

- Bagging만을 사용했을 때보다 Tree들의 상호 관련성이 줄어듬

Bagging and Random Forrest

□ Random Forrest



Bagging and Random Forrest

□ Code syntax (코드 문법)

- Import the class containing the classification method
`from sklearn.ensemble import RandomForestClassifier`

Bagging and Random Forrest

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.ensemble import RandomForestClassifier
```

- Create an instance of the class

```
RC = RandomForestClassifier(n_estimators=100,  
                           max_features=10)
```

Bagging and Random Forrest

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.ensemble import RandomForestClassifier
```

- Create an instance of the class

```
RC = RandomForestClassifier(n_estimators=100, max_features=10)
```

- Fit the instance on the data and then predict the expected value

e

```
RC = RC.fit(X_train, y_train)  
y_predict = RC.predict(X_test)
```

- Tune parameters with cross-validation. Use **RandomForestRegressor** for regression.

Bagging and Random Forrest

□ 주요 파라미터

- **n_estimators=100**

- Forrest에 생성할 Tree의 개수

- **max_features=10**

- Tree의 분할 시 고려할 Feature의 개수
 - 정수 : Feature의 개수
 - 실수 : 전체 feature수에 대한 비율
 - 'auto', 'sqrt' : $\text{sqrt}(\text{전체 feature 수})$
 - 'log2' : $\log_2(\text{전체 feature 수})$

Quiz

- (O/X 문제) Full-grown tree 는 Bias는 매우 낮고, Variance는 매우 높다.

- Random Forrest의 2단계 Randomization의 설명으로 틀린 것은?
 1. 1단계는 샘플 데이터의 랜덤 샘플링이다.
 2. 2단계는 Feature의 랜덤 선택이다.
 3. Bagging만을 사용했을 때보다 Tree들의 상호 관련성이 줄어든다.
 4. 1단계는 Bias를 낮추고 Variance를 높인다.

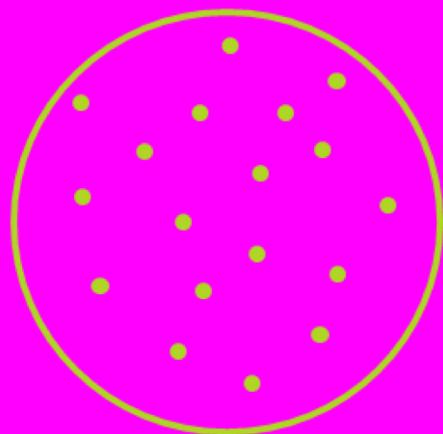


Boosting and XgBoost

□ Bagging과 Boosting

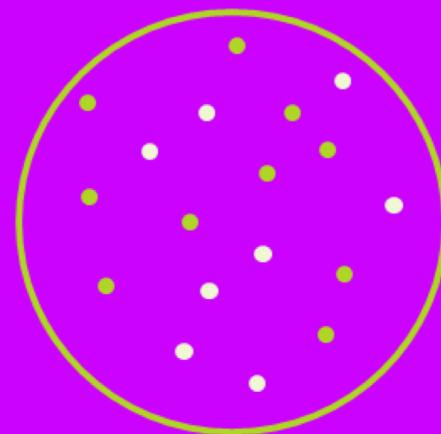
- Bagging은 복원추출을 통해 Random Sampling을 하고 Boosting은 가중치를 적용하여 복원추출 Random Sampling을 함

single



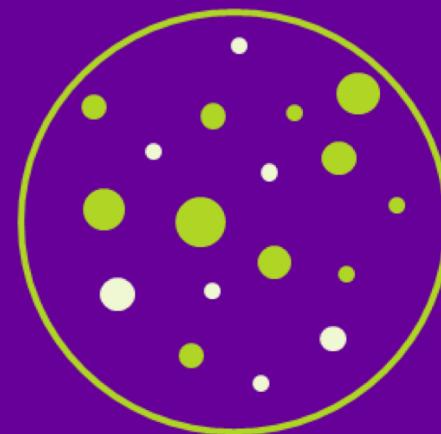
complete training set

bagging



random sampling with
replacement

boosting



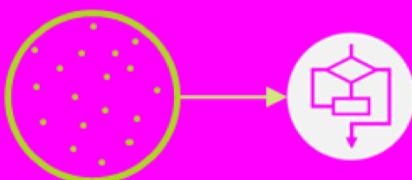
random sampling with
replacement
over weighted data

Boosting and XgBoost

□ Bagging과 Boosting

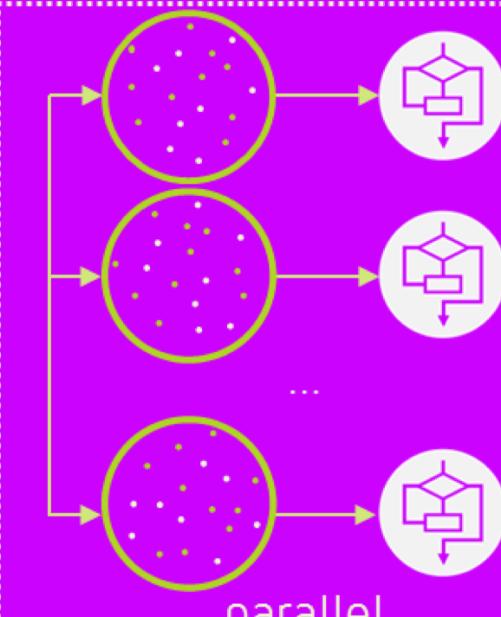
- Bagging은 각 모델이 병렬적으로 생성되지만 Boosting은 앞 모델의 결과를 반영하여 후속모델이 생성되므로 연속적임

single

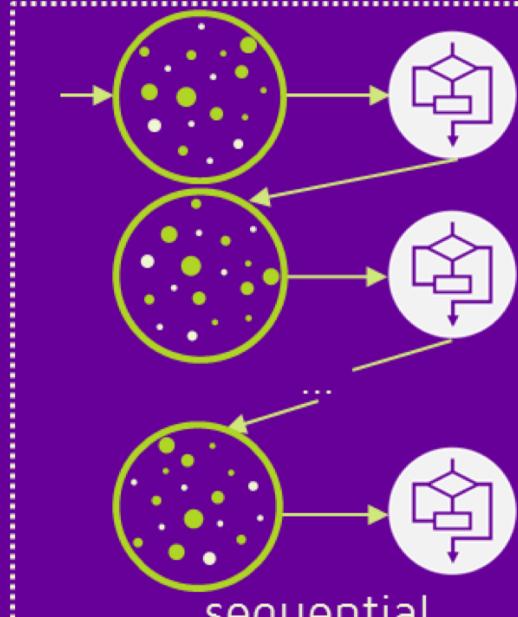


1 iteration

bagging



boosting



Boosting and XgBoost

□ Bagging과 Boosting

- Bagging은 각 모델의 결과를 단순 평균하여 최종 결과를 도출하지만
Boosting은 가중평균을 통해 최종 결과를 도출함

single



e

single estimate

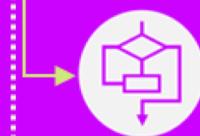
bagging



e1



e2



eN

simple average

boosting



e1



e2



eN

$$e = \sum_{i=1}^N w_i e_i$$

weighted average

Boosting and XgBoost

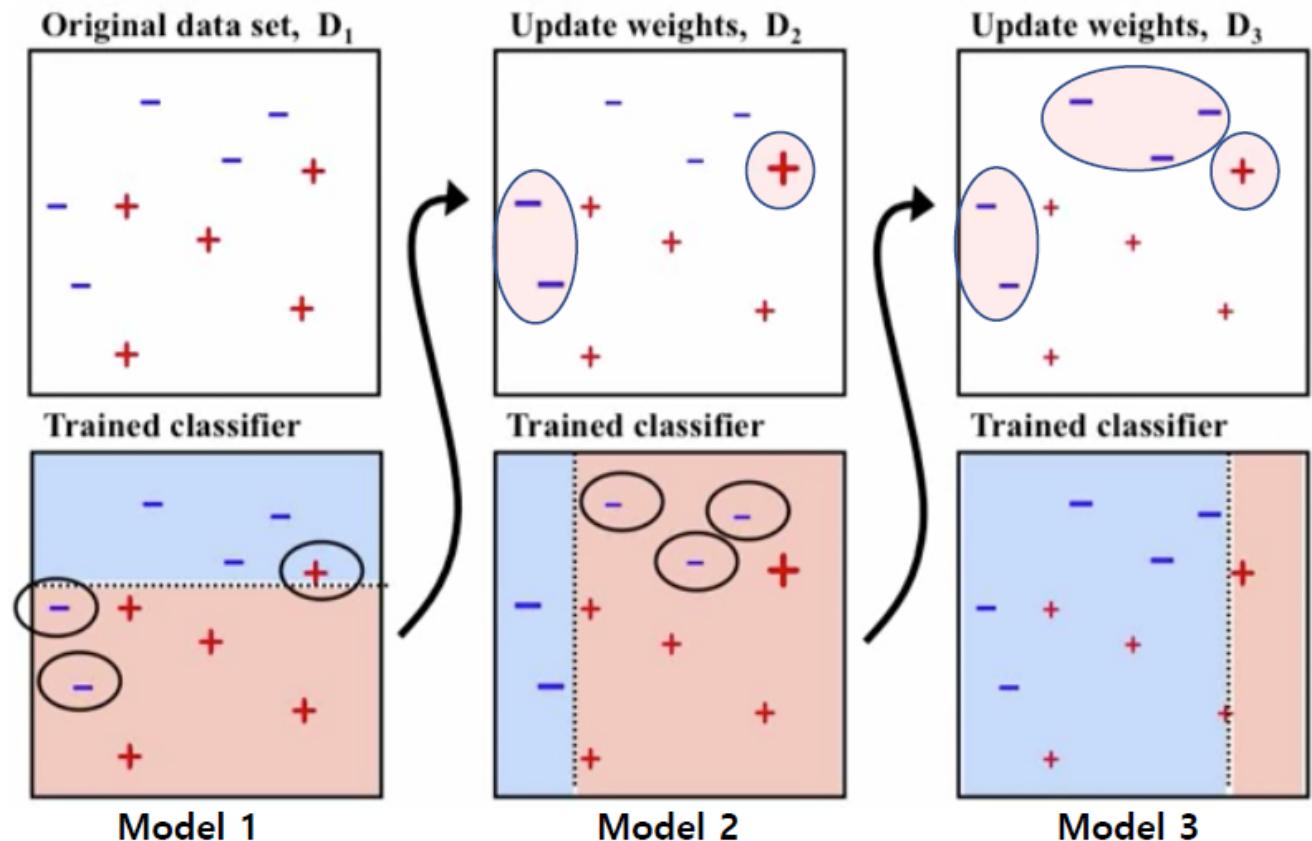
□ Boosting 알고리즘의 종류

| 알고리즘 | 특징 | 비고 |
|-----------|---|--|
| AdaBoost | <ul style="list-style-type: none">다수결을 통한 정답 분류 및 오답에 가중치 부여 | |
| GBM | <ul style="list-style-type: none">Loss Function에서 Gradient Decent를 이용하여 오답에 가중치 부여 | gradient_boosting.pdf |
| Xgboost | <ul style="list-style-type: none">GBM 대비 성능향상(GBM+분산 병렬처리)시스템 자원 효율적 활용 (CPU, Mem)Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용) | 2014년 공개 boosting-algorithm-xgboost |
| Light GBM | <ul style="list-style-type: none">Xgboost 대비 성능향상 및 자원소모 최소화Xgboost가 처리하지 못하는 대용량 데이터 학습 가능Approximates the split (근사치의 분할)을 통한 성능 향상 | 2016년 공개 light-gbm-vs-xgboost |

Boosting and XgBoost

□ 알고리즘

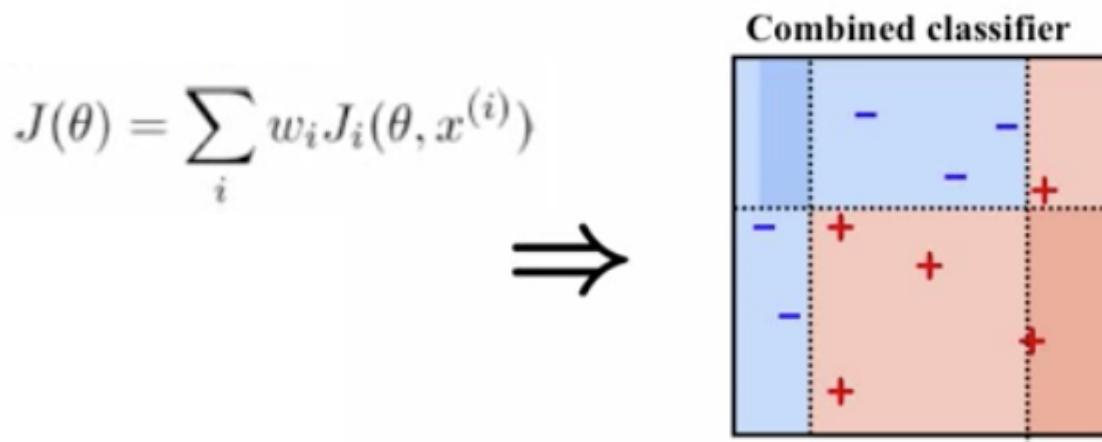
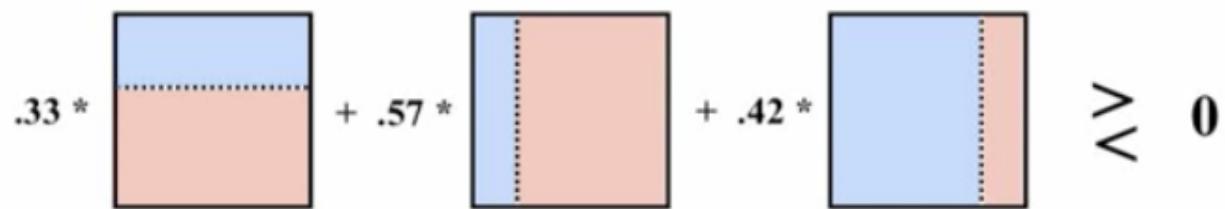
- Model 1에서 잘못 예측한 데이터에 가중치 부여
- Model2는 잘못 예측한 데 이터를 분류하는데 집중
- Model3는 Model1,2가 잘 못 예측한 데이터를 분류하는데 집중



Boosting and XgBoost

□ 알고리즘

- 각 모델별 가중치를 반영하여 가중 평균하여 최종 모델을 생성
- 오류(ε) = 오류 개수/전체, 가중치 = $\frac{1}{2} \ln \frac{1-\varepsilon}{\varepsilon}$



1-node decision trees
“decision stumps”
very simple classifiers

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.ensemble import GradientBoosting
```

Classifier



Boosting and XgBoost

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.ensemble import GradientBoostingClassifier
```

- Create an instance of the class

```
GBC = GradientBoostingClassifier(learning_rate=0.1,  
                                 max_features=10, subsample=0.5,  
                                 n_estimators=200)
```

Boosting and XgBoost

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from sklearn.ensemble import GradientBoostingClassifier
```

- Create an instance of the class

```
GBC = GradientBoostingClassifier(learning_rate=0.1,  
                                 max_features=10, subsample=0.5,  
                                 n_estimators=200)
```

- Fit the instance on the data and then predict the expected value

```
GBC = GBC.fit (X_train, y_train)
```

```
y_predict = GBC.predict(X_test)
```

- Tune with cross-validation. Use GradientBoostingRegressor for regression.

□ 주요 파라미터

▪ **Learning_rate=0.1**

- 학습을 진행할 때의 학습률, 순차적으로 오류값을 보정해 나갈때의 계수.
- 너무 작게 설정 : 예측성능이 높아지나 학습이 오래 걸림
- 너무 크게 설정 : 최소 오류값을 지나쳐서 예측성능이 떨어질 수 있음

▪ **N_estimators=200**

- Tree의 개수, 개수가 많을수록 예측 성능이 일정 수준까지는 높아짐

Boosting and XgBoost

□ 주요 파라미터

▪ Subsample=0.5

- 학습에 사용할 데이터의 샘플링 비율, 기본값은 1



Boosting and XgBoost

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from xgboost import XGBClassifier
```



Boosting and XgBoost

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from xgboost import XGBClassifier
```

- Create an instance of the class

```
XGB = XGBClassifier(learning_rate=0.1,  
                     max_features=10, subsample=0.5,  
                     n_estimators=200,  
                     )
```



Boosting and XgBoost

□ Code syntax (코드 문법)

- Import the class containing the classification method

```
from xgboost import XGBClassifier
```

- Create an instance of the class

```
XGB = XGBClassifier(learning_rate=0.1,  
                    max_features=10, subsample=0.5,  
                    n_estimators=200)
```

- Fit the instance on the data and then predict the expected value

```
XGB = SGB.fit(X_train, y_train)
```

```
y_predict = XGB.predict(X_test)
```

- Tune with cross-validation. Use XGBRegressor for regression.

Quiz

□ Boosting에 대해 맞는 것을 모두 고르시오.

1. 이전 모델에서 잘못 예측한 데이터에 대해 가중치를 적용하여 모델을 생성한다.
2. Gradient Decent Boosting은 손실함수로 Gradient Decent를 사용한다.
3. 최종 모델 생성 시 모델들을 단순 평균한다.
4. Learning Rate를 작게 설정하면 학습 시간이 오래 걸린다.



□ Support Vector Machine

- **Linear Support Vector Machine**

1. Logistic Regression 와 Linear Support Vector Machine 비교
2. 알고리듬 이해
3. Regularization (제약 조건)
4. Code syntax (코드 문법)

- **Non-linear Support Vector Machine**

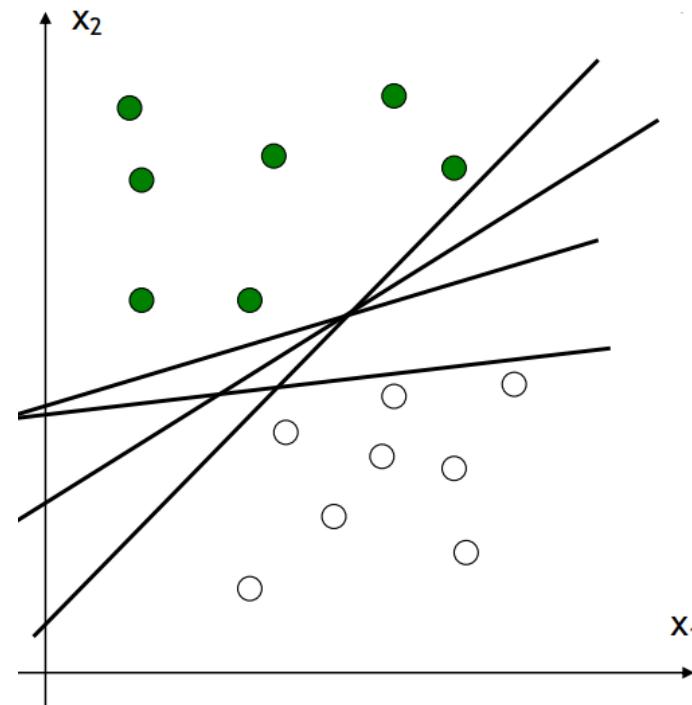
1. 알고리듬 이해
2. RBF : 가우스 함수 변환을 이용한 차원 변경
3. Code syntax (코드 문법)

Linear SVM/SVC

□ Logistic Regression 와 Linear Support Vector Machine 비교

- Logistic Regression 의 경우 두 그룹의 분류를 위해 어느 것이 가장 적합한 것일까?
- 그 이유는??

$$y_{\beta}(x) = \frac{1}{1+e^{-(\beta_0 + \beta_1 x + \varepsilon)}}$$

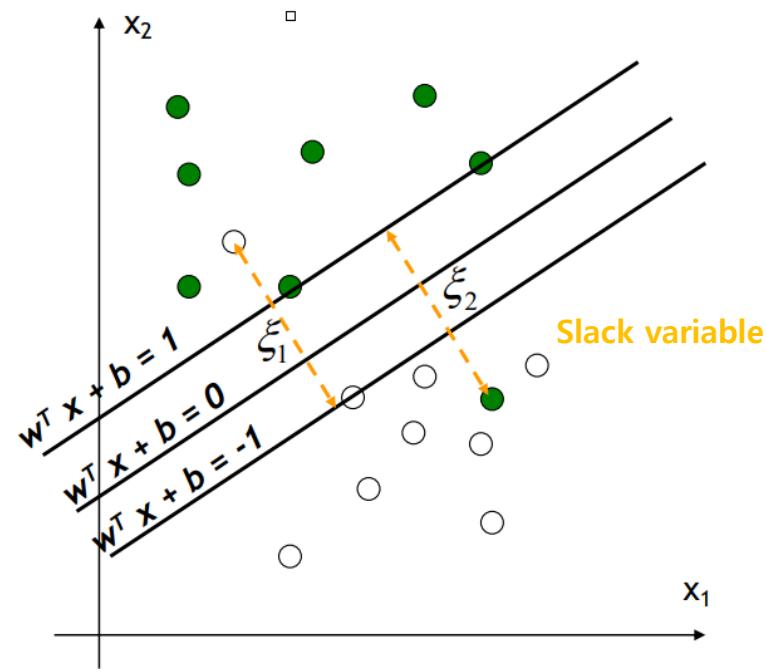
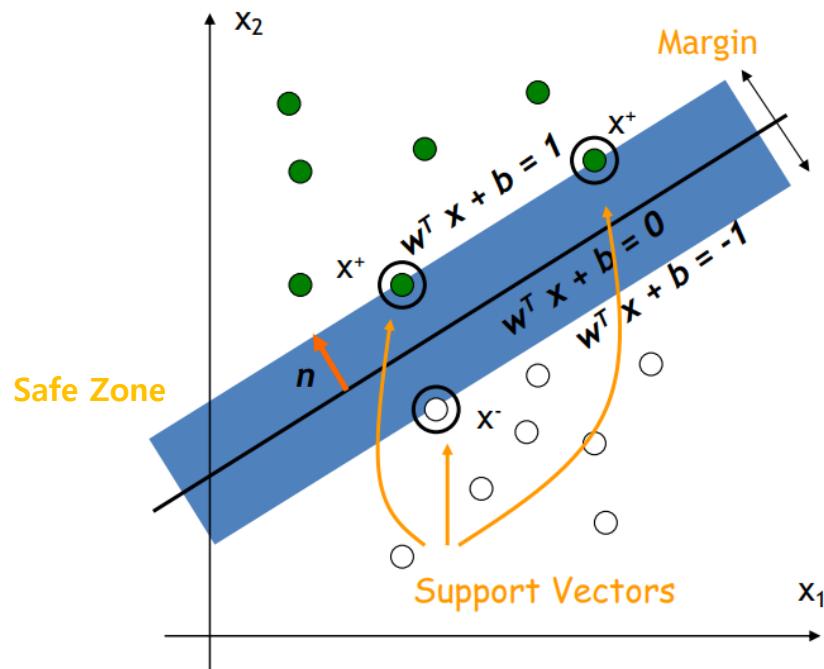


두 그룹의 가장 가장자리 데이터로 부터 가장 멀리 떨어진 직선을 선택하는 것이 최선
-> Margin 이 최대인 직선 찾기

Linear SVM/SVC

□ Linear Support Vector Machine 알고리듬 이해

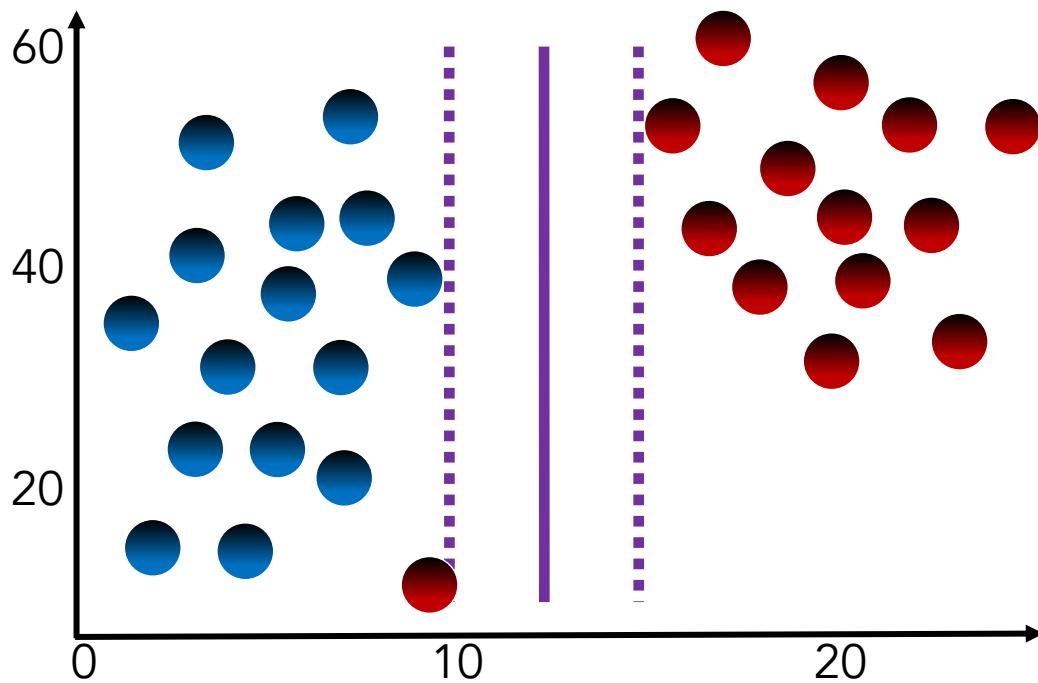
- Linear Support Vector Machine 이해를 위한 개념 정리
 - 1. Support vector
 - 2. Margin
 - 3. Safe Zone
 - 4. Slack variable



Linear SVM/SVC

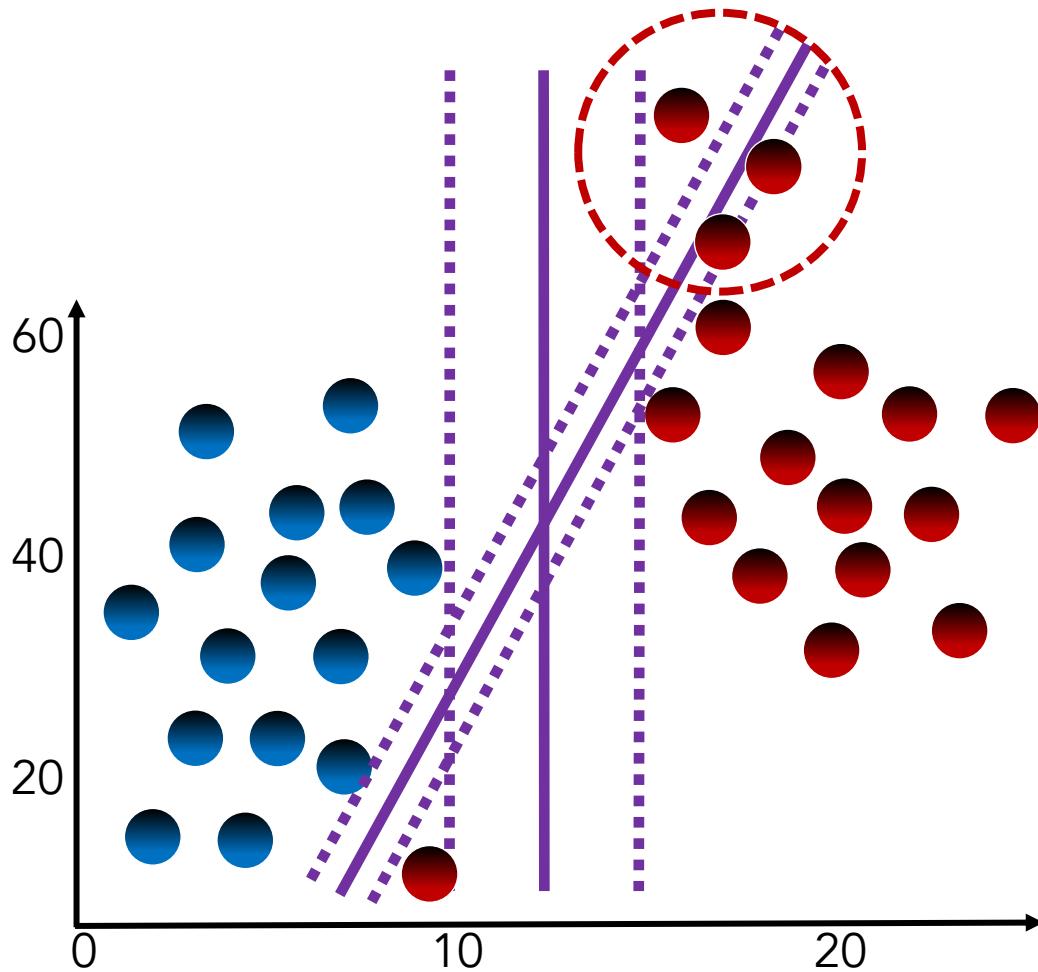
□ Regularization (제약 조건)

- Overfitting (과적합) 문제를 어떻게 해결할 것인가?
 1. Slack variable 을 이용해서 Cost function 에 항을 추가
 2. Hyperparameter (C 값) 를 조절하여 과적합 방지



Linear SVM/SVC

□ 하이퍼 파라미터 C 를 이용한 Regularization (제약 조건) 조절



하이퍼파라미터 C 로 Margin
컨트롤
-> C 가 크면 Margin이 작은
쪽으로 학습하므로 과적합 가
능성 증가

Linear SVM/SVC

□ Code syntax (코드 문법)

- Linear SVM Classifier 를 sklearn 으로 부터 불러 들임 (import)

```
from sklearn.svm import LinearSVC
```

- 클래스의 인스턴스 생성, 및 하이퍼 파라미터 세팅

```
LinSVC = LinearSVC(penalty='l2', C=10.0)  
          regularization  
          parameters
```

- 적합 (fit) 및 예측(predict) 함수 실행

```
LinSVC = LinSVC.fit(X_train, y_train)
```

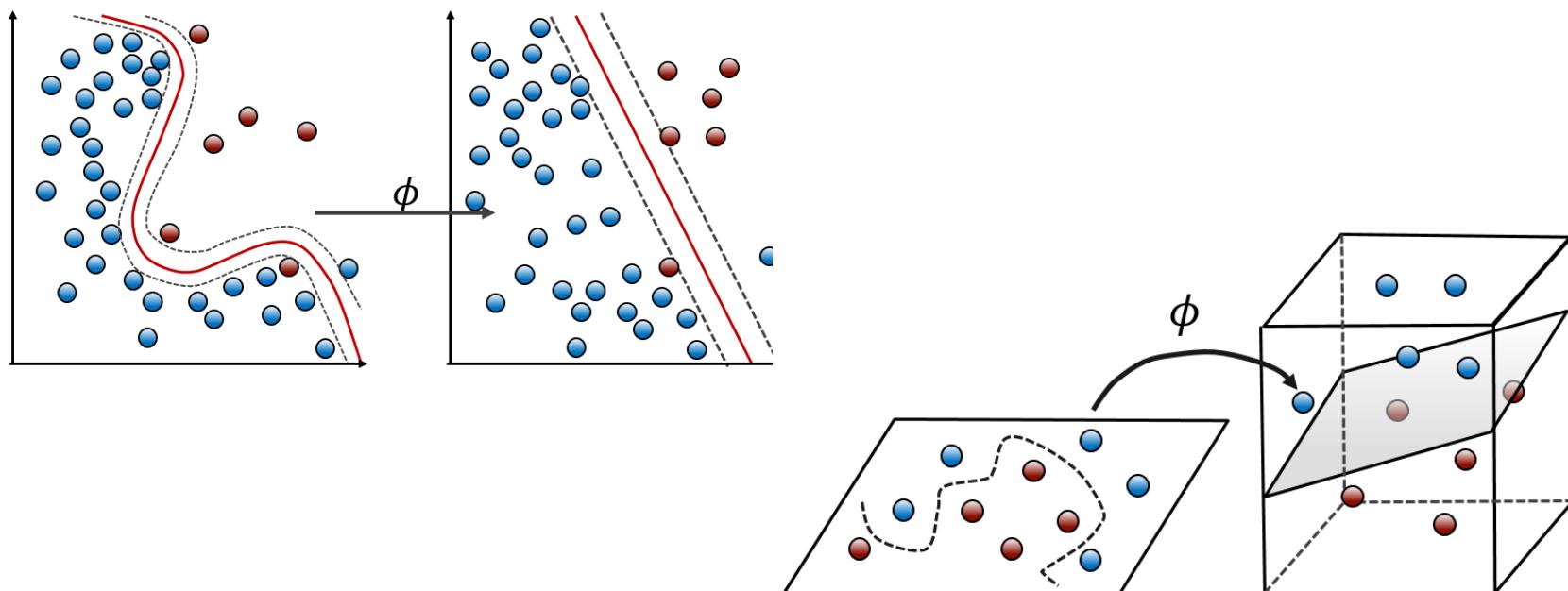
```
y_predict = LinSVC.predict(X_test)
```

- 교차 검증 (cross-validation) 방법을 사용하여 최적의 하이퍼 파라미터 선택

Non-linear SVM/SVC

□ Non-linear Support Vector Machine 알고리듬 이해

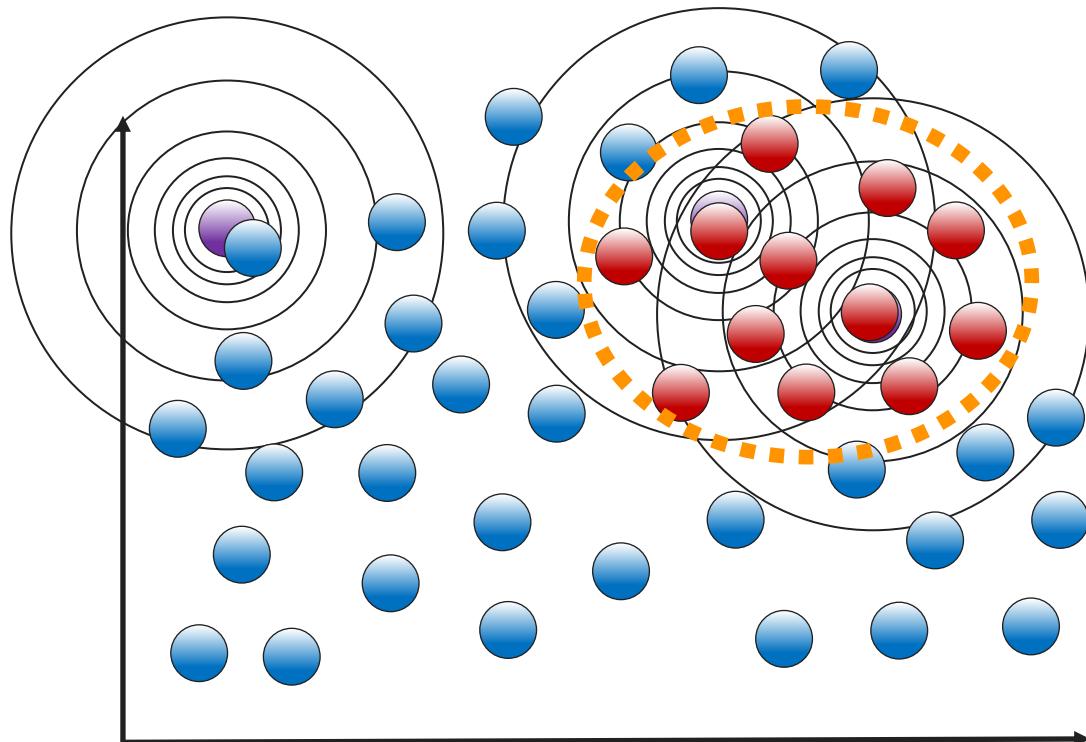
- Non-linear Support Vector Machine 이해를 위한 개념 정리
 1. Non-linear 특성을 보이는 데이터 분류를 어떻게 linear 특성을 보이도록 할 것인가?
 2. 차원을 변환하여 관찰하면 분석이 쉬운 linear 형태로 분류 가능
 3. Kernel Trick (커널 변환) 을 이용해서 높은 차원으로 변환



Non-linear SVM/SVC

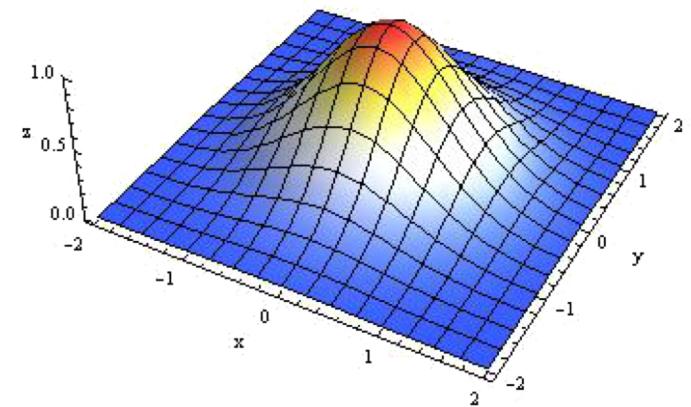
- RBF (Radial Basis Function) : 가우스 함수 변환을 이용한 차원 변경

- RBF (Radial Basis Function)



Radial Basis Function
(RBF) Kernel

$$a_1(x^{obs}) = \exp \left[-\frac{\sum (x_i^{obs} - x_i^{Pulp Fiction})^2}{2\sigma^2} \right]$$



Non-linear SVM/SVC

□ Code syntax (코드 문법)

- Non-linear SVM Classifier 를 sklearn 으로 부터 불러 들임 (import)

```
from sklearn.svm import SVC
```

- 클래스의 인스턴스 생성, 및 하이퍼 파라미터 세팅

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

set kernel and a
ssociated coeffi
cient (gamma)

- 적합 (fit) 및 예측(predict) 함수 실행

```
rbfSVC = rbfSVC.fit(X_train, y_train)
```

```
y_predict = rbfSVC.predict(X_test)
```

"C" is penalty a
ssociated with t
he error term

- 교차 검증 (cross-validation) 방법을 사용하여 최적의 하이퍼 파라미터 선택

Voting Classifier

□ Voting Classifier

- **Voting Classifier**

1. 알고리듬 이해
2. Code syntax (코드 문법)

http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

Voting Classifier

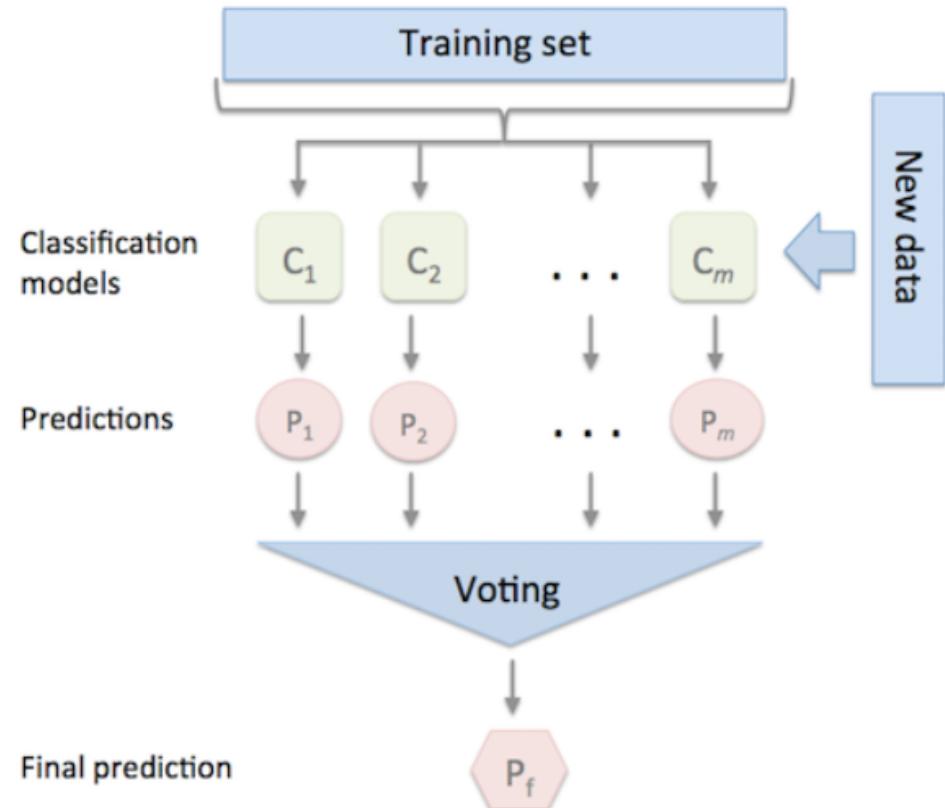
□ 알고리듬 이해

▪ 개념

1. 몇개의 다른 classifier 를 대상으로 학습한 결과값을 비교한 후 다수결이나 평균값등의 방법을 적용하여 최적 모델을 선택하는 방법
2. 사용할 classifier 는 각 모델간 성능의 차이가 심하게 나지 않을 경우에 한해 적용하는 것이 의미가 있음

▪ Voting 방법

1. "hard" / "soft" 한 투표 방식으로 나뉨
2. Hard voting : 가장 많이 나온 Class label 을 비교하여 선택 (다수결 원칙)
3. Soft voting : 결과로 나온 Class label 의 가중평균치 확률 (Weighted Average)을 비교하여 선택



http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

Voting Classifier

□ Code syntax (코드 문법)

- Voting Classifier 를 sklearn 으로 부터 불러 들임 (import)

```
from sklearn.svm import VotingClassifier
```

- 클래스의 인스턴스 생성, 및 하이퍼 파라미터 세팅

```
vClf = VotingClassifier(estimators=['Classifier1', '2', ...],  
voting='hard')
```

- 적합 (fit) 및 예측(predict) 함수 실행

```
vClf = vClf.fit(X_train, y_train)  
y_predict = vClf.predict(X_test)
```

- 교차 검증 (cross-validation) 방법을 사용하여 최적의 하이퍼 파라미터 선택



Voting option 선택

ExtraTree Classifier

□ ExtraTree Classifier

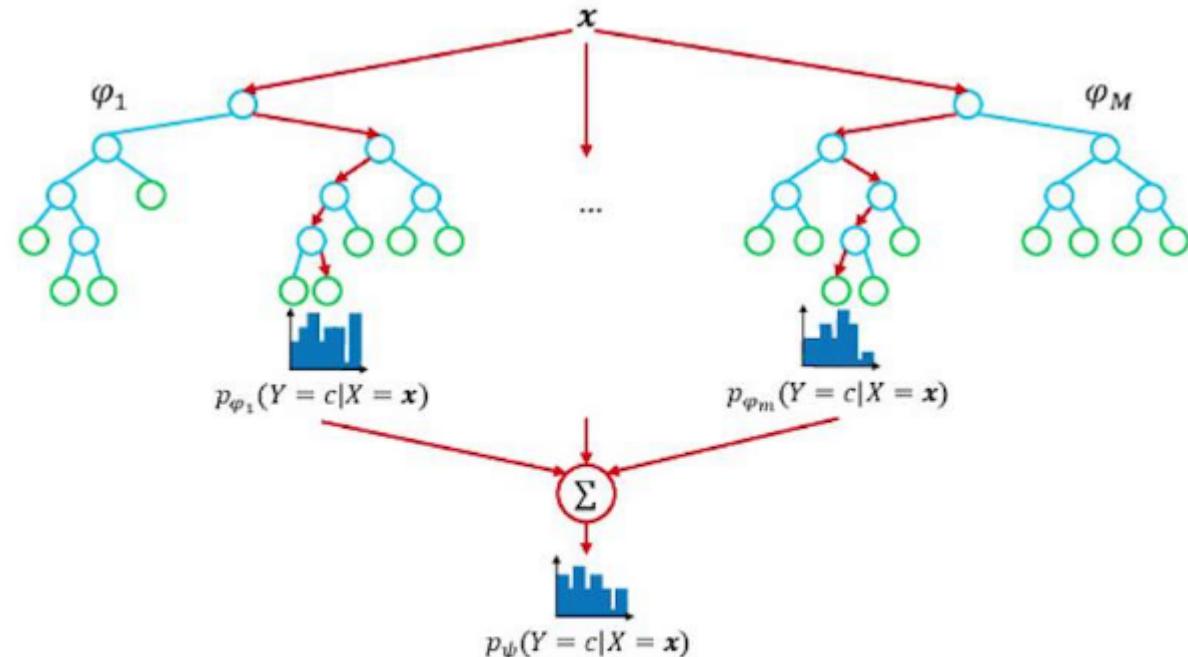
- ExtraTree Classifier
 - 1. 알고리듬 이해
 - 2. Code syntax (코드 문법)

ExtraTree Classifier

□ ExtraTree Classifier

▪ 알고리듬 이해

1. 추가적인 랜덤 조건을 Random Forest에 부여하는 것
2. Feature 를 랜덤으로 선택하면서 Split 도 랜덤하게 하고, decision boundaries 마저도 랜덤하게 적용하는 방법.



Randomization

- Bootstrap samples
- Random selection of $K \leq p$ split variables
- Random selection of the threshold

} Random Forests } Extra-Trees

ExtraTree Classifier

□ Code syntax (코드 문법)

- Voting Classifier 를 sklearn 으로 부터 불러 들임 (import)

```
from sklearn.svm import ExtraTreesClassifier
```

- 클래스의 인스턴스 생성, 및 하이퍼 파라미터 세팅

```
etclf = ExtraTreesClassifier(n_estimators=100, max_features=10)
```

- 적합 (fit) 및 예측(predict) 함수 실행

```
etclf = etclf.fit(X_train, y_train)
```

```
y_predict = etclf.predict(X_test)
```

- 교차 검증 (cross-validation) 방법을 사용하여 최적의 하이퍼 파라미터 선택

□ AdaBoost

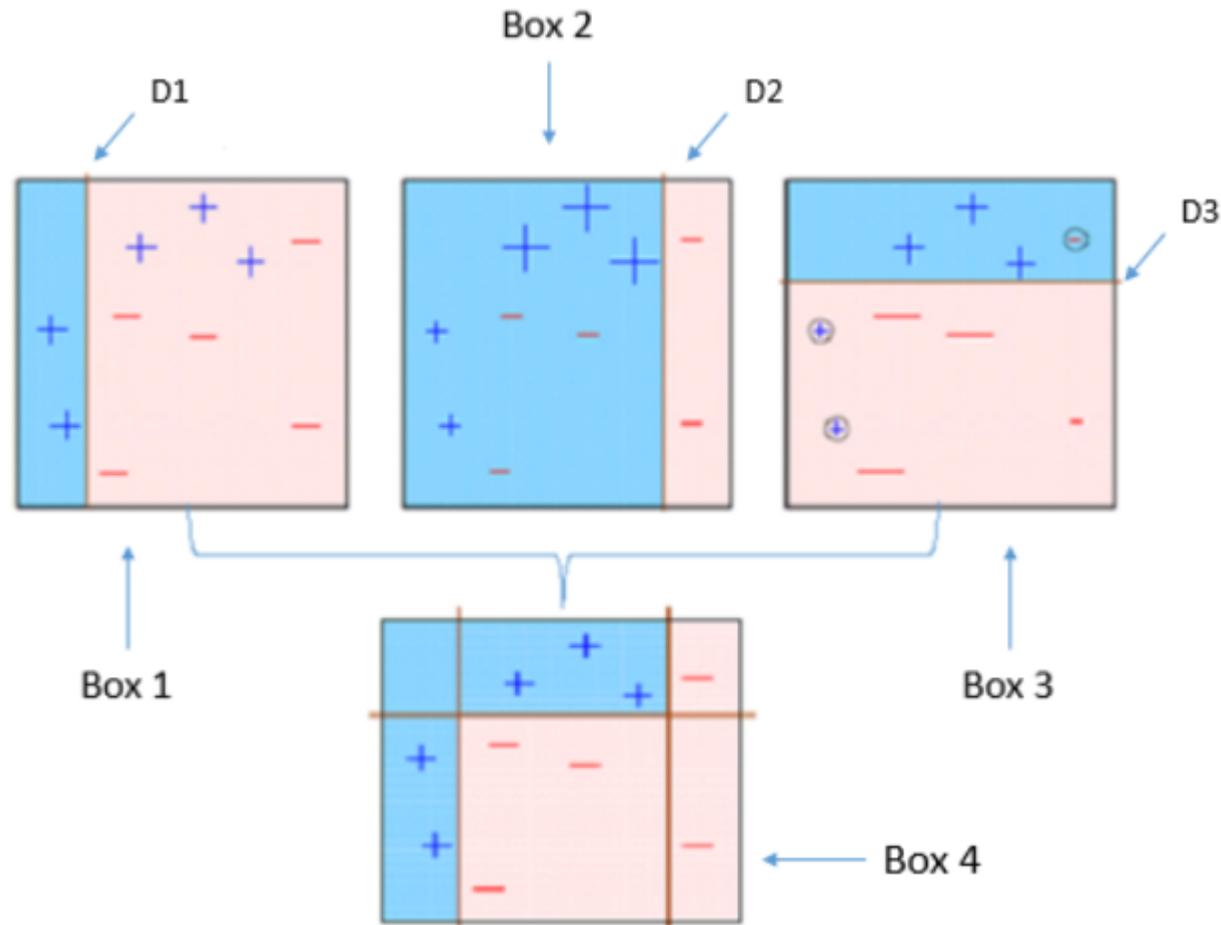
- AdaBoost
 - 1. 알고리듬 이해
 - 2. Code syntax (코드 문법)

AdaBoost

□ AdaBoost

▪ 알고리듬 이해

1. AdaBoost 는 기본적으로 decision tree 방식을 사용함.
2. 각각의 학습데이터를 사용하여 분류가 잘 안된 것으로 보일 경우 가중치를 증가. 분류가 잘 된것으로 보일 경우 가중치 감소.
3. 이과정을 반복하면서 최종 tree 구조를 찾아냄



□ Code syntax (코드 문법)

- Voting Classifier 를 sklearn 으로 부터 불러 들임 (import)

```
from sklearn.svm import AdaBoostClassifier
```

- 클래스의 인스턴스 생성, 및 하이퍼 파라미터 세팅

```
Clf = AdaBoostClassifier(n_estimators=50, learning_rate=1.0)
```

- 적합 (fit) 및 예측(predict) 함수 실행

```
Clf = Clf.fit(X_train, y_train)
```

```
y_predict = Clf.predict(X_test)
```

- 교차 검증 (cross-validation) 방법을 사용하여 최적의 하이퍼 파라미터 선택