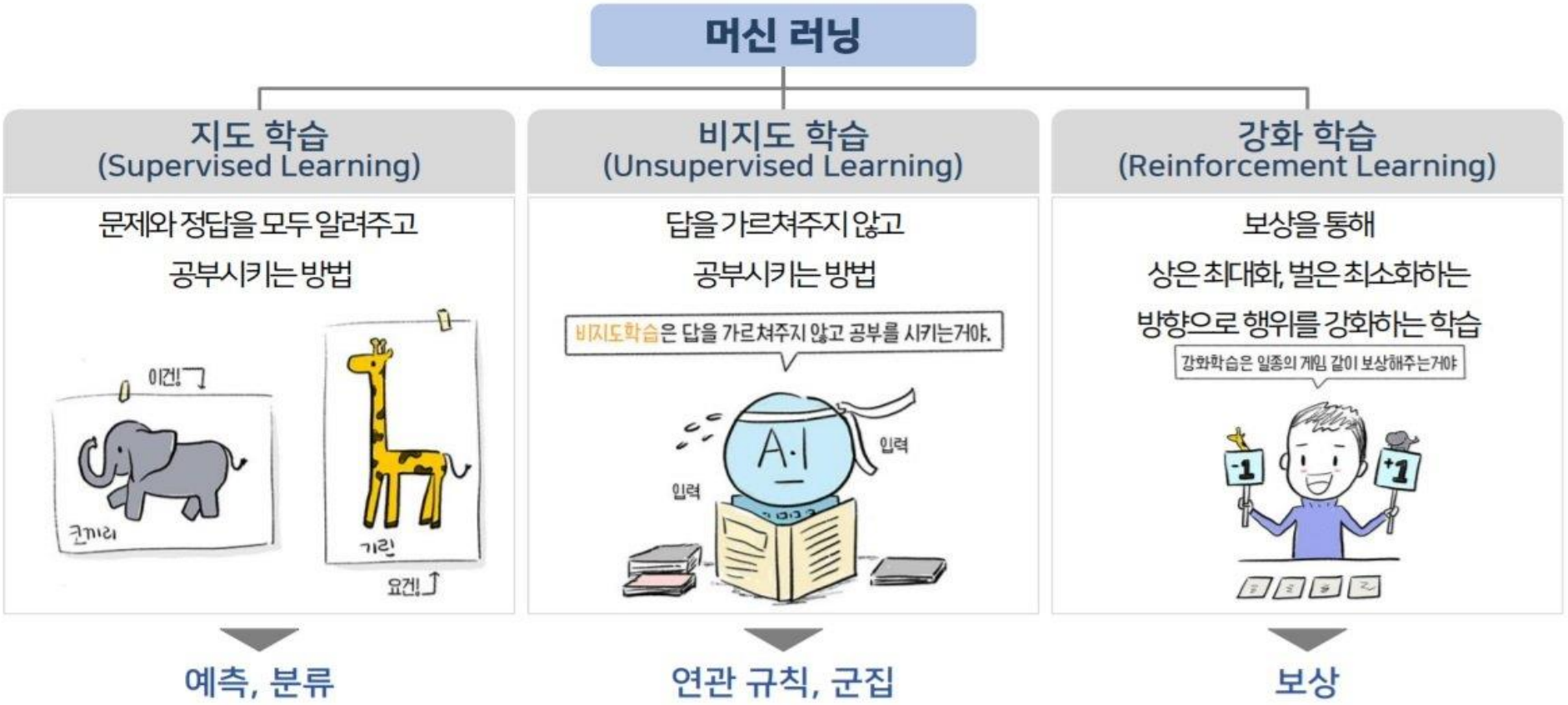
UNIT

05

분류 (Classification)

머신 러닝의 종류



https://steemitimages.com/0x0/https://cdn.steemitimages.com/DQmXQfK5MVgU63WQYRcBaZFXhsLK5xxYZ73fNpz51UrZmmF/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D.jpg

지도 학습 (Supervised Learning)

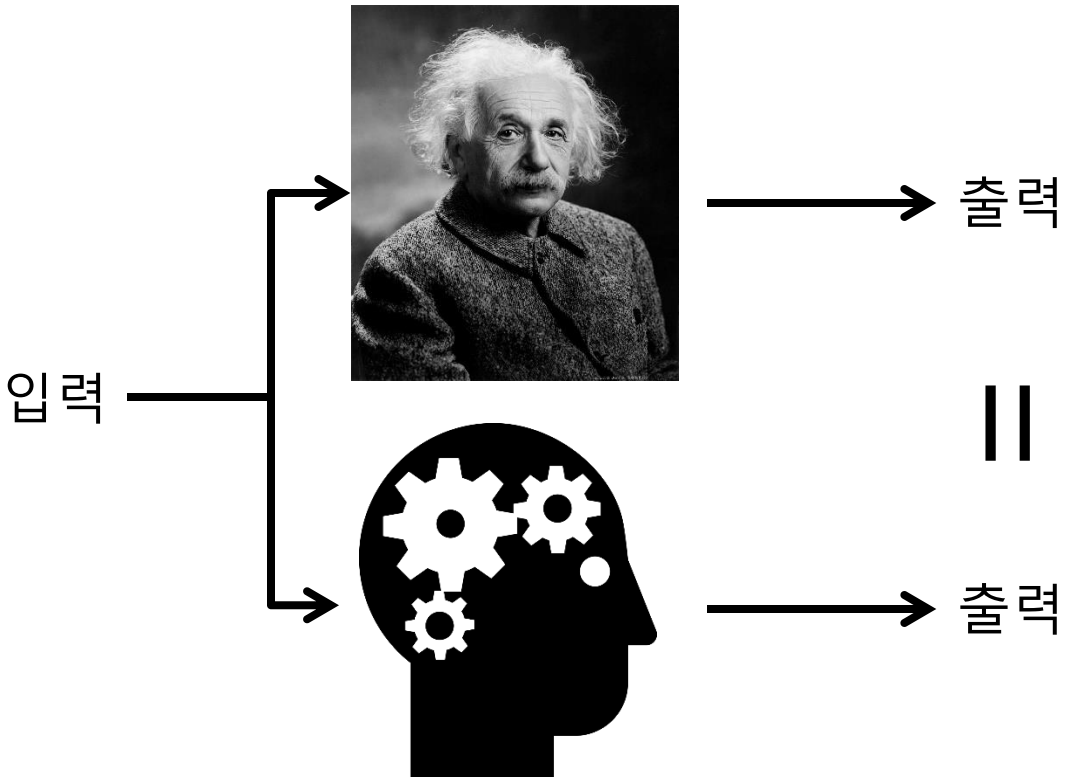
- 입출력 쌍의 예시들을 기반으로 입력을 출력으로 맵핑하는 함수를 학습하는 기계 학습 방법

  The machine learning task of learning a function that maps an input to an output based on example input-output pairs

  REF: Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Third Edition, Prentice Hall, 2010

- 라벨이 붙어 있는 학습 데이터(labeled training data)로부터 함수를 추론한다.

지도 학습 (Supervised Learning)



- 입출력 데이터 세트의 수학적 모델을 구한다.
- 적용 분야: 분류, 회귀 등

https://upload.wikimedia.org/wikipedia/commons/thumb/2/28/Albert_Einstein_Head_cleaned.jpg/484px-Albert_Einstein_Head_cleaned.jpg
https://pixabay.com/vectors/brain-business-credit-intelligence-1294854

지도 학습 (Supervised Learning)

- 입력에 대한 출력을 예측하는 것이 목표이다.
- 입력
  - 예측변수(Predictor)
  - 독립변수(Independent Variable)
  - 특성/특징(Feature)
- 출력
  - 반응(Response)
  - 종속변수(Dependent Variable)

변수 타입과 용어

- 양적(Quantitative) 변수
  - 수치 데이터
- 질적(Qualitative) 변수
  - 붓꽃 판별 예제: $G = \{Virginica, Setosa, Versicolor\}$
  - 손 글씨 숫자 예제: $G = \{0, 1, \cdots, 9\}$
  - 클래스에는 명시적인 순서가 없다.
  - 클래스를 표시하기 위해 숫자보다 라벨(Label)이 더 자주 사용된다.
  - 다른 말로 범주형(Categorical) 변수, 이산(Discrete) 변수, 요인(Factor)

변수 타입과 용어

- 입력에 대한 출력을 예측
  - 오늘과 어제의 특정한 대기 측정치
    → 내일의 오존 수준 예측
  - 손 글씨 숫자의 회색 값 → 클래스 라벨 예측
- 입력 타입의 종류
  - 양적 입력
  - 질적 입력
- 출력 타입의 종류
  - 양적 출력: 회귀(Regression)
  - 질적 출력: 분류(Classification)

변수 타입과 용어

- 세 번째 변수 타입
  - Small, Medium, Large와 같이
    순서 있는 범주형(Ordered Categorical)
  - 값 사이에 순서가 있지만 적절한 측량 개념이 없다.
- 질적 변수를 숫자로 표시
  - 코드를 부여하여 숫자로 표현한다.
  - 목표(Target)라고 부르기도 한다.
  - 성공과 실패, True 또는 False
    → 0과 1 또는 -1과 1의 단일 이진 숫자나 비트로 표현
  - 두 개 이상의 범주: 몇 가지 대안이 존재
    - 더미 변수(Dummy Variable) 사용
      K 수준의 질적 변수를 K개의 이진변수 또는 비트의 벡터로
      표현하고 한 번에 오직 하나만 ON

변수 타입과 용어

- 입력 변수를 기호 X로 표시
  - X가 벡터라면 구성요소는 X_j로 표시
- 양적 출력은 Y로 표시
- 질적 출력은 G(그룹)로 표시
- 관측된 값은 소문자로 표시
  - X의 i번째 관측 값: x_i
- 행렬은 굵은 대문자
- 학습 과제 예시
  - 입력 벡터 X에 대한 출력 Y의 예측 $\hat{Y}$을 구하라.

변수 타입과 용어

- 두 개의 클래스를 갖는 G에 대해
  - 양적인 출력을 Y라고 표기한다.
  - 예측 $\hat{Y}$은 [0, 1]의 범위를 가지며 $\hat{y} > 0.5$인지에 따라 $\hat{G}$ 클래스 라벨을 할당한다.
- 훈련 데이터(Training Data)
  - 사용가능한 관측치 집합 (x_i, y_i) 또는 $(x_i, g_i), i = 1, \ldots, N$

예측을 위한 단순한 두 가지 접근 방법

- 최소 제곱을 통한 선형 모델
- k-최근접이웃(k-nearest neighbor)

선형 모델과 최소 제곱

- 선형 모델은 통계학에서 지난 30여년 동안 중추적 역할을 하였으며 가장 중요한 도구 중 하나이다.
- 입력 벡터 $X^T = (X_1, X_2, \ldots, X_p)$가 주어졌을 때 모델을 통해 출력 Y를 예측

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

  - $\hat{\beta}_0$: 절편(Intercept), 편향(Bias)
  - 선형 모형을 벡터와 내적으로 표현할 수 있다.

$$\hat{Y} = \boldsymbol{X}^T \hat{\beta}$$

선형 모델과 최소 제곱

- 어떻게 선형 모델을 훈련 데이터 집합에 적합(Fit) 시킬까? → 최소 제곱이 하나의 방법
- 잔차제곱합(Residual Sum of Squares)을 최소화하는 계수 β를 구한다.

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i{}^T \beta)^2$$

$$RSS(\beta) = (y - \boldsymbol{X}\beta)^T (y - \boldsymbol{X}\beta)$$

- β에 대해 미분 → 정규방정식(Normal Equation)

$$\boldsymbol{X}^T (y - \boldsymbol{X}\beta) = 0$$

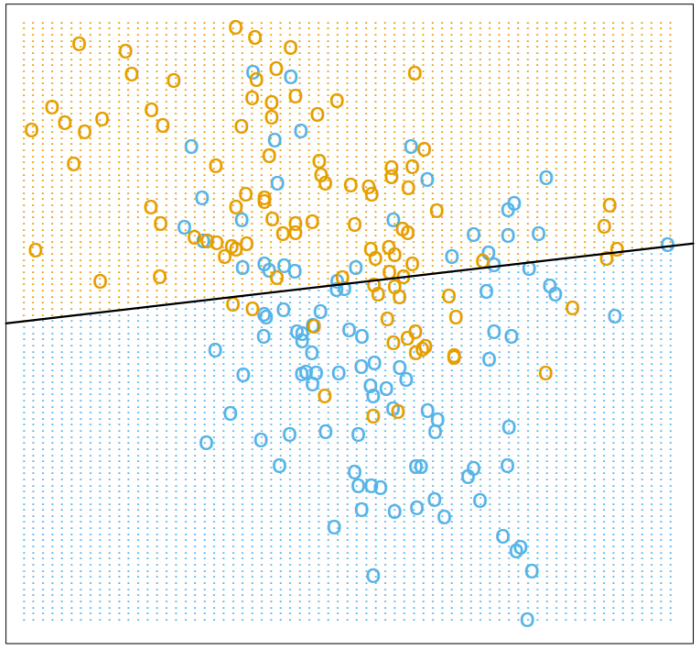
- $\boldsymbol{X}^T \boldsymbol{X}$가 정칙(Nonsingular) → 유일한 해

$$\beta = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y$$

선형 모델과 최소 제곱

- 분류에서 선형 모형의 예
  - 입력 X_1과 X_2
  - 출력 클래스 변수 G는 값 BLUE와 ORANGE를 가진다.
  - 응답 Y는 BLUE가 0, ORANGE가 1로 코딩되어 있다.
  - 적합된 값 $\hat{Y}$은 클래스 변수 $\hat{G}$으로 변환된다.

$$\hat{G} = \begin{cases} ORANGE & if\ \hat{Y} > 0.5 \\ BLUE & if\ \hat{Y} \leq 0.5 \end{cases}$$



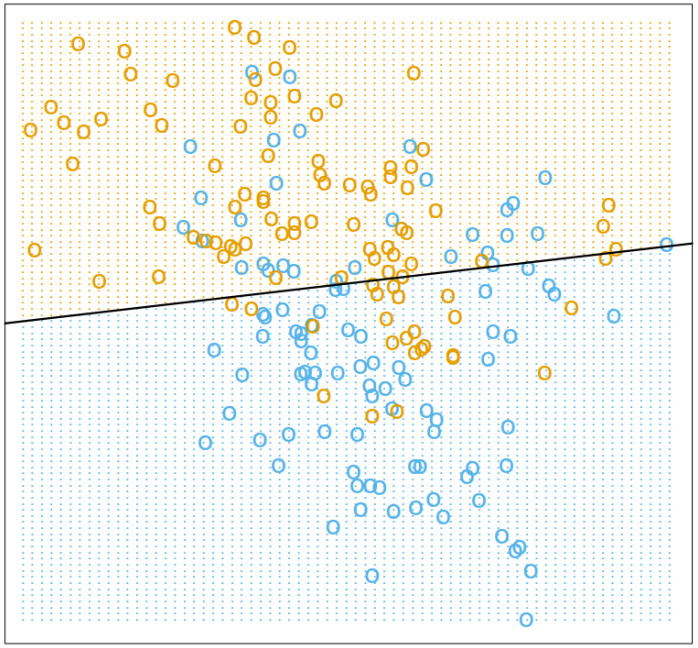
$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j = 0.5$$

$$\hat{\beta}_0 + X_1\hat{\beta}_1 + X_2\hat{\beta}_2 = 0.5$$

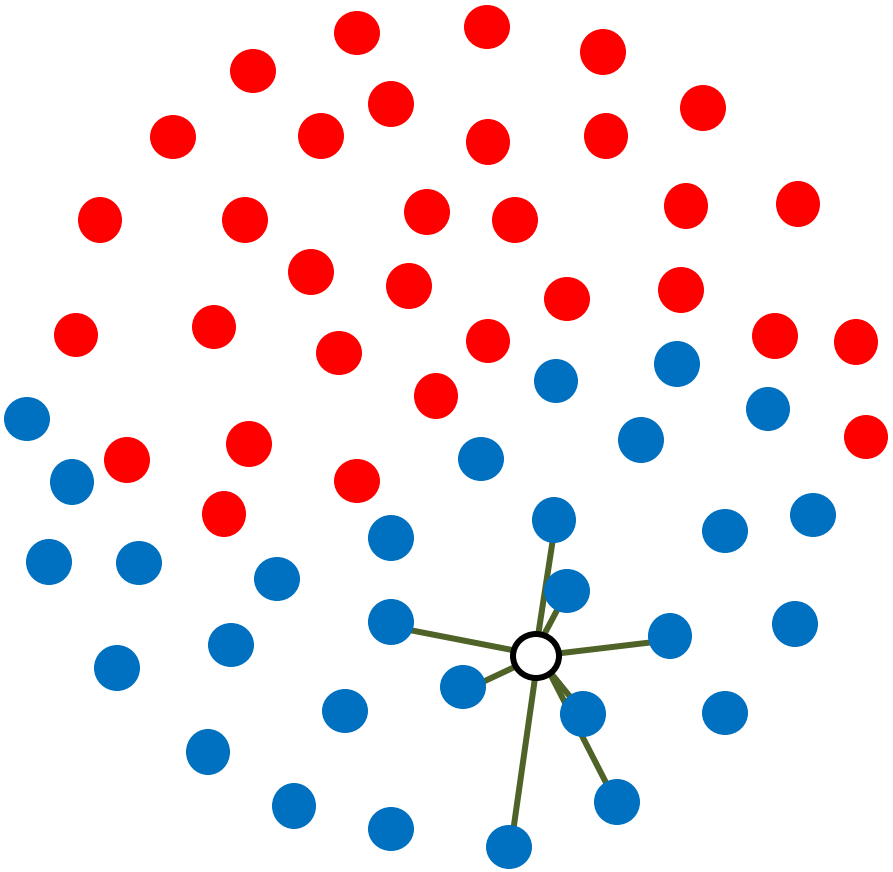
$$X_2 = -\frac{\hat{\beta}_1}{\hat{\beta}_2} X_1 + \frac{0.5 - \hat{\beta}_0}{\hat{\beta}_2}$$

선형 모델과 최소 제곱

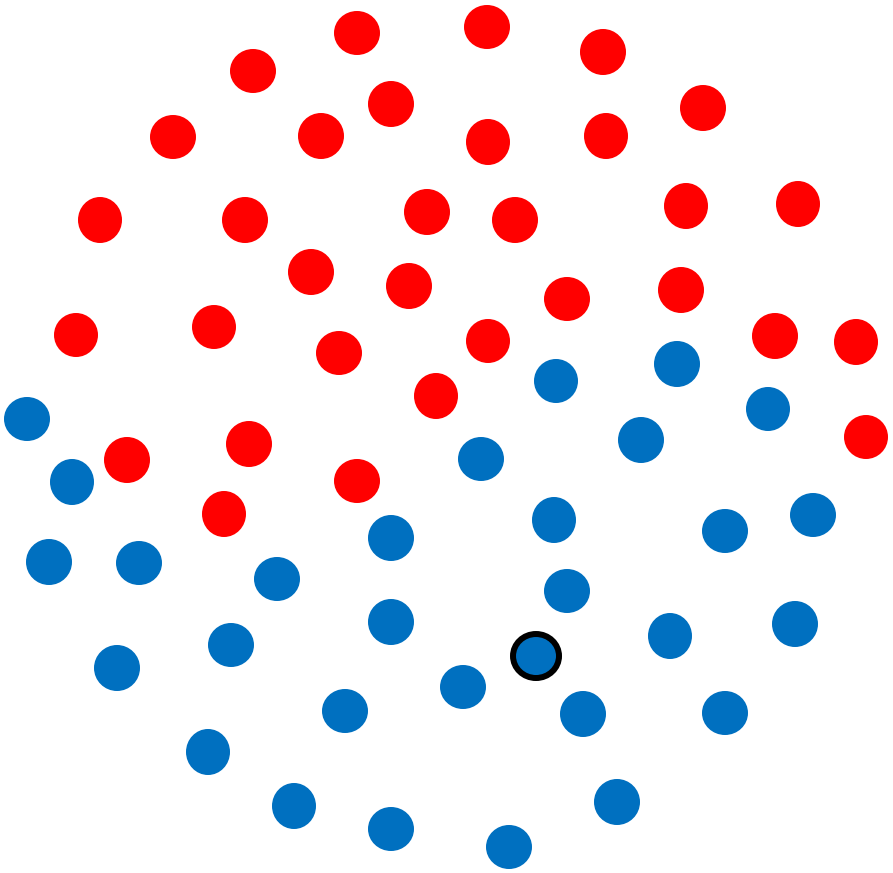
- 분류에서 선형 모형의 예
  - 두 개의 예측된 클래스가 결정 경계(Decision Boundary) $\{x: x^T\hat{\beta} = 0.5\}$에 의해 분리되어 있다. (선형)
  - 결정 경계 양쪽에 몇 개의 오분류가 있다.



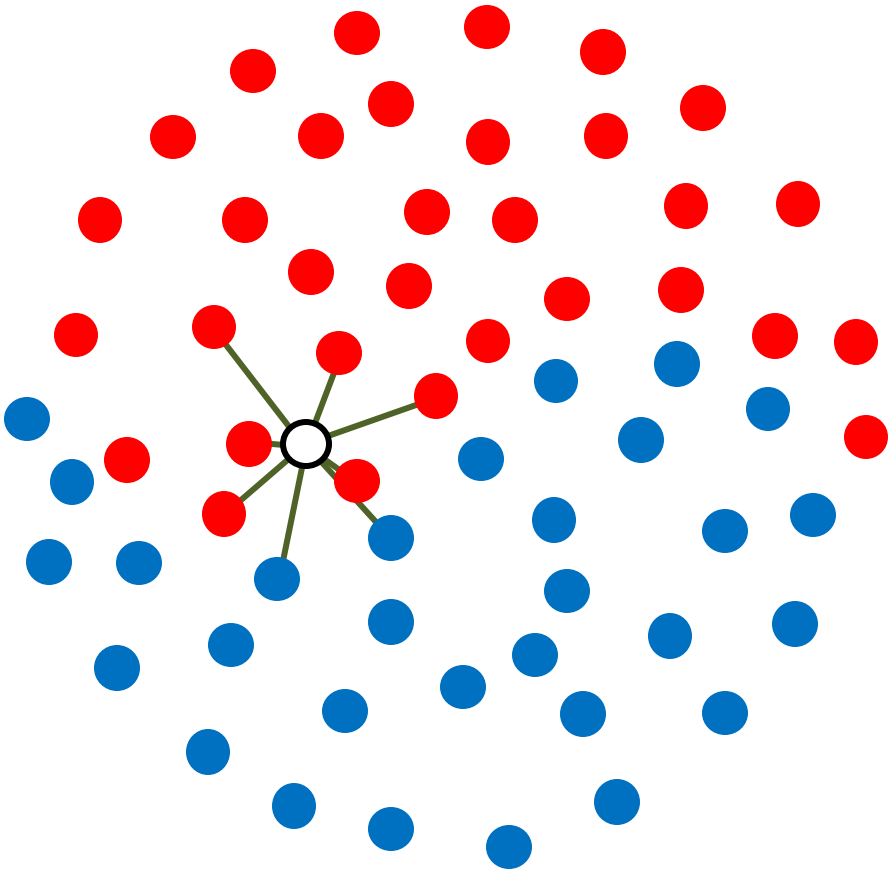
최근접이웃(k-nearest neighbor) 방법



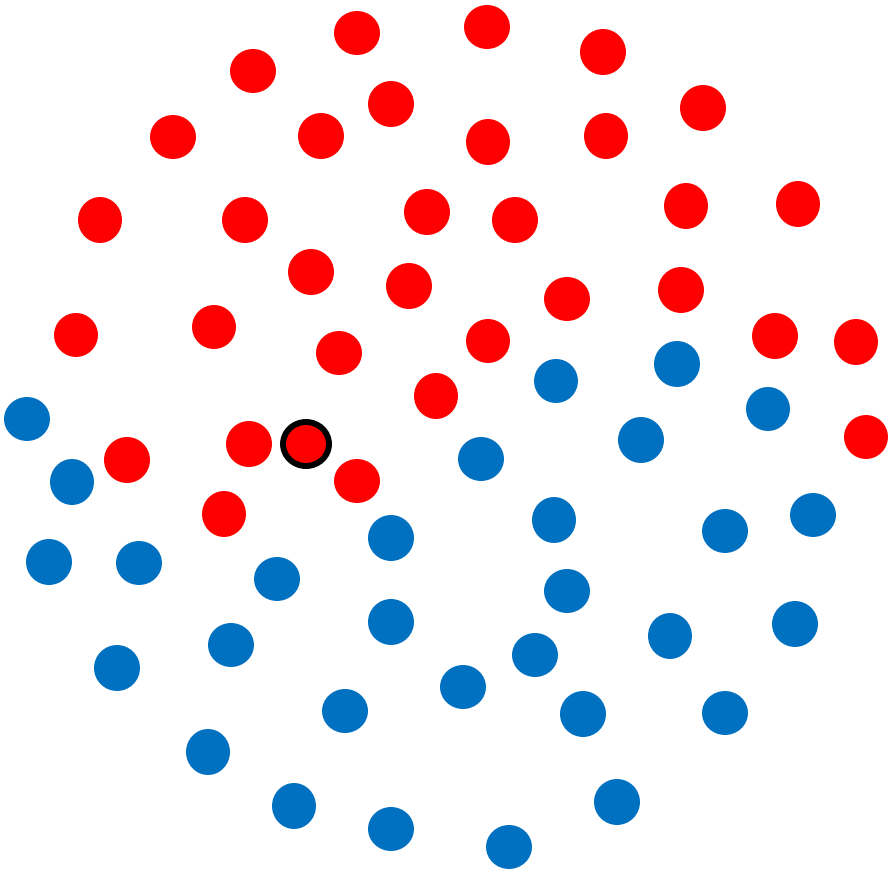
최근접이웃(k-nearest neighbor) 방법



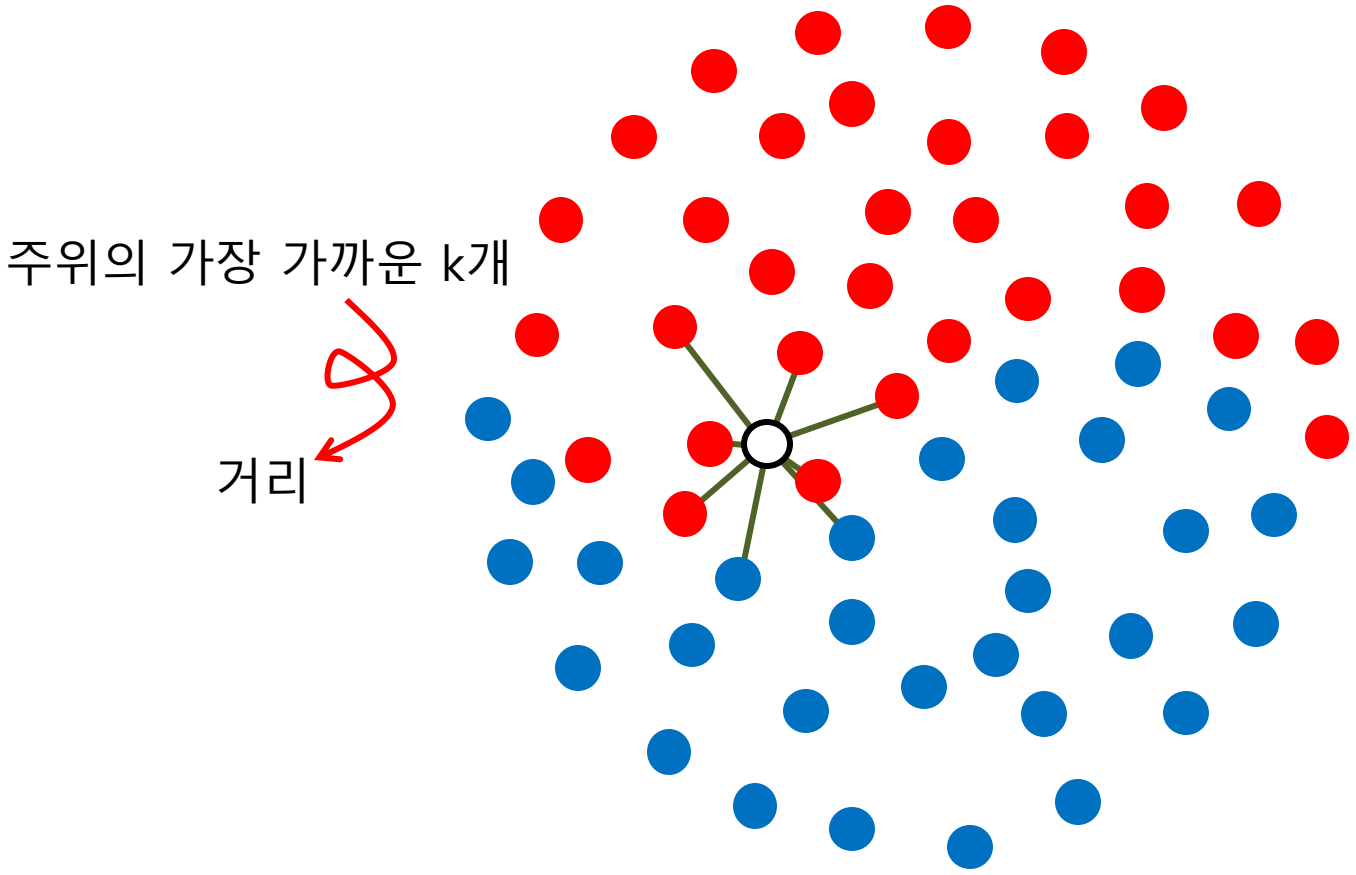
최근접이웃(k-nearest neighbor) 방법



최근접이웃(k-nearest neighbor) 방법



최근접이웃(k-nearest neighbor) 방법



최근접이웃(k-nearest neighbor) 방법

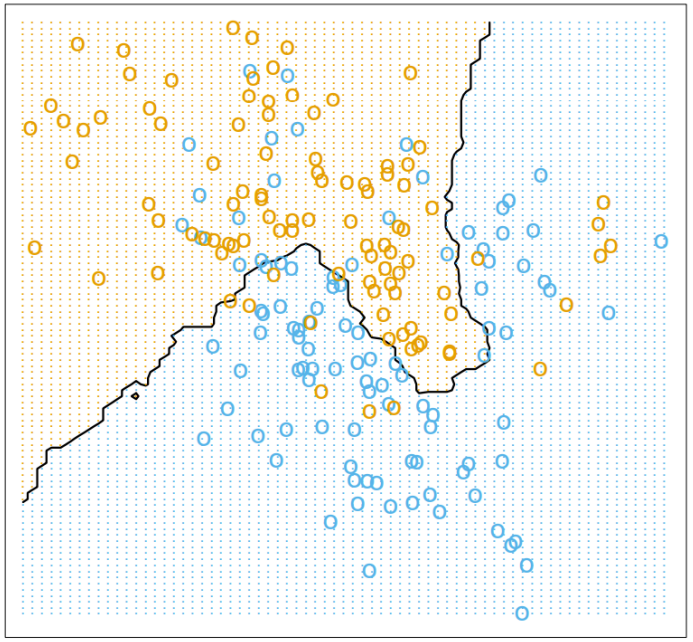
- $\hat{Y}$을 구성하기 위해
  훈련 집합의 x와 가장 가까운 k개의 x_i에 대한 관측치 y_i를 사용한다.

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

  - $N_k(x)$: x의 이웃.
    훈련 데이터 중에서 k개의 가장 가까운 x_i의 집합
  - 근접성(Closeness): 유클리드 거리로 가정
  - x와 가장 가까운 k개의 x_i로부터 k개의 관측치 y_i를 찾고 이들의 평균을 구한다.

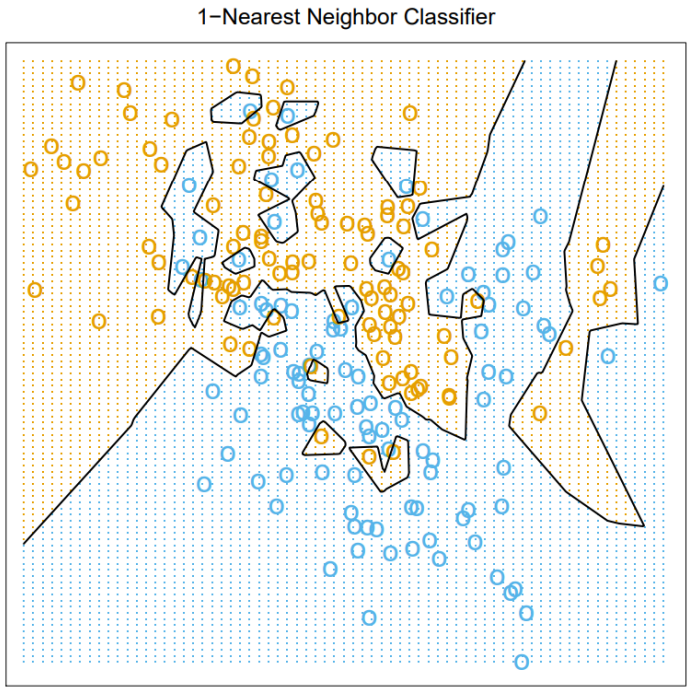
최근접이웃(k-nearest neighbor) 방법

- $k = 15$ BLUE가 0, ORANGE가 1로 코딩
- $\hat{Y}(x) = \frac{1}{k}\sum_{x_i \in N_k(x)} y_i$
- $\hat{Y}$: 이웃 내 ORANGE의 비율
- $\hat{Y} > 0.5$일 때
  클래스 ORANGE를 할당
  - 이웃 내에서 다수결
  - 입력 공간의 모든 점이
    이러한 규칙에 의해
    BLUE나 ORANGE로 분류
  - 결정 경계가 훨씬 불규칙적

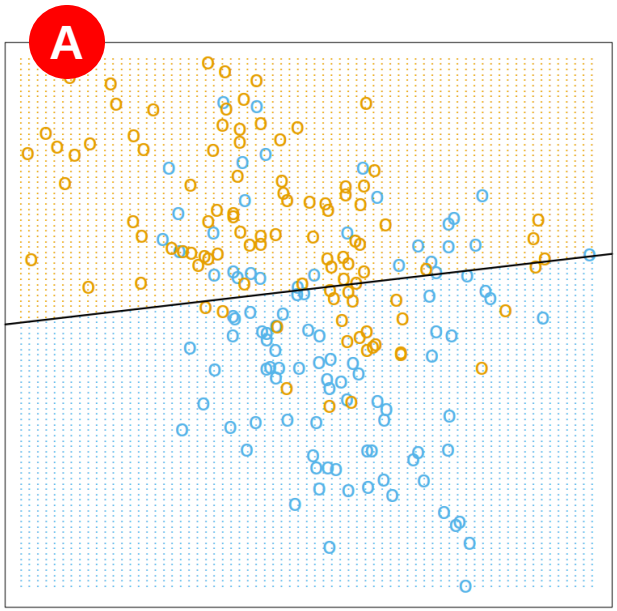


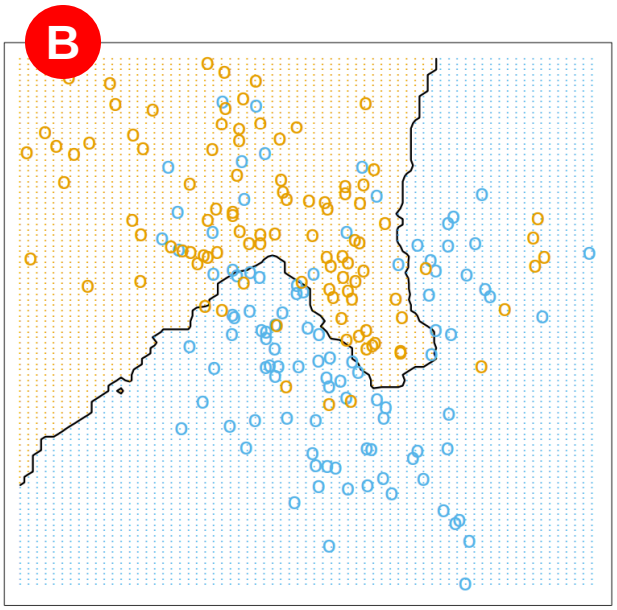
최근접이웃(k-nearest neighbor) 방법

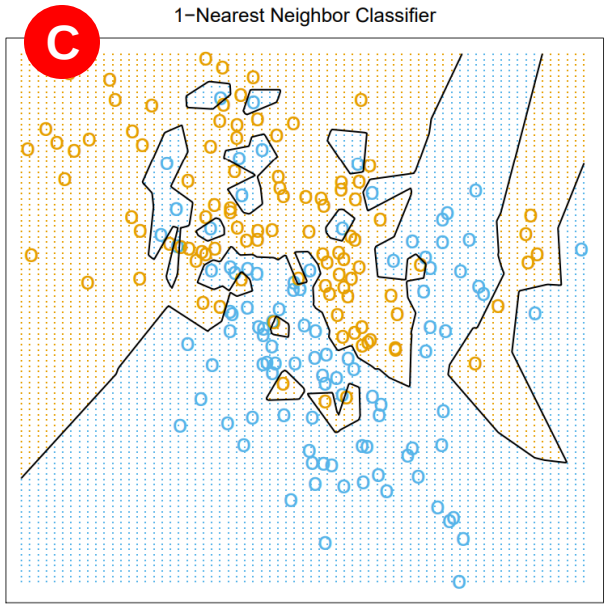
- 1-최근접이웃 분류의 결과
- $\hat{Y}$은 x에 가장 가까운 하나의 x_i에 대한 y_i 값으로 할당



최근접이웃(k-nearest neighbor) 방법

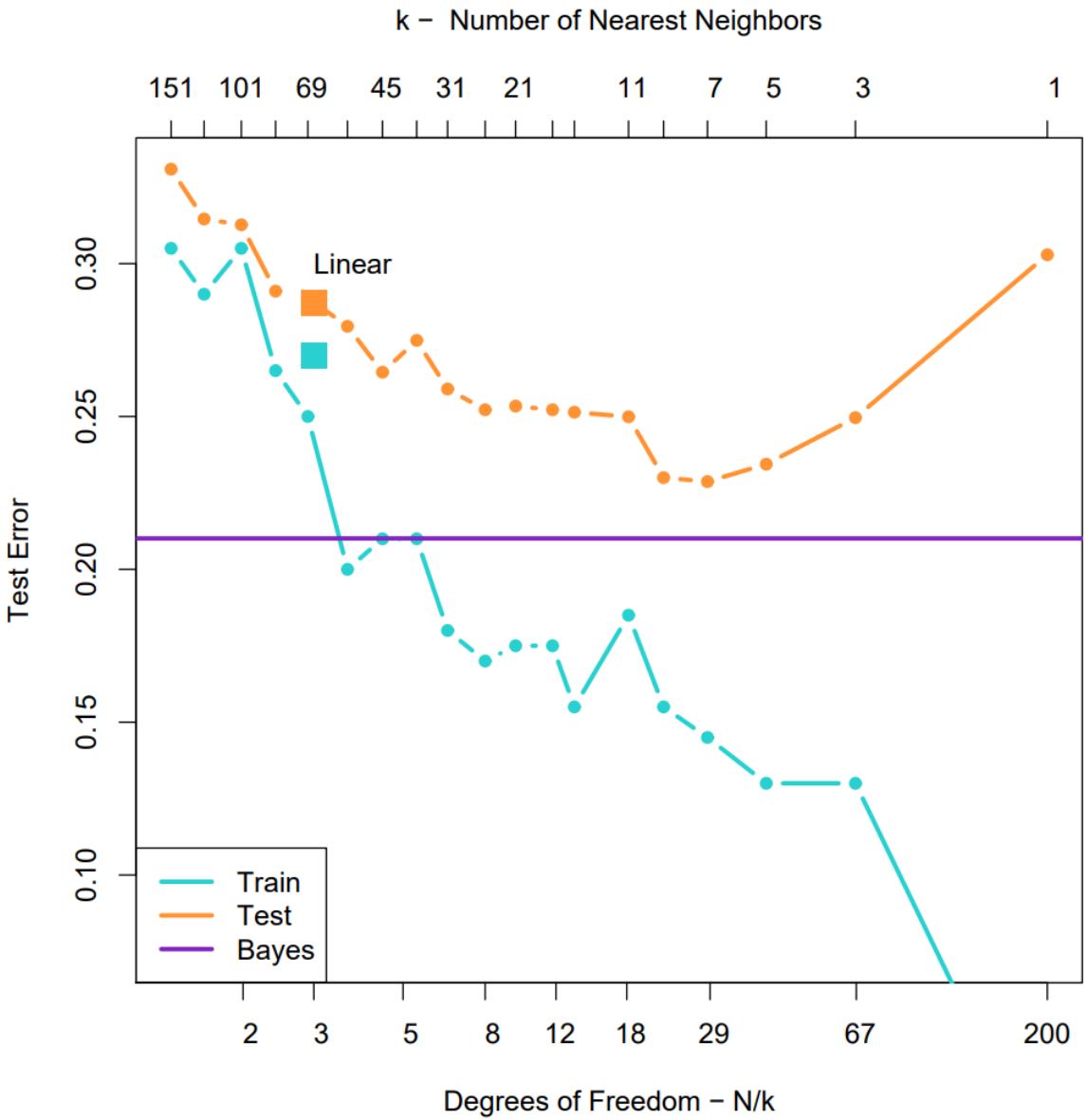






- 훈련 데이터에 대해
  - B가 A보다 적은 오분류
  - C는 오분류 없음
- 독립된 데이터 집합으로 비교하는 것이 필요

최소 제곱에서 최근접이웃까지



분류

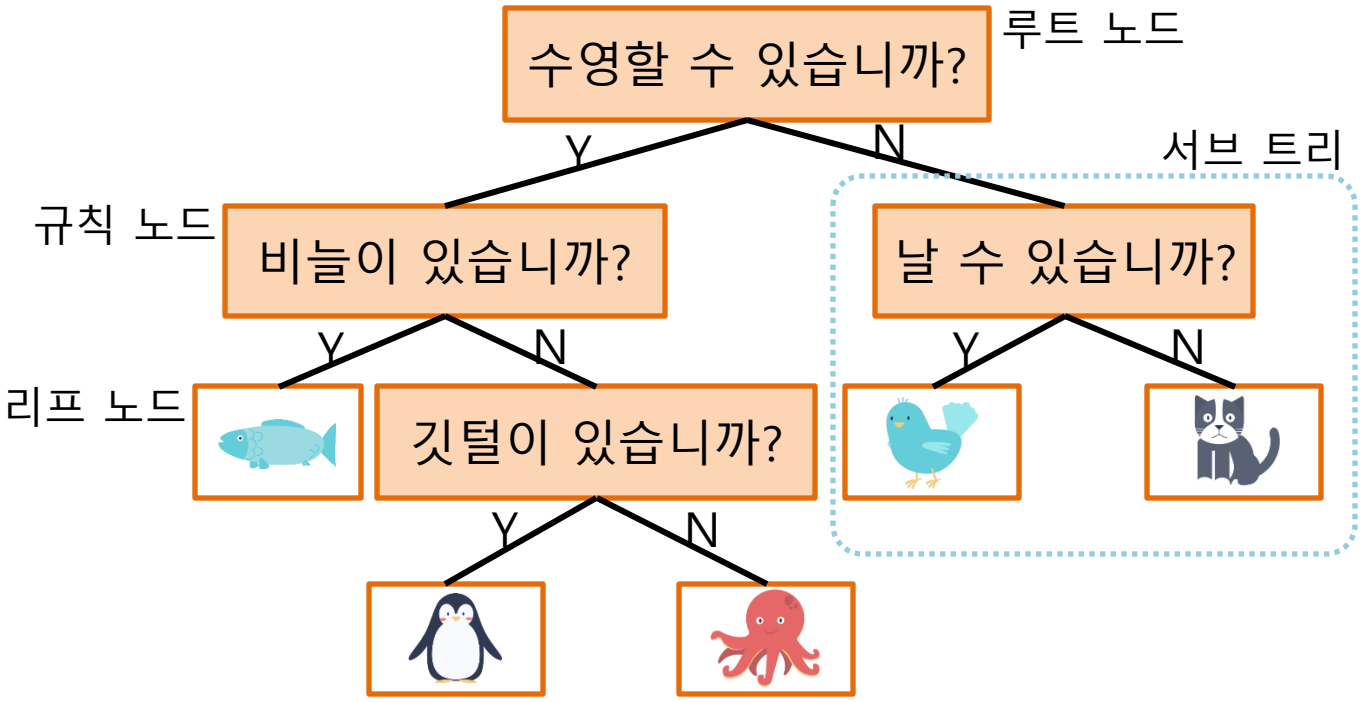
- 지도 학습의 대표적인 유형

- 다양한 알고리즘
  - 나이브 베이즈 (Naive Bayes)
  - 로지스틱 회귀 (Logistic Regression)
  - 결정 트리 (Decision Tree)
  - 서포트 벡터 머신 (Support Vector Machine)
  - 최소 근접 (Nearest Neighbor) 알고리즘
  - 인공 신경망 (Neural Network)
  - 앙상블 (Ensemble)

결정 트리 (Decision Tree)

- 데이터의 규칙을 학습을 통해 자동으로 찾아서 트리 기반의 분류 규칙을 만드는 것
- 스무고개 게임과 유사하며 if/else를 자동으로 찾아서 규칙을 만드는 알고리즘이라 생각할 수도 있음
- 어떤 기준으로 규칙을 만드는지에 따라 성능이 달라짐

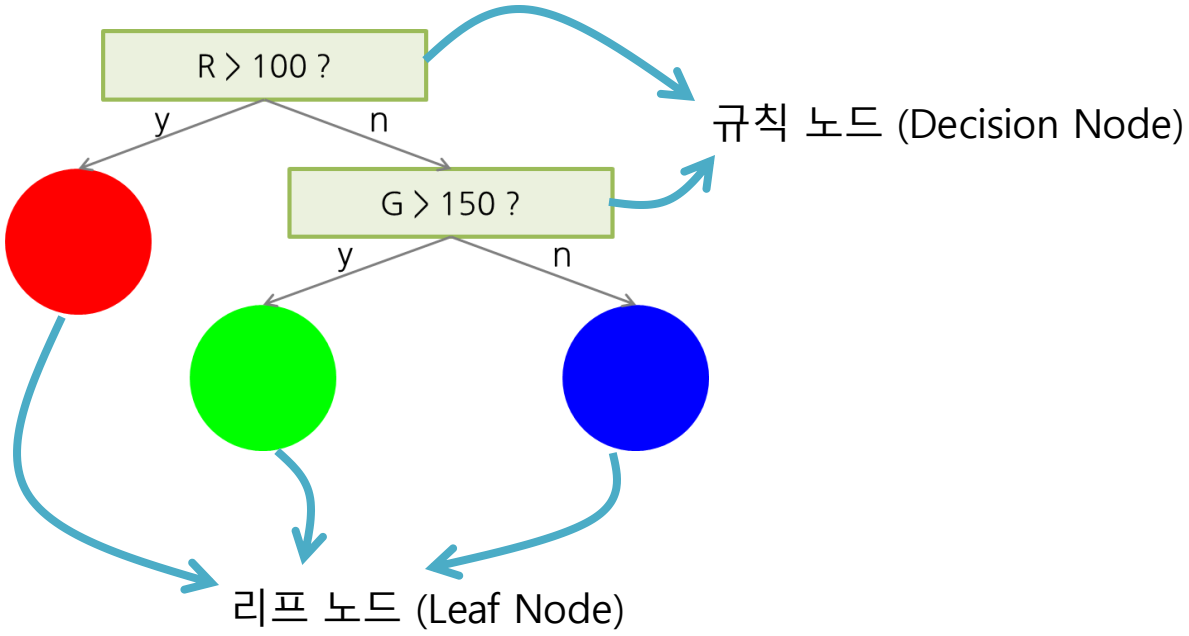
결정 트리의 구조



- 규칙 노드 (Decision Node): 규칙 조건
- 리프 노드 (Leaf Node): 클래스 라벨

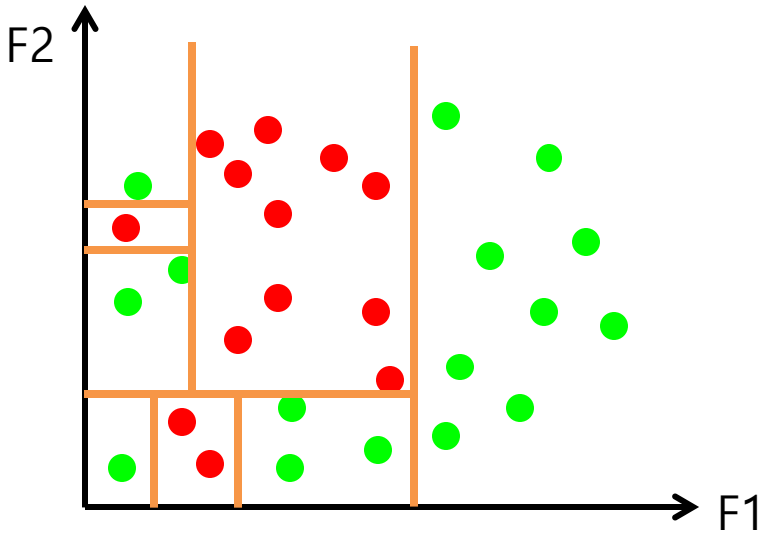
결정 트리 (Decision Tree)

- 연속적인 값은 어떻게 할 것인가?
- 어떤 특징(Feature)부터 검사할 것인가?
- 둘로 나누는 기준 값은?



결정 트리 (Decision Tree)

- 연속적인 값은 어떻게 할 것인가?
- 어떤 특징(Feature)부터 검사할 것인가?
- 둘로 나누는 기준 값은?



- 많은 노드로 분류

  → 성능이 떨어질 수 있음 (Overfitting)
- 적은 노드로 분류

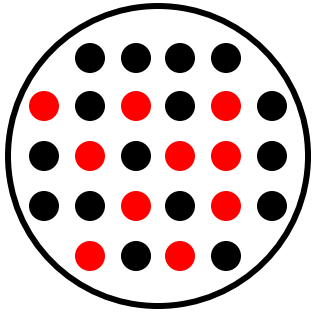
  → 최대한 많은 데이터가 분할될 수 있도록 노드의 규칙을 정해야 함

  → 최대한 균일하게(균일도가 높게) 되도록 분할

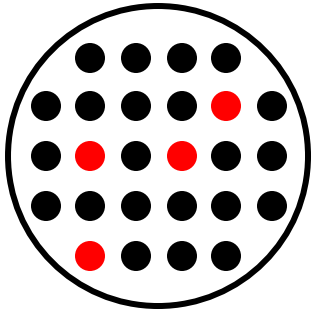
결정 트리 (Decision Tree)

- 균일도가 높다라는 것은?

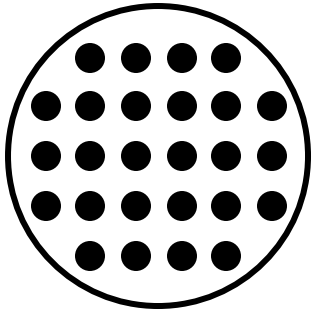
데이터 세트 A



데이터 세트 B



데이터 세트 C

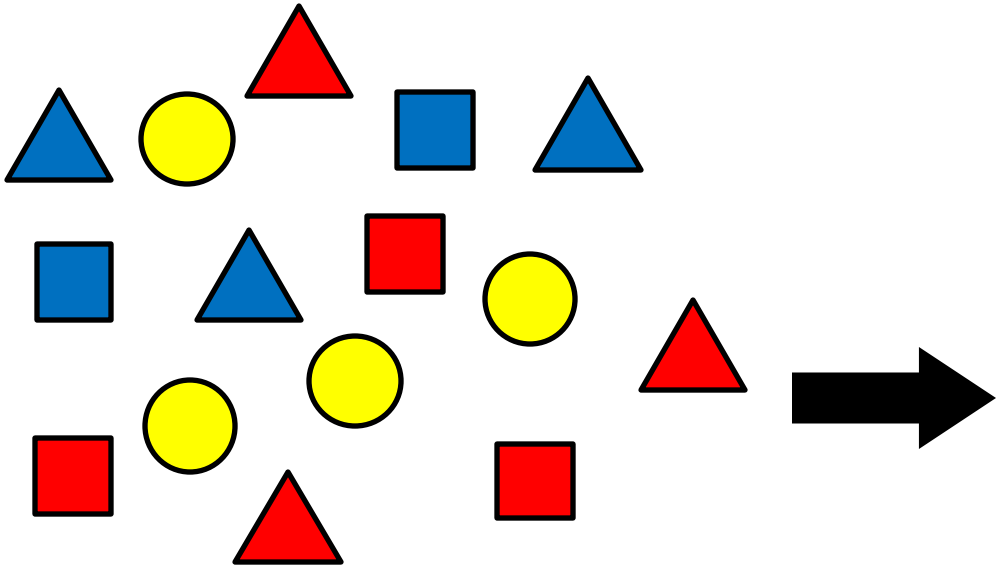


균일도 : A < B < C

판단에 필요한 정보의 양 : A > B > C

결정 노드의 규칙 조건

- 정보 균일도가 높은 데이터 세트를
  먼저 선택할 수 있도록 규칙 조건을 만듦



첫 번째 조건은
if 색깔 == 노랑

- 균일도를 판단하는 방법은?
  - 엔트로피 (Entropy)
  - 지니 불순도 (Gini Impurity)

결정 트리 (Decision Tree)

- 엔트로피 (Entropy)

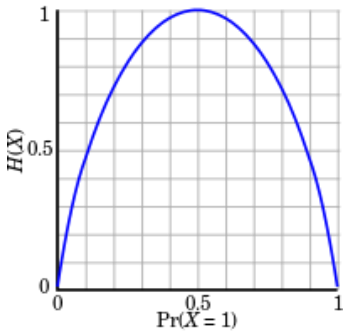
정보량 (Information Content)

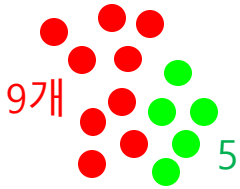
$$\log_2 \frac{1}{p(x_i)} = -\log_2 p(x_i)$$

엔트로피 (Entropy) : 정보량의 기대 값

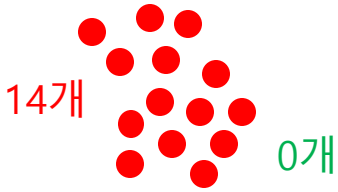
$$\mathrm{H}(X) = E[-\log_2 p(X)]$$

$$= -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$





$$\mathrm{H}(X) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = \mathbf{0.94}$$



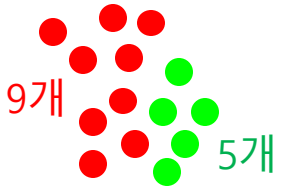
$$\mathrm{H}(X) = -\frac{14}{14}\log_2\frac{14}{14} - \frac{0}{14}log_2\frac{0}{14} = \mathbf{0}$$

결정 트리 (Decision Tree)

- 지니 불순도 (Gini Impurity)

$$1 - \sum_{i=1}^{c} p_i^2$$

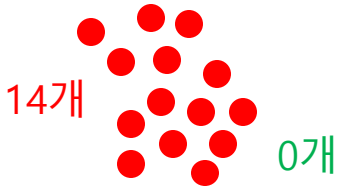
$$p_i = \frac{N_i}{\sum_{i=1}^{c} N_i}$$



$$p_1 = \frac{9}{14}$$

$$p_2 = \frac{5}{14}$$

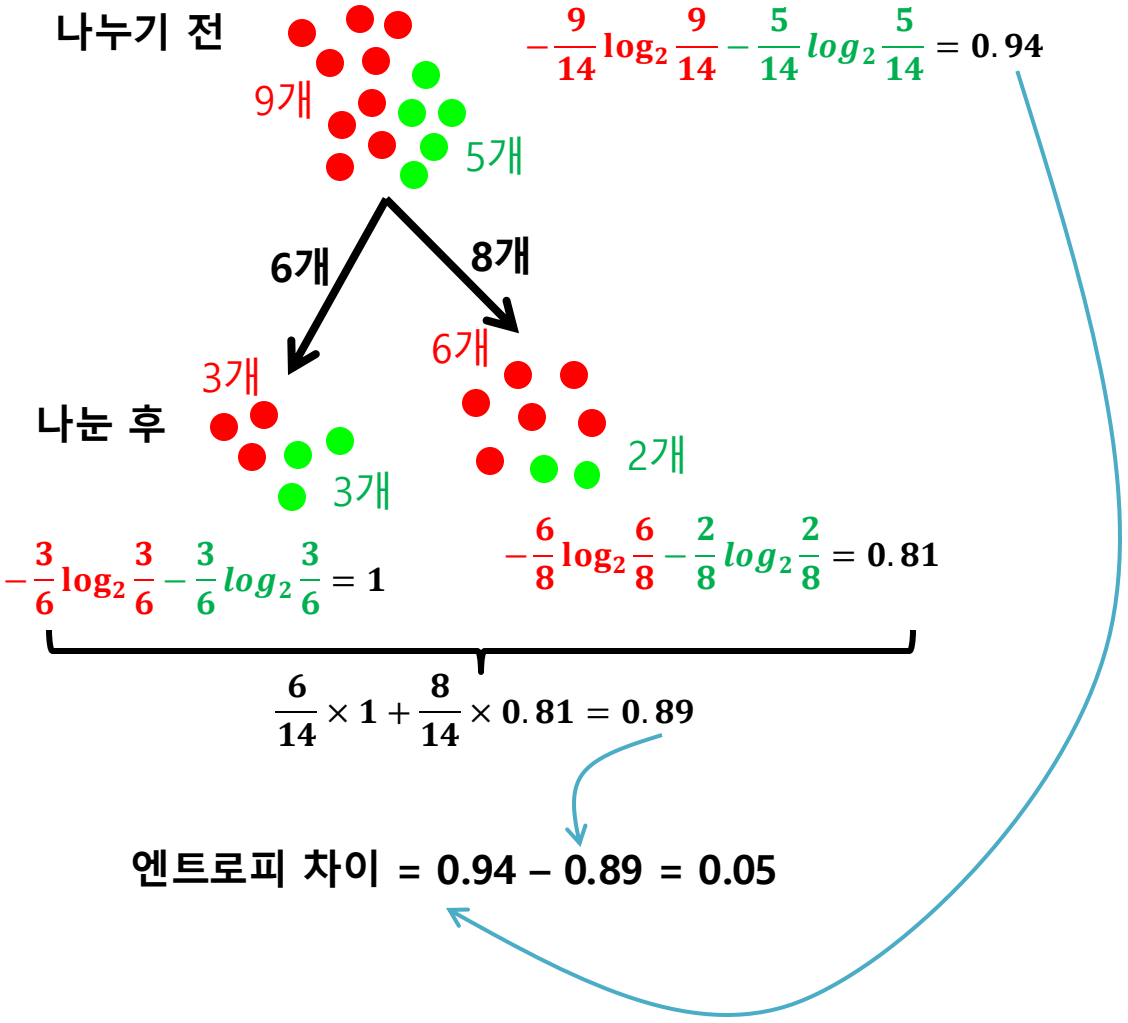
$$1 - \sum_{i=1}^{c} p_i^2 = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$



$$1 - \sum_{i=1}^{c} p_i^2 = 1 - \left(\frac{14}{14}\right)^2 - \left(\frac{0}{14}\right)^2 = 0$$

결정 트리 (Decision Tree)

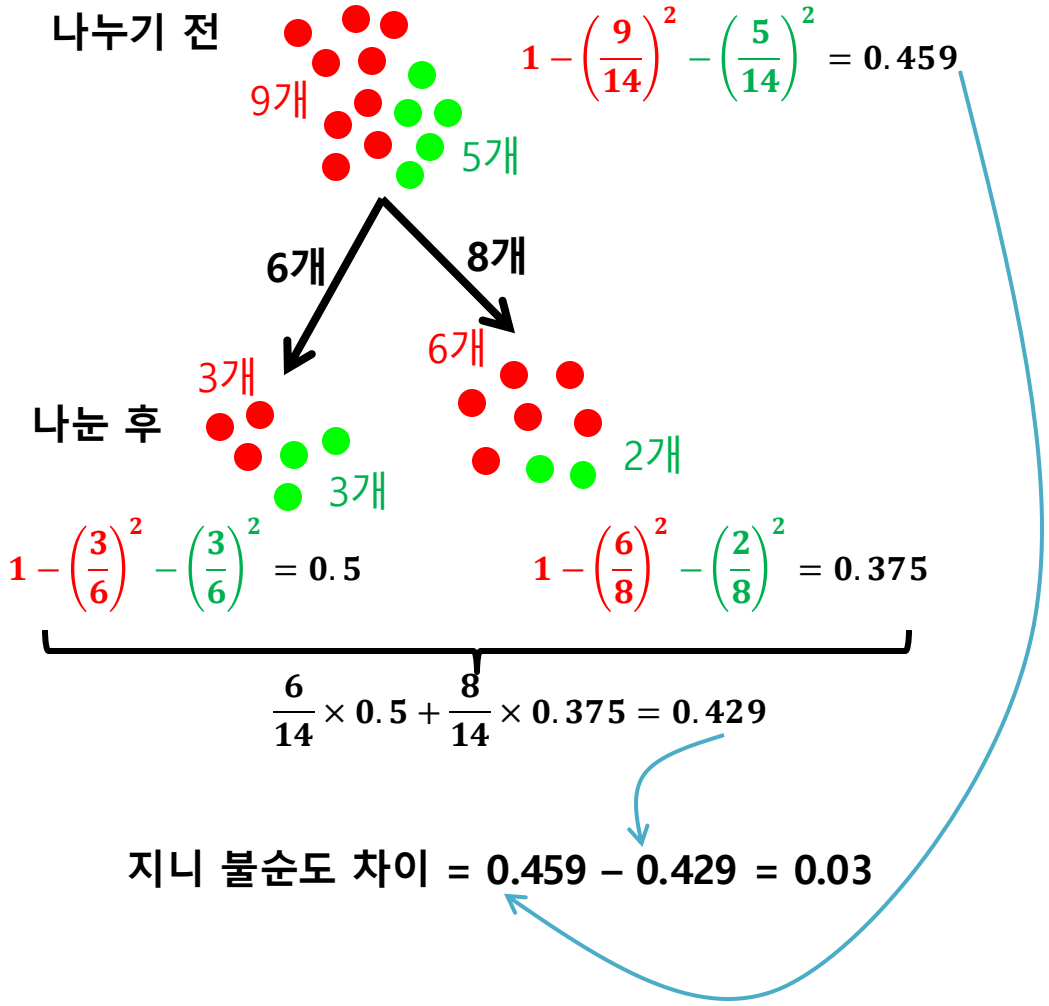
- 엔트로피 (Entropy)



결정 트리 (Decision Tree)

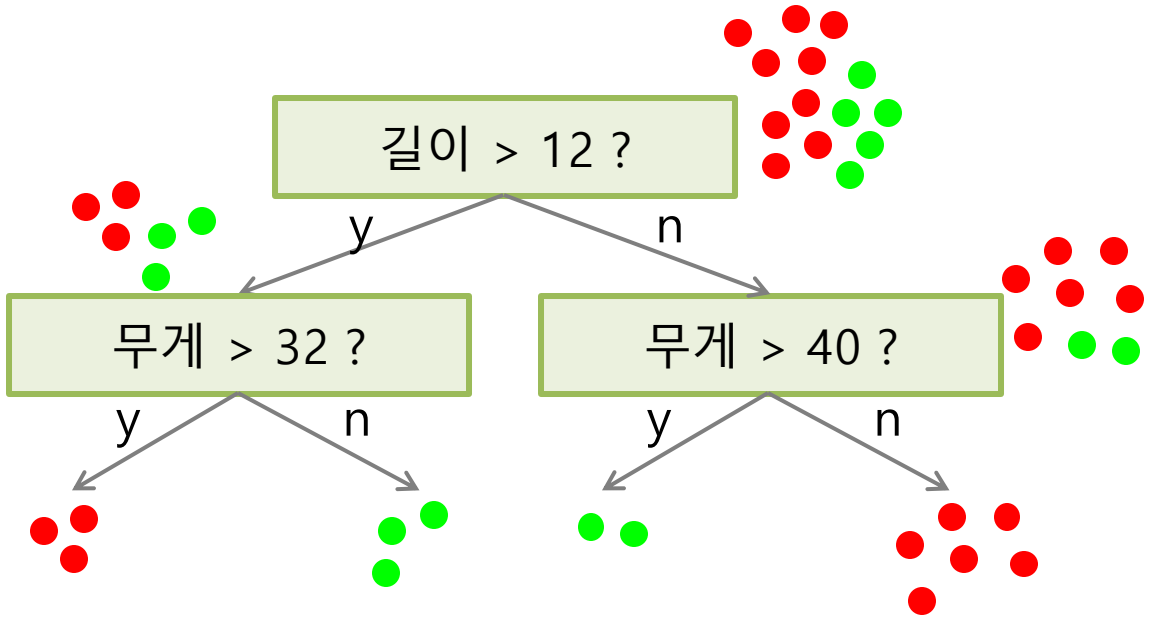
- 지니 불순도 (Gini Impurity)

$$1 - \sum_{i=1}^{c} p_i^2$$



결정 트리 (Decision Tree)

길이	무게
10	30
11	31
12	32
13	33
14	34
…	…
20	50



사이킷런의 DecisionTreeClassifier는 지니 불순도 사용

결정 트리의 특징

- 장점
  - 알고리즘이 쉽고 직관적
  - 룰이 명확하고 시각화가 쉬움
  - 각 피처의 스케일링, 정규화 등의 전처리 작업 필요 없음
- 단점
  - 과적합으로 정확도가 떨어짐
  - 정확도를 높이기 위해 계속 조건을 추가하면
    트리가 깊고 복잡해짐
    → 테스트 데이터에 대해 유연하게 대처하지 못함

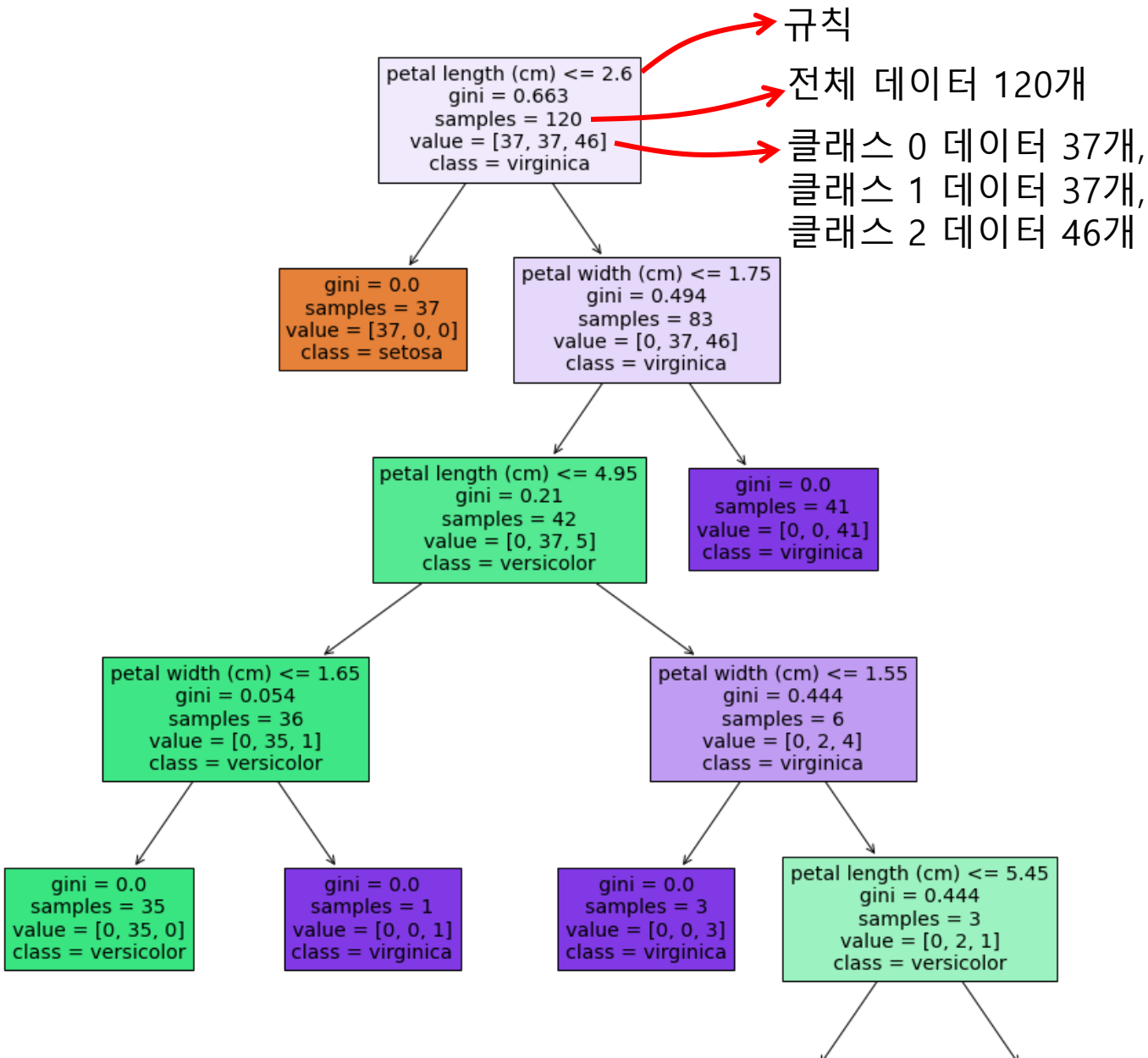
DecisionTreeClassifier 파라미터

- min_samples_split
  - 노드를 분할하기 위한 최소한의 샘플 데이터 수 (기본 값: 2)
- min_samples_leaf
  - 말단 노드가 되기 위한 최소한의 샘플 데이터 수
- max_features
  - 최대 피처 수 (기본 값: None, 모든 피처 사용)
  - int 값: 피처의 개수
  - float 값: 퍼센트
  - 'sqrt': $\sqrt{\text{전체 피처 개수}}$
  - 'auto': 'sqrt'와 같음
  - 'log': $\log_2(\text{전체 피처 개수})$
  - 'None': 전체 피처
- max_depth
  - 트리의 최대 깊이
  - 기본 값 None: 클래스 결정될 때까지 또는 노드의 데이터 개수가 min_samples_split 보다 작을 때까지 깊이 증가
- max_leaf_nodes
  - 말단 노드의 최대 개수

결정 트리


```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2)
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)

plt.figure(figsize=(16,16))
tree.plot_tree(dtc, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
```

결정 트리



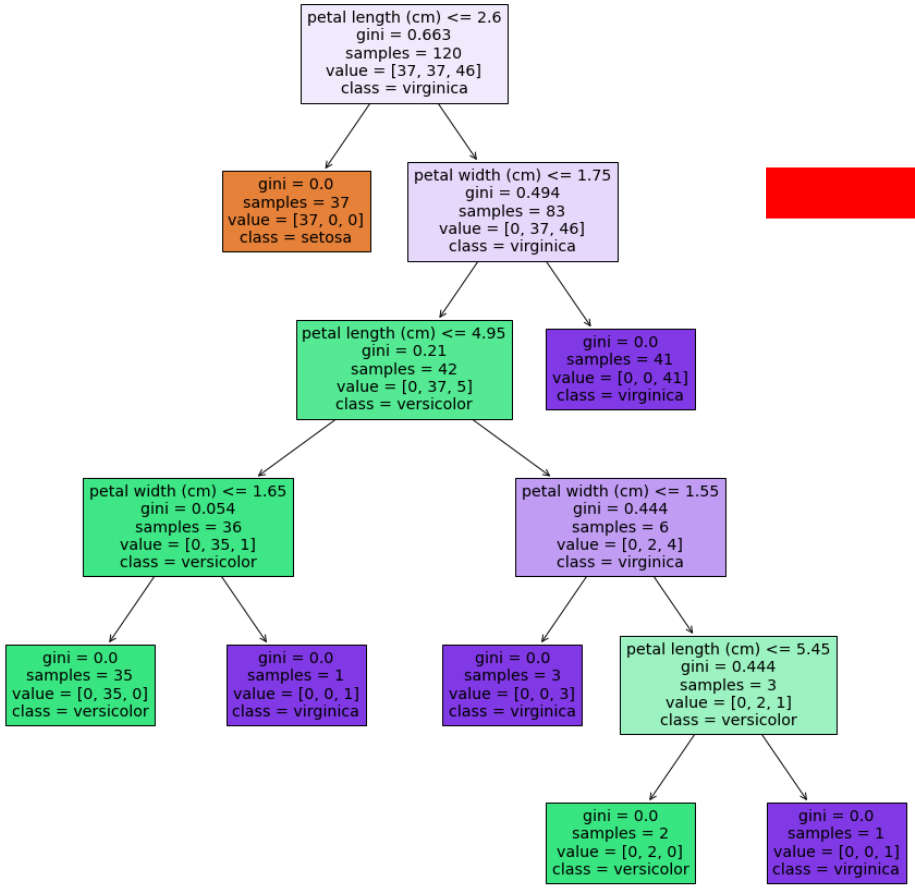
규칙

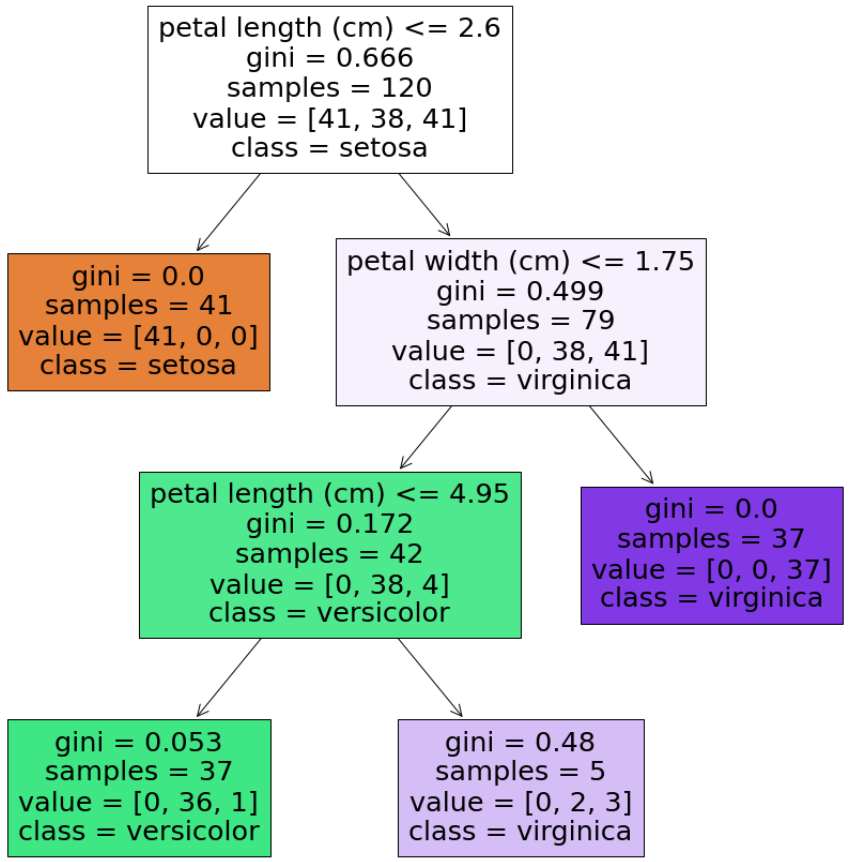
전체 데이터 120개

클래스 0 데이터 37개,
클래스 1 데이터 37개,
클래스 2 데이터 46개

결정 트리

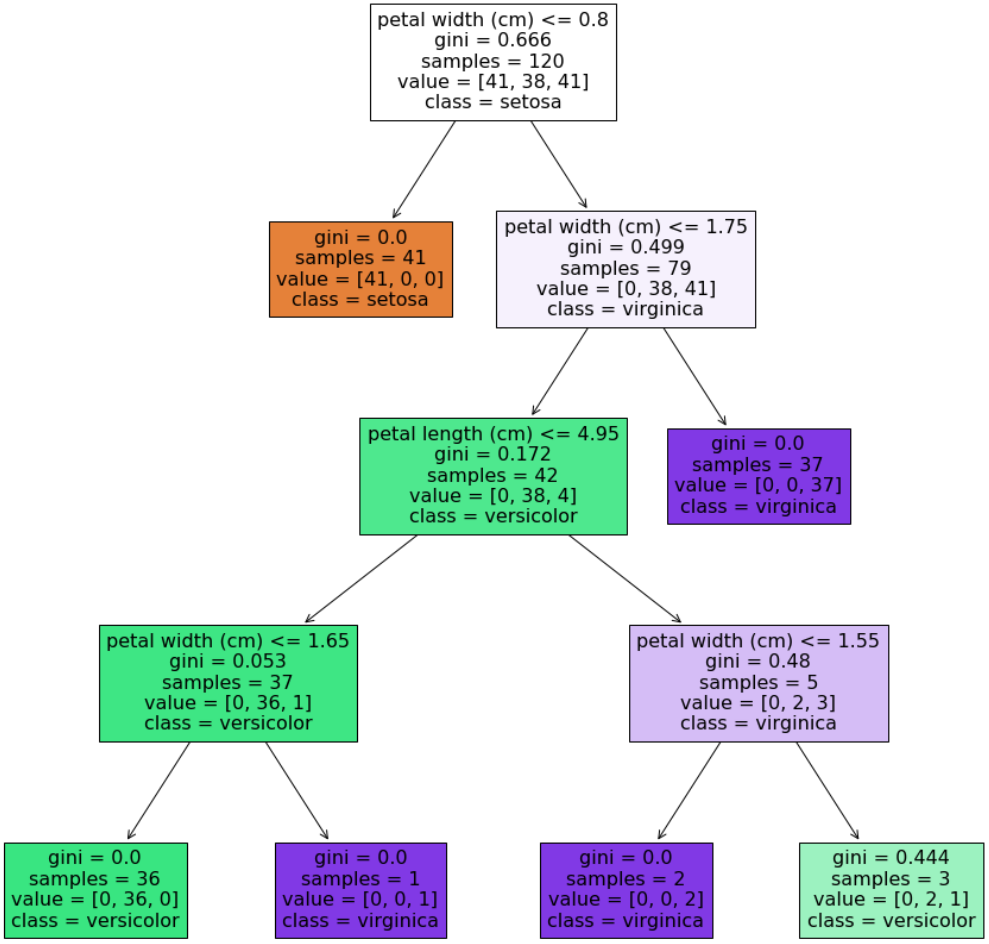

```
dtc = DecisionTreeClassifier(max_depth=3)
dtc.fit(X_train, y_train)

plt.figure(figsize=(16,16))
tree.plot_tree(dtc, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
```





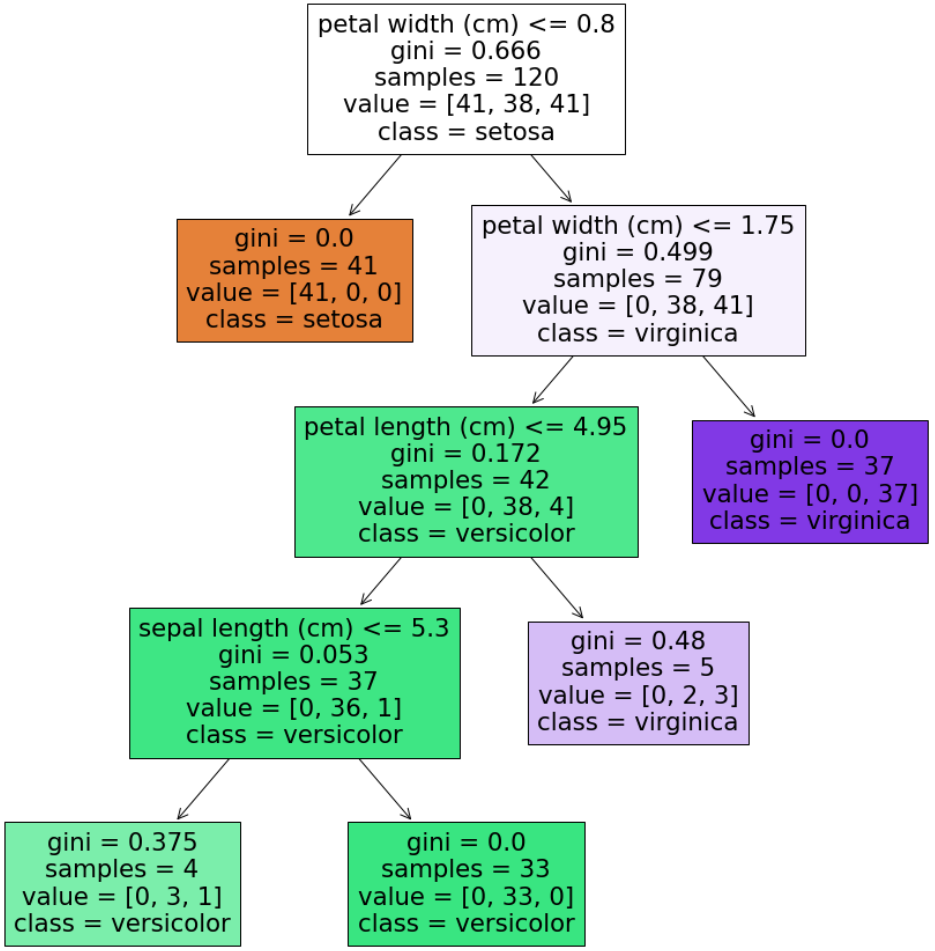
결정 트리


```
dtc = DecisionTreeClassifier(min_samples_split=4)
dtc.fit(X_train, y_train)

plt.figure(figsize=(16,16))
tree.plot_tree(dtc, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
```



결정 트리

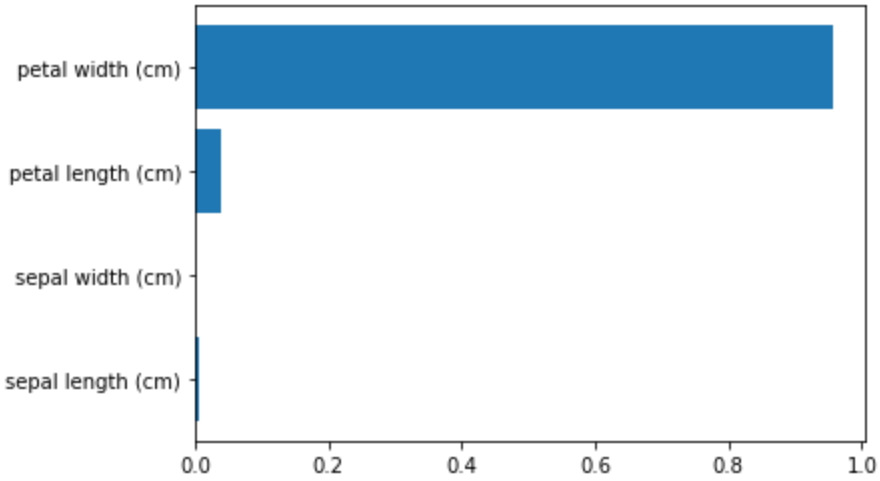

```
dtc = DecisionTreeClassifier(min_samples_leaf=4)
dtc.fit(X_train, y_train)

plt.figure(figsize=(16,16))
tree.plot_tree(dtc, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
```



피처의 중요도

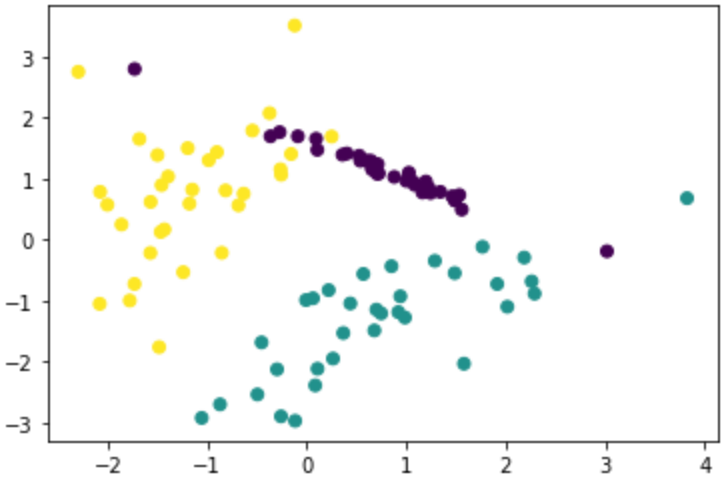
- DecisionTreeClassifier
  - feature_importances_ 속성

```
print(dtc.feature_importances_)
plt.barh(iris.feature_names, dtc.feature_importances_)
```



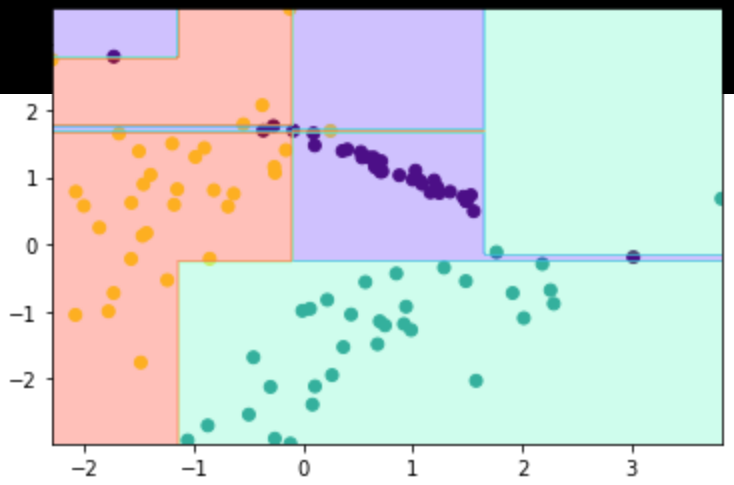
결정 트리 과적합 (Overfitting)


```
from sklearn.datasets import make_classification

X, y = make_classification(n_features=2, n_redundant=0, n_classes=3, n_clusters_per_class=1)
plt.scatter(X[:,0], X[:,1], c=y)
```



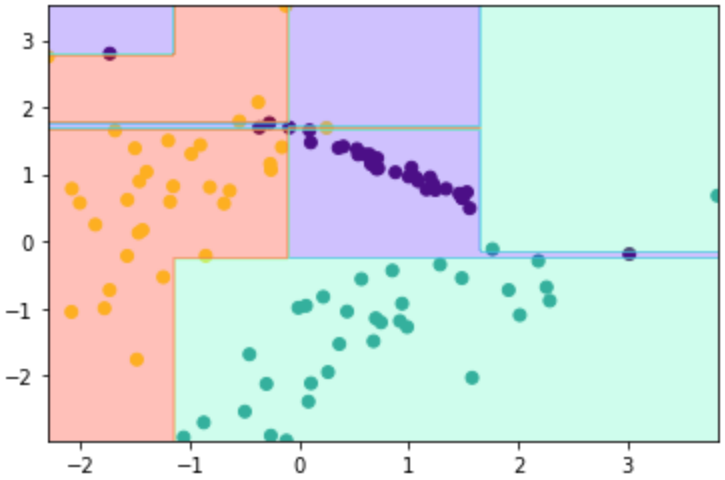
결정 트리 과적합 (Overfitting)

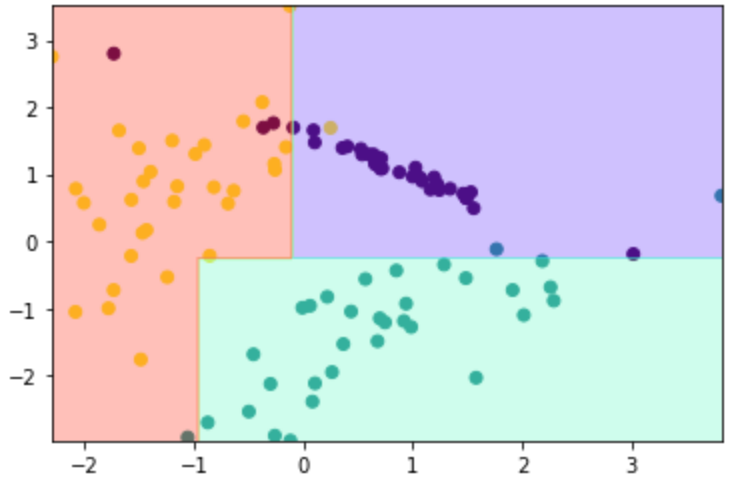

```
import numpy as np

def visualize(model, X, y):
    X1 = X[:,0]
    X2 = X[:,1]
    plt.scatter(X1, X2, c=y)
    x1_min = np.min(X1)
    x1_max = np.max(X1)
    x2_min = np.min(X2)
    x2_max = np.max(X2)
    xx, yy = np.meshgrid(np.linspace(x1_min, x1_max, num=200), np.linspace(x2_min, x2_max, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    n_classes = len(np.unique(y))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='rainbow')

dtc = DecisionTreeClassifier()
dtc.fit(X, y)
visualize(dtc, X, y)
```



결정 트리 과적합 (Overfitting)

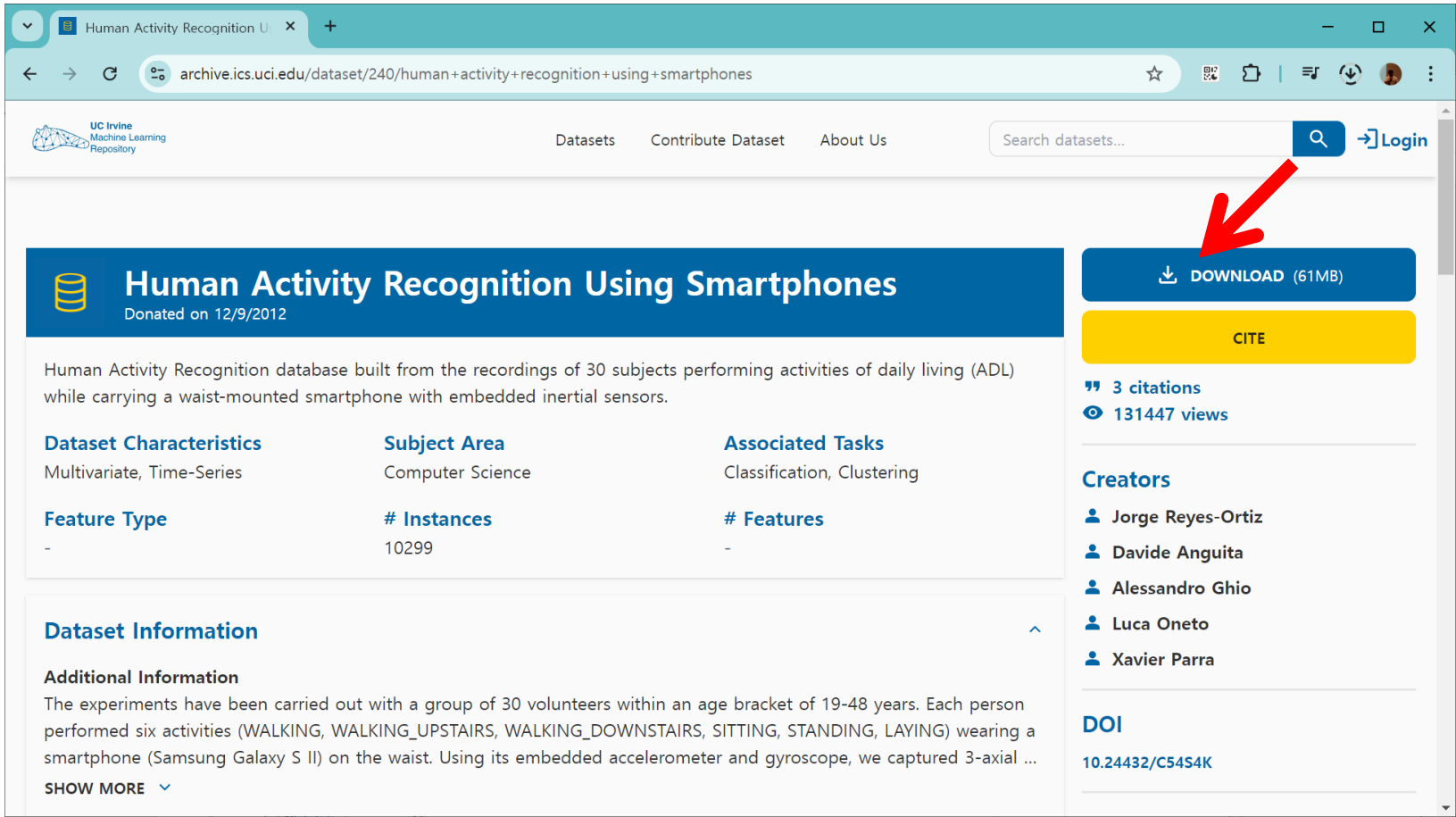

```
dtc = DecisionTreeClassifier(min_samples_leaf=6)
dtc.fit(X, y)
visualize(dtc, X, y)
```





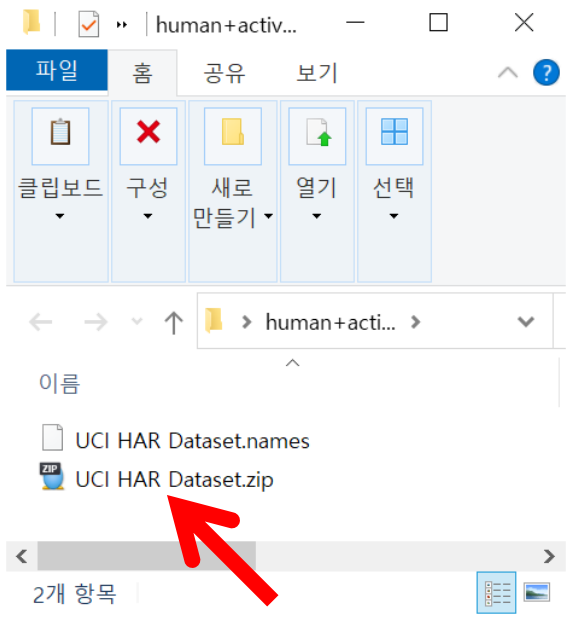
사용자 행동 인식

- UCI 사용자 행동 데이터 세트
https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones



사용자 행동 인식

- 내려 받은 zip 파일의 압축을 푼다.
- UCI HAR Dataset.zip의 압축을 푼다.
- 압축 푼 폴더 이름을 uci로 변경



사용자 행동 인식


```
import pandas as pd

df = pd.read_csv('c:/Example/uci/features.txt',
                 sep=r'\s+',
                 header=None,
                 names=['column_index', 'column_name'])
df
```

features.txt

```
1 tBodyAcc-mean()-X
2 tBodyAcc-mean()-Y
3 tBodyAcc-mean()-Z
4 tBodyAcc-std()-X
5 tBodyAcc-std()-Y
6 tBodyAcc-std()-Z
7 tBodyAcc-mad()-X
8 tBodyAcc-mad()-Y
9 tBodyAcc-mad()-Z
10 tBodyAcc-max()-X
11 tBodyAcc-max()-Y
12 tBodyAcc-max()-Z
13 tBodyAcc-min()-X
14 tBodyAcc-min()-Y
15 tBodyAcc-min()-Z
16 tBodyAcc-sma()
17 tBodyAcc-energy()-X
18 tBodyAcc-energy()-Y
19 tBodyAcc-energy()-Z
20 tBodyAcc-iqr()-X
```

	column_index	column_name
0	1	tBodyAcc-mean()-X
1	2	tBodyAcc-mean()-Y
2	3	tBodyAcc-mean()-Z
3	4	tBodyAcc-std()-X
4	5	tBodyAcc-std()-Y
...	...	...
556	557	angle(tBodyGyroMean,gravityMean)
557	558	angle(tBodyGyroJerkMean,gravityMean)
558	559	angle(X,gravityMean)
559	560	angle(Y,gravityMean)
560	561	angle(Z,gravityMean)

561 rows × 2 columns

사용자 행동 인식

- 중복된 피처 확인

```
dup = df.groupby('column_name').count()
print(dup)
dup2 = dup[dup['column_index'] > 1]
print(dup2.count())
dup2.head()
```

```
                                        column_index
column_name
angle(X,gravityMean)                               1
angle(Y,gravityMean)                               1
angle(Z,gravityMean)                               1
angle(tBodyAccJerkMean),gravityMean)               1
angle(tBodyAccMean,gravity)                        1
...                                              ...
tGravityAccMag-max()                               1
tGravityAccMag-mean()                              1
tGravityAccMag-min()                               1
tGravityAccMag-sma()                               1
tGravityAccMag-std()                               1

[477 rows x 1 columns]
column_index    42
dtype: int64
```

column_name	column_index
fBodyAcc-bandsEnergy()-1,16	3
fBodyAcc-bandsEnergy()-1,24	3
fBodyAcc-bandsEnergy()-1,8	3
fBodyAcc-bandsEnergy()-17,24	3
fBodyAcc-bandsEnergy()-17,32	3

사용자 행동 인식

- 중복된 피처 처리

```
df2 = pd.DataFrame(data=df.groupby('column_name').cumcount(), columns=['cnt'])
df2 = df2.reset_index()
df3 = pd.merge(df.reset_index(), df2, how='outer')
df3['column_name'] = df3[['column_name', 'cnt']].apply(lambda x:
                                x.iloc[0]+'_'+str(x.iloc[1]) if x.iloc[1] > 0 else x.iloc[0],
                                axis=1)
df3 = df3.drop(['index'], axis=1)
feature_name = df3.iloc[:,1].values.tolist()
```

사용자 행동 인식

- 데이터 읽기

```
X_train = pd.read_csv('c:/Example/uci/train/X_train.txt', sep=r'\s+', names=feature_name)
X_test = pd.read_csv('c:/Example/uci/test/X_test.txt', sep=r'\s+', names=feature_name)
y_train = pd.read_csv('c:/Example/uci/train/y_train.txt', sep=r'\s+', header=None, names=['action'])
y_test = pd.read_csv('c:/Example/uci/test/y_test.txt', sep=r'\s+', header=None, names=['action'])
```

사용자 행동 인식


```
print(X_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 561 entries, tBodyAcc-mean()-X to angle(Z,gravityMean)
dtypes: float64(561)
memory usage: 31.5 MB
None
```

```
print(y_train['action'].value_counts())
```

```
action
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: count, dtype: int64
```

사용자 행동 인식

- 하이퍼 파라미터 값들

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_hat = dtc.predict(X_test)
accuracy = accuracy_score(y_test, y_hat)

print(accuracy)
print(dtc.get_params())
```

```
0.8561248727519511
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': None, 'splitter': 'best'}
```

사용자 행동 인식

- 최적의 하이퍼 파라미터 값 찾기

```
from sklearn.model_selection import GridSearchCV

params = {'max_depth': [6, 8, 10, 12, 16, 20, 24]}
gs = GridSearchCV(dtc, param_grid=params, scoring='accuracy', cv=5)
gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

실행 시간 오래 걸림

```
0.8551503211753768
{'max_depth': 8}
```

```
results = pd.DataFrame(gs.cv_results_)
results[['param_max_depth', 'mean_test_score']]
```

	param_max_depth	mean_test_score
0	6	0.842765
1	8	0.855150
2	10	0.847941
3	12	0.838829
4	16	0.841141
5	20	0.844135
6	24	0.843455

사용자 행동 인식

- 최적의 하이퍼 파라미터 값 찾기 실행 시간 오래 걸림

```
params = {'max_depth': [6, 8, 10, 12, 16, 20, 24], 'min_samples_split': [16, 24]}
gs = GridSearchCV(dtc, param_grid=params, scoring='accuracy', cv=5)
gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

```
0.8518883447328627
{'max_depth': 10, 'min_samples_split': 16}
```

```
estimator = gs.best_estimator_
y_hat = estimator.predict(X_test)
accuracy = accuracy_score(y_test, y_hat)
print(accuracy)
```

```
0.8632507634882932
```

사용자 행동 인식

- 중요도 높은 피처

```
importances = estimator.feature_importances_
df = pd.DataFrame(data=importances, columns=['value'])
df['name'] = X_train.columns
top20 = df.sort_values(by='value', ascending=False)[:20]
plt.barh(top20['name'], top20['value'])
```

```
<BarContainer object of 20 artists>
```