

# 인공지능

## 04. kNN, K-means, 결정트리

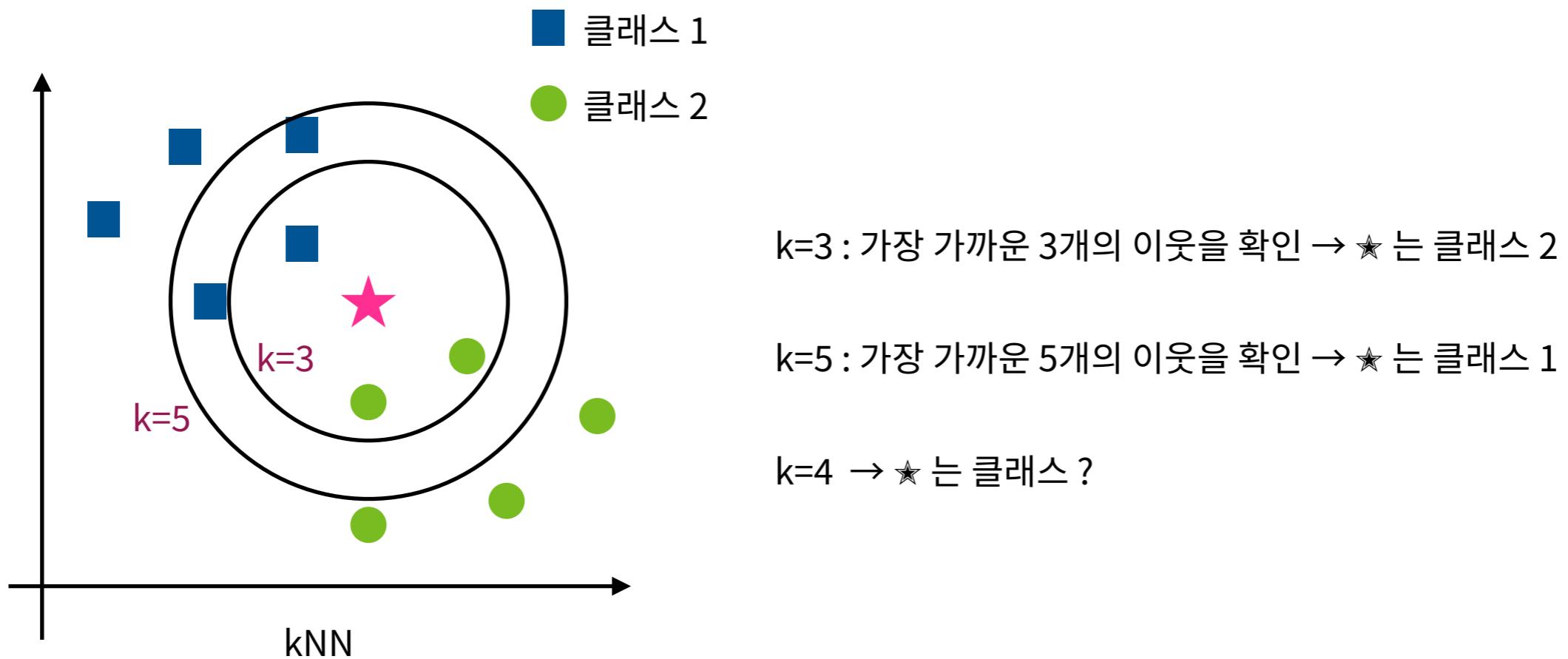
Dept. of Digital Contents



# 1. kNN

## ○ kNN(k-Nearest Neighbor)

- 지도학습의 한 종류
- 데이터를 가장 가까운 유사 속성에 따라 분류하는 거리 기반 분류 분석 모델



# 1. kNN

---

## ○ kNN(k-Nearest Neighbor)

### ■ kNN 알고리즘의 장단점

#### 장점

- 단순하고 효율적
- 기저 데이터 분포에 대한 가정을 하지 않음
- 특별한 훈련을 하지 않음(게으른 학습)
- 수치 기반 데이터 분류 작업에서 성능이 우수

#### 단점

- 모델을 생성하지 않아 특징과 클래스간 관계를 이해하는데 제한적
- 적절한 k의 선택이 필요
- 데이터가 많아지면 분류 단계가 느려짐
- 명목 특징 및 누락 데이터를 위한 추가 처리가 필요

# 1. kNN

---

## ○ 붓꽃(iris) 분류

- sklearn의 iris 데이터

- 붓꽃의 품종과 꽃잎과 꽃받침의 폭과 길이를 담은 데이터 세트
- 품종 : setosa, versicolor, virginica (품종별 50개씩 총 150 샘플)



Iris setosa



Iris versicolor



Iris virginica

# 1. kNN

---

## ○ 붓꽃(iris) 분류

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** IRIS.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cells:**
  - [ ] from sklearn.datasets import load\_iris
  - {x} # IRIS 데이터 확인  
iris = load\_iris()  
print(iris.data)
  - [ ] # 특징 이름 출력  
print(iris.feature\_names)
  - # 레이블(타겟) 출력  
print(iris.target)

# 1. kNN

---

## ○ 붓꽃(iris) 분류

```
▶ import matplotlib.pyplot as plt  
  
x_index = 0  
y_index = 1  
  
formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])  
  
plt.figure(figsize=(5,4))  
plt.scatter(iris.data[:, x_index], iris.data[:, y_index], c=iris.target)  
plt.colorbar(ticks=[0,1,2], format=formatter)  
plt.xlabel(iris.feature_names[x_index])  
plt.ylabel(iris.feature_names[y_index])  
  
plt.tight_layout()  
plt.show()
```

# 1. kNN

---

## ○ 붓꽃(iris) 분류

```
▶ from sklearn.model_selection import train_test_split  
  
X = iris.data  
y = iris.target  
  
# 훈련 데이터와 테스트 데이터를 80:20으로 분할  
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.2, random_state=4)  
  
print(X_train.shape)  
print(X_test.shape)
```

# 1. kNN

---

## ○ 붓꽃(iris) 분류

```
✓ ⏎ from sklearn.neighbors import KNeighborsClassifier  
from sklearn import metrics  
  
knn = KNeighborsClassifier(n_neighbors = 5)  
knn.fit(X_train, y_train)  
  
y_pred = knn.predict(X_test)  
scores = metrics.accuracy_score(y_test, y_pred)  
  
print(scores)  
→ 0.9666666666666667
```

# 1. kNN

---

## ○ 필기 숫자 데이터셋

### ■ sklearn의 digit 데이터

#### - MNIST

- 미국 국립표준 기술 연구소(NIST)가 미국인 대상으로 수집한 필기 숫자 데이터
- 해상도 28x28의 숫자당 7천개씩 총 7만개 샘플
- 픽셀당 [0,255]사이의 명암값을 가짐

#### - sklearn

- 해상도 8x8의 1,797자 저장
- 픽셀당 [0,16] 사이의 명암값을 가짐

# 1. kNN

---

## ○ 필기 숫자 데이터셋

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** DIGITS.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** + 코드 + 텍스트
- Code Content:**

```
▶ import matplotlib.pyplot as plt
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()

print(digits.images[0])

plt.imshow(digits.images[0], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```

# 1. kNN

---

## ○ 필기 숫자 데이터셋

```
0초   ▶ n_samples = len(digits.images)
          data = digits.images.reshape(n_samples, -1)

          print(data[0])
→ [ 0.  0.  5. 13.  9.  1.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
    15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
    0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
    0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

# 1. kNN

---

## ○ 필기 숫자 데이터셋

```
▶ X_train, X_test, y_train, y_test = train_test_split(  
    data, digits.target, test_size=0.2)  
  
# kNN 분류기  
from sklearn.neighbors import KNeighborsClassifier  
  
knn = KNeighborsClassifier(n_neighbors=6)  
  
# 학습  
knn.fit(X_train, y_train)  
  
# 예측 및 평가  
y_pred = knn.predict(X_test)  
  
# 정확도 계산  
scores = metrics.accuracy_score(y_test, y_pred)  
print(scores)
```

# 1. kNN

---

## ○ 필기 숫자 데이터셋

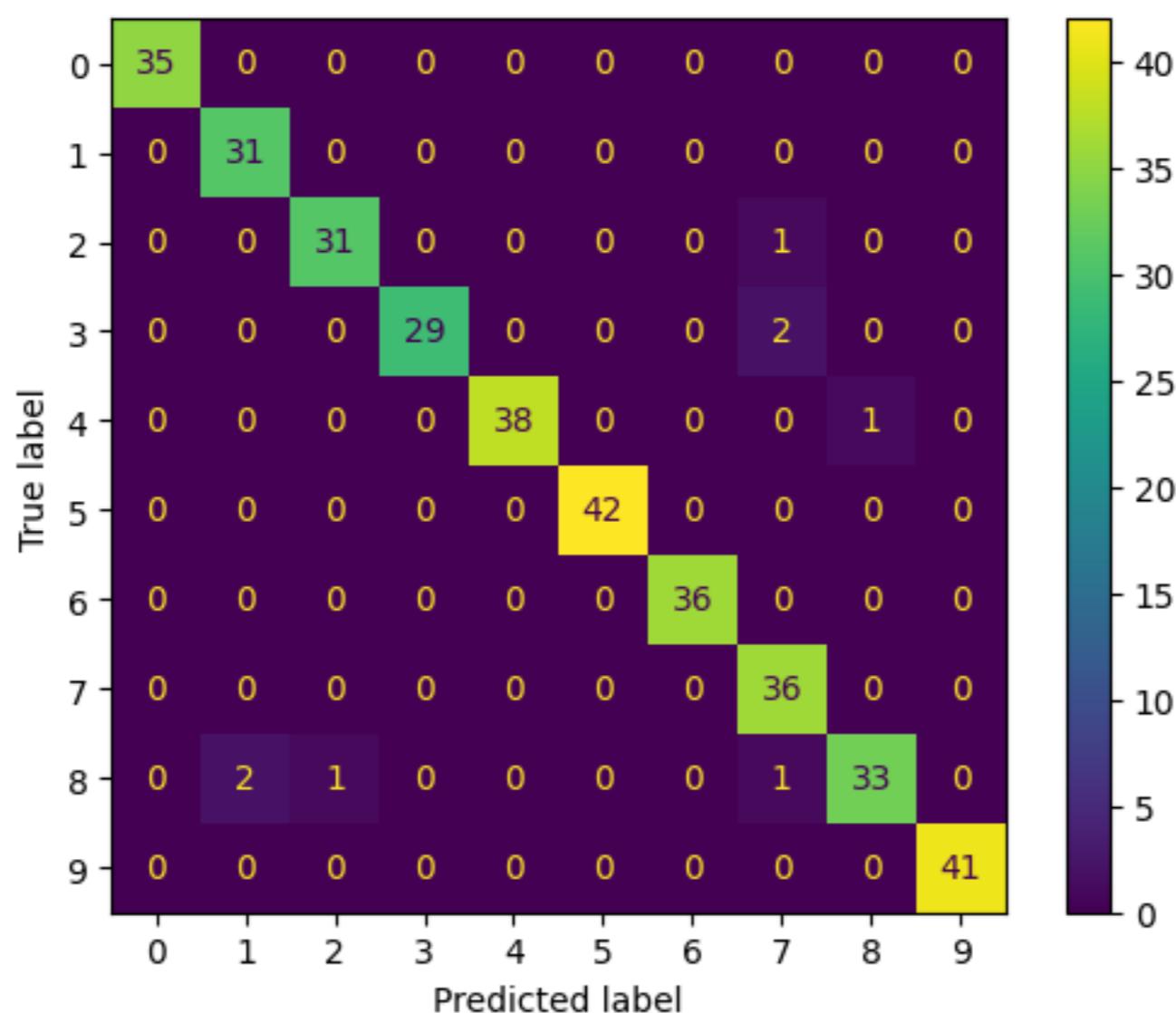
```
✓ 0초   plt.imshow(X_test[10].reshape(8,8), cmap=plt.cm.gray_r, interpolation='nearest')
    plt.show()

    y_pred = knn.predict([X_test[10]])
    print(y_pred)
```

## 2. 머신 러닝 알고리즘 성능 평가

## ○ 혼동 행렬 출력

```
▶ from sklearn.metrics import ConfusionMatrixDisplay  
disp = ConfusionMatrixDisplay.from_estimator(knn, X_test, y_test)
```



## 2. 머신 러닝 알고리즘 성능 평가

### ○ 분류 리포트

0초

```
▶ y_pred = knn.predict(X_test)
print(f"metrics.classification_report(y_test, y_pred)\n")
```



	precision	recall	f1-score	support
0	1.00	1.00	1.00	31
1	0.95	1.00	0.97	38
2	1.00	0.97	0.98	31
3	1.00	1.00	1.00	47
4	1.00	1.00	1.00	36
5	1.00	0.97	0.98	32
6	0.98	1.00	0.99	43
7	0.94	1.00	0.97	29
8	1.00	0.93	0.96	40
9	1.00	1.00	1.00	33
accuracy			0.99	360
macro avg	0.99	0.99	0.99	360
weighted avg	0.99	0.99	0.99	360

정확도(accuracy) :  $(TP+TN)/(TP+TN+FN+FP)$

정밀도(Precision) : 예측값이 얼마나 정확하게

예측되었는가,  $TP/(TP+FP)$

재현율(recall) : 실제값 중 모델이 검출한

실제값의 비율,  $TP/(TP+FN)$

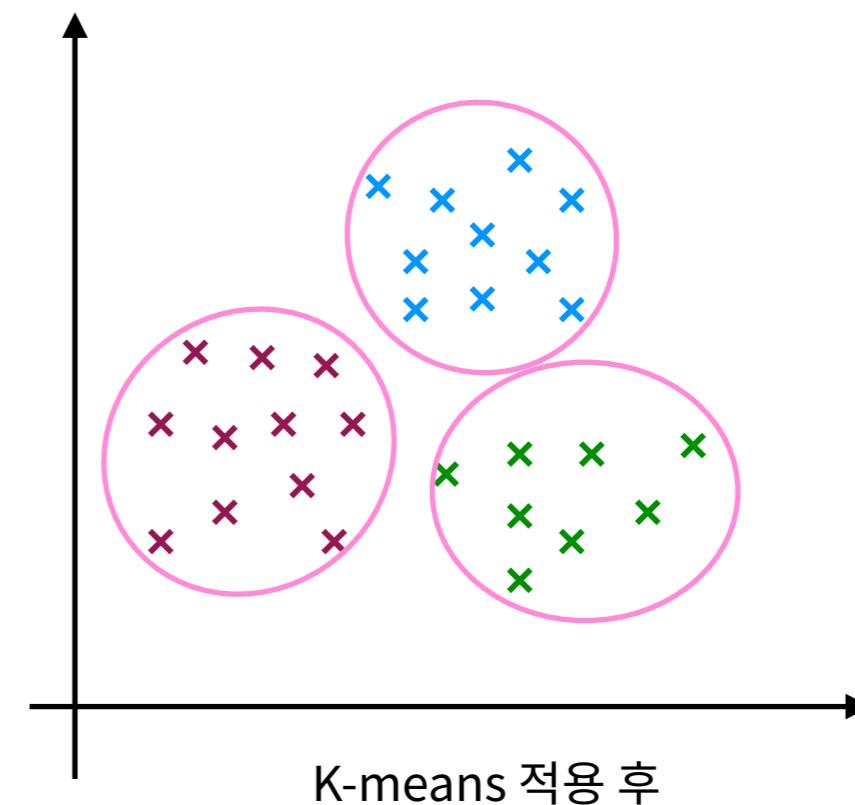
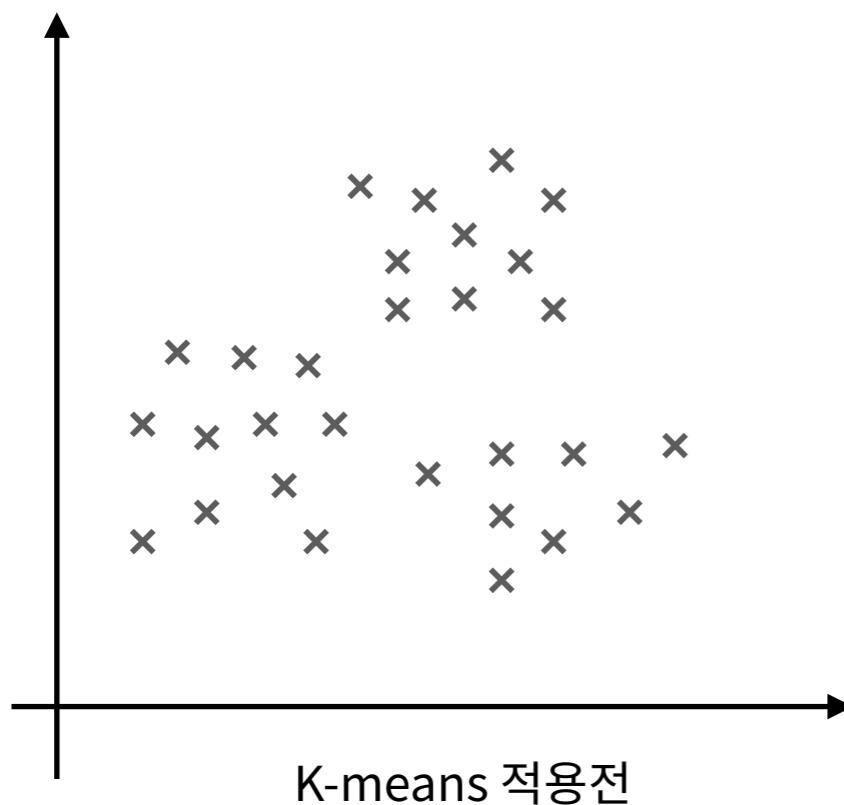
F1점수(F1-score) :

$2 \times \text{재현율} \times \text{정밀도} / (\text{재현율}_\text{정밀도})$

# 3. K-means 클러스터링

## ○ K-means 알고리즘

- 비지도 학습의 한 종류
- n개의 관측값을 거리가 최소인 k개의 그룹(클러스터)으로 분할하는 알고리즘



# 3. K-means 클러스터링

---

## ○ K-means 알고리즘

- 입력
  - $k$  : 클러스터 수
  - $D$  :  $n$ 개의 데이터
- 출력 :  $k$ 개의 클러스터
- 알고리즘
  - 초기값 설정 :  $k$ 개의 난수값을 선택하고 이를 클러스터의 중심으로 설정
  - 집합  $D$ 의 각 데이터에 대해  $k$ 개의 클러스터 중심과 거리를 계산 → 유사도가 높은 중심적으로 데이터 할당
  - 할당된 데이터를 기준으로 클러스터 중심점을 다시 계산
  - 바뀐 중심점에 대해 데이터의 소속 클러스터를 다시 계산 → 바뀌지 않을때까지 반복

# 3. K-means 클러스터링

## ○ K-means 알고리즘

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** KMeans.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cells:**
  - Cell 1: 

```
[ ] import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
```
  - Cell 2: 

```
[ ] # 데이터 준비
X = np.array([
    [6,3], [11,15], [17,12], [24,10], [20,25], [22,30],
    [85,70], [71,81], [60,79], [56,52], [81,91], [80,81]
])

plt.scatter(X[:,0], X[:,1])
```

# 3. K-means 클러스터링

---

## ○ K-means 알고리즘



```
# 클러스터 만들기
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

print(kmeans.cluster_centers_)
print(kmeans.labels_)

plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
```

### 3. K-means 클러스터링

---

#### ○ K-means 알고리즘

- 팔꿈치 방법(elbow method)
  - k를 결정하는 방법
  - k=1부터 증가시키면서 K-means 클러스터링을 수행
  - 각 k값에 대해 SSE(Sum of squared error) 값을 계산하여 감소하기 시작하는 지점을 선택

# 3. K-means 클러스터링

## ○ K-means 알고리즘

- 팔꿈치 방법(elbow method) - 수정

```
▶ # 클러스터 만들기  
# [1,10]사이의 k값 각각에 대해 kmeans 생성  
n_clusters = range(1,10)  
kmeans = [KMeans(n_clusters=i) for i in n_clusters]  
  
# SSE 계산 : kmeans의 interia_  
score = [kmeans[i].fit(X).inertia_ for i in range(len(kmeans))]  
  
# Elbow Curve 그리기  
plt.plot(n_clusters, score)  
plt.xlabel('Number of Clusters')  
plt.ylabel('Score')  
plt.title('Elbow Curve')  
plt.show()
```

# 3. K-means 클러스터링

---

## ○ K-means 알고리즘

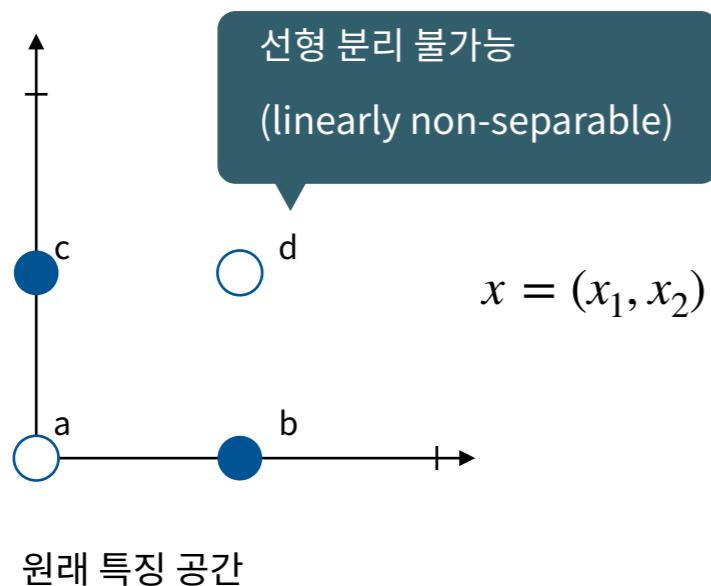
- 팔꿈치 방법(elbow method) - 수정

```
▶ o_kmeans = KMeans(n_clusters=2)
o_kmeans.fit(X)
plt.scatter(X[:,0], X[:,1], c=o_kmeans.labels_, cmap='rainbow')
```

# 4. 서포트 벡터 머신

## ○ 결정 경계

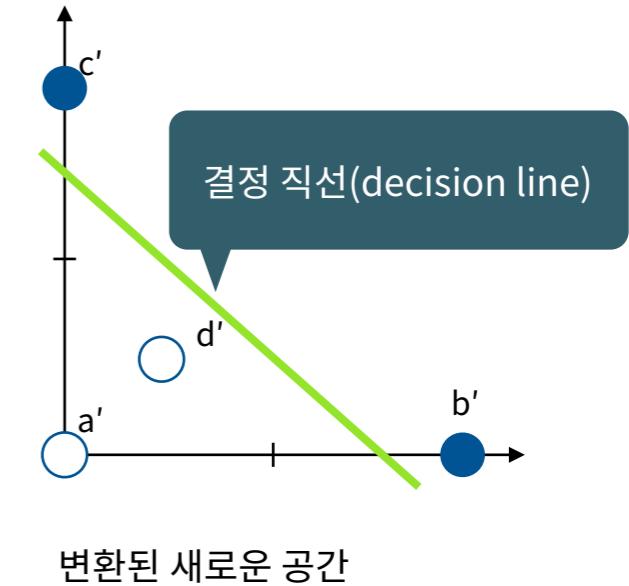
- 인식 알고리즘 : 특징 공간 변환과 특징 공간 분할로 분류 문제를 해결



$$x = (x_1, x_2) \rightarrow x' = \left( \frac{x_1}{2x_1x_2 + 0.5}, \frac{x_2}{2x_1x_2 + 0.5} \right)$$

공간변환

원래 특징 공간



변환된 새로운 공간

- 결정 경계(decision boundary)

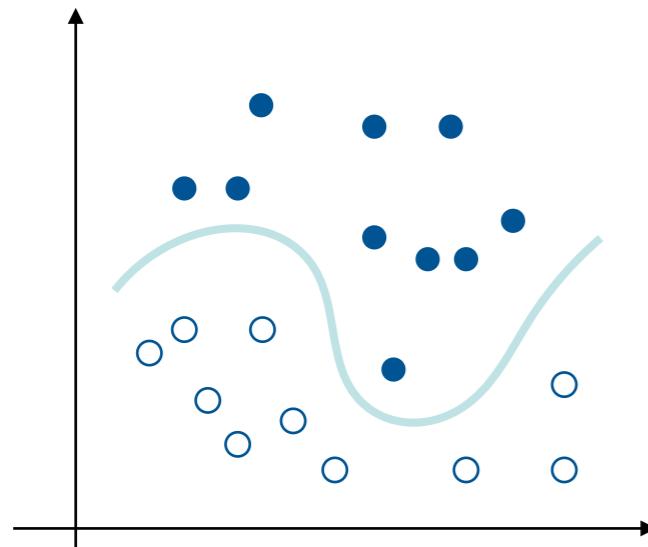
- 특징 공간을 분할하는 경계
- 2D는 선(직선 또는 곡선), 3D는 면(평면 또는 곡면)

# 4. 서포트 벡터 머신

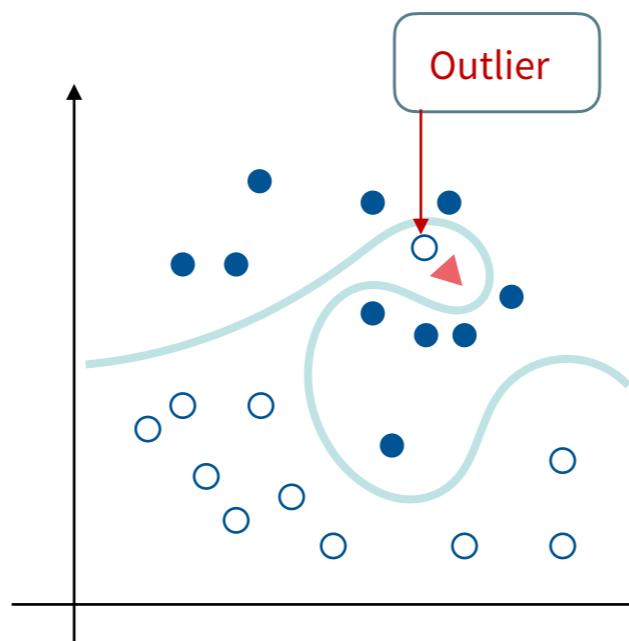
## ○ 결정 경계

### ■ 고려 사항

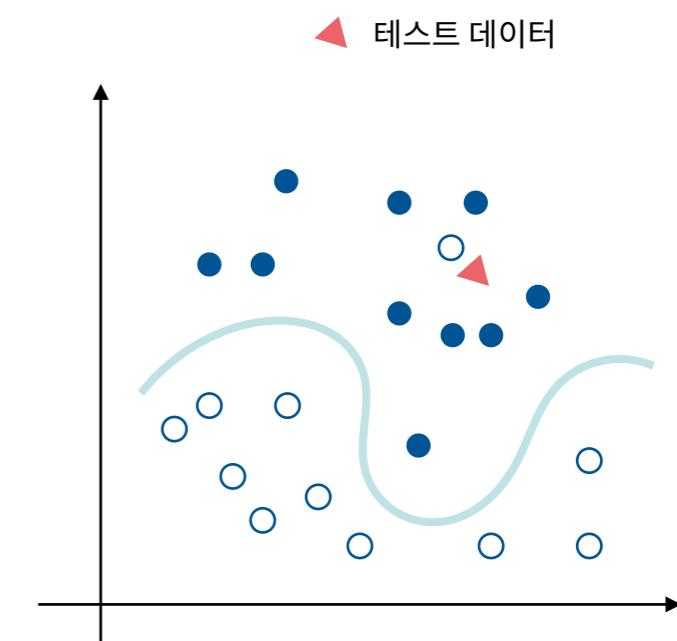
- 대부분 데이터는 선형 분리가 불가능 → 비선형 분리기(non-linear classifier)가 필요
- 과잉 적합(overfitting)을 피해야 함 → 과다하게 복잡한 결정 경계는 일반화 능력 저하



비선형 분류기



아웃라이어에 의한 과잉적합



아웃라이어를 걸러낸 경우

## 4. 서포트 벡터 머신

---

### ○ 서포트 벡터 머신(SVM: Support Vector Machine)

- 기계학습 분야 중 하나
  - 패턴인식, 자료 분석을 위한 지도 학습 모델
  - 분류와 회귀 분석에 사용
  - 데이터 마이닝 및 인공지능에 쓰이는 대표적인 알고리즘
- 장점
  - 과적합을 피할 수 있고, 분류 성능이 괜찮음
  - 잡음에 강하고 데이터 특성이 적어도 좋은 성능
- 단점
  - 커널 함수 선택이 명확하지 않음
  - 계산량 부담이 있으며, 데이터 특성의 스케일링에 민감

## 4. 서포트 벡터 머신

---

### ○ 서포트 벡터 머신(SVM: Support Vector Machine)

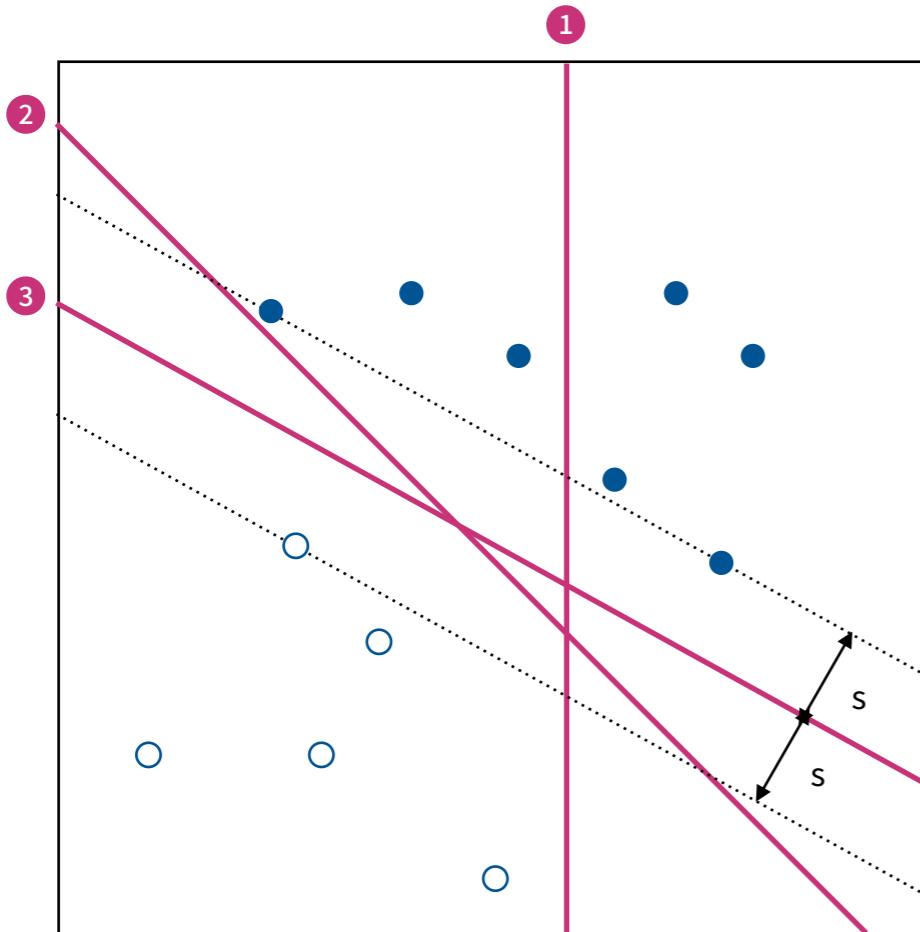
#### ■ SVM

- 여백(margin) : 결정 경계에서 각 부류 데이터까지의 거리
- 여백을 최대화하는 결정 경계를 찾음
- C : 여백 크기와 잘못 분류한 샘플 수간의 trade-off를 조정하는 하이퍼매개변수
  - 값이 크면 : 정확률은 높지만 일반화 능력은 떨어짐
  - 값이 작으면 : 정확률은 낮아지지만 일반화 능력은 커짐

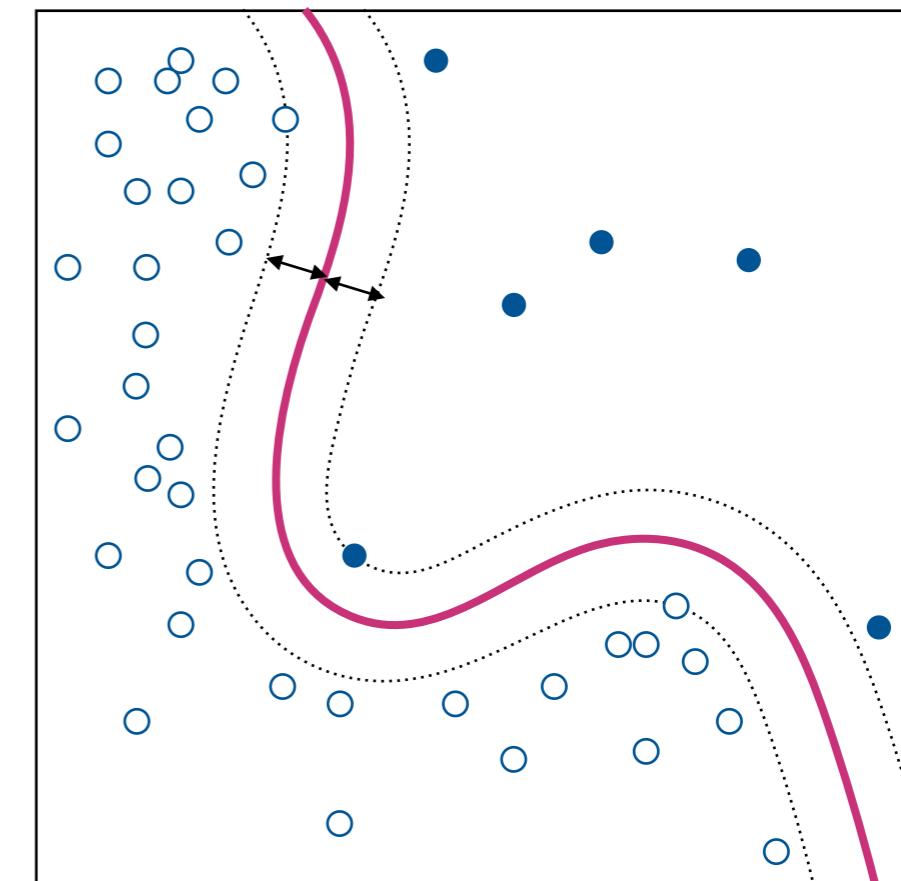
# 4. 서포트 벡터 머신

## ○ 서포트 벡터 머신(SVM: Support Vector Machine)

### ■ SVM



선형 SVM



비선형 SVM

# 4. 서포트 벡터 머신

## ○ 서포트 벡터 머신(SVM: Support Vector Machine)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** CO SVM.ipynb ☆
- Menu Bar:** 파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨
- Toolbar:** + 코드 + 텍스트, search icon, {x} icon, key icon, folder icon.
- Code Cells:**
  - Cell 1 (Top):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```
  - Cell 2 (Bottom):

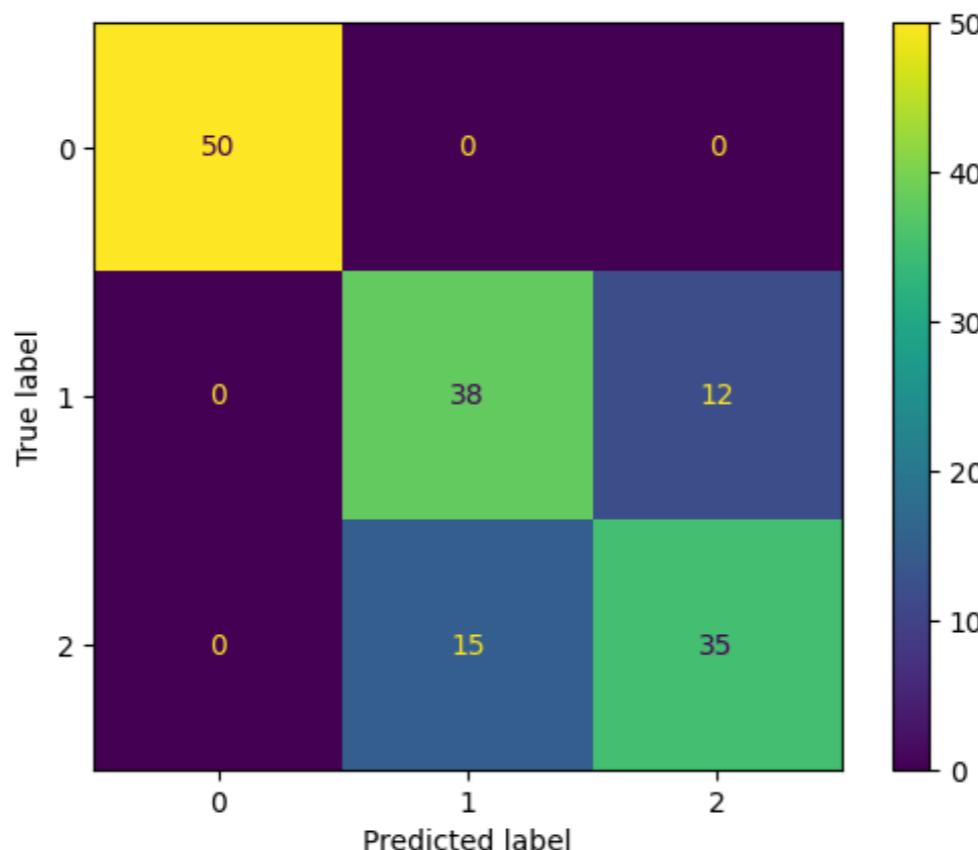
```
# IRIS 데이터
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

# 기본적인 SVM 모델
clf = svm.SVC(kernel='linear', C=1)
clf.fit(X, y)
```

# 4. 서포트 벡터 머신

## ○ 서포트 벡터 머신(SVM: Support Vector Machine)

```
▶ # 혼동 행렬로 정확도 확인  
from sklearn.metrics import ConfusionMatrixDisplay  
  
y_pred = clf.predict(X)  
disp = ConfusionMatrixDisplay.from_estimator(clf, X, y)
```



# 4. 서포트 벡터 머신

## ○ 서포트 벡터 머신(SVM: Support Vector Machine)

▶ # rbf : radial basis function SVC 활용

```
셀 실행 (⌘/Ctrl+Enter)
셀이 이 세션에서 실행되지 않음
clf = svm.SVC(kernel='rbf', gamma=0.7, C=1, max_iter=10000)
# gamma : sigma 제곱에 해당하는 스케일 파라미터
# max_iter : 학습 반복횟수
```

```
clf.fit(X, y)
y_pred = clf.predict(X)
disp = ConfusionMatrixDisplay.from_estimator(clf, X, y)
```

▶ # polynomial 활용

```
clf = svm.SVC(kernel='poly', degree=3, gamma='auto', C=1, max_iter=10000)
# degree : 다항식의 차수
```

```
clf.fit(X, y)
y_pred = clf.predict(X)
disp = ConfusionMatrixDisplay.from_estimator(clf, X, y)
```

# 5. 의사 결정 트리

---

## ○ 의사 결정 트리(DT: Decision Tree)

- 기계학습 분야 중 하나

- 통계, 데이터 마이닝, 머신 러닝에 사용되는 지도 학습 모델
- 분류와 회귀 분석, 다중 출력까지 가능
- 질문을 하면서 질문의 대답에 따라 입력을 분류하는 알고리즘

- 장점

- 인간의 사고를 모방하여 이해하고 해석하기 쉬움
- 숫자 및 범주 데이터를 모두 처리할 수 있음
- 데이터를 분류하는 논리의 흐름을 시각적으로 볼 수 있음

# 5. 의사 결정 트리

## ○ 의사 결정 트리(DT: Decision Tree)

- 트리 구축 방법

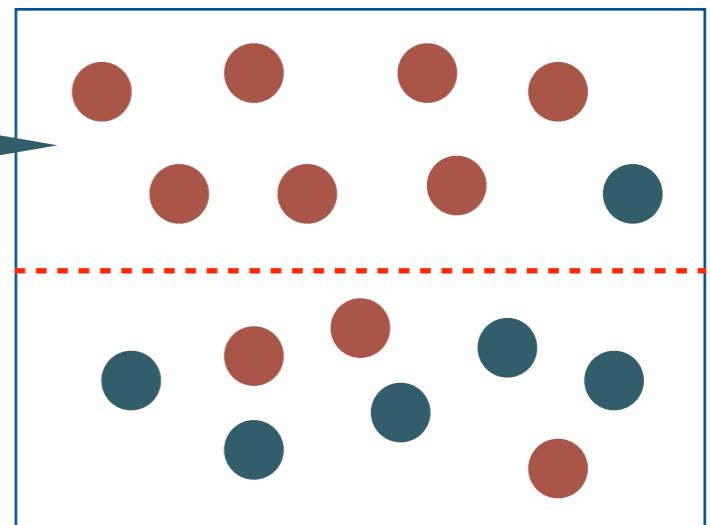
- 최대 변별력을 갖는 질문으로 데이터를 분리
- 불순도(impurity) : 서로 다른 데이터가 얼마나 섞여 있는지를 나타냄
- 불순도 사용 지표 : 엔트로피(Entropy), 지니 계수(Gini Index) 등

- 지니 계수

- 해당 범주에 각 클래스가 존재하는 확률의 제곱의 합을 1에서 빼는 것
- 0일 때 가장 순도가 높음

$$I(A) = 1 - \sum_{K=1}^m p_k^2$$

지니 계수 =  $1 - \{ (1/8)^2 + (7/8)^2 \}$   
= 0.21875



# 5. 의사 결정 트리

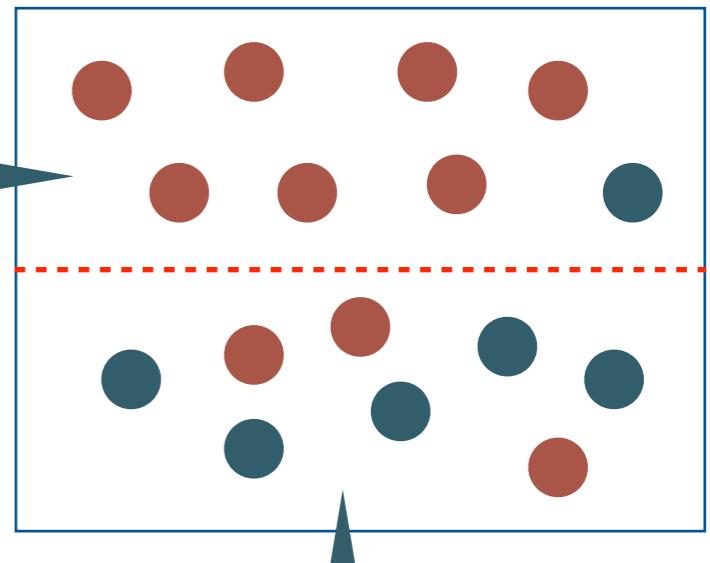
## ○ 의사 결정 트리(DT: Decision Tree)

### ■ 엔트로피

- 클래스의 확률과 클래스의 개수를 계산
- 0일 때 가장 순도가 높음

$$E(S) = \sum_{K=1}^m p_k \log_2 \frac{1}{p_k} = \sum_{K=1}^m -p_k \log_2 p_k$$

엔트로피 =  $\{-(7/8) \cdot \log_2(7/8) - (1/8) \cdot \log_2(1/8)\} = 1.0573549$



엔트로피 = 1.6367889

# 5. 의사 결정 트리

### ○ 의사 결정 트리를 이용한 봇꽃 분류

DT\_IRIS.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

import numpy as np  
from sklearn import datasets, metrics  
from sklearn.metrics import accuracy\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier

# IRIS 데이터 읽어와 Train, Test 데이터로 나누기  
iris = datasets.load\_iris()  
X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=50,  
test\_size=0.2)

# 5. 의사 결정 트리

## ○ 의사 결정 트리를 이용한 봇꽃 분류

```
▶ # 의사 결정 트리 모델 생성  
clf = DecisionTreeClassifier()  
clf.fit(X_train, y_train)  
  
y_pred = clf.predict(X_train)  
print('학습 데이터 정확도: ', accuracy_score(y_true=y_train, y_pred=y_pred))  
  
y_pred = clf.predict(X_test)  
print('테스트 데이터 정확도: ', accuracy_score(y_true=y_test, y_pred=y_pred))
```

# 5. 의사 결정 트리

---

## ○ 의사 결정 트리를 이용한 붓꽃 분류

```
▶ !pip install graphviz
import graphviz
from sklearn.tree import export_graphviz

dot_data = export_graphviz(clf,
                           out_file=None,
                           class_names=iris.target_names,
                           feature_names=iris.feature_names)
graph = graphviz.Source(dot_data)
graph.render(filename='iris', directory='drive/MyDrive/Colab Notebooks/',
            format='png')
```

# 5. 의사 결정 트리

## ○ 의사 결정 트리를 이용한 붓꽃 분류

