



5장 분류와 군집화로 이해하는 지도학습과 비지도 학습

이장에서 배울 것들

- 데이터를 여러 그룹으로 구분할 수 있는 방법은 무엇일까.
- 군집화는 어떤 방법을 통해 비슷한 특징을 가지는 데이터들끼리 묶을 수 있나.
- 지도 학습과 비지도 학습의 차이는 무엇인가.
- 데이터 기반 학습의 성능 평가지표란 무엇인가.

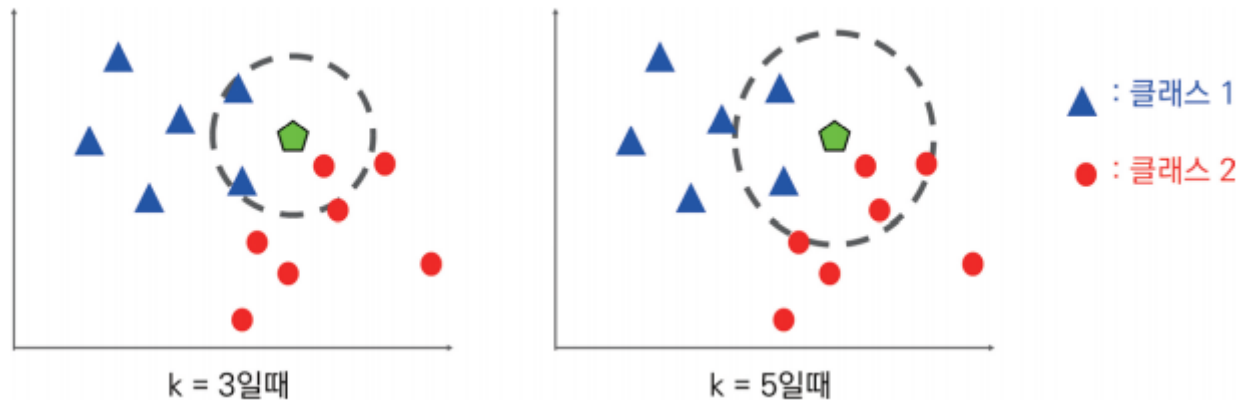
5.1 k-NN 알고리즘과 분류문제

- 선형회귀에 이어 **분류**classification와 **군집화**clustering에 대해서 살펴보도록 하자.
- 두 가지 방법 모두 데이터를 비슷한 집단으로 묶는다는 점에서는 유사
 - 분류는 소속 집단의 정보를 이미 알고 있는 상태에서 비슷한 집단으로 묶는 방법
 - 군집화란 소속 집단의 정보가 없는 상태에서 비슷한 집단으로 묶는 방법

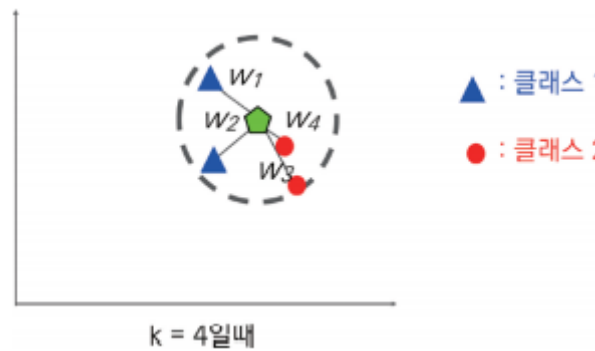
- 분류 문제의 가장 간단한 사례로 k-NN 알고리즘을 먼저 살펴보자.
- 우리 주변에서 볼 수 있는 개의 품종 중에는 **닥스훈트**^{dachshund}와 **사모예드**^{samoyed}라는 종이 있는데, 두 종은 몸의 높이와 길이 비율이 서로 다름
- 일반적으로 닥스훈트는 몸통 길이에 비해서 높이가 낮고, 사모예드의 경우 몸통 길이와 높이가 비슷한 특징을 가짐
- 몸통의 길이와 높이 **특징**^{feature}을 각각 x_1, x_2 라고 두고 이 개들의 표본 집합에 대하여 x_1, x_2 를 측정하면 아래 그림의 오른쪽과 같은 산포도 그래프를 얻을 수 있음



- 산포도 그래프의 위쪽에 분포한 파란색 점은 사모예드 종이고, 아래쪽의 붉은 점들은 닥스훈트 종
- 데이터를 효과적으로 분류하려면 어떤 기법을 적용하는 것이 바람직할까?
 - k-NN 알고리즘에서 k-NN은 **k-최근접 이웃**(*k-Nearest Neighbor*)의 약자로 특징 공간에 분포하는 데이터에 대하여 k개의 가장 가까운 이웃을 살펴보고 다수결 방식으로 데이터의 레이블을 할당 하는 분류방식



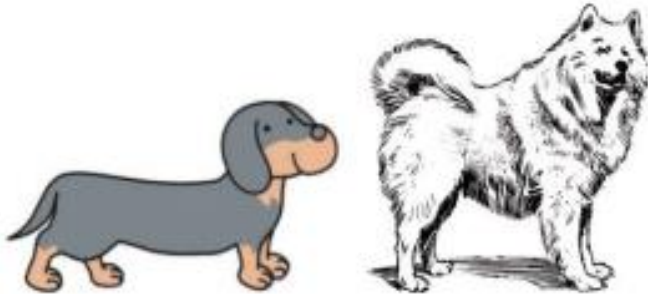
- k 가 4와 같이 짝수이고 클래스 1과 클래스 2의 개수가 같은 경우에도 새 데이터의 클래스를 판정하는 방법
 - 예를 들어 2개의 빨간색 원은 다른 2개의 파란색 삼각형보다 더 가까이 있다는 것을 고려해 클래스 2로 분류하는 것



- k-NN 방법은 특징 공간에 있는 모든 데이터에 대한 정보가 필요
 - 데이터 인스턴스, 클래스, 특징의 요소들의 개수가 많다면, 많은 메모리 공간과 계산 시간이 필요하다는 단점
 - 알고리즘이 매우 단순하고 직관적이며, 사전 학습이나 특별한 준비 시간이 필요 없다는 점은 장점

5.2 k-NN 알고리즘을 위한 데이터 준비하기

- 이제 다음과 같이 닥스훈트와 사모예드의 데이터를 바탕으로 k-NN 알고리즘을 익혀보도록 하자



닥스훈트 8마리의 길이와 높이								
길이	77	78	85	83	73	77	73	80
높이	25	28	29	30	21	22	17	35
사모예드 8마리의 길이와 높이								
길이	75	77	86	86	79	83	83	88
높이	56	57	50	53	60	53	49	61

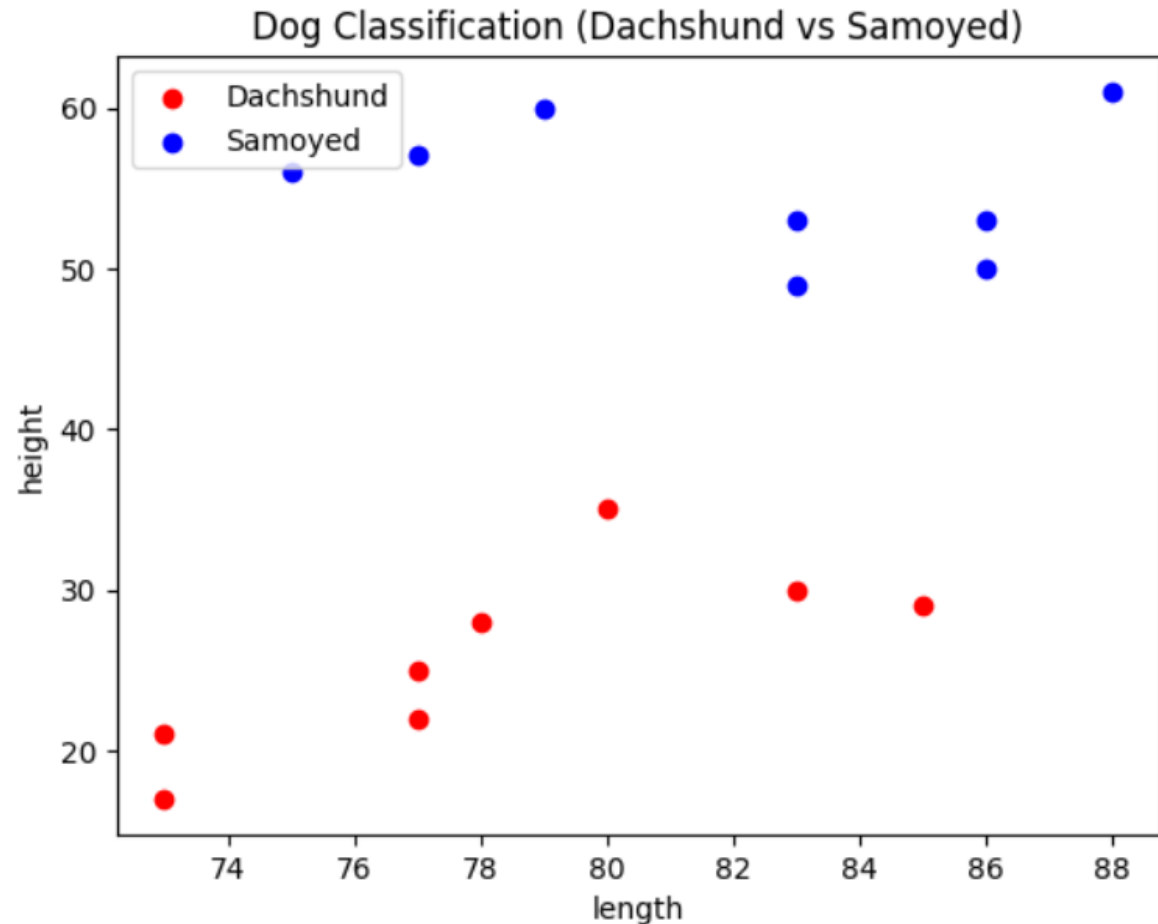

```
import matplotlib.pyplot as plt
import numpy as np
```

```
dach_size = np.array(
    [[77, 25],
     [78, 28],
     [85, 29],
     [83, 30],
     [73, 21],
     [77, 22],
     [73, 17],
     [80, 35]])
```

```
samo_size = np.array(
    [[75, 56],
     [77, 57],
     [86, 50],
     [79, 60],
     [83, 53],
     [83, 49],
     [88, 61],
     [86, 53]])
```

```
# plt.scatter( x축_data, y축_data )
plt.scatter(dach_size[:,0], dach_size[:,1], color='red', label='Dachshund')
plt.scatter(samo_size[:,0], samo_size[:,1], color='blue', label='Samoyed')
plt.xlabel('length')
plt.ylabel('height')
plt.title('Dog Classification (Dachshund vs Samoyed)')
plt.legend()
```

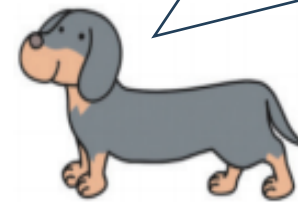
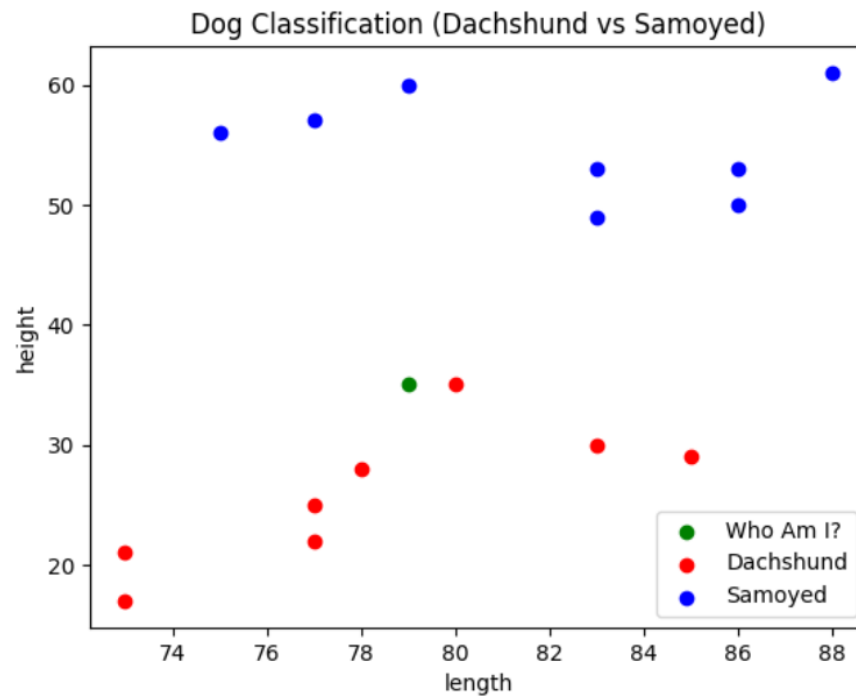
<matplotlib.legend.Legend at 0x79d840219510>



- 이제 길이와 높이가 각각 79, 35인 새로운 데이터가 들어왔고 이 특징을 가진 개의 종류를 모를 경우, 이 데이터는 닥스훈트와 사모예드 중에서 어떤 개로 분류하는 것이 바람직할지 살펴 보자

```
new_dog = [79, 35]
plt.scatter(new_dog[0], new_dog[1], color='green', label="Who Am I?")
plt.scatter(dach_size[:,0], dach_size[:,1], color='red', label='Dachshund')
plt.scatter(samo_size[:,0], samo_size[:,1], color='blue', label='Samoyed')
plt.xlabel('length')
plt.ylabel('height')
plt.title('Dog Classification (Dachshund vs Samoyed)')
plt.legend()
```

<matplotlib.legend.Legend at 0x7f3aa40b7e80>



비슷하게 생긴(특징이 비슷한 곳에 있는) 녀석들은 닥스훈트네요!

5.3 k-NN 분류기를 실행하자

- k-NN 알고리즘을 적용해 보자

KNN 분류기를 준비하여 학습을 시키자

```
[4] from sklearn.neighbors import KNeighborsClassifier  
  
     classes = { 0: 'Dachshund', 1: 'Samoyed'}  
  
     k = 3  
     knn = KNeighborsClassifier(n_neighbors = k)
```

- 우선 모델의 훈련을 위해서 데이터를 준비한다.

```
zeros = np.zeros(len(dach_size))  
dachs = np.column_stack((dach_size, zeros))
```

dachs

```
array([[77., 25., 0.],  
       [78., 28., 0.],  
       [85., 29., 0.],  
       [83., 30., 0.],  
       [73., 21., 0.],  
       [77., 22., 0.],  
       [73., 17., 0.],  
       [80., 35., 0.]])
```

```
[7] ones = np.ones(len(samo_size))  
samos = np.column_stack((samo_size, ones))
```

samos

```
array([[75., 56., 1.],  
       [77., 57., 1.],  
       [86., 50., 1.],  
       [79., 60., 1.],  
       [83., 53., 1.],  
       [83., 49., 1.],  
       [88., 61., 1.],  
       [86., 53., 1.]])
```



```
dogs = np.concatenate((dachs, samos))
```

dogs

```
array([[77., 25., 0.],  
       [78., 28., 0.],  
       [85., 29., 0.],  
       [83., 30., 0.],  
       [73., 21., 0.],  
       [77., 22., 0.],  
       [73., 17., 0.],  
       [80., 35., 0.],  
       [75., 56., 1.],  
       [77., 57., 1.],  
       [86., 50., 1.],  
       [79., 60., 1.],  
       [83., 53., 1.],  
       [83., 49., 1.],  
       [88., 61., 1.],  
       [86., 53., 1.]])
```

- 학습을 위한 데이터 준비

$f(X) \rightarrow y$: X 를 주면 y 를 예측

X : size

y : label = 분류 그룹의 정답

```
▶ X = dogs[:, 0:2]  
y = dogs[:, 2]
```

X, y

```
↔ (array([[77., 25.],  
         [78., 28.],  
         [85., 29.],  
         [83., 30.],  
         [73., 21.],  
         [77., 22.],  
         [73., 17.],  
         [80., 35.],  
         [75., 56.],  
         [77., 57.],  
         [86., 50.],  
         [79., 60.],  
         [83., 53.],  
         [83., 49.],  
         [88., 61.],  
         [86., 53.]]),  
   array([0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.]))
```

- 선형회귀와 마찬가지로 학습은 fit(), 예측은 predict()
- 아래의 코드에서 dog_classes는 키key 값이 0일때 'Dachshund', 1일때 'Samoyed'라는 레이블을 가지도록 생성한 딕셔너리
- 딕셔너리는 fit() 함수를 통해 생성한 모델이 잘 적용되는가를 보기 쉽게 출력하기 위하여 사용

```
▶ knn.fit(X, y)
```

```
↔ KNeighborsClassifier ⓘ ?  
KNeighborsClassifier(n_neighbors=3)
```

```
▶ my_dog = [ [79, 35] ]  
  
y_pred = knn.predict(my_dog)  
  
y_pred
```

```
↔ array([0.])
```

```
▶ print(classes[y_pred[0]])
```

```
↔ Dachshund
```

```
▶ y_pred = knn.predict(X)  
y_pred
```

```
↔ array([0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.])
```



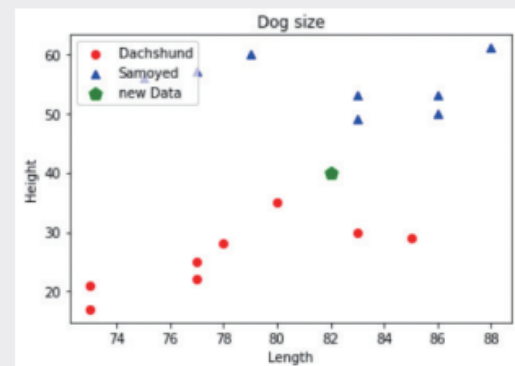
도전문제 5.1

1. 길이 82, 높이 40인 데이터를 다음과 같이 녹색 점으로 표시하여라.
2. 위의 데이터에 k-NN 알고리즘을 적용하여라. 이때 k값을 1, 5, 9로 두어서 다음과 같이 분류가 됨을 확인하여라.

k = 1 일때 판정 결과: Dachshund

k = 5 일때 판정 결과: Dachshund

k = 9 일때 판정 결과: Samoyed



5.4 k-NN 활용 예제 - 붓꽃 데이터 준비하기

- 붓꽃^{iris}은 관상용으로도 재배되는 아름다운 꽃이며 크기와 색상이 다른 여러 종이 있음
- 예제에서는 사이킷런 라이브러리에서 제공하는 붓꽃 데이터 세트를 사용해 볼 예정



- 세 붓꽃 종의 이름은 **Versicolor**, **Setosa**, **Virginica**
- 각 종에 따라 꽃받침의 길이와 너비, 꽃잎의 길이와 너비가 약간씩 차이
- 꽃받침과 꽃잎의 크기를 측정한 데이터를 기반으로 새로운 종을 분류하는 k-NN 모델 구축

```
from sklearn.datasets import load_iris

iris = load_iris()

iris.data.shape
```


(150, 4)


- 각각의 속성은 **꽃받침 길이**sepal length와 **너비**sepal width 및 **꽃잎 길이**petal length와 **너비**petal width를 cm 단위로 측정한 값

```
iris.data
```


```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.8, 3.1, 1.5, 0.1]]
```


- target은 분류 그룹

 iris.target

 array([0,
0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
2, 2])

- feature_names는 각 특징에 부여된 이름

 iris.feature_names

 ['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)']

- 넘파이 다차원 배열인 iris.data를 조금 다루기 쉽게 판다스의 데이터프레임으로 만들어서 보도록 하자
- 다섯 번째 속성이 0, 1, 2의 소속 클래스의 값을 갖는 target 속성임을 알 수 있음

```
import pandas as pd
iris_df = pd.DataFrame( iris.data, columns = iris.feature_names)
iris_df['target'] = iris.target
iris_df
```

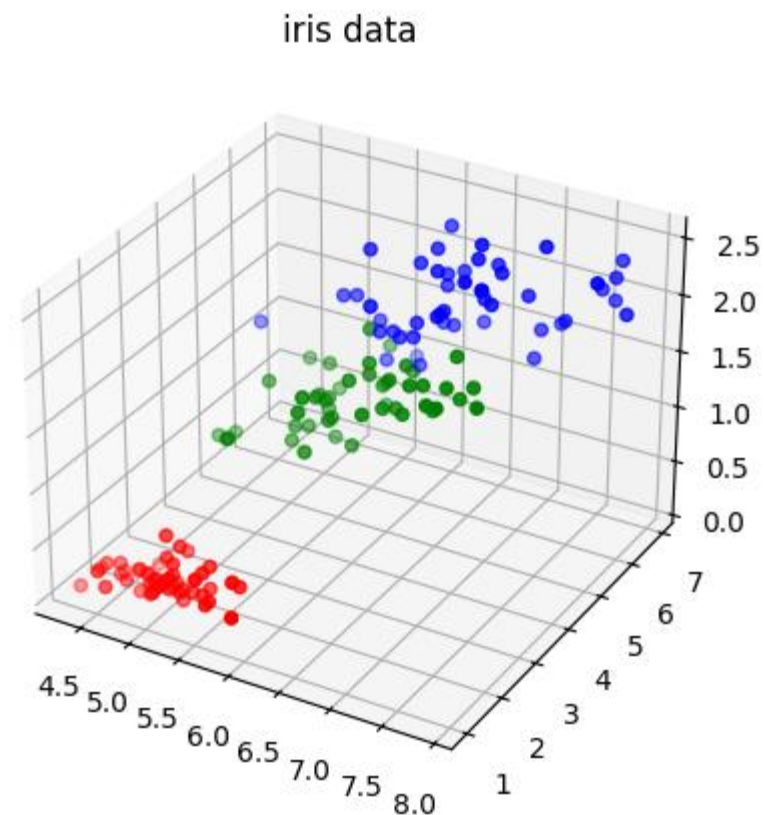
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Next steps: [Generate code with iris_df](#) [View recommended plots](#) [New interactive sheet](#)

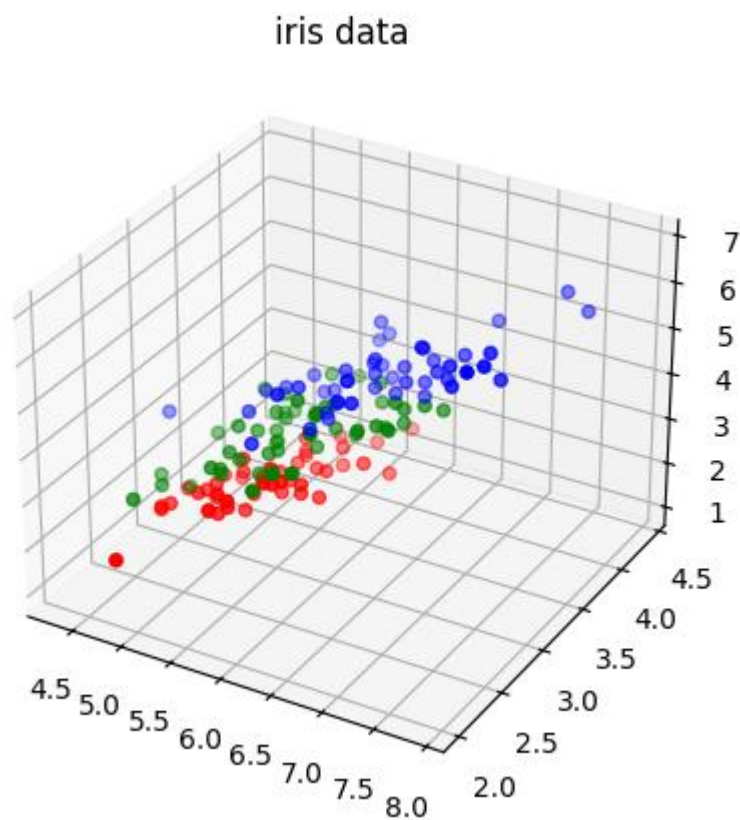
• 붓꽃 데이터 가시화

```
▶ def draw_iris_data(x_axis, y_axis, z_axis):  
    x = iris_df.iloc[:, x_axis]  
    y = iris_df.iloc[:, y_axis]  
    z = iris_df.iloc[:, z_axis]  
  
    fig = plt.figure()  
    ax = fig.add_subplot(111, projection = '3d')  
  
    ax.scatter(x[0:50], y[0:50], z[0:50], color='red')  
    ax.scatter(x[50:100], y[50:100], z[50:100], color='green')  
    ax.scatter(x[100:150], y[100:150], z[100:150], color='blue')  
  
    ax.set_title('iris data')  
    return ax  
  
draw_iris_data(0, 2, 3)
```

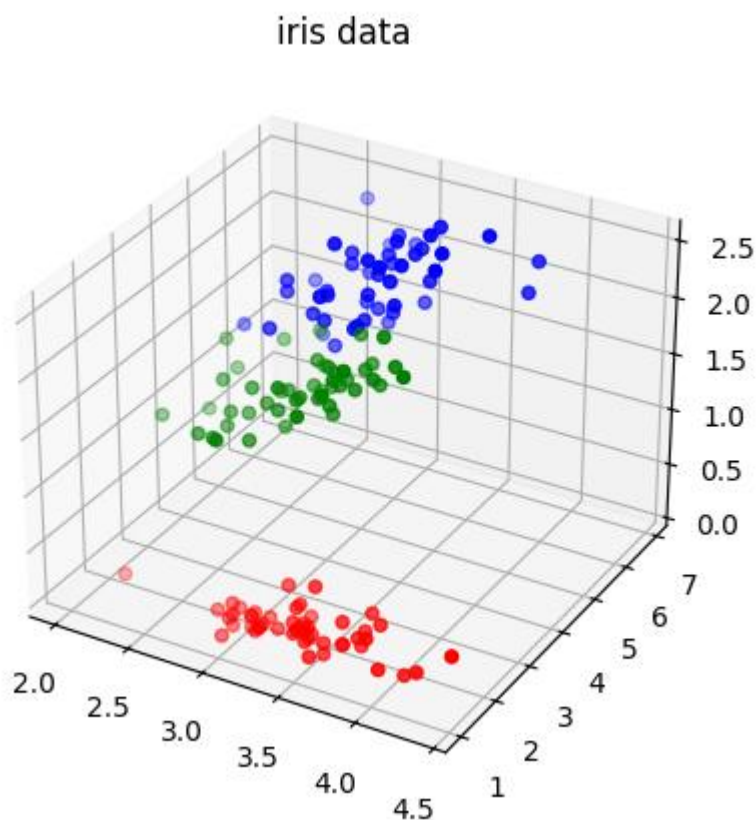


- 붓꽃 데이터 가시화

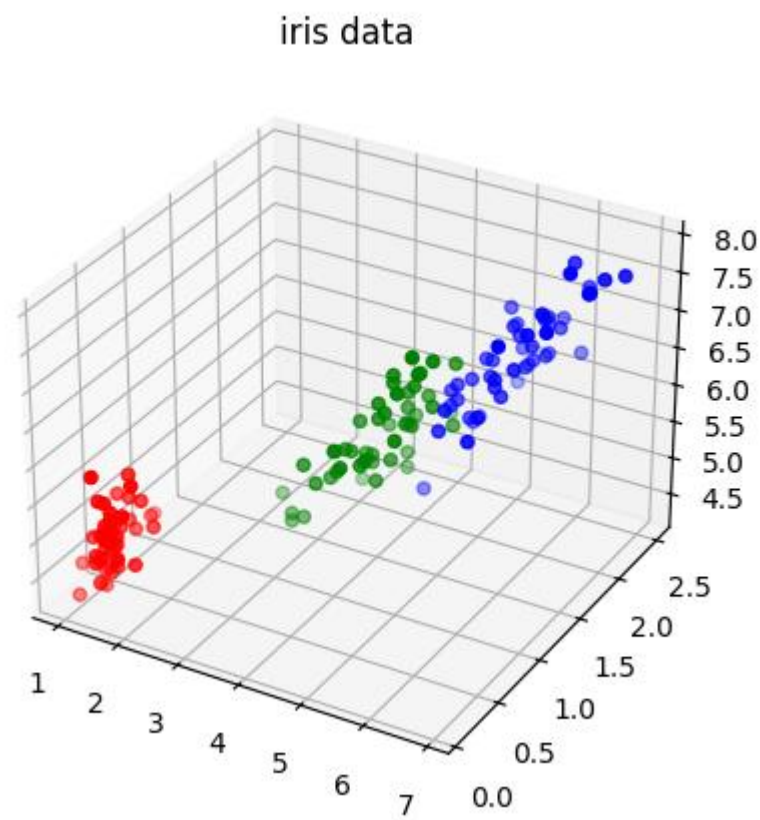
```
draw_iris_data(0, 1, 2)
```



```
draw_iris_data(1, 2, 3)
```




```
draw_iris_data(2, 3, 0)
```



5.5 k-NN 활용 예제 - 붓꽃 데이터로 학습하기

- describe() 메소드를 통해서 그 속성을 상세히 살펴보고 다음과 같은 결과를 얻을 수 있음

 iris_df.describe()



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

• X, y 데이터 준비

```
▶ from sklearn.model_selection import train_test_split  
X = iris_df.iloc[:, 0:4] # 0, 1, 2, 3 칼럼  
y = iris_df.iloc[:, -1] # 마지막 칼럼
```

X, y

```
⇒ (   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  
0           5.1           3.5           1.4           0.2  
1           4.9           3.0           1.4           0.2  
2           4.7           3.2           1.3           0.2  
3           4.6           3.1           1.5           0.2  
4           5.0           3.6           1.4           0.2  
..  
145          6.7           3.0           5.2           2.3  
146          6.3           2.5           5.0           1.9  
147          6.5           3.0           5.2           2.0  
148          6.2           3.4           5.4           2.3  
149          5.9           3.0           5.1           1.8
```

[150 rows x 4 columns],

```
0      0  
1      0  
2      0  
3      0  
4      0  
..  
145    2  
146    2  
147    2  
148    2  
149    2
```

Name: target, Length: 150, dtype: int64)

- X, y 데이터를 학습용과 검증용으로 분리하여 준비

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
print(X_train.shape, X_test.shape)
```

```
(105, 4) (45, 4)
```

y_test



target

101 2

96 1

144 2

15 0

67 1

49 0

66 1

129 2

44 0

22 0

38 0

8 0

3 0

97 1

- X_train, y_train 데이터로 학습 실시

```
▶ k = 5  
knn = KNeighborsClassifier(n_neighbors = k)  
  
knn.fit(X_train, y_train)
```



▼ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()

- X_test를 이용하여 예측 실시

```
▶ y_test_predicted = knn.predict(X_test)  
  
np.array(y_test), y_test_predicted
```



```
(array([2, 1, 2, 0, 1, 0, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 1, 2, 0,  
        1, 1, 0, 0, 2, 2, 2, 2, 1, 1, 2, 0, 1, 2, 0, 0, 1, 1, 0, 0, 0, 0,  
        2])),  
array([2, 1, 2, 0, 1, 0, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 1, 2, 0,  
        2, 1, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 2, 0, 0, 1, 1, 0, 0, 0, 0,  
        2]))
```

- 예측 정확도

```
▶ from sklearn import metrics  
  
accuracy = metrics.accuracy_score(y_test, y_test_predicted)  
  
accuracy
```

```
↔ 0.9333333333333333
```



도전문제 5.2

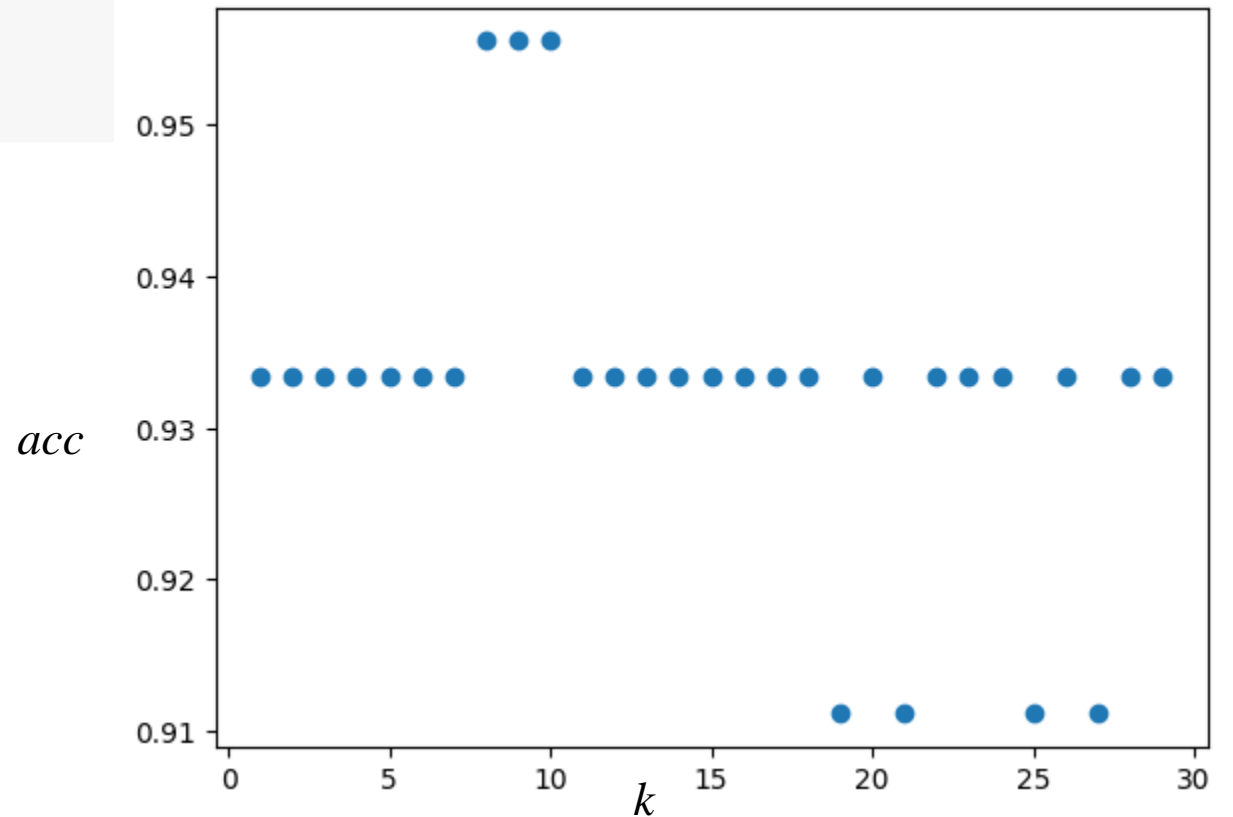
위의 코드에서 k 값을 수정하여 n_neighbors가 1, 5, 10, 20, 30일 때의 정확도를 각각 다음과 같이 출력하여라.

n_neighbors가 1일때 정확도: 0.956
n_neighbors가 5일때 정확도: 0.978
n_neighbors가 10일때 정확도: 0.956
n_neighbors가 20일때 정확도: 0.978
n_neighbors가 30일때 정확도: 0.933

```
for k in range(1, 10) :  
    knn = KNeighborsClassifier(n_neighbors = k)  
    knn.fit(X_train, y_train)  
  
    y_test_predicted = knn.predict(X_test)  
    print('k = ', k, 'accuracy = ', metrics.accuracy_score(y_test, y_test_predicted))
```

```
k = 1 accuracy = 0.9333333333333333  
k = 2 accuracy = 0.9333333333333333  
k = 3 accuracy = 0.9333333333333333  
k = 4 accuracy = 0.9333333333333333  
k = 5 accuracy = 0.9333333333333333  
k = 6 accuracy = 0.9333333333333333  
k = 7 accuracy = 0.9333333333333333  
k = 8 accuracy = 0.9555555555555556  
k = 9 accuracy = 0.9555555555555556
```

```
▶ acc_list = []  
k_range = range(1, 30)  
for k in k_range :  
    knn = KNeighborsClassifier(n_neighbors = k)  
    knn.fit(X_train, y_train)  
  
    y_test_predicted = knn.predict(X_test)  
    acc = metrics.accuracy_score(y_test, y_test_predicted)  
    acc_list.append(acc)  
    print('k = ', k, 'accuracy = ', acc)  
  
plt.scatter(k_range, acc_list)
```



5.6 새로운 꽃에 대해서 모델을 적용하고 분류해 보자

- 이번에는 훈련 데이터와 검증 데이터를 나누어서 훈련하지 않고 사용가능한 모든 데이터를 사용해서 모델을 학습

```
[47] k = 5
      knn = KNeighborsClassifier(n_neighbors = k)

      knn.fit(X, y)
```



▼ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()

- 테스트 데이터에 적용 (사실 이미 학습한 내용)



```
accuracy = metrics.accuracy_score(y_test, y_test_predicted)

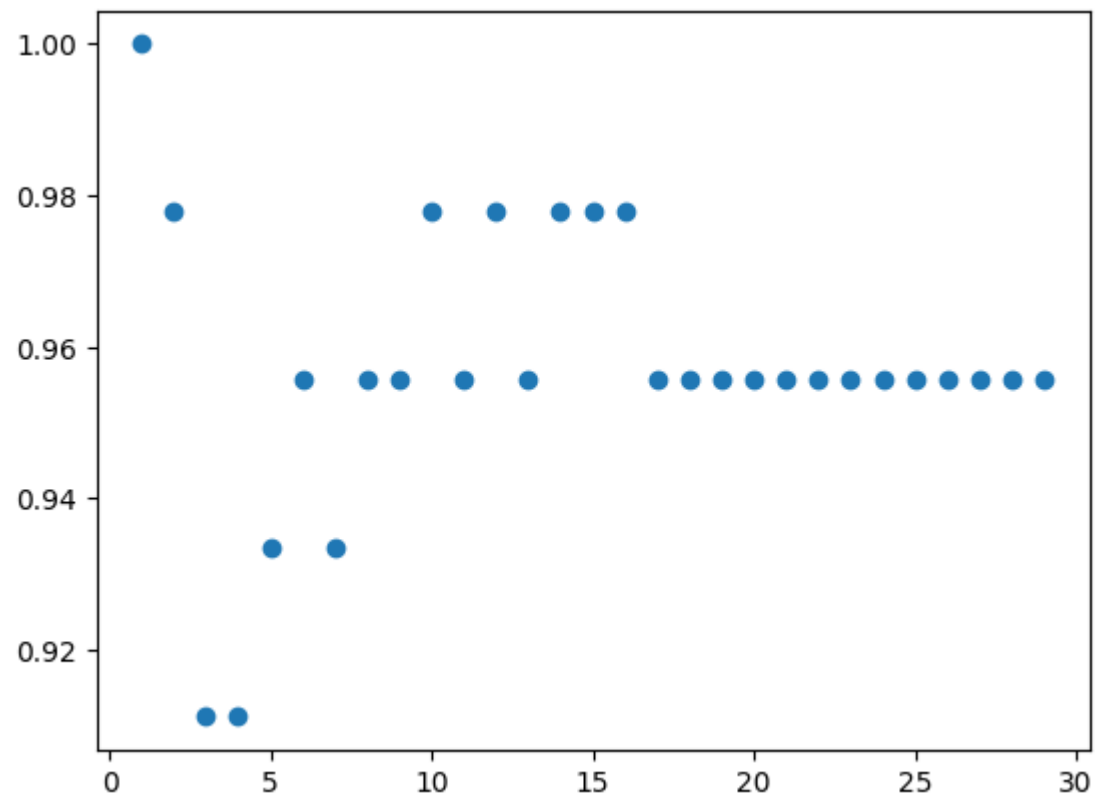
accuracy
```



0.9555555555555556

• K 변경

```
[45] acc_list = []  
     k_range = range(1, 30)  
     for k in k_range :  
         knn = KNeighborsClassifier(n_neighbors = k)  
         knn.fit(X, y)  
  
         y_test_predicted = knn.predict(X_test)  
         acc = metrics.accuracy_score(y_test, y_test_predicted)  
         acc_list.append(acc)  
         print('k = ', k, 'accuracy = ', acc)  
  
     plt.scatter(k_range, acc_list)
```



- `knn.fit()`을 통해서 k-NN 분류기 모델을 얻었으며 이 분류기 모델이 전혀 다른 적이 없는 새로운 데이터를 가지고 예측

```
[51] k = 5
     knn = KNeighborsClassifier(n_neighbors = k)
     knn.fit(X, y)
```



```
▼ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()
```



```
classes = { 0: 'setosa', 1: 'versicolor', 2: 'virginica' }
```

```
new_flowers = np.array(
    [[6.0, 3.0, 1.3, 0.4],
     [4.0, 3.0, 3.2, 1.2]]
)
```

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
pred = knn.predict(new_flowers)
```

```
for label in pred:
    print(classes[label])
```

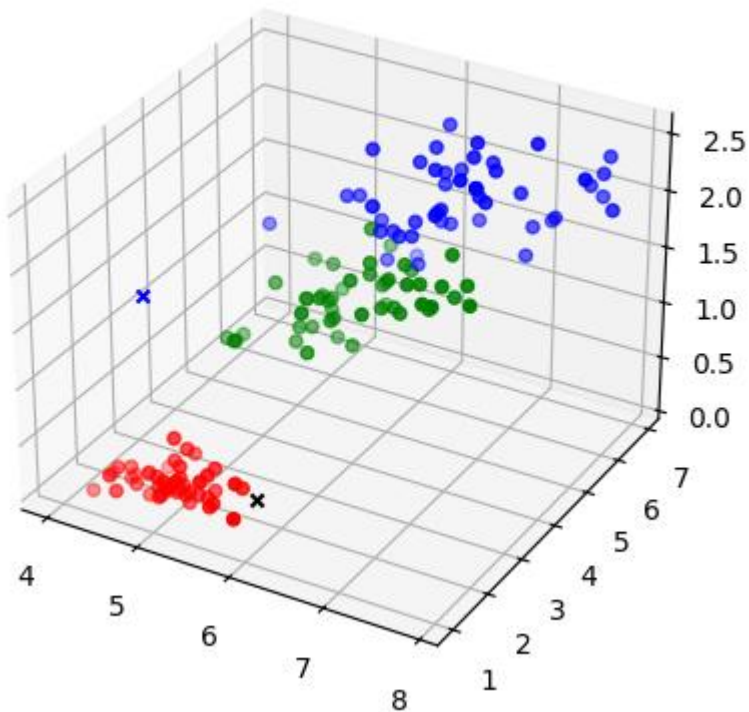


```
setosa
versicolor
```

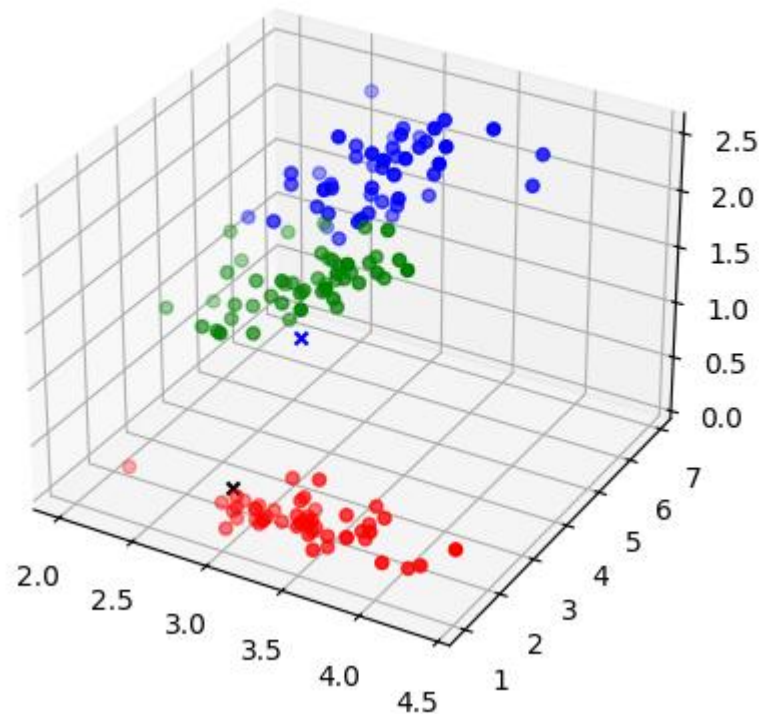
• 시각화

```
▶ x_axis, y_axis, z_axis = 0, 2, 3  
ax = draw_iris_data(x_axis, y_axis, z_axis)  
  
ax.scatter(new_flowers[0, x_axis], new_flowers[0, y_axis], new_flowers[0, z_axis], c='black', marker='x')  
ax.scatter(new_flowers[1, x_axis], new_flowers[1, y_axis], new_flowers[1, z_axis], c='blue', marker='x')
```

iris data



iris data



- 성능평가 도구 - 정확도

```
[100] y_pred = knn.predict(X)  
      acc = metrics.accuracy_score(y_pred, y)  
      acc
```

→ 0.9733333333333334

- 2.67%의 잘못된 분류가 있음을 확인
- 혼동행렬로 확인
- 사이킷런 모듈에서는 이러한 시각화 결과를 **혼동 행렬** `confusion matrix`이라는 이름으로 아래와 같이 구할 수 있음

```
▶ from sklearn.metrics import confusion_matrix
```

```
conf_mat = confusion_matrix(y_pred, y)  
conf_mat
```

```
⇒ array([[50,  0,  0],  
         [ 0, 47,  1],  
         [ 0,  3, 49]])
```

5.7 표집 편향과 성능 측정을 위한 평가지표

- 실제 데이터를 다룰 경우 99%의 정확도를 가지는 분류기는 매우 구현하기 어렵다
- 단 두 줄 만으로 만든 아래의 분류기는 매우 신뢰하기 힘들
- 그런데 경우에 따라 이런 분류기가 99%의 정확도를 가질 수 있다?



```
def classifier_A(length, height): # 입력값에 관계없이 'Samoyed'를 반환  
    return 'Samoyed'
```

- 분류기 A가 입력으로 9,900 마리의 사모예드와 100 마리의 닥스훈트 종의 길이와 높이를 받는다고 가정
- 분류기 A의 정확도는 다음과 같다.

$$\begin{aligned}\text{분류기 A의 정확도} &= \text{제대로 분류한 데이터} / \text{전체 데이터} \\ &= 9,900 / 10,000 \\ &= 0.99 (= 99 \%)\end{aligned}$$

- **표본**sample 데이터가 편향된 경우 분류기의 성능이 제대로 판정되지 않은
 - 학습도 어려워 짐
 - 우수한 머신 러닝 알고리즘을 사용한다고 할지라도 학습 성능의 개선을 기대하기는 힘들.
 - **표집편향**sampling bias이라고 함

- 정밀도^{precision}와 재현율^{recall}을 살펴보고 이 성능을 행렬형태로 표시한 혼동행렬^{confusion matrix} 혹은 오차 행렬에 대해서 알아보도록 하자.
- 100명의 환자와 100 명의 건강한 사람에 대하여 이 검사 키트의 성능을 테스트한다고 가정
 - 검사 키트가 COVID-19 환자(양성^{positive:P})에 대해서 5명을 음성
 - COVID-19에 감염되지 않은 건강한 사람(음성^{negative:N})에 대해서 89명을 음성으로, 11명을 양성으로 판정

		KoKIT22의 예측값 (검사결과)					
		음성			양성		
환자의 실제 상태값		N			P		
음성 (COVID 안걸림)	N	89 TN	T 일치	N 예측	11 FP	F 불일치	P 예측
양성 (COVID 걸림)	P	5 FN	F 불일치	N 예측	95 TP	T 일치	P 예측

- 정확도 Acc는 다음 식을 이용해서 구할 수 있음
- 전체 데이터(FP+FN+TP+TN)중에서 제대로 판정한 데이터(TP + TN)의 비율이 바로 이 값

$$Acc = \frac{TP+TN}{FP+FN+TP+TN} = \frac{95+89}{11+5+95+89} = 0.92$$

- 진짜 양성 비율 True Positive Rate: TPR은 재현율 recall이라고도 함
- 양성환자 중에서 이 키트가 올바르게 양성이라고 분류한 환자의 비율로 다음과 같은 식 (TPR 혹은 Rec로 표기함)

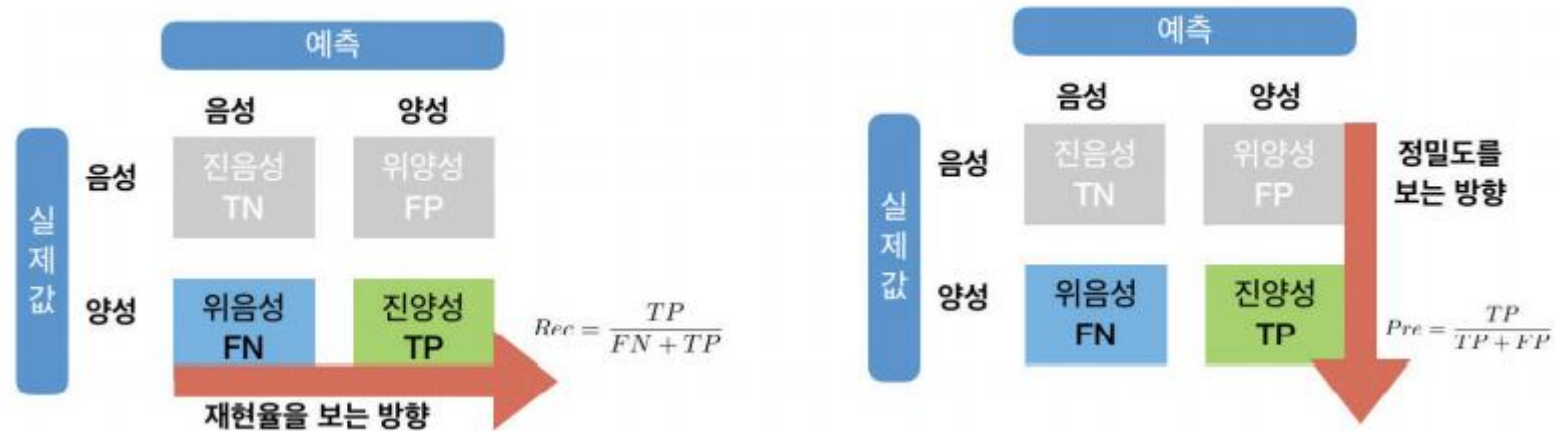
$$TPR = Rec = \frac{TP}{P} = \frac{TP}{FN + TP} = \frac{95}{100} = 0.95$$

- 정밀도^{precision}은 검사 키트가 확진자로 분류한 사람들 중 실제 양성인 경우로 다음과 같은 식(Pre로 표기함)

$$Pre = \frac{TP}{TP+FP} = \frac{95}{106} = 0.896$$

5.8 사이킷 런의 성능지표 함수들

- 재현율 : 실제로 COVID-19에 양성인 대상자를 양성으로 분류할 확률
- 정밀도 : 양성으로 예측한 사람들 중에서 실제 COVID-19 양성자일 확률



- 재현율과 정밀도라는 각각의 지표는 관심있는 척도가 다르기 때문에 하나의 척도만을 측정 방법으로 사용할 경우 왜곡이 발생할 수 있음
- 두 지표를 조합한 F1 점수도 사용 하는데 이 점수는 다음과 같은 식
- F1 점수는 정밀도와 재현율의 **조화 평균**harmonic mean

$$F_1 = \frac{2}{\frac{1}{Pre} + \frac{1}{Rec}} = 2 \frac{Pre \times Rec}{Pre + Rec}$$

F1 점수를 **쇠렌센-다이스 계수**Sørensen-Dice coefficient 혹은 **다이스 유사도 계수**Dice similarity coefficient라고 부르기도 함

- Sklearn이 제공하는 함수

```
▶ from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score

prec = precision_score(y_pred, y, average='micro')
rec = recall_score(y_pred, y, average = 'micro')
f1 = f1_score(y_pred, y, average = 'micro')

print(f'precision: {prec}')
print(f'recall: {rec}')
print(f'f1: {f1}')
```

```
⇒ precision: 0.9733333333333334
recall: 0.9733333333333334
f1: 0.9733333333333334
```

- 다음과 같이 10개의 0 값(음성), 10개의 1 값(양성)으로 이루어진 목표값 target
- 이 목표값에 대한 예측값이 pred 넘파이 배열에 들어있는 경우를 가정
- 목표값과 예측값을 혼동행렬로 표시

```
[92] target = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
      pred = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1]
```

```
[93] from sklearn.metrics import confusion_matrix
      conf_mat = confusion_matrix(pred, target)
      conf_mat
```

```
⇒ array([[8, 3],
        [2, 7]])
```

```
[80] # accuracy
acc = ( conf_mat[0, 0] + conf_mat[1, 1] ) / conf_mat.sum()
acc_sklearn = accuracy_score(pred, target)
acc, acc_sklearn
```

⇒ (np.float64(0.75), 0.75)

```
[81] # recall: 진양성률 : 양성 판정자 중에서 실제 양성인 비율
rec = conf_mat[1, 1] / (conf_mat[1, 0] + conf_mat[1, 1])
rec_sklearn = recall_score(pred, target)
rec, rec_sklearn
```

⇒ (np.float64(0.7777777777777778), 0.7777777777777778)

```
[85] # precision: 정밀도 : 양성자 중에서 양성으로 판정되는 비율
prec = conf_mat[1, 1] / (conf_mat[0, 1] + conf_mat[1, 1])
prec_sklearn = precision_score(pred, target)
prec, prec_sklearn
```

⇒ (np.float64(0.7), 0.7)

```
[86] # f1 score
# 2 * prec*rec / (prec+red)
f1 = 2.0 * prec*rec / (prec+rec)
f1_sklearn = f1_score(pred, target)
f1, f1_sklearn
```

⇒ (np.float64(0.7368421052631577), 0.7368421052631579)



도전문제 5.3

1. 입력 데이터가 10,000개이고, 이 중 9,900개가 사모예드, 100개가 닥스훈트라 하자. 5.7절의 classifier_A로 분류했을 때, 정밀도, 재현율, 정확도, F_1 점수를 계산하라.

정밀도 : 0.99

재현율 : 1.0

정확도 : 0.99

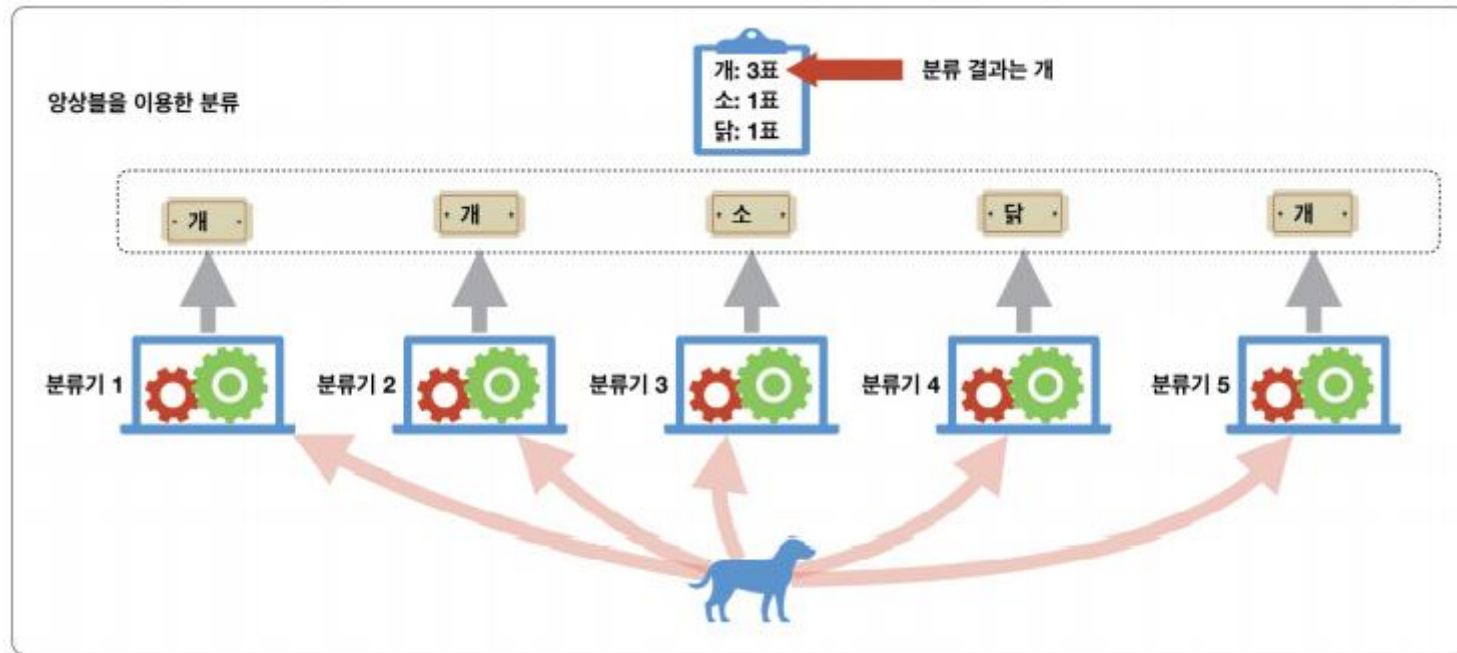
F1 점수 : 0.9949748743718593

2. 표집 편향(sampling bias)의 사례를 수집하고 이로 인한 잘못된 예측 결과를 기술해 보라.

5.9 앙상블 - 머신러닝의 민주주의

- 어느 수준의 성능에 도달하면 분류기를 개선하는 일이 매우 어려워짐
- 모델의 특성에 따라 일정 수준 이상의 성능을 넘어서는 것도 어려움
- 이를 해결하기 위한 앙상블^{Ensemble} 기법은 여러 가지 종류의 분류기를 개별적으로 개선하는 데에 한계가 있을 때 이들의 협력을 이용하는 방법

- 앙상블은 머신러닝 분야의 집단 지성
- 여러 분류기가 각자의 분류 결과를 투표하듯이 내어 놓으면 가장 많은 표를 얻은 결과를 최종 결과로 선택하는 것



- 분류기의 성격이 모두 동일해 오분류도 비슷한 상황에서 일어난다면 모두가 한꺼번에 잘못된 투표를 할 확률이 높아짐
 - 모델의 성격이 서로 다른 것이 좋다.
 - 개별 분류기의 **다양성**diversity이라고 한다.
- 분류기의 다양성을 확보하는 방법
 1. 서로 다른 모델로 각각의 분류기를 만들기
 2. 각각의 분류기에 대해 서로 다른 학습 데이터를 제공해 훈련
- 분류기에 선택된 데이터가 다른 분류기의 학습에도 사용될 수 있으면 **배깅**bagging 기법
 - 불가능하면 **페이스팅**pasting이라고 부름
 - 확률 통계 분야에서 이러한 추출을 각각 복원 추출, 비복원 추출
 - 배깅 기법을 개선한 방법이 **부스팅**boosting 기법

- 부스팅 알고리즘은 AdaBoost 알고리즘이 가장 일반적으로 사용
- AdaBoost 학습
 - 각 데이터 인스턴스는 동일한 가중치로 초기화
 - 예측기를 순차적으로 학습
 - 한 예측기가 잘못 분류한 데이터 인스턴스의 가중치를 높임
 - 다음 예측기는 높은 가중치의 데이터 인스턴스에 적합하게 학습

5.10 군집화의 개념과 비지도 학습

- **군집화**clustering란 소속집단의 정보가 없고 모르는 상태에서 비슷한 집단으로 묶는 비지도 학습
- 입력 데이터를 바탕으로 출력값을 예측하는 목적으로 사용되기 보다는 데이터에서 의미를 파악하고 기준을 만드는 목적으로 사용

	분류	군집화
소속집단에 대한 정보	있다	없다
레이블 유무	있다	없다
종류	지도 학습	비지도 학습
공통점	데이터를 비슷한 집단으로 묶는 방법	

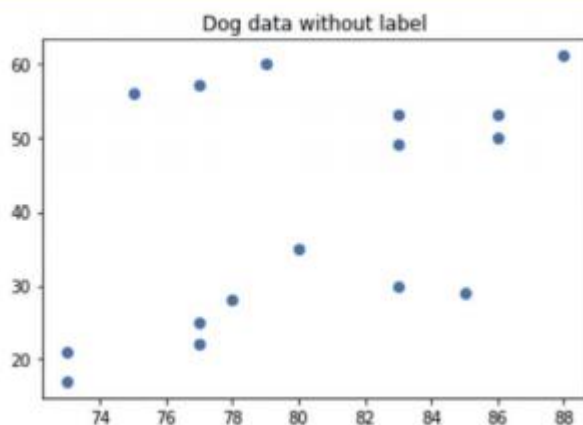


```
import numpy as np
import matplotlib.pyplot as plt

..
# 사모예드와 닥스훈트의 길이, 높이 데이터는 중복되어 생략한다
# 개의 길이와 높이를 각각 ndarray 형태로 만든다
dog_length = np.array(dach_length + samo_length) # 리스트를 이어 ndarray로
dog_height = np.array(dach_height + samo_height) # 리스트를 이어 ndarray로

dog_data = np.column_stack((dog_length, dog_height))

plt.title("Dog data without label")
plt.scatter(dog_length, dog_height)
```



이전 절의 데이터와는 달리
어느 데이터가 닥스훈트인지
사모예드인지 사전 정보가
없어요.

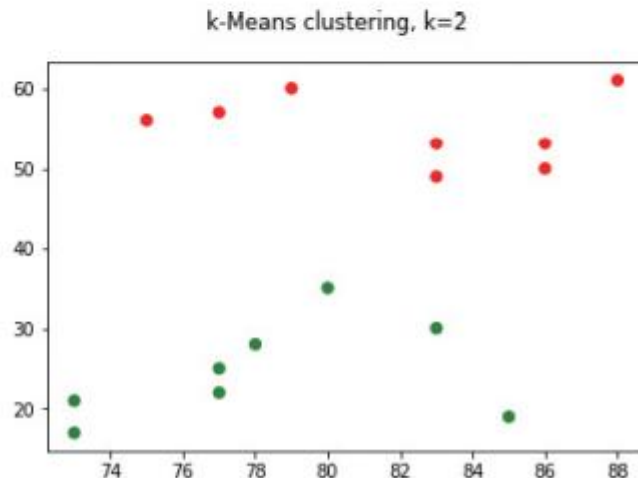
- 데이터를 두 개의 그룹으로 나누는 k-평균 알고리즘을 적용
 - k-평균 k-means 알고리즘은 sklearn에서 제공하는 cluster 모듈에 존재



```
from sklearn import cluster
```

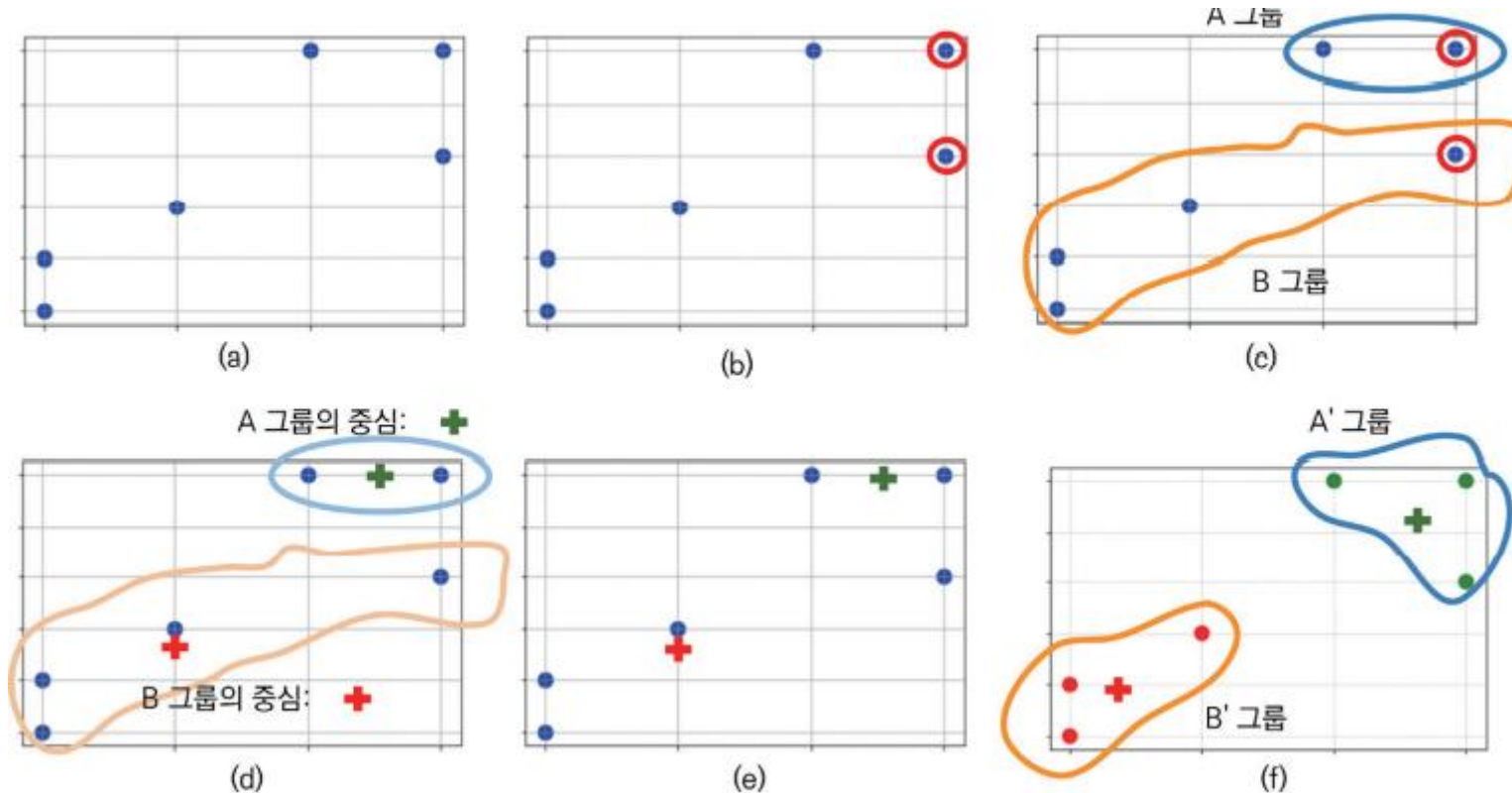
```
def kmeans_predict_plot(X, k):  
    model = cluster.KMeans(n_clusters=k)  
    model.fit(X)  
    labels = model.predict(X)  
    colors = np.array(['red', 'green', 'blue', 'magenta'])  
    plt.suptitle('k-Means clustering, k={}'.format(k))  
    plt.scatter(X[:, 0], X[:, 1], color=colors[labels])
```

```
kmeans_predict_plot(dog_data, k = 2)
```



5.11 k-평균 알고리즘 살펴보기

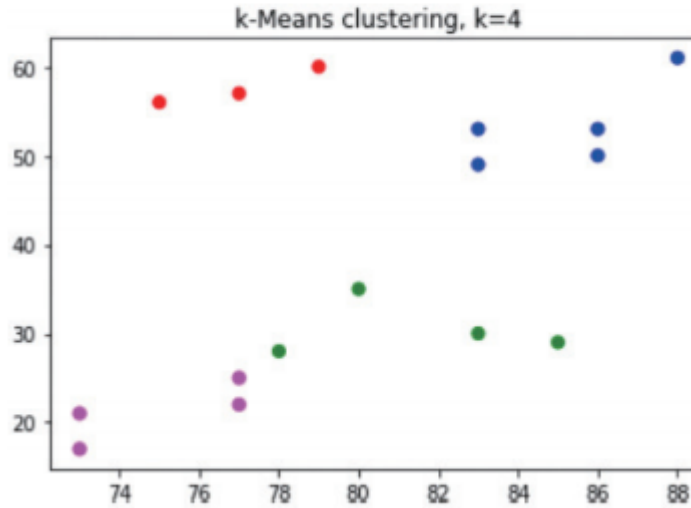
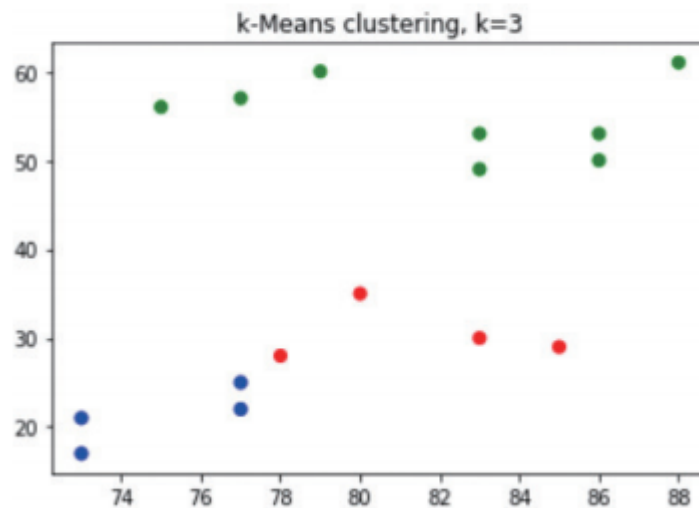
- k-평균 알고리즘은 원리가 단순하고 직관적이며 성능이 좋은 군집화 알고리즘
 - 사전에 군집의 개수 k 값을 지정해야 하는 단점.



- k-평균 알고리즘은 데이터 분포에 대한 사전 지식이 없을 경우에도 사용가능
- 군집에 대한 사전 정보가 있을 경우 이를 바탕으로 모범 샘플을 사용하여 그 샘플을 초기화 에 사용할 경우 더 나은 성능을 보일 수 있음

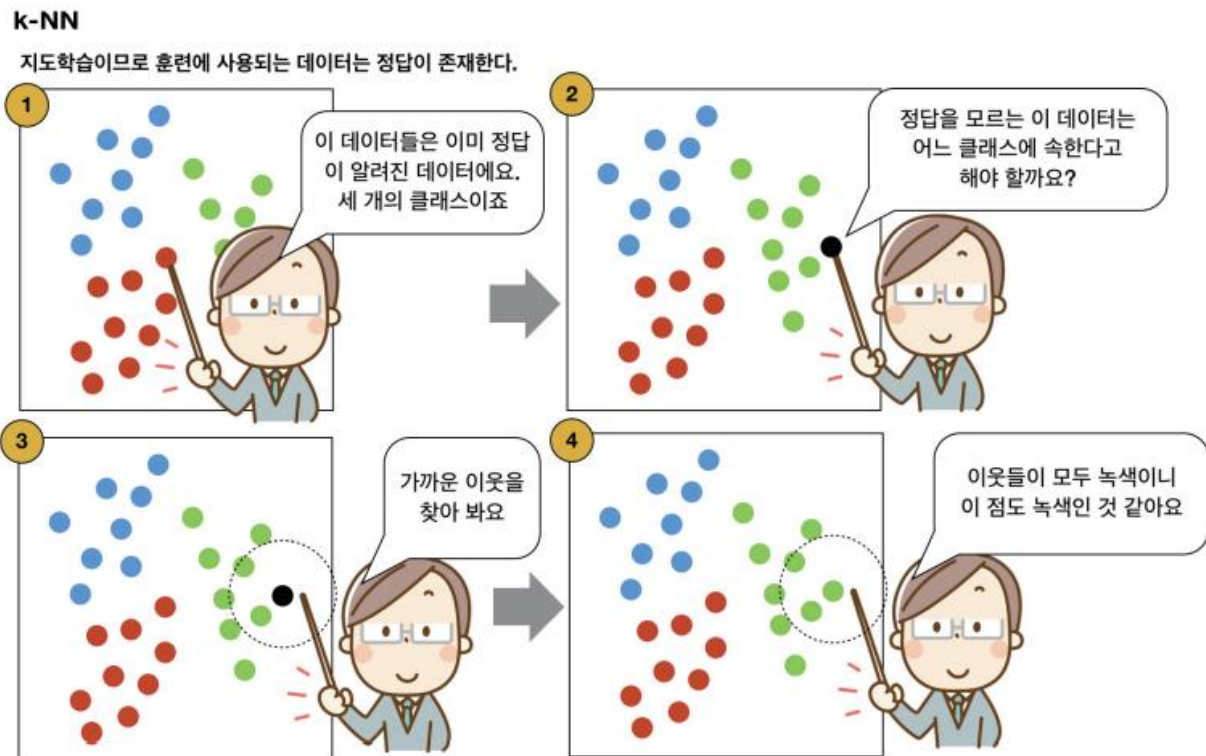


```
kmeans_predict_plot(dog_data, k = 3) # 3개의 군집 생성  
kmeans_predict_plot(dog_data, k = 4) # 4개의 군집 생성
```

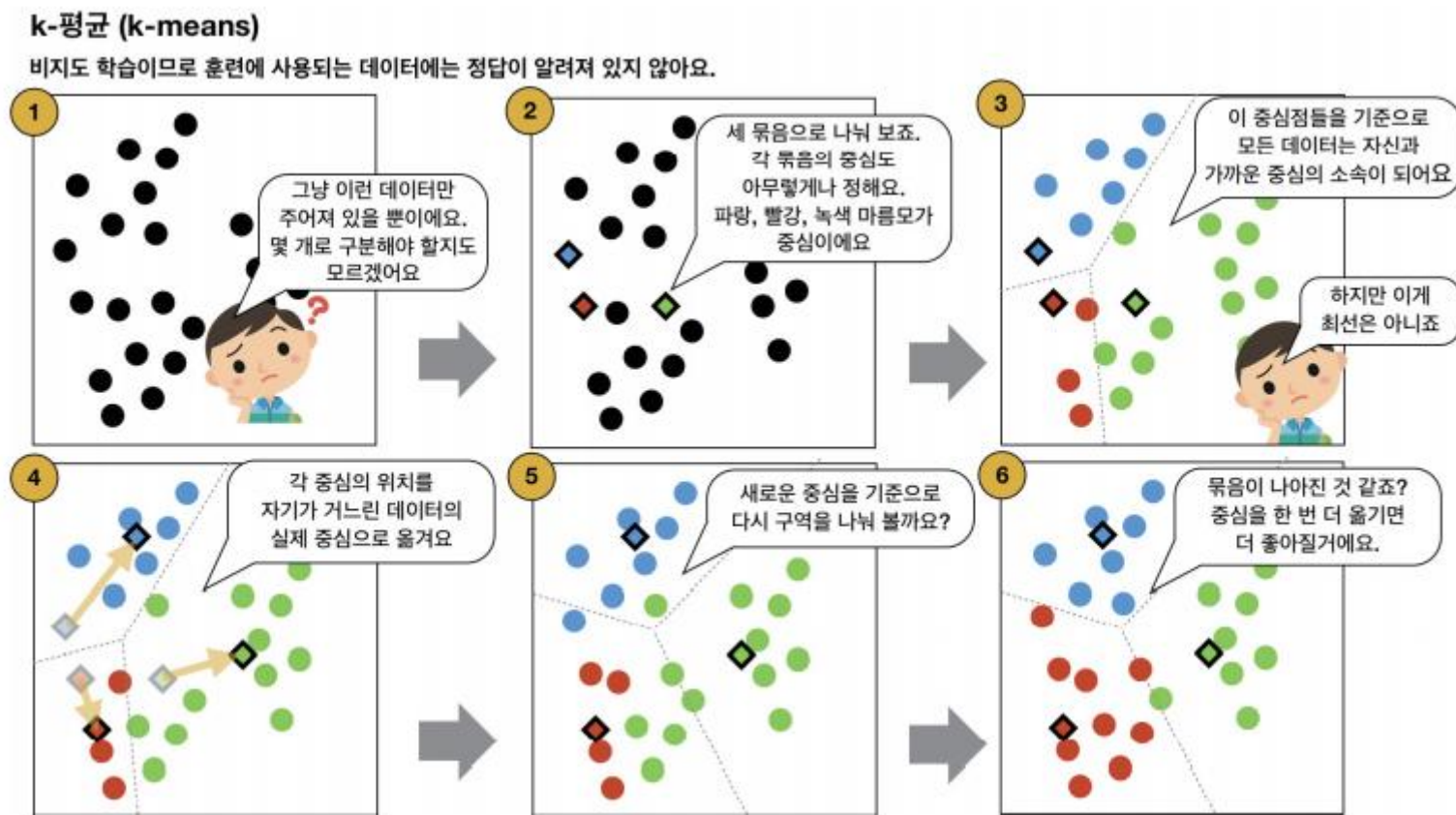


5.12 k-평균 알고리즘과 k-NN 알고리즘의 비교: 지도 학습과 비지도 학습

- k-평균 알고리즘은 **비지도 학습** unsupervised learning으로 데이터에 부여된 정답이 없음
- k-NN과 비교를 통해 비지도 학습의 동작방식을 이해해 보자.
- 가장 가까운 이웃 k개를 찾음
- 이웃들이 어떤 클래스에 속하는지를 보고, 새로운 데이터의 클래스를 4번 그림과 같이 결정하면 됨



- k-평균 알고리즘은 k-NN과 달리 정답 레이블이 없는 문제
- 데이터를 몇 개의 묶음, 혹은 클러스터로 나눌 것인지를 결정 해야 하며 이것이 바로 k-평균 알고리즘의 k가 됨



- 데이터 군집화 예시
- 데이터 준비

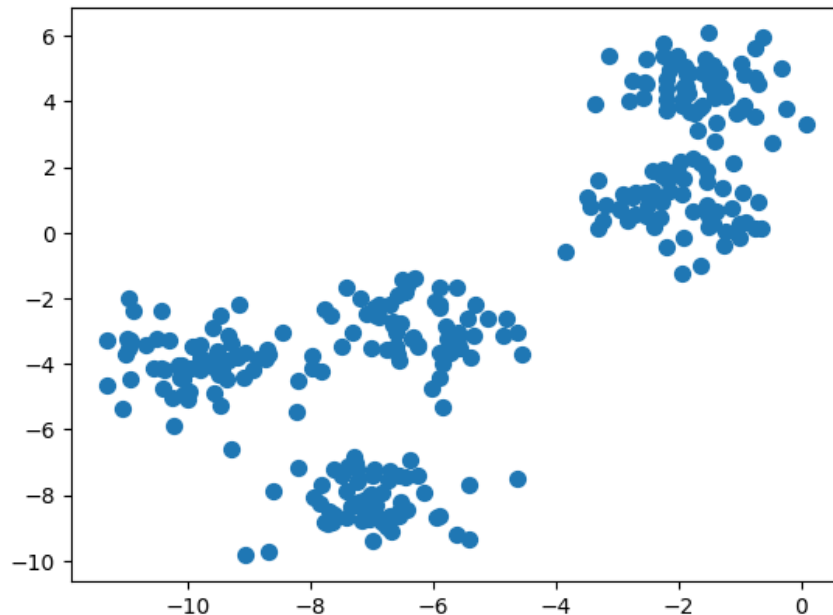
```
[102] from sklearn.datasets import make_blobs
```

```
# Generate random data
```

```
X, _ = make_blobs(n_samples=300, centers=5, cluster_std=0.8, random_state=1)  
X.shape
```

```
⇒ (300, 2)
```

```
[▶] plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')
```



- 학습: 비지도 학습

```
[105] from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5)

### 비지도 학습 실시
## 지도학습의 구조 Model.fit( X:입력 , y:정답레이블)
## 비지도학습의 구조 Model.fit ( X )

kmeans.fit(X)
```



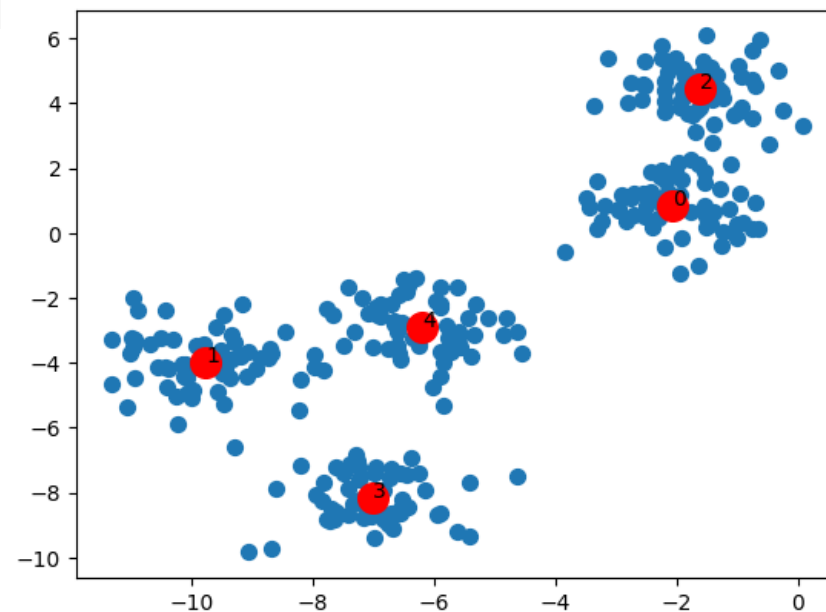
▼ KMeans ⓘ ?
KMeans(n_clusters=5)

• 학습 결과

```
[112] kmeans.cluster_centers_
```

```
array([[ -2.07108619,  0.84455573],  
       [-9.76939793, -3.97511428],  
       [-1.62791547,  4.44598374],  
       [-7.02207108, -8.16109089],  
       [-6.2033337 , -2.89681989]])
```

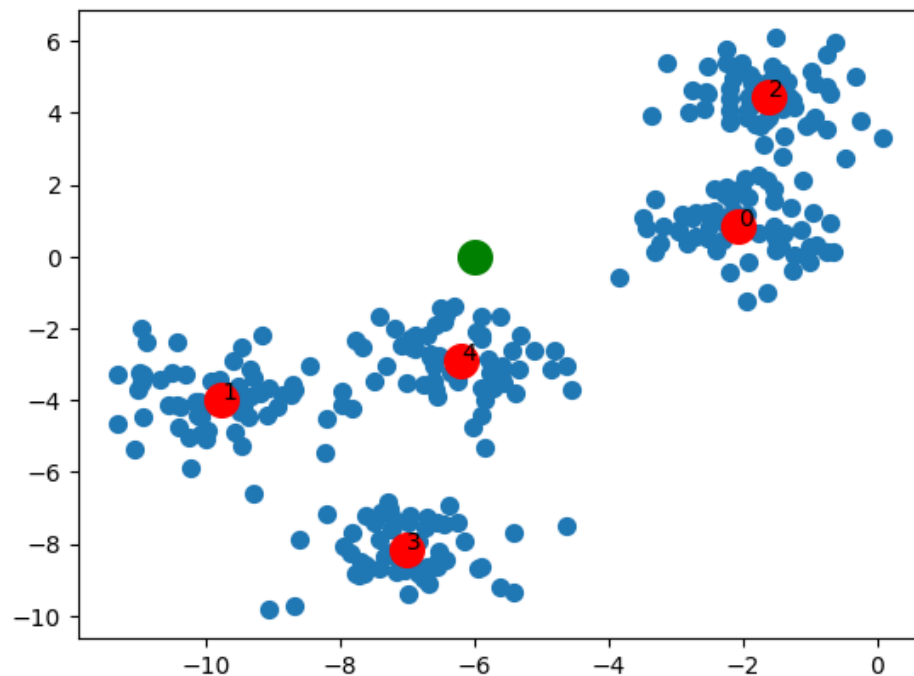
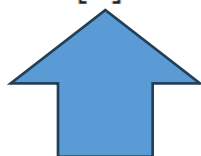
```
plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')  
plt.scatter(kmeans.cluster_centers_[ :, 0], kmeans.cluster_centers_[ :, 1], s=200, c='red', label='Centroids')  
for i, center in enumerate(kmeans.cluster_centers_):  
    plt.annotate(i, center, fontsize=10)
```



• 예측 능력

```
▶ new_point = [-6, 0]
cluster = kmeans.predict([new_point])
print(f'new point {new_point} belongs to cluster {cluster}')
plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1], s=200, c='red')
plt.scatter(new_point[0], new_point[1], s=200, c='green')
for i, center in enumerate(kmeans.cluster_centers_):
    plt.annotate(i, center, fontsize=10)
```

⇒ new point [-6, 0] belongs to cluster [4]



핵심 정리

- **분류**는 소속 집단의 정보를 이미 알고 있는 상태에서 비슷한 집단으로 묶는 방법이며, **군집화**란 소속 집단의 정보가 없는 상태에서 비슷한 집단으로 묶는 방법이다.
- **k-NN 알고리즘**은 k-최근접 이웃(k-Nearest Neighbor)의 약자로 특징 공간에 분포하는 데이터에 대하여 k개의 가장 가까운 이웃을 살펴보고 **다수결 방식**으로 데이터의 레이블을 할당하는 분류방식이다.
- 데이터를 분류하는 알고리즘의 대표적인 예인 k-NN 알고리즘은 매우 단순하고 직관적이며, 사전 학습이 나 특별한 준비 시간이 필요없다는 점은 장점이다.
- **샘플 데이터가 편향**된 경우 우수한 머신러닝 알고리즘을 사용한다고 할지라도 학습 성능의 개선을 기대하기는 힘들다.
- **정확도**는 전체 데이터 중에서 예측값과 정답값이 일치하는 데이터들의 비율이다.
- **진짜 양성 비율**은 **재현율**이라고도 하는데 양성 데이터 중에서 이 키트가 양성이라고 제대로 분류한 데이터의 비율이다.
- **정밀도**는 양성으로 분류된 것들 중에서 실제 양성인 데이터의 비율이며, **F1 점수**는 정밀도와 재현율의 조화 평균값이다.

핵심 정리

- 정밀도와 재현율은 트레이드오프가 존재한다.
- 앙상블 기법은 다수의 분류기가 협력하여 전체적인 분류 성능을 높이는 기법이다.
- 앙상블을 구성하는 분류기는 동질성을 갖는 것보다 다양성을 갖는 것이 좋다.
- 분류기의 다양성을 확보하기 위해 서로 다른 학습 데이터를 제공하는 배깅과 페이스팅이 사용된다.
- 배깅과 페이스팅의 구분은 데이터를 복원 추출하는냐 비복원 추출하는냐의 차이이다.
- 부스팅은 분류기를 순차적으로 학습시키면서 이전 분류기가 제대로 처리하지 못 한 데이터에 더 집중하는 방법이다.
- 군집화란 소속집단의 정보가 없고 모르는 상태에서 비슷한 집단으로 묶는 비지도 학습의 하나이다.
- k-평균 알고리즘은 군집의 개수에 따라 데이터를 군집으로 분류하는 알고리즘으로 원리가 단순하고 직관적이며 성능이 좋은 군집화 알고리즘으로 사전에 군집의 개수 k 값을 지정해야 한다.