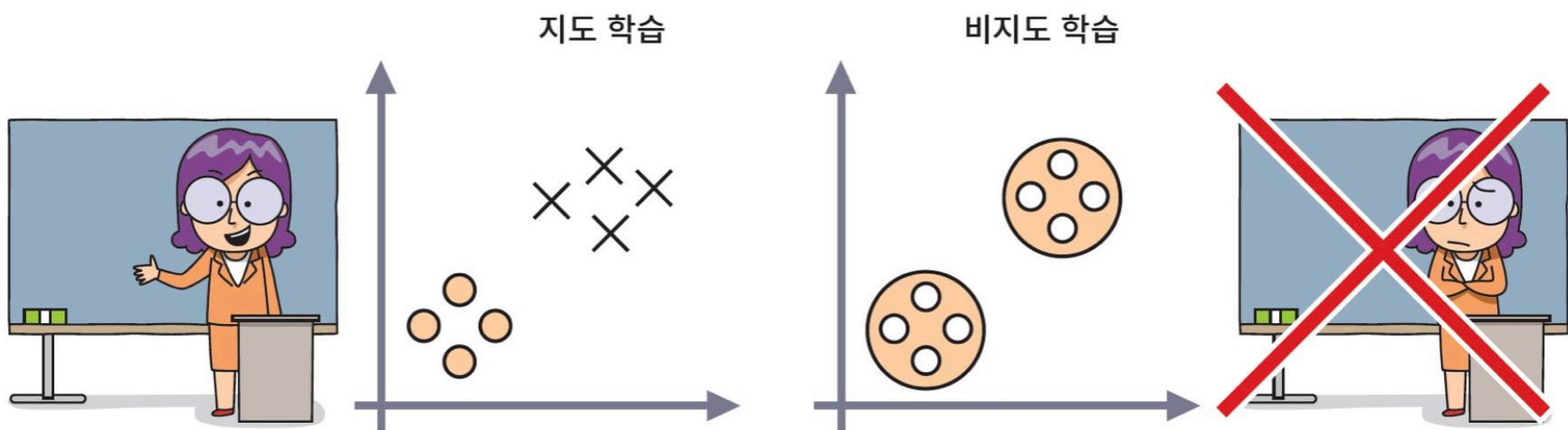


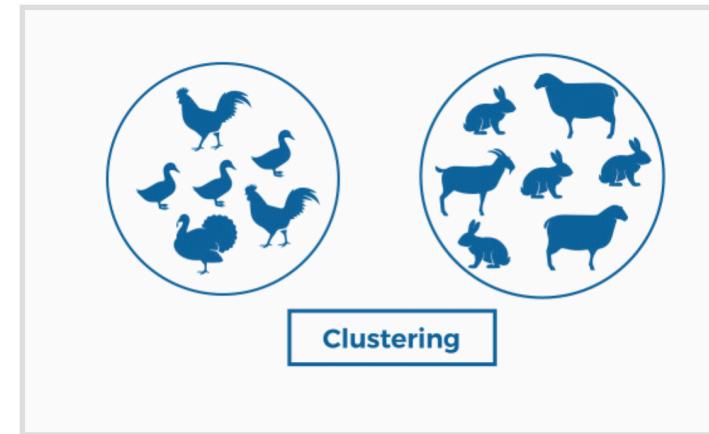
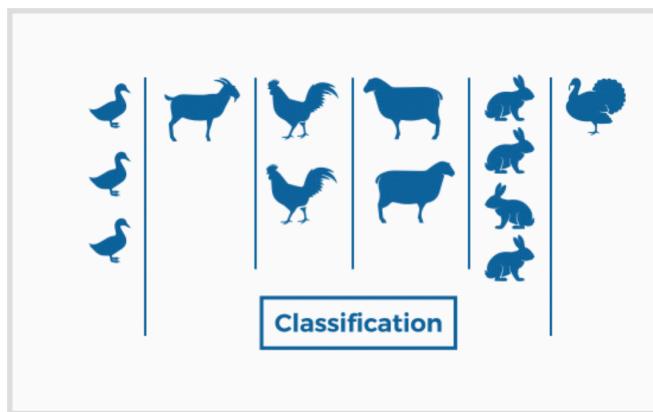
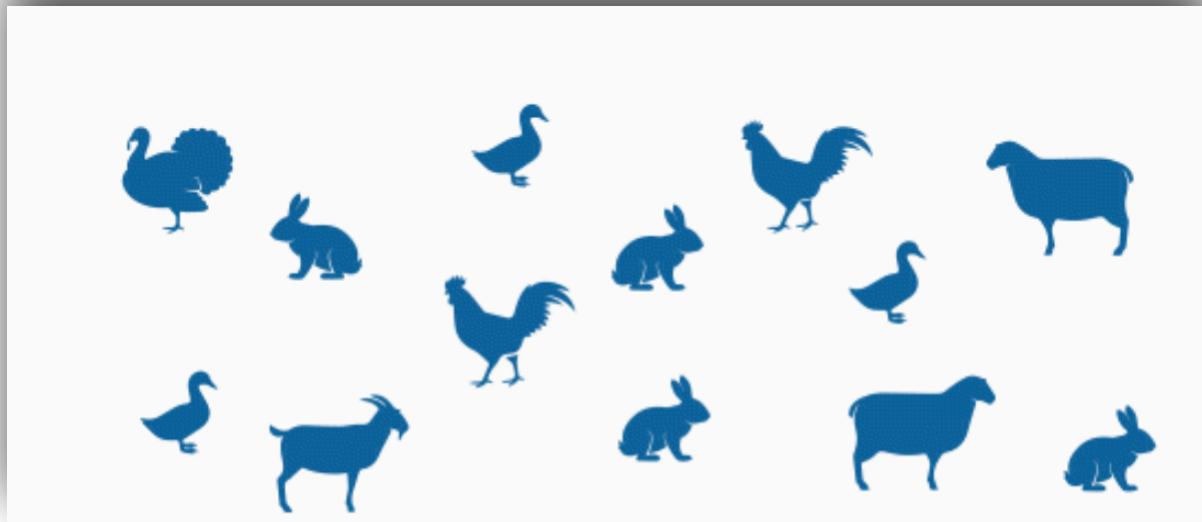
k-Nearest Neighbor & k-Means

내용

- 분류와 클러스터링
- k -NN 알고리즘
- k -means 알고리즘
- k -NN 알고리즘의 구현
- k -means 알고리즘의 구현

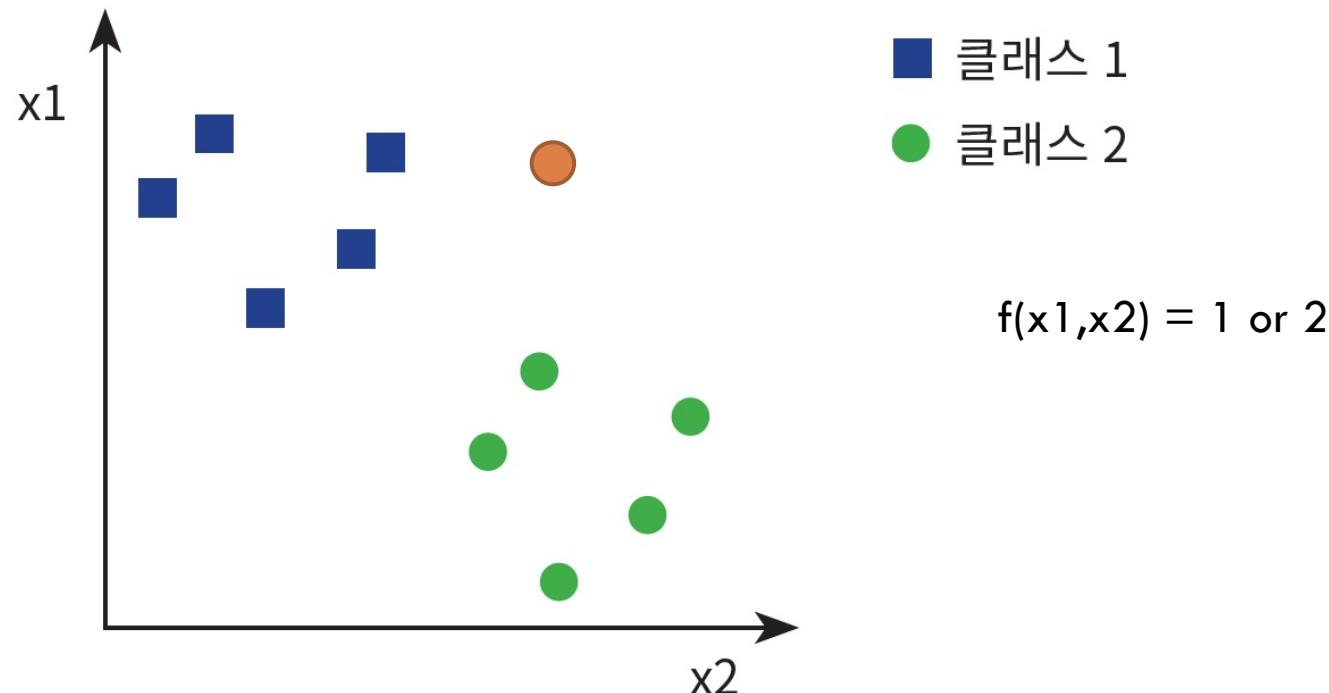
분류와 클러스터링





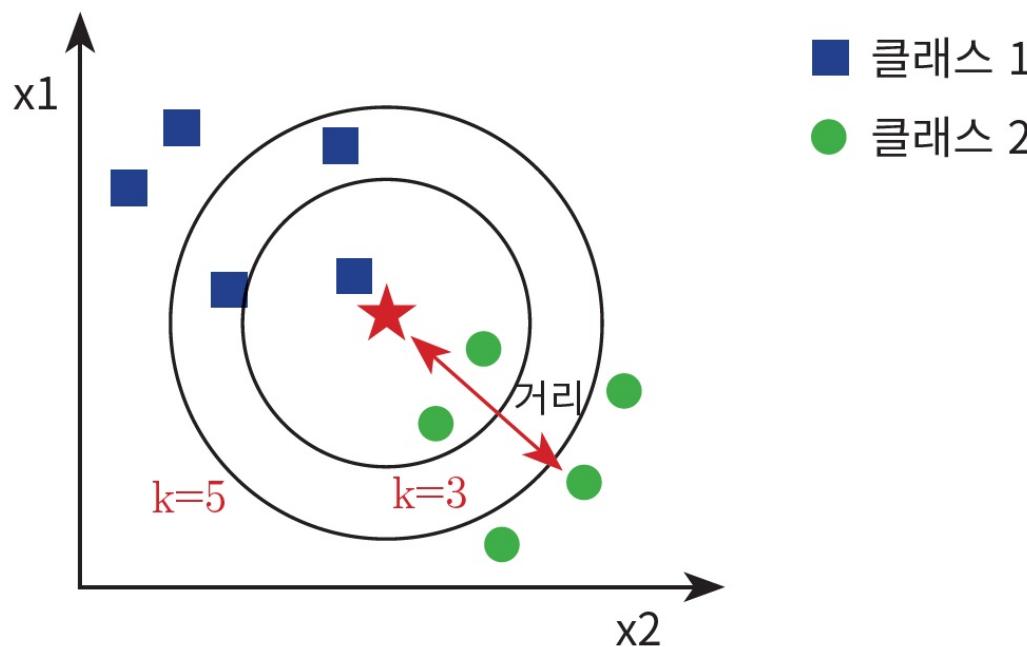
k-NN 알고리즘

- k-Nearest Neighbor(k-NN) 알고리즘은 간단하고 이해하기 쉬운 분류 알고리즘



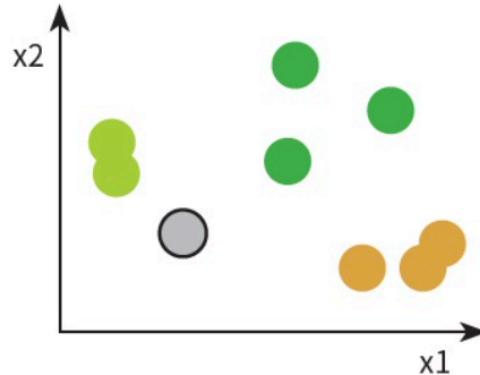
k -NN 알고리즘

- 새로운 데이터는 두 클래스의 데이터와 가까운 거리에 위치한 클래스 정보를 따름



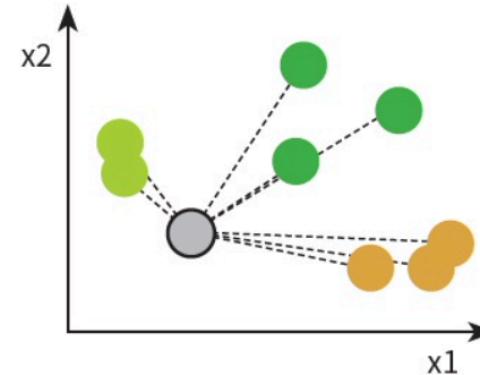
k -NN 알고리즘

1. 데이터를 관찰한다.



회색 원은 어디에 속해야 할까?

2. 거리를 계산한다.



회색 원과 다른 원들간의 거리를 계산한다.

3. 이웃을 찾는다.

점	거리	
●	2.1	→ 1등
●	2.4	→ 2등
●	3.1	→ 3등
●	4.5	→ 4등

거리에 따라서 이웃 원들을 정렬한다.

4. 새로운 데이터에 대하여 투표한다.

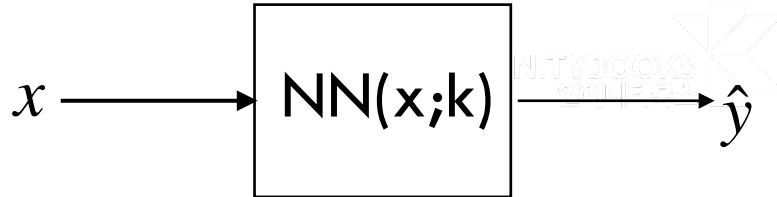
클래스 투표수

●	2
●	1
●	1

●색 원이 가장 많았으므로 새로운 원은 ●에 속한다.

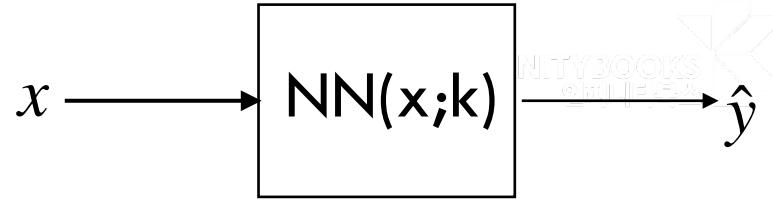
가장 가까운 k 개의 이웃 중에서 가장 많은 표를 얻은 클래스로 분류한다.

다중 분류 문제 평가



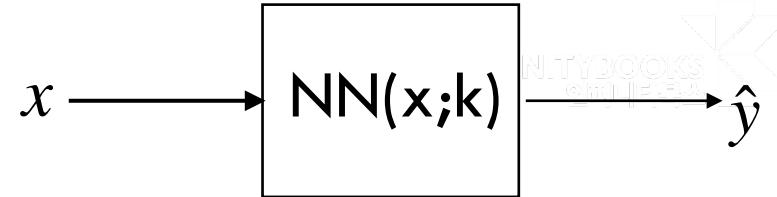
- 데이터셋 $D = \{(x_i, y_i) \mid i = 1, \dots, N\}$
 - 입력값 $x_i \in \mathbb{R}^d, d \geq 2$
 - 실제값 $y_i \in \{1, 2, \dots, C\}$
- 학습된 모델 $\hat{y} = \text{NN}(x; k)$
 - 입력값 $x \in \mathbb{R}^d$
 - 출력값 $\hat{y} \in \{1, 2, \dots, C\}$
- 데이터 (x_i, y_i) 의 평가
 - 모델 $\hat{y}_i = \text{NN}(x_i; k)$
 - 만약 $\hat{y}_i \neq y_i$ 이면 오류 발생

다중 분류 문제 평가

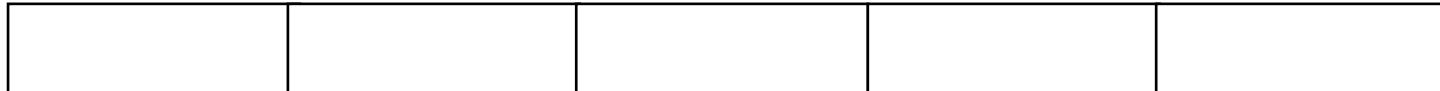


- 객관적 평가를 여러 경우에 모델을 평가
- 데이터셋을 훈련과 테스트 데이터셋으로 분리 $D = D_{train} \cup D_{test}$
 - 훈련 데이터셋 $D_{train}, N_1 = |D_{train}|$
 - 테스트 데이터셋 $D_{test}, N_2 = |D_{test}|$
 - 나누는 비율은 8:2가 보편적으로 선택. 그러나 데이터셋 크기에 따라 다름
 - 임의 선택을 이용하여 데이터셋을 나눔
- 주어진 데이터셋에서 반복적으로 평가를 진행
 - 훈련과 테스트 데이터셋에 대해 동일하게 평가
 - k-way 교차 평가: 훈련과 테스트 데이터셋으로 나누어 k번 평가 진행
 - k-way 검증 교차 평가: 훈련, 검증 과 테스트 데이터셋으로 나누어 k번 평가 진행

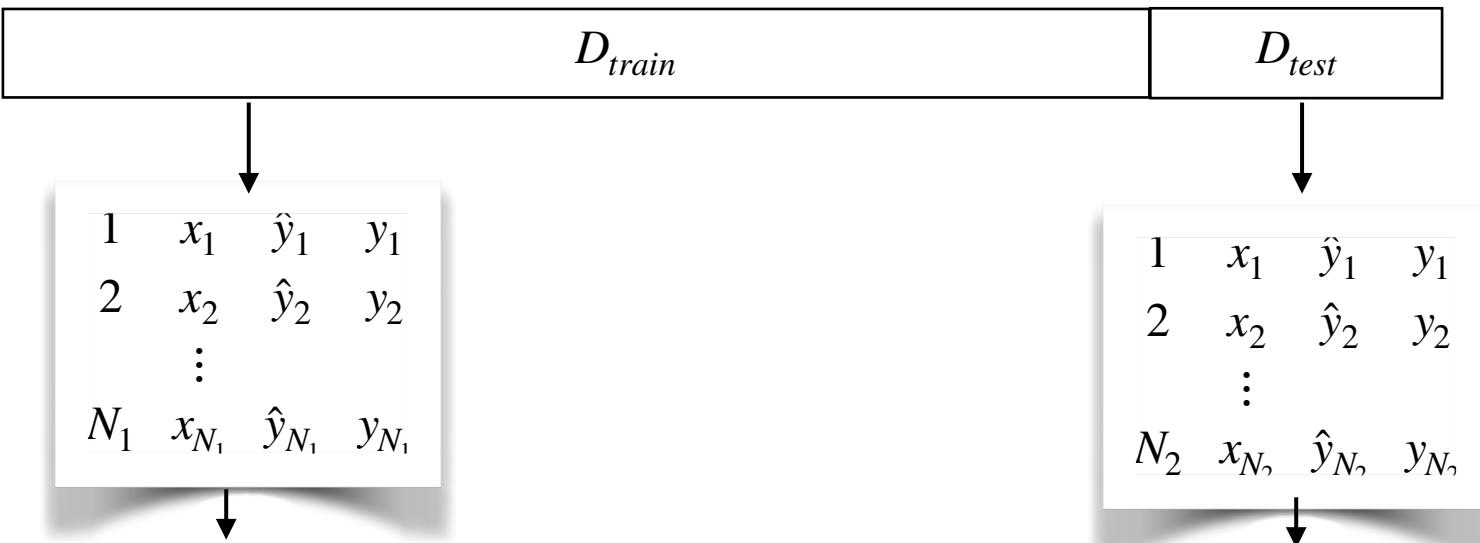
다중 분류 문제 평가



$$D = \{(x_i, y_i) \mid i = 1, \dots, N\}$$



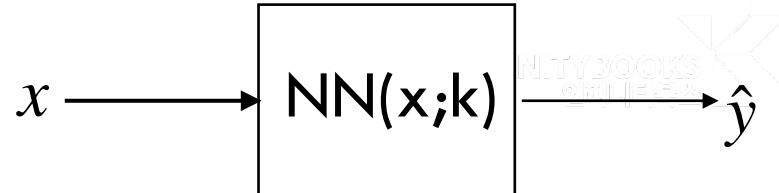
$$D = D_{train} \cup D_{test}$$



$$\text{Accuracy}_{train} = \frac{1}{N_1} \sum_{i=1}^{N_1} \mathbb{1}(y_i = \hat{y}_i)$$

$$\text{Accuracy}_{test} = \frac{1}{N_2} \sum_{i=1}^{N_2} \mathbb{1}(y_i = \hat{y}_i)$$

5-way 다중 분류 문제 평가



- Accuracy와 error의 관계

$$\text{Accuray} = \frac{1}{N} \sum_{i=1}^N \mathbb{1}(y_i = \hat{y}_i) \quad \text{Error} = 1 - \text{Accuray}$$

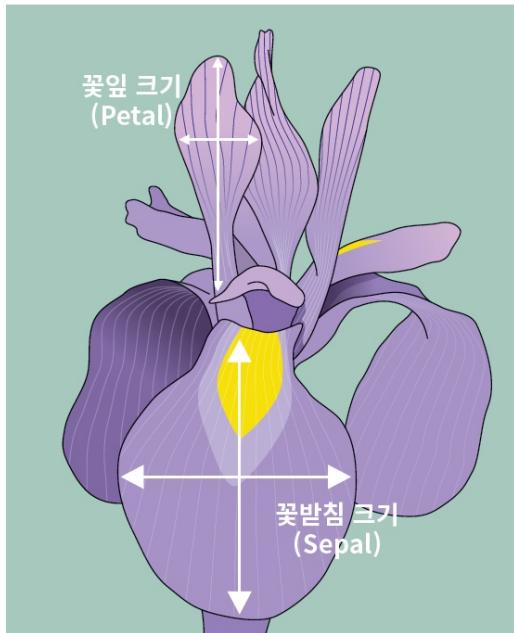
- 테스트셋 선택이 가능한 모든 조합에 대해 평가
- 모델 최종 평가는 모든 평가의 평균임
- 5-way 교차 검증의 경우 훈련 데이터셋에서 검증 데이터셋을 일부 선택
 - 훈련 모델은 검증 데이터셋을 제외한 데이터로 구성
 - 훈련 모델을 검증 데이터셋으로 평가 후 가장 성능이 높은 훈련 모델을 선택하여 테스트셋을 평가

k -NN 알고리즘의 특징

- 모든 데이터에 대한 정보가 필요
- 왜냐하면, 가장 가까운 이웃을 찾기 위해 새로운 데이터에서 모든 기존 데이터까지의 거리를 확인해야 하기 때문
- 데이터와 클래스가 많이 있다면, 많은 메모리 공간과 계산 시간이 필요
- 학습이나 준비 시간이 필요 없음

사례

- Iris 데이터 세트는 3종의 각 50개의 아이리스 꽃 샘플에 정보를 갖음



(아이리스 꽃: 꽃받침 길이와 너비, 꽃잎 길이와 너비)

setosa
versicolor
Virginia

5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor

	sepal length	sepal width	petal length	petal width	class
0	4.8	3.4	1.6	0.2	0
1	5.7	2.5	5.0	2.0	2
2	6.3	2.7	4.9	1.8	2
.
.
.
148					
149					

사례

```
from sklearn.datasets import load_iris  
  
iris = load_iris() # 데이터셋 로드  
print(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],  
       ...
```

사례

```
# 4개의 특징 이름을 출력한다.  
print(iris.feature_names)
```

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

```
# 정수는 꽃의 종류를 나타낸다.: 0 = setosa, 1=versicolor, 2=virginica  
print(iris.target)
```

사례

```
from sklearn.model_selection import train_test_split

X = iris.data
y = iris.target

# (80:20)으로 분할한다.
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)
```

(120, 4)
(30, 4)

사례

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

knn = KNeighborsClassifier(n_neighbors=6) # 학습 단계
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test) # 예측 단계
scores = metrics.accuracy_score(y_test, y_pred)
```

0.9666666666666667

사례

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)

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

# 새로운 데이터에 대한 예측
x_new = [[3,4,5,2],
          [5,4,2,2]]
y_predict = knn.predict(x_new)

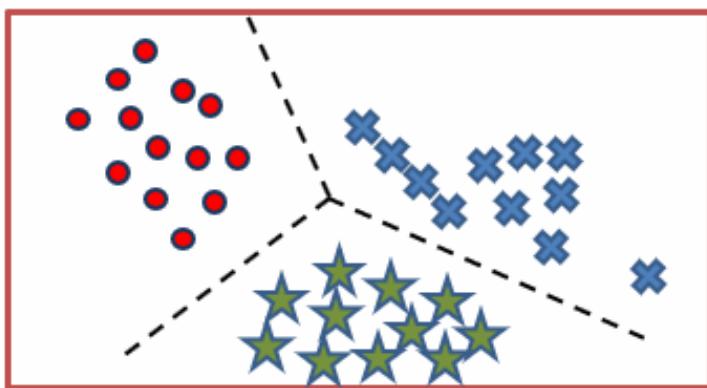
print(classes[y_predict[0]])
print(classes[y_predict[1]])
```

versicolor
setosa

k-means 클러스터링

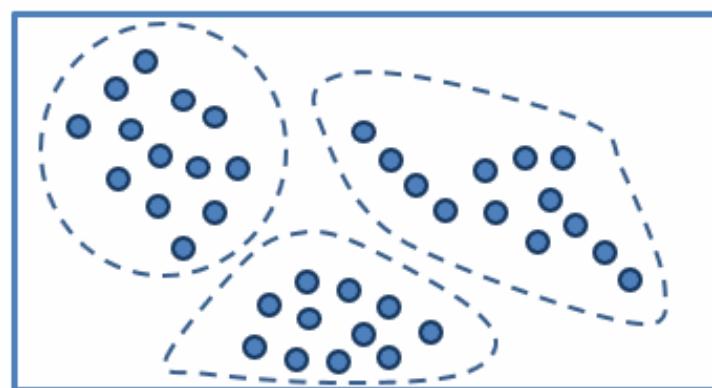
- 비지도 학습 알고리즘
- 주어진 n 개의 관측값을 k 개의 클러스터로 분할
- 관측값들과 거리가 최소인 클러스터로 군집화

Classification



Supervised learning

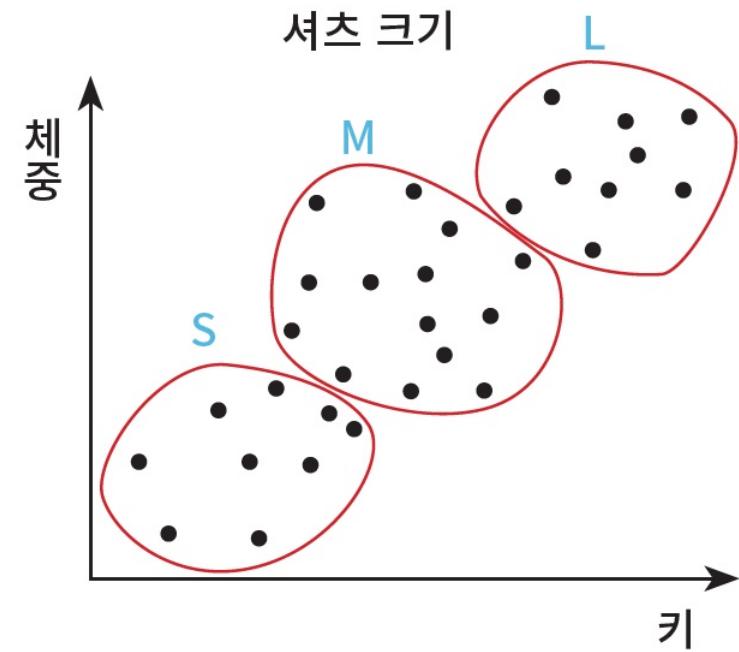
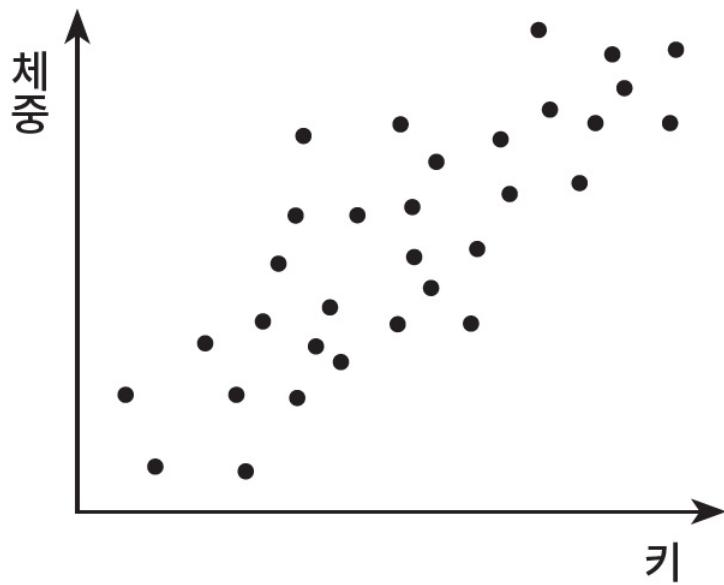
Clustering



Unsupervised learning

k-means 클러스터링

- 셔츠를 만들어서 판매하는 회사는 시장에 새로운 셔츠 모델을 공개
- 회사는 사람들의 키와 체중을 조사하여 그래프로 분석



k-means 클러스터링

● **입력값** : 클러스터 수 k , 훈련 데이터셋 D

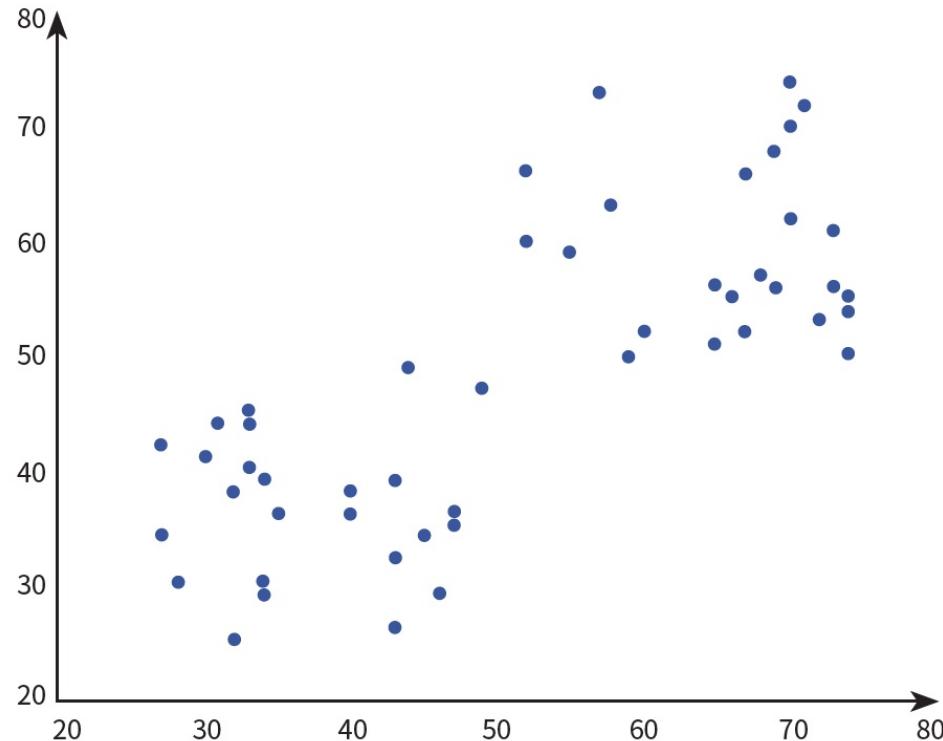
● **출력값** : k 개의 클러스터 데이터

● **알고리즘**

1. D 에서 k 개의 데이터를 임의로 추출하고, 이 데이터들을 각 클러스터의 중심 (centroid)으로 설정 (초기값 설정)
2. D 의 각 데이터에 대해 k 개의 클러스터 중심과의 거리를 계산하고, 각 데이터가 어느 중심점 (centroid)과 가장 유사도가 높은지 파악하고 찾아낸 중심점으로 각 데이터들을 배정
3. 클러스터의 중심점을 다시 계산
4. 각 데이터의 소속 클러스터가 바뀌지 않을 때까지 과정 2, 3을 반복

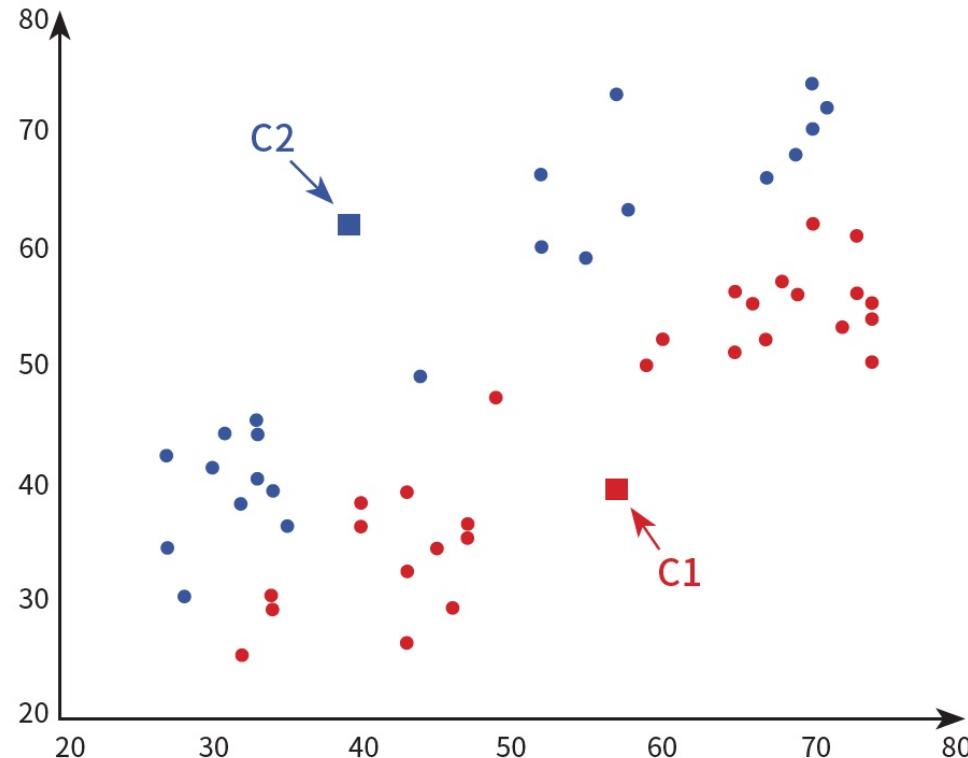
k -means 클러스터링

- 주어진 데이터셋을 2개의 클러스터로 구룹화 ($k=2$)



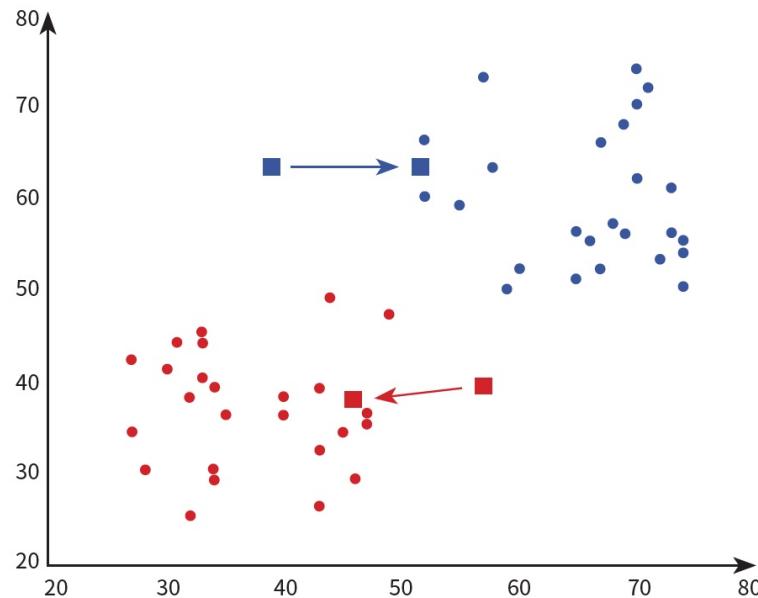
k -means 클러스터링

- 무작위로 2개의 중심점을 선택



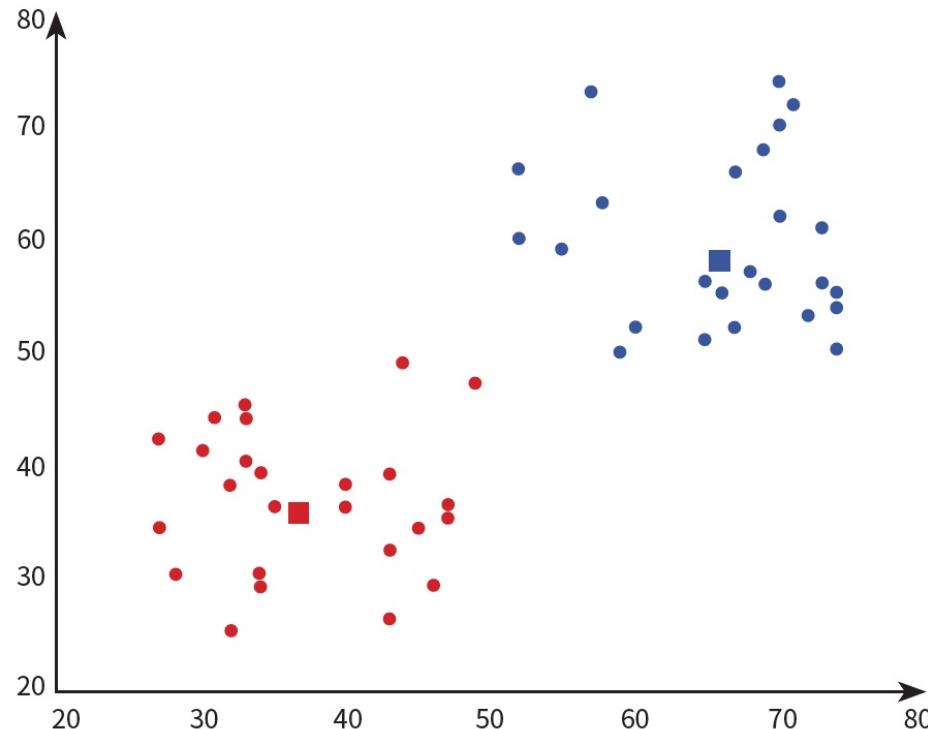
k -means 클러스터링

- 모든 파란색 점과 빨간색 점의 평균을 따로 계산
- 이 점이 클러스터의 새로운 중심이 됨

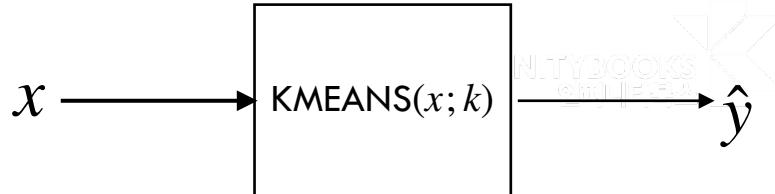


k -means 클러스터링

- 2개의 중심점의 위치가 변하지 않을 때까지 2단계와 3단계를 반복

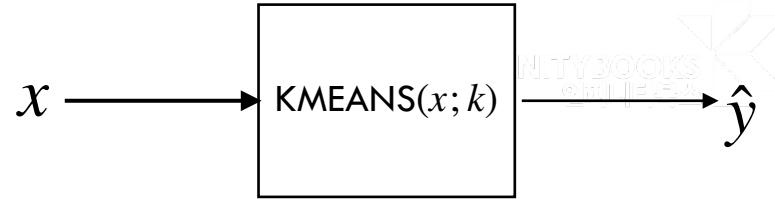


KMeans 평가



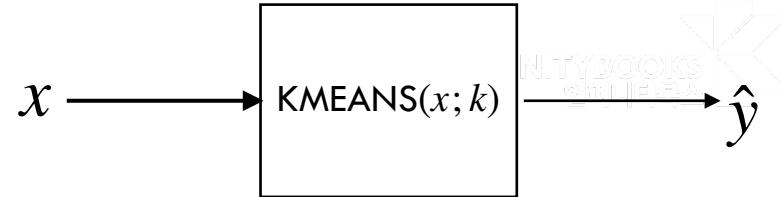
- 데이터셋 $D = \{x_i \mid i = 1, \dots, N\}$
 - 입력값 $x_i \in \mathbb{R}^d, d \geq 2$
- 학습된 모델 $\hat{y} = \text{KMEANS}(x; k)$
 - 입력값 $x \in \mathbb{R}^d$
 - 출력값 $\hat{y} \in \{1, 2, \dots, k\}$
- 데이터 (x_i, y_i) 의 평가
 - 모델 $\hat{y}_i = \text{KMEANS}(x_i; k)$

KMeans 평가

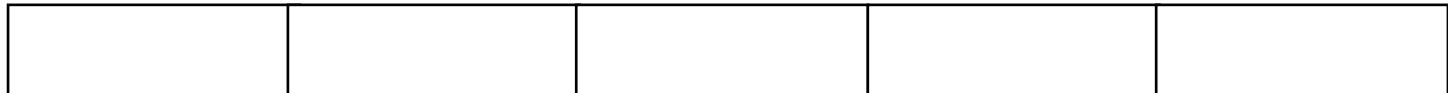


- 객관적 평가를 여러 경우에 모델을 평가
- 데이터셋을 훈련과 테스트 데이터셋으로 분리 $D = D_{train} \cup D_{test}$
 - 훈련 데이터셋 $D_{train}, N_1 = |D_{train}|$
 - 테스트 데이터셋 $D_{test}, N_2 = |D_{test}|$
 - 나누는 비율은 8:2가 보편적으로 선택. 그러나 데이터셋 크기에 따라 다름
 - 임의 선택을 이용하여 데이터셋을 나눔
- 주어진 데이터셋에서 반복적으로 평가를 진행
 - 훈련과 테스트 데이터셋에 대해 동일하게 평가
 - k-way 교차 평가: 훈련과 테스트 데이터셋으로 나누어 k번 평가 진행
 - k-way 검증 교차 평가: 훈련, 검증 과 테스트 데이터셋으로 나누어 k번 평가 진행

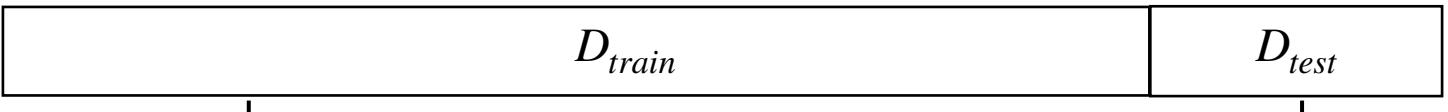
KMeans 평가



$$D = \{x_i \mid i = 1, \dots, N\}$$



$$D = D_{train} \cup D_{test}$$

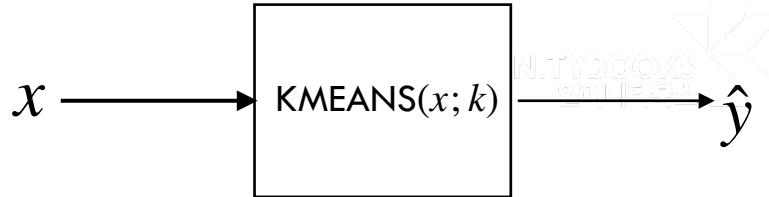


1	x_1	\hat{y}_1	y_1
2	x_2	\hat{y}_2	y_2
\vdots			
N_1	x_{N_1}	\hat{y}_{N_1}	y_{N_1}

1	x_1	\hat{y}_1	y_1
2	x_2	\hat{y}_2	y_2
\vdots			
N_2	x_{N_2}	\hat{y}_{N_2}	y_{N_2}

$$\hat{y} = \arg \min_c D(x, u_i), i = 1, 2, \dots, k$$

KMeans 평가



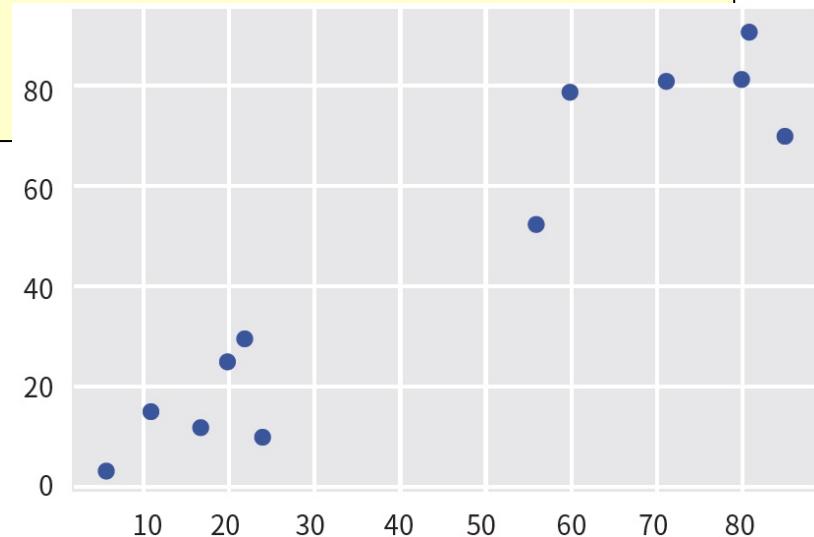
- 클러스터의 중심점의 수가 적절한가?
- 훈련된 클러스터내 데이터들 간의 평균거리가 최소인가?
- 훈련된 클러스터들 간의 데이터들의 거리가 최소인가?

사례

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import Kmeans

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])
```



사례



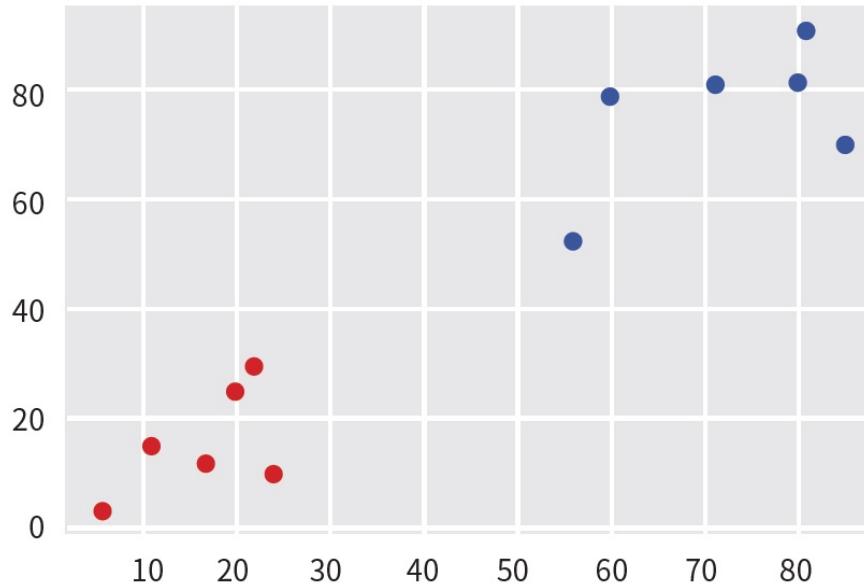
```
kmeans = KMeans(n_clusters=2)  
kmeans.fit(X)  
  
print(kmeans.cluster_centers_)
```

```
[[72.16666667 75.66666667]  
 [16.66666667 15.83333333]]
```

사례



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

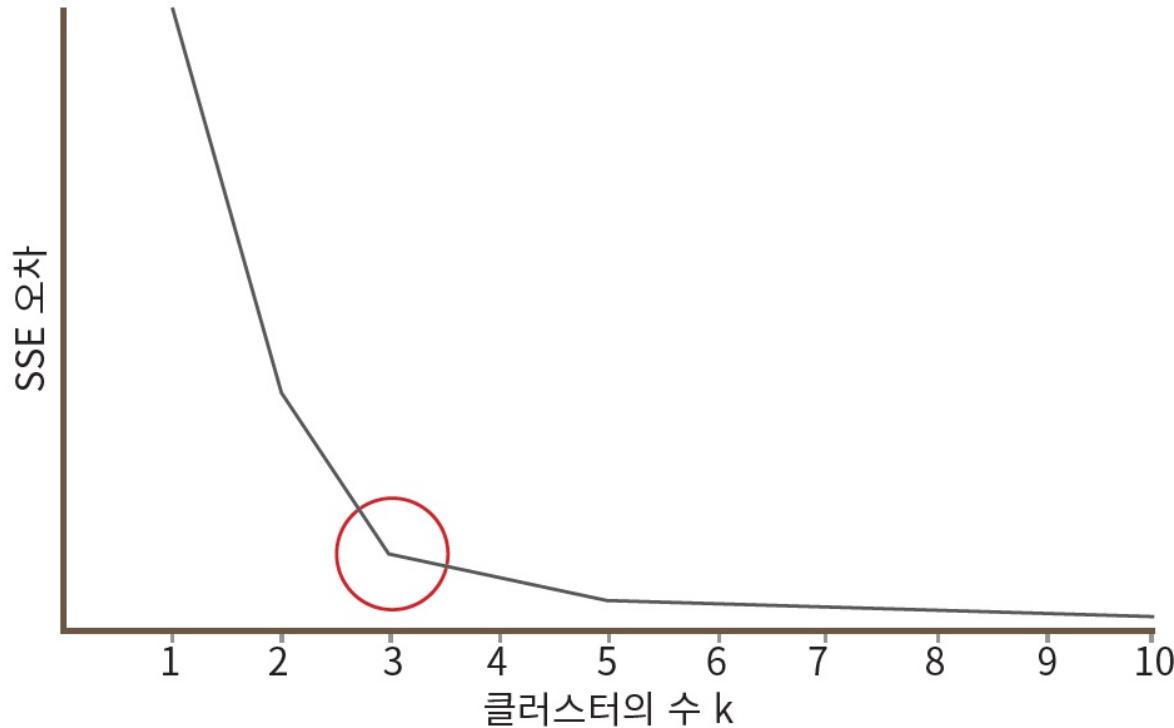


*k*의 결정

- "팔꿈치" 방법(elbow method)에서는 *k*를 1부터 증가시키면서 *k*-means 클러스터링을 수행
- 각 *k*의 값에 대하여 SSE(sum of squared errors)의 값을 계산

```
var sse = {};
for (var k = 1; k <= maxK; ++k) {
  sse[k] = 0;
  clusters = kmeans(dataset, k);
  clusters.forEach(function(cluster) {
    mean = clusterMean(cluster);
    cluster.forEach(function(datapoint) {
      sse[k] += math.pow(datapoint - mean, 2);
    });
  });
}
```

k를 결정 방법



k를 결정 방법



```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

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]])

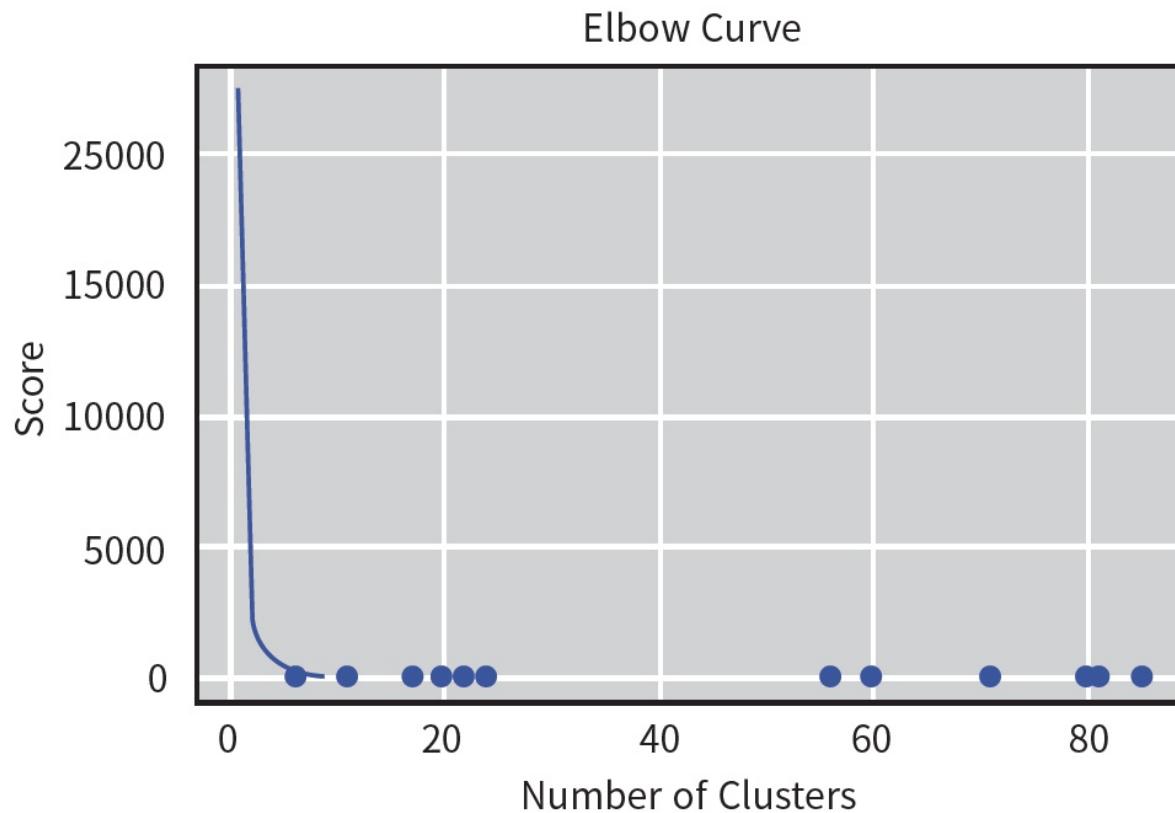
plt.scatter(X[:,0],X[:,1])

n_clusters = range(1, 10)
kmeans = [KMeans(n_clusters=i) for i in n_clusters]

# 모든 샘플에 대하여 제곱 오차를 계산하여 리스트에 추가
score = [kmeans[i].fit(X).inertia_ for i in range(len(kmeans))]

plt.plot(n_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```

k를 결정 방법



사례

● make_blobs()하는 함수 이용

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs

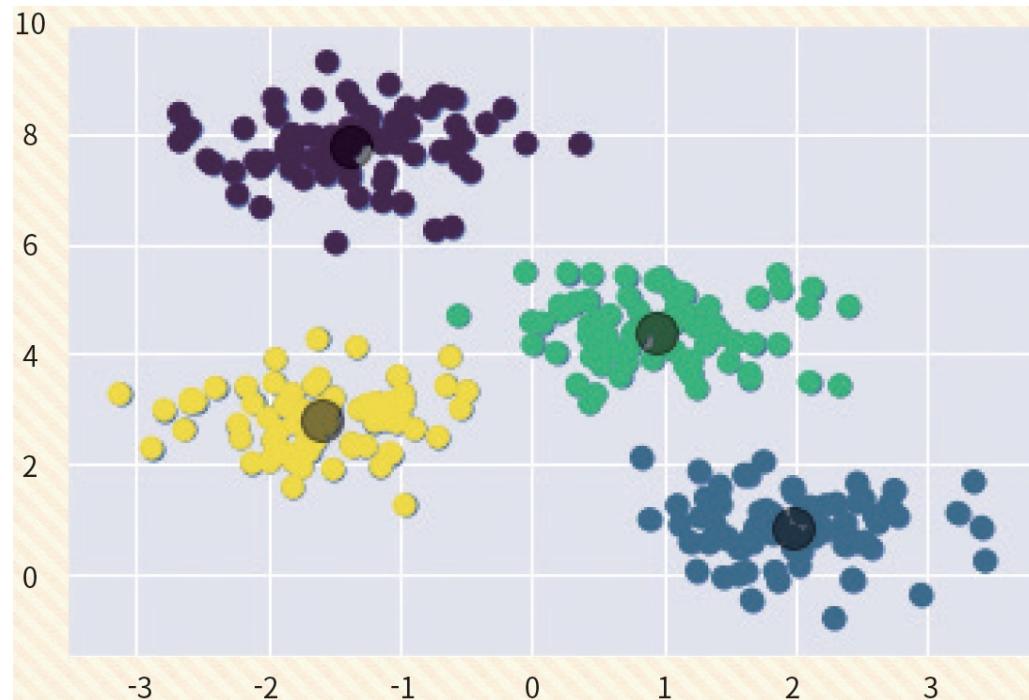
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);

kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

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

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

사례



정리

- k -NN 알고리즘은 학습 시에 클래스 정보가 제공되는 지도 학습임
 - k -NN 알고리즘은 새로운 데이터를 가장 가까운 이웃 클래스로 할당
 - k 는 홀수로 선택
 - k -NN 알고리즘은 어떤 종류의 학습이나 준비 시간이 필요 없지만 가장 가까운 이웃을 찾기 위해, 많은 메모리 공간과 계산 시간이 필요
- k -means 알고리즘은 비지도 학습
 - k -means 알고리즘은 각 클러스터의 중심 (centroid)과 클러스터 내의 데이터와의 거리의 제곱합을 비용 함수로 정의
 - 이 함수값을 최소화하는 방향으로 각 데이터의 소속 클러스터를 업데이트 해 클러스터링을 수행