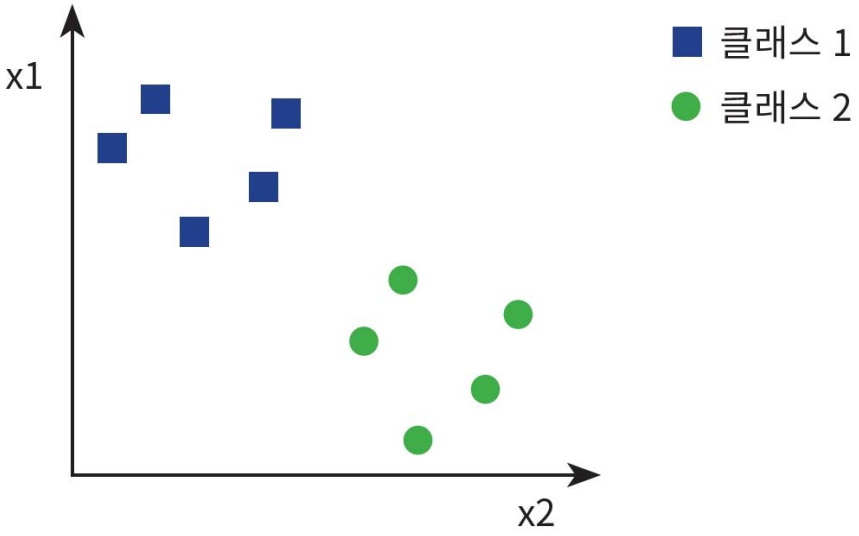
제11장 KNN알고리즘과 K-MEANS 알고리즘

학습 목표

- kNN 알고리즘을 이해한다.
- K-means 알고리즘을 이해한다.
- 비지도 학습을 이해한다.
- sklearn을 사용하여 kNN 알고리즘과 K-means 알고리즘을 구현할 수 있다.

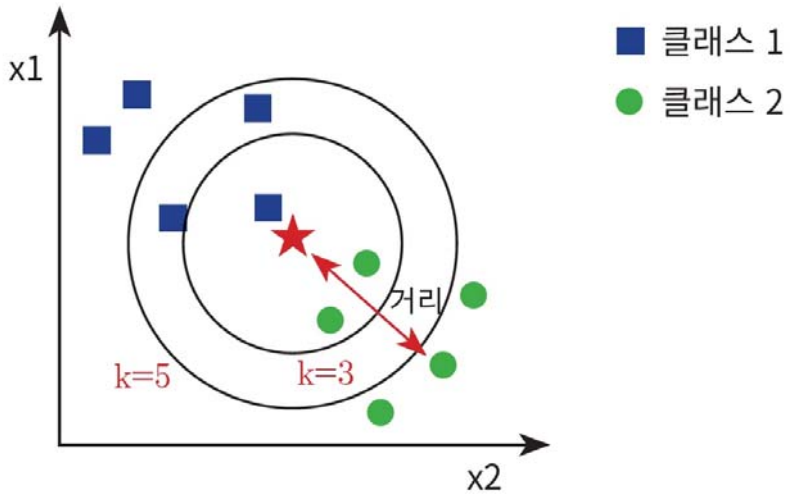
01 kNN 알고리즘

- k-Nearest Neighbor(kNN) 알고리즘은 분류 알고리즘으로 지도학습
  - 레이블이 정해진 데이터 자체가 학습된 결과임



02 kNN 알고리즘

- 이제 새로운 데이터가 입력되어서 그래프 상에 별표로 표시되었다고 하자. 별표는 파랑색 사각형과 빨강색 원 중에서 하나에 속해야 한다. 이것을 분류(classification)라고 한다.



kNN 알고리즘

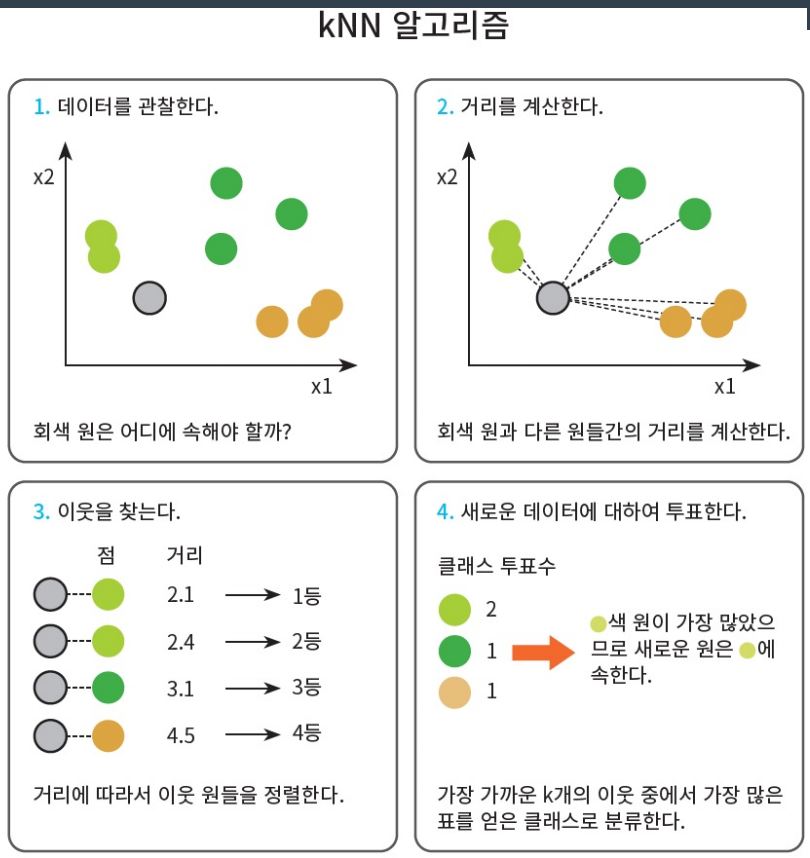


그림 11-3 kNN 알고리즘

ex.

- 어떤 사람이 양장점을 운영하고 있다. 한 고객의 성별을 잊어버렸는데 다행이도 키와 허리 사이즈는 적어두었다. 키는 165cm, 허리는 28인치이다. kNN 분류기를 이용하여 이 고객의 성별을 판단해보자. 학습 데이터는 다음과 같다. K = 3으로 하자.

성별	키(cm)	허리(인치)	거리계산
남성	175	33	
남성	180	35	
여성	160	27	
여성	170	31	

kNN 알고리즘

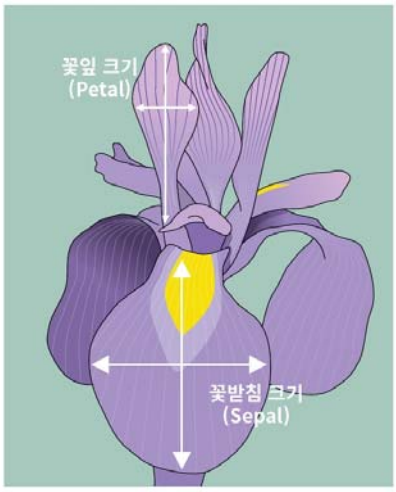
- 수정된 kNN 알고리즘
  - 신입 멤버와의 거리에 따라 각 도형에 약간의 가중치를 준다
    - 신입 멤버와 가까운 도형들은 더 높은 가중치를 얻고, 다른 도형들은 더 낮은 가중치를 갖는다
  - 그런 다음 모든 도형의 가중치를 각 도형별로 전부 합한다
  - 가장 큰 가중치를 얻은 도형이 새로운 멤버의 클래스가 된다

kNN 알고리즘의 장점과 단점

- 장점
  - 어떤 종류의 학습이나 준비 시간이 필요 없다.
- 단점
  - 특징 공간에 있는 모든 데이터에 대한 정보가 필요하다. 왜냐하면, 가장 가까운 이웃을 찾기 위해 새로운 데이터에서 모든 기존 데이터까지의 거리를 확인해야 하기 때문이다. 데이터와 클래스가 많이 있다면, 많은 메모리 공간과 계산 시간이 필요하다.

03 sklearn을 이용한 kNN 알고리즘 실습

- sklearn 라이브러리에는 아이리스 데이터 세트가 있다. 아이리스 데이터 세트에는 150개의 아이리스 꽃 샘플이 있다.
  - 꽃받침(sepal) 길이, 너비, 꽃잎(petal) 길이, 너비, 꽃 이름(label)




```
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
```

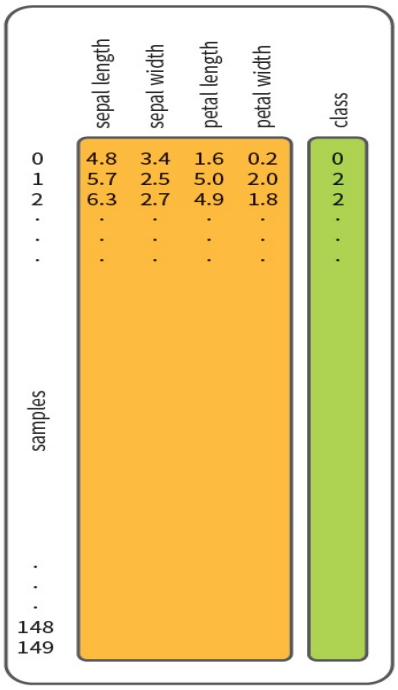
(아이리스 꽃: 꽃받침 길이와 너비, 꽃잎 길이와 너비)

sklearn을 이용한 kNN 알고리즘 실습

```
from sklearn.datasets import load_iris
iris = load_iris()
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],
        …
```

아이리스 데이터 세트



sklearn을 이용한 kNN 알고리즘 실습

```
# 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)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 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 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

kNN학습

```
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)
```


kNN학습

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

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

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

```
0.9666666666666667
```

kNN예측 KNN2.py

- 데이터를 나누지 않고 모두 훈련에 사용하고 새 데이터로 테스트하는 경우

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

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

# 아직 보지 못한 새로운 데이터를 제시해보자.
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
```


04 비지도 학습(k-means 클러스터링)

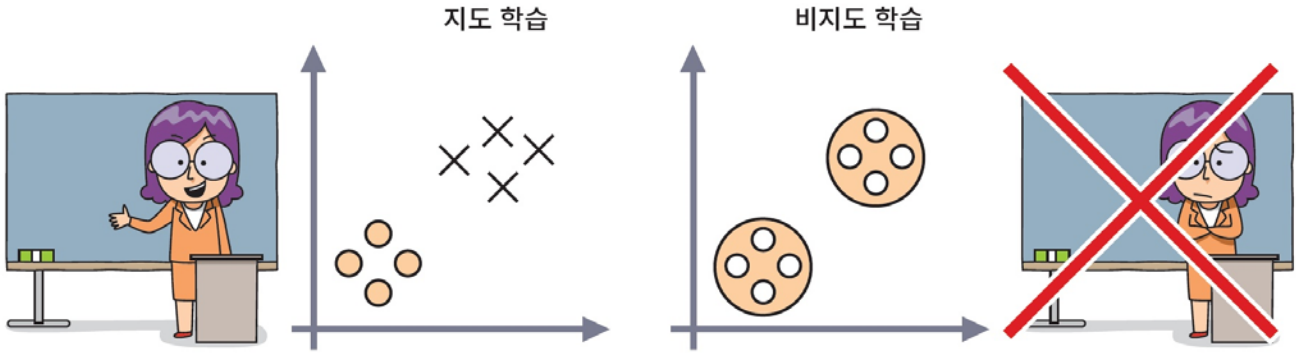


그림 11-6 지도 학습과 비지도 학습

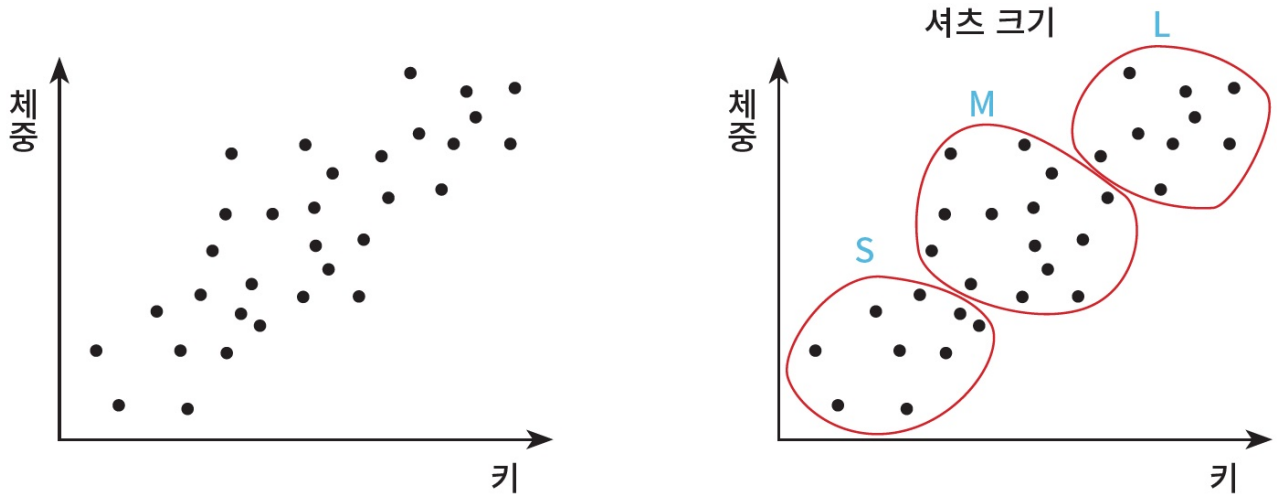
unsupervised learning

K-means 클러스터링

- 비지도 학습 중에서 가장 대표적인 것이 K-means 알고리즘이다.
- K-means 알고리즘(K-means algorithm)은 주어진 n개의 관측값을 k개의 클러스터로 분할하는 알고리즘으로, 관측값들은 거리가 최소인 클러스터로 분류된다.
  - 단점: NP-hard로 시간이 많이 걸린다.

K-means 클러스터링

- 셔츠를 만들어서 판매하는 회사를 생각해보자. 회사는 시장에 새로운 셔츠 모델을 공개하여야 한다. 회사는 사람들의 키와 체중을 조사하여 그래프로 그려보았다고 하자.



K-means 알고리즘

- 분할법
  - 주어진 데이터를 여러 그룹으로 나누는 방법
- 클러스터를 나누는 과정
  - 거리 기반의 비용 함수(cost function)를 최소화하는 방식
    - 비용 함수: 각 클러스터의 중심(centroid)과 클러스터 내의 데이터와의 거리를 제곱한 값
    - 이 함수값을 최소화하는 방향으로 각 데이터의 소속 클러스터를 업데이트 해 줌
  - 같은 클러스터 내 데이터끼리의 유사도는 증가
  - 다른 클러스터에 있는 데이터와의 유사도는 감소

K-means 클러스터링 알고리즘

입력값

1. k: 클러스터 수
2. D: n개의 데이터

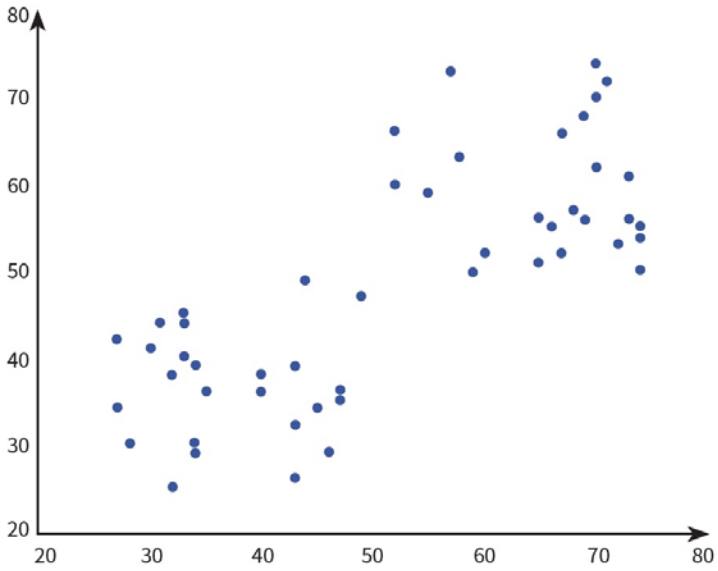
출력값: k개의 클러스터

알고리즘

1. 집합 D에서 k개의 데이터를 임의로 추출하고, 이 데이터들을 각 클러스터의 중심 (centroid) 으로 설정한다. (초기값 설정)
2. 집합 D의 각 데이터에 대해 k개의 클러스터 중심과의 거리를 계산하고, 각 데이터가 어느 중심점 (centroid)과 가장 유사도가 높은지 알아낸다. 그리고 그렇게 찾아낸 중심점으로 각 데이터들을 할당한다.
3. 클러스터의 중심점을 다시 계산한다. 즉, 2에서 재할당된 클러스터들을 기준으로 중심점을 다시 계산한다.
4. 각 데이터의 소속 클러스터가 바뀌지 않을 때까지 과정 2, 3을 반복한다.

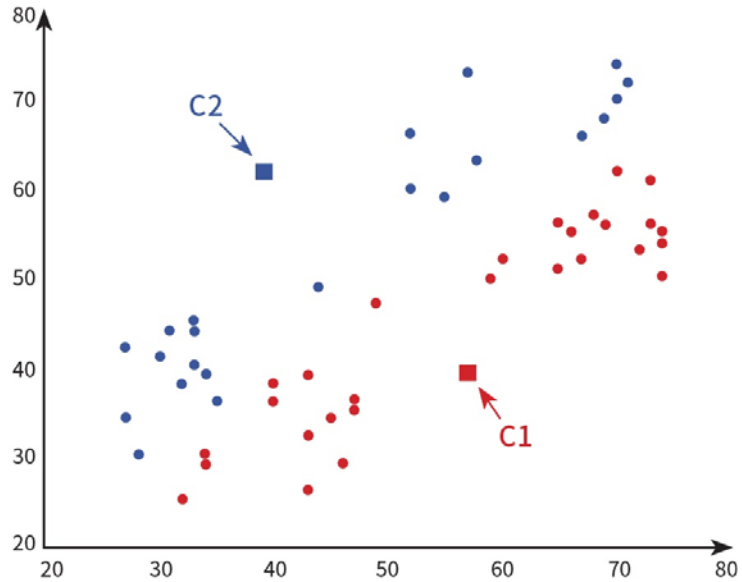
알고리즘 #1

- 알고리즘을 설명하기 위하여 다음과 같이 데이터들이 주어졌다고 하자. 우리는 데이터를 2개의 그룹으로 나누어야 한다. 즉 k=2이다.



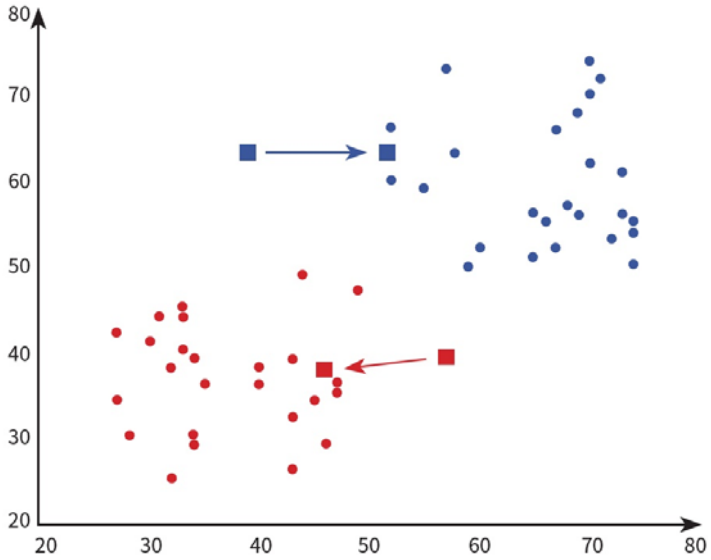
알고리즘 #2

- 알고리즘은 무작위로 2개의 중심점을 선택한다. 이것을 C1과 C2라고 하자.



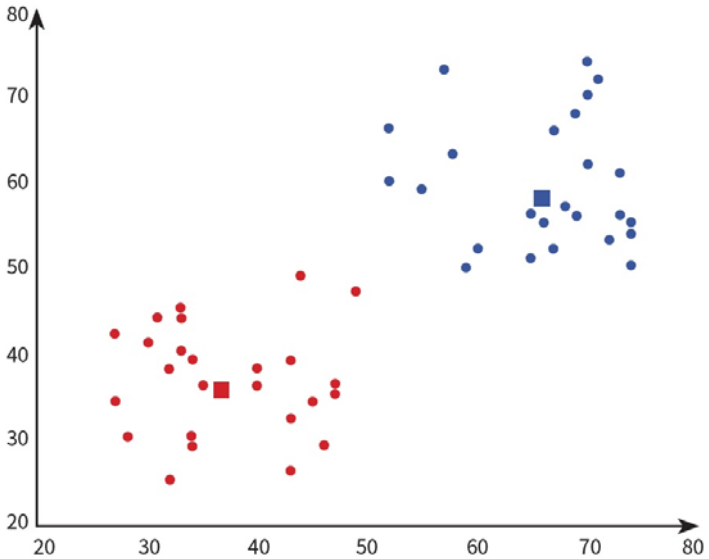
알고리즘 #3

- 모든 파란색 점과 빨간색 점의 평균을 따로 계산한다. 이 점이 클러스터의 새로운 중심이 된다.



알고리즘 #4

- 2개의 중심점의 위치가 변하지 않을 때까지 2단계와 3단계를 반복한다.



e.g.

- 다음과 같은 데이터가 있다고 하자. k=2이고 첫 번째 클러스터의 초기 중심은 P2, 두 번째 클러스터의 초기 중심은 P8로 한다. K-means 알고리즘의 한 단계만을 수행하라.

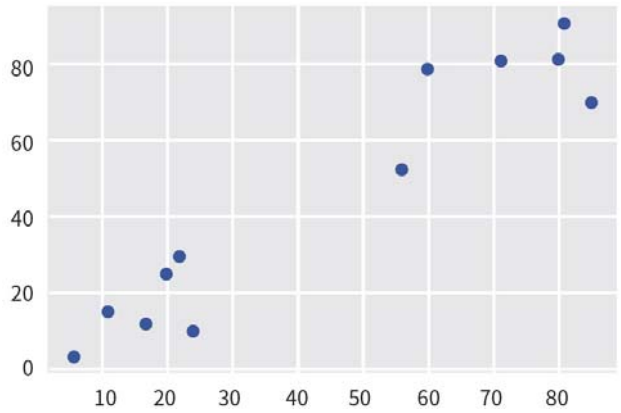
번호	X	Y
P1	2	3
P2	3	1
P3	4	2
P4	11	5
P5	12	4
P6	12	6
P7	7	5
P8	8	4
P9	8	6

05 sklearn을 이용한 K-means 클러스터링

- 여기서는 sklearn 라이브러리를 이용하여 K-means 클러스터링 알고리즘을 실습해보자.

```
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])
```

sklearn을 이용한 K-means 클러스터링



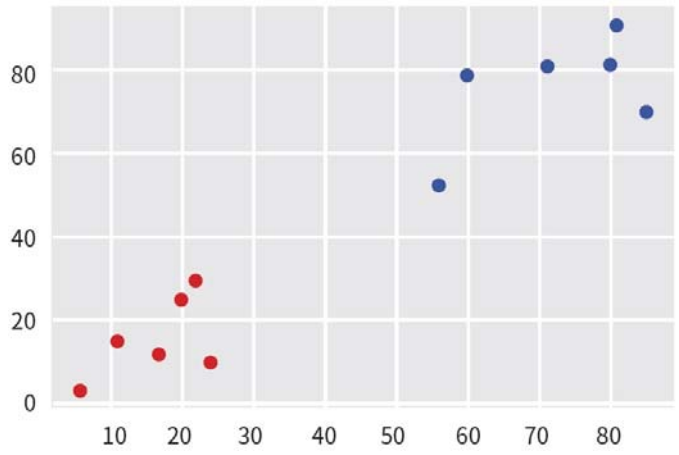
sklearn을 이용한 K-means 클러스터링

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

print(kmeans.cluster_centers_)
```

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

sklearn을 이용한 K-means 클러스터링

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

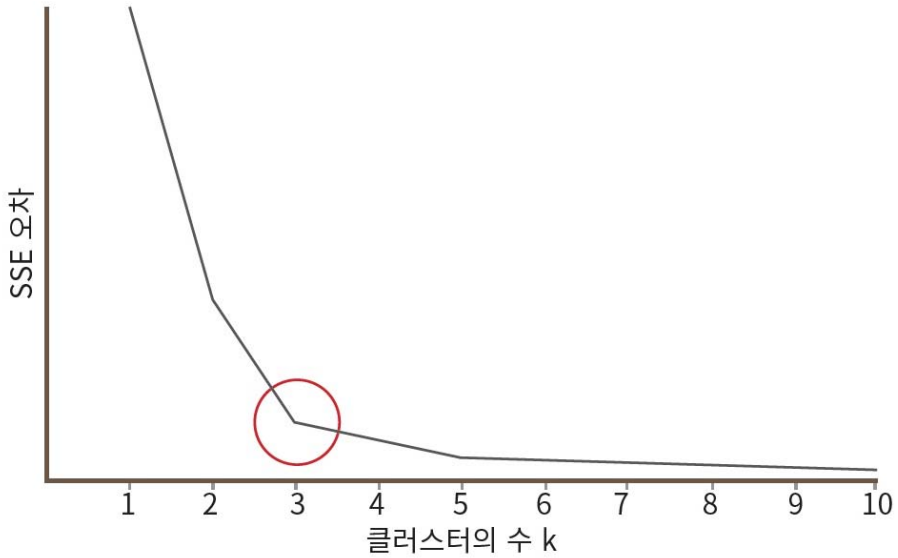


06 k를 결정하는 방법

- "팔꿈치" 방법(elbow method)에서는 k를 1부터 증가시키면서 K-means 클러스터링을 수행
  - 각 k의 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);                    // 각 cluster의 평균을 구하고
        cluster.forEach(function(datapoint) {
            sse[k] += Math.pow(datapoint - mean, 2); // cluster안의 에러 제곱합 계산
        });
    });
}
```

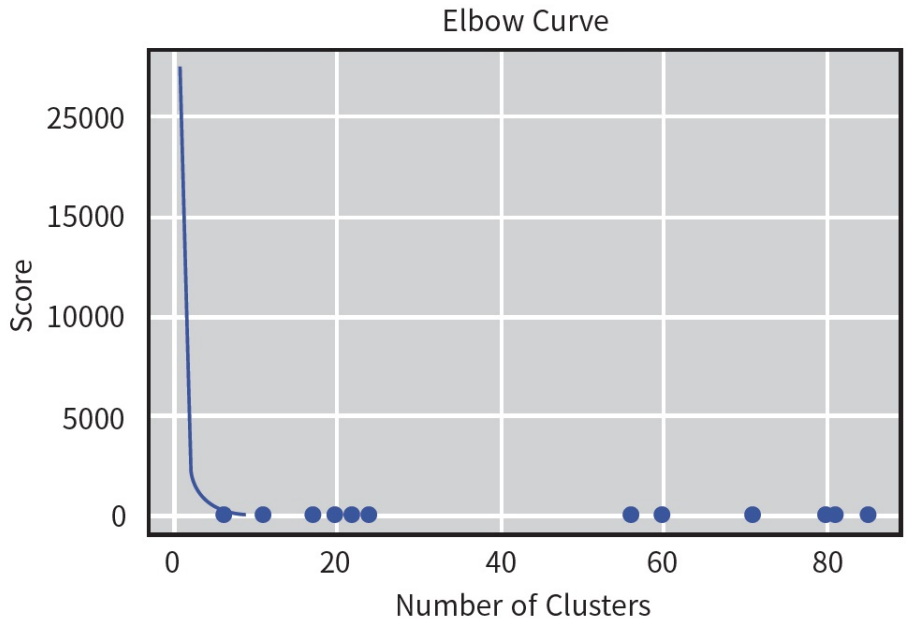
k를 결정하는 방법



k를 결정하는 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 추가한다.
score = [kmeans[i].fit(X).inertia_ for i in range(len(kmeans))]
plt.plot([min(n_clusters), max(n_clusters)], [max(score), min(score)], color='red')
plt.plot(n_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```

k를 결정하는 방법

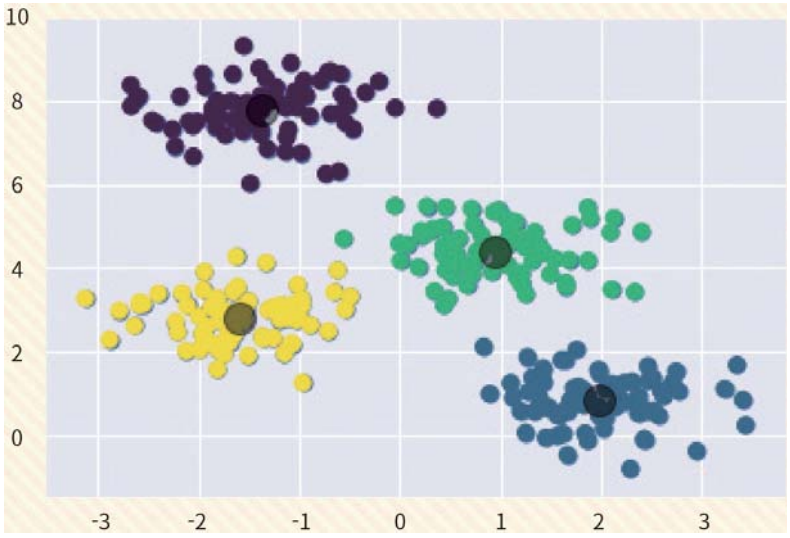


Lab: K-means 알고리즘 실습

- sklearn.datasets의 make_blobs() 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 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);
plt.show()
```

Lab: K-means 알고리즘 실습



Summary

- k-Nearest Neighbor(kNN) 알고리즘은 학습 시에 교사가 존재하는 "지도 학습"에 속한다.
- kNN 알고리즘은 새로운 데이터를 가장 가까운 이웃 클래스로 할당한다. k는 홀수로 하는 것이 좋다.
- kNN 알고리즘은 어떤 종류의 학습이나 준비 시간이 필요 없지만 가장 가까운 이웃을 찾기 위해, 많은 메모리 공간과 계산 시간이 필요하다.
- K-means 알고리즘은 비지도 학습이다.
- K-means 알고리즘은 각 클러스터의 중심 (centroid)과 클러스터 내의 데이터와의 거리의 제곱합을 비용 함수로 정하고, 이 함수값을 최소화하는 방향으로 각 데이터의 소속 클러스터를 업데이트 해 줌으로써 클러스터링을 수행하게 된다.

Q & A