



SESSION # 10

By Team 3
@ 김소정 이영준 김재훈
Date _ 2018.01.02

CONTENTS

01 Clustering이란?

02 k-Means Clustering

03 상향군집화

04 k-Means Clustering with Scikit-Learn

01 Clustering이란?

Clustering

군집 분석

1. 자료를 동질적인 군집 혹은 부분으로 나누는 분석.
2. 각 군집을 쉽게 설명하므로 전체를 설명하고자 한다.

→ 변수들 간 유사성을 기준으로



거리

01 Clustering이란?

Clustering

군집 분석

1. 자료를 동질적인 군집 혹은 부분으로 나누는 분석.
2. 각 군집을 쉽게 설명하므로 전체를 설명하고자 한다.

Supervised learning vs. Unsupervised learning

- | | |
|---|--|
| <ol style="list-style-type: none"> 1. 자료가 반응변수 Y 와 설명변수 X_1, \dots, X_p로 구성 2. 설명변수 X_1, \dots, X_p가 주어졌을 때, 반응변수 Y의 예측에 목적 | <ol style="list-style-type: none"> 1. 반응변수 Y는 없고 변수들 X_1, \dots, X_p만 있는 경우 2. 예측에는 관심X
변수들 사이의 특별한 관계가 있는가?
관측치들에 그룹이 있는가 등에 관심 |
|---|--|

01 Clustering이란?

Clustering

군집 분석

1. 자료를 동질적인 군집 혹은 부분으로 나누는 분석.
2. 각 군집을 쉽게 설명하므로 전체를 설명하고자 한다.

적용 예시 → Market Segmentation을 하기 위해

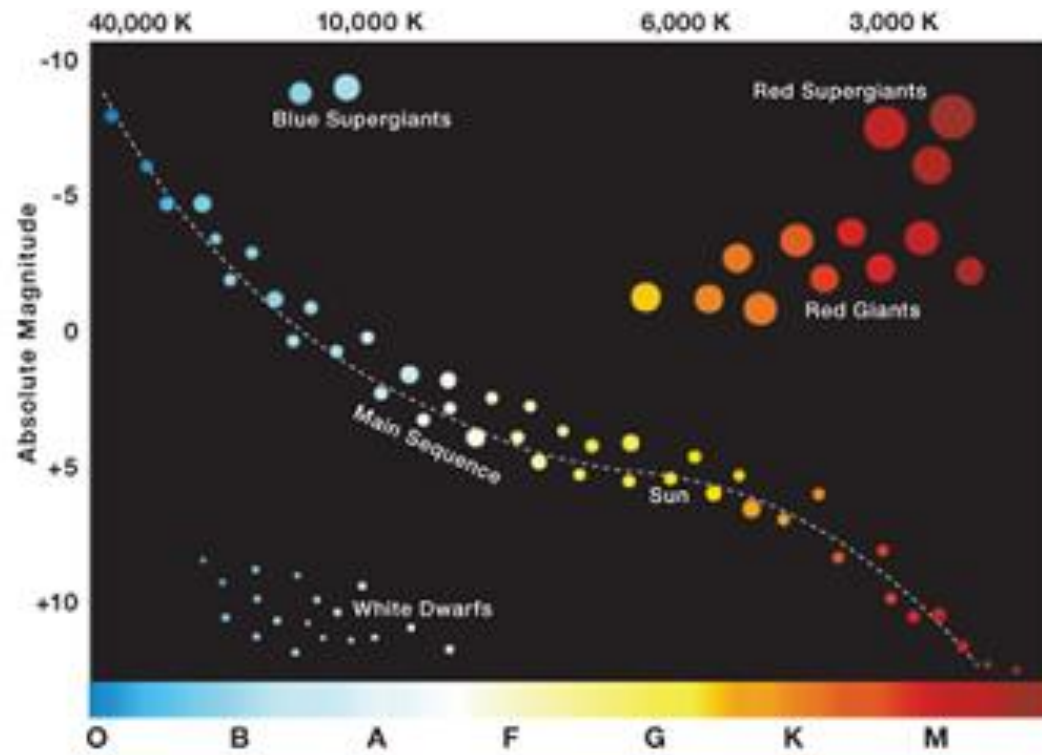
- Mkt Segmentation

: the process of dividing a **market** of potential customers into groups, or segments, based on different characteristics. The segments created are composed of consumers who will respond similarly to **marketing** strategies and who share traits such as similar interests, needs, or locations

O1 Clustering이란?

Clustering

헤르츠스프롱-러셀 다이어그램



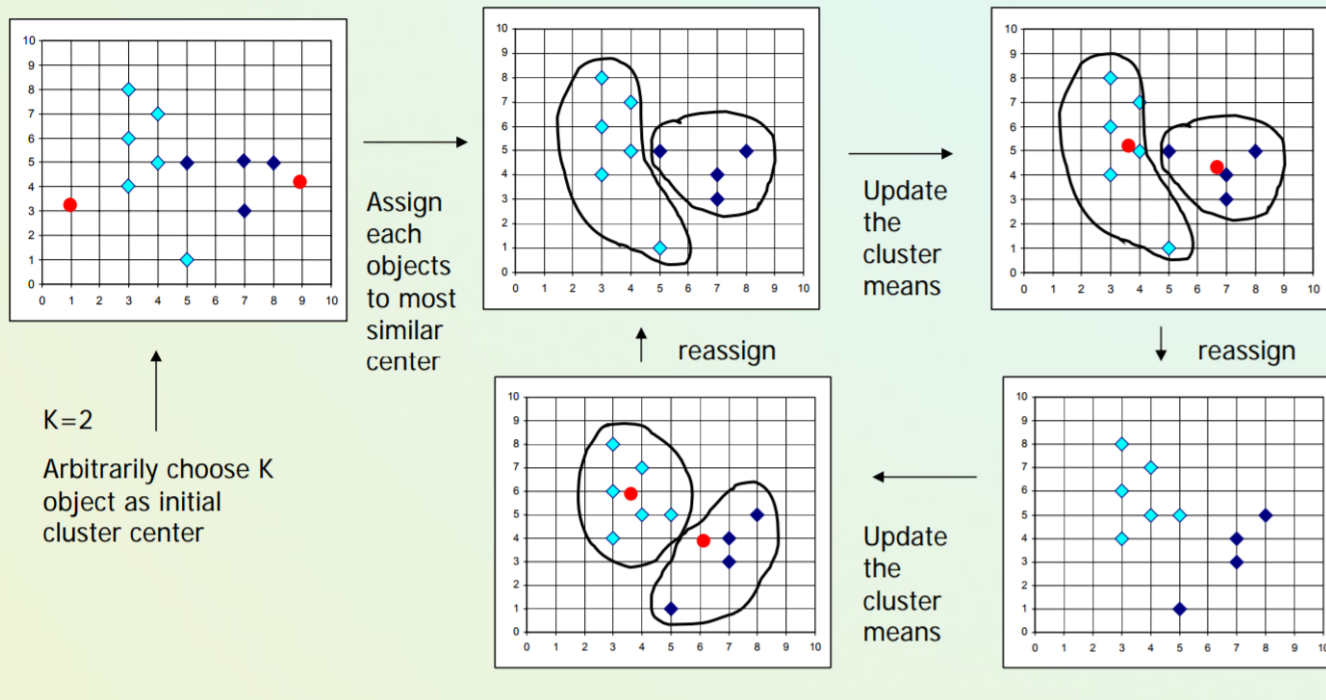
02 k-Means Clustering

시각화 된 예시

<http://www.cs.put.poznan.pl/jstefanowski/sed/DM-7clusteringnew.pdf>

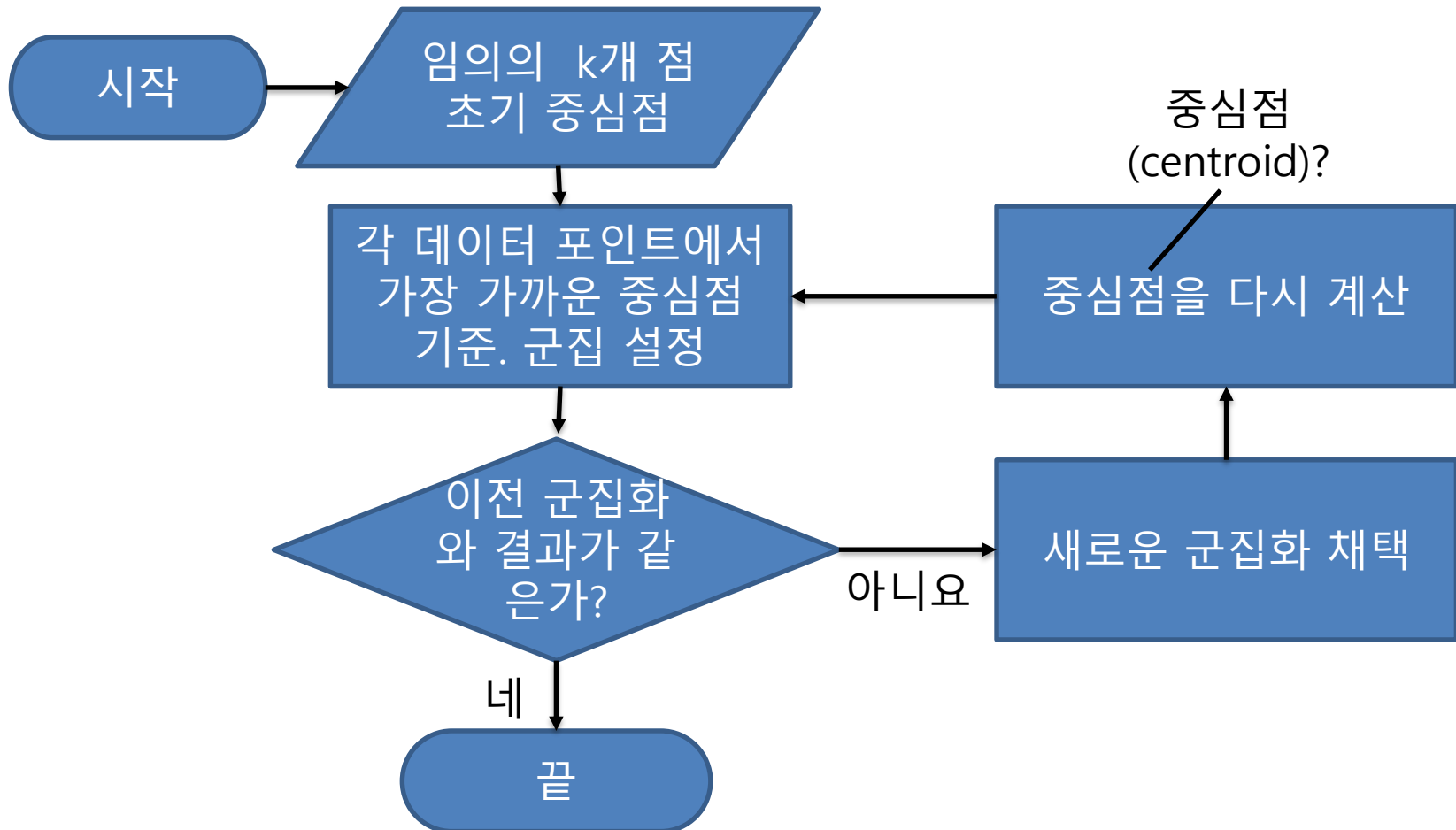
Illustrating *K-Means*

- Example



02 k-Means Clustering

알고리즘

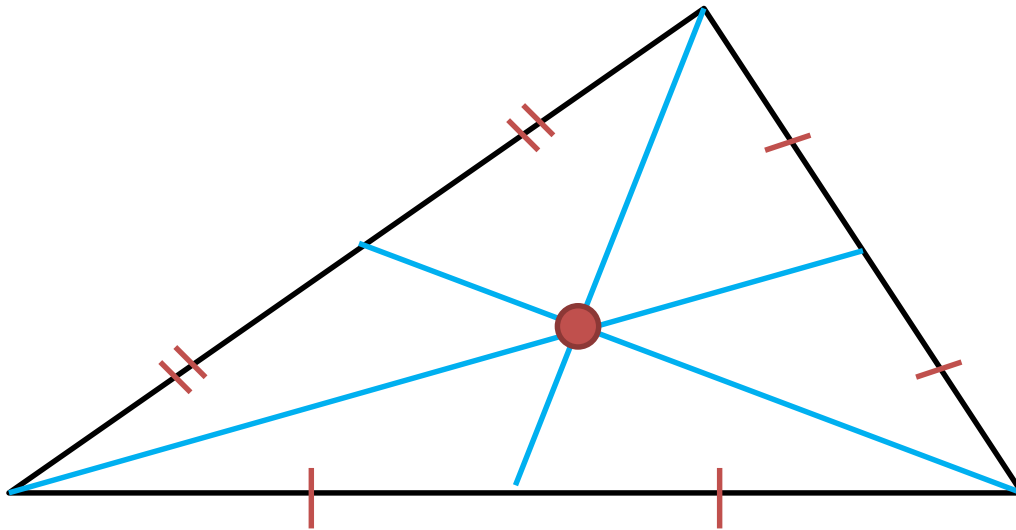


02 k-Means Clustering

중심점(centroid)

the **center** of mass of a geometric object of uniform density.

10여년 전... 중학교 2학년 수학 기억하시나요...??
삼각형의 (무게)중심 구하기

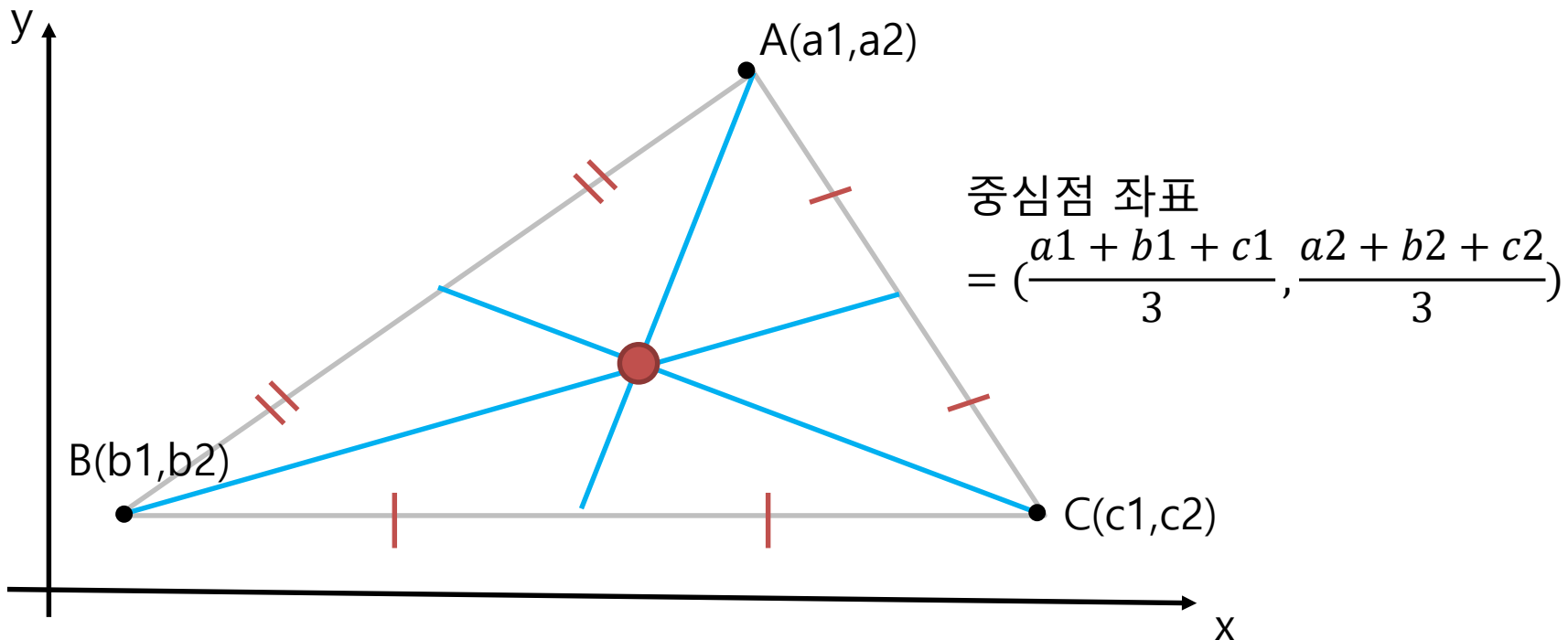


02 k-Means Clustering

중심점(centroid)

the **center** of mass of a geometric object of uniform density.

10여년 전... 중학교 2학년 수학 기억하시나요...??
삼각형의 (무게)중심 구하기

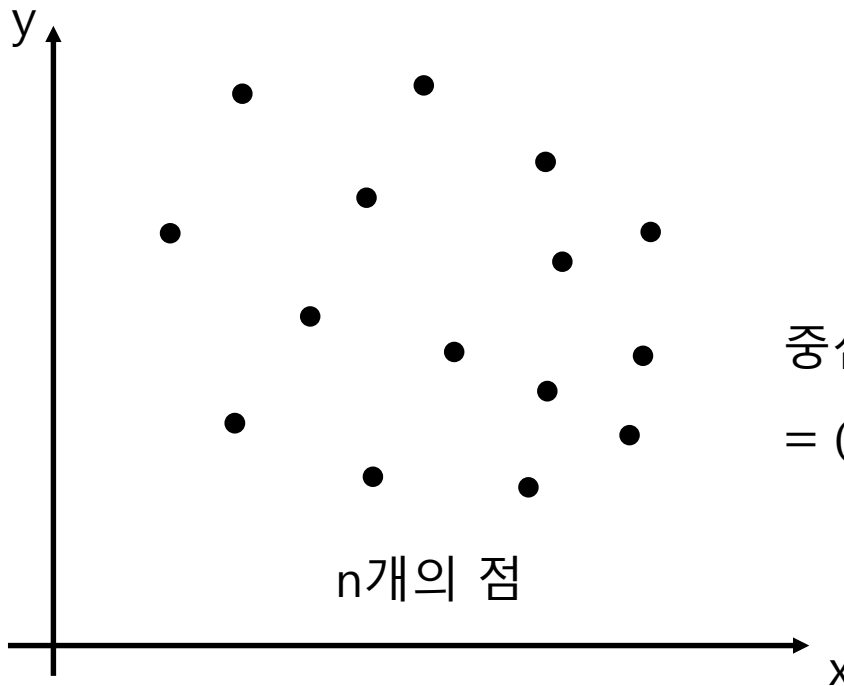


02 k-Means Clustering

중심점(centroid)

the **center** of mass of a geometric object of uniform density.

삼각형에서 확장하기!

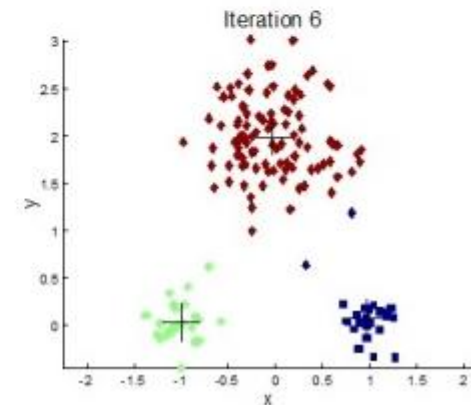
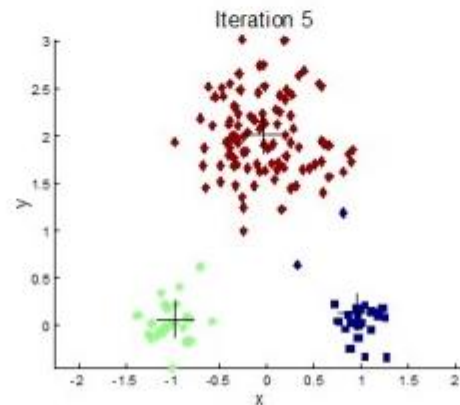
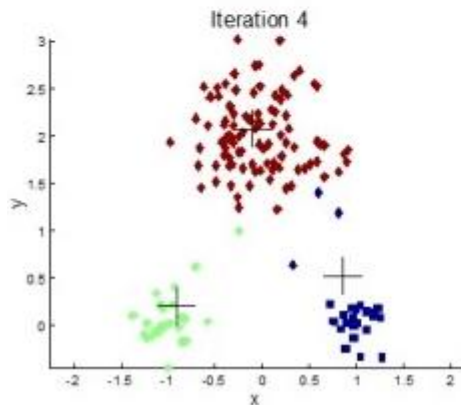
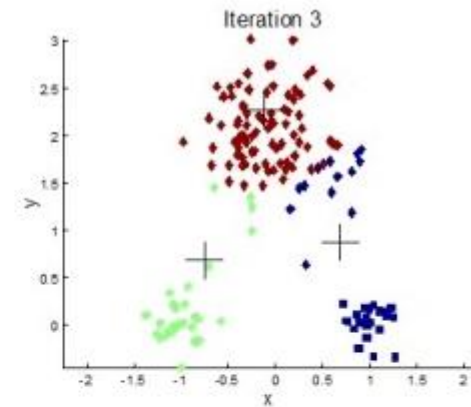
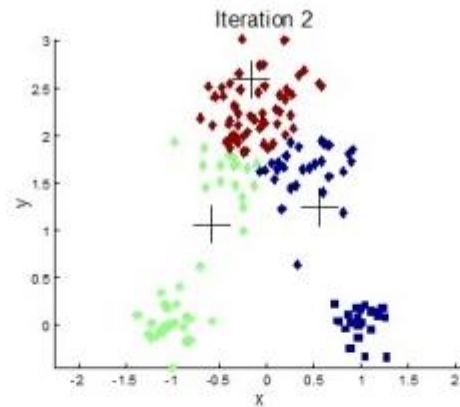
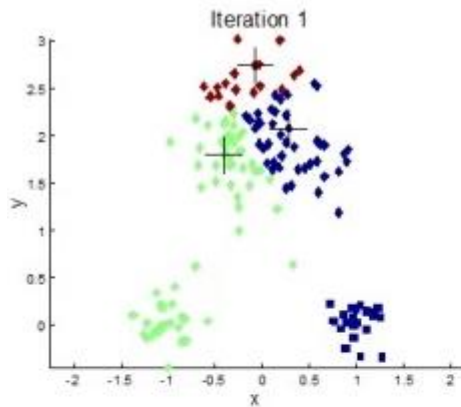


$$\begin{aligned} &\text{중심점} \\ &= \left(\frac{a_{11} + a_{12} + \cdots + a_{1n}}{n}, \frac{b_{11} + b_{12} + \cdots + b_{1n}}{n} \right) \end{aligned}$$

02 k-Means Clustering

시각화 된 예시

https://datasciencelab.files.wordpress.com/2013/12/p_n200_k3.gif?w=404&zoom=2



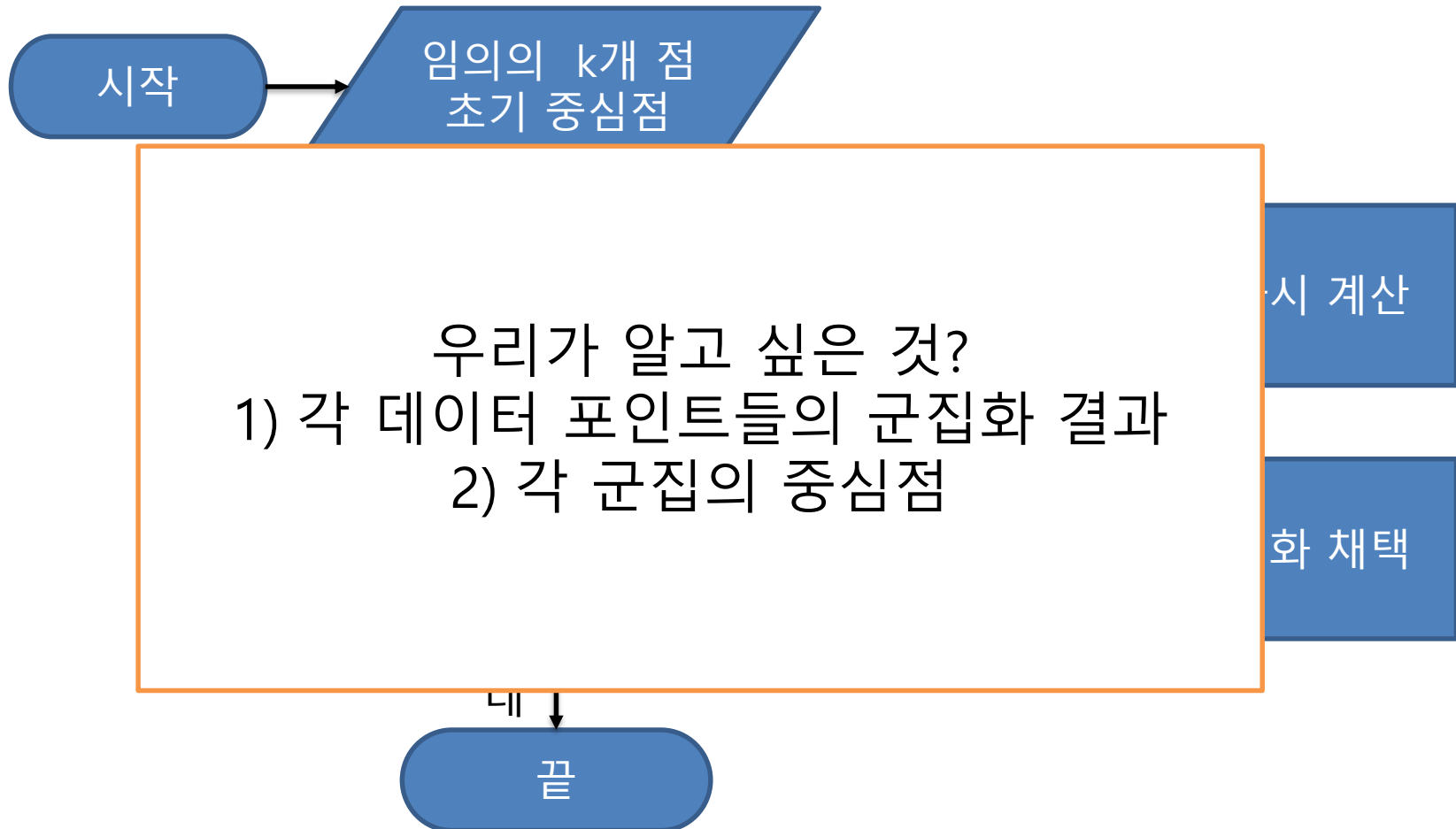
02 k-Means Clustering

시각화 된 예시

<https://datasciencelab.wordpress.com/2013/12/12/clustering-with-k-means-in-python/>

02 k-Means Clustering

알고리즘



02 k-Means Clustering

Coding

```
import os
os.chdir("_____")

from linear_algebra import squared_distance, vector_mean, distance

import math, random
import matplotlib.pyplot as plt
```

02 k-Means Clustering

Coding

```
class KMeans:
```

```
    def __init__(self, k):
```

```
        self.k = k          # number of clusters
```

```
        self.means = None    # cluster의 중심점
```

```
        self.assignments = None # 각 데이터 포인트들의 군집화 결과
```

꺼내고 싶은 값들을 적어주세요!
Empty Space로 지정

02 k-Means Clustering

Coding

```
def train(self, inputs):
```

벡터 데이터
포인트

Train 함수
→ 군집화, 각 군
집의 중심점을
찾는다!

```
self.m = inputs, self.k)
self.m
```

초기 설정: 임의의 k개
점 뽑아서 '중심점'으로!

무한반복

주어진 중심점으로 각 데이터 점들의 소속되는 군집 찾기

이전 군집화와 결과가 같은가?
Yes → 끝!

No
→ 새로운 중심점 계산하기

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))

if self.assignments == new_assignments:
    return

self.assignments = new_assignments

for i in range(self.k):
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
    # avoid divide-by-zero if i_points is empty
    if i_points:
        self.means[i] = vector_mean(i_points)
```

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

#요소들이 어느 (변경된)중심점에 가까이 있는지 매핑 [1,1,0,2,1, ... 이런식

```
self.assignments = new_assignments
```

```
for i in range(self.k):
```

```
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
    # avoid divide-by-zero if i_points is empty
```

```
    if i_points:
```

```
        self.means[i] = vector_mean(i_points)
```

```
map(func, seq)
```

- seq: list와 같은 개념!
- seq에 있는 모든 원소들에 대하여 func을 적용
- 결과값으로 새롭게 계산된 원소들의 list를 내놓음

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments == new_assignments:  
    return
```

정의가 필요하겠구나!
→ 위로!

```
self.assignments = new_assignments
```

```
for i in range(self.k):
```

```
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]  
    # avoid divide-by-zero if i_points is empty
```

```
    if i_points:
```

```
        self.means[i] = vector_mean(i_points)
```

02 k-Means Clustering

Coding

```
def classify(self, input):
```

해당 데이터 포인트를
어떤 클러스터에 넣을까?

들어가는 클러스터를 반환하게! **min 함수**를 써서!

`min(iterable, key = lambda function)`
→ iterable에서 key 함수를 거친 결과값
중에 가장 큰 결과값의 원래값을 반
환

EX) `min([1, 2, 3])`
> 1

02 k-Means Clustering

Coding

```
def classify(self, input):  
    return min(range(self.k),  
               key=lambda i: squared_distance(input, self.means[i]))
```

lambda는 함수를 생성할 때 사용(def와 동일). 함수를 한 줄로 간결하게 만들 때 사용.

lambda 인수1, 인수2, ... : 인수를 이용한 표현식

EX) sum = **lambda** a, b: a+b
sum(3,4)
>>7

02 k-Means Clustering

Coding

```
def classify(self, input):  
    return min(range(self.k),  
               key=lambda i: squared_distance(input, self.means[i]))
```

```
range(10)  
→ [0, 1, ..., 9]
```

#input 값이 k개의 means 중 어디와 가장 가까운지 찾아서 1,2,..k 중 하나를 리턴

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))

if self.assignments == new_assignments:
    return

self.assignments = new_assignments

for i in range(self.k):
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
    # avoid divide-by-zero if i_points is empty
    if i_points:
        self.means[i] = vector_mean(i_points)
```


02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments:
    return
```

```
self.assignments =
```

```
for i in range(self.k):
```

```
    i_points = [p for p in inputs if p == i]
```

```
    # avoid divide-by-zero
```

```
    if i_points:
```

```
        self.means[i] = vector_mean(i_points)
```

```
In [83]: X=[[1,2],[3,4]]
```

```
In [84]: Y=map(sum, X)
```

```
In [85]: print(Y)
```

```
<map object at 0x000002ACD3B69B38>
```

```
In [86]: Y=list(map(sum, X))
```

```
In [87]: print(Y)
```

```
[3, 7]
```

```
In [88]:
```

Python console

History log

IPython console

Permissions: **RW**

End-of-lines: **LF**

Encoding: **UTF-8-GUESSE**

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments == new_assignments:  
    return
```

기존 assignments와 new_assignments
S 같다면 while loop 끝낸다!

```
for i in range(self.k):  
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]  
    # avoid divide-by-zero if i_points is empty  
    if i_points:  
        self.means[i] = vector_mean(i_points)
```

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments == new_assignments:
    return
```

```
self.assignments = new_assignments
```

f 기존 assignments를 new_assignments
로 대체한다!

```
for i, (a, p) in enumerate(zip(new_assignments, self.inputs)):
    if a == i:
```

```
        # avoid divide-by-zero if i_points is empty
```

```
        if i_points:
```

```
            self.means[i] = vector_mean(i_points)
```

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments == new_assignments:
    return
```

```
self.assignments = new_assignments
```

```
for i in range(self.k):
```

```
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
```

```
    if i_points:
```

```
        self.means[i] = vec
```

```
In [90]: X=[1,2,3]
```

```
...: Y=['a','b','c']
```

```
...: Z=list(zip(X,Y))
```

```
...: print(Z)
```

```
...:
```

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

```
In [91]:
```

Python console

History log

IPython cor

Permissions: **RW**

End-of-lines: **LF**

Enc

새로운 중심점
계산

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))
```

```
if self.assignments == new_assignments:  
    return
```

```
self.assignments = new_assignments
```

```
for i in range(self.k):
```

```
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
```

```
    i 군집에 해당되는 데이터 포인트들 뽑아서 묶어주기
```

```
    self.means[i] = vector_mean(i_points)
```

새로운 중심점
계산

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))

if self.assignments == new_assignments:
    return

self.assignments = new_assignments

for i in range(self.k):
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]

    if i_points:
        self.means[i] = vector_mean(i_points)
```

i 클러스터의 중심점 계산
→ 다시 올라가서 classify
작업 다시 시작!

새로운 중심점
계산

02 k-Means Clustering

Coding

```
new_assignments = list(map(self.classify, inputs))

if self.assignments == new_assignments:
    return

self.assignments = new_assignments

for i in range(self.k):
    i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
    if i_points:
        self.means[i] = vector_mean(i_points)
```

새로운 중심점
계산

linear_algebra.py 파일에 들어있어요!
 $\text{vector_mean}([[1,2], [3,4], [5,6]])$
 $= [(1+3+5)/3, (2+4+6)/3]$
 $= [3, 4]$

02 k-Means Clustering

Coding

```
class KMeans:
    def __init__(self, k):
        self.k = k          # number of clusters
        self.means = None    # means of clusters
        self.assignments = None #results of clustering
    def classify(self, input):
        return min(range(self.k),
                    key=lambda i: squared_distance(input, self.means[i]))
    def train(self, inputs):
        self.means = random.sample(inputs, self.k)
        self.assignments = None

        while True:
            new_assignments = list(map(self.classify, inputs))
            if self.assignments == new_assignments:
                return

            self.assignments = new_assignments
            for i in range(self.k):
                i_points = [p for p, a in zip(inputs, self.assignments) if a == i]
                # avoid divide-by-zero if i_points is empty
                if i_points:
                    self.means[i] = vector_mean(i_points)
```


02 k-Means Clustering

Coding 예시

```
inputs = [[-14,-5],[13,13],[20,23],[-19,-11],[-9,-16],[21,27],[-49,15],[26,13],[-46,5],[-34,-1],[11,15],[-49,0],[-22,-16],[19,28],[-12,-8],[-13,-19],[-41,8],[-11,-6],[-25,-9],[-18,-3]]
```

```
random.seed(0) # so you get the same results as me
clusterer = KMeans(3)
clusterer.train(inputs)
```

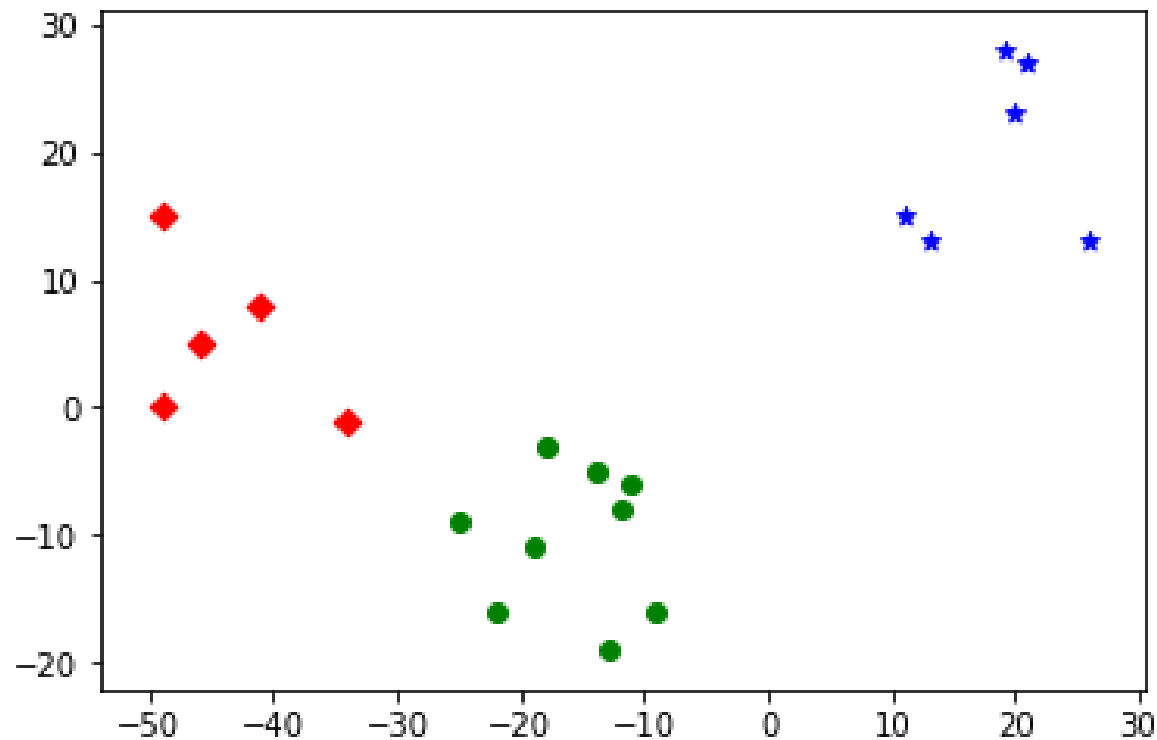
```
print(clusterer.means)
print(clusterer.assignments)
```

```
for i in range(len(inputs)):
    xs, ys = zip(inputs[i])
    if clusterer.assignments[i] == 0:
        plt.scatter(xs, ys, marker = 'D', color = 'r')
    elif clusterer.assignments[i] == 1:
        plt.scatter(xs, ys, marker = 'o', color = 'g')
    else:
        plt.scatter(xs, ys, marker = '*', color = 'b')
```

그림을 보여주고 싶었는데 작가 분이 코딩을 안 적어놓으셔서... 주먹구구식으로 썼는데 너무... 흑...8ㅸ8

02 k-Means Clustering

Coding 예시



02 k-Means Clustering

k값 선택하기

k값 정하는 방법 매우 다양함!

→ 'k값에 대해 중심점과 각 데이터 포인트 사이의 거리의 제곱합'을 그래프로 그리고, 그래프가 어디서 꺾이는지 관찰함

02 k-Means Clustering

k값 선택하기

squared_clustering_errors

: Inputs 에 대해 k-means 군집화를 실행하고 총 오류의 제곱 값을 계산

```
def squared_clustering_errors(inputs, k):
    clusterer = KMeans(k)
    clusterer.train(inputs)
    means = clusterer.means
    assignments = list(map(clusterer.classify, inputs))

    return sum(squared_distance(input, means[cluster])
               for input, cluster in zip(inputs, assignments))
```

```
inputs = [[-14,-5],[13,13],[20,23],[-19,-11],[-9,-16],[21,27],[-49,15],[26,13],
          [-46,5],[-34,-1],[11,15],[-49,0],[-22,-16],[19,28],[-12,-8],[-13,-19],
          [-41,8],[-11,-6],[-25,-9],[-18,-3]]
```

```
[2, 0, 0, 2, 2, 0, 1, 0, 1, 1, 0, 1, 2, 0, 2, 2, 1, 2, 2, 2]
```

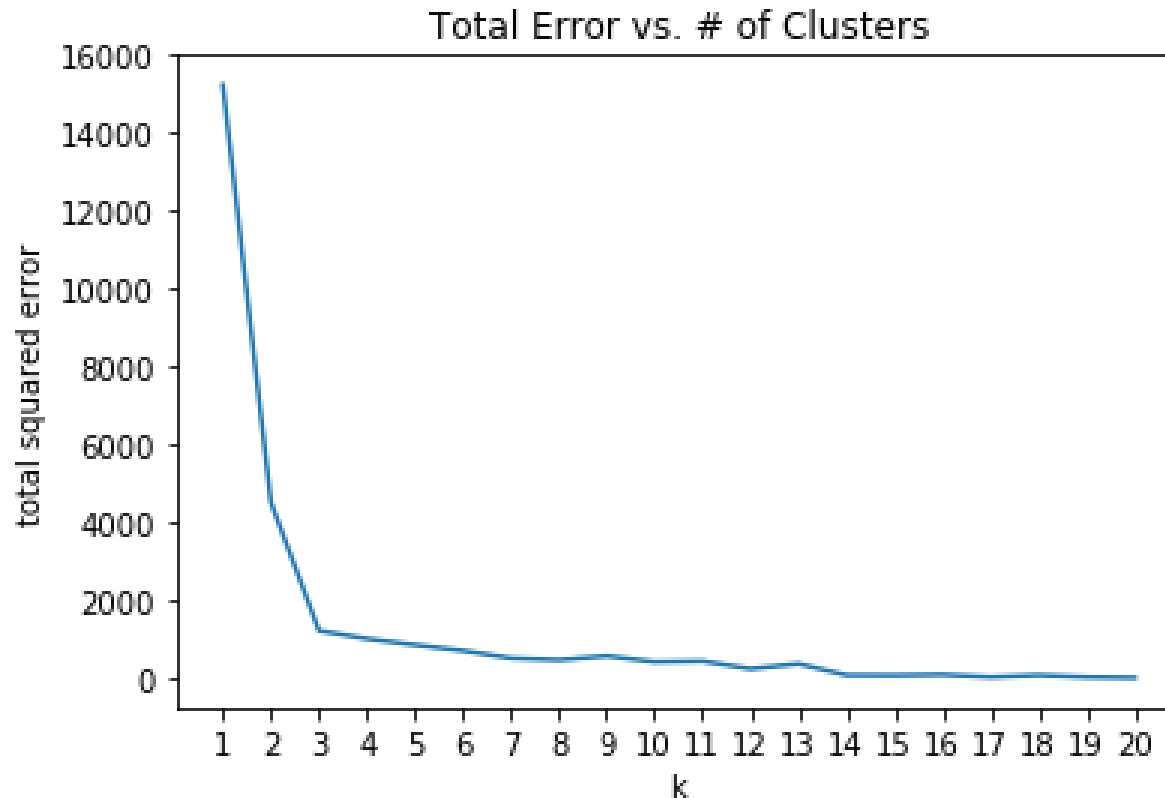
02 k-Means Clustering

k값 선택하기

plot_squared_clustering_errors
: 첫 번째부터 len(inputs) 번째 군집까지 시각화

```
def plot_squared_clustering_errors():  
    ks = range(1, len(inputs) + 1)  
    errors = [squared_clustering_errors(inputs, k) for k in ks]  
  
    plt.plot(ks, errors)  
    plt.xticks(ks)  
    plt.xlabel("k")  
    plt.ylabel("total squared error")  
    plt.title("Total Error vs. # of Clusters")  
    plt.show()
```

02 k-Means Clustering



<http://hamait.tistory.com/740>

03 상향식 군집 계층화

K-Means VS. Hierarchy

사용자가 군집 개수 K 를 명시적으로 지정하는 K-Means 군집화와는 달리 계층적 군집화는 클러스터 수를 지정할 필요가 없다.

장점

- 군집화하는 과정을 시각화하기 용이하다.
- 적절한 수준에서 트리를 자르면 원하는 군집 수로 조정할 수 있다.

단점

- K-Means보다 계산복잡도가 높다.
- 늘 한 쌍의 군집만 비교하기 때문에 2^n 개의 클러스터만 생성할 수 있다.

03 상향식 군집 계층화

상향식 군집 계층화란?

1. 각 데이터 포인트를 하나의 군집으로 간주한다.
→ 최하위 계층의 군집은 데이터를 하나만 갖게 된다.
2. 군집이 두 개 이상이라면,
가장 가까운 두 개의 군집을 찾아 하나의 군집으로 묶는다.



03 상향식 군집 계층화


상향식 군집 계층화 과정

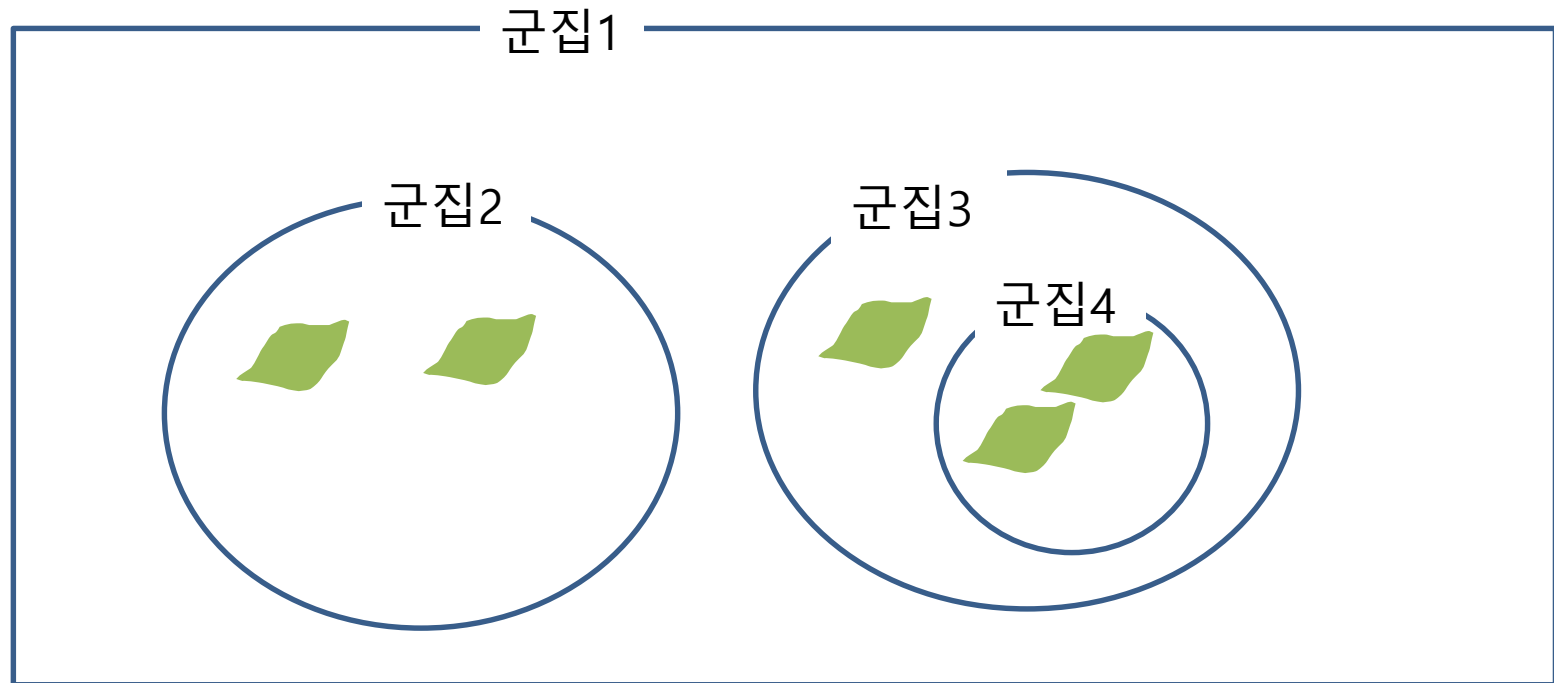
1. 각 데이터 포인트를 하나의 군집으로 간주한다.
 → 최하위 계층의 군집은 데이터를 하나만 갖게 된다.
2. 군집이 두 개 이상이라면,
 가장 가까운 두 개의 군집을 찾아 하나의 군집으로 묶는다.



03 상향식 군집 계층화

상향식 군집 계층화 과정 군집 2, 3은 군집1의 children

 = 잎(leaf) 군집



03 상향식 군집 계층화

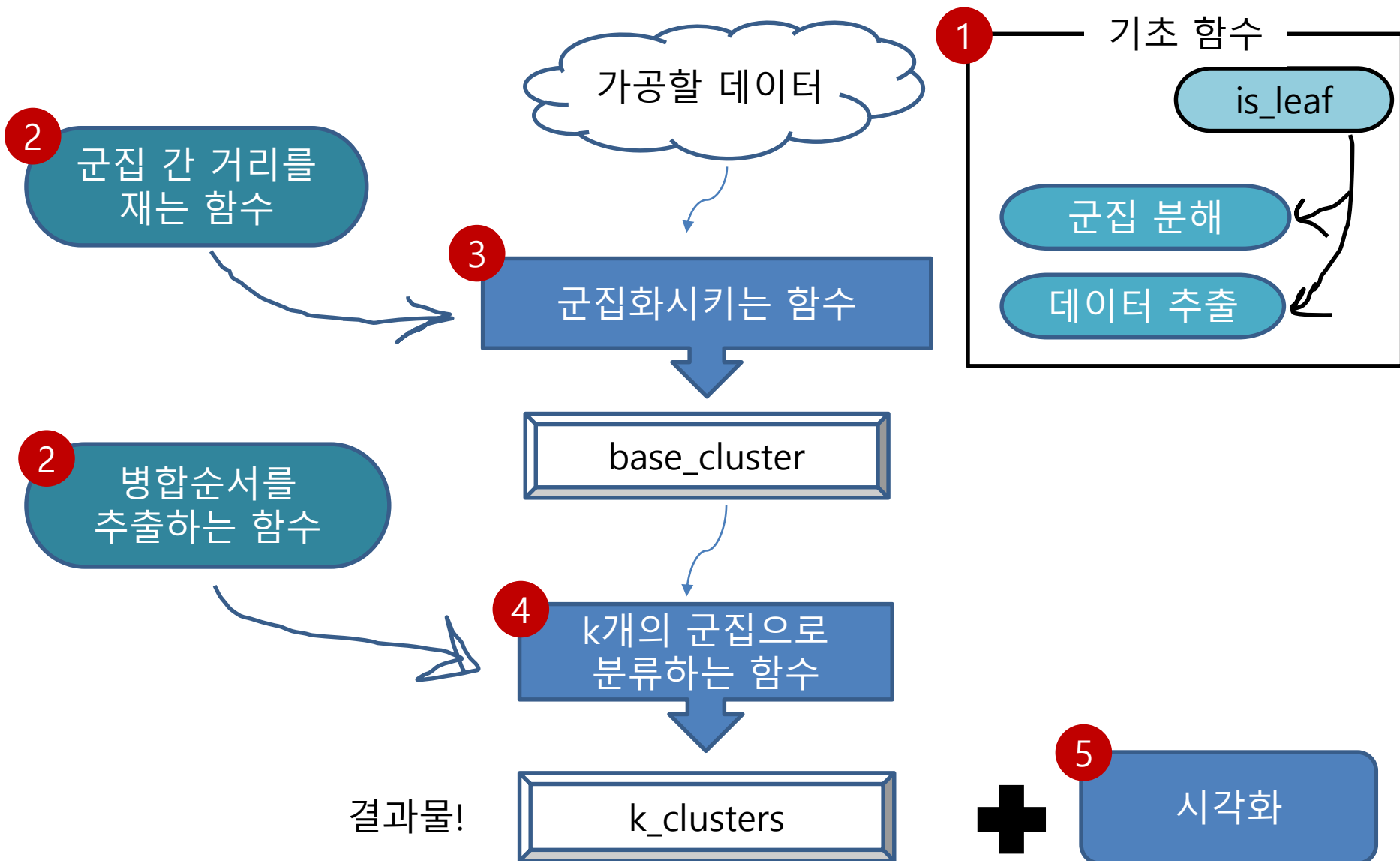
상향식 군집 계층화 과정

leaf1 = ([10,20],)

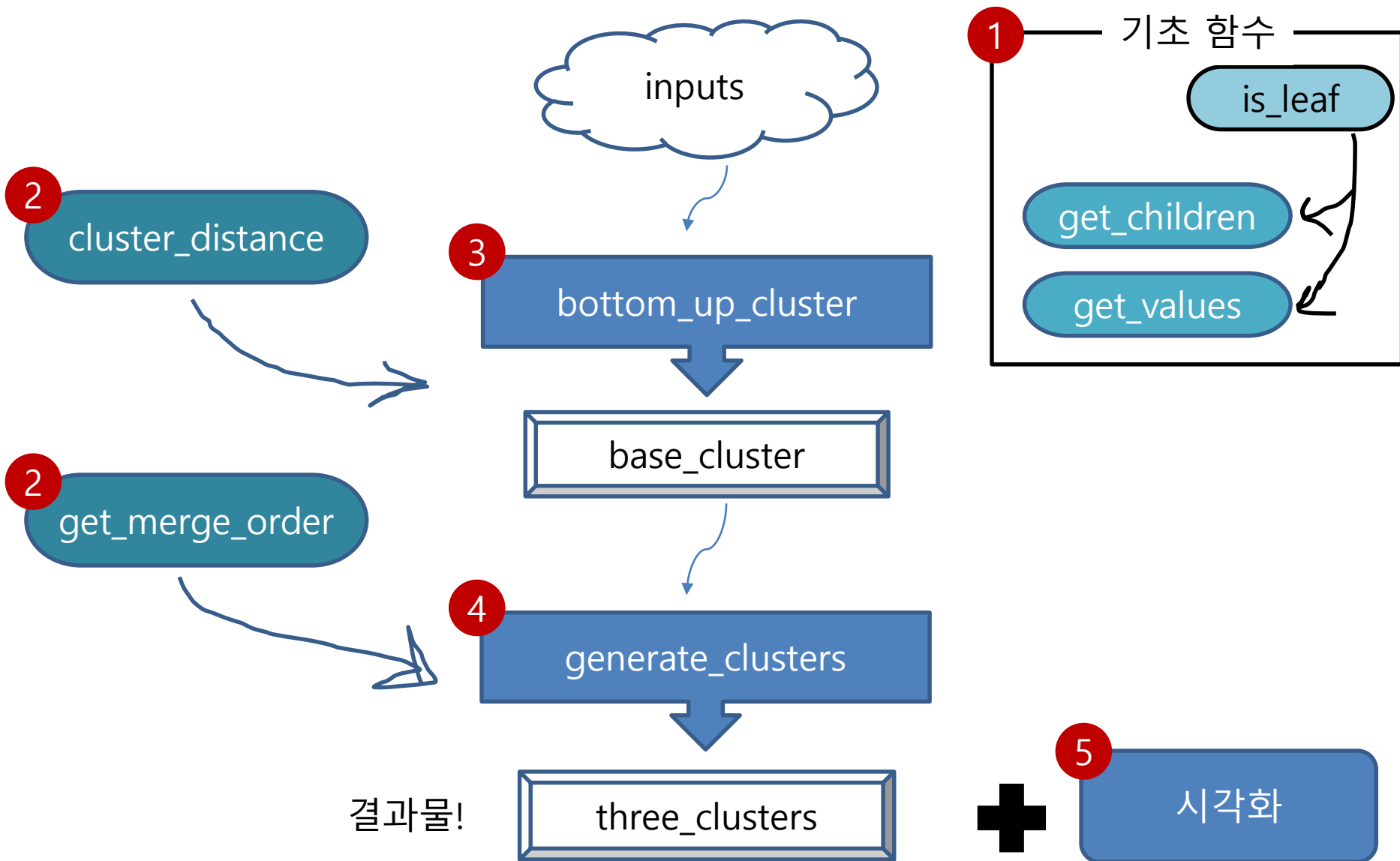
leaf2 = ([30,-15],)

병합된 군집 = (1, [leaf1, leaf2]) #(병합순서, [children])

03 상향식 군집 계층화



03 상향식 군집 계층화



03 상향식 군집 계층화

사전 작업 (내용물 파헤치는 함수)

cluster = (병합순서, [자식들])

def is_leaf(cluster): 잎 함수인지 감별해준다.

"""a cluster is a leaf if it has length 1"""

return len(cluster) == 1

def get_children(cluster):

"""returns the two children of this cluster if it's a merged cluster;
raises an exception if this is a leaf cluster"""

if is_leaf(cluster):

raise TypeError("a leaf cluster has no children")

else:

return cluster[1]

순서가 아닌 군집을 가져와야하므로
[1]로 슬라이스를 해야 한다.

def get_values(cluster):

"""returns the value in this cluster (if it's a leaf cluster)
or all the values in the leaf clusters below it (if it's not)"""

if is_leaf(cluster):

return cluster *# is already a 1-tuple containing value*

else:

return [value

for child **in** get_children(cluster)

for value **in** get_values(child)]

앞군집이 아닐 경우 재귀해서
앞군집이 나올 때까지 찾는다.

03 상향식 군집 계층화

```
(0,
  [(1,
    [(3, [(14, [(18, [(19, 28],), (21, 27],,)], (20, 23],,)], (26, 13],,)],
    (16, [(11, 15],, (13, 13],,)]),
  (2,
    [(4,
      [(5,
        [(9, [(11, [(-49, 0],), (-46, 5],,)], (-41, 8],,)], (-49, 15],,)],
        ([-34, -1],,)],
    (6,
      [(7,
        [(8, [(10, [(-22, -16],,)], (-19, -11],,)], (-25, -9],,)],
        (13,
          [(15, [(17, [(-11, -6],,)], (-12, -8],,)], (-14, -5],,)],
            ([-18, -3],,)]),
      (12, [(13, -19],, (9, -16],,)])))]))
```

03 상향식 군집 계층화

■ 사전 작업 (거리 재는 함수, 병합 순서 알려주는 함수)

```
def cluster_distance(cluster1, cluster2, distance_agg=min):
    """finds the aggregate distance between elements of cluster1
    and elements of cluster2"""
    return distance_agg([distance(input1, input2)
                        for input1 in get_values(cluster1)
                        for input2 in get_values(cluster2)])
    * distance() 는 linear_algebra.py 에 있는 함수이다.

def get_merge_order(cluster):
    if is_leaf(cluster):
        return float('inf')
    else:
        return cluster[0] # merge_order is first element of 2-tuple
```

※ 숫자가 작을수록 나중에 병합된 것!

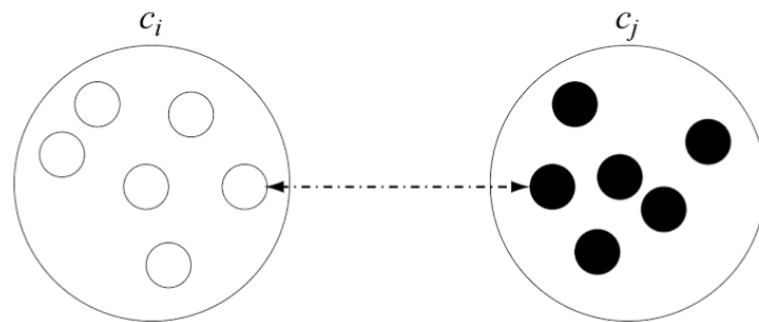
03 상향식 군집 계층화

min vs. MAX

헛갈리지 말아야 할 것은,
어떤 연결법을 썼든 간에 **가장 짧은 숫자를 선택할 것!**

min
단일연결법

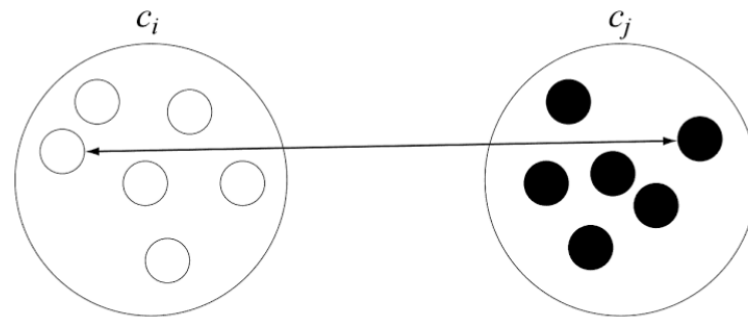
두 군집에 속하는 데이터들의 거리 중에 **가장 짧은 거리**를 군집 간의 거리로 간주한다. 그림 4-3 연결 방법. $d(c_i, c_j)$ 는 클러스터 c_i, c_j 사이의 거리, $d(i, j)$ 는 데이터 i, j 사이의 거리



단일 연결법 $d(c_i, c_j) = \min(d(i, j))$

MAX
완전연결법

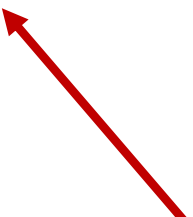
두 군집에 속하는 데이터들의 거리 중에 **가장 먼 거리**를 군집 간의 거리로 간주한다.



완전 연결법 $d(c_i, c_j) = \max(d(i, j))$

03 상향식 군집 계층화

```
inputs = [[-14,-5],[13,13],[20,23],[-19,-11],[-9,-16],[21,27],  
[-49,15],[26,13],[-46,5],[-34,-1],[11,15],[-49,0],[-22,-16],  
[19,28],[-12,-8],[-13,-19],[-41,8],[-11,-6],[-25,-9],[-18,-3]]
```



최하위 계층 데이터
(앞군집)

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]

    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
        c1, c2 = min([(cluster1, cluster2)
                       for i, cluster1 in enumerate(clusters)
                       for cluster2 in clusters[:i]],
                      key=lambda p: cluster_distance(p[0], p[1], distance_agg))

        # remove them from the list of clusters
        clusters = [c for c in clusters if c != c1 and c != c2]

        # merge them, using merge_order = # of clusters left
        merged_cluster = (len(clusters), [c1, c2])

        # and add their merge
        clusters.append(merged_cluster)

    # when there's only one cluster left, return it
    return clusters[0]
```

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]
```

```
>>> clusters
[([-14, -5],), ([13, 13],), ([20, 23],), ([-19, -11],), ([-9, -16],), ([21, 27],), ([-49, 15],),
 ([26, 13],), ([-46, 5],), ([-34, -1],), ([11, 15],), ([-49, 0],), ([-22, -16],), ([19, 28],),
 ([-12, -8],), ([-13, -19],), ([-41, 8],), ([-11, -6],), ([-25, -9],), ([-18, -3],)]
```

```
# as long as we have more than one cluster left...
```

```
while len(clusters) > 1:
```

```
    # find the two closest clusters
```

```
    c1, c2 = min([(cluster1, cluster2)
```

```
                    for i, cluster1 in enumerate(clusters)
```

```
                    for cluster2 in clusters[:i]],
```

```
                    key=lambda p: cluster_distance(p[0], p[1], distance_agg))
```

```
>>> c1, c2
([19, 28],), ([21, 27],)
```

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]

    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
        c1, c2 = min([(cluster1, cluster2)
                      for i, cluster1 in enumerate(clusters)
                      for cluster2 in clusters[:i]],
                     key=lambda p: cluster_distance(p[0], p[1], distance_agg))
```

```
>>> [(i, cluster1) for i, cluster1 in enumerate(clusters)]
[(0, ([-14, -5],)), (1, ([13, 13],)), (2, ([20, 23],)), (3, ([-19, -11],)), (4, ([-9, -16],)),
 (5, ([-49, 15],)), (6, ([26, 13],)), (7, ([-46, 5],)), (8, ([-34, -1],)), (9, ([11, 15],)),
 (10, ([-49, 0],)), (11, ([-22, -16],)), (12, ([-12, -8],)), (13, ([-13, -19],)), (14, ([-41,
 8],)), (15, ([-11, -6],)), (16, ([-25, -9],)), (17, ([-18, -3],))]
```

```
>>> [(cluster1, cluster2) for i, cluster1 in enumerate(clusters) for cluster2 in clusters[:i]]
[(([13, 13],), ([-14, -5],)), (([20, 23],), ([-14, -5],)), (([20, 23],), ([13, 13],)), (([-19,
-11],), ([-14, -5],)), (([-19, -11],), ([13, 13],)), (([-19, -11],), ([20, 23],)), (([-9, -16],),
([-14, -5],)), (([-9, -16],), ([13, 13],)), (([-9, -16],), ([20, 23],)), (([-9, -16],),
([-19, -11],)), (([-49, 15],), ([-14, -5],)), (([-49, 15],), ([13, 13],)), (([-49, 15],),
([-19, -11],)), (([-49, 15],), ([20, 23],)), (([-49, 15],), ([11, 15],)), (([-49, 15],),
([-41, 8],)), (([-49, 15],), ([-11, -6],)), (([-49, 15],), ([-25, -9],)), (([-49, 15],),
([-18, -3],)), (([26, 13],), ([-14, -5],)), (([26, 13],), ([13, 13],)), (([26, 13],), ([20, 23],)),
([26, 13],), ([-19, -11],)), ([26, 13],), ([-9, -16],)), ([26, 13],), ([11, 15],)), ([26, 13],),
([-41, 8],)), ([26, 13],), ([-11, -6],)), ([26, 13],), ([-25, -9],)), ([26, 13],), ([-18, -3],)),
([20, 23],), ([-14, -5],)), ([20, 23],), ([13, 13],)), ([20, 23],), ([-19, -11],)), ([20, 23],),
([-9, -16],)), ([20, 23],), ([11, 15],)), ([20, 23],), ([-41, 8],)), ([20, 23],), ([-11, -6],)),
([20, 23],), ([-25, -9],)), ([20, 23],), ([-18, -3],)), ([-19, -11],), ([-14, -5],)), ([-19, -11],),
([13, 13],)), ([-19, -11],), ([20, 23],)), ([-19, -11],), ([-9, -16],)), ([-19, -11],), ([11, 15],)),
([-19, -11],), ([-41, 8],)), ([-19, -11],), ([-11, -6],)), ([-19, -11],), ([-25, -9],)), ([-19, -11],),
([-18, -3],)), ([-9, -16],), ([-14, -5],)), ([-9, -16],), ([13, 13],)), ([-9, -16],), ([20, 23],)),
([-9, -16],), ([11, 15],)), ([-9, -16],), ([-41, 8],)), ([-9, -16],), ([-11, -6],)), ([-9, -16],),
([-25, -9],)), ([-9, -16],), ([-18, -3],)), ([-49, 15],), ([-14, -5],)), ([-49, 15],), ([13, 13],)),
([-49, 15],), ([-19, -11],)), ([-49, 15],), ([-9, -16],)), ([-49, 15],), ([11, 15],)), ([-49, 15],),
([-41, 8],)), ([-49, 15],), ([-11, -6],)), ([-49, 15],), ([-25, -9],)), ([-49, 15],), ([-18, -3],)),
([13, 13],), ([-14, -5],)), ([13, 13],), ([20, 23],)), ([13, 13],), ([-19, -11],)), ([13, 13],),
([-9, -16],)), ([13, 13],), ([11, 15],)), ([13, 13],), ([-41, 8],)), ([13, 13],), ([-11, -6],)),
([13, 13],), ([-25, -9],)), ([13, 13],), ([-18, -3],)), ([20, 23],), ([-19, -11],)), ([20, 23],),
([-9, -16],)), ([20, 23],), ([11, 15],)), ([20, 23],), ([-41, 8],)), ([20, 23],), ([-11, -6],)),
([20, 23],), ([-25, -9],)), ([20, 23],), ([-18, -3],)), ([-14, -5],), ([-19, -11],)), ([-14, -5],),
([-9, -16],)), ([-14, -5],), ([11, 15],)), ([-14, -5],), ([-41, 8],)), ([-14, -5],), ([-11, -6],)),
([-14, -5],), ([-25, -9],)), ([-14, -5],), ([-18, -3],)), ([-19, -11],), ([-49, 15],)), ([-19, -11],),
([-14, -5],)), ([-19, -11],), ([20, 23],)), ([-19, -11],), ([-41, 8],)), ([-19, -11],), ([-11, -6],)),
([-19, -11],), ([-25, -9],)), ([-19, -11],), ([-18, -3],)), ([-9, -16],), ([-49, 15],)), ([-9, -16],),
([-14, -5],)), ([-9, -16],), ([20, 23],)), ([-9, -16],), ([-41, 8],)), ([-9, -16],), ([-11, -6],)),
([-9, -16],), ([-25, -9],)), ([-9, -16],), ([-18, -3],)), ([-49, 15],), ([-41, 8],)), ([-49, 15],),
([-11, -6],)), ([-49, 15],), ([-25, -9],)), ([-49, 15],), ([-18, -3],)), ([-41, 8],), ([-11, -6],)),
([-41, 8],), ([-25, -9],)), ([-41, 8],), ([-18, -3],)), ([-11, -6],), ([-25, -9],)), ([-11, -6],),
([-18, -3],)), ([-25, -9],), ([-18, -3],))]
```

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]

    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
```

```
>>> clusters
[([-14, -5],), ([13, 13],), ([20, 23],), ([-19, -11],), ([-9, -16],), ([21, 27],), ([-49, 15],), ([26, 13],), ([-46, 5],), ([-34, -1],), ([11, 15],), ([-49, 0],), ([-22, -16],), ([19, 28],), ([-12, -8],), ([-13, -19],), ([-41, 8],), ([-11, -6],), ([-25, -9],), ([-18, -3],)]
```

```
# remove them from the list of clusters
clusters = [c for c in clusters if c != c1 and c != c2]
```

```
>>> c1, c2
(([-19, -11],), ([21, 27],))
```

해당 앞군집을 삭제

```
>>> clusters
[([-14, -5],), ([13, 13],), ([20, 23],), ([-19, -11],), ([-9, -16],), ([21, 27],), ([-49, 15],), ([26, 13],), ([-46, 5],), ([-34, -1],), ([11, 15],), ([-49, 0],), ([-22, -16],), ([19, 28],), ([-12, -8],), ([-13, -19],), ([-41, 8],), ([-11, -6],), ([-25, -9],), ([-18, -3],)]
```

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]

    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
        c1, c2 = min([(cluster1, cluster2)
                       for i, cluster1 in enumerate(clusters)
                       for cluster2 in clusters[:i]],
                      key=lambda p: cluster_distance(p[0], p[1], distance_agg))

        # remove them from the list of clusters
        clusters = [c for c in clusters if c != c1 and c != c2]

        # merge them, using merge_order = # of clusters left
        merged_cluster = (len(clusters), [c1, c2])

    >>> merged_cluster
    (18, (((19, 28),), ([21, 27],)))
```

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
# merge them, using merge_order = # of clusters left
merged_cluster = (len(clusters), [c1, c2])

# and add their merge
clusters.append(merged_cluster)

# when there's only one cluster left, return it
return clusters[0]
```

```
>>> clusters
[([[-14, -5],), ([13, 13],), ([20, 23],), ([-19, -11],), ([-9, -16],), ([-49, 15],), ([26, 13],),
, ([-46, 5],), ([-34, -1],), ([11, 15],), ([-49, 0],), ([-22, -16],), ([-12, -8],), ([-13, -19],),
, ([-41, 8],), ([-11, -6],), ([-25, -9],), ([-18, -3],), (18, [[([19, 28],), ([21, 27],),
1]])]
```


03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]

    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
        c1, c2 = min([(cluster1, cluster2)
                      for i, cluster1 in enumerate(clusters)
                      for cluster2 in clusters[i:]],
                     key=lambda p: cluster_distance(p[0], p[1], distance_agg))

        # remove them from the list of clusters
        clusters = [c for c in clusters if c != c1 and c != c2]

        # merge them, using merge_order = # of clusters left
        merged_cluster = (len(clusters), [c1, c2])

        # and add their merge
        clusters.append(merged_cluster)

    # when there's only one cluster left, return it
    return clusters[0]
```

어쨌든 가장 거리가 작은 값으로 !!

Merged_cluster도 계산이 가능한가?

03 상향식 군집 계층화

본격적인 알고리즘 만들기

```
def cluster_distance(cluster1, cluster2, distance_agg=min):  
    """finds the aggregate distance between elements of cluster1  
    and elements of cluster2"""  
    return distance_agg([distance(input1, input2)  
                        for input1 in get_values(cluster1)  
                        for input2 in get_values(cluster2)])
```

```
>>> cluster_distance((17, [([-11, -6],), ([-12, -8],)]), ([13, 13],))  
30.610455730027933
```

* merged_cluster도 거리 계산이 가능하다 !!

03 상향식 군집 계층화

min vs. MAX

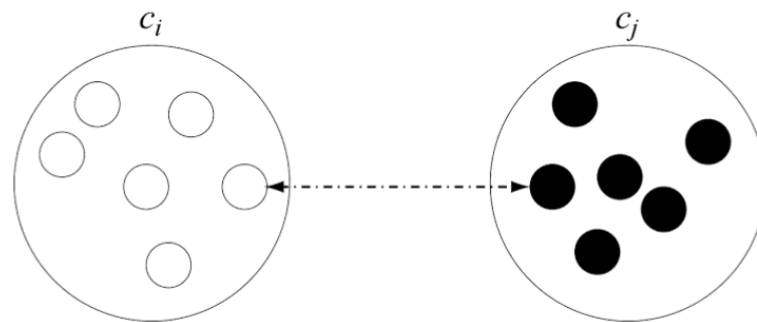
min
단일연결법

distance_agg

MAX

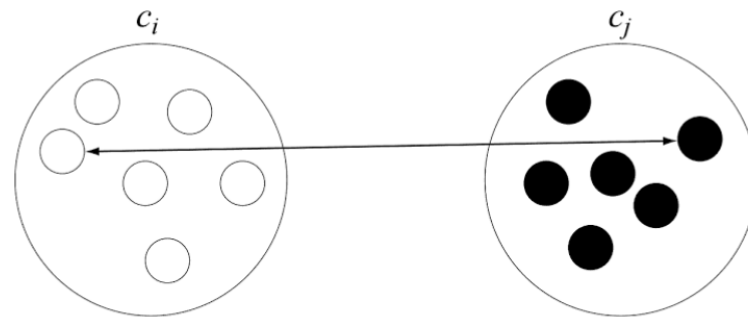
완전연결법

두 군집에 속하는 데이터들의 거리 중에 **가장 짧은 거리**를 군집 간의 거리로 간주한다. 그림 4-3 연결 방법. $d(c_i, c_j)$ 는 클러스터 c_i, c_j 사이의 거리, $d(i, j)$ 는 데이터 i, j 사이의 거리



단일 연결법 $d(c_i, c_j) = \min(d(i, j))$

두 군집에 속하는 데이터들의 거리 중에 **가장 먼 거리**를 군집 간의 거리로 간주한다.



완전 연결법 $d(c_i, c_j) = \max(d(i, j))$

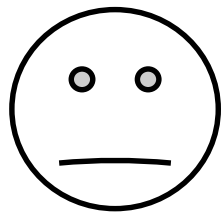
03 상향식 군집 계층화

```
In [21]: bottom_up_cluster(inputs)
```

```
Out [21]: (0,
  [(1,
    [(3, [(14, [(18, [(19, 28),], ([21, 27],)], ([20, 23],)], ([26, 13],)],
      (16, [(11, 15),], ([13, 13],)]))],
    (2,
      [(4,
        [(5,
          [(9, [(11, [(-49, 0),], ([-46, 5],)], ([-41, 8],)], ([-49, 15],)],
            ([-34, -1],)]),
          (6,
            [(7,
              [(8, [(10, [(-22, -16),], ([-19, -11],)], ([-25, -9],)],
                (13,
                  [(15, [(17, [(-11, -6),], ([-12, -8],)], ([-14, -5],)],
                    ([-18, -3],)]))],
                  (12, [(-13, -19),], ([-9, -16],)])))])))]))]
```

```
In [23]: base_cluster = bottom_up_cluster(inputs)
```

03 상향식 군집 계층화



그럼 특정 개수의 군집으로 나누려면 어떻게?

```
def generate_clusters(base_cluster, num_clusters):
    # start with a list with just the base cluster
    clusters = [base_cluster]

    # as long as we don't have enough clusters yet...
    while len(clusters) < num_clusters:
        # choose the last-merged of our clusters
        next_cluster = min(clusters, key=get_merge_order)
        # remove it from the list
        clusters = [c for c in clusters if c != next_cluster]
        # and add its children to the list (i.e., unmerge it)
        clusters.extend(get_children(next_cluster))

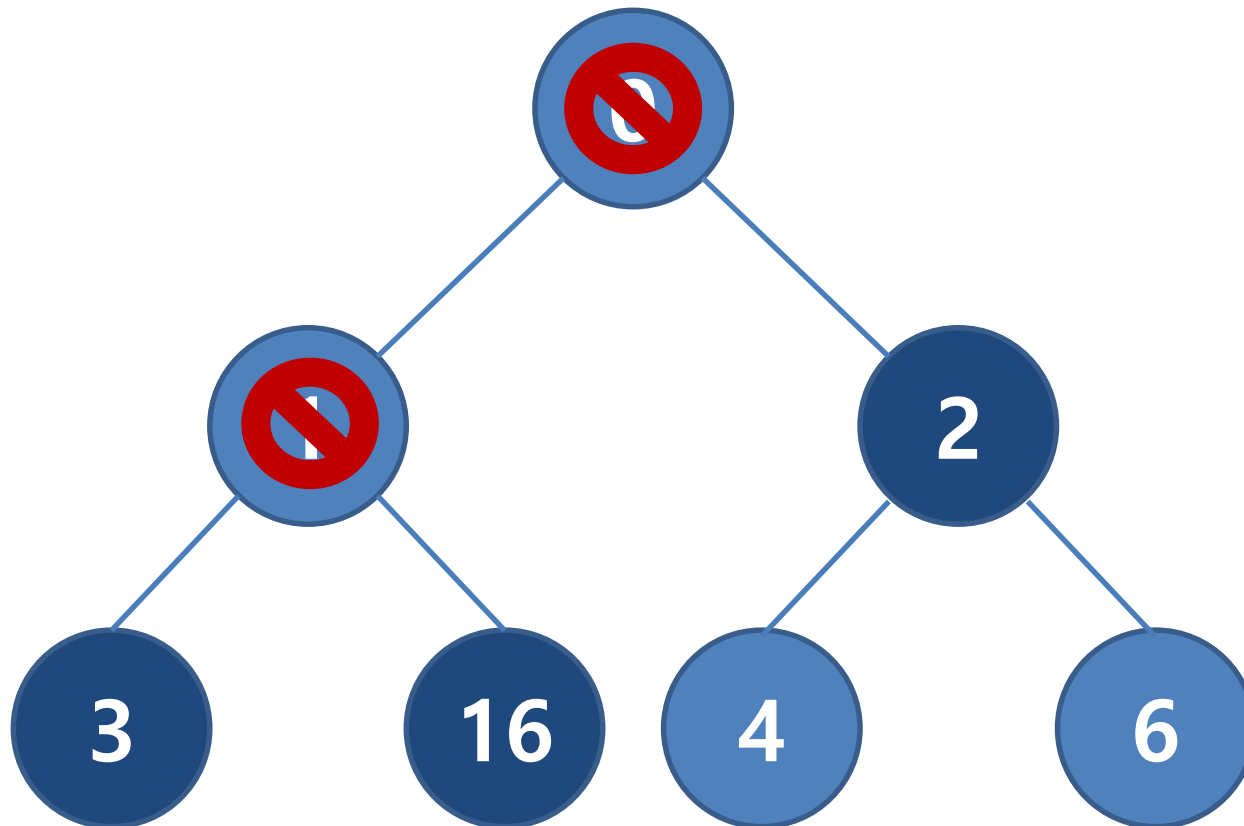
    # once we have enough clusters...
    return clusters
```

03 상향식 군집 계층화

```
In [8]: generate_clusters(base_cluster, 3)
```

```
Out [8]: [(2,
  [(4,
    [(5, [(9, [(11, [([-49, 0],), ([-46, 5],)], ([-41, 8],)], ([-49, 15],)],
      ([-34, -1],)]),
    (6,
      [(7,
        [(8, [(10, [([-22, -16],), ([-19, -11],)], ([-25, -9],)]),
          (13,
            [(15, [(17, [([-11, -6],), ([-12, -8],)], ([-14, -5],)],
              ([-18, -3],)])]),
          (12, [([-13, -19],), ([-9, -16],)])))]),
    (3, [(14, [(18, [(19, 28],), (21, 27],)], (20, 23],)], (26, 13],)]),
    (16) [([11, 15],), ([13, 13],)])]
```

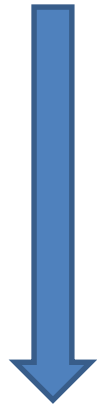
03 상향식 군집 계층화



03 상향식 군집 계층화

```
def generate_clusters(base_cluster, num_clusters):
    # start with a list with just the base cluster
    clusters = [base_cluster]

    # as long as we don't have enough clusters yet...
    while len(clusters) < num_clusters:
        # choose the last-merged of our clusters
        next_cluster = min(clusters, key=get_merge_order)
```



```
def get_merge_order(cluster):
    if is_leaf(cluster):
        return float('inf')
    else:
        return cluster[0] # merge_order is first element of 2-tuple
```

- 앞군집은 '무한'으로 취급되어서 무조건 가장 큰 수가 된다.
- 가장 마지막에 군집화 된 군집부터 순번이 골라진다.

03 상향식 군집 계층화

```
def generate_clusters(base_cluster, num_clusters):  
    # start with a list with just the base cluster  
    clusters = [base_cluster]  
  
    # as long as we don't have enough clusters yet...  
    while len(clusters) < num_clusters:  
        # choose the last-merged of our clusters  
        next_cluster = min(clusters, key=get_merge_order)  
        # remove it from the list  
        clusters = [c for c in clusters if c != next_cluster]
```

아까 상향식 군집 계층화를 하던 함수처럼,
next_cluster로 선택된 군집은 cluster에서 제외된다.

03 상향식 군집 계층화

```
def generate_clusters(base_cluster, num_clusters):  
    # start with a list with just the base cluster  
    clusters = [base_cluster]  
  
    # as long as we don't have enough clusters yet...  
    while len(clusters) < num_clusters:  
        # choose the last-merged of our clusters  
        next_cluster = min(clusters, key=get_merge_order)  
        # remove it from the list  
        clusters = [c for c in clusters if c != next_cluster]  
        # and add its children to the list (i.e., unmerge it)  
        clusters.extend(get_children(next_cluster))
```



해당 군집에 있던 자식들(children)을 cluster에 포함시킨다. (next_cluster의 군집을 해제한다.)

03 상향식 군집 계층화

```
def generate_clusters(base_cluster, num_clusters):  
    # start with a list with just the base cluster  
    clusters = [base_cluster]  
  
    # as long as we don't have enough clusters yet...  
    while len(clusters) < num_clusters:  
        # choose the last-merged of our clusters  
        next_cluster = min(clusters, key=get_merge_order)  
        # remove it from the list  
        clusters = [c for c in clusters if c != next_cluster]  
        # and add its children to the list (i.e., unmerge it)  
        clusters.extend(get_children(next_cluster))  
  
    # once we have enough clusters...  
    return clusters
```

03 상향식 군집 계층화

In [8]: `generate_clusters(base_cluster, 3)`

Out [8]:

```

((2,
  [(4,
    [(5, [(9, [(11, [([-49, 0],), ([-46, 5],)], ([-41, 8],)], ([-49, 15],)],
      ([-34, -1],)]),
    (6,
      [(7,
        [(8, [(10, [([-22, -16],), ([-19, -11],)], ([-25, -9],)]),
          (13,
            [(15, [(17, [([-11, -6],), ([-12, -8],)], ([-14, -5],)],
              ([-18, -3],)])]),
            (12, [([-13, -19],), ([-9, -16],)])))]),
    (3, [(14, [(18, [(19, 28],), (21, 27],)], (20, 23],)], (26, 13],)]),
    (16) [([11, 15],), ([13, 13],)])])

```

03 상향식 군집 계층화

```
In [26]: three_clusters = [get_values(cluster) for cluster in generate_clusters(base_cluster,3)]
```

```
In [27]: three_clusters
```

```
Out [27]: [[[-49, 0],  
            [-46, 5],  
            [-41, 8],  
            [-49, 15],  
            [-34, -1],  
            [-22, -16],  
            [-19, -11],  
            [-25, -9],  
            [-11, -6],  
            [-12, -8],  
            [-14, -5],  
            [-18, -3],  
            [-13, -19],  
            [-9, -16]],  
           [[19, 28], [21, 27], [20, 23], [26, 13]],  
           [[11, 15], [13, 13]]]
```

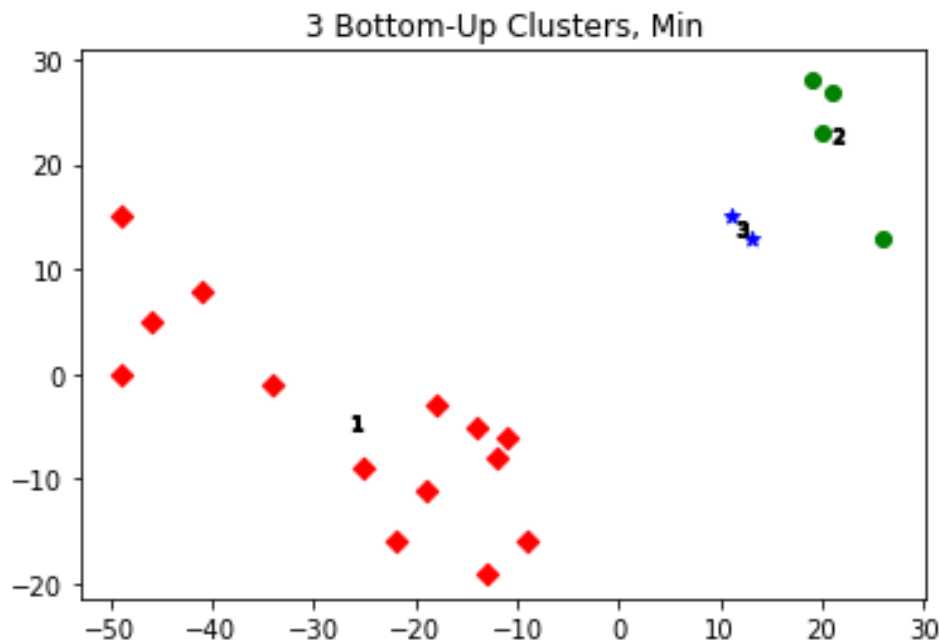
03 상향식 군집 계층화

시각화하기

```
In [30]: for i, cluster, marker, color in zip([1,2,3], three_clusters, ['D','o','*'], ['r','g','b']):
          xs, ys = zip(*cluster)
          plt.scatter(xs, ys, color=color, marker=marker)

          x, y = vector_mean(cluster)
          plt.plot(x, y, marker='$'+str(i)+'$', color='black')

plt.title("3 Bottom-Up Clusters, Min")
plt.show()
```



03 상향식 군집 계층화

min vs. MAX

이번엔 Max로 해볼까요?

```
base_cluster_max = bottom_up_cluster(inputs, max)
three_clusters_max = [get_values(cluster) for cluster in generate_clusters(base_cluster_max, 3)]

for i, cluster, marker, color in zip([1, 2, 3], three_clusters_max, ['D', 'o', '*'], ['r', 'g', 'b']):
    xs, ys = zip(*cluster)
    plt.scatter(xs, ys, color=color, marker=marker)

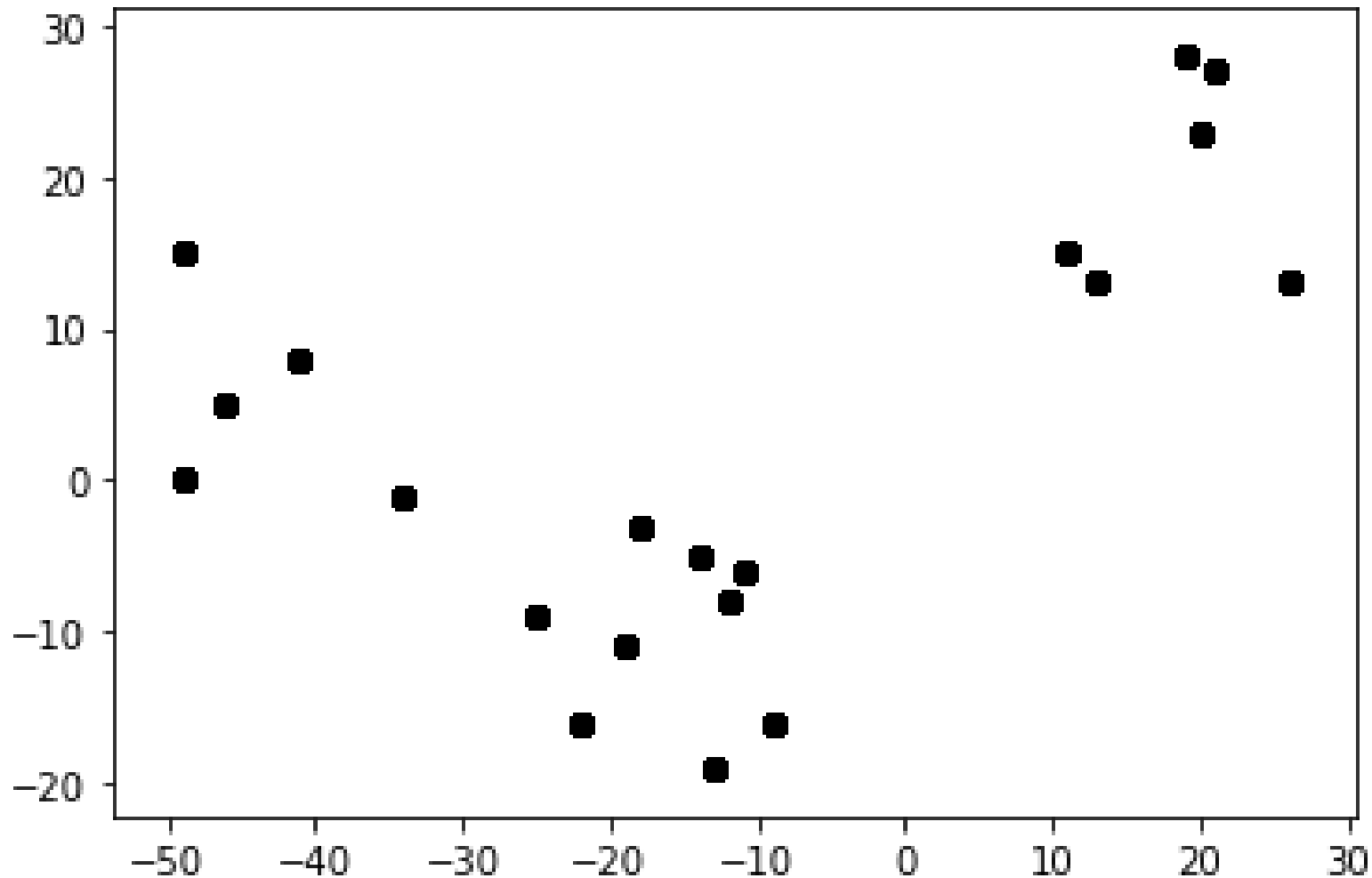
    x, y = vector_mean(cluster)
    plt.plot(x, y, marker='$'+str(i)+'$', color='black')

plt.title("3 Bottom-Up Clusters, Max")
plt.show()
```

03 상향식 군집 계층화

min vs. MAX

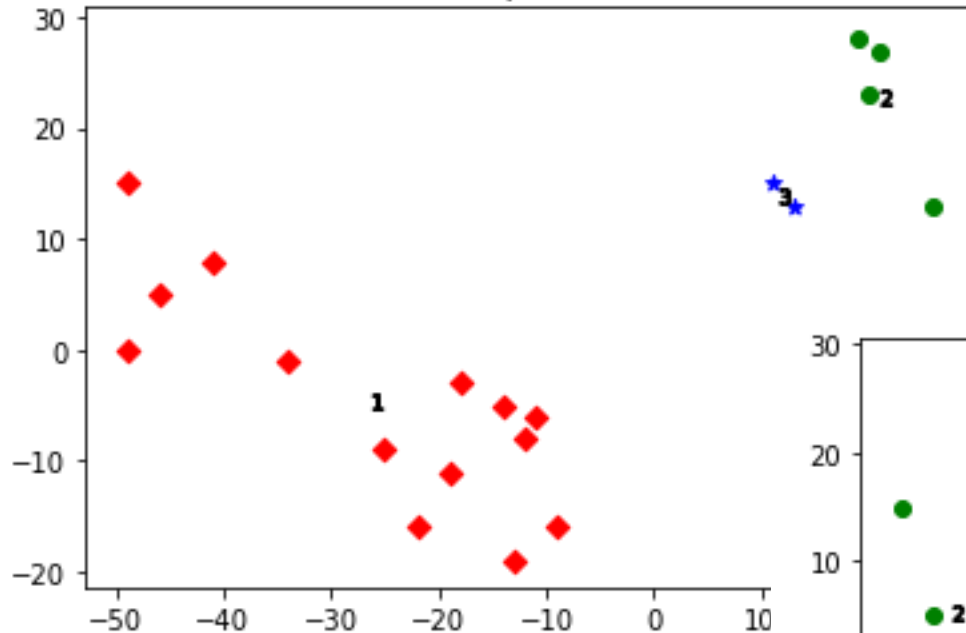
All Inputs



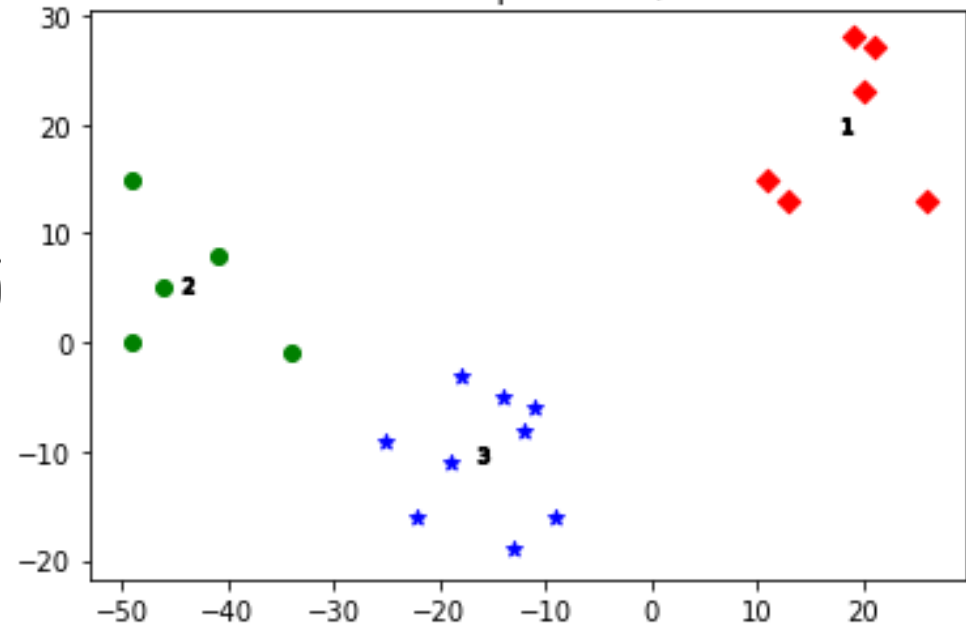
03 상향식 군집 계층화

min vs. MAX

3 Bottom-Up Clusters, Min



3 Bottom-Up Clusters, Max



O4 Scikit-Learn Cluster

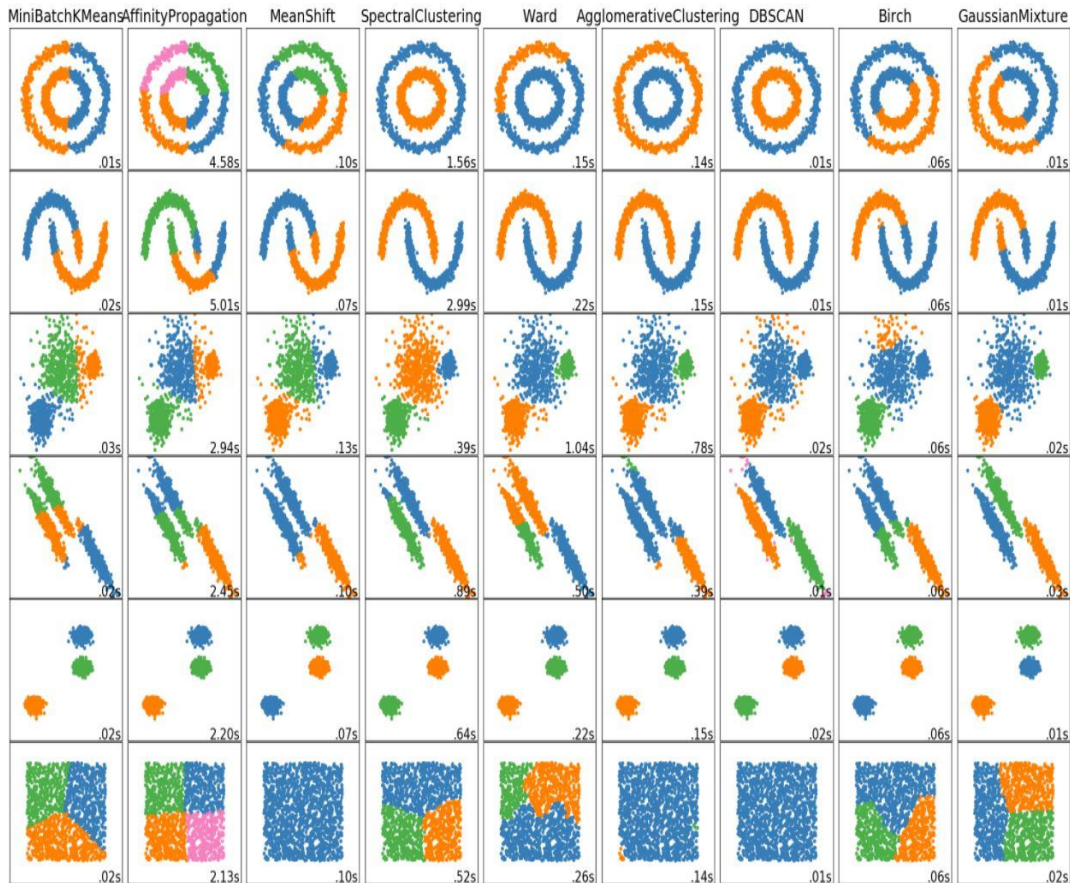
Overview

1. Scikit-learn 에서 **sklearn.cluster** 모듈에서 다양한 군집화 알고리즘 간단한 소개
2. sklearn.cluster에서 K-Means 모듈 설명
 - Parameters
 - K-Means++

O4 Scikit-Learn Cluster

sklearn.cluster 모듈의 군집화 알고리즘 소개

2.3.1. Overview of clustering methods



→ 알고리즘에 따라 결과 다름

→ 데이터에 적절한 알고리즘 선택의 필요성

O4 Scikit-Learn Cluster

sklearn.cluster

A comparison of the clustering algorithms in scikit-learn

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large <code>n_samples</code> , medium <code>n_clusters</code> with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium <code>n_samples</code> , small <code>n_clusters</code>	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large <code>n_samples</code> , medium <code>n_clusters</code>	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large <code>n_clusters</code> and <code>n_samples</code>	Large dataset, outlier removal, data reduction.	Euclidean distance between points

O4 Scikit-Learn Cluster

sklearn.cluster 모듈의 다양한 군집화 알고리즘

<http://scikit-learn.org/stable/modules/clustering.html>

1. K-Means
 - 1-1. K-Means Minibatch
2. Affinity Propagation
3. Spectral Clustering
4. Hierarchical Clustering
 - 4-1. Agglomerative Clustering
5. DBSCAN

O4 Scikit-Learn Cluster

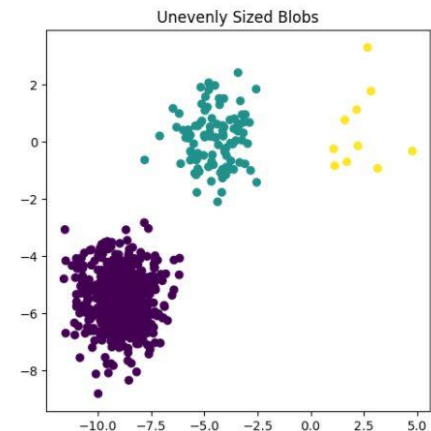
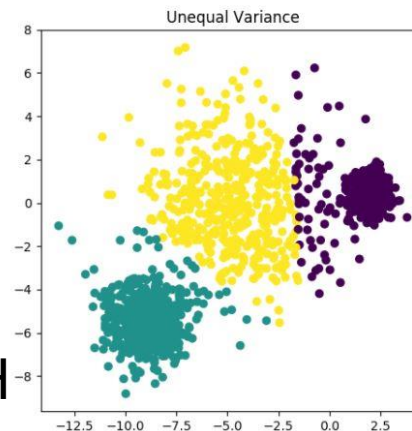
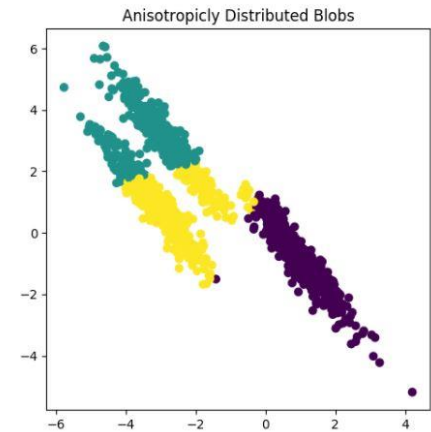
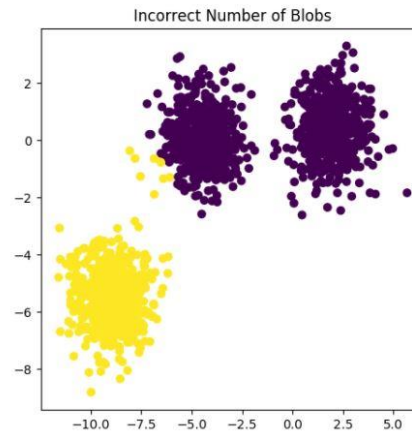
K-Means in Scikit-learn

특징

1. 샘플의 크기가 클 때도 잘 작동함
2. 각 군집이 같은 분산을 가지도록 함
3. 각 군집의 SSE를 최소화함

주의사항

1. 군집의 개수를 잘 지정해야 함
2. 군집들이 convex & isotropic
3. 군집들이 비슷한 분산을 가져야 함
4. 군집들의 크기가 비슷해야함
5. local optimum으로 빠질 가능성 존재



O4 Scikit-Learn Cluster

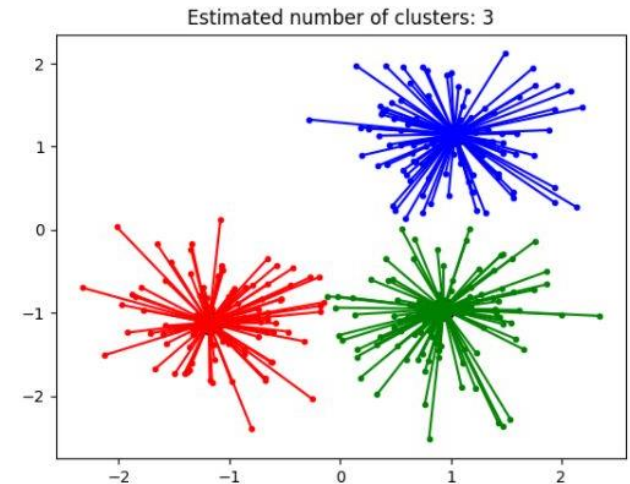
Affinity Propagation in Scikit-learn

데이터 사이에 message passing을 실시해 군집화
다른 데이터들을 가장 잘 대표하는 데이터를 찾음

장점: k-mean와 다르게 군집 개수 정할 필요 x

단점: Time complexity $O(N^2T)$

Memory complexity $O(N^2)$



O4 Scikit-Learn Cluster

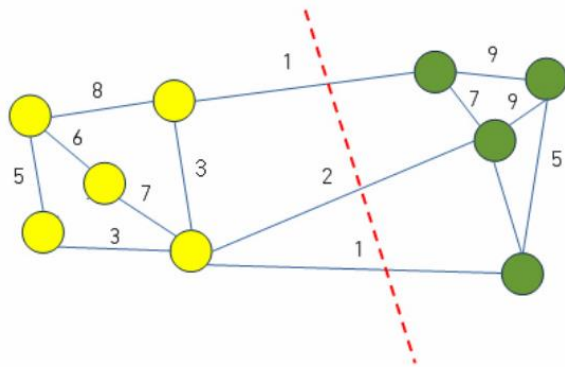
Spectral Clustering in Scikit-learn

데이터의 상대적인 위치를 바탕으로 affinity matrix를 구함

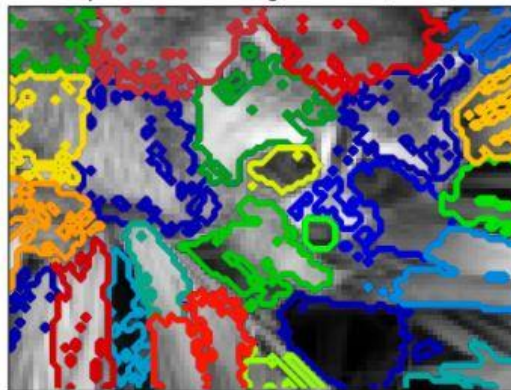
Minimum Cut, Normalized Cut 등으로 군집 설정

Non-convex / Non-flat geometry

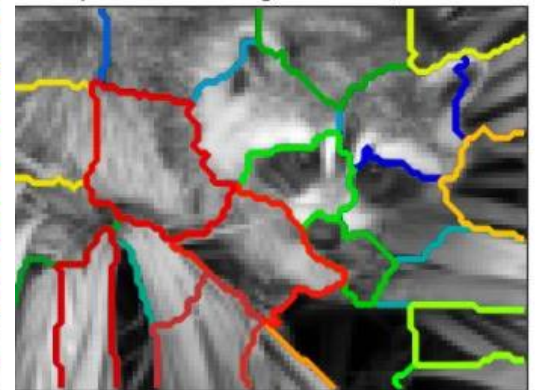
군집이 measure of the center + spread을 통해 설명 되지 않을 때 적합



Spectral clustering: kmeans, 6.60s



Spectral clustering: discretize, 5.69s



O4 Scikit-Learn Cluster

Hierarchical Clustering in Scikit-learn

계층 군집화

중첩된 클러스터를 합치거나 분할하는 일을 하는 알고리즘을 묶어 말함
앞서 말한 상향식 계층 군집화 (Agglomerative Clustering)이 대표적임

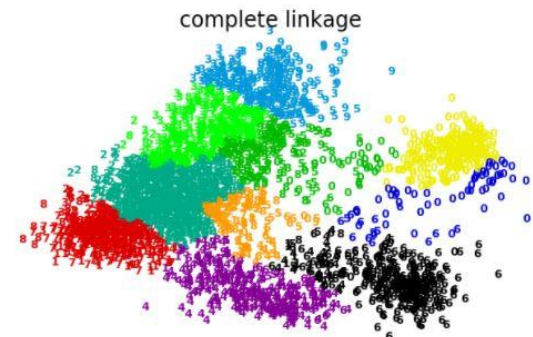
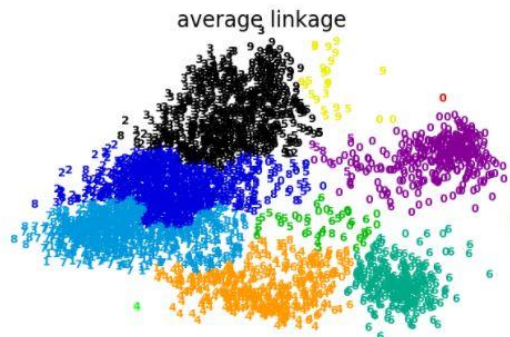
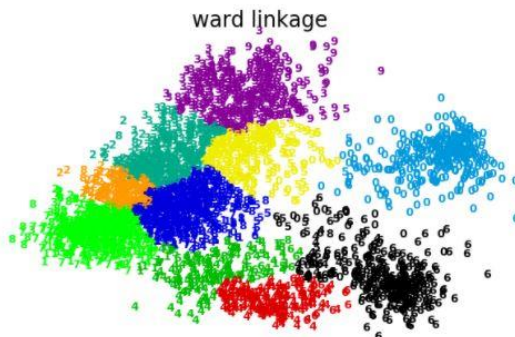
군집 방식

Ward linkage - 모든 군집의 SSE 최소화

Complete linkage - 군집 사이의 최대거리 최소화

Average linkage - 군집사이의 평균거리 최소화

AgglomerativeClustering supports Ward, average, and complete linkage strategies.



O4 Scikit-Learn Cluster

Agglomerative Clustering in Scikit-learn

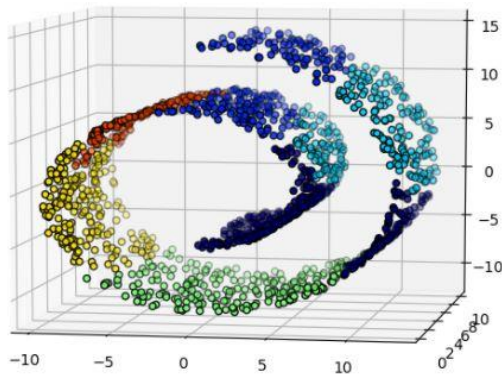
Adding connectivity constraints

→ 인접한 데이터의 구조를 미리 정의할 수 있음

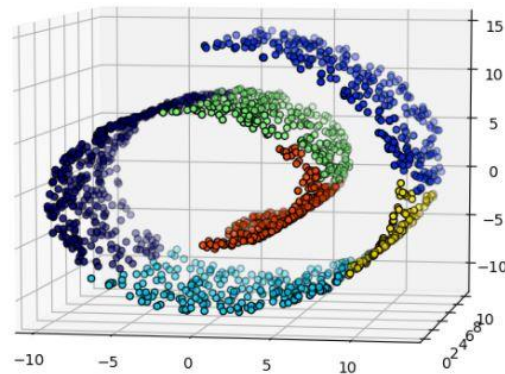
데이터가 특정한 모양을 지닐 때 유용함

알고리즘을 더 빠르게 만듦 (특히 데이터가 클 때)

Without connectivity constraints (time 0.06s)



With connectivity constraints (time 0.15s)



O4 Scikit-Learn Cluster

DBSCAN in Scikit-learn

DBSCAN(Density-based spatial clustering of applications with noise)

밀도에 따른 군집화 알고리즘 <-> K-Means(거리)

핵심벡터(core points): 일정 수준 이상의 밀도를 가지는 데이터

직접접근가능성: 핵심벡터에서 일정 거리 내에 있으면 직접접근가능

접근가능성: 유한한 직접접근을 통해 도달할 수 있으면 접근가능

→ 군집: 한 핵심벡터에서 접근가능한 모든 데이터들의 집합

→ 노이즈: 어떠한 군집에도 속하지 않는 데이터

O4 Scikit-Learn Cluster

DBSCAN in Scikit-learn

장점

convex하지 않고 특징적인 모형의 데이터 분포를 효율적으로 군집화함
밀도를 기준으로 군집화하기 때문에 노이즈 처리에 강함

단점

비교적 많은 연산을 수행하기 때문에 속도가 느림



O4 Scikit Learn

sklearn.cluster.KMeans

```
class sklearn.cluster. KMeans (n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto')
```

[\[source\]](#)

- **n_clusters:** 클러스터 갯수 설정
- **init:** 첫 centroid의 위치 설정
- **n_init:** kmeans가 다른 위치의 centroid를 갖고 몇번 실행 할 것인지
- **max_iter:** 군집의 알맞는centroid를 찾기 위한 최대 iteration 수
- **tol:** Relative tolerance with regards to inertia to declare convergence
- **precompute_distances:** 각 데이터 포인트의 거리를 미리 측정
- **verbose:** verbosity mode 설정
- **random_state:**
 - int일 경우 random.seed(int) = 똑같은 결과를 도출 해내고 싶을때
 - None 일 경우: 랜덤으로 돌아감
- **copy_x:** 기존에 있는 데이터를 건드릴지 안건드릴지

O4 Scikit Learn

K-means++

<http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf>

The exact algorithm is as follows:

1. Choose one center uniformly at random from among the data points.
2. For each data point x , compute $D(x)$, the distance between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
4. Repeat Steps 2 and 3 until k centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard [k-means clustering](#).

Scikit-Learn KMeans References

Scikit Learn: Overview of Clustering Methods

<http://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods>

Sklearn.cluster.Kmeans

<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans>

KMeans Clustering Algorithm Source Code (GitHub)

https://github.com/scikit-learn/scikit-learn/blob/ef5cb84a/sklearn/cluster/k_means_.py#L707

Sarah Guido K-Means Clustering in Scikit Learn

<https://github.com/sarguido/k-means-clustering/blob/master/k-means-clustering.ipynb>

K-means ++

<https://en.wikipedia.org/wiki/K-means%2B%2B>

References

<https://www.cs.utah.edu/~piyush/teaching/4-10-print.pdf>

<http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/Sessions45/ClusterAnalysisReading.html>

https://datasciencelab.files.wordpress.com/2013/12/p_n200_k3.gif?w=404&zoom=2

<http://www.cs.put.poznan.pl/jstefanowski/sed/DM-7clusteringnew.pdf>

<http://gentlej90.tistory.com/29>

<http://elecs.tistory.com/167>

다변량 분석 II 수업자료 (이재용, 임요한)