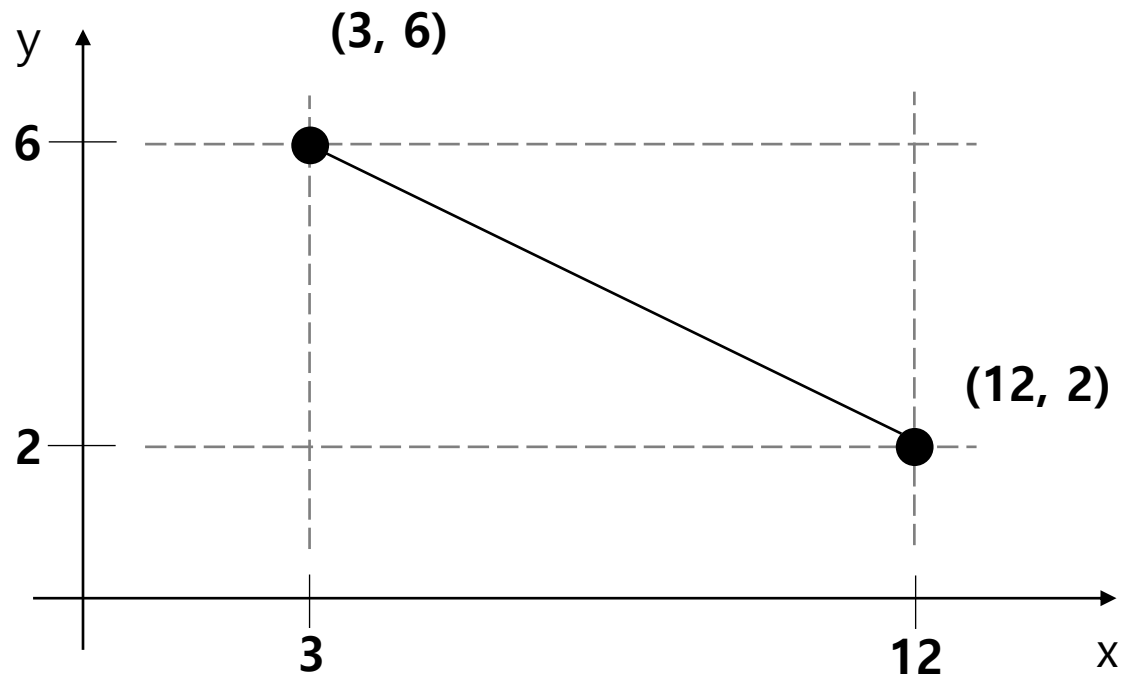
파이썬 입문

한국폴리텍대학

2023.03.24

두 점 사이 거리 구하기



연습

3차원 거리 구하기

p1 = (2, 6, 4)

p2 = (5, 2, 8)

다차원 거리 구하기

p1 = (5, 3, 2, 6, 3, 7, 1, 4, 3)

p2 = (8, 1, 4, 7, 3, 5, 5, 4, 8)

유클리드 거리 (Euclidean Distance)

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

연습

P1과 p2, p1과 p3의 코사인 유사도 구하기

p1 = (2, 6)

p2 = (4, 12)

P3 = (6, 3)

p1 = (2, 6, 4, 3, 8)

p2 = (5, 2, 8, 1, 2)

코사인 유사도 (Cosine Similarity)

1에 가까울수록 유사도 높음
0에 가까울수록 유사도 낮음

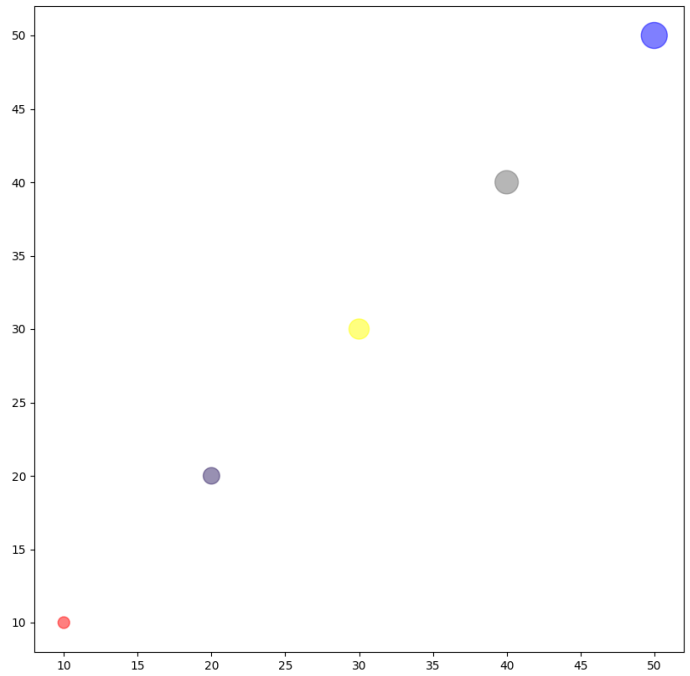
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

그래프 그리기

```
import matplotlib.pyplot as plt

x = [10,20,30,40,50]
y = [10,20,30,40,50]
area = [100,200,300,400,500]   < -- size
Colors =['red', '#332266','yellow','#6f6f6f','blue']  < -- color

plt.figure(figsize=(10,10))
plt.scatter(x, y, s=area, alpha=0.5, c=colors)    <--  산점도
plt.show()     < -- 그래프 화면에 표시
```

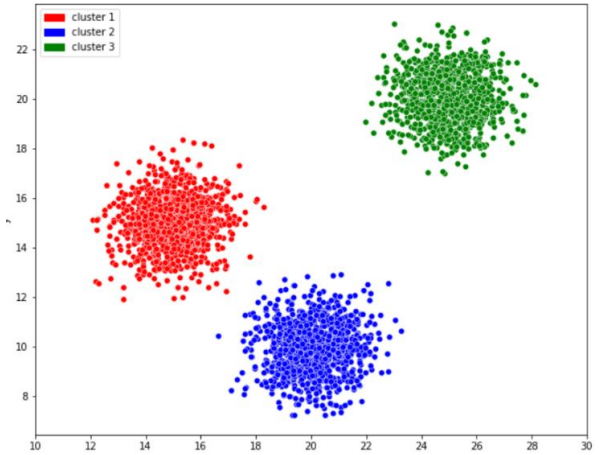


연습

K-평균 군집화 알고리즘 (K-means clustering)

1. 데이터 100개 생성 (0 ~ 100사이 2차원 데이터, random)
2. 군집(cluster or group)의 개수 설정 (3개)
3. 군집의 초기 중심점(centroid) 설정(임의의 위치 3곳)
4. 각 데이터를 가장 가까운 군집에 할당 (유클리드 거리)
5. 중심점 재설정 (군집 내의 데이터들의 평균 위치)
6. 각 데이터를 가장 가까운 군집에 할당
7. 군집화 완료. 데이터와 중심점 3곳을 화면에 출력

중심점 위치가 변하지 않을 때까지 반복



- 머신러닝
- 비지도 학습 (unsupervised learning)

filter(function, iterable)

```
def func1(n):
    return n < 3

ls = [1, 2, 3, 4, 5]
ls2 = filter(func1, ls)

print(list(ls2))    # [1, 2]
```

reduce(function, iterable)

```
from functools import reduce

num = [1, 2, 3, 4, 5]
total = reduce(lambda a, b: a + b, num)

print(f"total = {total}")      # 15

#print(sum(num))
```

zip(iterable, iterable, iterable,...)

```
a = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
b = [1, 2, 3, 4, 5]
c = ['A', 'B', 'C']

s = list(zip(a, b, c))
print(s)          # [('a', 1, 'A'), ('b', 2, 'B'), ('c', 3, 'C')]  # 튜플형
print(s[1][1])  # 2

ss = list(map(list, s))
print(ss)         # [['a', 1, 'A'], ['b', 2, 'B'], ['c', 3, 'C']]
```

iterable, iterator

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9]     # lst -> class 'list'
i = iter(lst)       # I -> class 'list_iterator'

print(next(i))    # 1
print(next(i))    # 2
print(next(i))    # 3

print(i.__next__())          # 4
print(i.__next__())          # 5
print(i.__next__(), 100)    # 6    < -- 모두 호출한 후에 100을 호출하여 예외 발생 방지
print(i)                  # <list_iterator object at 0x00……>

for s in i:
    print(s)       # 7, 8, 9
```

next()
__next__()

iter(list) -> list iterator object

재귀 함수 (Recursive Function)

재귀 호출

```
def recur():
    print("Recursive Function")
    recur()

recur()
```

```
def recur(count):
    if count == 0:
        return
    print(count)

    count -= 1
    recur(count)

recur(5)
```

파일 읽기 / 쓰기

```
f  = open('test.txt', 'r')

s1 = f.read()          # all  <class 'str'>
#s1 = f.read(5)       # 5 char <class 'str'>
print(s)

s2 = f.readline()      # 1 line <class 'str'>
print(s2)

s3 = f.readlines()     # all  <class 'list'>
print(s3)
Print(s3[1])

f.close()
```

```
f  = open('test.txt', 'w')
for i in range(10):
        f.write(f'{i} Python write")
        #f.write(f'{i} Python write₩n")

f.close()
```

파일 읽기 / 쓰기

```
with open('test.txt', 'r') as f:
    lines = f.readlines()

# close가 없음
```

```
with open('test.txt', 'w') as f:
    for i in range(10):
        f.write("python write")

# close가 없음
```

자동으로 파일이 close

JSON

```
# JSON (JavaScript Object Notation)

import json

with open('test.json', 'r') as f:
        #json_data = f.read()        <-- str
        #json_data = f.readline()    <-- str
        #json_data = f.readlines()   <-- list
        json_data = json.load(f)     <-- dict

print(json.dumps(json_data))    <-- dict -> str

# load, dump  : 파일 입출력
# loads, dumps  : 데이터형 변환
```

Python은 JSON 데이터를
Dictionary형식으로 불러온다.
(다중 사전형)

loads -- > str -> dict
dumps --> dict -> str

```
# JSON 파일 쓰기

s1 = dict()    # s1 = {}
s1['a1'] = 'apple'
s1['a2'] = 'banana'

s2 = dict()
s2['a1'] = 'gold'
s2['a2'] = 'silver'

s3 = dice()
s3['s1'] = s1
s3['s2'] = s2

print(s3)
```

```
import json

with open('test.json', 'w', encoding='utf-8-sig') as s:
        json.dump(s3, s, indent = 4)
        #json.dump(s3, s)


# indent = 4  → 들여쓰기 4칸
                저장할 데이터가 많을 경우
                용량 커짐 주의
```

2차원 리스트

```
data = [ [10, 15],
         [20, 25],
         [30, 35] ]

print(data[1][1])   # 25

data[2][0] = 50
print(data[2][0])   # 50
```

```
# 톱니형 리스트 (jagged list)

data = [ [10, 15],
         [20, 25, 10, 8, 55, 1],
         [3],
         [14, 7, 40, 0] ]

print(data[1][1])   # 25

data[2][0] = 50
print(data[2][0])   # 50
```

2차원 리스트

```
data = []
data.append([])
data[0].append{10)
data[0].append{20)
data.append([])
data[1].append{1)
data[1].append{2)
data[1].append{3)

print(data)
```

```
data1 = ((0, 1), (2, 3), (4, 5))
data2 = ([0, 1], [2, 3], [4, 5])
data3 = [(0, 1), (2, 3), (4, 5)]

data1[1][1] = 10          # TypeError
data1[1]    = (10, 10)    # TypeError
data2[1][1] = 10
data2[1]    = (10, 10)    # TypeError
data3[1][1] = 10          # TypeError
data3[1]    = (10, 10)
```

3차원 리스트

```
# 2차원 리스트 초기화

[[0 for i in range(3)] for j in range(5)]

# 3차원 리스트 초기화

[[[0 for i in range(3)] for j in range(5)] for k in range(10)]
```

__name__

```
# 모듈 직접 실행
# mod1.py

def main():
        pass

if __name__ == “__main__”:
        print(__name__)      # __main__
        main()
else:
        print(__name__)      # mod1
```

__name__은 파이썬 내장변수로
현재 모듈의 이름을 담고 있다.

모듈 실행 방법

1. 직접 실행

2. import해서 실행

클로저 (closure)

```
def c_func():
    a = 5
    b = 2
    def calc(x):
        return x * a + b
    return calc

cf = c_func()

print(cf(3), cf(10))  # 17, 52
```

변수 숨김

( )없음

예외처리

try:
내용

except:

try문 예외 발생하면 실행

else:

try문 오류가 없으면 실행

finally:

try문 예외 발생 여부에 상관없이 항상 실행

배열 (array)

```
import numpy as np                                  # numerical python

data1 = np.array([1, 2, 3, 4])
data2 = np.array([[1, 2], [3, 4]])
data3 = np.array([[1, 2, 3, 4]])

print(data1, type(data1), data1.shape)              # [1,2,3,4], numpy.ndarray, (4, )
print(data2, type(data2), data2.shape)              # [[1,2]
                                                        [3,4]], numpy.ndarray, (2, 2)
print(data3, type(data3), data3.shape)              # [[1,2,3,4]], numpy.ndarray, (1, 4)
```

ndarray: n-dimensional array (n – 차원 배열)