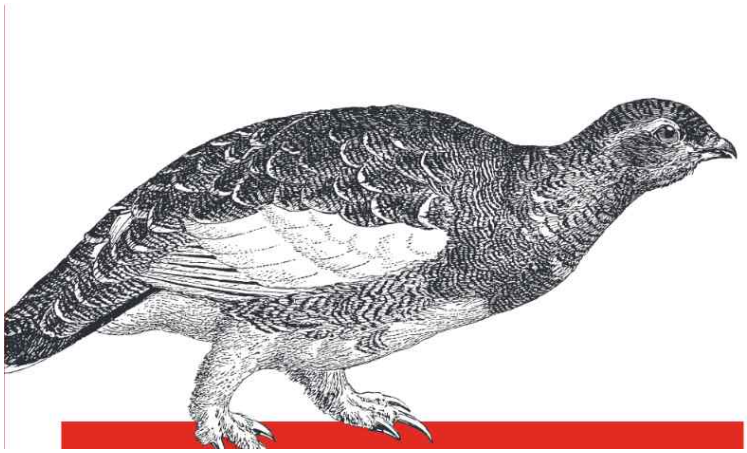
DataScience from Scratch

7기 김대규 deepvine@gmail.com

CHAPTER 19 Clusering

0. Python : List and Functions
1. KMEANS CLASS
2. EXAMPLE : MEETSUP
3. Choosing K
4. EXAMPLE : Clustering Colors



Python : List and Functions

시작하기 전에 알아야할 python 자료구조와 함수들

List

리스트를 이용하면 1, 3, 5, 7, 9라는 숫자 모음을 다음과 같이 간단하게 표현할 수 있다.

```
>>> odd = [1, 3, 5, 7, 9]
```

리스트를 만들 때는 위에서 보는 것과 같이 대괄호([ ])로 감싸 주고 각 요소값들은 쉼표(,)로 구분해 준다.

리스트명 = [요소1, 요소2, 요소3, ...]

리스트 예시들

```
>>> a = [ ]
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

좀더 자세한 내용 : https://wikidocs.net/14

zip()

zip(iterable*)은 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수이다.

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]

>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

```
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

map()

map(f, iterable)은 함수(f)와 반복 가능한(iterable) 자료형을 입력으로 받는다. map은 입력받은 자료형의 각 요소가 함수 f에 의해 수행된 결과를 묶어서 리턴하는 함수이다.

Ex)

```
def two_times(numberList):
    result = [ ]
    for number in numberList:
        result.append(number*2)
    return result

result = two_times([1, 2, 3, 4])
print(result)
```

Lambda 함수

Ex)

다음은 두 수를 더하는 함수입니다.

```
>>> def hap(x, y):
...     return x + y
...
>>> hap(10, 20)
30
```

이것을 람다 형식으로는 어떻게 표현하면

```
>>> (lambda x,y: x + y)(10, 20)
30
```

x,y에 10과 20을 넘겨주고 x+y를 리턴한다. 함수를 선언 할 필요게 최대 장점

min()

min(iterable[, key])¶

min(arg1, arg2, *args[, key])

min(iterable)은 max 함수와 반대로, 인수로 반복 가능한 자료형을 입력받아 그 최소값을 리턴하는 함수입니다.

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

1

>>> min("python")

'h'

key에는 함수가 사용되고, 인자들을 함수에 대입하여 나온 리턴값들 중 최소를 리턴하는 인자를 리턴합니다.

좀 더 자세한 내용 https://wikidocs.net/32

KMEANS class

Original source code https://github.com/joelgrus/data-science-from-scratch/blob/master/code/clustering.py

Revision version 첨부파일 Boaz.zip (주석 포함되어 있습니다.)

벡터 계산을 위해 본 책 ch4에 있는 linear_algebra.py를 import 시켜줍니다.

from linear_algebra import *

linear_algebra.py source code https://github.com/joelgrus/data-science-from-scratch/blob/master/code/linear_algebra.py

(이건 그냥 갔다 써도 무방합니다)

먼저 clustering 하기 위해 아래 공식을 python으로 구현되어 있는걸 보겠습니다.

$$d(x,y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}$$

Squared_distance 함수 입니다.

```
def squared_distance(v, w):
    return sum_of_squares(vector_subtract(v, w))


def vector_subtract(v, w):
    """subtracts two vectors componentwise"""
    return [v_i - w_i for v_i, w_i in zip(v,w)]


def sum_of_squares(v):
    """v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)


def dot(v, w):
    """v_1 * w_1 + ... + v_n * w_n"""
    return sum(v_i * w_i for v_i, w_i in zip(v, w))
```

결과적으로 v, w 벡터의 차를 list로 계산한 뒤 내적을 구하게 됩니다.

```
def classify(self, input):
        """return the index of the cluster closest to the input"""
        return min(range(self.k), #k개의 array 생성
                   key=lambda i: squared_distance(input, self.means[i])) #유클리드 거리 구하기
```

그래서 squared_distance를 통해 2개의 포인터가 최소 거리인 index를 구합니다.

다음은 clustering.py의 train 함수 입니다

```
def train(self, inputs):
        self.means = random.sample(inputs, self.k) #임의의 k개 점을 중심점으로 선택. list
        assignments = None

        while True:
            # 소속되는 군집을 반복적으로 찾기
            new_assignments = map(self.classify, inputs)
```

```
        # 소속되는 군집이 바뀌지 않았다면 종료
        if assignments == new_assignments:
            return
        # 아니라면 새로운 군집을 찾기
        assignments = new_assignments
        for i in range(self.k):
            # 군집 i에 속하는 모든 데이터 탐색
            i_points = [p for p, a in zip(inputs, assignments) if a == i]
            # 0으로 나누는 일이 없도록 i_points가 비어 있지 않는지 확인
            if i_points:
                self.means[i] = vector_mean(i_points)
```

EXAMPLE MEETSUP

DataSciencer 사의 급성장을 축하하기 위해 고객 보상 팀의 부사장은 치맥과 DataSciencer 티셔츠가 완비된 오프라인 미팅을 준비하고 싶어한다. 다만 그는, 팀원의 거주지 데이터를 기반으로 모두가 만족할 수 있는 모임 장소를 선정하자고 제안했다

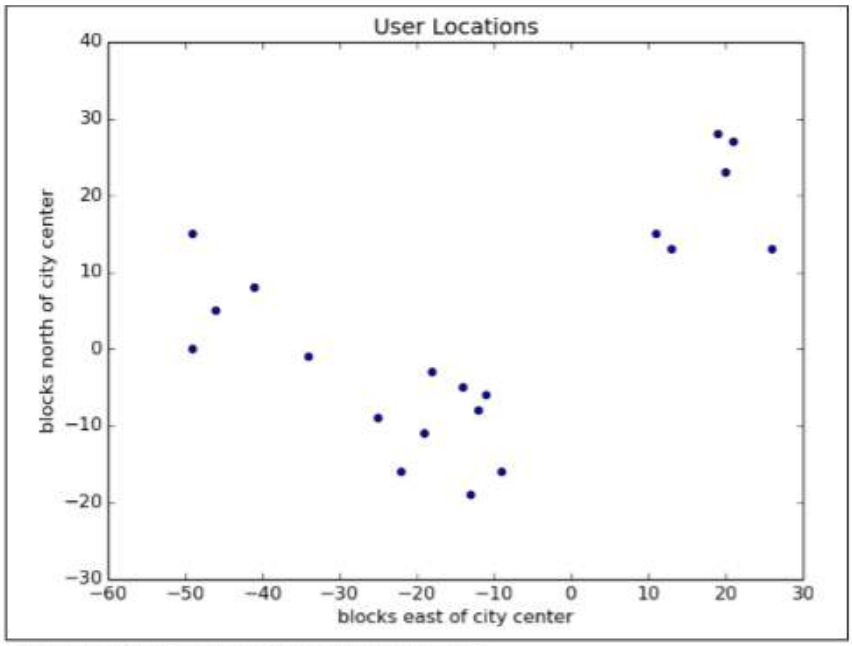


Figure 19-1. The locations of your hometown users

k가 3일 때

```
#-*- coding: utf-8 -*-
from clustering import KMeans
import random
from clustering import squared_clustering_errors
from clustering import *
import matplotlib.pyplot as plt

data = [[-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]] #20개


#1 MEETSUP 3 K
#random.seed(0)
clusterer = KMeans(3)
clusterer.train(data)
```

```
print clusterer.means
```

결과값

```
[[18.333333333333332, 19.833333333333332], [-43.800000000000004, 5.4], [-15.888888888888888, -10.333333333333332]]
```

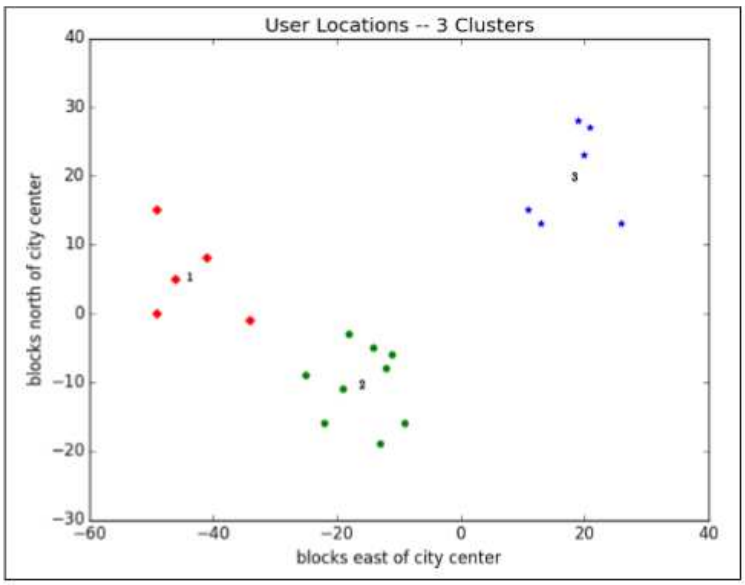


Figure 19-2. User locations grouped into three clusters

As shown in Figure 19-3, one meetup should still be near [18, 20], but now the other should be near [-26, -5].

위의 그림처럼 3개의 중심좌표가 설정되어 있는 것을 볼 수 있습니다.

k가 2일 때

```
#-*- coding: utf-8 -*-
from clustering import KMeans
import random
from clustering import squared_clustering_errors
from clustering import *
import matplotlib.pyplot as plt

data = [[-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]] #20개


#1 MEETSUP 3 K
#random.seed(0)
clusterer = KMeans(2)
clusterer.train(data)
print clusterer.means
```

결과값

```
[[-25.857142857142854, -4.714285714285714], [18.333333333333332, 19.833333333333332]]
```

이 정도로 나오면 성공입니다

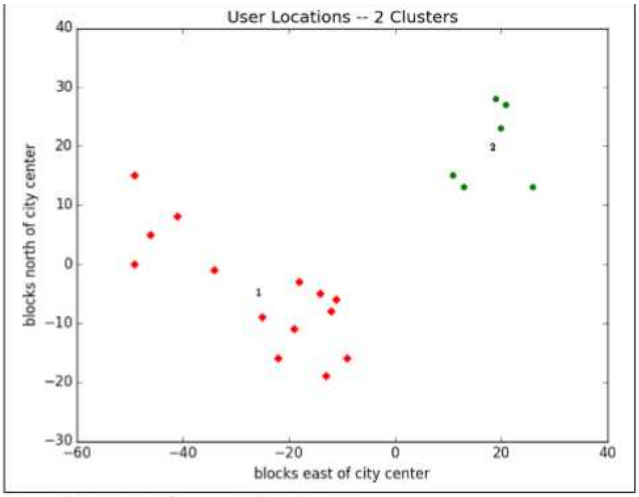


Figure 19-3. User locations grouped into two clusters

Choosing K

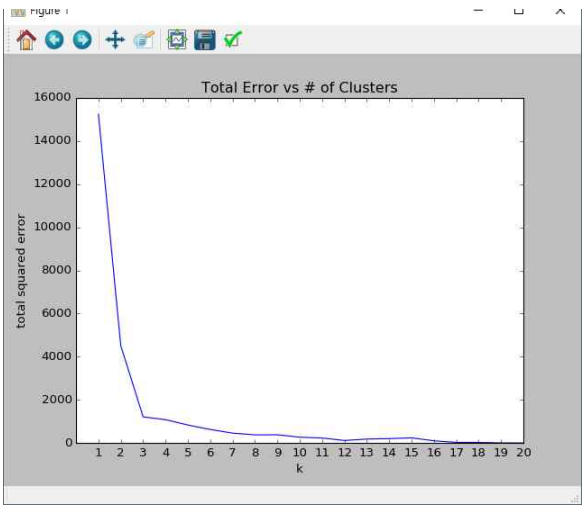
앞의 예시에서는 k값을 미리 설정하였지만 실제로는 k값을 미리 정하는 경우는 드물다고 합니다. 아래는 k값에 대해 중심 점과 각 데이터 포인트 사이의 거리의 제곱합을 통해서 k값을 정하는 방법입니다.

```
def squared_clustering_errors(inputs, k):
    """finds the total squared error from k-means clustering the inputs"""
    clusterer = KMeans(k)
    clusterer.train(inputs)
    means = clusterer.means
    assignments = map(clusterer.classify, inputs)

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

main에서

```
ks = range(1, len(data)+1)
errors = [squared_clustering_errors(data, 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()
```

실행하면 아래와 같이 k가 3이 가장 최적값임을 알 수 있습니다.

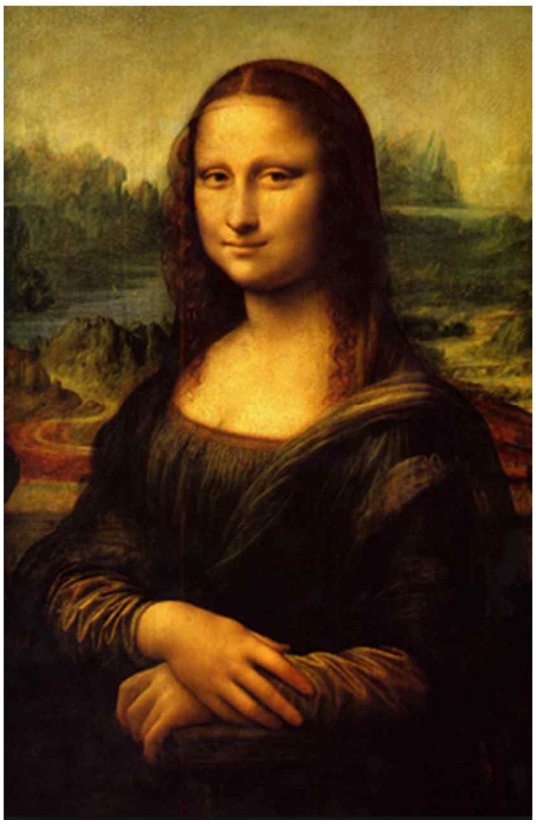


EXAMPLE : Clustering Colors

부사장은 오프라인 모임에서 제공할 수 있는 스티커를 여러장 제작했다. 그런데 아쉽게도 스티커 프린터는 총 다섯 가지 색밖에 출력할 수 가 없다. 이미지 파일을 5가지 색으로 바꿔보자.

1. 다섯가지 색을 고른다
2. 각 픽셀을 다섯 가지 색 중 하나로 매핑한다.

실습 이미지는 모나리자로 했습니다.(아무거나 해도 상관없습니다 이미지 path만 수정해주세요.)



먼저 matplotlib로 이미지를 불러 올 수 있습니다.

```
img = mpimg.imread(path_to_png_file)
```

그리고 아래와 같이 리스트 형식으로 이미지를 저장하면 됩니다.

```
pixels = [pixel for row in img for pixel in row]
```

실습코드 :

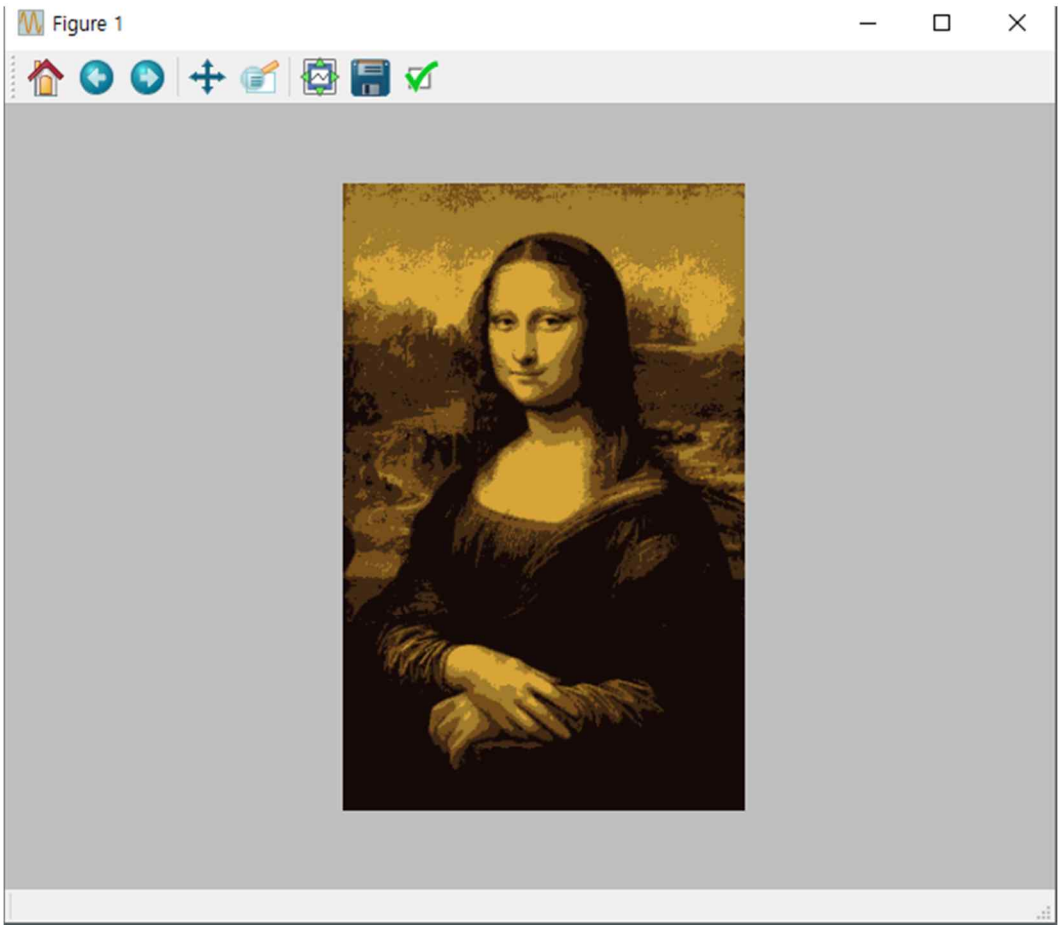
```
path_to_png_file = "D:\example_image.png"

recolor_image(data, 5, path_to_png_file)
```

```
def recolor_image(input_file, k, path_to_png_file):
    img = mpimg.imread(path_to_png_file)
    pixels = [pixel for row in img for pixel in row]
    clusterer = KMeans(k)
    clusterer.train(pixels) # this might take a while
```

```
    def recolor(pixel):
        cluster = clusterer.classify(pixel) # index of the closest cluster
        return clusterer.means[cluster]     # mean of the closest cluster
    new_img = [[recolor(pixel) for pixel in row]
               for row in img]
    plt.imshow(new_img)
    plt.axis('off')
    plt.show()
```

대략 10분정도 돌아갈 것입니다……

결과물



끝