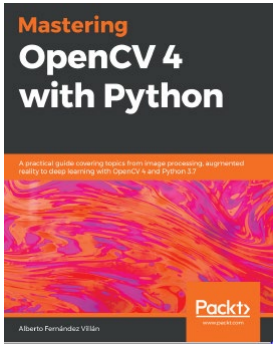
OpenCV Python

디지털영상처리/컴퓨터비전

K means Clustering

교재 10장의 일부..



Mastering OpenCV 4 with Python, Alberto, Packt, Pub. 2019

2024년 1학기

서경대학교 김진헌

차례

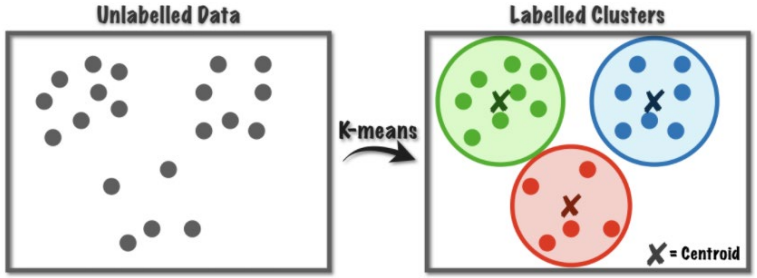
1

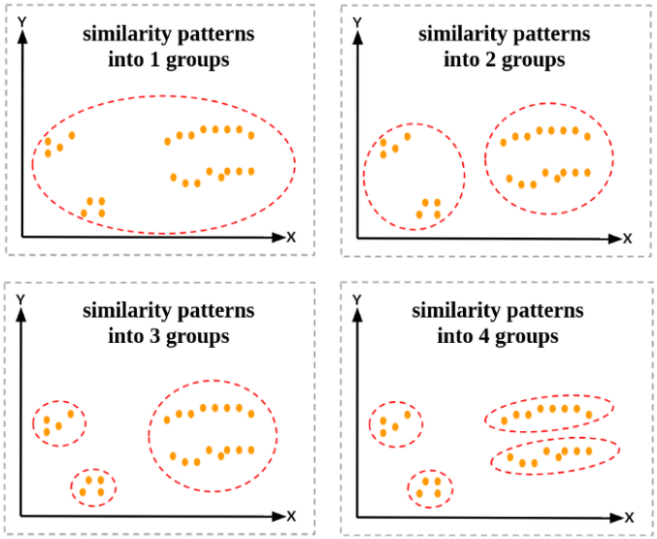
1. 무엇을 공부하나요?

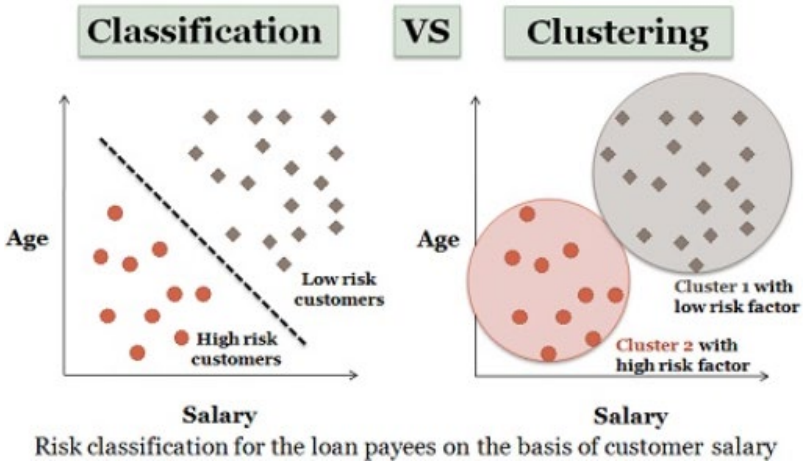
- 본 topic의 목표
  - 기계학습의 입문 알고리즘이라 할 수 있는 k-means clustering의 특징과 원리를 이해한다.
- 내용
  - K 군집화알고리즘을 24비트 컬러를 64, 256 컬러로 줄이는데 활용해보자.
    - Indexed color를 사용하는 영상 파일(PNG, BMP 등)에서 사용하고 있음
- 코딩상의 초점
  - OpenCV의 kmeans() 함수의 사용법
  - 2채널, 3채널 자료에 군집화 알고리즘 적용
  - scatter(), 각종 파이썬 함수-flatten(), ravel()

2. K-means Clustering이란?

- 주어진 데이터 셋트를 K개의 cluster(군집)로 나누는 알고리즘을 말한다.
- 용어는 1967년, 아이디어와 초창기 표준 알고리즘은 각각 1956년, 1957년에 만들어짐.
  - 비지도 학습(Unsupervised Learning)이다.







2.1 수학적 표현을 사용한 정의

K-means Clustering의 목표를 수학적 표현을 사용하여 정의하면?

- n개의 데이터가 d 차원으로 나열되어 있을 때 k-means clustering 목표는 n개의 관찰데이터를 각 클러스터의 중심점과 데이터와의 거리의 제곱의 합이 최소가 되도록 k개의 집합으로 나누는 것이다.

Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance). Formally, the objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

where $\boldsymbol{\mu}_i$ is the mean (also called centroid) of points in S_i, i.e.

$$\boldsymbol{\mu}_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x}, \qquad |S_i| \text{ is the size of } S_i, \text{ and } \|\cdot\| \text{ is the usual } L^2 \text{ norm}.$$

2.2 그림으로 본 표준 알고리즘

동영상 링크

5

문제 정의

In this case the data make three, relatively obvious, clusters.

But, rather than rely on our eye, let's see if we can get a computer to identify the same 3 clusters.

To do this, we'll use K-means clustering.

Step 1: Select the number of clusters you want to identify in your data. This is the "K" in "K-means clustering".

In this case, we'll select K=3. That is to say, we want to identify 3 clusters.

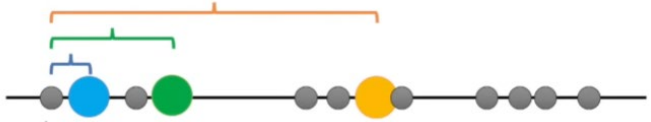
There is a fancier way to select a value for "K", but we'll talk about that later.

1차 시도

Step 2: Randomly select 3 distinct data points.

These are the initial clusters.

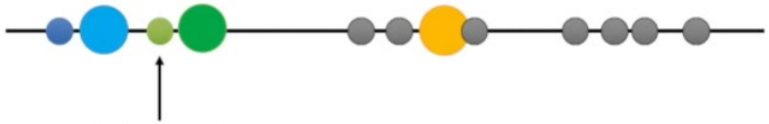
Distance from the 1st point to the orange cluster



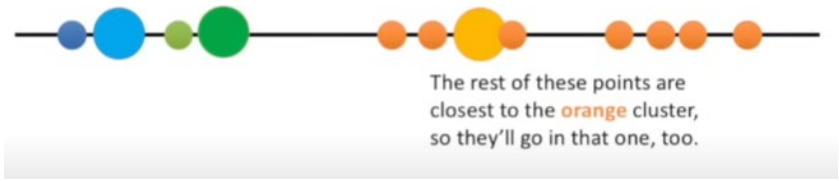
Step 3: Measure the distance between the 1st point and the three initial clusters.

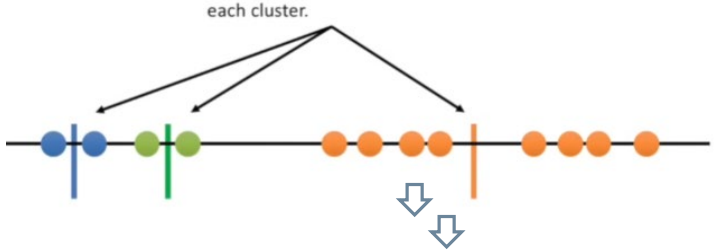
Step 4: Assign the 1st point to the nearest cluster. In this case, the nearest cluster is the blue cluster.

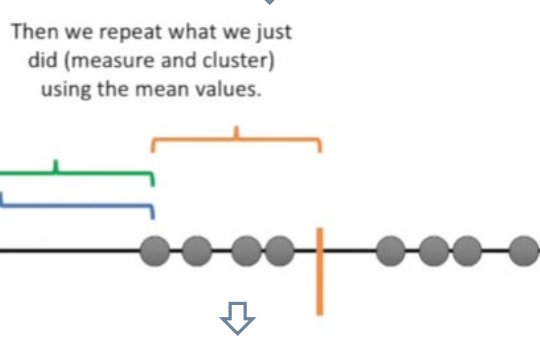
Measure the distances...

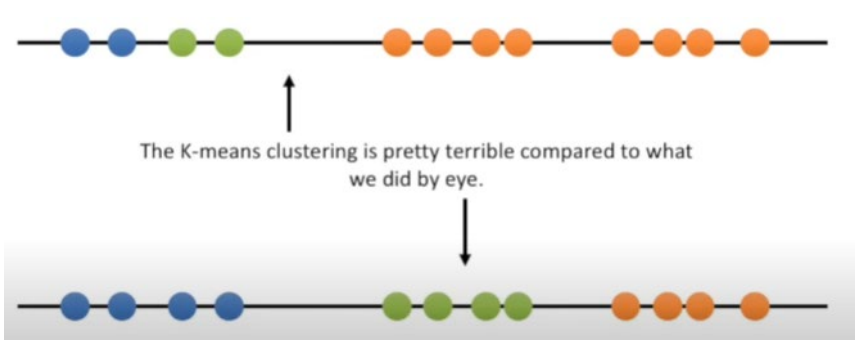


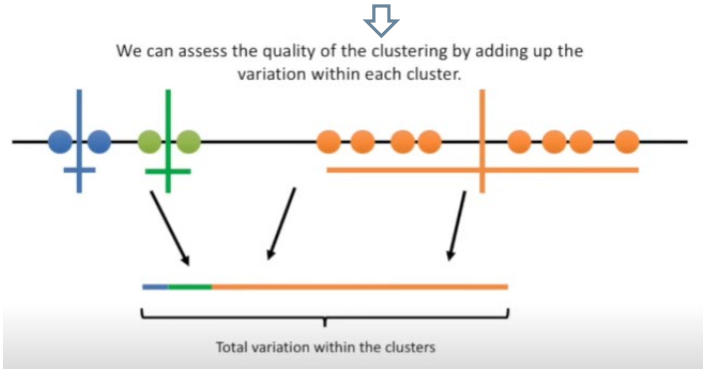
Assign the point to the nearest cluster.







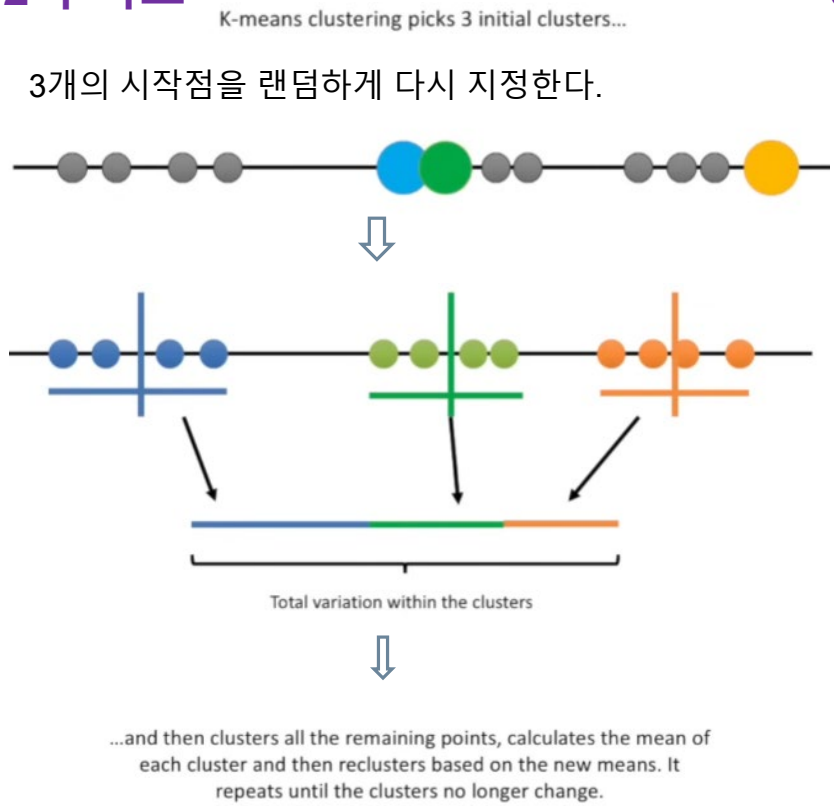




Since K-means clustering can't "see" the best clustering, its only option is to keep track of these clusters, and their total variance, and do the whole thing over again with different starting points.

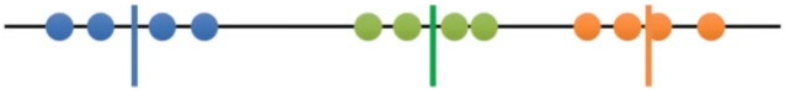
1차 시도에서는 더 이상 중심점 조정이 일어날 여지가 없다. 그렇다고 클러스터링이 잘된 것도 아니며 Total variance만 보고는 클러스터링이 잘되었는지 판단할 수 없다.
이 결과를 기록해두고 다른 step1의 새 초깃값을 바탕으로 모든 과정을 다시 수행한다.

2차 시도

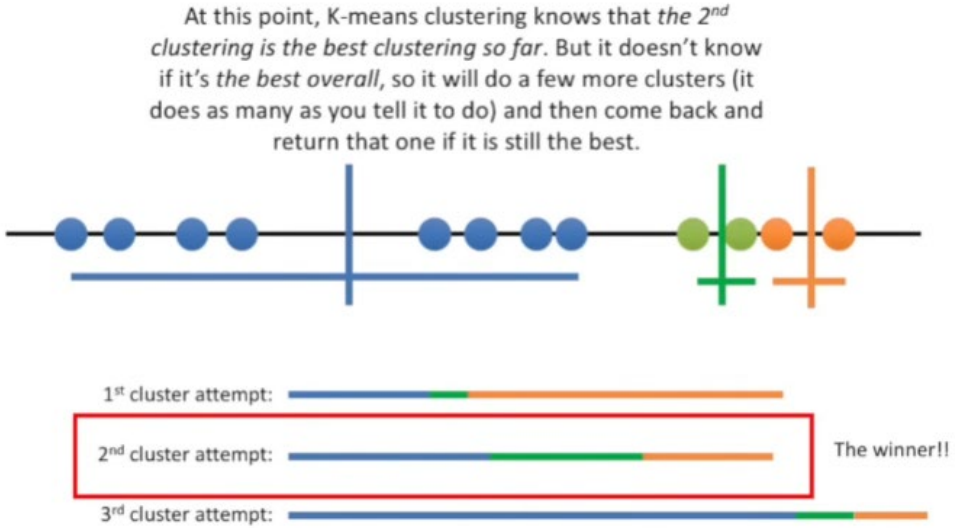


3개의 시작점을 랜덤하게 다시 지정한다.

2차 시도의 최종 결과



3차 시도: 초기치 및 최종 variance



2차 시도 결과보다 variance가 나쁘므로 2차 시도 클러스터링 데이터는 폐기한다.
이후 더 시도해 볼지를 결정한다.
→ 프로그래머의 몫(iteration parameter)

1차 시도 결과보다 variance가 좋으므로 1차 시도 클러스터링 데이터는 폐기한다.

2.3 글로 기술한 표준 알고리즘

발췌 링크 + α

- **방법 - 가장 단순한 방법**
- **1. 임의의 k개 중심을 선정합니다. (초기치 중심을 설정해줘야 합니다)**
  - 랜덤하게 각 샘플의 중심이라고 간주할 위치를 임의로 선택합니다.
  - 초기 중심점을 어떻게 선정하냐에 따라 결과값이 달라지게 됩니다.
- **2. 모든 데이터에 대하여 가장 가까운 중심을 선택하여 이동합니다.**
  - 중심점 간의 거리의 중간으로 영역을 분할합니다.
  - 초평면 안에 있는 데이터들의 평균위치를 계산하게 됩니다.
- **3. 각 군집에 대해 중심을 다시 계산합니다.**
  - 중심을 이동하고 클래스에 해당하는 영역을 다시 선정합니다.
- **4. 중심이 변경되면 2~3 과정을 반복합니다.**
  - 중심이 바뀌지 않을 때까지 반복합니다.
  - OpenCV 함수에서는 종료 기준을 선정할 수 있습니다.→ maxCount, epsilon
- **5. 중심이 변경되지 않으면 종료합니다.**

1차시도

- **초기 위치가 잘못 설정되면 군집화가 잘못 시행될 수 있다. 이 때문에 초기 위치를 바꾸어 다시 1의 위치로 돌아간다. → attempts**

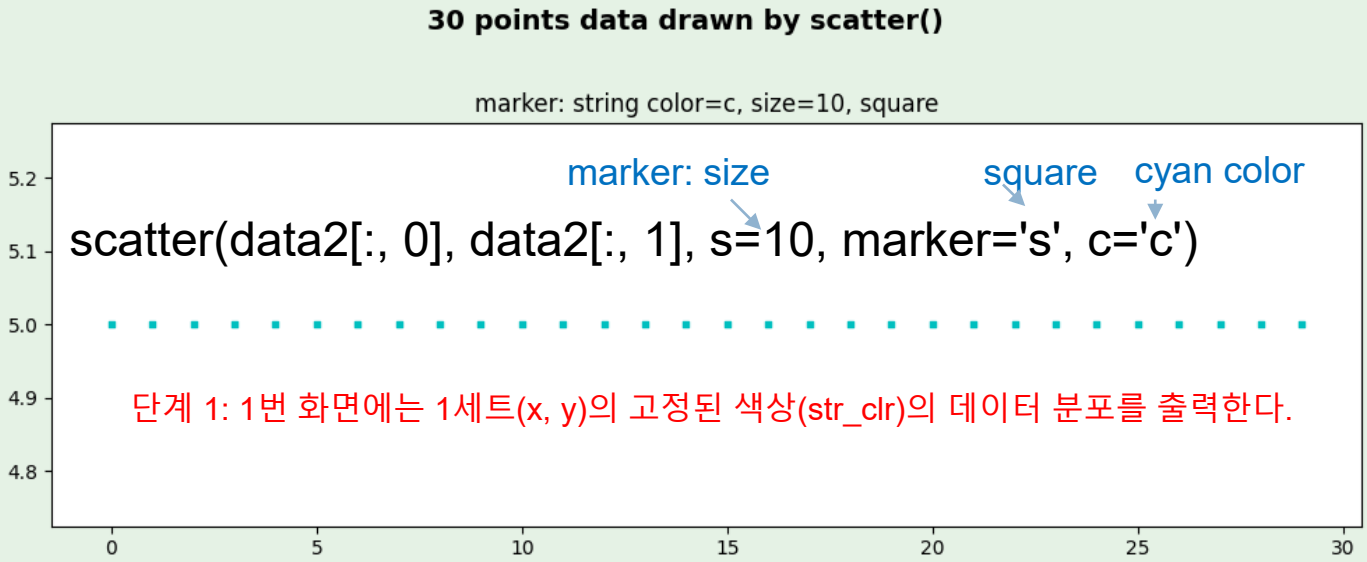
3. 사전 연습

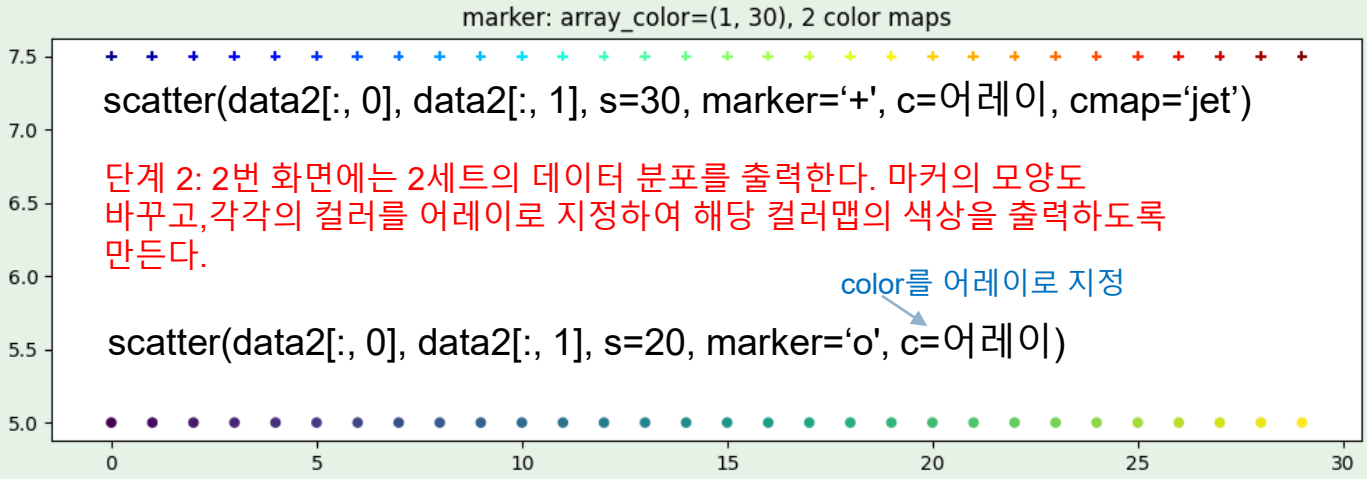
- scatter() 함수 - 2차 평면의 원하는 지점에 marking하는 함수

scatter() 함수의 장점

비교	circle() 함수	scatter() 함수
마킹을 여러 곳에 할 경우	Loop문이 필요	한 번의 호출
모양, 색상	고정	여러가지..

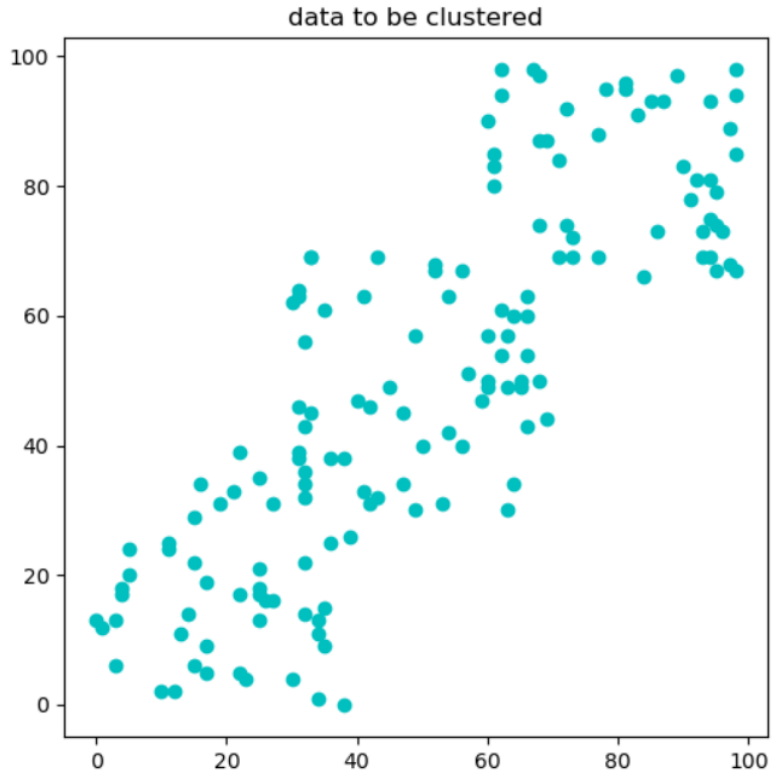
3.1 scatter 함수 연습





3.2 랜덤 좌표에 점 그리기

1_0_k_means_clustering_data_visualization.py



- 30개의 좌표 x 3세트,총 150개의 랜덤 좌표에 cyan 색상의 점으로 표시한다.
- 각 세트는 지정된 범위의 랜덤 좌표(x, y)로 구성되어 있다.
  - 수행할 때마다 다른 위치에 점이 찍힌다.
- 랜덤 좌표 정보는 randint() 함수로 구현함.
  - numpy.random.randint(low, high=None, size=None, dtype='l')
  - 사례: a = np.random.randint(0, 40, (50, 2))
  - 위치값이 [0, 40)까지의 좌표 값이 존재하는 50개의 2차원 좌표 어레이를 생성한다.
  - 3개의 어레이를 vstack() 함수로 묶어 50x3개의 랜덤 좌표 데이터를 생성하여 그 지점에 점을 그린다.

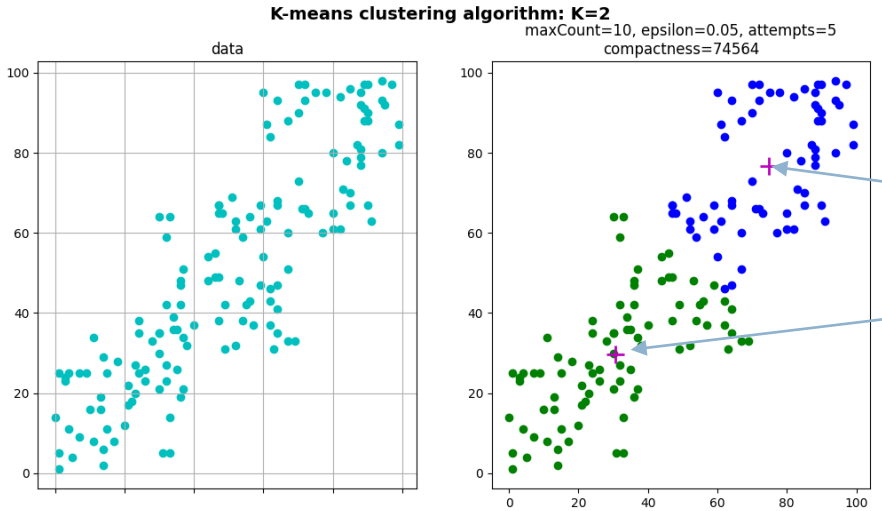
```
data = np.float32(np.vstack((a, b, c)))
scatter(data[:, 0], data[:, 1], c='m')
```

이 같은 방식으로 생성된 3개의 그룹 데이터 a, b, c를 scatter()로 표현하였다.

4. K(=2) means clustering

kmeams() 함수의 입력 및 반환 값의 데이터 관찰: 입력 데이터 셋트 5를 사용함

1_1_k_means_clustering_k_2.py




```
5. center:
 [[74.818184 76.77273 ]
 [30.714287 29.607143]]
```

클러스터 A(label=0)의 중심점

클러스터 B(label=1)의 중심점

```
num_maxCount = 10; eps = 0.05; num_attempts = 5
criteria = (cv2.TERM_CRITERIA_MAX_ITER+cv2.TERM_CRITERIA_EPS, num_maxCount, eps)
```

$$\sum_i \|samples_i - centers_{labels_i}\|^2$$

ret: kmeans() 함수의 반환값

K=2개로 그룹핑

```
data: <class 'numpy.ndarray'>, shape=(150, 2), len=150
ret, label, center = cv2.kmeans(data, 2, None, criteria,
                                5, cv2.KMEANS_RANDOM_CENTERS)
ret: 71623.28
label: type=<class 'numpy.ndarray'>, shape=(150, 1)
center: type=<class 'numpy.ndarray'>, shape=(2, 2)
label.ravel(): shape=(150,)
```

5회의 중심점 초기화를 통한 탐색 시도

data의 각 원소가 배정 받은 레이블을 저장

```
A = data[label.ravel() == 0]
B = data[label.ravel() == 1]
```

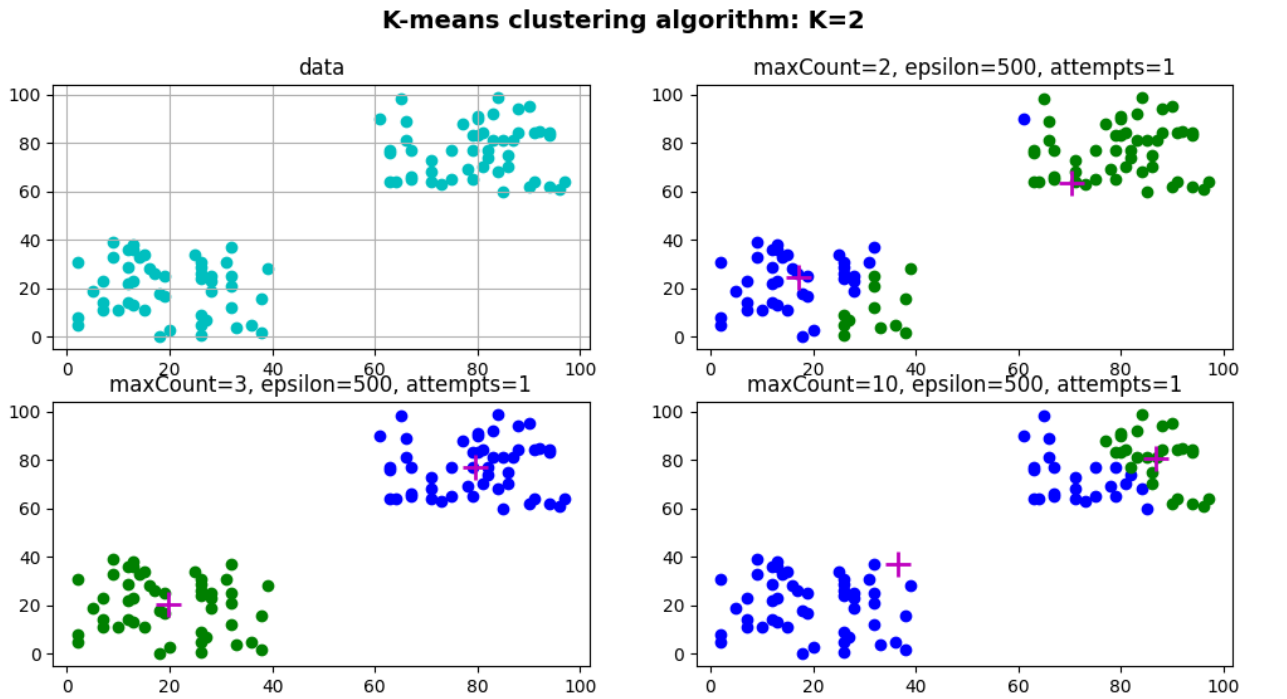
```
label.ravel(): shape=(150,)
A: shape=(73, 2)
B: shape=(77, 2)
```

4.1 attempts=1 사례

극단적인 epsilon의 설정에 대한 maxCount의 영향: 입력 데이터 세트 1을 사용함

1_1_k_means_clustering_k_2_execution_3_times.py

- epsilon의 값을 너무 크게 설정하면 클러스터 중심의 이동 거리에 대한 감도가 낮아져서 클러스터링 알고리즘이 더 빠르게 종료될 수 있다. 이 때문에 compactness의 값을 더 줄이지 못한다. 이 경우 num_maxCount의 설정 값은 종료 시점에 거의 영향을 미치지 못한다.
- epsilon의 값을 극히 작게 설정하면 num_maxCount가 클수록 compactness의 값을 더 줄일 수 있어 만족한 결과를 얻을 수 있다.
- 위 명제들을 실험을 통해 보이고자 한다.



화면 3: maxCount가 화면 4보다 작은데도 클러스터링에 성공. 사실상 eps가 극한 적으로 크면 사실상 maxCount에 영향을 받지 않는다.

2번 화면 ret, compactness=87640.65

3번 화면. ret, compactness=22488.90

4번 화면. ret, compactness=106217.65

* 예상되는 결과는maxCount값이 커질 수록compactness가 작아져야 한다.(오류: epsilon이 작을 때에 국한된다.)

-> epsilon이 너무크면 maxCount의 설정에 영향받지 않는다에 부합한다.

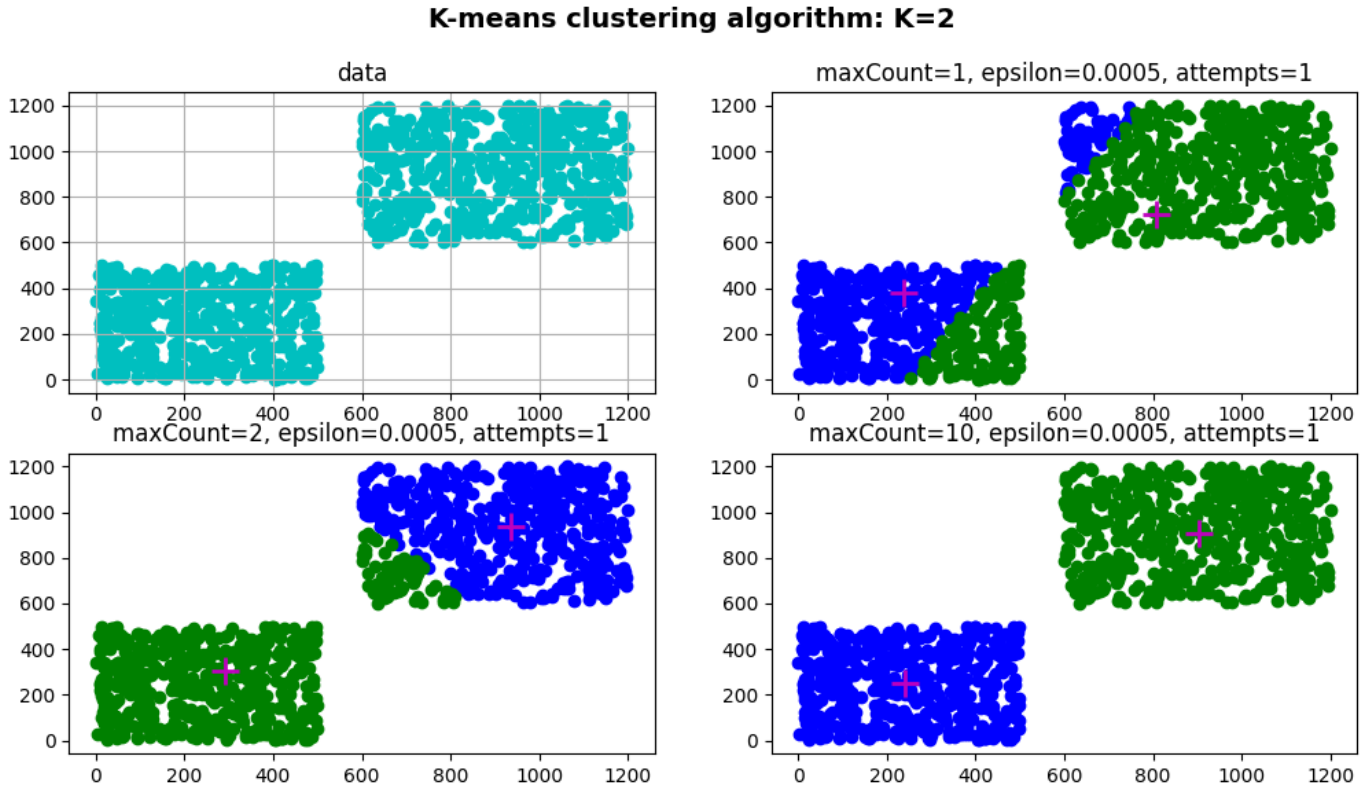
* 이는 위의 논리적인 추정과 일치한다.

4.1 attempts=1 사례

극단적인 epsilon의 설정에 대한 maxCount의 영향: 입력 데이터 세트 6를 사용함

1_1_k_means_clustering_k_2_execution_3_times.py

- 예상한 대로 epsilon이 충분히 작으면 maxCount가 증가하면 compactness가 작아져 정밀도가 증가한다.

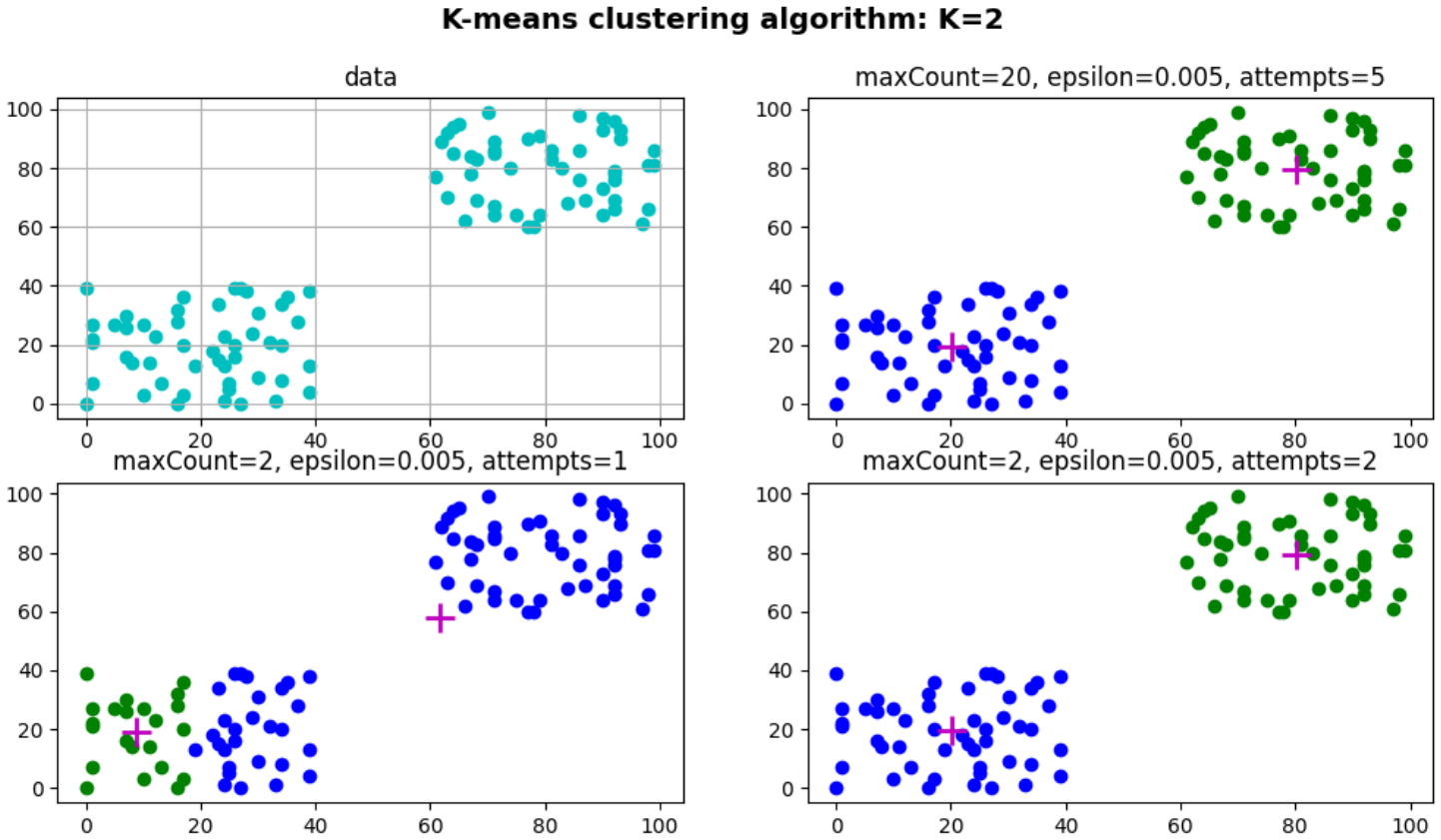


* maxCount의 영향을 보이기 위해서는 가능한 데이터의 개수가 많아야 한다. 데이터세트 6은 1,000개의 점이 사용되었다.
* 데이터의 개수가 작으면(세트 1) maxCount 의 크기에 관계없이 운에 따라 1회~2회에 수렴해 버릴 수 있다.

```
2번 화면 ret, compactness=162292668.78
3번 화면. ret, compactness=70470447.56
4번 화면. ret, compactness=52001223.08
```

4.2 attempts=1 이상의 사례

- 4번: 2회의 초기 위치 지정으로 군집화 성공



5. kmeans() 함수 분석

- 입력
- 반환 값

retval, bestLabels, centers=**cv.kmeans**(*data*, K, None, criteria, attempts, flags)

5.1 입력

- *retval, bestLabels, centers*=**cv.kmeans**(*data*, K, None, criteria, attempts, flags)

data •Data for clustering. An array of N-Dimensional points with float coordinates is needed. N 차원을 가진 Z개의 데이터이다. 이 경우 data.shape=(Z, N)이다.
•(x, y) 위치 정보를 가진 Z개의 데이터: data.shape=(Z, 2)
•후술할 색상 양자화 예제에서 3채널 영상 전체(면적이 A라 하자)의 화소를 데이터로 지정하면 data.shape=(A, 3)이다.

K Number of clusters to split the set by.

attempts Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness (see the last function parameter).

flags Flag that can take values of cv::KmeansFlags

KMEANS_RANDOM_CENTERS Python: cv.KMEANS_RANDOM_CENTERS	Select random initial centers in each attempt.
KMEANS_PP_CENTERS Python: cv.KMEANS_PP_CENTERS	Use kmeans++ center initialization by Arthur and Vassilvitskii [Arthur2007].
KMEANS_USE_INITIAL_LABELS Python: cv.KMEANS_USE_INITIAL_LABELS	During the first (and possibly the only) attempt, use the user-supplied labels instead of computing them from the initial centers. For the second and further attempts, use the random or semi-random centers. Use one of KMEANS_*_CENTERS flag to specify the exact method.

flags = KMEANS_USE_INITIAL_LABELS) flag를 선택하고 attempts=1로 선정하면 프로그래머가 직접 자신의 알고리즘으로 그 중심값의 초기화 값을 지정하면서 중심점을 찾아나간다.

criteria

- *retval, bestLabels, centers*=**cv.kmeans**(*data*, K, None, criteria, attempts, flags)

criteria The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy.

criteria= `TermCriteria (int type, int maxCount, double epsilon)`

Constructor

생성자 함수 없이 튜플형으로 (type, maxCount, epsilon)으로 써도 됨

type The type of termination criteria, one of TermCriteria::Type

maxCount The maximum number of iterations or elements to compute.

epsilon The desired accuracy or change in parameters at which the iterative algorithm stops.

5.2 반환값

- *retval, bestLabels, centers*=**cv.kmeans**(*data*, K, None, criteria, attempts, flags)

Returns *retval* = compactness

The function returns the compactness measure that is computed as

$$\sum_i \|\texttt{samples}_i - \texttt{centers}_{\texttt{labels}_i}\|^2$$

attempts 에서 지정한 회수 만큼 초기 중심값을 바꾸어가면서 가장 적은 compactness를 갖는 클러스터링 결과를 물색한다.

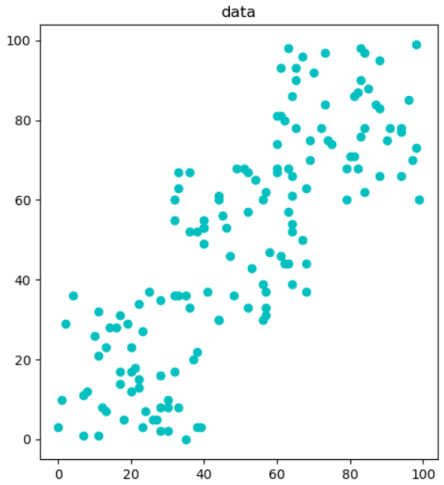
- *retval, bestLabels, centers*=**cv.kmeans**(*data*, K, None, criteria, attempts, flags)

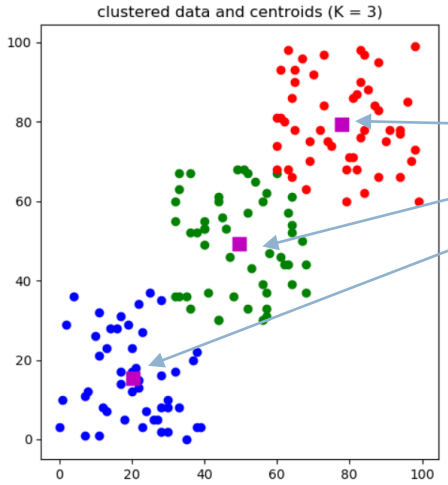
bestLabels
: Input/output integer array that stores the cluster indices for every sam ple. 0-based cluster index for the *data*

centers
: Output matrix of the cluster centers, one row per each cluster center.

6. K(=3) means clustering 분석

k_means_clustering_k_3.py





각 클러스터의 중점

kmeans() 함수의 반환값, compactness,
ret= $\sum_i \|samples_i - centers_{labels_i}\|^2$

10번의 초기값 설정: 이 만큼의 시도를 해본 후 가장 적은 compactness(편차)를 내는 클러스터링 결과를 반환한다.

3개로 그룹핑

```
data: <class 'numpy.ndarray'>, shape=(150, 2), len=150
ret, label, center = cv2.kmeans(data, 3, None, criteria,
                                10, cv2.KMEANS_RANDOM_CENTERS)
ret: 39487.91
label: type=<class 'numpy.ndarray'>, shape=(150, 1)
center: type=<class 'numpy.ndarray'>, shape=(3, 2)
```

data의 각 원소가 배정 받은 레이블을 저장

```
A = data[label.ravel() == 0]
B = data[label.ravel() == 1]
C = data[label.ravel() == 2]
```

⇨

```
A: shape=(50, 2)
B: shape=(52, 2)
C: shape=(48, 2)
```

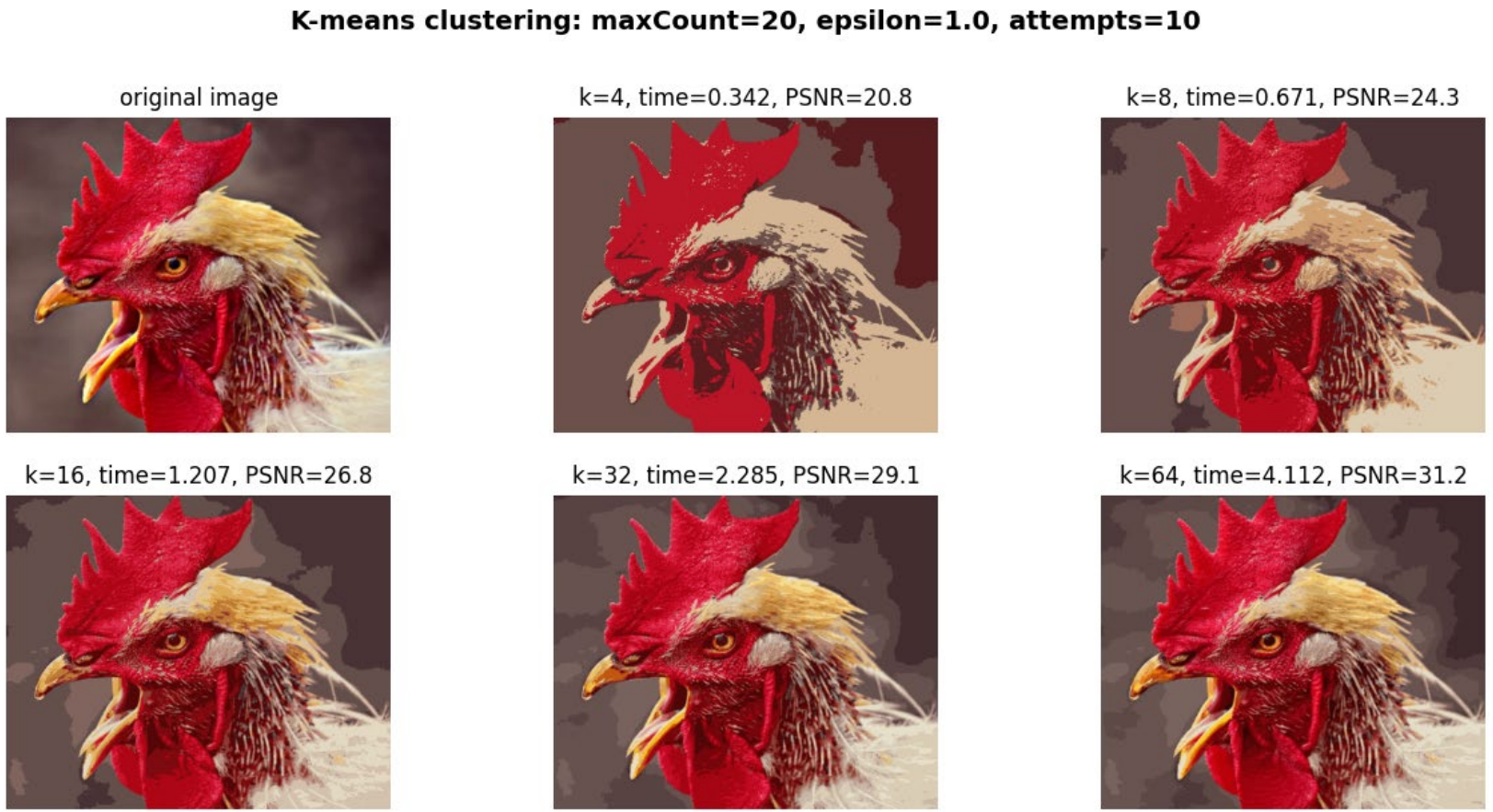
7. k-means clustering 응용

- 색상의 분포를 조사하여 몇 개의 클러스터링으로 단순화함으로써 색상을 표현하기 위한 비트 수를 줄인다. → 본 자료에서 소개
  - 왜 이 일이 필요했는가?
- 비지도학습의 특징을 이용하여 레이블이 없는 데이터에 레이블을 할당하여 이를 기반으로 분류기를 설계한다. → 사례 미 제시

7.1 사례 1

영상존재하는 24비트 색상을 k개의 색상으로 압축: k개의 군집화
각 군집의 중심은 (b, g, r) 값을 갖고 있다.

2_1_k_means_color_quantization.py

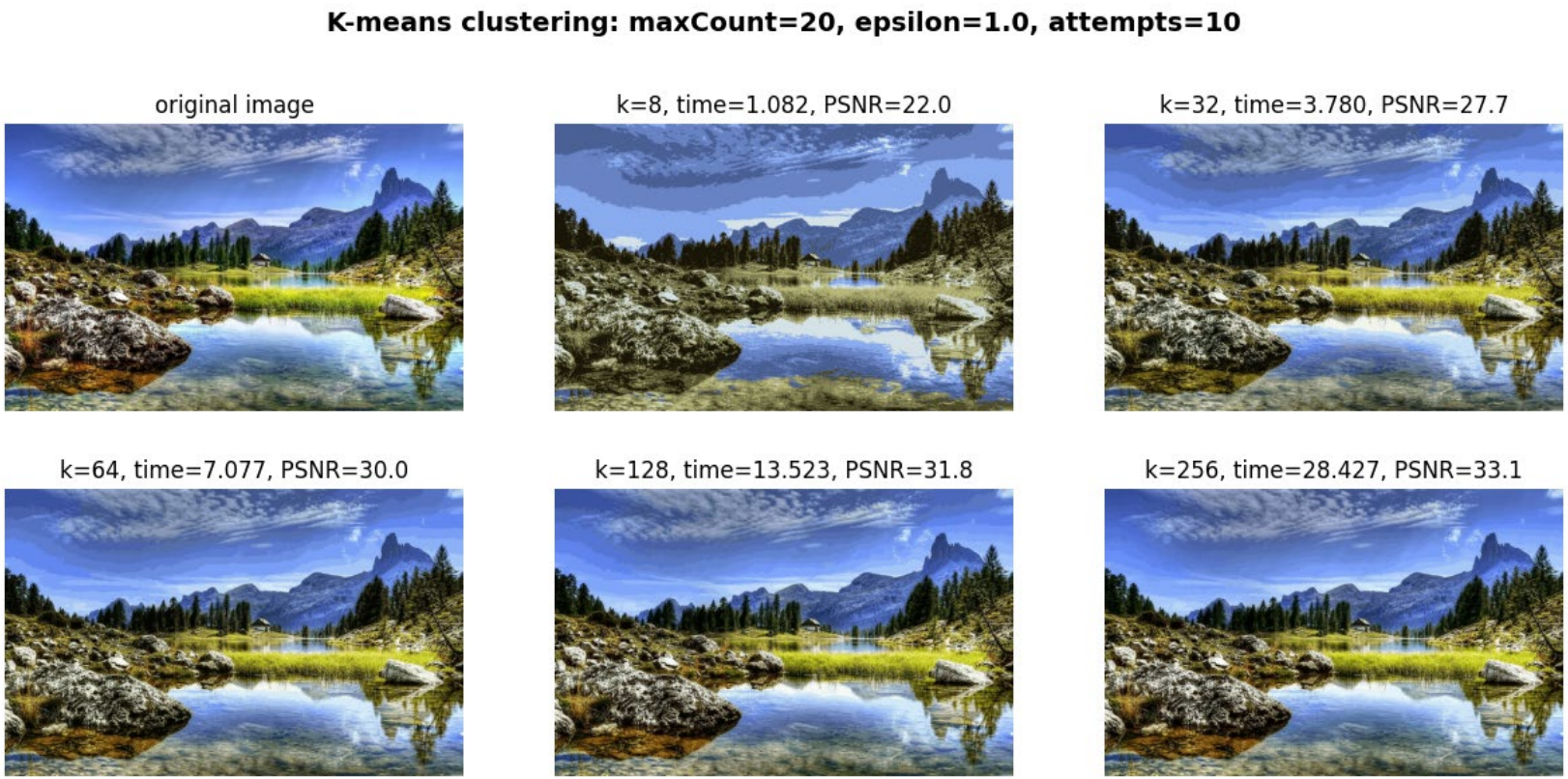



```
center = np.uint8(center)          # center.shape=(k, 3). k개의 센터를 정수형으로 바꿈.
result = center[label.flatten()]   # label.flatten().shape=(가로X세로,) result.shape=(가로X세로, 3)
출력_영상 = result.reshape(img.shape) # 입력 영상과 같은 shape=(row, column, 3)로 만든다.
```

7.2 사례 2

영상존재하는 24비트 색상을 k개의 색상으로 압축: k개의 군집화
각 군집의 중심은 (b, g, r) 값을 갖고 있다.

2_1_k_means_color_quantization.py



K=256: 압축이 없다는 가정하에
영상 데이터는 1/3으로 됨

7.3 처리과정 요약

data = np.float32(image).reshape((-1, 3))
# 맨 뒤자리 차원 3은 고정하고 나머지는 정리하여 1차원으로 바꾼다.

3채널 영상 전체(면적이 A라 하자)의 화소를 데이터로 지정하면 data.shape=(A, 3)이다.

```
ret, label, center = cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
```

k=2: data.shape=(400000, 3)
k=2: center.shape=(2, 3)
k=3: data.shape=(400000, 3)
k=3: center.shape=(3, 3)
k=6: data.shape=(400000, 3)
k=6: center.shape=(6, 3)

클러스터의 갯수와 같은 수의
중심값이 반환된다.
이는 (r, g, b)에 따라 갖고 있어서
두번째 차원이 3이다.

각 화소(r, g, b)의 값을 배정받은
레이블의 중앙값으로 매핑시킨다.

center = np.uint8(center) 좌표의 정수 변환

```
result = center[label.flatten()]
result = result.reshape(img.shape)
psnr = cv2.PSNR(image, result)
```

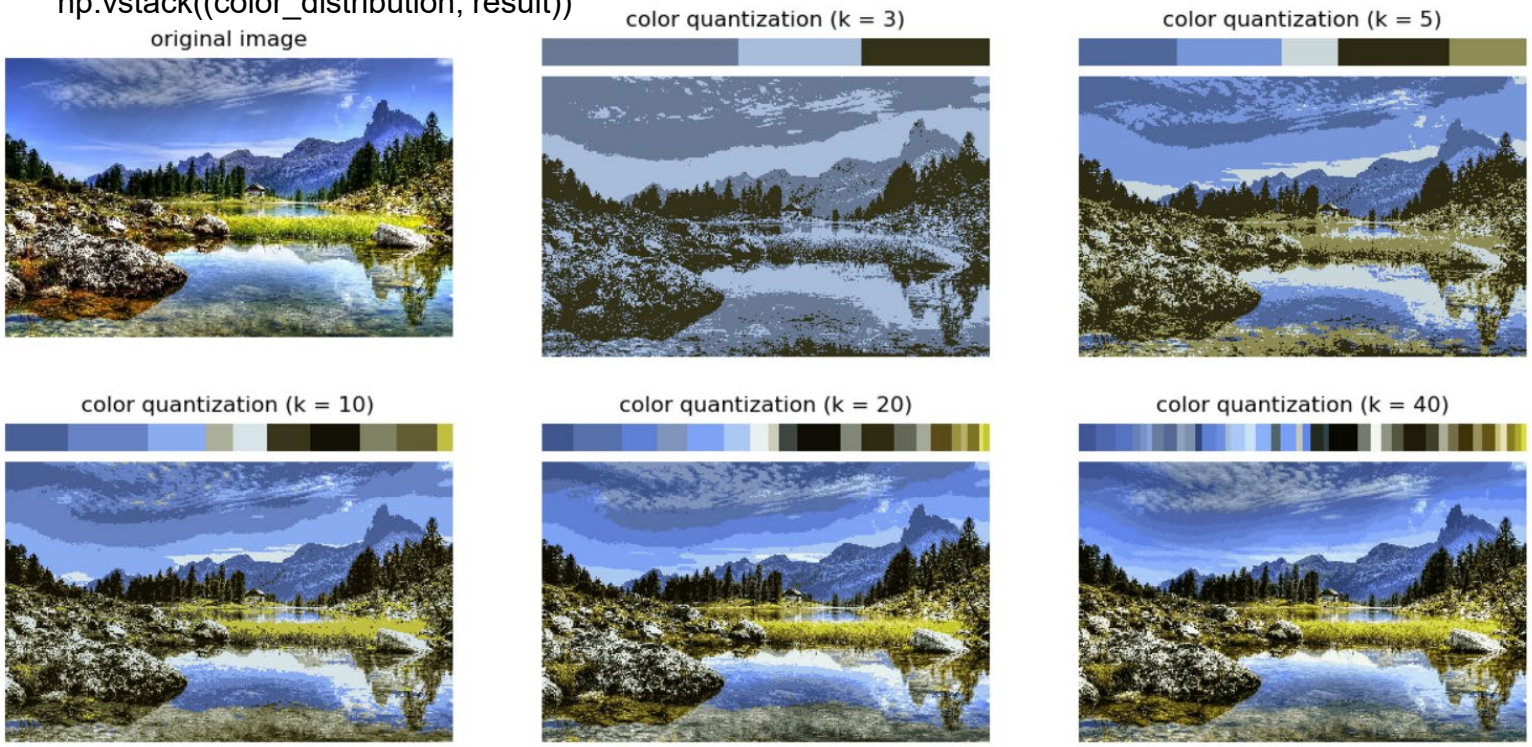
원래 영상과 같이 3차원의 영상으로 복원
면적x 3 => 행x열x3

두 영상의 PSNR을 연산한다.

7.4 레이블의 분포도 표현

2_2_k_means_color_quantization_distribution.py

```
color_distribution = np.ones((desired_height, desired_width, 3), dtype="uint8") * 255 # 흰색 띠를 정의
cv2.rectangle(color_distribution, (int(start), 0), (int(end), desired_height_colors), center[key].tolist(), -1)
np.vstack((color_distribution, result))
```



각 k에 대하여 화소의 수량이 많은 순으로 레이블을 재배치 하여 출력한 결과.

collection.Counter(): 중복된 데이터가 저장된 배열을 인자로 넘기면 각 원소의 빈도 순서대로 사전형 자료처럼 저장된 객체를 반환한다.

```
k=3: counter=Counter({0: 175300, 2: 114788, 1: 109912})
k=5: counter=Counter({0: 99656, 2: 92850, 3: 88215, 4: 68037, 1: 5
k=10: counter=Counter({5: 71295, 9: 56275, 1: 51161, 8: 44675, 0:
k=20: counter=Counter({16: 43130, 12: 38141, 9: 33107, 18: 32526,
k=40: counter=Counter({38: 25871, 23: 19068, 32: 17077, 9: 15989,
```

참고 문헌

K-means Clustering 알고리즘

- 위키백과: 개념적 설명, 알고리즘의 목표, 알고리즘의 설명 및 애니메이션, K-means clustering 알고리즘의 한계점

OpenCV machine learning Tutorials -python, K-means Clustering

opencv machine learning tutorials

- docs.opencv.org => main page => opencv-python tutorials -> machine learning

•**K-Nearest Neighbour**

•Learn to use kNN for classification Plus learn about handwritten digit recognition using kNN

•**Support Vector Machines (SVM)**

•Understand concepts of SVM

•**K-Means Clustering**

•Learn to use K-Means Clustering to group data to a number of clusters. Plus learn to do color quantization using K-Means Clustering

부록

- A.1 PSNR
- A.2 Indexed Color
- A.3 미술 작품 활용

A.1 PSNR

- **PSNR (Peak Signal-to-Noise Ratio)**
  - 영상의 품질을 측정하는 방법 중의 하나로 영상에서 손괴가 일어난 정도를 수치로 표현한다. 즉, 원본(A)과 새로 만든 영상(B)와의 영상의 손괴 정도를 수치로 나타낸다.
  - 단위는 [dB], decibel: **데시벨(decibel, dB)**
  - 영상의 경우 보통 40dB 이상 넘어가면 육안으로는 두 영상의 차이를 분간하기 어려워진다.
- PSNR=10 log(신호전력/ 잡음전력)
  - 전력의 개념을 각 계조치의 제곱으로 대용
  - Peak란 말은 신호전력이 최대일 때를 일컫는 말. 따라서 신호 255의 제곱(=255^2)을 사용한다.

$$PSNR = 10 \cdot \log_{10}\left(\frac{255^2}{MSE}\right)[dB]$$

$$MSE = \frac{1}{NM}\sum_{x=1}^{N}\sum_{y=1}^{M}\left(\ f(x,y) - g(x,y)\ \right)^2$$

  - F(x,y) : 좌표(x,y)의 원본영상 픽셀값. 영상의 크기는 가로*세로=N*M.
  - G(x,y) : 좌표(x,y)의 비교영상 픽셀값. 영상의 크기는 가로*세로=N*M.
  - MSE : Mean Square Error. 두 영상의 오차의 제곱을 영상의 면적으로 나눔.
- PSNR을 측정하기 위해 OpenCV에서는 다음 함수를 지원된다.
  - psnr = cv.PNSR(원본_영상A, 측정대상_영상_B)
  - 같은 shape이어야 함. 영상들의 자리를 바꾸어도 됨.

A.2 Indexed Color

- 칼라 팔레트를 사용하는 파일의 색상 정보 표현 기법
  - 8비트 픽셀 정보의 실제 색상 정보는 칼라 팔레트에 정의되어 있다.
  - 이 팔레트는 파일의 헤더 부분에 정의되어 있다.
  - 8비트는 Video Card에서 실제 색상으로 변환과정에 RGB용 3개의 DAC를 거친다.
  - 색상의 분포를 잘 활용하면 데이터를 줄이면서 제법 실사와 가까운 영상을 만들어 낼 수 있다.

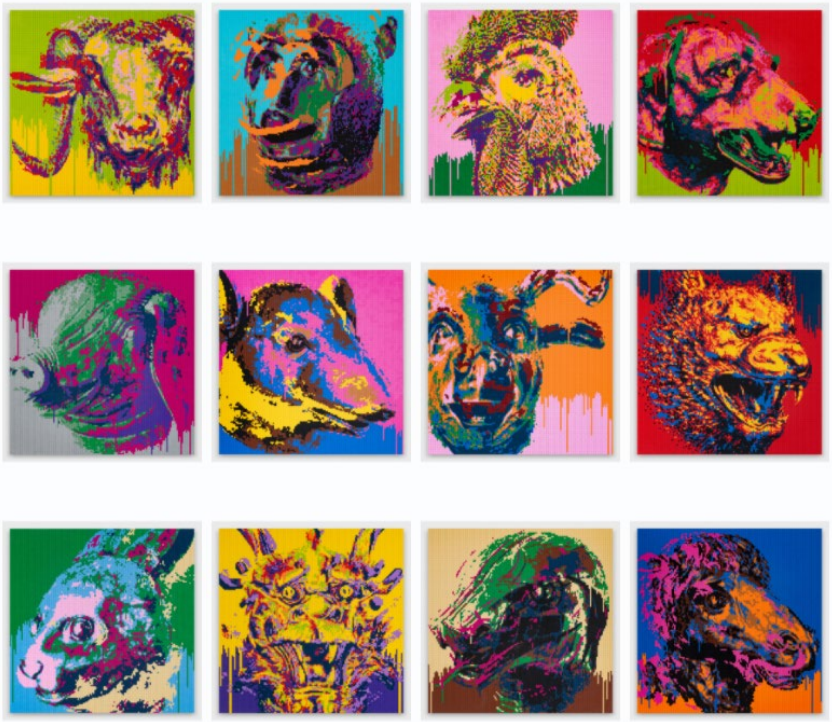
Color Palette (Registers of RAM DAC)

https://en.wikipedia.org/wiki/Indexed_color

- 인덱스 색상 지원 그래픽 파일 형식
  - Photoshop(PSD), BMP, DICOM(Digital Imaging and Communications in Medicine), GIF, Photoshop EPS, 대용량 문서 형식(PSB), PCX, Photoshop PDF, Photoshop Raw, Photoshop 2.0, PICT, PNG, Targa® 또는 TIFF 형식
  - 각종 그래픽 파일 형식 소개
    - URL: https://helpx.adobe.com/kr/photoshop/using/file-formats.html
  - **Web에서 PNG , GIF, JPEG , SVG 중 어떤 것을 사용하면 좋을까요?**
- 인덱스 칼라 파일의 Python 코딩
  - https://stackoverflow.com/questions/33022983/read-in-an-indexed-color-image-in-python
  - https://pillow.readthedocs.io/en/latest/
  - How to reduce color palette with PIL
-

A.3 미술 작품 활용 - Weiwei

- Ai Weiwei's Zodiac series



국립현대미술관 전시

아마추어 기자의 소개 기사

Weiwei simulation

original image

quantized image

color quantization (k = 16)