TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

CS231. Nhập môn Thị giác máy tính

Image segmentation – Part 02

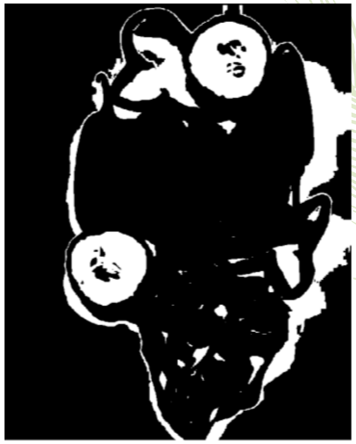
Mai Tiến Dũng

Clustering-Based Segmentation

(TT)

K-means on image pixels

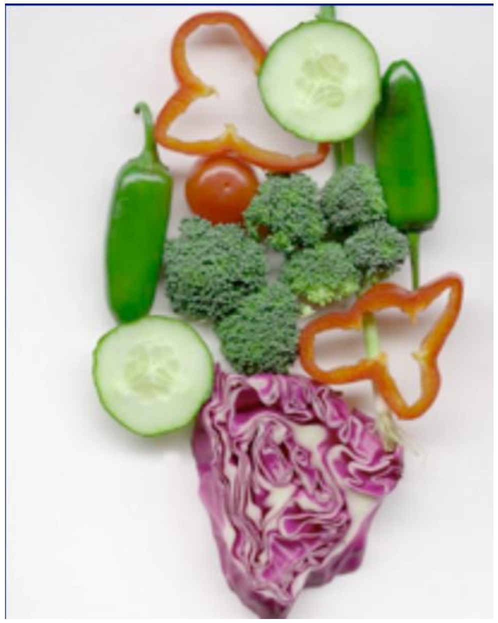
- What is wrong?
- Pixel position
  - Nearby pixels are likely to belong to the same object
  - Far-away pixels are likely to belong to different objects
- How do we incorporate pixel position?
  - Instead of representing each pixel as (r,g,b)
  - Represent each pixel as (r,g,b,x,y)

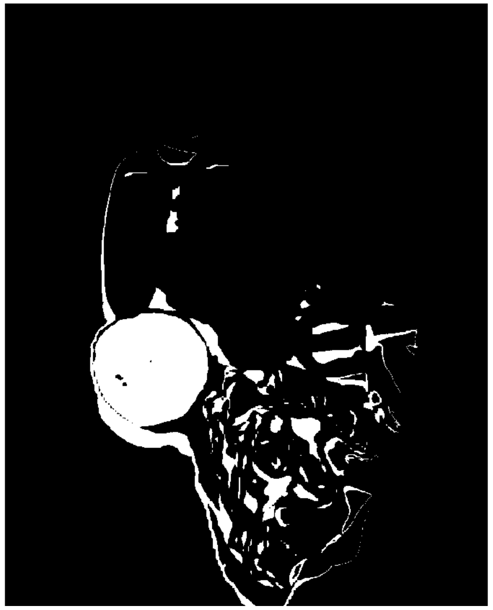


K-means on image pixels

- Representing each pixel as (r,g,b)
  - – Áp dụng cho 4 ảnh: vegetables.jpg, hand.jpg, thuoc.jpg và dogcat.jpg
  - – Represent each pixel as (r,g,b,x,y)
    - Áp dụng cho ảnh: vegetables.jpg

K-means on image pixels+position



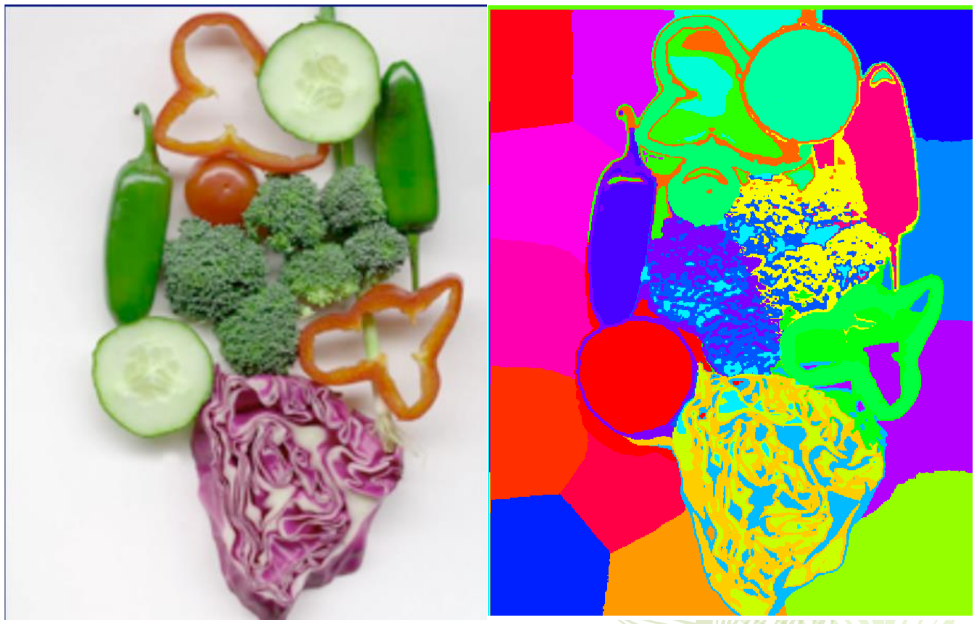


The issues with k-means

- Captures pixel similarity but
  - Doesn’t capture continuity of contours
  - Captures near/far relationships only weakly
  - Can merge far away objects together
- Requires knowledge of k!
- Can it deal with texture?

Oversegmentation and superpixels

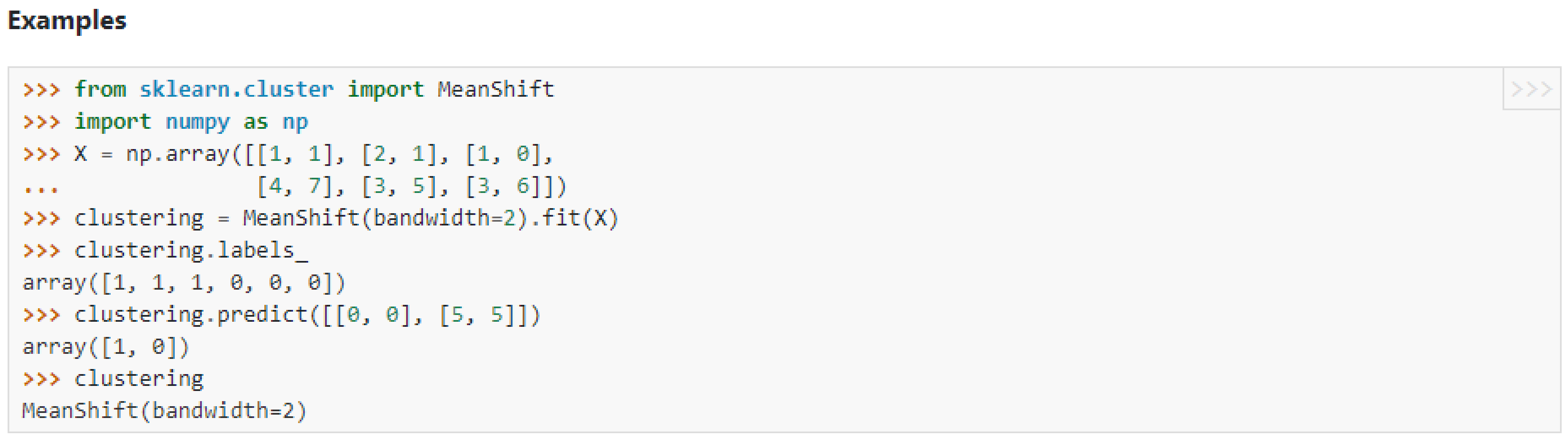
- We don't know k. What is a safe choice?
- Idea: Use large k
  - Can potentially break big objects, but will hopefully not merge unrelated objects
  - Later processing can decide which groups to merge
  - Called *superpixels*



MeanShift

from sklearn.cluster import MeanShift

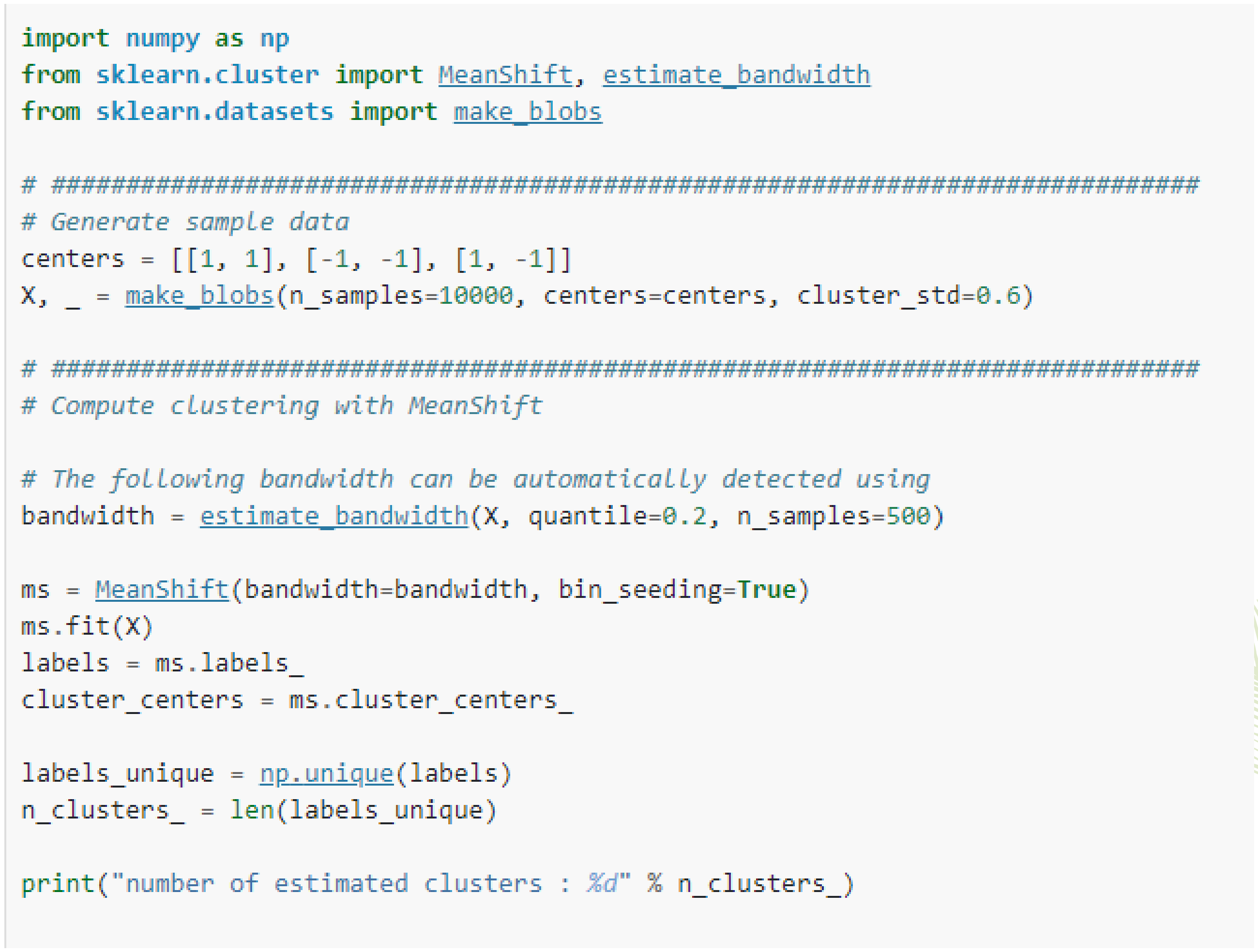
https://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html



Examples

```
>>> from sklearn.cluster import MeanShift
>>> import numpy as np
>>> X = np.array([[1, 1], [2, 1], [1, 0],
...               [4, 7], [3, 5], [3, 6]])
>>> clustering = MeanShift(bandwidth=2).fit(X)
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
>>> clustering.predict([[0, 0], [5, 5]])
array([1, 0])
>>> clustering
MeanShift(bandwidth=2)
```

MeanShift



```python
import numpy as np
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets import make_blobs

# #############################################################################
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, _ = make_blobs(n_samples=10000, centers=centers, cluster_std=0.6)

# #############################################################################
# Compute clustering with MeanShift

# The following bandwidth can be automatically detected using
bandwidth = estimate_bandwidth(X, quantile=0.2, n_samples=500)

ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

print("number of estimated clusters : %d" % n_clusters_)
```

Fuzzy Clustering

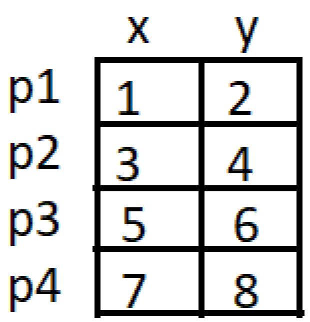
- Fuzzy Clustering is a hard clustering type while Partitioning Clustering is called soft.
- The reason for that is while in Partitioning Clustering, 1 data point may have only in 1 cluster, in Fuzzy Clustering we have the probabilities of a data point for each cluster and they may belong to any cluster at this probability level.

Fuzzy C-Mean Clustering

1. Initialize the probability matrix randomly. So, assign weights to each data — cluster pair which refers to the probability of being in cluster C for data X.
2. Calculate the center of clusters (centroids),
3. Calculate new probabilities according to the new center of clusters.
4. Repeat 2. and 3. steps until the centers doesn’t change or for a given iteration number

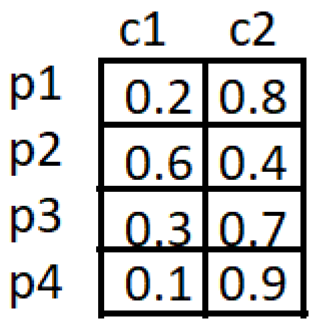
Fuzzy C-Mean Clustering

- We have 4 data points p1, p2, p3, p4 2-dimensional, so we have x and y coordinates of the points and we want to group them into 2 clusters.
- Step 1: We initialize the weight matrix randomly:



	x	y
p1	1	2
p2	3	4
p3	5	6
p4	7	8

data



	c1	c2
p1	0.2	0.8
p2	0.6	0.4
p3	0.3	0.7
p4	0.1	0.9

weight (probability) matrix

Fuzzy C-Mean Clustering

- Step 2: We calculate the centroids of clusters according to that initial probabilities with the following formula:

$$\text{Centroid}_j = \frac{\sum_{i=1}^{n} w_{ij}^{m} \times p_i}{\sum_{i=1}^{n} w_{ij}^{m}}$$

n = number of points
j = cluster index
m = fuzzy parameter
w_{ij} = weight of j. cluster for i. point
p_i = i. point

Note that **fuzzy parameter** is something we should choose like Cluster number and it can be chosen between $1 < m < \infty$

Fuzzy C-Mean Clustering

- Let’s apply the formula for our example case by choosing **m = 2** :

$$\sum_{i=1}^{n} w_{i1}^{m} \times p_i = (0.2)^2 \times 1 + (0.6)^2 \times 3 + (0.3)^2 \times 5 + (0.1)^2 \times 7 = 1.64$$

$$\sum_{i=1}^{n} w_{i1}^{m} = (0.2)^2 + (0.6)^2 + (0.3)^2 + (0.1)^2 = 0.5$$

$$\Rightarrow C_{1x} = 1.64 / 0.5 = 3.28$$

$$\Rightarrow C_1 = (3.28 , 5.08)$$

$$\sum_{i=1}^{n} w_{i1}^{m} \times p_i = (0.2)^2 \times 2 + (0.6)^2 \times 4 + (0.3)^2 \times 6 + (0.1)^2 \times 8 = 2.14$$

$$\sum_{i=1}^{n} w_{i1}^{m} = (0.2)^2 + (0.6)^2 + (0.3)^2 + (0.1)^2 = 0.5$$

$$\Rightarrow C_{1y} = 2.54 / 0.5 = 5.08$$

Fuzzy C-Mean Clustering

- Let’s apply the formula for our example case by choosing **m = 2** :

$$\sum_{i=1}^{n} w_{i2}^{m} \times p_i = (0.8)^2 \times 1 + (0.4)^2 \times 3 + (0.7)^2 \times 5 + (0.9)^2 \times 7 = 9{,}24$$

$$\sum_{i=1}^{n} w_{i2}^{m} = (0.8)^2 + (0.4)^2 + (0.7)^2 + (0.9)^2 = 2{,}1$$

$$\Rightarrow C_{2x} = 9{,}24 / 2{,}1 = 4{,}4$$

$$\Rightarrow C_2 = (4.4 , 5.4)$$

$$\sum_{i=1}^{n} w_{i2}^{m} \times p_i = (0.8)^2 \times 2 + (0.4)^2 \times 4 + (0.7)^2 \times 6 + (0.9)^2 \times 8 = 11{,}34$$

$$\sum_{i=1}^{n} w_{i2}^{m} = (0.8)^2 + (0.4)^2 + (0.7)^2 + (0.9)^2 = 2{,}1$$

$$\Rightarrow C_{2y} = 11{,}34 / 2{,}1 = 5{,}4$$

We obtained our cluster centers, now it’s time to calculate the probabilities of points according to that new cluster centers.

Fuzzy C-Mean Clustering

- Step 3: We calculate new probabilities — weights using the following formula:

$$w_{ij} = \frac{\left(\frac{1}{\text{dist}(p_i, Cj)}\right)^{1/m-1}}{\left(\sum_{k=1}^{n}\frac{1}{\text{dist}(p_i, C_k)}\right)^{1/m-1}}$$

$\longrightarrow$ distance between i. point and j. cluster center

$\longrightarrow$ sum of the distance between i. point and each cluster center

Fuzzy C-Mean Clustering

- Step 3:

$$w11 = \frac{\dfrac{1}{(1-3.28)^2 + (2-5.08)^2}}{\dfrac{1}{(1-3.28)^2 + (2-5.08)^2} + \dfrac{1}{(1-4.4)^2 + (2-5.4)^2}} = \frac{0.06}{0.06 + 0.04} = 0.6$$

probability of "first point belongs to the first cluster"

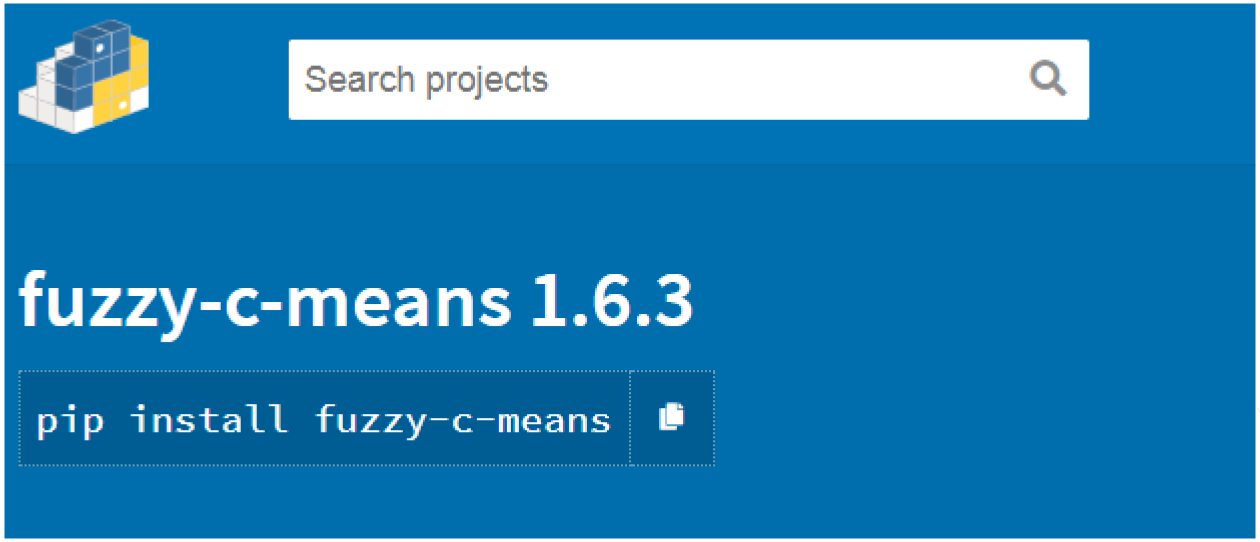
$$w12 = \frac{\dfrac{1}{(1-4.4)^2 + (2-5.4)^2}}{\dfrac{1}{(1-3.28)^2 + (2-5.08)^2} + \dfrac{1}{(1-4.4)^2 + (2-5.4)^2}} = \frac{0.04}{0.06 + 0.04} = 0.4$$

probability of "first point belongs to the second cluster"

$$\vdots$$

fuzzy-c-means

- https://pypi.org/project/fuzzy-c-means/



fuzzy-c-means 1.6.3

```
pip install fuzzy-c-means
```

- pip install fuzzy-c-means

fuzzy-c-means

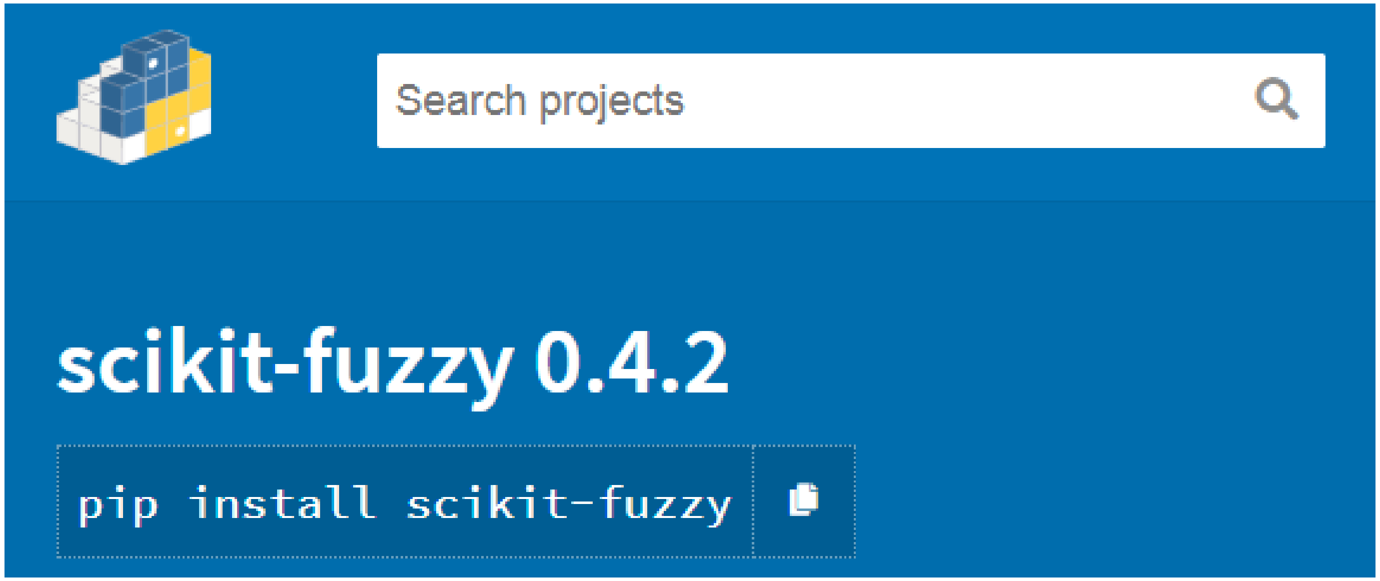
```
from fcmeans import FCM

fcm = FCM(n_clusters=2)
fcm.fit(X)

fcm_labels = fcm.predict(X)
```

scikit-fuzzy

- https://pypi.org/project/scikit-fuzzy/



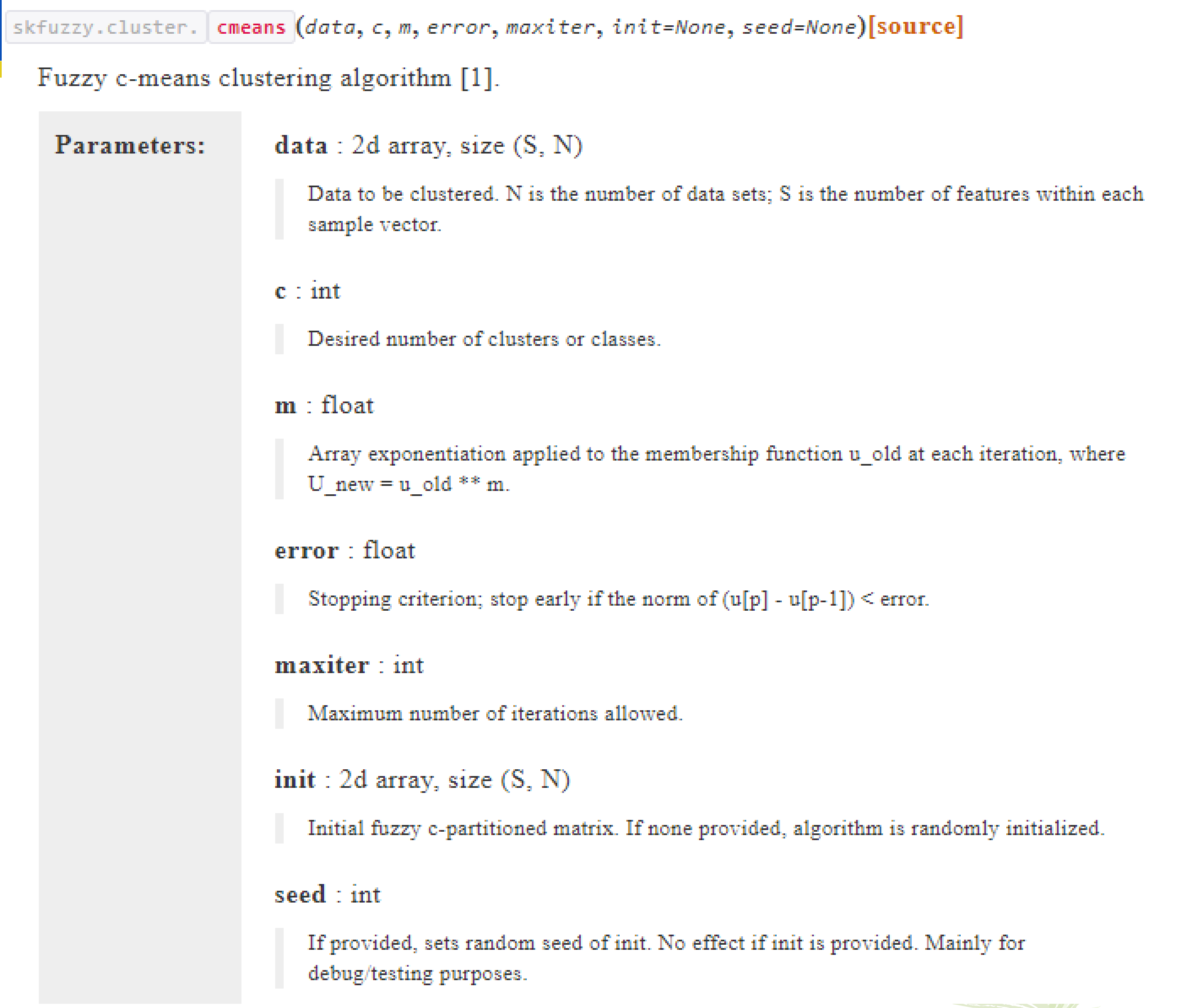
scikit-fuzzy 0.4.2

```
pip install scikit-fuzzy
```

- pip install scikit-fuzzy

- https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.cluster.html

scikit-fuzzy



`skfuzzy.cluster.cmeans(data, c, m, error, maxiter, init=None, seed=None)`[source]

Fuzzy c-means clustering algorithm [1].

Parameters:

data : 2d array, size (S, N)

> Data to be clustered. N is the number of data sets; S is the number of features within each sample vector.

c : int

> Desired number of clusters or classes.

m : float

> Array exponentiation applied to the membership function u_old at each iteration, where U_new = u_old ** m.

error : float

> Stopping criterion; stop early if the norm of (u[p] - u[p-1]) < error.

maxiter : int

> Maximum number of iterations allowed.

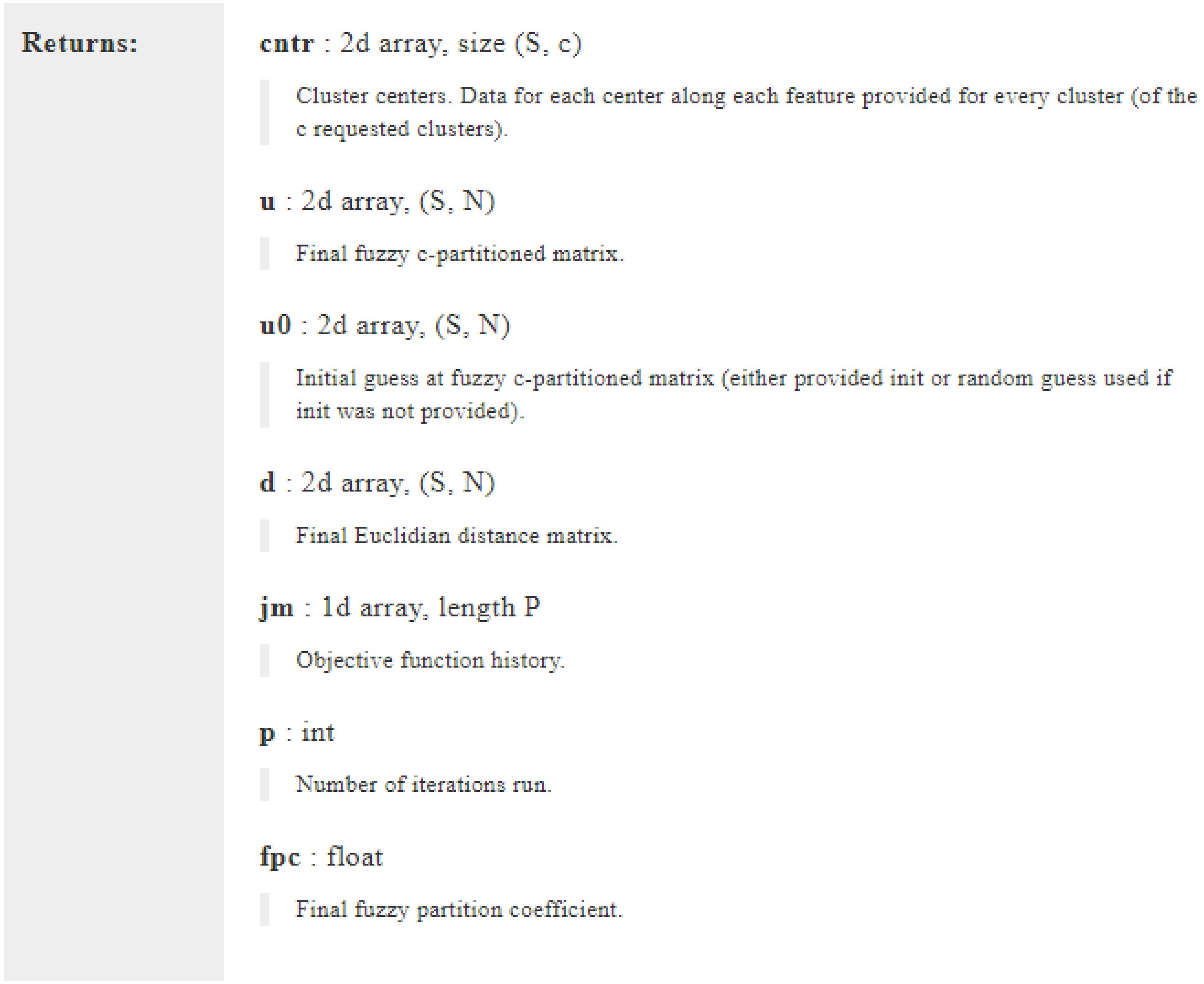
init : 2d array, size (S, N)

> Initial fuzzy c-partitioned matrix. If none provided, algorithm is randomly initialized.

seed : int

> If provided, sets random seed of init. No effect if init is provided. Mainly for debug/testing purposes.

scikit-fuzzy



Returns:

cntr : 2d array, size (S, c)

Cluster centers. Data for each center along each feature provided for every cluster (of the c requested clusters).

u : 2d array, (S, N)

Final fuzzy c-partitioned matrix.

u0 : 2d array, (S, N)

Initial guess at fuzzy c-partitioned matrix (either provided init or random guess used if init was not provided).

d : 2d array, (S, N)

Final Euclidian distance matrix.

jm : 1d array, length P

Objective function history.

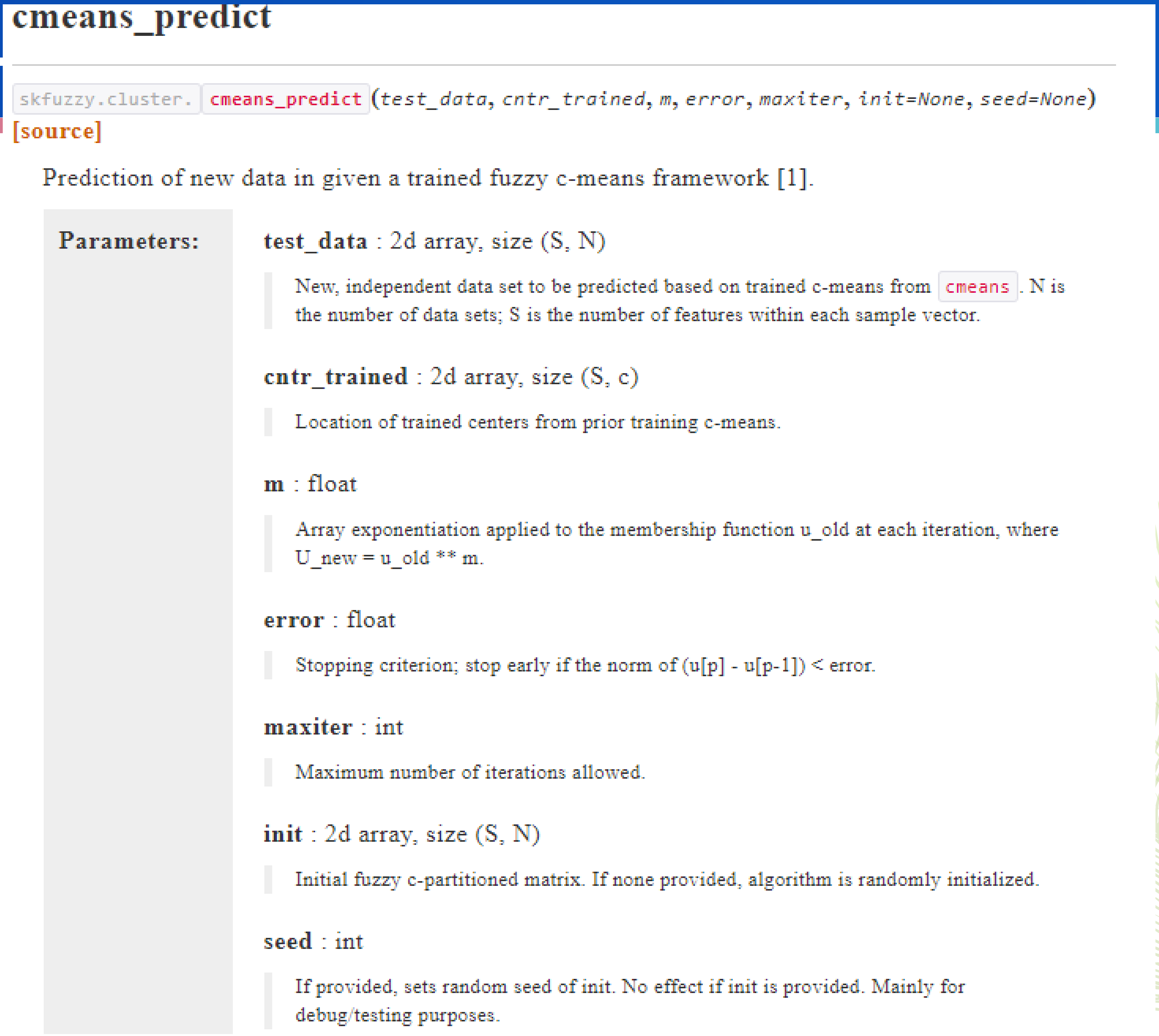
p : int

Number of iterations run.

fpc : float

Final fuzzy partition coefficient.

scikit-



cmeans_predict

`skfuzzy.cluster.cmeans_predict`(*test_data*, *cntr_trained*, *m*, *error*, *maxiter*, *init=None*, *seed=None*)
[source]

Prediction of new data in given a trained fuzzy c-means framework [1].

Parameters:

test_data : 2d array, size (S, N)

New, independent data set to be predicted based on trained c-means from `cmeans`. N is the number of data sets; S is the number of features within each sample vector.

cntr_trained : 2d array, size (S, c)

Location of trained centers from prior training c-means.

m : float

Array exponentiation applied to the membership function u_old at each iteration, where U_new = u_old ** m.

error : float

Stopping criterion; stop early if the norm of (u[p] - u[p-1]) < error.

maxiter : int

Maximum number of iterations allowed.

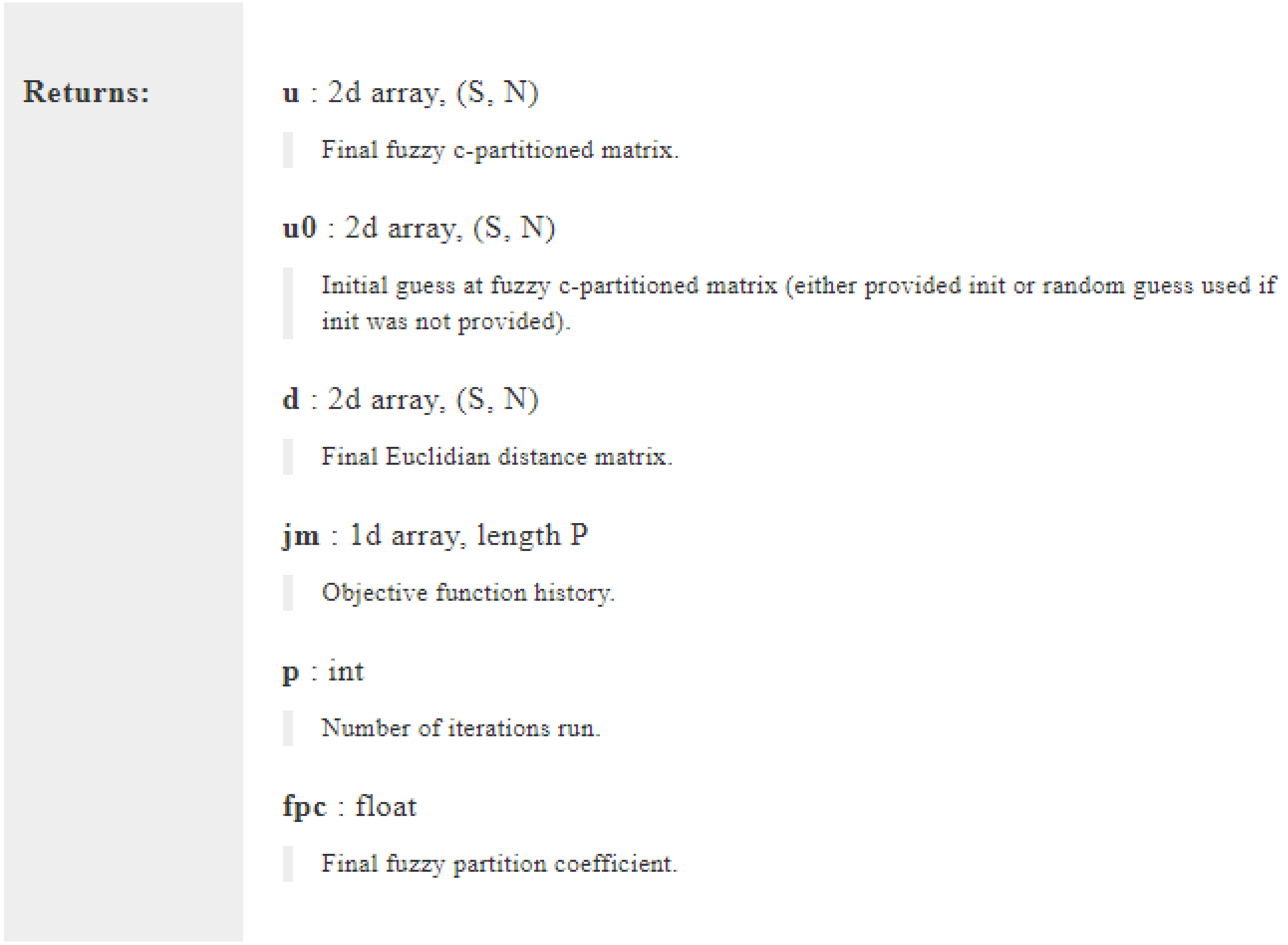
init : 2d array, size (S, N)

Initial fuzzy c-partitioned matrix. If none provided, algorithm is randomly initialized.

seed : int

If provided, sets random seed of init. No effect if init is provided. Mainly for debug/testing purposes.

scikit-fuzzy



Returns:

u : 2d array, (S, N)

Final fuzzy c-partitioned matrix.

u0 : 2d array, (S, N)

Initial guess at fuzzy c-partitioned matrix (either provided init or random guess used if init was not provided).

d : 2d array, (S, N)

Final Euclidian distance matrix.

jm : 1d array, length P

Objective function history.

p : int

Number of iterations run.

fpc : float

Final fuzzy partition coefficient.

scikit-fuzzy

- Ví dụ:

https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html

Region Growing Technique

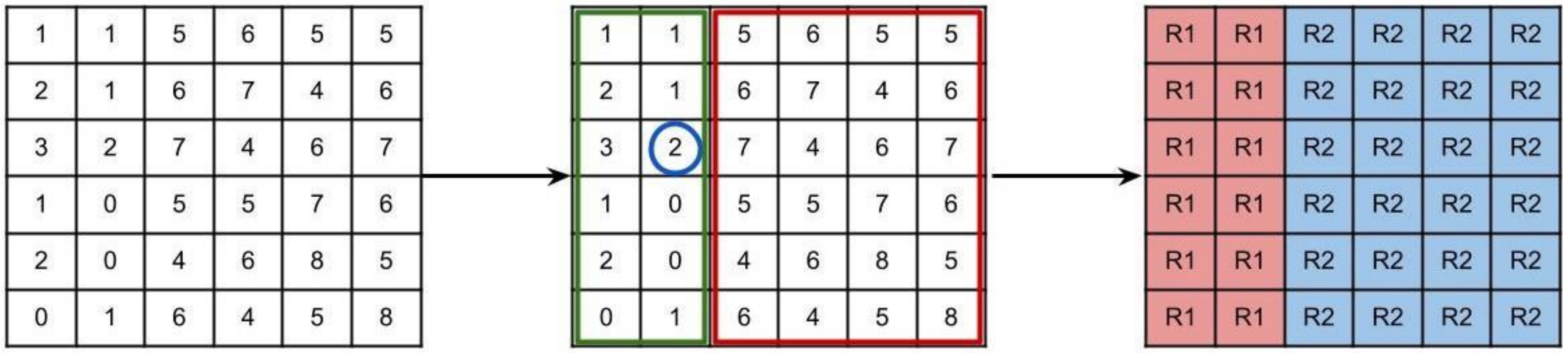
Region Growing Technique

- In the case of the Region growing method, we start with some pixel as the seed pixel and then check the adjacent pixels. If the adjacent pixels abide by the predefined rules, then that pixel is added to the region of the seed pixel and the following process continues till there is no similarity left. This method follows the bottom-up approach. In case of a region growing, the preferred rule can be set as a threshold.

Region Growing Technique

- For example: Consider a seed pixel of 2 in the given image and a threshold value of 3, if a pixel has a value greater than 3 then it will be considered inside the seed pixel region. Otherwise, it will be considered in another region. Hence 2 regions are formed in the following image based on a threshold value of 3.

Region Growing Technique



1	1	5	6	5	5
2	1	6	7	4	6
3	2	7	4	6	7
1	0	5	5	7	6
2	0	4	6	8	5
0	1	6	4	5	8

Original Image

1	1	5	6	5	5
2	1	6	7	4	6
3	2	7	4	6	7
1	0	5	5	7	6
2	0	4	6	8	5
0	1	6	4	5	8

Region growing process with 2 as the seed pixel.

R1	R1	R2	R2	R2	R2
R1	R1	R2	R2	R2	R2
R1	R1	R2	R2	R2	R2
R1	R1	R2	R2	R2	R2
R1	R1	R2	R2	R2	R2
R1	R1	R2	R2	R2	R2

Splitting image into two regions based on a threshold.

Region Splitting and Merging Technique

- In Region splitting, the whole image is first taken as a single region. If the region does not follow the predefined rules, then it is further divided into multiple regions (usually 4 quadrants) and then the predefined rules are carried out on those regions in order to decide whether to further subdivide or to classify that as a region. The following process continues till there is no further division of regions required i.e every region follows the predefined rules.

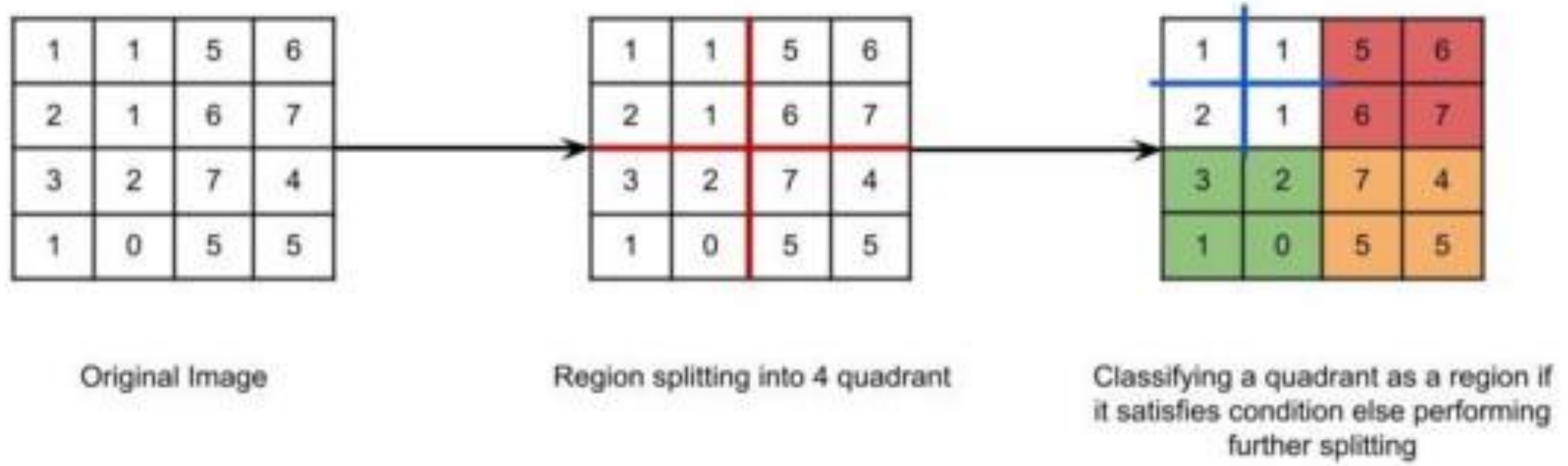
Region Splitting and Merging Technique

- In Region merging technique, we consider every pixel as an individual region. We select a region as the seed region to check if adjacent regions are similar based on predefined rules. If they are similar, we merge them into a single region and move ahead in order to build the segmented regions of the whole image. Both region splitting and region merging are iterative processes. Usually, first region splitting is done on an image so as to split an image into maximum regions, and then these regions are merged in order to form a good segmented image of the original image.

Region Splitting and Merging Technique

- In case of Region splitting, the following condition can be checked in order to decide whether to subdivide a region or not. If the absolute value of the difference of the maximum and minimum pixel intensities in a region is less than or equal to a threshold value decided by the user, then the region does not require further splitting.

Region Splitting and Merging Technique


1 1 5 6
2 1 6 7
3 2 7 4
1 0 5 5
Original Image
Region splitting into 4 quadrant
Classifying a quadrant as a region if it satisfies condition else performing further splitting

$$|Z_{max} - Z_{min}| <= threshold$$

$$Z_{max} \rightarrow Maximum\ pixel\ intensity\ value\ in\ a\ region.$$

$$Z_{min} \rightarrow Minimum\ pixel\ intensity\ value\ in\ a\ region.$$

Tài liệu tham khảo

- https://towardsdatascience.com/image-segmentation-part-1-9f3db1ac1c50
- https://towardsdatascience.com/image-segmentation-part-2-8959b609d268
- https://towardsdatascience.com/image-segmentation-with-clustering-b4bbc98f2ee6
- https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Thực hành

- Representing each pixel as (r,g,b)
  - Áp dụng cho 4 ảnh: vegetables.jpg, hand.jpg, thuoc.jpg và dogcat.jpg
- Represent each pixel as (r,g,b,x,y)
  - Áp dụng cho ảnh: vegetables.jpg và thuoc.jpg

======================

Các thuật toán: MeanShift, FCM