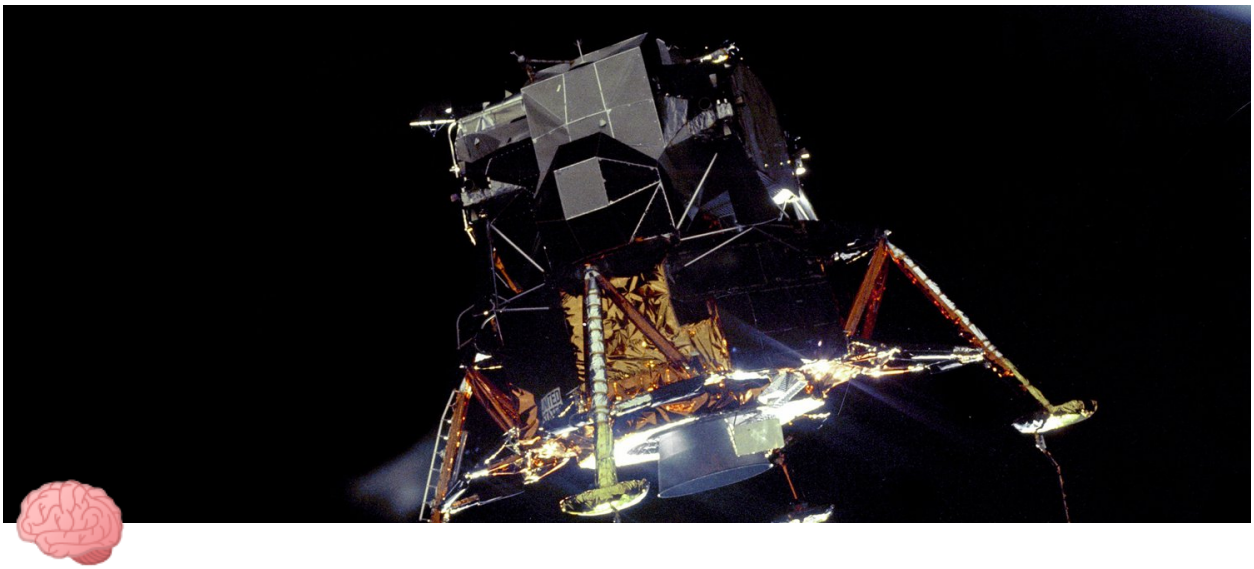

Knowledge Engineering Lab Work

> This paper is prepared by **Nuran Tarlan** from 682.19E group (**student number: 19**)

Knowledge Modeling by using Clustering methods

Clustering is one of the most essential processes in **pattern recognition**, since it plays a key role in finding the **structures in data**.

Fuzzy clustering can be divided into **two** basic groups, namely **fuzzy c-means clustering** (**FCM**) based on **fuzzy partitions**, and **fuzzy hierarchical clustering** method based on **fuzzy equivalence relations**. Other methods derived from the two above were also proposed by researchers.

Fuzzy C-means

The aim of this work is to apply fuzzy c-means clustering model to **image segmentation process**.

> 💡 Texture segmentation problem is one of the fundamental one in the pattern recognition and image processing, such as in remote sensing, medical imaging, textile products inspection, etc.
>
> Given an image with multiple texture regions, fuzzy c-means algorithm can be used to cluster the image vectors into several classes, each corresponding to the different regions.
> The output is the segmented image which can be further processed for noise removal if required.

The fuzzy c-means is one of the most popular ongoing area of the research among all types of researchers including CS, Mathematics and other areas of engineering, as well as all areas of optimization practices.

The algorithm for fuzzy c-means clustering consists of an iterative clustering method whose output is optimal **c** partition, obtained by **minimizing the objective function FCM**.

The basic **FCM** algorithm is formulated as follows:

- **Randomly** initialize the **membership matrix** using the equation inside

$$\Sigma_{j=1}^{c}\mu_{\mathrm{j}}(x_i) = 1 \qquad i = 1,2,\ldots,k$$

▼ Calculate the **centroids** using the equation inside

- $$C_{\mathrm{j}} = \frac{\sum_i [\mu_j(x_i)]^m x_i}{\sum_i [\mu_j(x_i)]^m}$$

▼ Calculate **dissimilarity** between the data **points and centroid** using the **Euclidean** (vector-based) distance using the equation inside

- $$D_i = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

▼ Update the **new membership matrix** using the equation inside

- $$\mu_j(x_j) = \frac{[\frac{1}{d_{ji}}]^{\frac{1}{m-1}}}{\sum_{k=1}^{c}[\frac{1}{d_{ki}}]^{\frac{1}{m-1}}}$$

m her is a **fuzzification parameter** which is in range `[1.25, 2]`

Finally go back to step 2, unless the centroids are not changing!

Problem's Input

my student number in our group is **19**

X	Y	C_1	C_2
1	6	$19 * 0.005$	$1 - 19 * 0.005$
2	5	0.9	0.1
3	8	0.7	0.3
4	4	0.3	0.7
5	7	0.5	0.5
6	9	$19 * 0.007$	$1 - 19 * 0.007$

Implementation of algorithm in pure Python only importing `numpy` & math `dependencies`

```
# Laboratory Work of KE: Knowledge modeling by using clustering method: Fuzzy C-means

from colorama import Fore, Style
import numpy as np
import math

# initialize inputs
STUDENT_NUMBER = 19

x = np.array([1, 2, 3, 4, 5, 6])
y = np.array([6, 5, 8, 4, 7, 9])
centroid1 = np.array([STUDENT_NUMBER * 0.005, 0.9, 0.7, 0.3, 0.5, STUDENT_NUMBER * 0.007])
centroid2 = 1 - centroid1

# represent all initial data
print("\nInitial inputs data:")
print(Fore.GREEN + "x: points ->", x)
print("y: points ->", y)
print("centroid1: points-> ", centroid1)
print("centroid2: points-> ", centroid2, Style.RESET_ALL)

# declare fuzzification parameter m
M = 1.5

centroid1_prev = np.copy(centroid1)
centroid2_prev = np.copy(centroid2)

# repeat process of finding centroids points until you get same result twice
attempt = 1
```

```
while True:
    print(Fore.CYAN + "\nATTEMPT:", attempt, Style.RESET_ALL)
    attempt += 1

    # find the constraints
    centroid1_pow = np.power(centroid1, M)
    centroid2_pow = np.power(centroid2, M)

    constraint1 = (round(np.sum(x * centroid1_pow) / np.sum(centroid1_pow), 2),
                   round(np.sum(y * centroid1_pow) / np.sum(centroid1_pow), 2))
    constraint2 = (round(np.sum(x * centroid2_pow) / np.sum(centroid2_pow), 2),
                   round(np.sum(y * centroid2_pow) / np.sum(centroid2_pow), 2))

    print("constraint1 ->", constraint1)
    print("constraint2 ->", constraint2)

    # find the distance of centroids from constraints
    for i in range(len(x)):
        d1 = round(math.sqrt(np.power(constraint1[0] - x[i], 2) + np.power(constraint1[1]
 - y[i], 2)), 2)
        d2 = round(math.sqrt(np.power(constraint2[0] - x[i], 2) + np.power(constraint2[1]
 - y[i], 2)), 2)
        print(
            Fore.MAGENTA + f"distance of datapoint ({x[i]}, {y[i]}) for: cluster1: {d1} |
 cluster2: {d2}" + Style.RESET_ALL)
        distance_temp = np.power(1 / d1, 1 / (M - 1))
        centroid1[i] = round(distance_temp / (distance_temp + np.power(1 / d2, 1 / (M -
 1))), 3)

    centroid2 = 1 - centroid1

    print("centroid1: new points ->", centroid1)
    print("centroid2: new points ->", centroid2)

    if np.array_equal(centroid1, centroid1_prev):
        break

    centroid1_prev = np.copy(centroid1)

print(Fore.GREEN + f"\n\nFinal points of centroid1: {centroid1} & centroid2: {centroid2}",
Style.RESET_ALL)


def my_floor(floating_num, precision=1):
    return math.floor(floating_num * 10 ** precision) / 10 ** precision
```

Output of executed code above

```
F:\PythonProjects\KnowledgeEngineeringLab1\venv\Scripts\python.exe F:/PythonProjects/Knowl
edgeEngineeringLab1/main.py

Initial inputs data:
x: points -> [1 2 3 4 5 6]
y: points -> [6 5 8 4 7 9]
centroid1: points->  [0.095 0.9   0.7   0.3   0.5   0.133]
centroid2: points->  [0.905 0.1   0.3   0.7   0.5   0.867]

ATTEMPT: 1
constraint1 -> (3.05, 6.24)
constraint2 -> (3.7, 6.68)
distance of datapoint (1, 6) for: cluster1: 2.06 | cluster2: 2.78
distance of datapoint (2, 5) for: cluster1: 1.62 | cluster2: 2.39
distance of datapoint (3, 8) for: cluster1: 1.76 | cluster2: 1.49
distance of datapoint (4, 4) for: cluster1: 2.43 | cluster2: 2.7
distance of datapoint (5, 7) for: cluster1: 2.09 | cluster2: 1.34
distance of datapoint (6, 9) for: cluster1: 4.04 | cluster2: 3.27
centroid1: new points -> [0.646 0.685 0.417 0.552 0.291 0.396]
centroid2: new points -> [0.354 0.315 0.583 0.448 0.709 0.604]

ATTEMPT: 2
constraint1 -> (2.94, 6.03)
constraint2 -> (4.05, 6.96)
distance of datapoint (1, 6) for: cluster1: 1.94 | cluster2: 3.2
distance of datapoint (2, 5) for: cluster1: 1.39 | cluster2: 2.84
distance of datapoint (3, 8) for: cluster1: 1.97 | cluster2: 1.48
distance of datapoint (4, 4) for: cluster1: 2.29 | cluster2: 2.96
distance of datapoint (5, 7) for: cluster1: 2.28 | cluster2: 0.95
distance of datapoint (6, 9) for: cluster1: 4.26 | cluster2: 2.82
centroid1: new points -> [0.731 0.807 0.361 0.626 0.148 0.305]
centroid2: new points -> [0.269 0.193 0.639 0.374 0.852 0.695]

ATTEMPT: 3
constraint1 -> (2.62, 5.69)
constraint2 -> (4.36, 7.29)
distance of datapoint (1, 6) for: cluster1: 1.65 | cluster2: 3.6
distance of datapoint (2, 5) for: cluster1: 0.93 | cluster2: 3.29
distance of datapoint (3, 8) for: cluster1: 2.34 | cluster2: 1.53
distance of datapoint (4, 4) for: cluster1: 2.18 | cluster2: 3.31
distance of datapoint (5, 7) for: cluster1: 2.72 | cluster2: 0.7
distance of datapoint (6, 9) for: cluster1: 4.73 | cluster2: 2.37
centroid1: new points -> [0.826 0.926 0.299 0.697 0.062 0.201]
centroid2: new points -> [0.174 0.074 0.701 0.303 0.938 0.799]

ATTEMPT: 4
constraint1 -> (2.39, 5.42)
constraint2 -> (4.6, 7.57)
distance of datapoint (1, 6) for: cluster1: 1.51 | cluster2: 3.93
distance of datapoint (2, 5) for: cluster1: 0.57 | cluster2: 3.66
distance of datapoint (3, 8) for: cluster1: 2.65 | cluster2: 1.66
distance of datapoint (4, 4) for: cluster1: 2.15 | cluster2: 3.62
```

```
distance of datapoint (5, 7) for: cluster1: 3.05 | cluster2: 0.7
distance of datapoint (6, 9) for: cluster1: 5.08 | cluster2: 2.0
centroid1: new points -> [0.871 0.976 0.282 0.739 0.05  0.134]
centroid2: new points -> [0.129 0.024 0.718 0.261 0.95  0.866]

ATTEMPT: 5
constraint1 -> (2.32, 5.32)
constraint2 -> (4.71, 7.7)
distance of datapoint (1, 6) for: cluster1: 1.48 | cluster2: 4.08
distance of datapoint (2, 5) for: cluster1: 0.45 | cluster2: 3.83
distance of datapoint (3, 8) for: cluster1: 2.76 | cluster2: 1.74
distance of datapoint (4, 4) for: cluster1: 2.14 | cluster2: 3.77
distance of datapoint (5, 7) for: cluster1: 3.16 | cluster2: 0.76
distance of datapoint (6, 9) for: cluster1: 5.2 | cluster2: 1.83
centroid1: new points -> [0.884 0.986 0.284 0.756 0.055 0.11 ]
centroid2: new points -> [0.116 0.014 0.716 0.244 0.945 0.89 ]

ATTEMPT: 6
constraint1 -> (2.31, 5.3)
constraint2 -> (4.74, 7.74)
distance of datapoint (1, 6) for: cluster1: 1.49 | cluster2: 4.12
distance of datapoint (2, 5) for: cluster1: 0.43 | cluster2: 3.87
distance of datapoint (3, 8) for: cluster1: 2.79 | cluster2: 1.76
distance of datapoint (4, 4) for: cluster1: 2.13 | cluster2: 3.81
distance of datapoint (5, 7) for: cluster1: 3.18 | cluster2: 0.78
distance of datapoint (6, 9) for: cluster1: 5.23 | cluster2: 1.78
centroid1: new points -> [0.884 0.988 0.285 0.762 0.057 0.104]
centroid2: new points -> [0.116 0.012 0.715 0.238 0.943 0.896]

ATTEMPT: 7
constraint1 -> (2.31, 5.29)
constraint2 -> (4.75, 7.76)
distance of datapoint (1, 6) for: cluster1: 1.49 | cluster2: 4.14
distance of datapoint (2, 5) for: cluster1: 0.42 | cluster2: 3.9
distance of datapoint (3, 8) for: cluster1: 2.8 | cluster2: 1.77
distance of datapoint (4, 4) for: cluster1: 2.13 | cluster2: 3.83
distance of datapoint (5, 7) for: cluster1: 3.19 | cluster2: 0.8
distance of datapoint (6, 9) for: cluster1: 5.23 | cluster2: 1.76
centroid1: new points -> [0.885 0.989 0.286 0.764 0.059 0.102]
centroid2: new points -> [0.115 0.011 0.714 0.236 0.941 0.898]

ATTEMPT: 8
constraint1 -> (2.31, 5.29)
constraint2 -> (4.75, 7.76)
distance of datapoint (1, 6) for: cluster1: 1.49 | cluster2: 4.14
distance of datapoint (2, 5) for: cluster1: 0.42 | cluster2: 3.9
distance of datapoint (3, 8) for: cluster1: 2.8 | cluster2: 1.77
distance of datapoint (4, 4) for: cluster1: 2.13 | cluster2: 3.83
distance of datapoint (5, 7) for: cluster1: 3.19 | cluster2: 0.8
distance of datapoint (6, 9) for: cluster1: 5.23 | cluster2: 1.76
centroid1: new points -> [0.885 0.989 0.286 0.764 0.059 0.102]
centroid2: new points -> [0.115 0.011 0.714 0.236 0.941 0.898]
```

```
Final points of centroid1: [0.885 0.989 0.286 0.764 0.059 0.102] & centroid2: [0.115 0.011
0.714 0.236 0.941 0.898]

Process finished with exit code 0
```