CRTP2 IA et Données

INFO0503

21 novembre 2024

VERPOORTE William

Sommaire

1 Implémentation de FCM avec Kmeans et l'indice de Dunn

```
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from skfuzzy.cluster import cmeans
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# G n ration des donn es
n_samples = 1500
random_state = 170
transformation = [[0.60834549, -0.63667341], [-0.40887718,
   0.85253229]]

X, y = make_blobs(n_samples=n_samples,
   random_state=random_state)
X_aniso = np.dot(X, transformation)  # Anisotropic blobs
X_varied, y_varied = make_blobs(
    n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5],
       random_state=random_state
)  # Unequal variance
X_filtered = np.vstack(
    (X[y == 0][:500], X[y == 1][:100], X[y == 2][:10])
)  # Unevenly sized blobs
y_filtered = [0] * 500 + [1] * 100 + [2] * 10

datasets = [(X, y), (X_aniso, y), (X_varied, y_varied),
   (X_filtered, y_filtered)]
titles = [
    "Mixture of Gaussian Blobs",
    "Anisotropically Distributed Blobs",
```

```
    "Unequal Variance",
    "Unevenly Sized Blobs",
]

from scipy.spatial.distance import cdist
import numpy as np

def calculate_dunn_index(X, labels):
    clusters = np.unique(labels)
    inter_cluster_distances = []
    intra_cluster_diameters = []

    # Calcul des distances inter-clusters
    for i in clusters:
        for j in clusters:
            if i != j:
                cluster_i = X[labels == i]
                cluster_j = X[labels == j]
                dist = cdist(cluster_i, cluster_j,
                   metric='euclidean')
                inter_cluster_distances.append(np.min(dist))

    # Calcul des diam tres intra-cluster
    for i in clusters:
        cluster_i = X[labels == i]
        dist = cdist(cluster_i, cluster_i, metric='euclidean')
        intra_cluster_diameters.append(np.max(dist))

    # Indice de Dunn
    dunn_index = np.min(inter_cluster_distances) /
       np.max(intra_cluster_diameters)
    return dunn_index
```

```
59 # Calcul de l'indice de Dunn pour K-Means
60 for i, (data, ground_truth) in enumerate(datasets):
61     kmeans = KMeans(n_clusters=3, random_state=random_state)
62     kmeans_labels = kmeans.fit_predict(data)
63     dunn_kmeans = calculate_dunn_index(data, kmeans_labels)
64     print(f"Indice de Dunn (K-Means) pour {titles[i]} :
           {dunn_kmeans:.4f}")
65
66
67
68 # Fonction pour appliquer Fuzzy C-Means
69 def apply_fcm(X, n_clusters):
70     cntr, u, _, _, _, _, _ = cmeans(X.T, c=n_clusters, m=2.0,
           error=0.005, maxiter=1000)
71     labels = np.argmax(u, axis=0)
72     return labels
73
74 # Cr ation des graphiques K-Means
75 fig_kmeans, axs_kmeans = plt.subplots(2, 2, figsize=(12, 12),
       constrained_layout=True)
76 fig_kmeans.suptitle("R sultats de K-Means", fontsize=16,
       y=0.95)
77
78 for i, (data, ground_truth) in enumerate(datasets):
79     row, col = divmod(i, 2)
80     kmeans = KMeans(n_clusters=3, random_state=random_state)
81     kmeans_labels = kmeans.fit_predict(data)
82     axs_kmeans[row, col].scatter(data[:, 0], data[:, 1],
           c=kmeans_labels, cmap="viridis")
83     axs_kmeans[row, col].set_title(titles[i])
84
85 # Cr ation des graphiques Fuzzy C-Means
86 fig_fcm, axs_fcm = plt.subplots(2, 2, figsize=(12, 12),
       constrained_layout=True)
```

```
87  fig_fcm.suptitle("R sultats de Fuzzy C-Means", fontsize=16,
        y=0.95)
88
89  for i, (data, ground_truth) in enumerate(datasets):
90      row, col = divmod(i, 2)
91      fcm_labels = apply_fcm(data, n_clusters=3)
92      axs_fcm[row, col].scatter(data[:, 0], data[:, 1],
            c=fcm_labels, cmap="viridis")
93      axs_fcm[row, col].set_title(titles[i])
94
95  # Calcul de l'indice de Dunn pour FCM
96  for i, (data, ground_truth) in enumerate(datasets):
97      fcm_labels = apply_fcm(data, n_clusters=3)
98      dunn_fcm = calculate_dunn_index(data, fcm_labels)
99      print(f"Indice de Dunn (FCM) pour {titles[i]} :
            {dunn_fcm:.4f}")
100
101 plt.show()
```

2 Comparaison de Kmeans et FCM

J'utilise un exemple d'utilisation de Kmeans présent sur le site de sklearn et je l'adapte pour FCM.

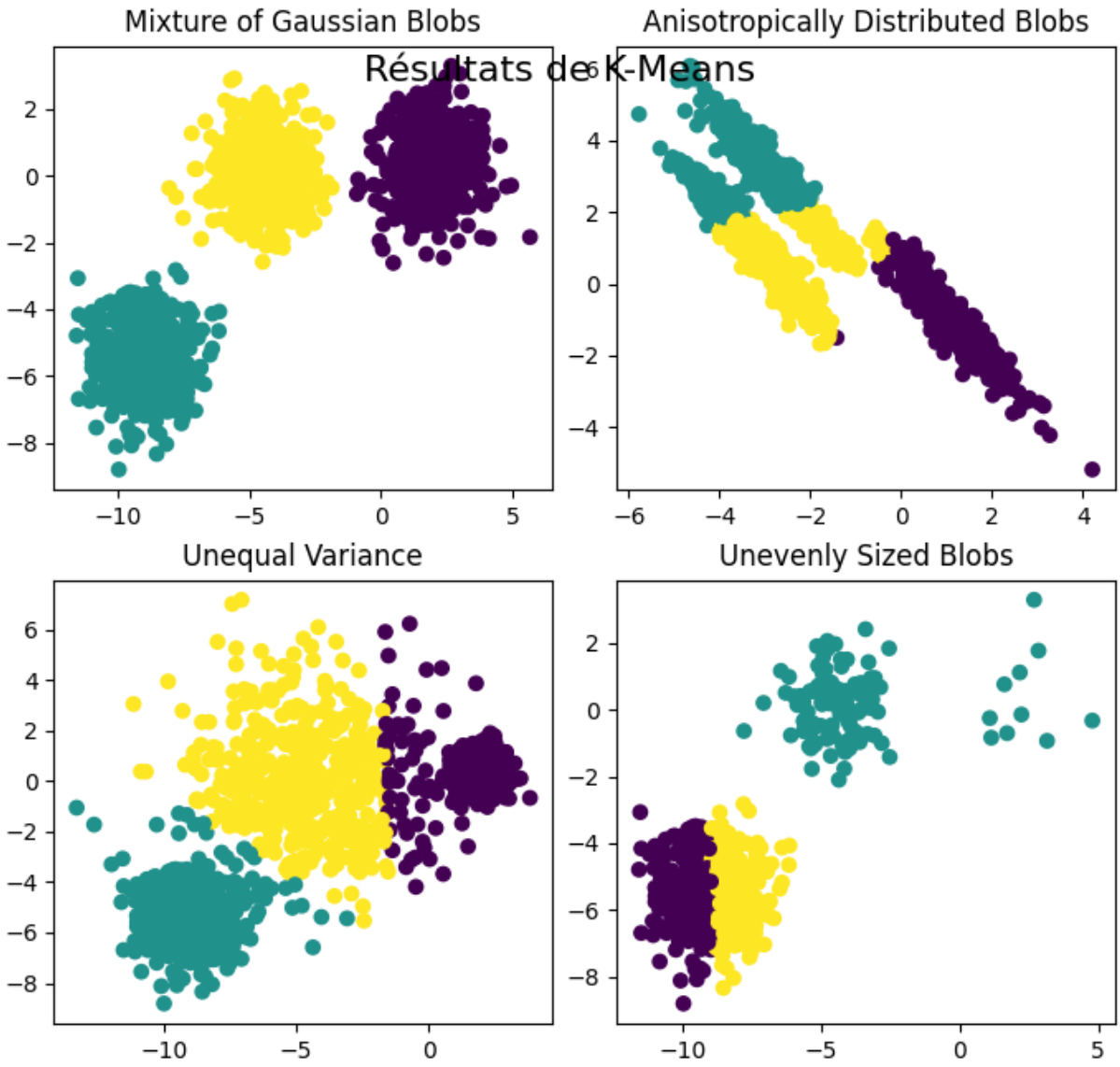

Figure 1: Résultat de Kmeans

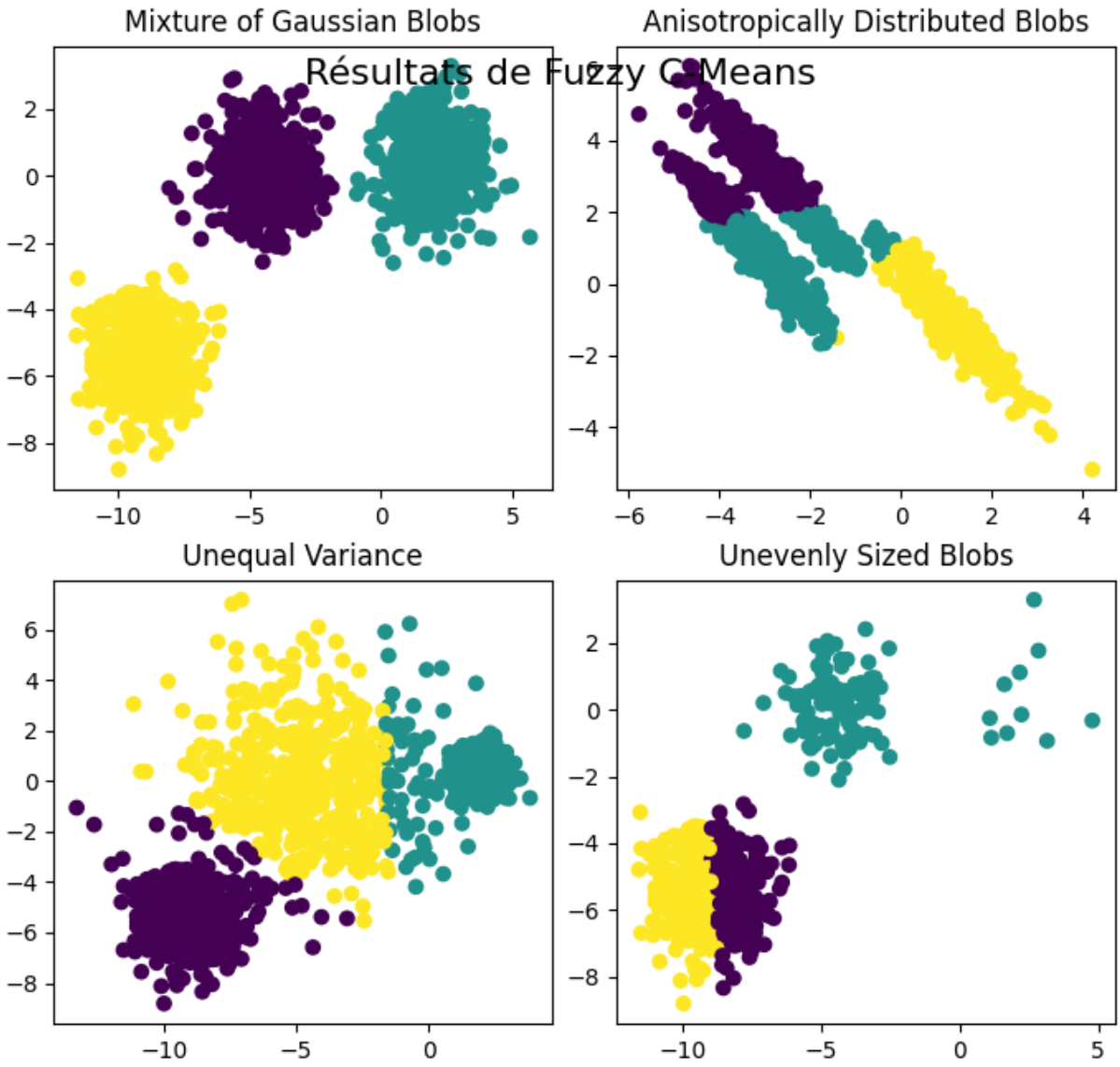

Figure 2: Résultat de FCM

Dans ce cas de figure on remarque une très faible différences entre les 2 algorithmes.

3 Résultat de l'indice de Dunn sur les algorithmes

L'indice de Dunn nous permet de savoir si l'algorithme de clustering utilisé est plus ou moins adapté à la situation. Un indice élevé montre une bonne performance et, à l'inverse, un indice faible montre une faible performance.

```
PS C:\Users\willi\OneDrive\Documents\INFO0503\TP> & C:/Users/willi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/wi
lli/OneDrive/Documents/INFO0503/TP/TP2/FCM.py
Indice de Dunn (K-Means) pour Mixture of Gaussian Blobs : 0.1434
Indice de Dunn (K-Means) pour Anisotropically Distributed Blobs : 0.0035
Indice de Dunn (K-Means) pour Unequal Variance : 0.0088
Indice de Dunn (K-Means) pour Unevenly Sized Blobs : 0.0033
Indice de Dunn (FCM) pour Mixture of Gaussian Blobs : 0.1434
Indice de Dunn (FCM) pour Anisotropically Distributed Blobs : 0.0030
Indice de Dunn (FCM) pour Unequal Variance : 0.0088
Indice de Dunn (FCM) pour Unevenly Sized Blobs : 0.0056
```

Figure 3: Comparaison indice de Dunn

On remarque donc ici que Kmeans est plus adapté pour les blobs gaussiens mais beaucoup moins pour les 3 autres configurations. FCM est un peu plus adapté que Kmeans notemment pour les Sized Blobs.

4 Implémentation code pour données circulaires

```python
import time
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import skfuzzy as fuzz
from scipy.spatial.distance import cdist


# Fonction pour calculer l'indice de Dunn
def calculate_dunn_index(X, labels):
    clusters = np.unique(labels)
    inter_cluster_distances = []
    intra_cluster_diameters = []

    # Calcul des distances inter-clusters
    for i in clusters:
        for j in clusters:
            if i != j:
                cluster_i = X[labels == i]
```

```
22                  cluster_j = X[labels == j]
23                  dist = cdist(cluster_i, cluster_j,
                        metric='euclidean')
24                  inter_cluster_distances.append(np.min(dist))
25
26      # Calcul des diam tres intra-cluster
27      for i in clusters:
28          cluster_i = X[labels == i]
29          dist = cdist(cluster_i, cluster_i, metric='euclidean')
30          intra_cluster_diameters.append(np.max(dist))
31
32      # Indice de Dunn
33      dunn_index = np.min(inter_cluster_distances) /
           np.max(intra_cluster_diameters)
34      return dunn_index
35
36
37  # Classe pour g rer les jeux de donn es
38  class DatasetManager:
39      def __init__(self, n_samples):
40          self.n_samples = n_samples
41          self.datasets = self._create_datasets()
42
43      def _create_datasets(self):
44          noisy_circles = datasets.make_circles(
45              n_samples=self.n_samples, factor=0.2, noise=0.05,
                   random_state=170
46          )
47          return {"noisy_circles": noisy_circles}
48
49      def get_dataset(self, name):
50          return self.datasets.get(name)
51
52
```

```
53 # Classe pour g rer les algorithmes de clustering
54 class ClusteringManager:
55     def __init__(self, n_clusters=2):
56         self.n_clusters = n_clusters
57 
58     def fit_predict(self, algorithm_name, X):
59         if algorithm_name == "Fuzzy C-Means":
60             # Utilisation de Fuzzy C-Means de scikit-fuzzy
61             cntr, u, _, _, _, _, _ = fuzz.cmeans(X.T,
                   self.n_clusters, 2, error=0.005, maxiter=1000)
62             # Choisir les classes bas es sur l'appartenance
                   maximale
63             y_pred = np.argmax(u, axis=0)
64             return y_pred
65         elif algorithm_name == "KMeans":
66             # Utilisation de KMeans de sklearn
67             kmeans = KMeans(n_clusters=self.n_clusters,
                   random_state=42)
68             kmeans.fit(X)
69             return kmeans.labels_
70 
71 
72 # Classe pour la visualisation
73 class PlotManager:
74     def __init__(self, X, clustering_results, dunn_indices):
75         self.X = X
76         self.clustering_results = clustering_results
77         self.dunn_indices = dunn_indices
78 
79     def plot(self):
80         plt.figure(figsize=(10, 6))
81         for i, (name, y_pred) in
               enumerate(self.clustering_results.items()):
82             colors = np.array(
```

```
                [
                    "#377eb8", "#ff7f00", "#4daf4a",
                        "#f781bf", "#a65628",
                    "#984ea3", "#999999", "#e41a1c", "#dede00",
                ]
            )
            plt.subplot(1, 2, i + 1)
            plt.scatter(self.X[:, 0], self.X[:, 1], s=10,
                color=colors[y_pred])
            plt.title(f"{name} Clustering\nDunn Index:
                {self.dunn_indices[name]:.4f}", size=15)
            plt.xlim(-2.5, 2.5)
            plt.ylim(-2.5, 2.5)
            plt.xticks(())
            plt.yticks(())
        plt.tight_layout()
        plt.show()


# Main
if __name__ == "__main__":
    n_samples = 1500
    n_clusters = 6

    # Gestion des donn es
    dataset_manager = DatasetManager(n_samples)
    X, _ = dataset_manager.get_dataset("noisy_circles")

    # Normalisation des donn es
    X = StandardScaler().fit_transform(X)

    # Gestion du clustering
    clustering_manager = ClusteringManager(n_clusters)
    clustering_results = {}
```

```
114     dunn_indices = {}
115
116     # Comparaison entre KMeans et Fuzzy C-Means
117     for algorithm_name in ["KMeans", "Fuzzy C-Means"]:
118         start_time = time.time()
119         y_pred =
                clustering_manager.fit_predict(algorithm_name, X)
120         end_time = time.time()
121
122         # Calcul de l'indice de Dunn pour chaque r sultat de
                clustering
123         dunn_index = calculate_dunn_index(X, y_pred)
124         clustering_results[algorithm_name] = y_pred
125         dunn_indices[algorithm_name] = dunn_index
126
127         print(f"{algorithm_name} clustering completed in
                {end_time - start_time:.2f}s")
128         print(f"{algorithm_name} Dunn Index: {dunn_index:.4f}")
129
130     # Visualisation
131     plot_manager = PlotManager(X, clustering_results,
            dunn_indices)
132     plot_manager.plot()
```

5 Test sur des données circulaires

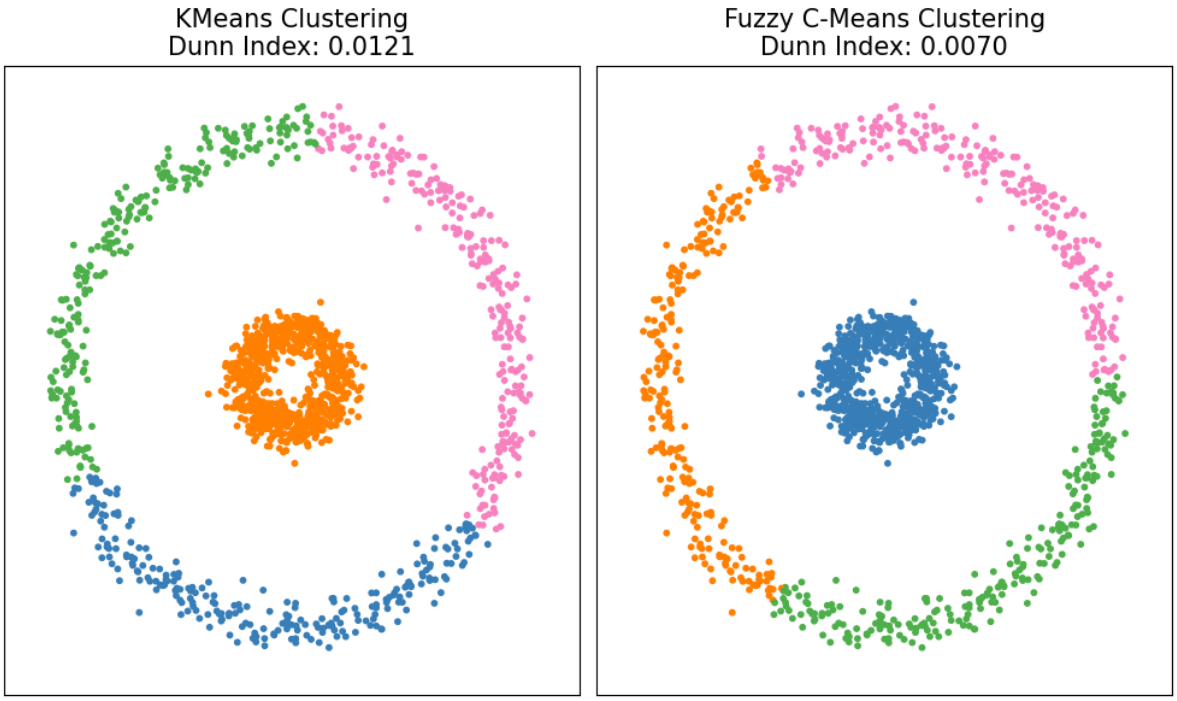

Figure 4: Données circulaires plus éloignées

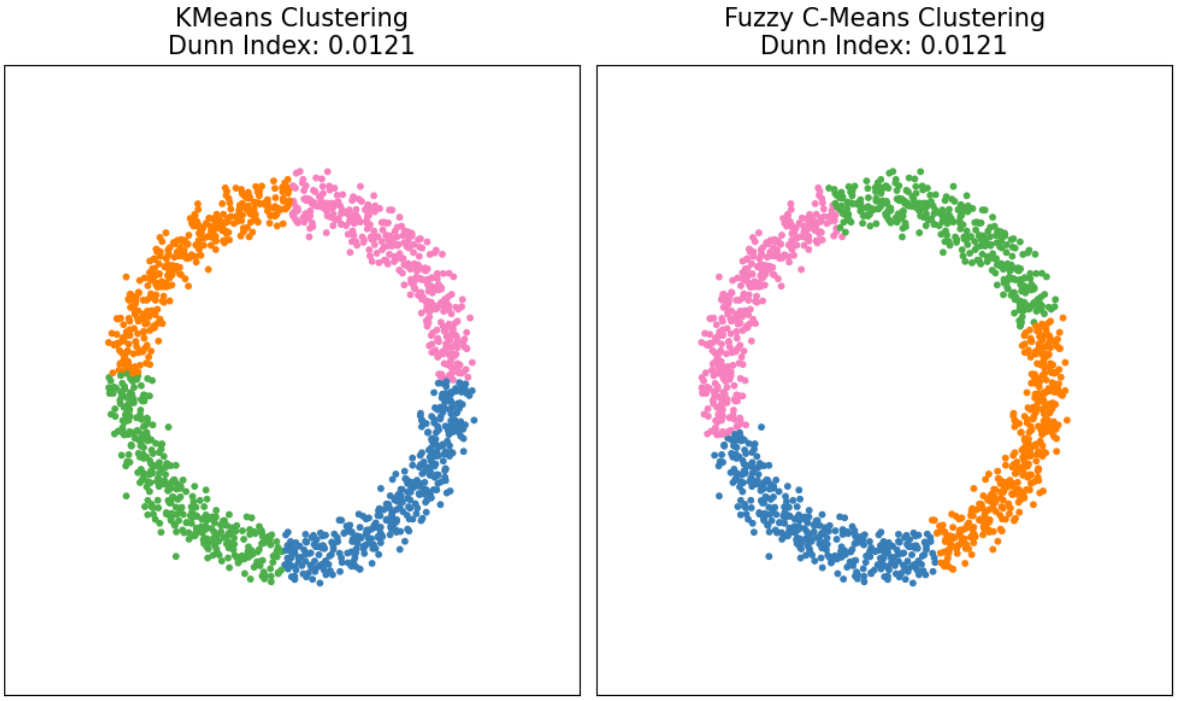

Figure 5: Données circulaires plus proches

Avec les données circulaires, on voit déjà que plus les échantillons sont proches plus FCM devient performants, jusqu'au point où les 2 algorithmes se valent sur le clustering de données circulaires alors que Kmeans est plus performant sur des données plus éloignées. Aussi FCM condenses légèrement plus les données lors du clustering que Kmeans.

6 Implémentation de test sur l'image de Lenna

```
import numpy as np
from PIL import Image
import skfuzzy as fuzz
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt


# Fonction pour calculer l'indice de Dunn avec centroids
def calculate_dunn_index(X, labels):
    clusters = np.unique(labels)
    inter_cluster_distances = []
    intra_cluster_diameters = []

    # Calcul des distances inter-clusters bas es sur les
       centroids
    centroids = []
    for i in clusters:
        cluster_i = X[labels == i]
        centroid_i = np.mean(cluster_i, axis=0)
        centroids.append(centroid_i)

    centroids = np.array(centroids)

    for i in range(len(centroids)):
        for j in range(i + 1, len(centroids)):
            dist = np.linalg.norm(centroids[i] - centroids[j])
```

```
27                inter_cluster_distances.append(dist)
28
29        # Calcul des diam tres intra-cluster bas s sur les
              centroids
30        for i in clusters:
31            cluster_i = X[labels == i]
32            centroid_i = np.mean(cluster_i, axis=0)
33            dist = np.max(np.linalg.norm(cluster_i - centroid_i,
                  axis=1))
34            intra_cluster_diameters.append(dist)
35
36        # Indice de Dunn
37        dunn_index = np.min(inter_cluster_distances) /
              np.max(intra_cluster_diameters)
38        return dunn_index
39
40
41    # Fonction pour effectuer un clustering KMeans
42    def kmeans_clustering(image, n_clusters=3):
43        # Convertir l'image en tableau numpy
44        img_array = np.array(image)
45
46        # V rifier la forme de l'image
47        print(f"Forme de l'image : {img_array.shape}")
48
49        # Si l'image est en niveaux de gris (1 canal), dupliquer
              les valeurs pour obtenir 3 canaux
50        if len(img_array.shape) == 2:  # Image en niveaux de gris
51            img_array = np.stack([img_array] * 3, axis=-1)
52
53        # Reshaping des pixels en une liste de pixels (chaque
              pixel a 3 valeurs RGB)
54        pixels = img_array.reshape((-1, 3))
55
```

```
56      # Normalisation des pixels (valeurs entre 0 et 1)
57      pixels = pixels / 255.0
58
59      # Appliquer KMeans
60      kmeans = KMeans(n_clusters=n_clusters, random_state=42)
61      kmeans.fit(pixels)
62
63      # R cup rer les labels et l'image segment e
64      labels = kmeans.labels_
65      kmeans_image = labels.reshape(img_array.shape[:2])  #
           Reshape pour l'image segment e
66
67      # Calculer l'indice de Dunn pour KMeans
68      dunn_index = calculate_dunn_index(pixels, labels)
69
70      return kmeans_image, kmeans, dunn_index
71
72
73  # Fonction pour effectuer un clustering FCM
74  def fcm_clustering(image, n_clusters=3):
75      # Convertir l'image en tableau numpy
76      img_array = np.array(image)
77
78      # V rifier la forme de l'image
79      print(f"Forme de l'image : {img_array.shape}")
80
81      # Si l'image est en niveaux de gris (1 canal), dupliquer
           les valeurs pour obtenir 3 canaux
82      if len(img_array.shape) == 2:  # Image en niveaux de gris
83          img_array = np.stack([img_array] * 3, axis=-1)
84
85      # Reshaping des pixels en une liste de pixels (chaque
           pixel a 3 valeurs RGB)
86      pixels = img_array.reshape((-1, 3))
```

```
87
88      # Normalisation des pixels (valeurs entre 0 et 1)
89      pixels = pixels / 255.0
90
91      # Appliquer Fuzzy C-Means
92      cntr, u, _, _, _, _, _ = fuzz.cmeans(pixels.T, n_clusters,
            2, error=0.005, maxiter=1000)
93
94      # Choisir les classes bas es sur l'appartenance maximale
95      labels = np.argmax(u, axis=0)
96
97      # Reshaping pour l'image segment e
98      fcm_image = labels.reshape(img_array.shape[:2])  # Reshape
            pour l'image segment e
99
100     # Calculer l'indice de Dunn pour FCM
101     dunn_index = calculate_dunn_index(pixels, labels)
102
103     return fcm_image, u, dunn_index
104
105
106 # Exemple d'utilisation avec une image (remplacez
        'image_path.jpg' par le nom de votre image)
107 image = Image.open('TP2/Lenna_gray.jpg')  # Remplacer
        'Lenna_gray.jpg' par votre image
108
109 # Effectuer un clustering avec KMeans
110 kmeans_image, kmeans_model, kmeans_dunn_index =
        kmeans_clustering(image, n_clusters=3)
111 print(f"KMeans Dunn Index: {kmeans_dunn_index:.4f}")  #
        Afficher l'indice de Dunn pour KMeans
112
113 # Effectuer un clustering avec Fuzzy C-Means
```

```
114 fcm_image, fcm_membership, fcm_dunn_index =
        fcm_clustering(image, n_clusters=3)
115 print(f"Fuzzy C-Means Dunn Index: {fcm_dunn_index:.4f}")  #
        Afficher l'indice de Dunn pour FCM
116
117 # Affichage des r sultats
118 plt.subplot(1, 2, 1)
119 plt.imshow(kmeans_image, cmap='viridis')
120 plt.title('KMeans Clustering')
121
122 plt.subplot(1, 2, 2)
123 plt.imshow(fcm_image, cmap='viridis')
124 plt.title('Fuzzy C-Means Clustering')
125
126 plt.show()
```

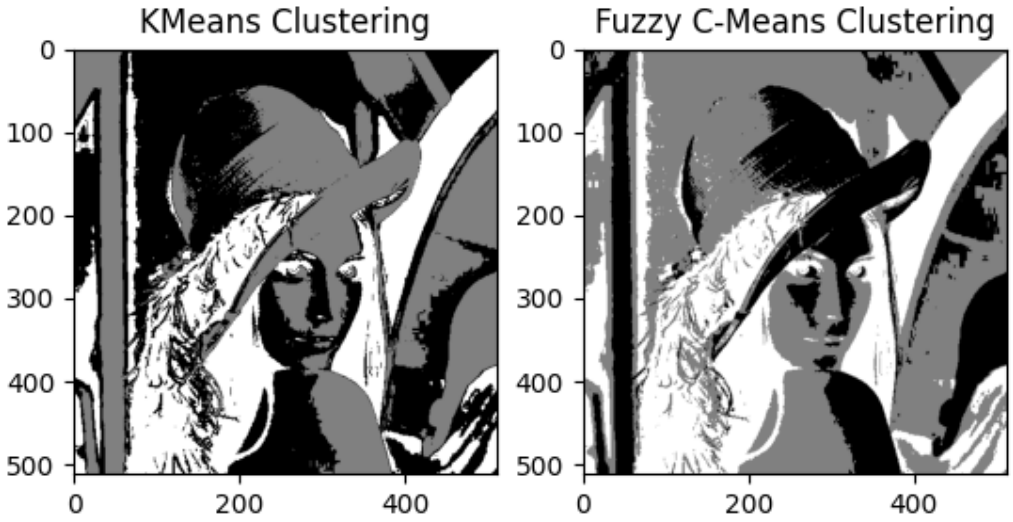

Figure 6: Comparaison de Kmeans et FCM sur Lenna

```
PS C:\Users\willi\OneDrive\Documents\INFO0503\TP> & C:/Users/willi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/w
lli/OneDrive/Documents/INFO0503/TP/TP2/lenna.py
Forme de l'image : (512, 512)
KMeans Dunn Index: 0.7025
Forme de l'image : (512, 512)
Fuzzy C-Means Dunn Index: 0.7856
```

Figure 7: Indices de Dunn sur Lenna

Sur le cas de l'image de Lenna en niveau de gris on observe une meilleur performance avec FCM que avec Kmeans. L'indice de Dunn montre aussi que FCM à fonctionner plus efficacement sur l'image. Au rendu on voit nettement que FCM crée bien plus précisément les clusters que Kmeans.