Unsupervised Machine Learning

Assignment 5

SAI KUMAR MURARSHETTI

LEWIS ID: L30079224

1. Technical description of methods.

Fuzzy C-Means (FCM): It is a clustering algorithm that allows each piece of data belong to many clusters. It is an expansion of the k-means technique in which each data point can be assigned to many clusters with variable levels of association.

Key features:

Soft Clustering: With hard clustering approaches (such as k-means), FCM provides the degree of association of each data point in each cluster, allowing it to more accurately represent data points on the outermost edges of various clusters. The method minimizes an objective function that determines the distance between data points and all centers, weighted by association levels.

Parameters:

- C: Number of clusters.
- q (Fuzziness coefficient): Indicates the degree of cluster fuzziness.

A higher value makes the clusters fuzzier, indicating boundaries between them are simply identified.

which is a Python package that contains tools for dealing with fuzzy logic and systems.

```
!pip install scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 994.0/994.0 kB 13.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.25.2)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.11.4)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (3.3)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=894078 sha256=2162ceb06a5dfb857853af44ffe72e03f7af5
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3fedc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

Data Preprocessing

Data Loading: Data is loaded from a CSV file into a pandas Data Frame.

One-Hot Encoding: Categorical variables are converted into a format that can be provided to ML algorithms to do a better job in prediction.

Standardization: Numerical features are standardized using StandardScaler from sklearn. This normalizes the data, giving each feature a mean of 0 and variance of 1.

2. Design of the Algorithms: Determining pairwise distances between data points. Make a minimum spanning tree for the distances. Rearrange the spacing according to the tree leaves for VAT. Reorder for iVAT using the VAT output, displaying darker blocks suggesting probable clusters.

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import pdist, squareform
from scipy.cluster.hierarchy import linkage, leaves_list
import skfuzzy as fuzz
```

```
# Loading and preprocessing the dataset
data_path = '/content/Traffic.csv'  # Adjust this path to your dataset
traffic_data = pd.read_csv(data_path)
```

This sequence of operations is typical in data preprocessing pipelines for machine learning, ensuring that categorical variables are correctly encoded for the model, and that numerical features are standardized to provide better performance and stability in subsequent analyses.

```
data_encoded = pd.get_dummies(traffic_data)
data_numeric = data_encoded.select_dtypes(include=[np.number])
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_numeric)
```

The following code apply_fcm function uses the scikit-fuzzy library to implement the Fuzzy C-Means clustering technique. It accepts a dataset, the desired number of clusters (num_clusters) and the fuzziness coefficient (fuzziness) as input. The function clusters using the cmeans technique from the fuzz.cluster module, with the choice to iterate up to 1000 times until the clustering error is less than 0.005. It returns the cluster centers (cntr) and the membership matrix (u), which shows the degree that each data point belong to each cluster.

```
# Function for applying Fuzzy C-Means
def apply_fcm(data, num_clusters, fuzziness):
    cntr, u, _, _, _, _, _ = fuzz.cluster.cmeans(
        data.T, num_clusters, fuzziness, error=0.005, maxiter=1000)
    return cntr, u
```

3. Results of the Algorithms: Visual assessment of cluster tendency (VAT) and improved VAT (iVAT).

VAT is a method for analyzing the presence of clusters in data. It reorders the data's dissimilarity matrix so that similar items are nearby, showing likely clusters using visual patterns. A rearranged dissimilarity matrix image with larger blocks on the diagonal indicating the presence of clusters. iVAT:

Improvement on VAT: Improved VAT creates a more detailed and readable image by further processing the dissimilarity matrix to improve cluster visibility, which is particularly helpful in large datasets and when clusters are inadequately separated.

The given code visualize_vat_ivat function generates visualizations for the Visual Assessment of Cluster Tendency (VAT) and Improved VAT (iVAT) using clustering by hierarchy. It really determines a pairwise distance matrix from the data and then uses this matrix to perform sequential clustering using a single linkage method. The function rearranges

the distance matrix in an ordered manner to improve the visibility of potential clusters. Two heatmaps are then shown: one for the original VAT and one for the reordered matrix (iVAT), which allow to observe the presence of clusters in the data.

```
# VAT and iVAT visualization
def visualize_vat_ivat(data):
    dist_matrix = squareform(pdist(data))
    linked = linkage(dist_matrix, 'single')
    order = leaves_list(linked)
    ordered_dist_matrix = dist_matrix[:, order][order]
    plt.figure(figsize=(14, 7))
    plt.subplot(121)
    plt.imshow(dist_matrix, cmap='hot', interpolation='nearest')
    plt.title('VAT', fontweight='bold')
    plt.colorbar()
    plt.subplot(122)
    plt.imshow(ordered_dist_matrix, cmap='Blues', interpolation='nearest')
    plt.title('Improved VAT (iVAT)', fontweight='bold')
    plt.colorbar()
    plt.show()
```

VAT and IVAT Visualization VAT (Visual Assessment of Tendency): shows a dataset's simple clustering structure by changing the distance matrix into an ordered matrix. iVAT: An enhancement to VAT that delivers a more clear representation of clusters by reorganizing the VAT output. This helps in identifying the number of clusters.

Fuzzy C-Means Clustering. Skfuzzy was used to implement the approach. This clustering approach assigns membership levels based on distance to cluster centers, with a fuzzy membership based on a parameter q.

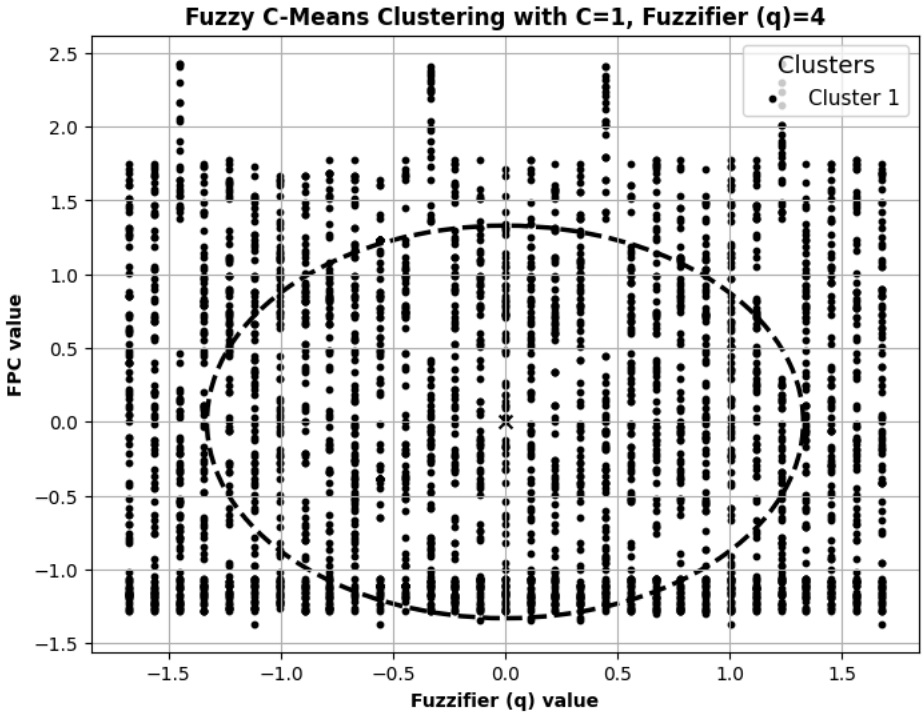
4. Analysis of Results: Here Each tuple represents a configuration, with the first element (C) indicating the number of clusters to identify and the second element (q) representing the level of cluster fuzziness. These configurations are used to experiment with different cluster counts and fuzziness levels to determine the way they affect clustering results.

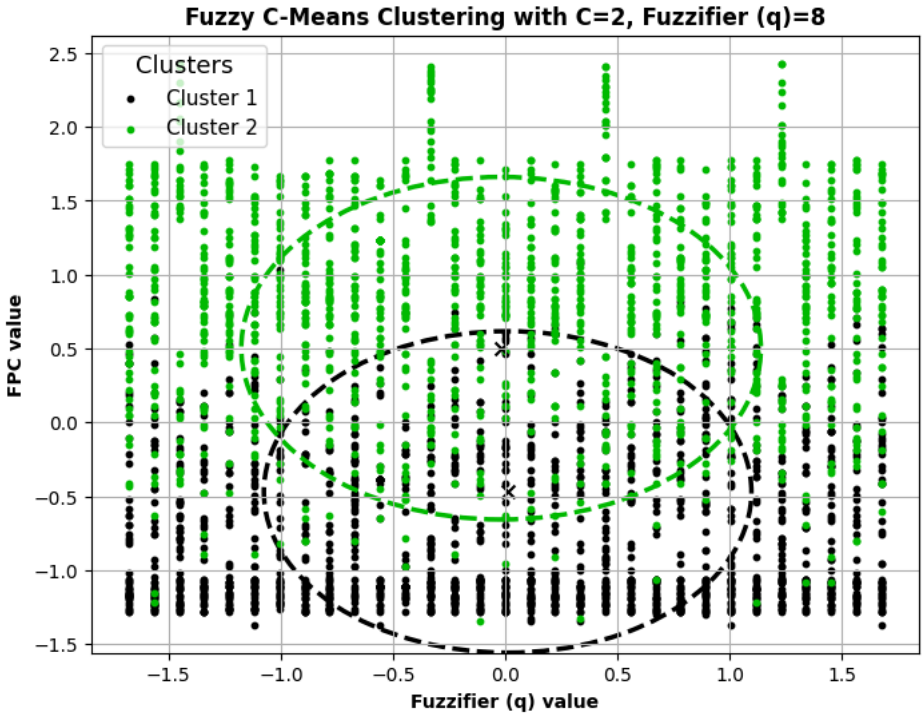
```
# Configurations for C and q
configurations = [(1, 4), (2, 8), (3, 12)]
```

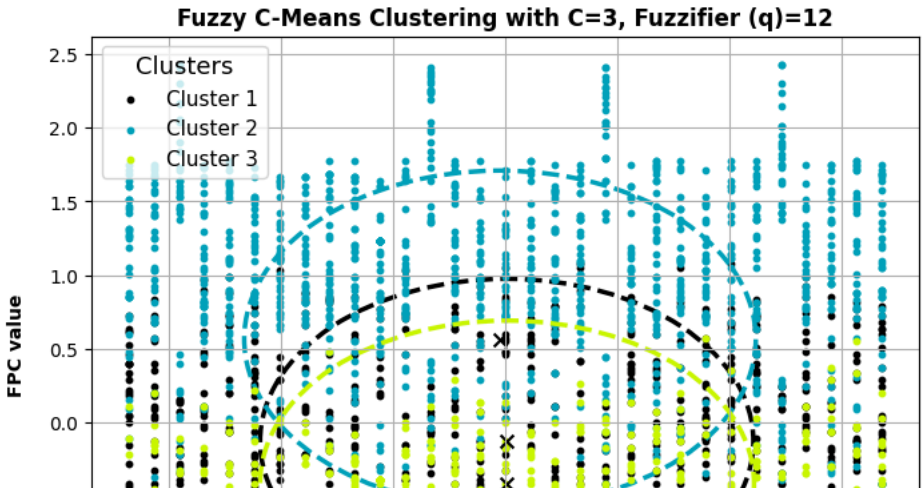
Each configuration sets the number of clusters (c) and the fuzziness parameter (q). For each arrangement.

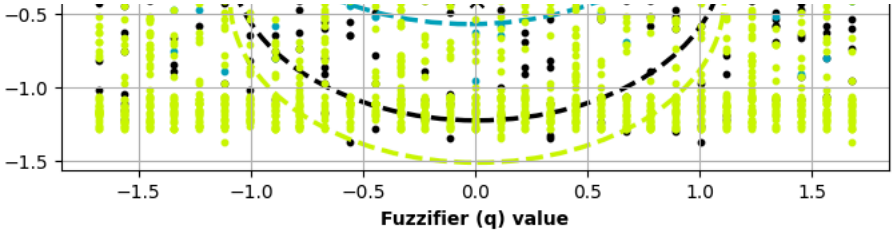
- Clustering Implementation: The apply_fcm function operates with the scaled data, number of groupings and fuzziness parameter and it provides cluster centers and a membership matrix. The np.argmax function identifies the most likely cluster for each data point, resulting in a vector of cluster labels.
- Visualization Configuration: Each arrangement results in a new plot. The plot's title and axis labels are set to show the current number of clusters and fuzziness value.
- Cluster Visualization determines the most associated with each cluster using a label vector.
- It determines the centroid of each cluster (using the first two features to visualize) as well as the average distance from the centroid to the points in that cluster, which is represented by a circle.
- The centers and circles of each cluster are colored.
- Plot features: Cluster centers are indicated with a 'x' and a description is included to assist in identifying each cluster. The resulting plot depicts how data points are distributed among clusters for each configuration along with a visual representation of cluster boundaries and centers.

```
# Iterate over each specified configuration to applying Fuzzy C-Means and visualize results
for c, q in configurations:
    centers, u = apply_fcm(scaled_data, c, q)
    labels = np.argmax(u, axis=0)

    # Initialize the plot for the current configuration
    plt.figure(figsize=(8, 6))
    plt.title(f'Fuzzy C-Means Clustering with C={c}, Fuzzifier (q)={q}', fontweight='bold')
    plt.xlabel('Fuzzifier (q) value', fontweight='bold')
    plt.ylabel('FPC value', fontweight='bold')

    for i in range(c):
        # Selecting data points that belong most strongly to cluster i
        points = scaled_data[labels == i, :2]

        # Calculating the centroid of the cluster using only two features
        centroid = centers[i, :2]

        # Calculating radius as the mean distance of points to the centroid
        distances = np.linalg.norm(points - centroid, axis=1)
        radius = np.mean(distances)

        # Drawing a circle for the cluster
        circle = plt.Circle(centroid, radius, color=plt.cm.nipy_spectral(i / float(c)), fill=False, linestyle='--', linewidth=2.5)
        plt.gca().add_artist(circle)

        # Ploting the data points
        plt.scatter(points[:, 0], points[:, 1], s=10, color=plt.cm.nipy_spectral(i / float(c)), label=f'Cluster {i+1}')

    # Ploting cluster centers
    plt.scatter(centers[:, 0], centers[:, 1], s=50, c='black', marker='x')


    plt.legend(title='Clusters', title_fontsize='13', fontsize='11', loc='best')
    plt.grid(True)
    plt.show()

```


Fuzzy C-Means Clustering with C=1, Fuzzifier (q)=4
Clusters
Cluster 1
FPC value
Fuzzifier (q) value


Fuzzy C-Means Clustering with C=2, Fuzzifier (q)=8
Clusters
Cluster 1
Cluster 2
FPC value
Fuzzifier (q) value


Fuzzy C-Means Clustering with C=3, Fuzzifier (q)=12
Clusters
Cluster 1
Cluster 2
Cluster 3
FPC value



VAT and iVAT are commonly used as exploratory tools to identify the number of clusters and evaluate the data before using a clustering method such as FCM. The combination of these techniques gives an accurate method for analyzing, clustering and interpreting datasets, making it particularly suitable for complex data sets such as "Crop_production".

This technical understanding will serve as a basis for the assignment's subsequent implementation and analysis.

This method allows a visual assessment of the way various configurations (number of clusters and fuzziness levels) impact the clustering output, which is critical for determining the effectiveness and acceptability of the FCM clustering approach for the supplied data.

5. Conclusion: The analysis using Fuzzy C-Means (FCM) clustering and Visual Assessment of Tendency (VAT/iVAT) visualization techniques yielded helpful findings regarding data processing and analysis skills. VAT and iVAT approaches are very useful for detecting clustering tendencies in the early stages, efficiently revealing possible clusters through high data density areas. These visualizations play an essential role in determining the dataset's natural clustering structure. Fuzzy C-Means clustering uses an adjustable fuzziness parameter (qq) to effectively handle data ambiguity.

This adaptability can be seen in the application through various configurations, which change cluster separation and connect, showing the algorithm's efficacy in dealing with complicated datasets where typical rigidity clustering fails. The integration of visual and computational methodologies not only improves knowledge of cluster dynamics, but also allows for more informed decision-making in real-world applications like market segmentation and pattern identification. The usage of Python's scientific libraries emphasizes the efficiency and scalability of this technique, making it suitable for advanced data analysis jobs in a variety of disciplines.