SUMAN KAYAL

2021CSB120

SOFT COMPUTING ASSIGNMENT-1

```
import pandas as pd
with open('iris.data','r') as file:
  csv_reader=pd.read_csv(file)
```

```
df=pd.read_csv('/content/iris.data')
```

```
print(df)
```

```
          5.1  3.5  1.4  0.2     Iris-setosa
     0    4.9  3.0  1.4  0.2     Iris-setosa
     1    4.7  3.2  1.3  0.2     Iris-setosa
     2    4.6  3.1  1.5  0.2     Iris-setosa
     3    5.0  3.6  1.4  0.2     Iris-setosa
     4    5.4  3.9  1.7  0.4     Iris-setosa
     ..   ...  ...  ...  ...             ...
     144  6.7  3.0  5.2  2.3  Iris-virginica
     145  6.3  2.5  5.0  1.9  Iris-virginica
     146  6.5  3.0  5.2  2.0  Iris-virginica
     147  6.2  3.4  5.4  2.3  Iris-virginica
     148  5.9  3.0  5.1  1.8  Iris-virginica

     [149 rows x 5 columns]
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
resulted_data=df.drop(['Iris-setosa'],axis=1)
print(resulted_data)
```

```
          5.1  3.5  1.4  0.2
     0    4.9  3.0  1.4  0.2
     1    4.7  3.2  1.3  0.2
     2    4.6  3.1  1.5  0.2
     3    5.0  3.6  1.4  0.2
     4    5.4  3.9  1.7  0.4
     ..   ...  ...  ...  ...
     144  6.7  3.0  5.2  2.3
     145  6.3  2.5  5.0  1.9
     146  6.5  3.0  5.2  2.0
     147  6.2  3.4  5.4  2.3
     148  5.9  3.0  5.1  1.8

     [149 rows x 4 columns]
```

```
import numpy as np

def min_max_normalization_2d(resulted_data):
  min_vals=np.min(resulted_data,axis=0)
  max_vals=np.max(resulted_data,axis=0)


  normalized_matrix=(resulted_data-min_vals)/(max_vals-min_vals)

  return normalized_matrix

normalized_matrix=min_max_normalization_2d(resulted_data)
print(normalized_matrix)
```

```
              5.1       3.5       1.4       0.2
     0    0.166667  0.416667  0.067797  0.041667
     1    0.111111  0.500000  0.050847  0.041667
     2    0.083333  0.458333  0.084746  0.041667
     3    0.194444  0.666667  0.067797  0.041667
     4    0.305556  0.791667  0.118644  0.125000
     ..        ...       ...       ...       ...
     144  0.666667  0.416667  0.711864  0.916667
     145  0.555556  0.208333  0.677966  0.750000
     146  0.611111  0.416667  0.711864  0.791667
     147  0.527778  0.583333  0.745763  0.916667
     148  0.444444  0.416667  0.694915  0.708333
```

```
[149 rows x 4 columns]
```

```
import numpy as np

#  'similarity_matrix' contains the m x m similarity matrix


# Function to find the average dissimilarity and form a cluster
def form_cluster(similarity_matrix):
    m = similarity_matrix.shape[0]
    clusters = []

    for i in range(m):
        # Calculate average dissimilarity of i-th object with others
        avg_dissimilarity = np.sum(similarity_matrix[i]) / (m - 1)

        # Form a cluster Ci with i-th object and objects having dissimilarity less than average
        cluster = [j for j in range(m) if j != i and similarity_matrix[i, j] < avg_dissimilarity]
        cluster.append(i)  # Include the i-th object in the cluster
        clusters.append(cluster)

    return clusters

# Call the function to form clusters
clusters = form_cluster(similarity_matrix)

# 'clusters' is a list of lists, where each inner list represents a cluster
print(clusters)
```

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35
```

```
import numpy as np

#  'df' contains the normalized dataset (m x n matrix)


# Function to calculate Euclidean distance between two objects
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))

# Function to compute the similarity matrix
def create_similarity_matrix(data):
    m, n = data.shape
    similarity_matrix = np.zeros((m, m))

    for i in range(m):
        for j in range(m):
            similarity_matrix[i, j] = euclidean_distance(data.iloc[i], data.iloc[j])


    return similarity_matrix

# Call the function to create the similarity matrix
similarity_matrix = create_similarity_matrix(normalized_matrix)

# 'similarity_matrix' is the m x m similarity matrix based on Euclidean distance
print(similarity_matrix)
```

```
[[0.         0.10157824 0.09469862 ... 1.08390691 1.17619813 0.95649502]
 [0.10157824 0.         0.06047157 ... 1.12088708 1.19544459 0.98859665]
 [0.09469862 0.06047157 0.         ... 1.11178383 1.18984212 0.97410913]
 ...
 [1.08390691 1.12088708 1.11178383 ... 0.         0.226928   0.18710825]
 [1.17619813 1.19544459 1.18984212 ... 0.226928   0.         0.28409587]
 [0.95649502 0.98859665 0.97410913 ... 0.18710825 0.28409587 0.        ]]
```

```
#  'clusters' contains the list of clusters obtained from the previous step

# Function to remove subsets from the list of clusters
def remove_subsets(clusters):
    p = len(clusters)
    subset_flags = [False] * p

    for i in range(p):
        for j in range(p):
            if i != j and set(clusters[i]).issubset(set(clusters[j])):
                subset_flags[i] = True
```

```
                break

    pruned_clusters = [cluster for i, cluster in enumerate(clusters) if not subset_flags[i]]
    return pruned_clusters

# Call the function to remove subsets from the list of clusters
pruned_clusters = remove_subsets(clusters)

# 'pruned_clusters' contains p (< m) clusters after removing any subsets
print(pruned_clusters)
```

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
```

```
#  'pruned_clusters' contains the list of pruned clusters obtained from the previous step

# Function to compute the similarity between two clusters
def compute_similarity(cluster1, cluster2):
    intersection = len(set(cluster1) & set(cluster2))
    union = len(set(cluster1) | set(cluster2))
    similarity = intersection / union
    return similarity

# Function to create the similarity matrix C
def create_similarity_matrix_p(pruned_clusters):
    p = len(pruned_clusters)
    similarity_matrix_p = np.zeros((p, p))

    for i in range(p):
        for j in range(p):
            similarity_matrix_p[i, j] = compute_similarity(pruned_clusters[i], pruned_clusters[j])

    return similarity_matrix_p

# Call the function to create the similarity matrix C
similarity_matrix_p = create_similarity_matrix_p(pruned_clusters)

# 'similarity_matrix_p' is the p x p similarity matrix between clusters
print(similarity_matrix_p)
```

```
[[1.         0.73611111 0.65       0.73611111 0.33018868 0.17592593
  0.07079646 0.07142857 0.26168224 0.27102804 0.04964539 0.04929577
  0.04929577]
 [0.73611111 1.         0.88461538 0.94444444 0.47169811 0.3271028
  0.19298246 0.19469027 0.40186916 0.41121495 0.14788732 0.14685315
  0.13103448]
 [0.65       0.88461538 1.         0.86075949 0.55238095 0.40566038
  0.26548673 0.26785714 0.48113208 0.49056604 0.20567376 0.20422535
  0.17931034]
 [0.73611111 0.94444444 0.86075949 1.         0.47169811 0.3271028
  0.19298246 0.19469027 0.40186916 0.41121495 0.14788732 0.14685315
  0.13103448]
 [0.33018868 0.47169811 0.55238095 0.47169811 1.         0.79545455
  0.61702128 0.62365591 0.90804598 0.91954023 0.46721311 0.46341463
  0.42857143]
 [0.17592593 0.3271028  0.40566038 0.3271028  0.79545455 1.
  0.74683544 0.75641026 0.87654321 0.84337349 0.54205607 0.53703704
  0.4954955 ]
 [0.07079646 0.19298246 0.26548673 0.19298246 0.61702128 0.74683544
  1.         0.95522388 0.67816092 0.67045455 0.69148936 0.68421053
  0.63265306]
 [0.07142857 0.19469027 0.26785714 0.19469027 0.62365591 0.75641026
  0.95522388 1.         0.68604651 0.67816092 0.68085106 0.67368421
  0.62244898]
 [0.26168224 0.40186916 0.48113208 0.40186916 0.90804598 0.87654321
  0.67816092 0.68604651 1.         0.96341463 0.50434783 0.5
  0.46218487]
 [0.27102804 0.41121495 0.49056604 0.41121495 0.91954023 0.84337349
  0.67045455 0.67816092 0.96341463 1.         0.5        0.4957265
  0.45833333]
 [0.04964539 0.14788732 0.20567376 0.14788732 0.46721311 0.54205607
  0.69148936 0.68085106 0.50434783 0.5        1.         0.96842105
  0.92783505]
 [0.04929577 0.14685315 0.20422535 0.14685315 0.46341463 0.53703704
  0.68421053 0.67368421 0.5        0.4957265  0.96842105 1.
  0.93814433]
 [0.04929577 0.13103448 0.17931034 0.13103448 0.42857143 0.4954955
  0.63265306 0.62244898 0.46218487 0.45833333 0.92783505 0.93814433
  1.        ]]
```

```
# Function to find the most similar clusters and merge them
```

```
# Function to find the most similar clusters and merge them
def merge_most_similar_clusters(similarity_matrix_p, pruned_clusters):
    p = len(pruned_clusters)
    max_similarity = -1.0
    most_similar_clusters = (None, None)

    # Find the most similar clusters Ck and Cl
    for k in range(p):
        for l in range(k + 1, p):  # To avoid checking pairs twice (symmetric matrix)
            similarity = similarity_matrix_p[k, l]
            if similarity > max_similarity:
                max_similarity = similarity
                most_similar_clusters = (k, l)

    # Get the indices of the most similar clusters
    k, l = most_similar_clusters

    # Merge the most similar clusters Ck and Cl to get a new cluster Ckl
    Ck = set(pruned_clusters[k])
    Cl = set(pruned_clusters[l])
    Ckl = list(Ck.union(Cl))

    # Remove the individual clusters Ck and Cl from the list and add the new cluster Ckl
    pruned_clusters.pop(max(k, l))
    pruned_clusters.pop(min(k, l))
    pruned_clusters.append(Ckl)

    return pruned_clusters

# Call the function to find the most similar clusters and merge them
merged_clusters = merge_most_similar_clusters(similarity_matrix_p, pruned_clusters)

# 'merged_clusters' contains the updated list of clusters after merging the most similar clusters
print(merged_clusters)
```

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
```

✓ 0s completed at 1:54 AM