CSC 730 Assignment 2 Writeup: Fuzzy Clustering and Cluster Scoring.

Mangesh Sakordekar
Minati Alphonso

Overview

Aim of the project is to:

- Implement our own version of the Fuzzy C-Means clustering algorithm from scratch.
- Apply the algorithm to skewed MNIST dataset.
- Apply the algorithm to PCA'd version of the skewed MNIST dataset reduced to a dimensionality of 2.
- Write our own version of the Adjusted Rand Index algorithm and use it to score the results of the clustering.

Fuzzy C-Means

Implementation

The Fuzzy C-Means algorithm was implemented using the equations discussed in the class.

$$J = \sum_{i=1}^{n} \sum_{k=1}^{c} \mu_{ik}^{m} |p_i - v_k|^2$$

Figure 1: Equation 1

$$|p_i - v_k| = \sqrt{\sum_{i=1}^{n} (p_i - v_k)}$$

Figure 2: Equation 2

$$v_k = \frac{\sum_{i=1}^{n} \mu_{ik}^{m} p_i}{\sum_{i=1}^{n} \mu_{ik}^{m}}$$

Figure 3: Equation 3

$$\mu_{ik} = \frac{1}{\sum_{l=1}^{c} \left(\frac{|p_i - v_k|}{|p_i - v_l|} \right)^{\frac{2}{m-1}}}$$

Figure 4: Equation 4

```python
def calculate_objective_function(data, cluster_centers, membership_matrix, fuzziness):

    distances = np.linalg.norm(data[:, np.newaxis] - cluster_centers.T, axis=2)
    objective_function_value = np.sum(membership_matrix**fuzziness * distances**2)

    return objective_function_value
```

Figure 5: Objective Value Implementation

```
1  def fuzzy_c_means(data, num_clusters, fuzziness, max_iter=100, tolerance=1e-4):
2      # Initialize membership matrix randomly
3      membership_matrix = np.random.rand(len(data), num_clusters)
4      membership_matrix /= np.sum(membership_matrix, axis=1, keepdims=True)
5
6      for _ in range(max_iter):
7          # Update cluster centers
8          cluster_centers = np.dot(data.T, membership_matrix**fuzziness) /
9                            np.sum(membership_matrix**fuzziness, axis=0, keepdims=True)
10
11         # Calculate distances between data points and cluster centers
12         distances = np.linalg.norm(data[:, np.newaxis] - cluster_centers.T, axis=2)
13
14         # Update membership matrix
15         new_membership_matrix = 1 / np.sum((distances[:, :, np.newaxis] /
16                                             distances[:, np.newaxis, :])**(2/(fuzziness-1)), axis=2)
17
18         new_membership_matrix /= np.sum(new_membership_matrix, axis=1, keepdims=True)
19
20         # Calculate the change in the objective function value
21         obj_function_value = calculate_objective_function(data, cluster_centers, membership_matrix, fuzziness)
22         new_obj_function_value = calculate_objective_function(data, cluster_centers, new_membership_matrix, fuzziness)
23         obj_function_change = abs(new_obj_function_value - obj_function_value)
24
25         # Check for convergence
26         if obj_function_change < tolerance:
27             break
28
29         membership_matrix = new_membership_matrix
30
31     return cluster_centers, membership_matrix
32
```

Figure 6: Fuzzy C-Means implementation

In the code, first we assign random values to the membership matrix and ensure that the sum of each row is 1 by diving each row by it's sum. Then we update the membership matrix by calculating centers and using the distances between the points.

Line 8 in Figure 6 calculates the centers of the clusters for given iteration by implementing equation 3. The numerator is the dot product of membership matrix raised to the fuzziness'th power and the transpose of the data. The denominator is just the sum of membership matrix raised to the fuzziness'th power.

Next we calculate the distances of the cluster centers from each point. Line 12 in Figure 6 calculates the distances and stores them in a variable using 2. These distances are then used in line 15 to calculate the new membership matrix using the equation 4.

Upon calculating the new membership matrix, we check for the stopping criteria. In this implementation we are using the objective value, which is the sum of squared distances between data points and their assigned cluster centers, as the stopping criteria. This implementation can be seen in Figure 5. We use equation 2 to calculate distances here as well.

If the difference between the new objective value and objective value of the previous iteration is less then the threshold, we stop iterating and return the cluster centers and the membership matrix.

Application and Results

The actual distribution of the classes is shown in figure 7.

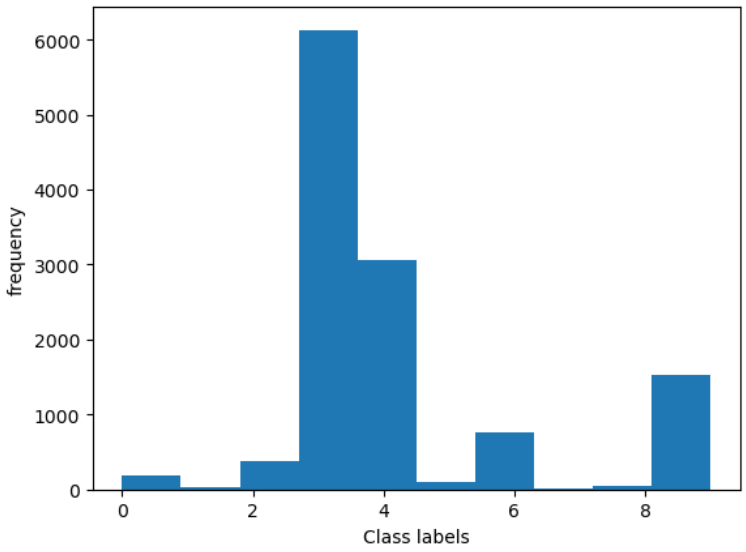

Figure 7: Class Distribution

Since the Fuzzy C-Means algorithm does not maintain the order of classes, we are just looking at the class sizes to compare the distributions.

Dataset with 784 Dimensions

The cluster centers generated by the FCM algorithm can be seen in figure 8

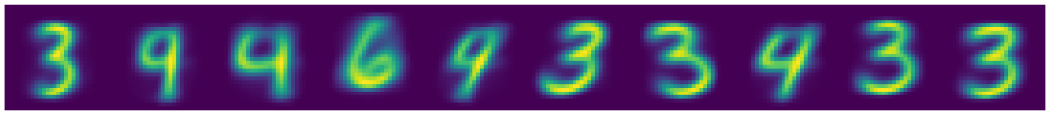

Figure 8: Cluster Centers in 784 dimensions

The class distribution given by the FCM algorithm for the 784 dimension data can be seen in figure 9

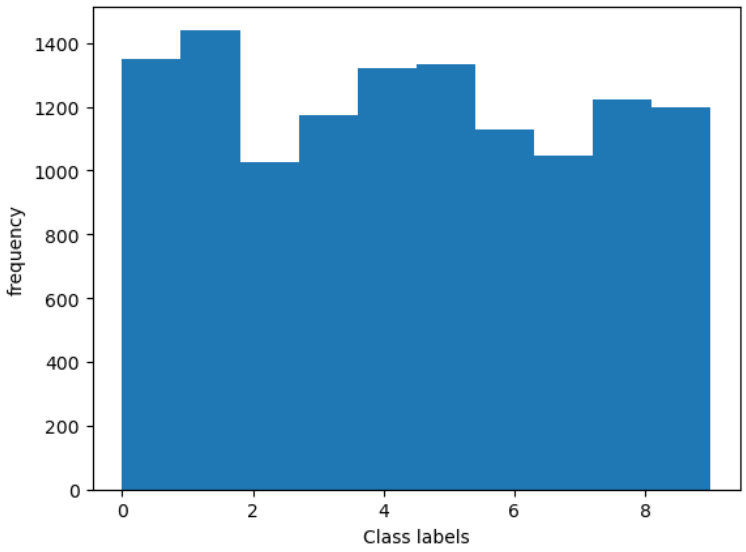

Figure 9: Class Distribution of FCM using 784 dimension data

As we can see, this distribution has more evenly sized classes which is not the case with the original data.

Dataset reduced to 2 dimensions (PCA)

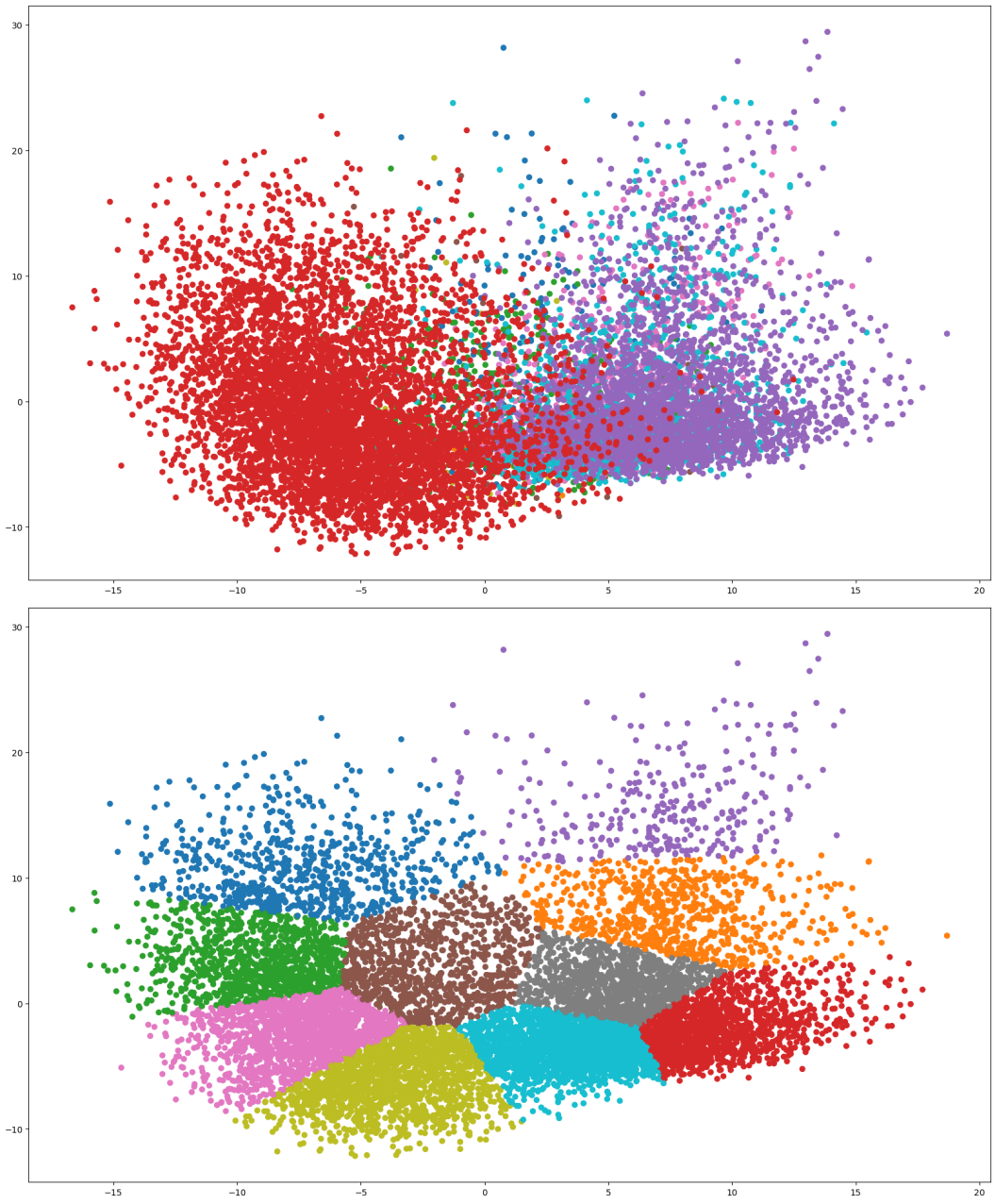

Figure 10: Scatter plot of 2-d data Top : original labels Bottom: Labels generated by FCM

Dataset reduced to 2 dimensions (t-SNE)

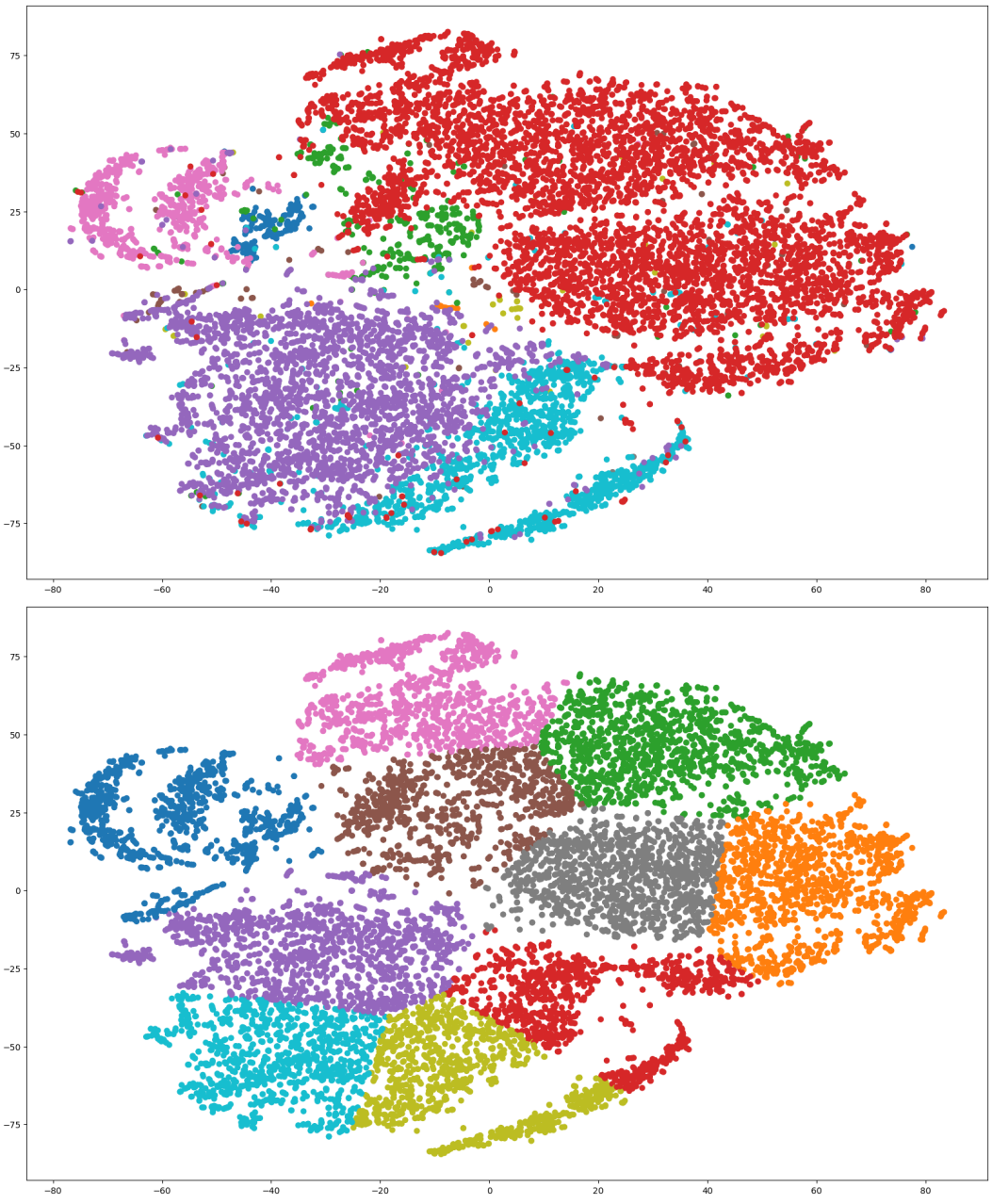

Figure 11: Scatter plot of 2-d data Top : original labels Bottom: Labels generated by FCM

Adjusted Rand Index (ARI)

Implementation

$$\text{AdjustedIndex} = \frac{\text{Index} - \text{ExpectedIndex}}{\text{MaxIndex} - \text{ExpectedIndex}}$$

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{N}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right]/\binom{N}{2}}$$

```python
def ari_from_scratch(true_labels, pred_labels):
    # Create a contingency table from true and predicted labels
    contingency_table = contingency_matrix(true_labels, pred_labels)

    n_choose_k = np.vectorize(math.comb)

    # Calculate the total number of ways to choose 2 items from all elements
    n_choose_2 = n_choose_k(np.sum(contingency_table),2)

    # Sum of combinations for each cell in the contingency table
    n_choose_2_sum = np.sum(n_choose_k(contingency_table,2))

    # Calculate the sum of elements in each row and each column of the contingency table
    row_sum = np.sum(contingency_table,axis=1)
    col_sum = np.sum(contingency_table,axis=0)

    # Sum of combinations for each row and each column
    row_sum_choose_2 = np.sum(n_choose_k(row_sum,2))
    col_sum_choose_2 = np.sum(n_choose_k(col_sum,2))

    # Actual index: total number of pairings in the contingency table
    index = n_choose_2_sum

    # Calculate the expected index (chance grouping)
    expectedIndex = row_sum_choose_2 * (col_sum_choose_2/n_choose_2)

    # Maximum index: average of the sum of row combinations and column combinations
    maxIndex = (row_sum_choose_2 + col_sum_choose_2)/2

    ari = (index - expectedIndex)/(maxIndex - expectedIndex)
    return contingency_table, ari
```

The core of the ARI calculation involves the construction of a contingency table, which tabulates the number of objects in common between pairs of clusters - one from the true labels and the other from the predicted labels. Our ARI implementation initiates by generating a contingency table.

The subsequent steps involve calculating combinations of elements in the contingency table and its margins, following the formula of ARI. Specifically, the key components are:

- `n_choose_2`: The total number of pairwise combinations in the dataset.
- `n_choose_2_sum`: The sum of pairwise combinations within each cell of the contingency table.
- `row_sum_choose_2` and `col_sum_choose_2`: The sum of combinations for each row and column margin, respectively.

The expected index and maximum index are then computed, leading to the final ARI value, which is normalized to lie between -1 (no agreement) and 1 (perfect agreement).

Application and Results

Dataset with 784 Dimensions

```
Contingency Table for 784 dimensions:
 [[  20    0    5  103   26    0   15    9   13    0]
 [   0    6   17    0    0    0    0    0    0    0]
 [  14   14   55  220   52   10    7    9    2    0]
 [1180   65 1146   71 1242  138 1098 1152   12   27]
 [   0  760   25   77    0  728    1    0  737  737]
 [   7   25    9    5   11    8   16    7    2    5]
 [   1    9   32  673    1    2    1    0   47    0]
 [   0    6    0    0    1    0    0    0    0    4]
 [   1   16   11    1    5    1    6    4    0    2]
 [   2  313   29    3    4  491    8   13  229  440]]
ARI for 784 dimensions:  0.18412797612546336
ARI using scikit-learn: 0.18412797612546336
```

For the 784-dimensional dataset, the ARI was 0.1841, indicating a moderate level of agreement between the clusters and the true labels.

Dataset reduced to 2 dimensions (PCA)

```
Contingency Table for 2 dimensions:
 [[  26   31   14   27   44    6    8    0   35    0]
 [   0   15    8    0    0    0    0    0    0    0]
 [  56  101   26    1  128    8   19    6   19   19]
 [  61  275 1613   11  703   10  704 1075   17 1662]
 [ 636  786    0  140   14 1099    2    0  388    0]
 [   5   38   21    2   13    4    5    5    0    2]
 [ 190  354   10   44   30   24    0    0  114    0]
 [   1    7    1    0    0    2    0    0    0    0]
 [   2   24   11    1    2    0    2    1    1    3]
 [ 249  673   13   75   33  334    5    0  148    2]]
ARI for 2 dimensions:  0.17130325910761843
ARI using scikit-learn: 0.1713032591076184
```

For the 2-dimensional dataset reduced using PCA, the ARI was 0.1713, suggesting a moderate level of agreement.

Dataset reduced to 2 dimensions (t-SNE)

```
Contingency Table for 2 dimensions (t-SNE):
 [[   0    2    0    0    1    0    0    0  187    1]
 [   0    0    1    0    0    0   17    4    0    1]
 [   5    7    1   10   58    0   29    3   20  250]
 [  12 1347  253 1375 1102   25    4 1255   14  744]
 [1002    6  165    0    2  363 1425    3   91    8]
 [   1    9    3   11    2    2   10   10   38    9]
 [   2    0    1    1    0    1   23    0  734    4]
 [   0    0    9    0    1    1    0    0    0    0]
 [   0    6    3    4    2    0    8   19    2    3]
 [ 110   12  638    1    1  712   24   19    4   11]]
ARI for 2 dimensions (t-SNE):  0.24576839074244772
ARI using scikit-learn (t-SNE): 0.2457683907424477
```

For the 2-dimensional dataset reduced using t-SNE, the ARI improved to 0.2458, indicating a better clustering performance.

Notice that the ARI values calculated by our implementation matched exactly with those obtained using the `adjusted_rand_score` function from scikit-learn, validating the accuracy of our code.

Conclusion

Looking at the results above, we can conclude that Fuzzy C-Means does not perform particularly well with this dataset. Our assumption is that this algorithm would perform better if there is a clear separation between the clusters as it performed slightly better with t-SNE dataset which had some separation. Fuzzy C-Means also fails in this case as it tries to create clouds of points around centers leading to circular clusters.