MACHINE LEARNING ASSIGNMENT 2 REPORT

NAME : DIPAN MANDAL
ROLL NO.: M23CSA009
M.TECH 1ST YEAR, ARTIFICIAL INTELLIGENCE

Task 1.a. Perform K-means clustering on MNIST data from scratch using cosine similarity as the distance metric.
Clustering should be done in 10, 7 and 4 clusters.

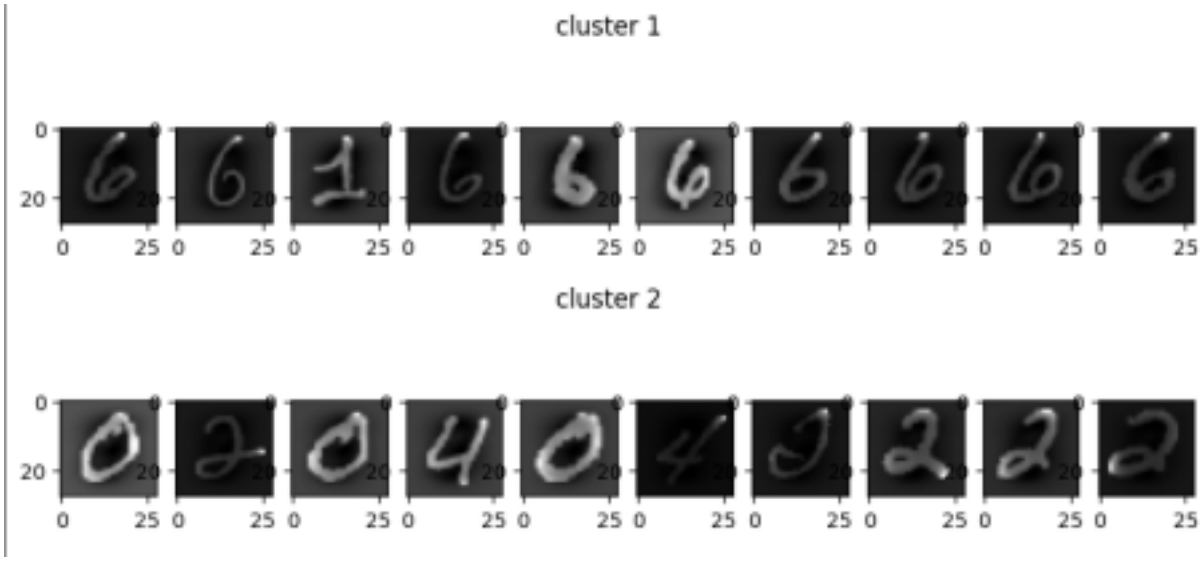
Colab notebook link:
https://colab.research.google.com/drive/1CThPHQSVbgQKaj8il7N48DpJ8_s7E_d1?usp=sharing

Fetching data and preprocessing:

The first step in the process was fetching the data and doing some preprocessing on it as the data contained some null values (NaN) which we replaced with 0. Then we applied standardization so that that mean is removed and the variance becomes 1. This mainly helps during the time of calculation.

For standardizing the data, we have used the `StandardScaler` method from `sklearn.preprocessing`.

During this process, I had to make a separate copy of the data as the images that I was getting while printing the standardized array lacked clarity. So we made another copy `'data2'` which I used later for printing the images.



Unclear images after applying standardization on the dataset

Implementing K-means clustering:

The K-means clustering is implemented in the function **`KMeansClustering`** which takes 3 arguments:

1. The dataset: **`X`**
2. Number of clusters: **`clusters_count`**
3. Number of iterations: **`max_iters`**

We have taken cosine similarity as the distance metric instead of Euclidean distance. In this case **we need to maximize the value of cosine similarity**(whereas in case of Euclidean distance we used to minimize the distance)

We have run the code for 100 iterations for each clusters_count (10, 7 and 4)

Task 1.b. Visualization of images getting clustered into different clusters.

In the question we were asked to use the cluster numbers 10, 7 and 4. Here are the results of the images present in the clusters. For printing the images we have used the function `showFigure` function which changes the shape of the 1D numpy array of size 784 to a 28X28 2D matrix and then plots it using matplotlib.

Below are the results for the different count of clusters (I have printed 10 images from each cluster to show the type of digits that are present in the cluster)

Cluster count = 10:

cluster 1

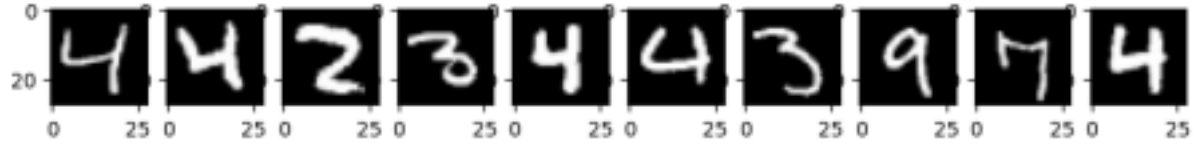
cluster 2

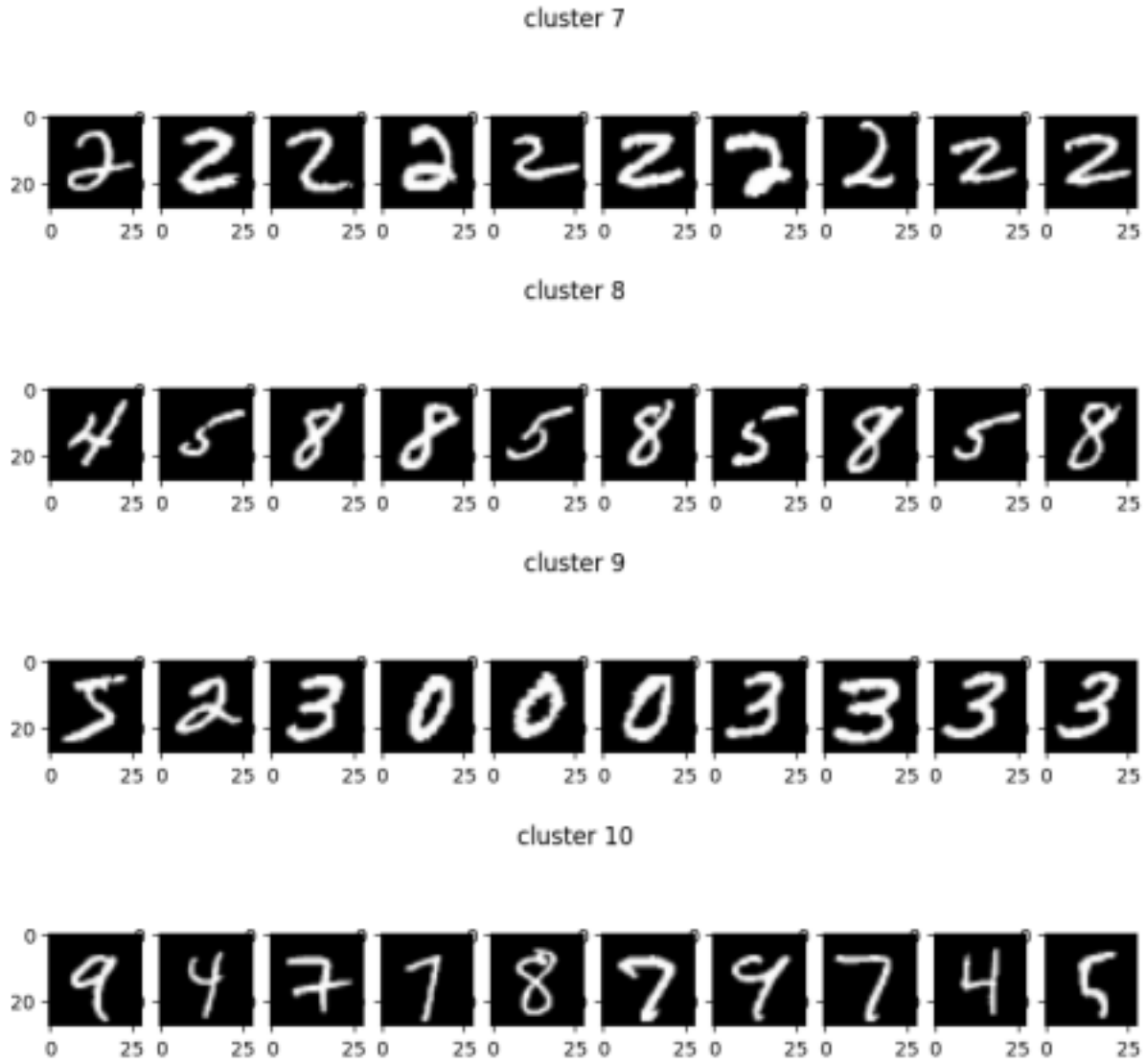
cluster 3

cluster 4

cluster 5

cluster 6



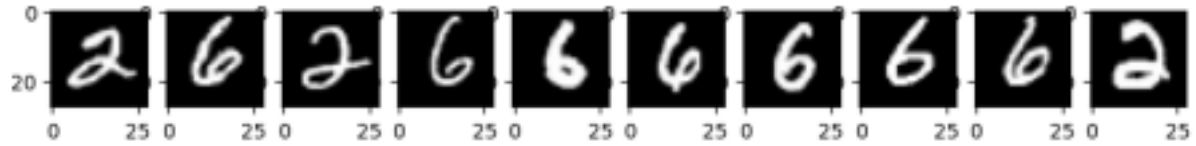


Comment on cluster formation:

As compared to other cluster numbers 10 performs better as there are 10 different digits present in the MNIST dataset. Digits like 6, 3, 1 and 0 can be seen to be aggregated in specific clusters (mainly because they have some unique shape). The other clusters seem to have a combination of digits. E.g. **cluster 8** seems to have a combination of 8 and 5 and **cluster 1** seems to have a combination of 9 and 7.

Cluster count = 7:

cluster 1



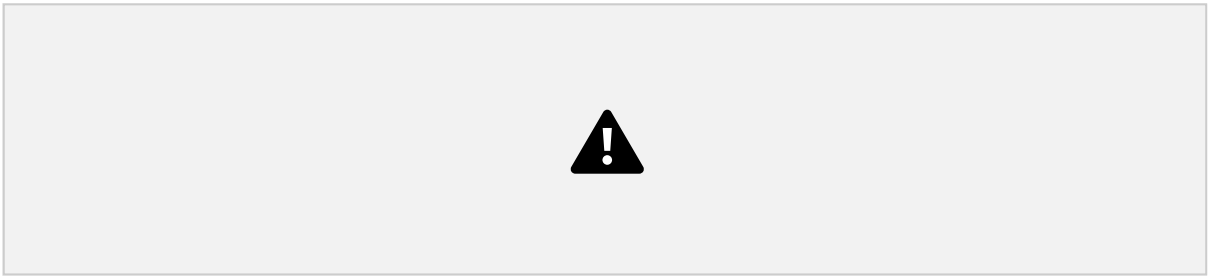
cluster 2

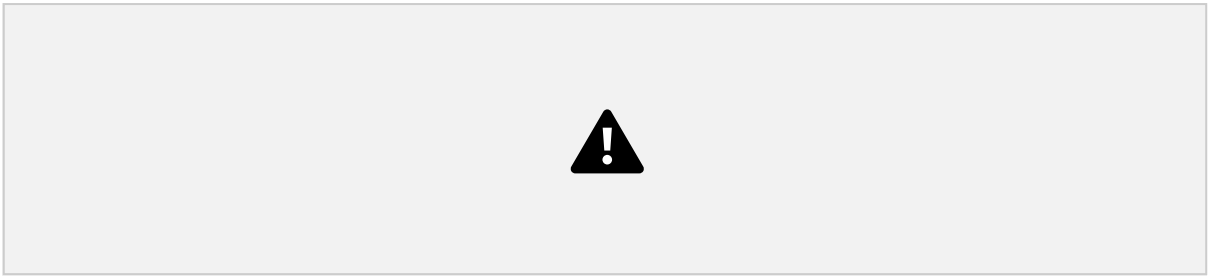
cluster 3

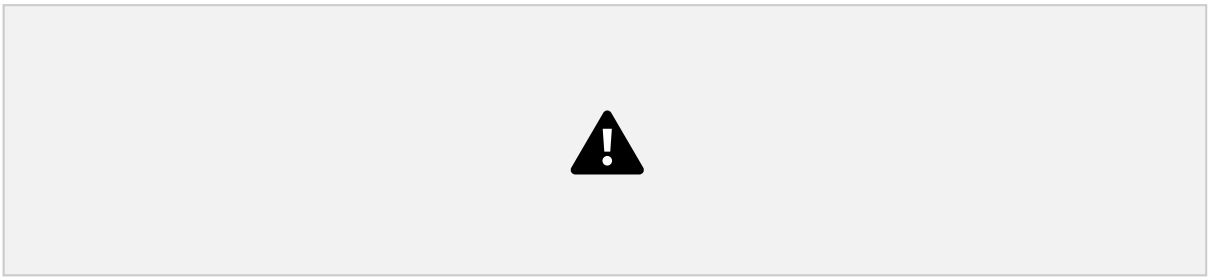
cluster 4

For cluster count of 7 multiple digits with similar kind of pattern seem to come under the same clusters. Such as digits 9,7 come in cluster 7, digits 2 and 6 are clustered in cluster 1.

Cluster count = 4:





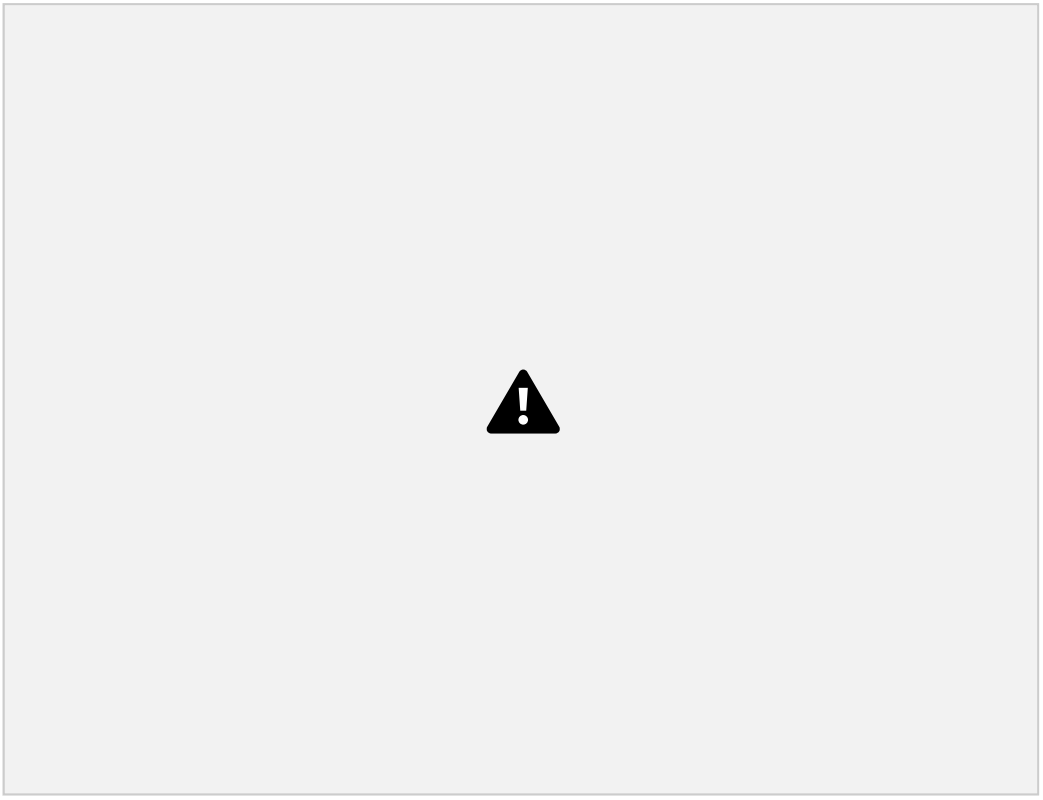


For cluster count of 4 we can't find any specific pattern in the digits present in the clusters. The results are quite intuitive as dividing 10 different digits in only 4 clusters can't provide much information. But we can say that digits having similar kinds of patterns (like 9 and 7) are present in the same cluster.

Task 2.a. Applying PCA on the MNIST dataset and then performing GMM clustering on the dataset. PCA should be applied for 32, 64 and 128 dimensions and the GMM clustering should be done for 10, 7 and 4 clusters respectively.

Colab notebook link:
https://colab.research.google.com/drive/1HXWRXdn4FZjNjKdT1TzsGF04t1iC_LAE?usp=sharing

Note: while performing the PCA and GMM on the full dataset with 60,000 entries using Google Colab, Colab was crashing showing that the allocated RAM (12 GB) has been used. The code was working fine in a system with 16GB RAM.
[A screenshot showing the error that was occurring]



So I decided to run the code on the MNIST_test data which contained 10,000 data points and the code worked fine.

But the problem was due to a single line of code:

While **`full_matrices = True`** the u matrix is calculated to be of **60000X60000 dimensions** i.e. the size of the full dataset for the `MNIST_train` which was causing overflow in the memory. But when we do **`full_matrices = False`**, only a **60,000X784 matrix is calculated**. This does not cause any memory overflow.
After this change I decided to go forward with the `MNIST_train` dataset. **Data fetching and preprocessing:**

For data preprocessing we have used the same procedure as the last task. I.e. replacing all the NaN values present in the data with 0 and standardizing the data using **`StandardScaler`** from Scikit Learn.
Implementing PCA and GMM clustering:

For PCA we have used the function **`findPrincipalComponents`** which takes the dataset and the number of required dimensions as input and returns **`X_transformed,`** the transformed value of X in lower dimensions.

We perform the GMM clustering on the **`X_transformed`** using the function **`GmmClustering`**. The functions take the dataset and the number of clusters as the input and returns **`clusters`, a list of lists containing the index of the data points belonging to each cluster**. The number of lists present in `clusters` is equal to the number of clusters formed by the algorithm.

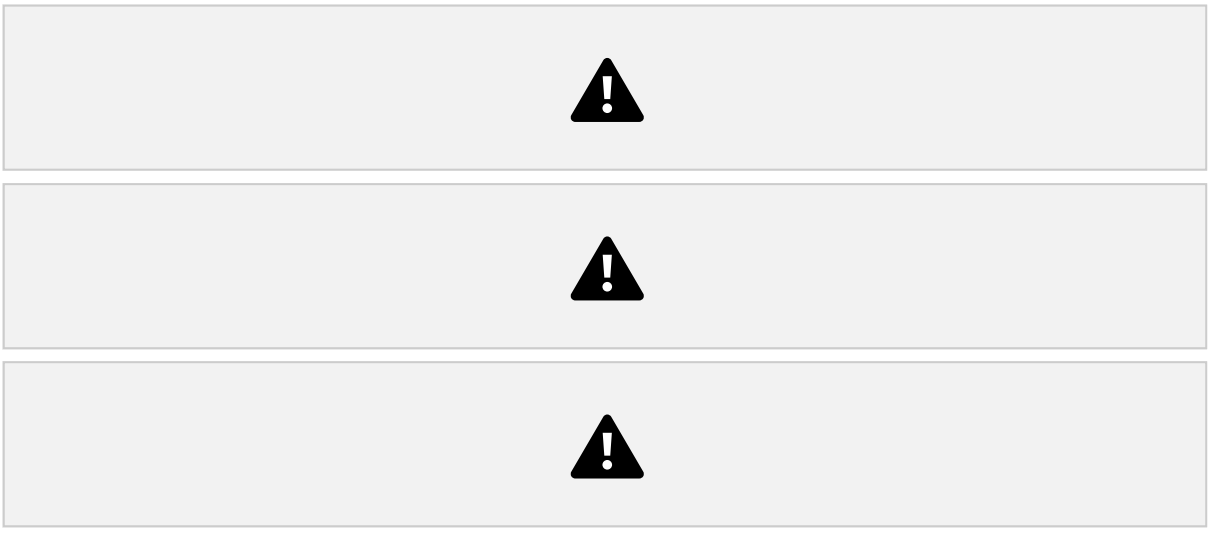
Later we combined those two algorithms in the function **`PCA_GMM_Combinations`** to get the required results.

Again for printing the images we have used the **`showFigure`** function described in the previous task.

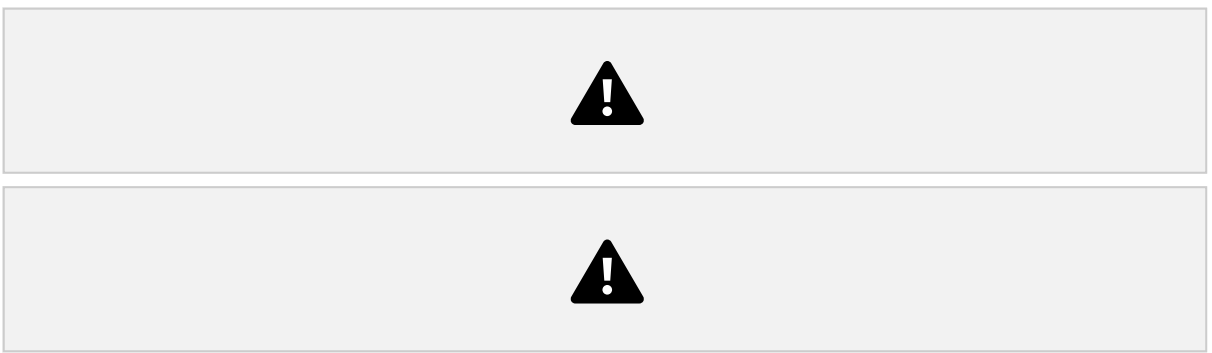
Task 2.b. Visualize the images getting clustered to different clusters.

Here we will be getting a total of 9 combinations of dimensions and clusters. I have only shown a few results in the report. The detailed result can be accessed from the colab notebook shared.

1. **Results for 32 dimensions, 10 clusters:**



2. Results for 64 dimensions, 7 clusters:






3. **Results for 128 dimensions, 4 clusters:**




Task 2.c. comparing the performances of the current clusters with the previous clusters.

We did not notice any significant improvement in the performance of the GMM clustering on the datasets with lower dimensions. This can occur due to a few reasons:

1. The **data does not follow any Gaussian distribution**, so it is hard to

fit any distribution to the different clusters and identify them. Although some of the digits like 1, 6, 2 were being clustered in distinct clusters, most of the clusters were a mixture of 2-4 different digits.

2. The **loss of information** due to decreasing the dimension from 784 to 128 may also be the reason for the model not performing well. We can expect the model to perform better in higher dimensions.