REPORT
HW4: Iris and Digits clustering using K-Means

Name	Pranav Agrawal
Miner2 Username	Vikramaditya
GMU ID	pagrawa

K-Mean - Iris

Rank	63
Public Score (V-Measure)	0.95

K-Mean - Image

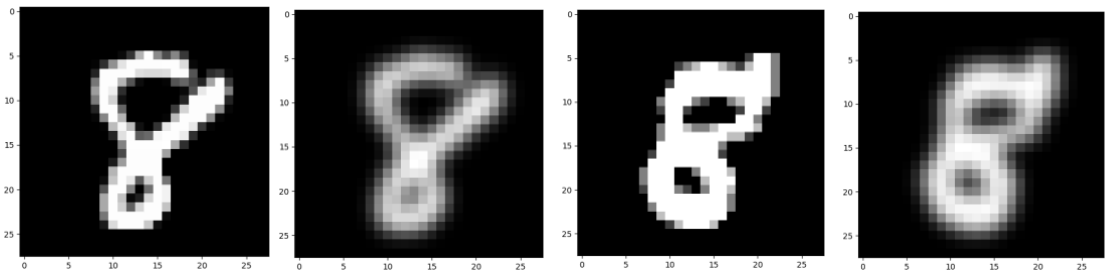
Rank	56
Public Score (V- Measure)	0.86

Objective:

Utilize the K-Means clustering algorithm to accurately label the data. The algorithm locates 'k' centroids, and then assigns each data point to the closest cluster while minimizing the count of the centroids.

Preprocessing:

1. Normalized the data to bring all the data to a common scale using from sklearn's Normalizer.
2. Then I tried to create the image from the pixel's values provided from 0 to 255, of size 28*28 pixels using cv2 library. Then I found multiple representation of single digit which can produce lots of difficulty in clustering, having lots of pixels as 0.
3. To Make different representation of same digit more similar, I tried averaging all the pixel values of Image. This resulted the image becoming more smoother, and further applied Gaussian smoothening in order to remove noise. However, it reduced the sharpness, but they are well generalized in terms of pixel colors.



Dimensionality Reduction:

For Dimensionality Reduction I have applied:

1- **Truncated SVD:** The truncated SVD algorithm produces matrices with the given number of columns. By performing linear dimensionality reduction and using truncated singular value decomposition (SVD), it works better when working with sparse matrices for feature output and minimizes the amount of output.

2- **T-SNE:** dimensionality was decreased using the t-SNE method. T-distributed Stochastic Neighbor Embedding is also known as nonlinear dimensionality reduction. This shows that an algorithm can be used to divide data that cannot be divided by a straight line using Student T distribution. Its primary job is to convert high-dimensional data to low-dimensional spaces.

In order to scale each feature to a range between 0 to 1(Iris dataset) and 0 and 1000(Image dataset), I additionally used MinMaxScaler from the sklearn.preprocessing package. To fit the set's range of 0 to 1(Iris dataset) and 0 and 1000(Image dataset), this estimator scales and transforms each feature independently. If there were any entries in the dataset with null values, it also checks them.

Iris Dataset

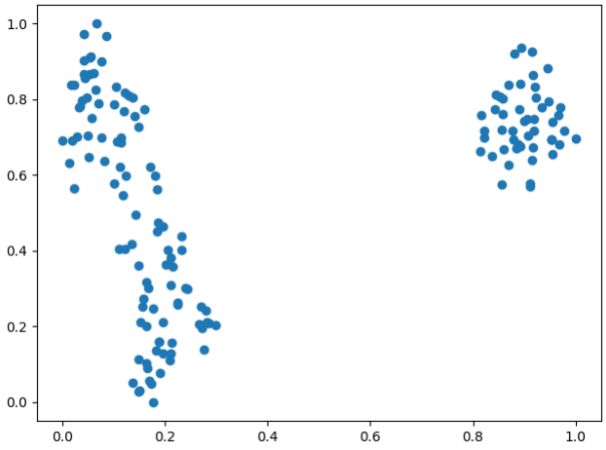
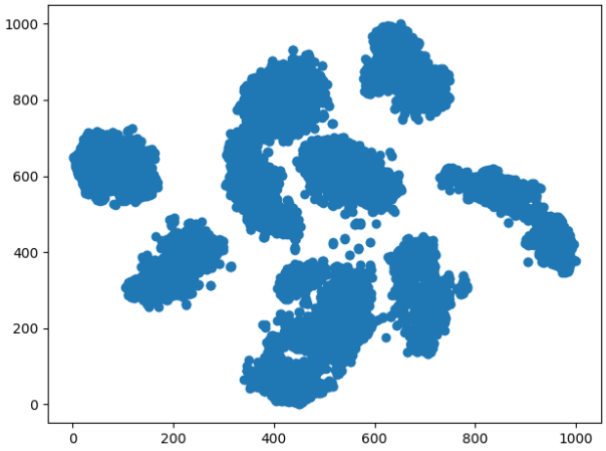


Image Dataset



After applying preprocessing and dimensionality reduction the clusters are clearly visible as above.

Algorithm:

For initializing first point which act as centroid, I tried random initialization, but there were the instances of bad initialization. Then I have tried random numbers from the data points, which performed better because centroid will lie within the region of data points.

Now, here are the steps to implement K-Means algorithm.

Step 1*:* select first centroid point randomly from within the data points.

Step 2: Calculate the Euclidean distance between each point in the dataset and the chosen centroid. We can calculate xi point's separation from the furthest centroid using

$$d_i = \max_{c_j : 1 \to m} || x_i - c_j ||^2$$

d_i: Euclidean Distance of x_i point from farthest centroid

m: Number of the centroids already picked up.

Step 3: Assign each data point to the closest centroid, then create a collection of all the points in that cluster.

Step 4: Compute mean of all collections of cluster containing data points, and designate this point (x_i) as the new centroid, which has the highest probability proportional to d_i.

Step 5: Repeat the above two steps till you find k-centroids.

Readings:

For Iris Dataset:

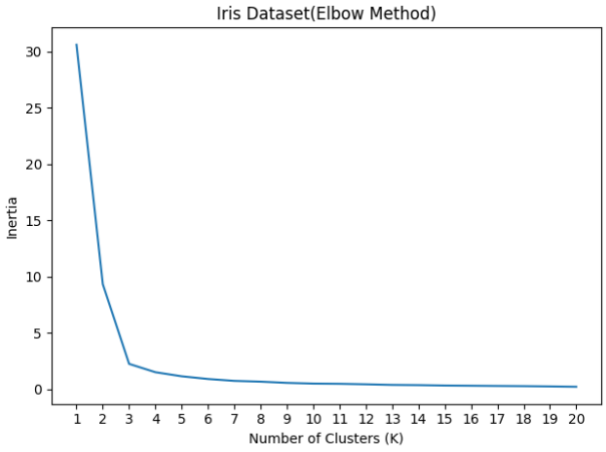
Number of Clusters	Silhouette score
2	0.67747307
3	0.73029447
4	0.64676136

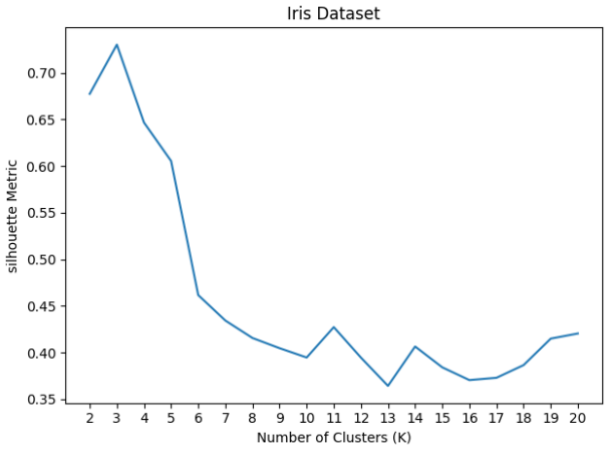
5	0.60551053
6	0.46172377
7	0.43443653
8	0.41571778
9	0.40477294
10	0.39477944
11	0.42737275
12	0.3947356
13	0.36433402
14	0.40659797
15	0.38425833
16	0.37054938
17	0.37302515
18	0.3866568
19	0.41503617
20	0.42052376

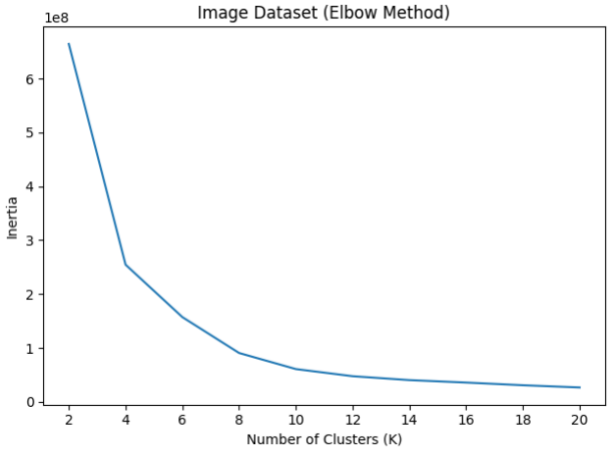
For Image Dataset:

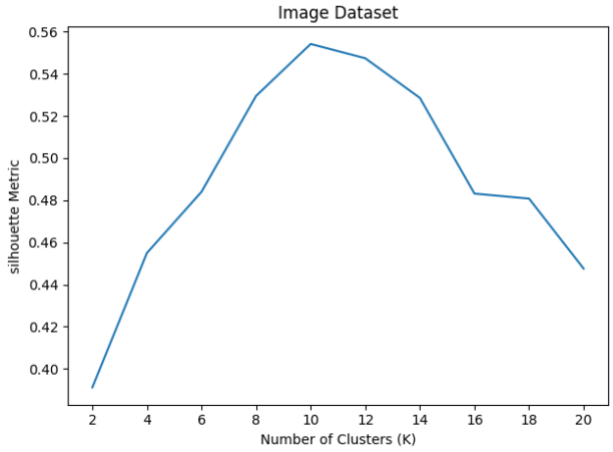
Number of Clusters	Silhouette score
2	0.39114857
4	0.4549693
6	0.48398995
8	0.52951556
10	0.5541511
12	0.5473621
14	0.52854955
16	0.48313844
18	0.48076737
20	0.44755903

Graph:









Observation and Inference:

1- Selecting centroid from within the data points performed better because it reduced chance of false selection and as we know the centroid will mostly lie within the region of cluster itself.

2- The number of times the k mean algorithm is iterated depends on whether the centroid is present for each of the k clusters properly.

3- In case of Iris dataset, I ran K-means clustering for cluster size 1 to 20, calculated inertia for each and calculated centroid based on lowest inertia. Also plotted silhouette score of each cluster, which displays lower inertia and highest Silhouette score at K=3. Which shows this data set is more optimal with 3 numbers of cluster.

4- Similarly for Image data, I ran K-means for cluster size 2 to 20 with increments of 2. With Each K I ran 10 iteration and finally chosen best centroid values based on lowest inertia. And then again plotted silhouette score with respect to cluster size and this Shows the lower inertia with the shortest K and highest Silhouette score often produces better outcomes. The Image data set's lowest inertia and maximum silhouette score are obtained with K = 10.

5- I also applied PCA, which seeks to locate principal components (PCs), also known as linearly uncorrelated orthogonal axes, in the m-dimensional space and projects the data points onto those PCs. but got better V measure with Truncated SVD on Miner2 so I have selected that.

Conclusion:

K-Means clustering algorithm was successfully implemented using Python, and the image data of the digits was labeled to the appropriate cluster. The data can be properly labeled using the model.

References:

https://distill.pub/2016/misread-tsne/
https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8