HomeWork-5

K-Means clustering

Before understanding k-Means first learn about few basic terminologies and why it needed at first place.

Unsupervised learning: there is no output variable to guide the learning process, and data is explored by algorithms to find patterns. Since the data has no labels, the algorithm identifies similarities on the data points and groups them into clusters.

From the universe of unsupervised learning algorithms, K-means is probably the most recognized one. This algorithm has a clear objective: partition the data space in such a way so that data points within the same cluster are as similar as possible (intra-class similarity), while data points from different clusters are as dissimilar as possible (inter-class similarity).

In **K-means**, each cluster is represented by its centre (called a "centroid"), which corresponds to the arithmetic mean of data points assigned to the cluster.
A **centroid** is a data point that represents the centre of the cluster (the mean), and it might not necessarily be a member of the dataset. This way, the algorithm works through an iterative process until each data point is closer to its own cluster's centroid than to other clusters' centroids, minimizing intra-cluster distance at each step.

Part-a

For this I implemented K-means using following concepts: -

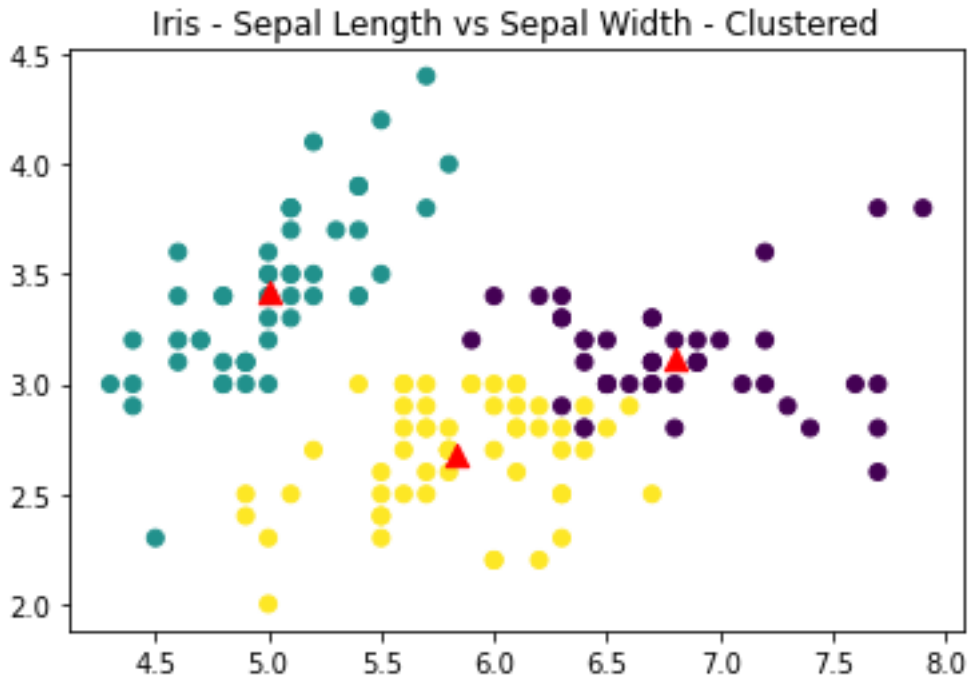
I used centroids from the given data set in the form of complete row with the help of random.choice function which gives me the index at random

rand_index = np.random.choice(Test_data.shape[0], K)

This random index I used to assign centroids which i used to calculate the min distance from remaining data set to assign it to cluster against each feature by giving axis=1. For which I used cdist and argmin to give me the min value.

labels = np.argmin(cdist(Test_data, centroids,'seuclidean'), axis=1)

I am running iteratively this in an infinite loop which breaks when the centroids stop updating.



Part-B

For this, first I saw the test data description and I can clearly saw that there are so many features that doesn't even contribute anything to the clustering which is a clear sign to use dimensionality reduction.

```
Test_data.describe()
```

Then I tried using different standardization like standard scaler which didn't give me good accuracy.

```
scaler=StandardScaler()

Test_data_scaled=scaler.fit_transform(Test_data)
```

Then I tried Normalization to see the effect on the clustering by using like this

```
Data_normalizer = Normalizer(norm='l2').fit(Test_data)

Data_normalized = Data_normalizer.transform(Test_data)
```

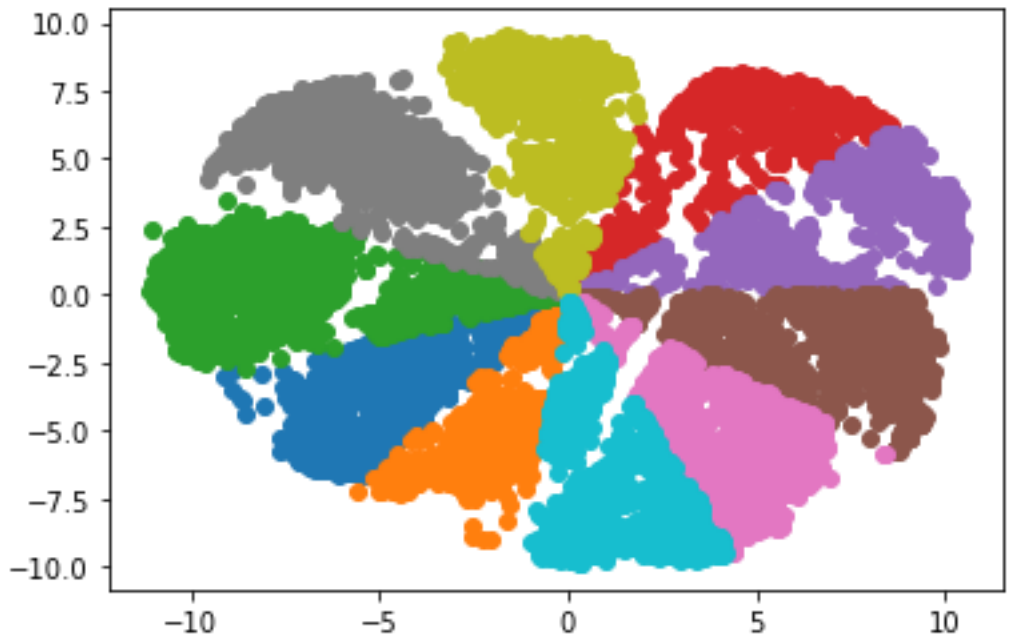
After completing the preprocessing part I tried using PCA with different n_components values to see the accuracy and I found out that it works best around 25 components and passed that data_value to my K_means algorithm that I built in part-a, but overall my accuracy isn't that good yet.

Data Set	With standardizing	Without standardizing
Iris data set	0.81	0.69
Image Data set	0.75	0.60

In the end I tried using TSNE which gives me better result as compare to PCA when passing the data_value in my K-Means algorithm.

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)

tsne_results = tsne.fit_transform(Data_normalized)



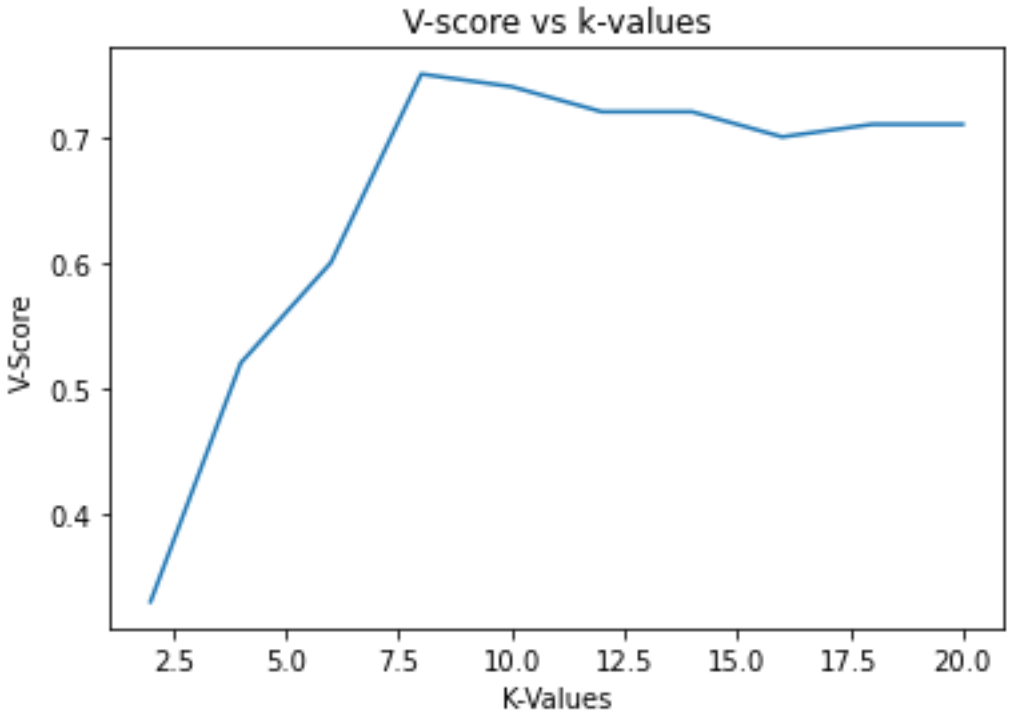
This is the final clustering looks after using TSNE with using normalized data

In the end I got this clusters in the labels variable which I plotted with the centroids in it marked as red triangles.

I got this variation in V-score against the different K-Values ranging from 2 to 20.

As shown below x belong to K-values and y belong to the V-score I got after running my k-means against each individual value of K.

X=K-Values	Y=V-Score
2	0.35
4	0.52
6	0.60
8	0.75
10	0.74
12	0.72
14	0.72
16	0.70
18	0.71
20	0.71



Username-rsharm8

Result-:

TEST SET	Rank	V-Score
IRIS Data Set	139	0.81
Image Data Set	161	0.75