# HW3 Report

**Miner2 username**: hkhurana
**Mason userID**: hkhurana (G01229319)
**Best Public Score (Iris)**: 0.86
**Best Public Score (Image)**: 0.81

I approached the assignment in the following three steps:

1. Data Preprocessing
2. Implementing the K-Means algorithm
3. Cluster Analysis

I first worked on the Iris dataset to implement and test the K-Means algorithm, and then later used this implementation to cluster the MNIST image dataset provided in Part 2.

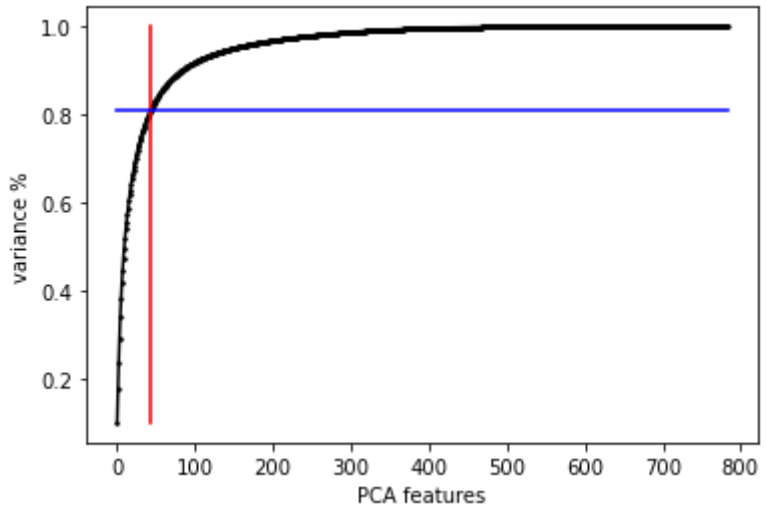
**TLDR**:
I first implemented the KMeans algorithm and worked with the Iris dataset, and a naive implementation gave me a V-measure of around 0.7. Later, I plotted the relationship between every 2 features of the dataset, and found that petal-width and petal-length clustered the dataset pretty well. Hence, I reduced the dimensions from 4 to the 2 I selected (petal-width and petal-length), and achieved a score of 0.86.

For part 2, I tried running the K-Means algorithm on the entire 10000x784 dataset, but it took a lot of time to execute, and it gave me an unsatisfactory score of around 0.5. Then, I tried applying dimensionality reduction techniques like PCA and t-SNE. This allowed me to reduce the number of features from 784 to 2. This allowed me to define my clusters well and visualize them on a 2D plane, as well as increased the V-measure to around 0.8. Proper selection of parameters yielded my highest V-measure of 0.81.

## Data Preprocessing:

First I imported all the essential libraries I would be needing for the program. Then I read the unlabelled data provided for clustering.

Then I did some **checking for null values** which turned out to be zero. Then, next for the image dataset, I performed dimensionality reduction to reduce the number of features to a reasonable amount. First, I used PCA to perform a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. I chose my variance to be around 80%, which yielded the reduction of number of features from 784 to around 40.



```
# PCA

pca = PCA(n_components=int(np.floor(y_interp)))
principalComponents = pca.fit_transform(data)

PCA_components = pd.DataFrame(principalComponents)
PCA_components.shape
```

```
(10000, 42)
```

Then, I applied t-SNE on the resulting PCA data which further reduced the number of features to 2. t-SNE, in simple terms, maps the multi-dimensional data to a lower dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features. This preprocessed data was then fed to the K-Means algorithm which is explained in the next section.

## Implementing the KMeans algorithm:

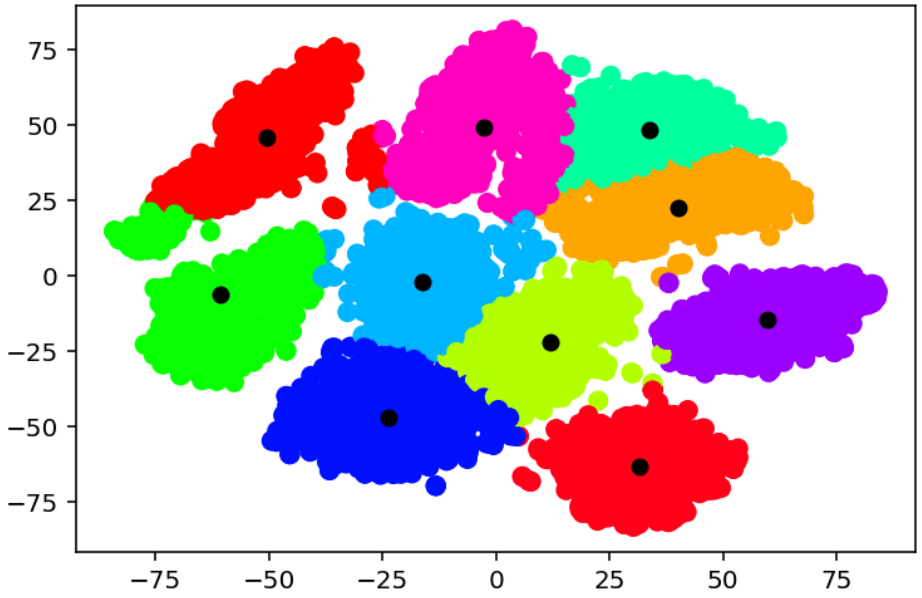
I divided the implementation of the algorithm into 2 main functions:

a. To calculate the initial centroids
b. The actual K-Means algorithm

```
def initialize_centroids(data, k):
    # no of attributes
    n = np.shape(data)[1]
    # initialize centroids as zero matrices
    centroids = np.mat(np.zeros((k,n)))

    # choose random centroids (PCA)
    for j in range(k):
        centroids[j,:] = data.iloc[np.random.randint(np.shape(data)[0])]

    return centroids
```

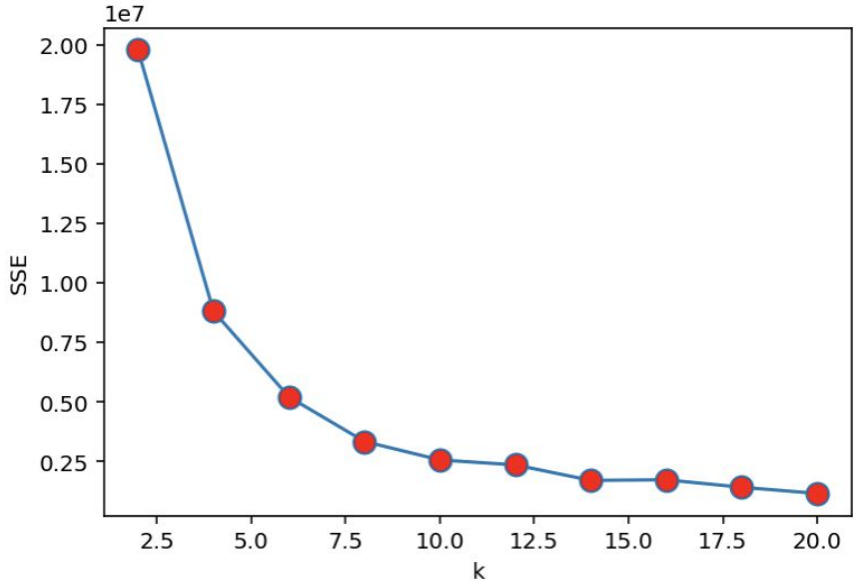
```
def kmeans(data, k):

    m = np.shape(data)[0]

    # to hold cluster assignments - cluster-index and (distance from cluster)^2
    cluster_assignments = np.mat(np.zeros((m, 2)))

    # initialize centroids
    cents = initialize_centroids(data, k)

    # preserve original centroids
    cents_orig = cents.copy()

    changed = True
    num_iter = 0

    # loop runs till no cluster assignment has changed
    while changed:

        changed = False

        for i in range(m):

            min_dist = np.inf
            min_index = -1

            for j in range(k):

                dist_ji = euclidean_dist(cents[j,:], data[i,:])
                if dist_ji < min_dist:
                    min_dist = dist_ji
                    min_index = j

            if cluster_assignments[i, 0] != min_index:
                changed = True

            cluster_assignments[i, :] = min_index, min_dist**2

        # update centroid location
        for p in range(k):
            indices = [i for i, x in enumerate(cluster_assignments) if x[0,0] == int(p)]
            cents[p] = np.mean(data[indices], axis=0)

        # keeping count of iterations
        num_iter += 1

    return cents, cluster_assignments, num_iter, cents_orig
```

## Cluster Analysis:




Following is the plot between SSE for different values of k:

**For MNIST Image dataset :**




**For Iris dataset :**