Guided Assignment On Unsupervised Learning

Name: **Shreya Shrivastava**

Course: **Artificial Intelligence And Machine Learning**

Batch Four

Duration: 12 Months

Problem statement: Take a bright colorful image (Eg: image having fruits in it) and implement **Image Segmentation using K-Means**.

Prerequisites:

The libraries as well as things required in order for the program to work:

I. **Python 3.6** : The following url https://www.python.org/downloads/ can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic. Second option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url : https://www.anaconda.com/download/

II. **OpenCV :** OpenCV can be downloaded from the following url: https://sourceforge.net/projects/opencvlibrary/. It is strongly recommended to download OpenCV in a virtual environment.

III. **ADDITIONAL PACKAGES :** You will also need to download and install below 3 packages- numpy, scikit learn and matplotlib after you install either python or anaconda from the steps above. If you have chosen to install python 3.6,then run the following commands in command prompt/terminal to install these packages :

NUMPY: pip install -U numpy

MATPLOTLIB: pip install -U matplotlib

If using Anaconda then run the following commands in anaconda prompt to install these packages:

NUMPY: conda install -c anaconda numpy

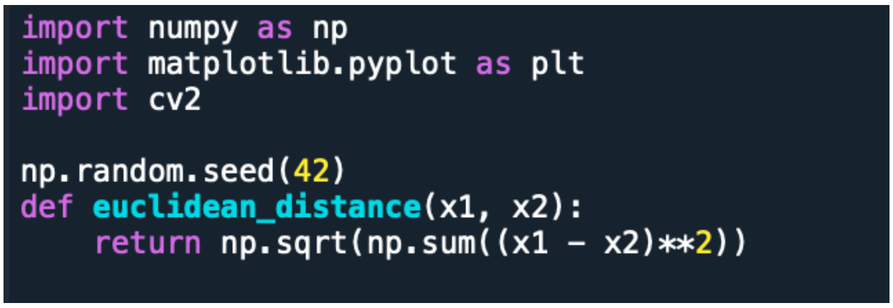
MATPLOTLIB: conda install -c anaconda matplotlib

IV. METHODS USED:

A. K MEANS CLUSTERING ALGORITHM

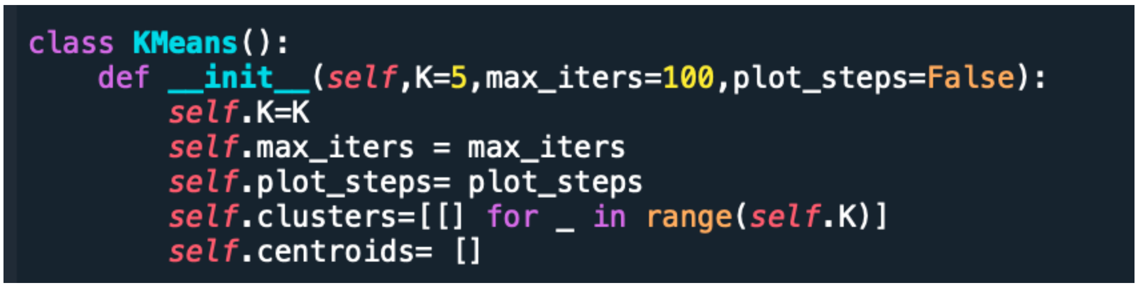
THE PROJECT :

1. Importing the libraries and fixing a seed. We then proceed to define a small helper function that will help us calculate the euclidean distance between two points.



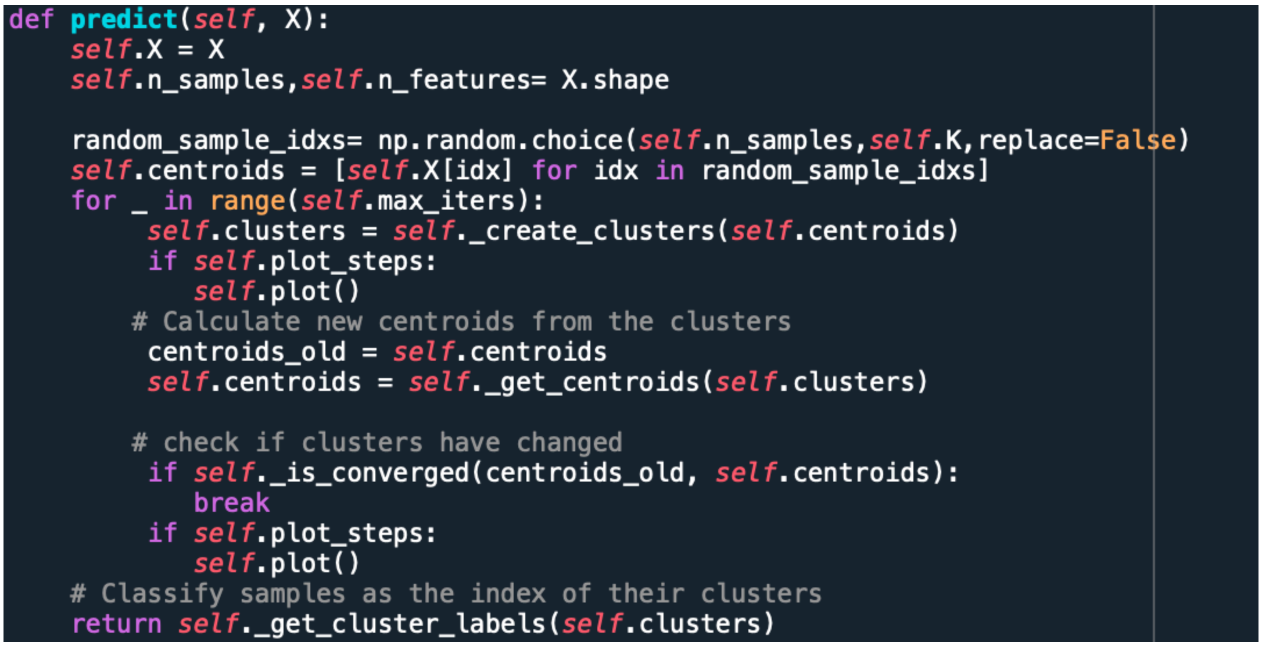
```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

np.random.seed(42)
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
```

2. Next we create a class KMeans and initialise our main attributes. We create empty lists for the clusters as well as the centroids.



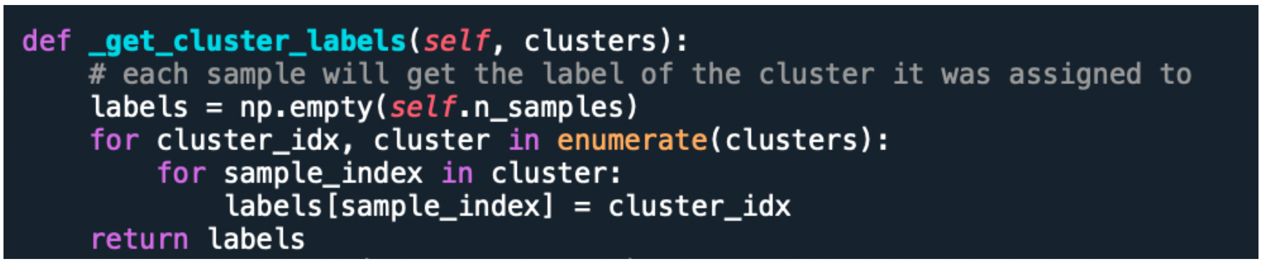
```python
class KMeans():
    def __init__(self,K=5,max_iters=100,plot_steps=False):
        self.K=K
        self.max_iters = max_iters
        self.plot_steps= plot_steps
        self.clusters=[[] for _ in range(self.K)]
        self.centroids= []
```

3. Here, we define a predict function and initialise the centroids as well, assigning clusters to the pixel values.



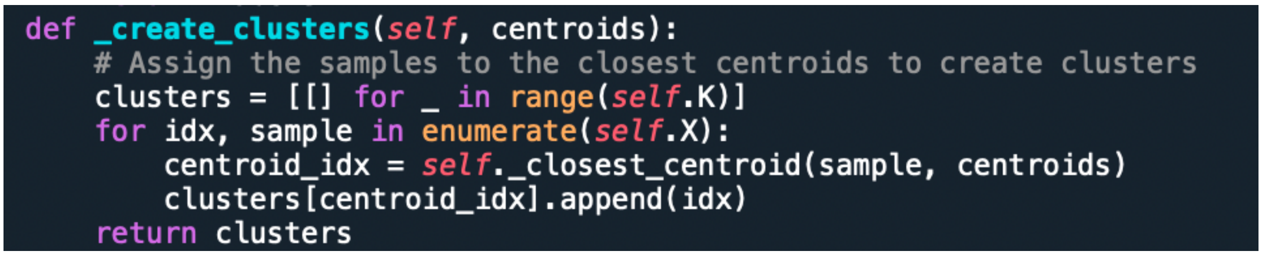
```python
def predict(self, X):
    self.X = X
    self.n_samples,self.n_features= X.shape

    random_sample_idxs= np.random.choice(self.n_samples,self.K,replace=False)
    self.centroids = [self.X[idx] for idx in random_sample_idxs]
    for _ in range(self.max_iters):
         self.clusters = self._create_clusters(self.centroids)
         if self.plot_steps:
            self.plot()
        # Calculate new centroids from the clusters
         centroids_old = self.centroids
         self.centroids = self._get_centroids(self.clusters)

        # check if clusters have changed
         if self._is_converged(centroids_old, self.centroids):
            break
         if self.plot_steps:
            self.plot()
    # Classify samples as the index of their clusters
    return self._get_cluster_labels(self.clusters)
```

4. Assigning the labels to the samples.



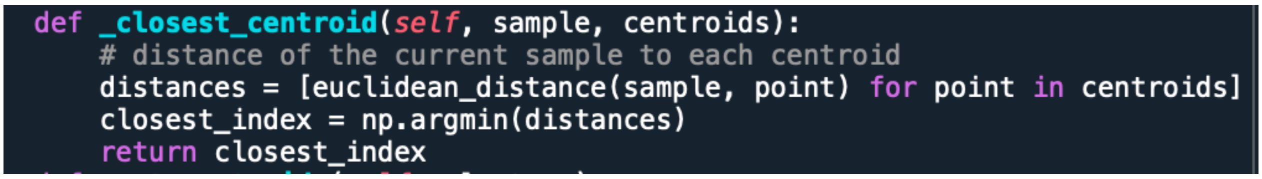
```python
def _get_cluster_labels(self, clusters):
    # each sample will get the label of the cluster it was assigned to
    labels = np.empty(self.n_samples)
    for cluster_idx, cluster in enumerate(clusters):
        for sample_index in cluster:
            labels[sample_index] = cluster_idx
    return labels
```

5. Iteratively updating the centroids.



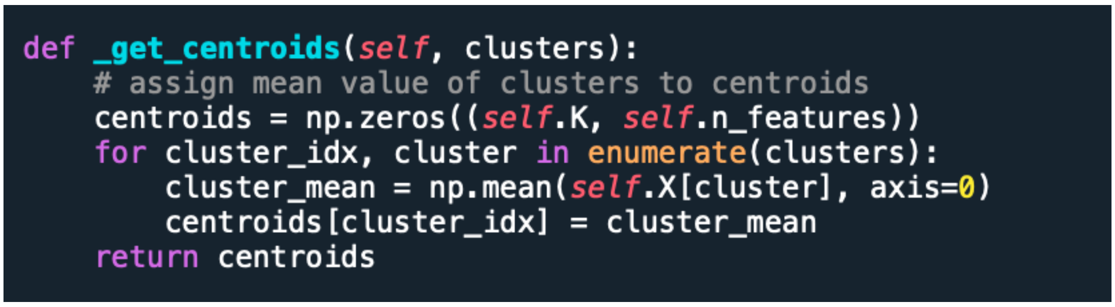
```python
def _create_clusters(self, centroids):
    # Assign the samples to the closest centroids to create clusters
    clusters = [[] for _ in range(self.K)]
    for idx, sample in enumerate(self.X):
        centroid_idx = self._closest_centroid(sample, centroids)
        clusters[centroid_idx].append(idx)
    return clusters
```

6. Finding the distance of the given sample with each of the centroids and returning the index of the centroid which is at the minimum distance.



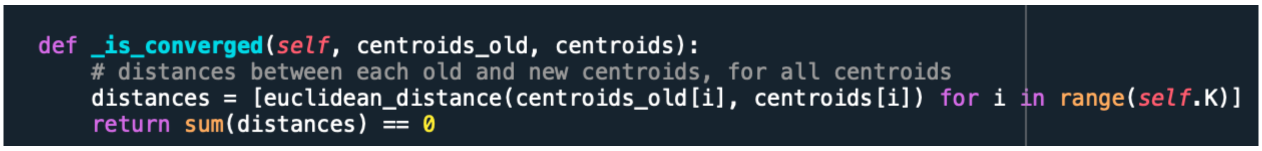
```
def _closest_centroid(self, sample, centroids):
    # distance of the current sample to each centroid
    distances = [euclidean_distance(sample, point) for point in centroids]
    closest_index = np.argmin(distances)
    return closest_index
```

7.Assigning mean value of clusters to the centroid.



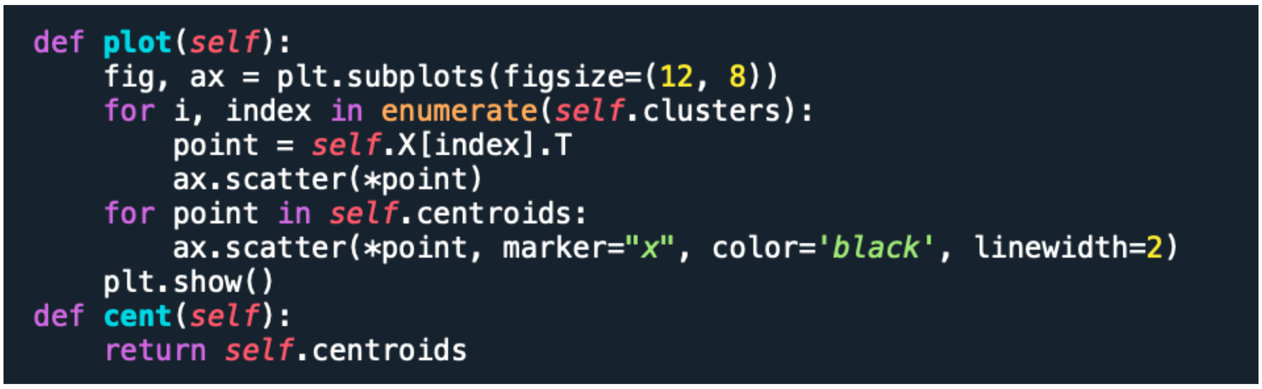
```
def _get_centroids(self, clusters):
    # assign mean value of clusters to centroids
    centroids = np.zeros((self.K, self.n_features))
    for cluster_idx, cluster in enumerate(clusters):
        cluster_mean = np.mean(self.X[cluster], axis=0)
        centroids[cluster_idx] = cluster_mean
    return centroids
```

8. Checking for convergence.



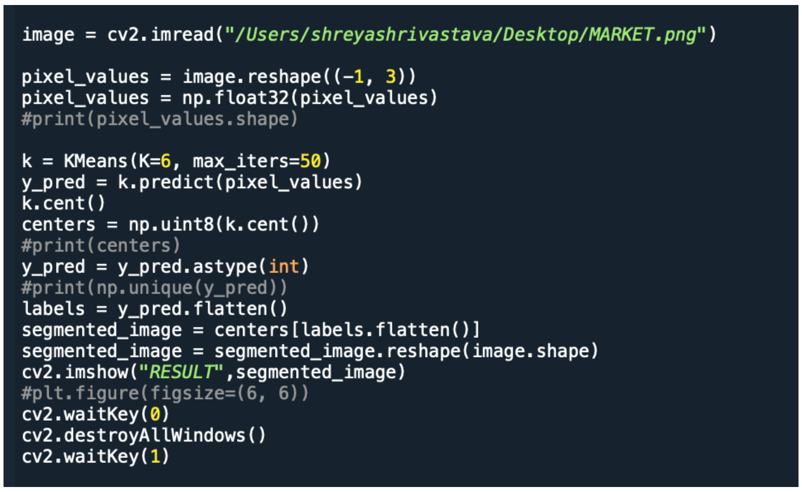
```
def _is_converged(self, centroids_old, centroids):
    # distances between each old and new centroids, for all centroids
    distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)]
    return sum(distances) == 0
```

9. Cent function will return us the centroids whereas the plot function will display the image

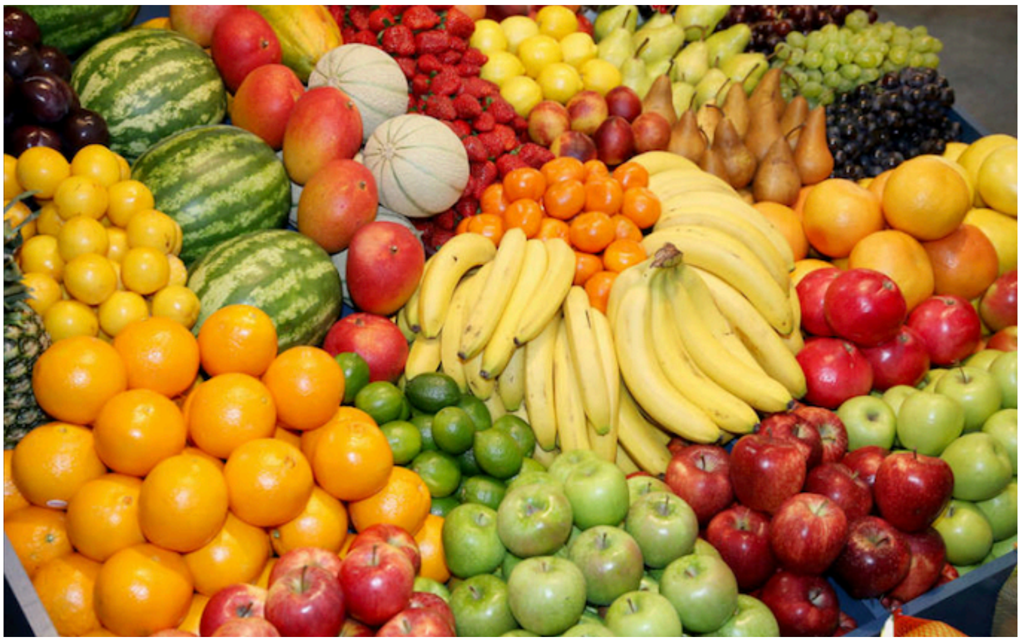


```
def plot(self):
    fig, ax = plt.subplots(figsize=(12, 8))
    for i, index in enumerate(self.clusters):
        point = self.X[index].T
        ax.scatter(*point)
    for point in self.centroids:
        ax.scatter(*point, marker="x", color='black', linewidth=2)
    plt.show()
def cent(self):
    return self.centroids
```

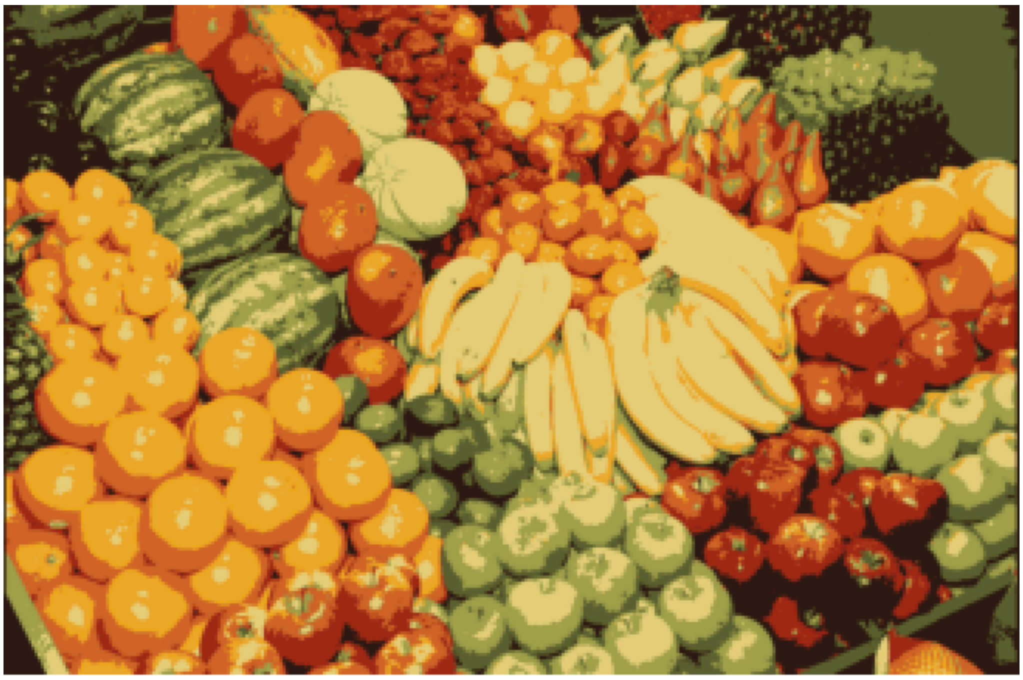
10. Running the final function



```
image = cv2.imread("/Users/shreyashrivastava/Desktop/MARKET.png")

pixel_values = image.reshape((-1, 3))
pixel_values = np.float32(pixel_values)
#print(pixel_values.shape)

k = KMeans(K=6, max_iters=50)
y_pred = k.predict(pixel_values)
k.cent()
centers = np.uint8(k.cent())
#print(centers)
y_pred = y_pred.astype(int)
#print(np.unique(y_pred))
labels = y_pred.flatten()
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image.shape)
cv2.imshow("RESULT",segmented_image)
#plt.figure(figsize=(6, 6))
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1)
```



ORIGINAL IMAGE



FINAL RESULT {K=7}