Help Guide

K-means

Exercise: Create a Python script file and perform the following tasks:

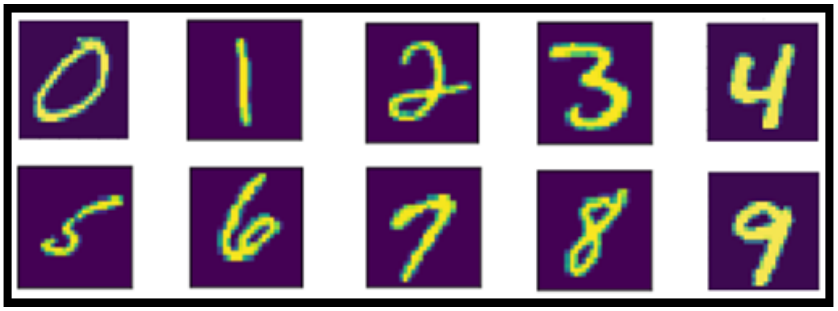
Pseudocode:

```
class K_means{

  constructor(k){
        this.k = k
  }

  fit(data){
     initialize k centroids with random data points
     while(True){
          finding the distance between centroids and all the data points in D
          array
          each point belongs to the cluster which has the lowest distance from
          its centroid
          calculate new centroids which is the result of the average of points of
          the cluster of previous_centroids
          if(centroids == previous_centroids){
                  break
          }
     }
     return centroids
  }
}
```

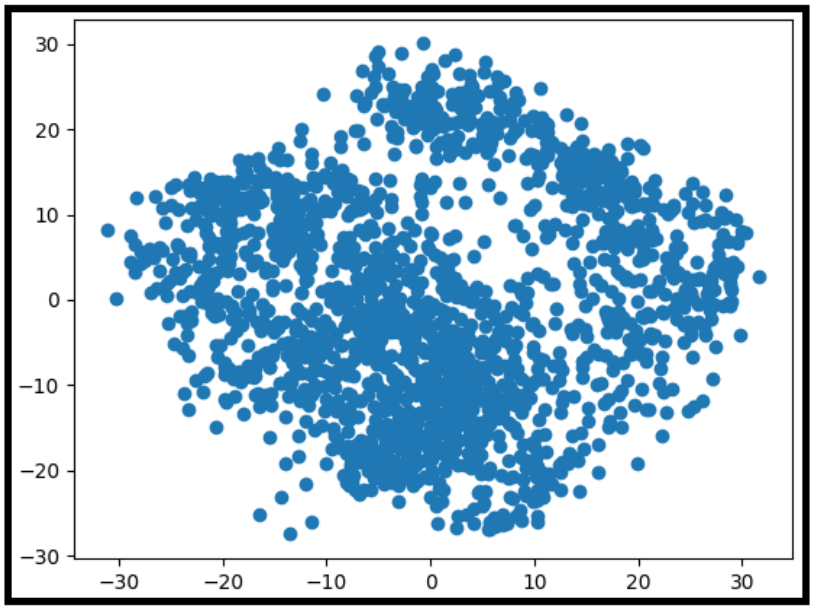
1. Import libraries:
   - numpy: For calculations of arrays.
   - cdist: To calculate Euclidean distance.
   - load_digits: Dataset with pictures of numbers 0 to 9.
   - PCA: To reduce the dimension of picture from 64 to 2.
   - Matplotlib: To plot points in 2-dimension figure.

2. Create a class of K-means with the name k_means, contains a constructor _init_ and a function fit. The constructor takes k which is an int and the number of unique labels (also the number of clusters). Function fit takes 2 parameters data and k. Data is a numpy.ndarray of 2-dimensions.

3. Load and use the dataset of sklearn load.digits. This dataset contains 1797 samples of 64 (8x8) pixels pictures of numbers 0 to 9. Some of them are shown below



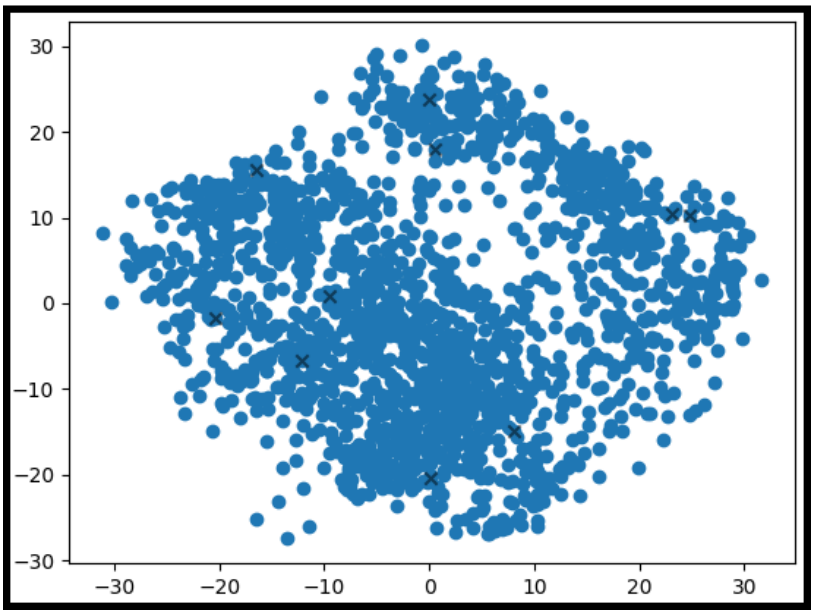
4. Load and use sklearn method PCA (Principal component analysis) which transforms the dimensions of pictures from 1797x64 to 1797x2. The reason of doing that is to plot the pictures as points in 2-dimensions board. Load and use the library matplotlib to plot the points. You should do it like the picture below.



5. Select k random points from data as centroids. Plot the centroids with mark ‘x’. You should do it like the picture below.

   Hint: You can use properly the method random.choice of numpy library.

   Warning: Centroids should be different one to each other.



6. Find the distances between centroids and all the data points. The metric for calculating the distances will be the Euclidean metric. For example, for points $A(x_A, y_A)$ and $B(x_B, y_B)$, distance d will be $d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$.

   Hint: You can use properly the method cdist of scipy library.

7. Assign all the points to the closest centroid, more specific the one with the minimum distance.

   Hint: You can use properly the method argmin of numpy library.

8. Calculate the new centroids. The calculation comes from the mean value of points for each cluster.

   Hint: You can use properly the method mean of numpy library.

9. The algorithm ends when newly centroids are the same with the previous ones. Another way to stop the algorithm faster to solve the problem of waiting too many times, which depends in sample size is by defining a number of iterations. In our case dataset is short and we don't need to consider that, but you should implement both ideas in your code.

10. The K-means function returns the labels of data points, more specific a list of values from 0 to 9.

11. Plot the diagram of points and centroids with unique color for each cluster. Make a .gif file of the changes of centroids and points that change during the process. You should do it like the picture below.

    Hint: Plot a figure of each iteration and save them to a file. Use the site Animated Gif Maker to upload all figures and make the gif.