Principles of Data mining

Under the supervision of: Dr. E. Nazerfard

Assignment 2

Spring 2020

Question 1

One of the problems of KMeans, is that with increasing size of the dataset being analyzed, the computation time of K-means increases because of its constraint of needing the whole dataset in main memory. Investigate how “Mini-Batch KMeans” solves this problem? Explain disadvantages of this algorithm.

Question 2

As you know KMeans clustering quality and number of iterations for the algorithm to converge, depends highly on the initial location of centroids. In the simplest form of KMeans, centroids are assigned randomly at the beginning. Express 2 ideas to improve the initial placement of Centroids.

Question 3

Which of the following is/are true about DBSCAN clustering algorithm? Explain your Answer for each option.

A) For data points to be in a cluster, they must be in a distance threshold to a core point

B) It has strong assumptions for the distribution of data points in dataspace

C) It has substantially high time complexity of order O(n3)

D) It does not require prior knowledge of the no. of desired clusters

E) It is robust to outliers

Question 4

The table below is a distance matrix for 6 objects.

	A	B	C	D	E	F
A	0					
B	0.12	0				
C	0.51	0.25	0			
D	0.84	0.16	0.14	0		
E	0.28	0.77	0.70	0.45	0	
F	0.34	0.61	0.93	0.20	0.67	0

A) Show the final result of hierarchical clustering with single link by drawing a dendrogram.

B) Show the final result of hierarchical clustering with complete link by drawing a dendrogram.

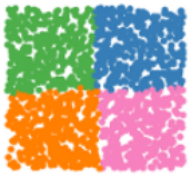
Note: The dendrogram should clearly show the order in which the points are merged.

Question 5

Consider the following images showing data points. Which among the clustering algorithms will perform well in accurately clustering data? K-Means, DBSCAN, or both? Explain your answer.

A) 

B) 

C) 

D) 

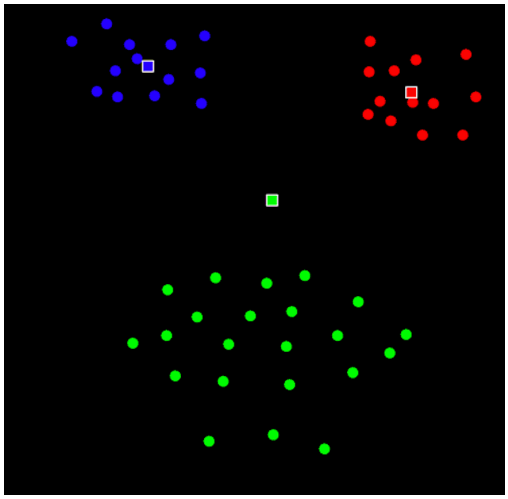
Implementation 1: KMeans

In this part, you will implement and use the K-means clustering algorithm. First you will learn to cluster a simple 2D dataset, next you will learn a method to evaluate the performance of clustering and finally you will learn about the restrictions of KMeans by running it on a complex dataset.

As discussed in the class, the main idea behind KMeans is an iterative process that starts by guessing the initial cluster centers, and then improves this guess by repeatedly assigning data points to their closest cluster center and then recalculating the centers based on the assignment. The pseudo-code of K-means is as follows:

```
Input:
    D= {t1, t2, …. Tn }   // Set of elements
    K                     // Number of desired clusters
Output:
    K                     // Set of clusters
K-Means algorithm:
   Assign initial values for m1, m2,…. mk
   repeat
     assign each item ti    to the clusters which has the closest mean;
     calculate new mean for each cluster;
   until convergence criteria is met;
```

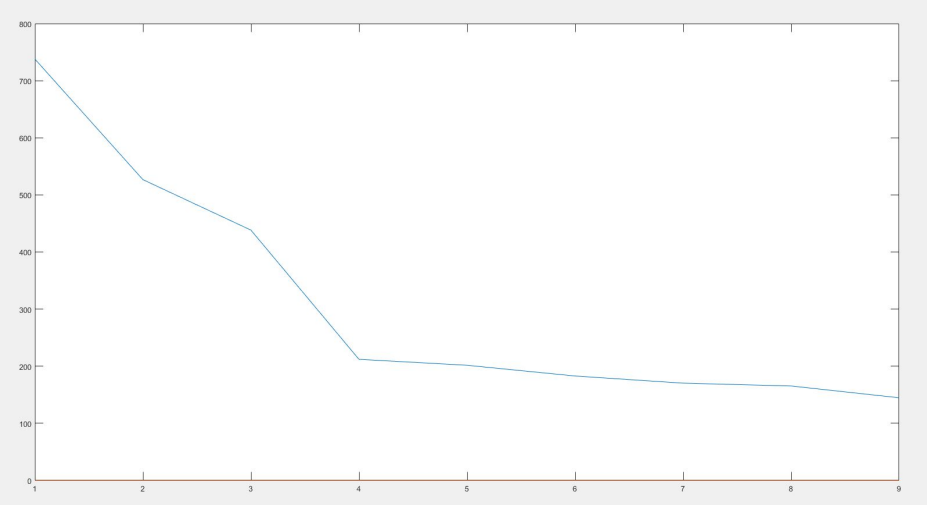
A) After completing the algorithm, run it on Dataset1. Set number of iteration to at least 15 and run K-means with k=2, 3, 4. After each run plot the clustered data points with each cluster having a different color. (you can use matplotlib to visualize the data)



B) After the clustering is done, compute for each cluster, the average distance between the cluster center and the data points in that cluster (this average distance is called cluster error).

C) compute the average cluster error and report it as the clustering error.

D) run the k-means with 0<k<15 on Dataset1 and compute the clustering error and plot these errors.

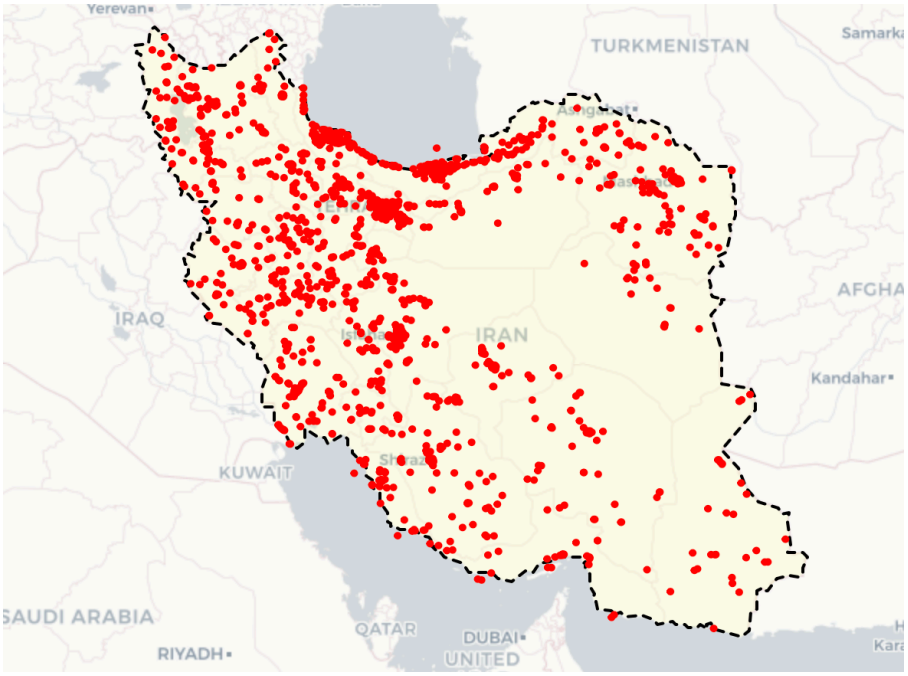


E) Use the “elbow” algorithm to find the optimum K.

F) Run the clustering once again on Dataset2, explain why KMeans fails on this dataset.

Implementation 2: DBSCAN

In the previous part, you got introduced to several inherent weaknesses of the K-means algorithm.
In this part, you start working with more sophisticated algorithm, called DBSCAN, which can somewhat handle problems mentioned above. Since DBSCAN is a little complicated you don't need to implement the algorithm in this part, and you can use available libraries (DBSCAN library in sklearn package is recommended) to complete this part.
Also, it is strongly recommended to use Jupyter notebooks to complete this part.



in this assignment, you've been supplied with geographical distribution of COVID-19 patients inside Iran (covid.csv). each row in the csv file represents a detected COVID-19 case, first column is the geographical latitude and second column is the geographical longitude.
for better visualization of geographical data we're using a python library called "folium".

You can install folium using pip: `pip install folium`

here is an example code for displaying a geolocation using folium:

```
import folium

# set iran as map starting point
m = folium.Map(location=[32.427910, 53.688046], zoom_start=5)

# mark an example location
loc = [35.703136,51.409126]
folium.Marker(location=loc).add_to(m)
```

for more information on folium see here.

As you already know DBSCAN requires 2 main parameters, eps and minPts.
eps is the maximum allowed distance between 2 data points in the same cluster, and minPts is the minimum number of data points to create a cluster. Using these two important parameters DBSCAN dynamically creates as many as needed clusters while discarding any clusters without enough data points and keeping the outliner data points out of any valid cluster. As a result, changing the value of these to parameters changes the final number of clusters and the way the algorithm works.
Our purpose is to use DBSCAN to find dense disease clusters inside Iran.

A) Load the dataset from csv file and plot it on map using folium. (for better visualization use folium circles)

```
folium.Circle(location=loc,
              radius=1,
              color="red",
              fill=True).add_to(m)
```

B) Run DBSCAN algorithm with arbitrary values for eps and minPts.

C) Fine-tune eps and minPts parameters so that each cluster only includes patients from heavily infected areas. these clusters must exclude outlier locations (Hint: Tehran and Qom are possible dense clusters)

D) Plot each cluster on map using a different color. Plot outliers with same color.

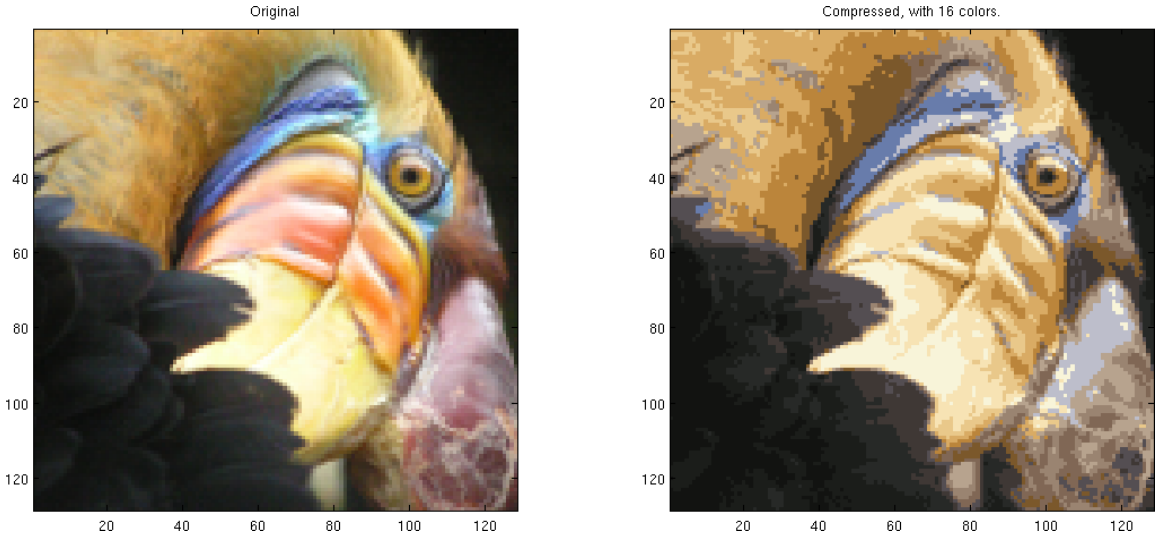
Implementation 3: Image Compression

In this part, you will use your previously written KMeans code for a useful real-world problem. In many RGB encodings each pixel is represented by 24 bits, 8 bits for each one of the main colors (Red, Green, Blue) ranging from 0 to 255, and therefore each pixel can have more than 16 million colors.
In this part, you will use K-means algorithm to reduce the number of colors to 16 (or 256) so that the color of each pixel can be represented by only 4 bits (or 8 bits). In fact, you only need to store the RGB values of the 16 (or 256) selected colors in an array, and for each pixel in the image you now need to only store the index of the color in the array. Since reducing the number of the colors result in lower quality for the image, we use K-means to find the 16 (or 256) colors that best group pixels in the image.

You can use the following python code to load, show an image inside a Jupyter notebook:

```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import image
img = image.imread('image.png')
plt.imshow(img)
plt.show()
```

Use the given code to load image.png. Note that by loading the image the variable img will be set to a 3D numpy array with 800 columns and 800 rows corresponding to the location of each pixel. The third dimension of this numpy array will hold the values of Red, Green, and Blue of the color.
In order to use the K-means Algorithm, that you've already implemented, on the image, first you need to reshape this 3D matrix into a 2D one in which every row represents a pixel.



The original image has 800x800 pixels and with 24 bits per pixel the image requires at least 800x800x24 bits to be stored. By reducing the number of colors to 16 and 256 the image only requires about 800x800x4 and 800x800x8 bits respectively to be stored.
By this point hopefully you have learnt the intuition behind the clustering and grasped a hands-on knowledge on how to use clustering for real world challenges.

In this section do the following steps:

A) Set K equal to 16 and run your KMeans algorithm on the image data.

B) Replace the RGB value of each pixel with the RGB value of the center of its cluster.

C) Set K equal to 256 and repeat the previous steps.

D) Show the result images alongside the original image, write compressed image to disk and compare its file size with the original image.

CAUTION

- For each part, write your codes in a .py (or .ipynb) file naming with the number of parts, and put it in the "supporting material" folder.

- For image compression part you must use the k-means that you have implemented yourself in the first part.

- Since your implementation of K-means and image compression will be tested by TAs on different data sets with different sizes, you need to make sure that your implementation is not dependent on the size of input data. Also try to use vectorized operation wherever you can to achieve higher performance and a slight bonus to your score.

- Deadline is on 7th of Ordibehesht and you will lose 10% of your grade after that on each day of delay.

- Report is an important part of your grade. So, write it completely and explain your analysis. Your report is only accepted in 'pdf' format. Put it in "report" folder. (There is no force on the language of the report)

- Put all your folders and files like the sample format in a "zip" file and upload it on moodle (http://courses.aut.ac.ir/)

- If you have any question regarding the assignment contact arminkz3@gmail.com

---

Please upload your homework in this format:

```
9******_FirstnameLastname_HW1.zip
├── [directory] Report
│   └── 9******_FirstnameLastname_Report1.pdf
└── [directory] Supporting_Material
    └── codes.py
```