ITS Assignment 2 Description

Under Supervision of

Dr. Zahra Amini

Teaching Assistant

Alireza Feyzyab

October 2023

This problem guides through an example of exploratory data analysis with clustering methods. You will implement a data exploration system that applies clustering methods to explore California's frequently visited locations from geo-referenced social media data. You will study the properties and computational requirements of some standard clustering methods on a medium-scale dataset.

Python is a recommended programming language for this assignment. **Scikit-learn** toolkit (https://scikit-learn.org/stable/index.html) contains the implementations of the required clustering methods and a brief description of the methods' parameters along with code examples (see Appendix 1 below).

Data description.

This assignment deals with the dataset of 100k tweets collected using the Twitter API over a period of time. The dataset is a JSON file named "tweets.json." The pair of (lat, lng) coordinates is associated with each tweet corresponding to the user's location (see Figure 1). The spatial extent of the dataset is the state of California. Other information in each tweet is its unique ID, the user_id, the timestamp, and textual content.

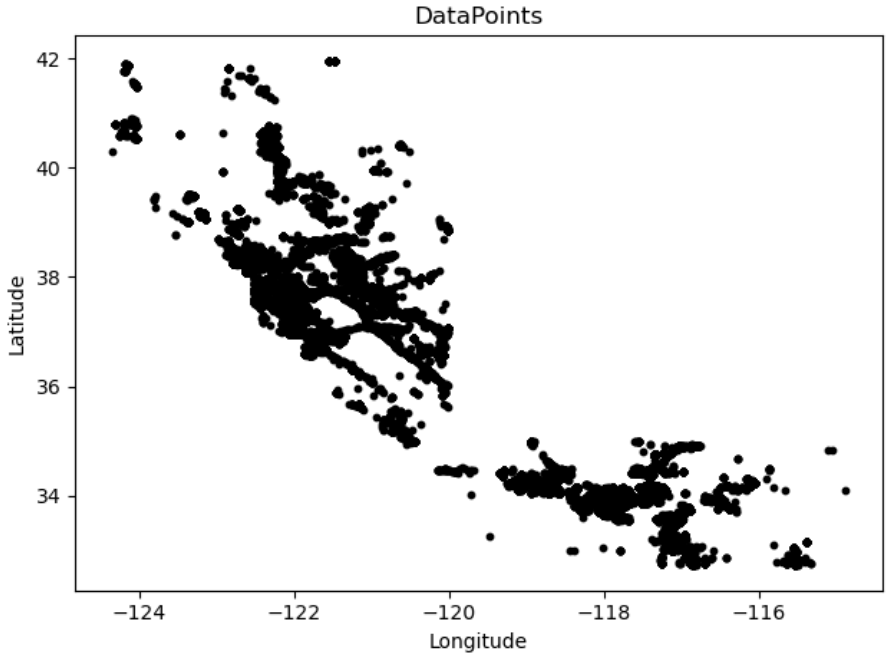

Figure 1: Plotted data points

You will work on **clustering** and **visualizing** the data samples in geographical space (2 dimensions).

Your submission will consist of a **single PDF document describing your results** and **a code** you used.

Part 1. Visualization

1. Plot all data points available based on the given coordinates in each tweet.
2. Prepare a brief write-up of Part 1 containing the execution time (see Appendix 1)

Part 2. Calculate data points density

1. In this section, you will calculate the density of data points within a 50 kilometers radius of the given point. The coordinates of the given center points are determined based on the last digit of your student ID in Table 1. Centroid centers are shown in Figure 2
2. You have to calculate distances in **meters.** To calculate distances in meters, use the **Haversine** function (see Appendix 1)

Table 1: Centroid Coordinates

Last digit of Student ID	Centroid Coordinates (degrees)
1	(34.212730341331145, -118.2859001567625)
2	(35.64882595915578, -120.85374495176804)
3	(37.225532146828996, -121.7293089322626)
7	(38.71838837734516, -122.87853161938223)
8	(39.38048783332028, -121.51380397819406)
9	(41.71838874769764, -122.34069473647976)

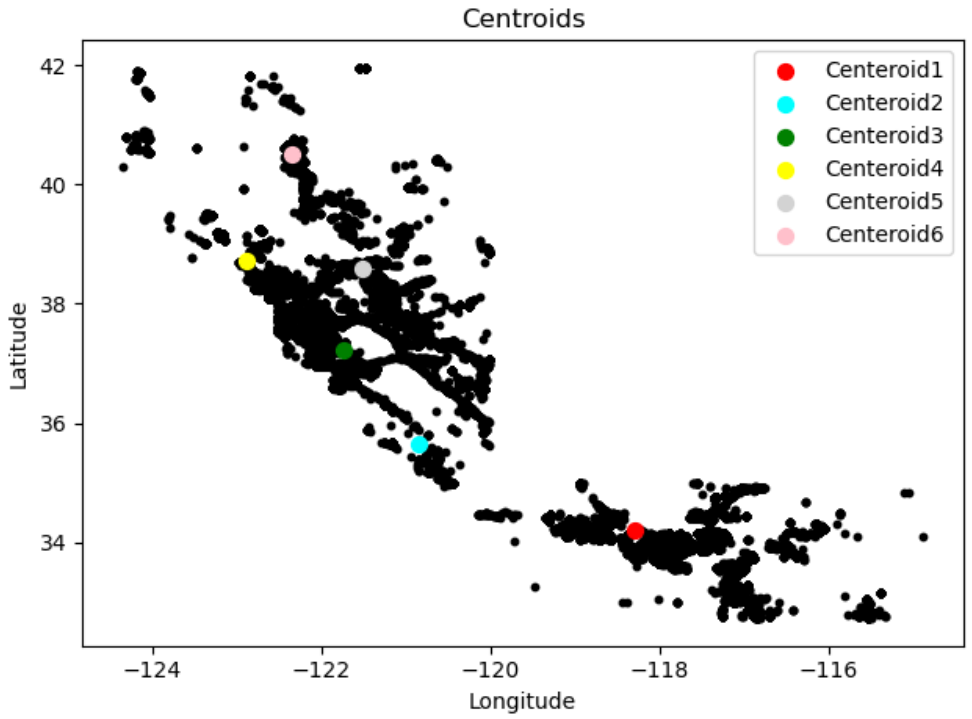

Figure 2: Centroids

Part 3. Clustering: the baseline.

In this part, you have to cluster the data points based on their locations. Since the variations in data points' coordinations are relatively small, transforming coordinations to meters can enhance the performance. As a result, you can create a coordination system based on the **relative distances** between the data points. To do this, you can use the **Haversine** function and create a virtual origin point (0, 0).
Hint: You can consider the origin point with the minimum latitude and the minimum longitude in all data points and calculate distances from this point.

1. **K-means Algorithm:**
   - Write a script that applies the **K-means** clustering method to the 100K dataset.
   - Compute the reference time of clustering samples into k=100 clusters.
   - Experimentally detect the maximum number of clusters *k* that can be handled by implementing the algorithm you are using (**Hint:** this can be confined by computational limits. If there are no limits, determine the maximum **rational** number of clusters that should be created, based on your results)
   - Detect the optimum number of clusters using the **elbow method** (see Appendix 2)

2. **Mini Batch K-means Algorithm:**
   - Write a script that applies the **Mini Batch K-means** method to the 100K dataset.
   - Select an appropriate value of batch size (Describe the reason in your documentation).
   - Compute the reference time of clustering samples into k=100 clusters.
   - Measure and note the gain in computational time.
   - Evaluate the maximum number of clusters *k* that can be handled by implementing the algorithm you are using (**Hint:** this can be confined by computational limits. If there are no limits, determine the maximum **rational** number of clusters that should be created, based on your results)
   - Detect the optimum number of clusters using the **elbow method** (see Appendix 2)

3. **DBSCAN Algorithm:**
   - Write a script that applies the **DBSCAN** method to the 100K dataset.
   - Fix the min number of samples in a cluster as 100.
   - Experimentally explore the influence of the connectivity threshold ε on the number of clusters detected by DBSCAN (**Hint**: this may significantly impact the number of non-included data points, i.e., noise points).
   - Compute the value of ε (call it **ε_100**) in DBSCAN resulting in approximately 100 clusters of a minimum of samples ('min_samples = 100') and the corresponding processing time

Prepare a brief write-up of Part 3 containing the following:
- Describe each step in your scripts.
- Plot essential graphs to show the impacts of parameters' variation (e.g., **elbow diagram**).

- Describe the reason for choosing parameters in each step (e.g., number of clusters, eps, and other parameters)
- Compare the results in each part and comment on the algorithms' efficiency based on the results and the computational time.
- Plot the final results in each algorithm and show the clusters with their centroids (Add your **name** and your **student ID** in the title).

Part 4. Clustering: scalability.

1. For K-means and Mini Batch K-means algorithms, run the experiments and plot the graphs representing the **computational time** as a function of:
   a) Number of data samples (consider the range of 100 to 100,000) for a fixed k =100
   b) Number of requested clusters k (consider the range of 2 to the **k_max** computed in part 3)
2. For the DBSCAN algorithm, plot the graphs representing computational time as a function of the number of samples (consider the range of 100 to 100,000) for a fixed 'ε= **ε_100**', 'min_samples = 100'.

- Include the graphs in the write-up of your submission. For each of the three methods, **extrapolate** the graphs and provide an estimated time required for your implementation to detect at least 100 clusters in a dataset of 1 million samples.

Part 5. Clustering: Finding relations

1. Choose **one** of the K-means or Mini Batch K-means algorithms and use your results in part 3 with **k=100** clusters.
2. Find the cluster that **contains** the centroid enlisted in Table 1 based on your student ID's last digit.
   Hint: The centroid is **not necessarily** a data point. In this case, you can find the nearest data point to the centroid and perform your analysis.
3. In the cluster found in section 2 of this part, search for the trending words or hashtags used in the tweets' text.
4. Find a relation between the used words/hashtags in the cluster and the cluster's geographical location. (e.g., find points of interest in the cluster using Google Maps)
   Hint: You may be unable to find pertinent points of interest in clusters with a limited number of data points. In this case, you can add **up to four adjacent clusters** to extend the number of data points and find more sound relations.
5. Describe the proposed relation and plot the cluster containing the suggested points of interest.

Appendix 1.

Loading data:

```
import json
with open('tweets.json', 'r') as f:
   tweets = json.load(f)
```

Distance Measurement:

```
from haversine import haversine
a = (37.44610, -121.88356)
b = (40.41085, -122.25005)
haversine(a, b, unit='m')

>>> 331185.41489855204
```

Find more about this function's arguments here:
https://pypi.org/project/haversine/

Time measurement:

```
import time
t0 = time.time()
# …
# your code
# …

t_compute = time.time()-t0
```

Clustering:

Example of using K-means and Mini Batch K-means in scikit-learn:
https://scikit-learn.org/stable/auto_examples/cluster/plot_mini_batch_kmeans.html#example-cluster-plot-mini-batch-kmeans-py

Example of using DBSCAN in scikit-learn:
https://scikit-learn.org/stable/auto_examples/cluster/plot_DBSCAN.html#example-cluster-plot-DBSCAN-py

Appendix 2.

The Elbow method:

The elbow method helps choose the optimum number of clusters in the K-means and Mini Batch K-means algorithms. This method defines the cost function as the sum of squared distances between data points and clusters' respective centroids. We call this cost function WCSS (Within-Cluster-Sum-of-Squares) and use it as a measure of available error. To find the optimum number of clusters, we can plot the WCSS as a function of the number of clusters. A sample elbow diagram is shown in Figure 3. We