University of California
Department of Civil Engineering

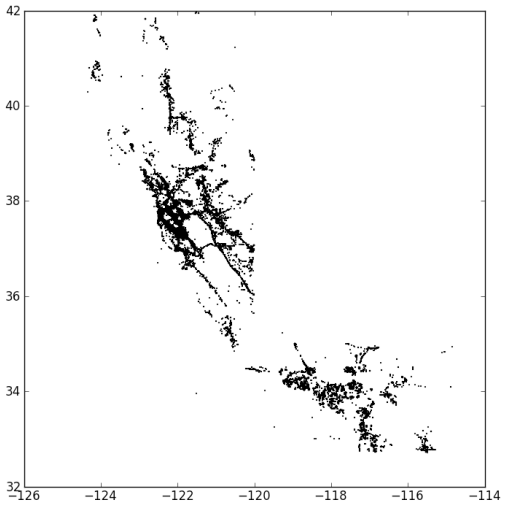
Pozdnukhov, Instructor

CE 263 Problem #1

Clustering

This problem guides through an example of exploratory data analysis with clustering methods. You will implement a data exploration system that applies a combination of clustering methods to explore frequently visited locations in California from geo-referenced social media data. You will study the properties and computational requirements of some common clustering methods on a medium-scale dataset and then design and implement a system for processing 1 million data samples.

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



Data description. This assignment deals with the dataset of 1M tweets collected using the Twitter API over a period of time. The dataset is located on bCourses as a zipped json file named "tweets_1M.zip". The pair of (lat, lng) coordinates is associated to each tweet corresponding to the location of the user (see Figure on the left). The spatial extent of the dataset is the state of California. Other information contained in each tweet is its unique id, the user_id, the timestamp and its textual content.

In Parts 1 and 2 of the assignment, you will use a subset of 100'000 samples contained in the tweets_1M.json data file.

You will work on clustering the data samples in geographical space (2 dimensions). Matplotlib/pyplot libraries provide useful tools to visualize the data and clustering results. You can find the examples of visualizations in scikit-learn online documentation. Be aware that visualization can take significantly more time than clustering itself. Use it wisely.

Your submission will consist of a single PDF document describing your results and a code you used in Part 3.

Part 1. Clustering: the baseline (20 points).

1. Write a script that applies the *k*-means clustering method to the 100K subset. Measure a typical processing time. Experimentally detect the maximum number of clusters *k* that can be handled with the implementation of the algorithm you are using.

2. Write a script that applies the MiniBatch k-means method to the 100K subset. Select an appropriate value of a batch size. Measure and note the gain in computational time. Evaluate the maximum number of clusters *k* that can be handled with the implementation of the algorithm you are using.
3. Write a script that applies the DBScan method to the 100K subset. 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.

Prepare a brief write-up of Part 1 containing the following:

- Reference time of clustering of 100K samples into k=100 clusters with k-means and mini-batch k-means.
- Maximum number of clusters **k_max** that your implementation of k-means and mini-batch k-means can handle. Explain the reasons behind this performance bottleneck.
- The value of ε (call it **ε_100**) in DBScan resulting in approximately 100 clusters of a minimum of samples (MinPts =100) and the corresponding processing time.

Part 2. Clustering: scalability (40 points).

2.1. For k-means and MiniBatch 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**)

2.2. For DBScan algorithm, plot the graphs representing computational time as a function of:

a) Number of samples (consider the range of 100 to 100'000) for a fixed ε_100, MinPts = 100.

Include the graphs in the write-up of your submission. For each of the 3 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 3. Clustering: 1 million samples problem (40 points)**.** This part deals with the full dataset of 1 million tweets. Your task is to design a system that can handle spatial clustering of 1M samples.

Considering the memory limitations and scaling properties of the algorithms studied in Part 2, design the clustering system that can be applied to the full dataset. Consider using a hierarchical approach with two (or more) processing stages, where DBScan is applied to each cluster obtained from a run of mini-batch k-means. By varying the parameters of the algorithms, optimize the processing time required to detect clusters of tweets that correspond to important locations in California. We will consider a location "important" if it is characterized with a cluster's core of at least 100 samples within a radius of 100 meters.

Describe your approach to the design of the system in a write-up document and provide the total number of clusters detected. Submit your code as a stand-alone script.

Extra credit (max 20 points)

Produce a visualization of a typical cluster of your choice, and describe its origins by inspecting its spatial location and/or content of the tweets forming the cluster.

Appendix.

Loading data:

```
import json

with open('tweets_1M.json','r') as f:
    tweets = json.load(f)
```

Consider transforming (lat, lng) into meters or miles with an approximate linear scaling. One degree of latitude is approx 89.7km and one degree of longtitude is 112.7 km in central CA.

Time measurement:

```
import time

t0 = time.time()
# your clustering code, for example
#
#db = DBSCAN(eps=0.05, min_samples=20).fit(X)
t_clustering = time.time() - t0
```

Clustering:

Example of using k-means and mini-batch k-means in scikit-learn:
http://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:
http://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#example-cluster-plot-dbscan-py