CS663 Spring, 2021: Assignment 01 - Clustering

The **goal** of this assignment is:

1. To demonstrate your understanding of clustering algorithms like K-Means, DBSCAN, Hierarchical and Spectral.
2. To develop your version K-Means using the algorithm specified below.
3. Extending the functionality of the developed K-means implementation through additional parameters.
4. Comparison of performance between K-Means on a given dataset.

Background

We covered the algorithms and hyperparameter tuning for k-means, DBSCAN, Hierarchical and Spectral clustering. The algorithm (in psuedocode) for the k-means algorithm is as follows:

```
place k centroids (𝜇1,𝜇2,…,𝜇k∈ ℝn) randomly
repeat to convergence:
  foreach x ∈ test_x:
    c(i) = index of closest centroid to x
  foreach k ∈ centroids:
    𝜇k = mean c(i) | index(c(i)) == k
```

DBSCAN works by starting in a random place in a dataset and moving outwards, adding items to the cluster. Hierarchical clustering puts each instance in its own cluster and associates pairs of clusters by distance. Spectral clustering imposes a graph over each instance and introduces graph cuts to isolate instances into clusters.

Requirements (Process)

There are four parts for this assignment, all detailed below:

1. Implement k-means as described above.
2. Extend k-means so that it balances the number of instances per cluster.
3. Run the clustering algorithms against some datasets and determine the performance of each; compare performances of the algorithms.
4. Do a performance analysis between your implementation of K-Means (excluding the extended version) and the version offered by Scikit-learn library. The dataset to be used is: Link to Dataset

Implement k-Means

Create a python-based implementation of the k-means algorithm.

This implementation must be a subclass of cluster.py, available here (Links to an external site.). As such, it must implement two member functions: __init__(...) and fit(...), as described below.

- __init__(...) must allow the class' users to set the algorithm's hyperparameters: k, which is the target number of cluster centroids, and max_iterations, which is maximum number of times to execute the convergence attempt (repeat loop in the above Background section). The default values are required to be k = 5 and max_iterations = 100.
- fit(...) must accept one parameter X, where X is a list (not columns of a Dataframe) of n instances in d dimensions (features) which describe the n instances. A successful call to the fit(...) function must return the following two items, in order:

    1. A list (of length n) of the cluster hypotheses, one for each instance.
    2. A list (of length at most k) containing lists (each of length d) of the cluster centroids' values.

For example, if the input (X) contains the following values in 2-dimensional space:

[[0, 0], [2, 2], [0, 2], [2, 0], [10, 10], [8, 8], [10, 8], [8, 10]]

… and k = 2, we expect the centroids should be [1, 1] and [9, 9]. The output of the fit(...) function should be as follows:

1. [0, 0, 0, 0, 1, 1, 1, 1] — indicating that the first four instances belong to one cluster and the second four belong to a different cluster.
2. [[1, 1], [9, 9]] — the values for the first and second centroid, respectively.

Test the python-based implementation using scikit-learn. Generate clusters using the make_blobs function with the following commands:

```
from sklearn.datasets.samples_generator import make_blobs
X, cluster_assignments = make_blobs(n_samples=200, centers=4, cluster_std=0.60, random_state=0)
```

This will generate 200 instances of data points in 2-dimensional space, with each of the instances belonging to one of 4 clusters. The coordinates for the 100 instances are returned as X. The cluster assignments are returned as cluster_assignments. Use X as the parameter to your fit(...) function listed above, and use cluster_assignments to determine whether your implementation's hypotheses are correct. (Given multiple — 10? — iterations of your implementation with k=4, the values for X from the commands above should generate no errors; however, the values in cluster_assignments may not align to the values from your implementation's hypotheses.)

Include your function's sample output from this test input as a .TXT file.

Extend k-Means

Extend your implementation of fit(...) to write a new function, fit_extended(...), with an additional optional Boolean (True/False) argument, balanced. The default value must be False. When balanced is set to True, the implementation changes so that each of the k clusters are (roughly) equal with respect to the number of instances per cluster — i.e. the implementation generates clusters of (roughly) the same size. When balanced is False, follow the same logic from fit(...).

As before, use makeblobs to test your input, and include your function's output as a .TXT file.

Your final code submission should have both versions of the k-means algorithm, with and without the additional Boolean argument. Include fit(...) and fit_extended(...) in the same .py file (kmeans.py) and in the same class (def KMeans).

Choose and run clustering algorithms

Execute one or more clustering algorithms (k-means, DBSCAN, Hierarchical, Spectral) against the datasets below as implemented in Scikit-learn. Explain the following:

1. The reason why you chose the clustering algorithm(s)
2. Any pre-processing of the data or any hyperparameter settings
3. Output from the algorithm(s) -- show what clusters were generated
4. The metrics you used to evaluate the output. What kind of performance did you get from that algorithm? Is that what you expected?

Use the following datasets for this part:

- Chicago taxi data (Links to an external site.), an approximately week-long subset of the full dataset (which can be found here (Links to an external site.)). Use either the pickup or dropoff location coordinates.
- Finnish location data (Links to an external site.) (taken from Mopsi data (Links to an external site.))

Submit the Jupyter notebooks in which you performed your analyses (one notebook per dataset).

Performance Comparison

1. Using Scikit-learn, run K-Means on the dataset provided. (Link shared above).
2. Using your own version of K-Means, fit the same dataset. (Link shared above).
3. Do a performance analysis between the two - drawing insights on the distribution(labelling) of points across the different clusters.

Using the dataset that has been fit using K-Means (both versions), load it into the Jupyter notebook to draw visualizations showing the clustering pattern. Make sure that the same number K value is used for both versions.

Grading

55% = Implementations, specifically:

- Canonical K-Means implementation (35%)
- Extended K-Means implementation (20%)

25% = Clustering algorithms for (Chicago Taxi Data and Finnish Location Data), specifically:

- Justifying clustering algorithm (10%)
- Fitting data and making predictions (15%)

15% = Performance analysis of implementations (your implementation vs. K-Means)

5% = Style and code quality

Submission

Submit the GitHub url on Canvas. The assignment will be due on Feb 22nd 11:59 pm 2021.