K-means clustering

Applied Machine Learning in Engineering - Exercise 03

TU Berlin, Winter Term 2023/24

Prof. Dr.-Ing. Merten Stender – `merten.stender@tu-berlin.de`

This exercise asks students to implement the basic K-means algorithm from scratch. Students can choose from two degrees of complexity:

- **Low complexity**: fill in gaps in a procedural implementation of K-means provided in ISIS. The following instructions are given for this task.
- **High complexity**: write an object-oriented implementation of K-means from scratch. You may find inspiration for the class methods and attributes in the procedural implementation.

The basic structure of the code is given (`my_kmeans.py`) and students need to fill in gaps. Data points are stored in the variable `x` (Numpy array of shape `(N, n)`, where N is the number of data points and n is the dimensionality of the data space. Cluster assigment labels are stored in `labels` (Numpy array of shape `(N,)`, index starting at 0). Centroid positions are stored in `centroids` (Numpy array of shape `(K, n)`. K denotes the number of clusters.

For a visual test, you can use the function `plot_clusters` from `utils_clustering.py`. Plot data points and clusters (no labels) using `plot_clusters(x=x, centroids=centroids)` and plot data points with cluster assignment using `plot_clusters(x=x, labels=labels, centroids=centroids)`. You can import functions from a different file using `from <otherFile> import <function>`.

Problem 1

(a) Implement the centroid updates in the function `update_centroids()` for the L_1 and L_2 norm. Consider the comments given in the code.

(b) Test your implementation on a minimal set of points for which you can compute the centroid position by hand. Plot the result.

Problem 2

(a) Implement the cluster assignment in the function `assign_cluster()` for the L_1 and L_2 norm. Consider the comments given in the code.

(b) Pick some points and two centroids. Test your implementation for correctness by inspecting the labels as returned by your function, and plot the results.

Problem 3

Study the function `is_converged()` and annotate each line by adding a comment that describes the action taking place. Discuss with your neighbor.

Problem 4

Now you have all three fundamental ingredients ready for implementing your own K-means algorithm in `kmeans_clustering(x:np.ndarray, K:int, norm:str='L2', init_centroids:np.ndarray=None)`.

(a) Implement the conditions that need to be `True` in the main `while` loop.

(b) Implement the cluster assignment, the centroid update, and the convergence check using the functions from Problems 1 to 3.

(c) Check the functionality of your algorithm using the sample data points given in the main function at the end of the file.

Problem 5

Validate your own implementation against the one in scikit-learn. Use the data set `example_data_Kmeans.csv`. Do you obtain the same results as scikit-learn?

Problem 6 (extra)

Re-visit `update_centroids()` and implement a centroid relocation if an empty cluster is encountered. Call the `relocate_empty_centroid()` function if the cluster is empty.

Problem 7 (extra)

Compute the cluster validity metrics SSE and BSS during each iteration of K-means and return a list of cost values as an additional return value of `kmeans_clustering()`. Use the functions `sse()` and `bss()` from `utils_clustering.py`. Plot the evolution of both metrics along the iterations of K-means for the sample data set.