Lab Assignment-3

Q.1 Implement `k_means (X, k, max_iter=100, tol=1e-4, random_state=None)` from scratch in python (do NOT use scikit-learn).

Algorithm steps:

1. Randomly initialise `k` centroids.
2. Assign each sample to the nearest centroid (squared-Euclidean distance).
3. Update centroids as the mean of their assigned points.
4. Stop when the maximum centroid shift < `tol` or when `max_iter` is reached.
5. Return the final centroids, the cluster labels, and the number of iterations used.
6. Test on a 2-D “blobs” data set (`sklearn.datasets.make_blobs`, 3 clusters, 300 points) and plot the labelled result.

Q.2 Load any small RGB image (Pillow). Reshape it to an `(N, 3)` pixel matrix.

1. Run `KMeans(n_clusters=16, init='k-means++', random_state=42)` from scikit-learn.
2. Replace each pixel by its cluster centroid to create a 16-colour image.
3. Display the original and compressed images side-by-side.
4. Save both images as PNG and report the percentage reduction in file size.

Q.3 Load the Iris data set (`sklearn.datasets.load_iris`).

For `k = 2 … 10`, fit scikit-learn K-Means with a fixed `random_state` and record:

1. inertia (`model.inertia_`)
2. average silhouette score (`sklearn.metrics.silhouette_score`).
3. Plot both metrics versus `k` (two lines on one figure).

Choose:

• `k_elbow`: the smallest `k` where the inertia drop is < 10 % of the previous drop.

• `k_silhouette`: the `k` that maximises the silhouette score.

Print the two suggested `k` values and state whether they agree.

Q.4 Generate 1 000 000 samples with 10 clusters in 10-D using `sklearn.datasets.make_blobs`.

1. Fit Full `KMeans(n_clusters=10)` and `MiniBatchKMeans(n_clusters=10, batch_size=10_000)`; Time both fits with `time.perf_counter()`.
2. Compute and compare: fit time, final inertia, and Adjusted Rand Index (ARI) against the true labels.
3. Wrap everything in a function `benchmark_kmeans()` that returns a dictionary and prints a small table.

Q.5 Implement K-Means using cosine distance (`1 − cosine_similarity`).

Steps:

1. L2-normalise all input vectors once at the start.
2. During each centroid update, re-normalise centroids to unit length.
3. If a cluster becomes empty, re-initialise its centroid to the sample farthest from any current centroid (max-min heuristic).
4. Test the algorithm on TF-IDF vectors from `sklearn.datasets.fetch_20newsgroups` (subset of the four "sci." categories, up to 5 000 documents).
5. Report the final cosine-distance inertia and the number of empty-cluster re-initialisations.