DA LAB 6 ONE-PAGER

Get_Clusters Function:

The "get_clusters" function was straightforward. The initialization gets the number of data points and selects k data points as initial centroids at random. These first centroids are picked without replacements, meaning no duplicates.

In the main loop, "for _ in range(100)" runs for a max of 100 iterations and can stop earlier if convergence is reached. The assignment step (distances = np.sqrt(((data[:, np.newaxis] - centroids) ** 2).sum(axis=2)) closest_centroids = np.argmin(distances, axis=1) calculates the Euclidean distance between points and all centroids and assigns each point to its nearest centroid.

During its update (for k in range(n_clusters): if np.sum(closest_centroids == k) > 0: # Avoid empty clusters centroids[k] = np.mean(data[closest_centroids == k], axis=0) each centroid is updates to the mean of all the respective points assigned to it, and includes a check to avoid empty clusters.

if np.all(np.abs(centroids - old_centroids) < 1e-4): break is simply a convergence check – stops if centroids move a minimal amount between iterations. After this, a "simple return centroids, closest_centroids" returns the final centroid locations along with the cluster assignment for each data point.

Performance of Four Clusters:

Firstly, the performance score we opted for was a silhouette score – a standard metric that adequately measures how well points of grouped into clusters. It is based on the average distance between data points within and between clusters.

When using four clusters, the result was below average, registering at a 0.417 silhouette score. A sizable number of the points are not particularly close to their centroid and are fairly scattered around. Based on our middling findings, we decided to experiment with the number of clusters used until we could maximize the silhouette score.