SENG 474: Data Mining
Assignment 3

Clustering Experiments: *Lloyd's Algorithm and Hierarchical Agglomerative Clustering*

Sean McAuliffe, V00913346 (A02)

March 23rd, 2023 AD

Contents

1 Introduction

This report summarises the results obtained from applying two clustering algorithms, Lloyd's Algorithm and Agglomerative Clustering to two provided datasets. The first dataset contains 3500 points in 2 dimensions, and the second dataset contains 14801 points in 3 dimensions. The purpose of this exercise is to gain familiarity with implementing clustering algorithms, and to gain an understanding of the differences between approaches; in order to be able to interpret their results, choose an appropriate number of clusters for a given problem and apply appropriate solutions for different situations.

Clustering is an unsupervised learning technique with the goals of categorising data into groups based on their similarity. In this way, labels can be learned from the data itself; without the need for pre-labelled data. Clustering is a useful technique in data mining which can be used to find patterns in datasets, perhaps as a pre-processing step for further supervised learning methods. The notion of a cluster is not easy to formalise (one recognises a cluster when one sees it). Accordingly, many algorithms exist for producing clusterings from data.

2 Implementation

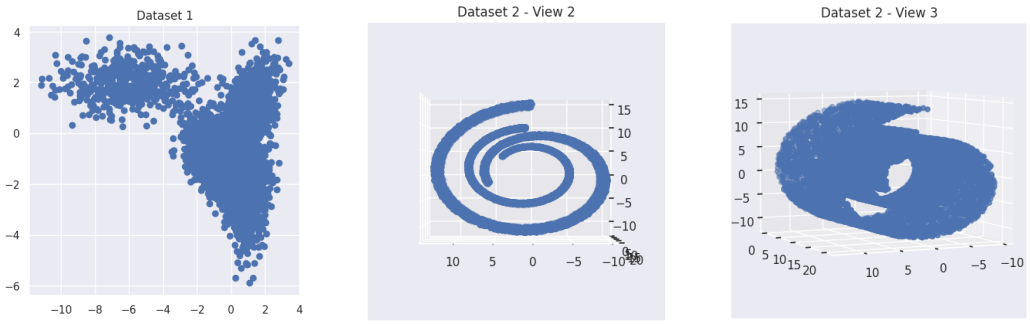
The experiments conducted for this report were composed in an accompanying Jupyter notebook running on a *Python 3.10.6* Kernel. An implementation of Lloyd's algorithm was written for this assignment, according to the description provided in the lecture 23 slides. An implementation of agglomerative clustering was imported from the *sci-kit learn* library. The *sklearn.cluster AgglomerativeClustering* class was used, which provides the necessary connectivity metrics for determining the distance between clusters.

All code for implementation of the algorithms, use of external libraries, data loading, and graph generation is contained in the accompanying Jupyter notebook.

2.1 Provided Datasets

In order to build an intuition for the behaviour of the algorithms in later experiments, the provided data were plotted in 2 and 3 dimensions respectively.

This is shown in Figure 1.



(a) Dataset 1, 2D (b) Dataset 2, 3D (c) Dataset 2, 3D

Figure 1: The shape of the provided datasets in 2 and 3 dimension

The first dataset seems like it ought to be separable into roughly 2 clusters. The clusters seem to be globular, but clearly oblong. The second dataset seems to also be separable into 2 clusters, however it is highly non-globular. Rather, it is composed of two intertwined spirals having the appearance of a dessert.

3 Lloyd's Algorithm

Lloyd's algorithm is a k-means clustering algorithm in which clusters are defined by the mean of the points in the cluster. Lloyd's algorithm is an iterative algorithm which converges on a clustering of minimum cost by repeatedly assigning points to their nearest clusters, and then recomputing the cluster means, until the assignments of points to clusters no longer changes. Through this process, the means which describe a clustering are learned.

The algorithm is as follows:

1. Initialise the means of the clusters (using one of the below methods)
2. Assign each point to the cluster with the nearest mean.
3. Recompute the means of the clusters.
4. Repeat steps 2 and 3 until the assignments of points to clusters no longer changes.

The goal is to learn a minimum-cost clustering of the dataset into k clusters. The cost of a k-clustering, C is defined as:

$$W(C) = \sum_{j=1}^{k} \sum_{x \in C_j} \|x - \mu_j\|^2 \tag{1}$$

This assignment required Lloyd's algorithm to be implemented, the implementation is shown in the code snippet below.

```
def lloyd(X, k, init):
    # Initialize the clusters
    C = init(X, k)
    # Repeat until the centres do not change
    previous_centres = get_centres(C)
    i = 0
    while True:
        i += 1
        # Assign each data point to the cluster with the closest centre
        for cluster in C:
            cluster.clear_data()
        assign_to_closest(X, C)
        # Update the cluster centres
        for cluster in C:
            ideal_centre = np.mean(cluster.data, axis=0)
            # Find the point in the data closest to the idealized centre
            min_dist = np.inf
            min_point = None
            for point in cluster.data:
              dist = euclidean_distance(point, ideal_centre)
                if dist < min_dist:
                    min_dist = dist
                    min_point = point
            cluster.centre = deepcopy(min_point)
        # Check if the centres have changed
        new_centres = get_centres(C)
        if all_centres_equal(previous_centres, new_centres):
            break
        previous_centres = new_centres
    return i, C
```

3.1 Initialisation

The algorithm is probabilistic in that the initialisation of the means is probabilistic, and so the algorithm may converge to a different clustering each time it is run, and each run may require a variable number of iterations to converge. To find a low cost clustering it may be necessary to run the algorithm multiple times, and to choose the clustering with the lowest cost.

Uniform random initialisation is a method of initialising the means of the clusters by randomly sampling, without replacement, points from the dataset with equal probability. This method is simple to implement, however it can lead to poor results and poor runtimes if the initial means are poorly chosen.

K-Means++ Initialisation proceeds from the notion that cluster centres ought not to be near to each other, as such points are likely to be assigned to the same cluster. K-Means++ initialisation can reduce runtime by preferring to sample points which are far from the current cluster centres. The first cluster centre is chosen uniformly at random from the dataset, subsequent centres are chosen with probability proportional to the squared distance from the nearest cluster centre. This method is more likely to produce good results than uniform random initialisation, resulting in lower expected runtimes.

This assignment involved implementing k-means++ initialisation, a snippet showing part of the implementation is below.

```
for x in X:
  min_dist = np.inf
  for cluster in clusters:
    dist = cluster.distance(x)
    if dist < min_dist:
      min_dist = dist
      dists.append(min_dist)
  probability_vector = dists/np.sum(dists)
  centre = X[np.random.choice(len(X), p=probability_vector)]
```

3.2 The Elbow Method

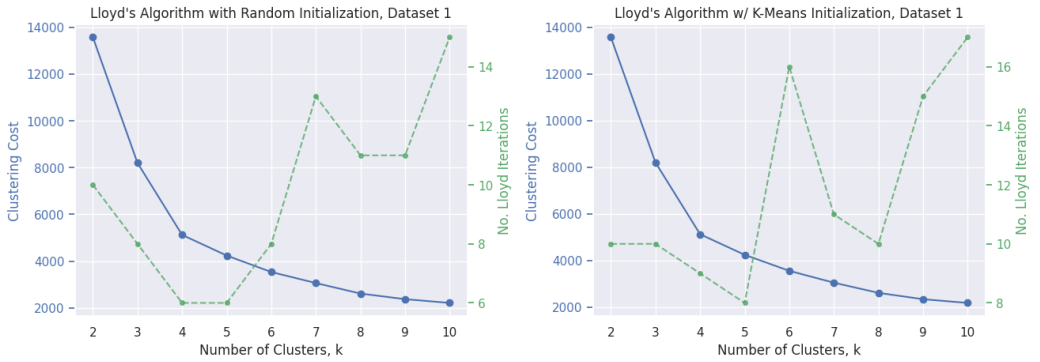
Determining how many clusters are present in a dataset is a difficult problem to formalise. Instead, for this assignment the elbow method was used to determine the proper number of clusters to use. The elbow method is a heuristic method in which the number of clusters, k is incremented, and

a clustering algorithm is run to produce a clustering for each k. The cost of each k-clustering is measured (for each k the algorithm is run multiple times and the lowest cost clustering is chosen as the result). The cost of each k-clustering is plotted as a function of k. The resulting plot should contain a steadily decreasing curve, ideally with a sharp bend at the point of diminishing returns. The point at which the curve bends is the optimal number of clusters to use.

3.3 Experimental Results

3.3.1 Lloyd's Algorithm on Dataset 1

The number of clusters was varied from [2, 10] and for each value of k the algorithm was run 10 times, and the lowest cost clustering was chosen as the result. For each of the chosen clusterings, its cost and the number of iterations required to converge were recorded. The results are shown in 2



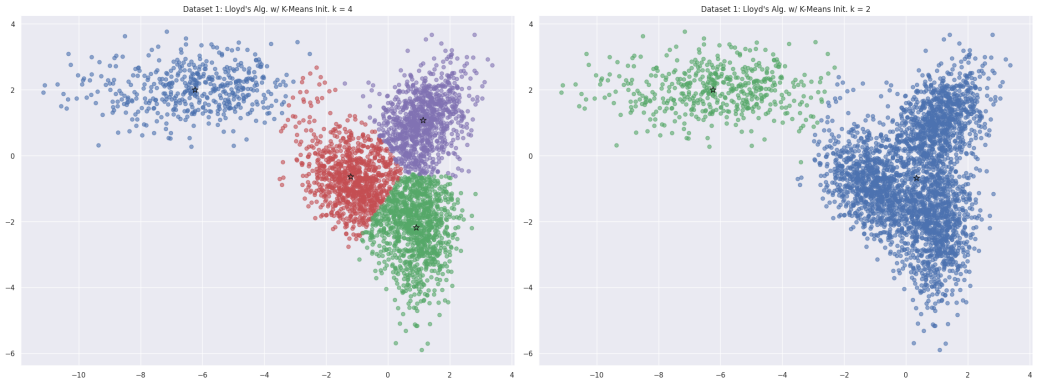
(a) Random Initialisation (b) K-Means++ Initialisation

Figure 2: The elbow method graph for dataset 1

One thing to notice is that the results are very similar. Indeed, the cost of the clusterings chosen for each value of k are almost identical. In both plots, there appears to be an elbow that appears sharpest at k = 4. This choice is somewhat arbitrary, and sensitive to such things as the presented scale of the axes.

Another thing to notice is that the number of iterations required to converge increases sharply after the elbow-point. And, that k-means++ initialisation

did not appear to significantly reduce the number of iterations required to converge. This is likely due to the small size of the dataset, with so few points and clusters, the initialisation is very sensitive to the choice of initial means.



(a) k = 4 (b) k = 2

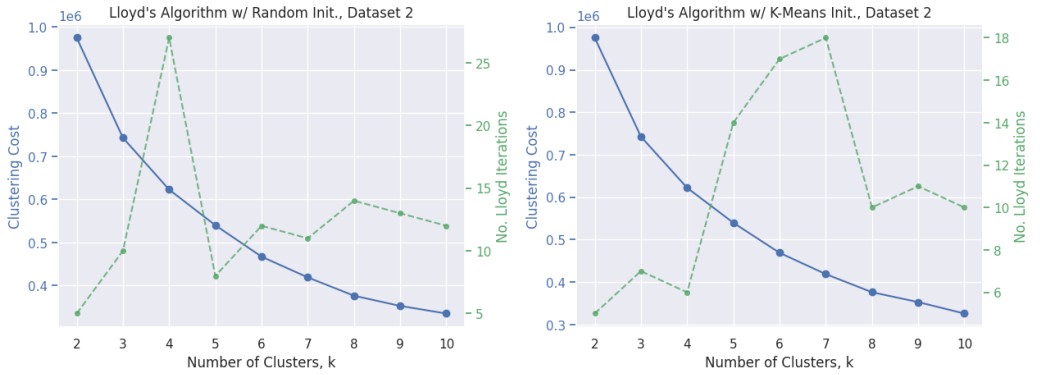
Figure 3: Clustering produced by Lloyd's Algorithm on dataset 1

Shown above in 3 is the clustering produced by Lloyd's Algorithm with the choice of $k = 4$. The clustering shown above was produces with k-means++ initialisation, the clustering produced with random initialisation is much the same. In contrast to later methods, this clustering produces sharp boundaries between adjacent clusters in the rightmost region (which seems like it ought to be a single cluster).

To satisfy the bias that the 'proper' clustering should be k=2, the result of Lloyd's algorithm with k=2 is also shown. The Elbow method, as here performed, may have a flaw in the case that the optimal result has two clusters, since the plot starts at k = 2, it will not appear to be the bend in the elbow.

3.3.2 Lloyd's Algorithm on Dataset 2

The number of clusters was varied from [2, 10] and for each value of k the algorithm was run 10 times, and the lowest cost clustering was chosen as the result. For each of the chosen clusterings, its cost and the number of iterations required to converge were recorded. The results are shown in 4



(a) Random Initialisation (b) K-Means++ Initialisation

Figure 4: The elbow method graph for dataset 2

The clustering costs produced by Lloyd's algorithm with the two initialisation methods are again very similar at each value of k, in fact, many are identical. Dataset 2 is highly non-globular, and we should expect Lloyd's algorithm not to be able to 'correctly' produce a clustering for it. Indeed, both plots here seem to admit of no obvious elbow, in contrast to the previous section.

The cost curve is smoothly decreasing in both plots, with the rate of decrease itself smoothly decreasing. Knowing that the data should be separable into 2 clusters, one could argue that the elbow is at k = 2 or k = 3. However, the elbow method is not particularly useful here.

In this case, k-means++ initialisation appears to perform slightly better, reducing the number of iterations required to converge at higher values of k (not the different scales of the two green y-axes), although it's not true at every point. The cost-reduction promised by k-means++ initialisation is not present in the results here, likely drowned out by the fact that each iteration was run 10 times to find the lowest cost clustering among all the runs.

The results are shown in Figure 5 Clearly, neither of the two choices for k produces the clustering we would intuitively expect to be correct for this data. This is expected, as Lloyd's algorithm is not designed to produce a clustering for non-globular data.

Figure 5: Clusterings produced by Lloyd's Algorithm on dataset 2

4 Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering (HAC) is a bottom-up connectivity based algorithm in which clusters are defined by their member points (depending on the connectivity, or linkage, metric used). The algorithm begins by assigning each point to its own cluster. Then, at each iteration, the two clusters with the smallest distance between them are merged together. This process is repeated until the desired number of clusters is reached. HAC can accomodate many different distance metrics. For this assignment, the Euclidean distance was used. Another parameter of HAC is the connectivity metric.

4.1 Connectivity Metrics

Different connectivity metrics can be used to determine the definition of the distance between two clusters. This metric is used to make decisions about which clusters to merge next.

Single Linkage is the most straightforward metric. It measured the distance between two adjacent clusters as being to (Euclidean in our case) distance between the two nearest points in each cluster. This method is better at

handing non-globular data.

$$d_{SL}(C_i, C_j) = \min_{x_i \in C_i, x_j \in C_j} d(x_i, x_j) \tag{2}$$

Average Linkage is a more computationally intensive metric. It defines the distance between clusters as the average (Euclidean) pairwise distance between all of the points in each cluster.

$$d_{AL}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x_i \in C_i} \sum_{x_j \in C_j} d(x_i, x_j) \tag{3}$$

4.2 Dendrograms

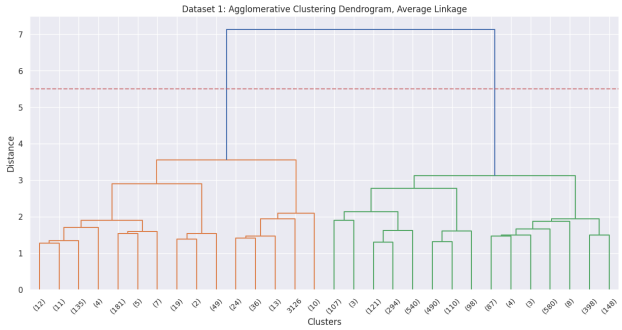
Dendrograms are a useful tool for visualising the results of each iterative merging decision made by HAC. A dendrogram is a tree-like diagram that shows the order in which the merges happen. It is organized according to the distance between two clusters at the point that they are selected as to be the next merge. If the tree is presented in the usual top-down way, then the leafs of the tree represent the original data points, and any horizontal cut through the diagram will produce a separation of the data into k clusters, where k is the number of vertical lines intersecting with the cut. This can be used as a heuristic visual aid, much like the Elbow method, to help determine what the optimal number of clusters is.

4.3 Experimental Results

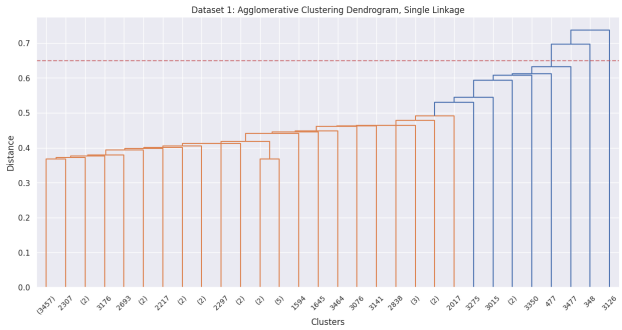
4.3.1 HAC on Dataset 1

Since dataset 1 is globular, we should expect HAC with average linkage to produce a good clustering. It is a question whether or not HAC with single linkage will work as well. The results of of using HAC to generate a dendrogram for dataset 1 are shown below in 6, using both connectivity metrics.

The results show that HAC with average linkage produces a dendrogram that contains a very clear optimal clustering a k = 2. This is expected given a manual inspection of the data. One can tell that two clusters is optimal from



(a) HAC, dataset 1, average linkage, cut at k = 2

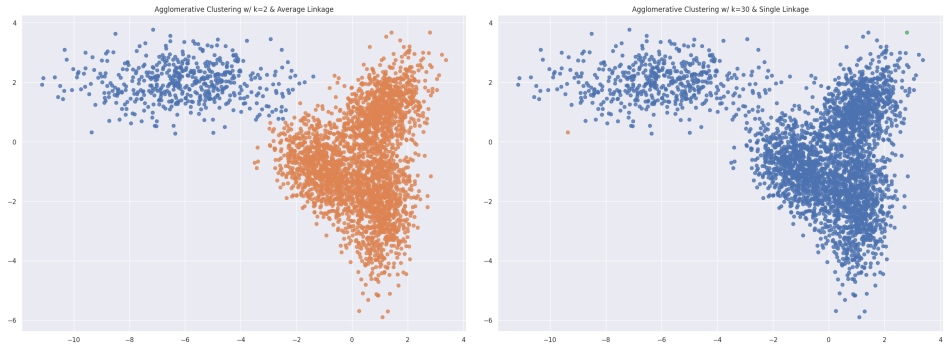


(b) HAC, dataset 1, single linkage, cut at k = 3

Figure 6: Dendrogram on dataset 1 using average and single linkage HAC

the dendrogram because the vertical distance travelled before another merge is made is very large, relative to the merges depicted lower on the tree. This is a good indication that the clusters are well separated.

HAC with single linkage produces a dendrogram that is not as clear. The optimal clustering is not obvious, however the largest vertical distance travelled between merges (ignoring the trivial cut which separates each point into its own cluster) is made at k = 3. Figure 7 shows the clustering produces by each method, this indicates that HAC with single linkage is not well suited to this data. It seems to choose two of the outlier points to be their own trivial clusters, and groups all remaining points together.



(a) Average Linkage (b) Single Linakge

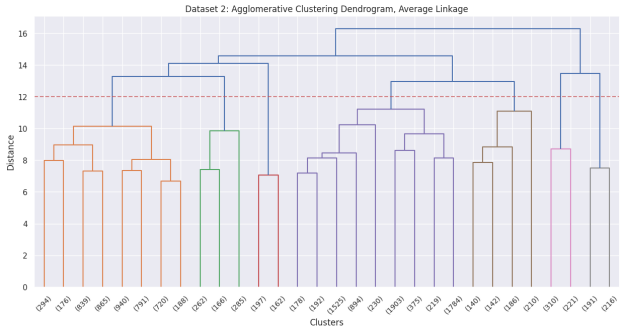
Figure 7: Clusterings produced by HAC on dataset 1

4.3.2 HAC on Dataset 2

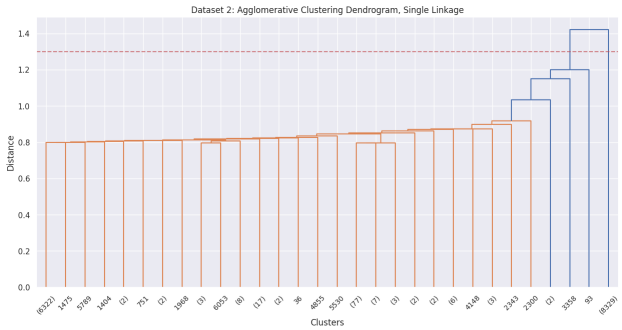
Dataset 2 is not globular, rather it is comprised of two interlocking spirals. The intuitive clustering based on manual inspection contains 2 clusters. Therefore, we should want a connectivity method which clusters all points based on the membership of their nearest neighbouring point, and not an absolute distance to a clsuter centre. We should expect single linkage HAC to produce a desireable result in this case. Average linkage should perform worse. The results of using HAC to generate dendrograms for both linkage methods are shown below in 8.

HAC w/ Single Linkage produces a dendrogram which offers one obvious (non-trivial) cut at k = 2. This is the desired result, and is in line with the manual inspection of the data. Figure 9 shows the clustering produced by HAC with single linkage. The clustering picks up on the spiral nature of the data, and the two clusters are clearly visible.

HAC w/ Average Linkage produces a dendrogram which does not paint as clear a picture of the optimal cut. Apart from the trivial cut in the lower part of the tree, there does appear to be a cut at k = 7. The height between merges appears to be 2 units, which is as large a height as can be found in the tree. The result of HAC with average linkage and k = 7 is shown in Figure 10. The clustering produced by HAC with average linkage is not as good as the one produced by HAC with single linkage. The clusters are not



(a) HAC, dataset 2, average linkage, cut at k = 7



(b) HAC, dataset 2, single linkage, cut at k = 2

Figure 8: Dendrogram on dataset 1 using average and single linkage HAC

as well separated, and the clustering does not pick up on the spiral nature of the data.

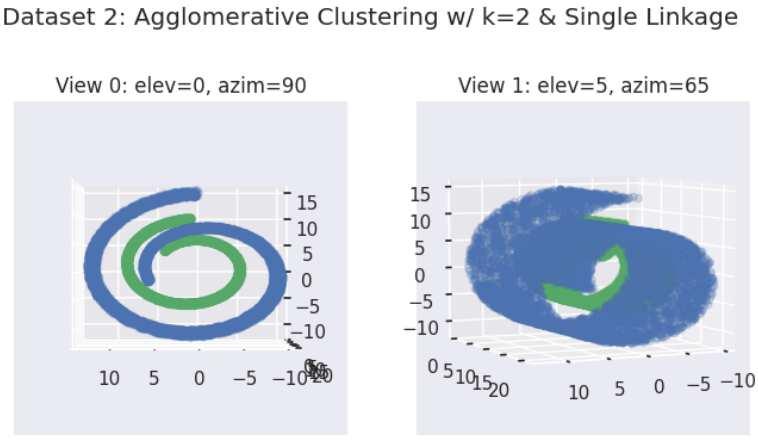


Figure 9: Clusterings produced by HAC on dataset 2

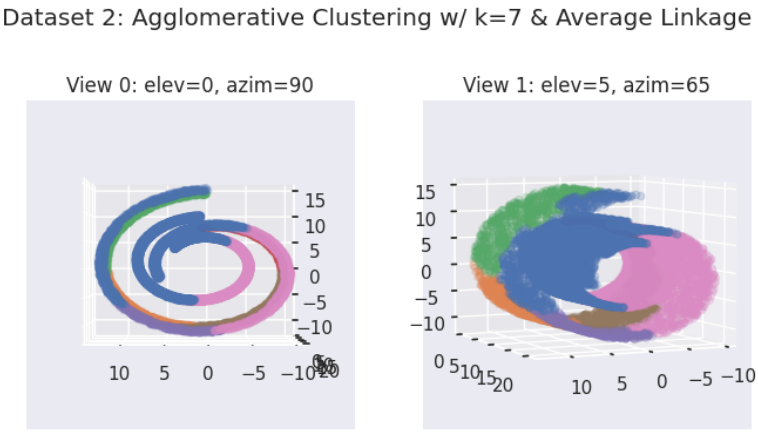


Figure 10: Clusterings produced by HAC on dataset 2