COMP 6721 Applied Artificial Intelligence (Summer 2023)

Assignment #5: Clustering

Question 1 Consider the following data set with two attributes for each of seven individuals.

	a1	a2
Data1	1.0	1.0
Data2	1.5	2.0
Data3	3.0	4.0
Data4	5.0	7.0
Data5	3.5	5.0
Data6	4.5	5.0
Data7	3.5	4.5

(a) The data set is to be grouped into two clusters. Initialize the clusters using Data1 and Data4 as initial centroids to the two clusters. That is, allocate the remaining individual to one of the two clusters using the Euclidean distance.

	Individuals	Centroid
Group 1	1	(1.0, 1.0)
Group 2	4	(5.0, 7.0)

(b) Recalculate the centroids based on the current partition, reassign the individuals based on the new centroids. Which individuals (if any) changed clusters as a result?

Question 2 scikit-learn also has an implementation for K-means.[1]

(a) Complete clustering the data from the previous question using scikit-learn's K-means implementation:

```
dataset = np.array([
[1.0, 1.0],
[1.5, 2.0],
[3.0, 4.0],
[5.0, 7.0],
[3.5, 5.0],
[4.5, 5.0],
[3.5, 4.5]])
```

Note that with K-means, you do not need any labels, since we are doing unsupervised learning:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
kmeans.fit(dataset)
```

To print the results, you can use:

```
print(kmeans.labels_)
```

This will show you for each element in the `dataset` which cluster it belongs to.

(b) Now cluster the *Iris* flower dataset: How well do the clusters represent the three species of *Iris?*

Using *Matplotlib*,[2] you can create a 3D-plot of your clusters:

```
# Plot the results using matplotlib
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 8))
plot = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
plot.set_xlabel('Petal width')
plot.set_ylabel('Sepal length')
plot.set_zlabel('Petal length')
plot.scatter(X[:, 3], X[:, 0], X[:, 2], c=kmeans.labels_, edgecolor='k')
```

The result should look similar to Figure 1.

[1] See https://scikit-learn.org/stable/modules/clustering.html#k-means

[2] See https://matplotlib.org/

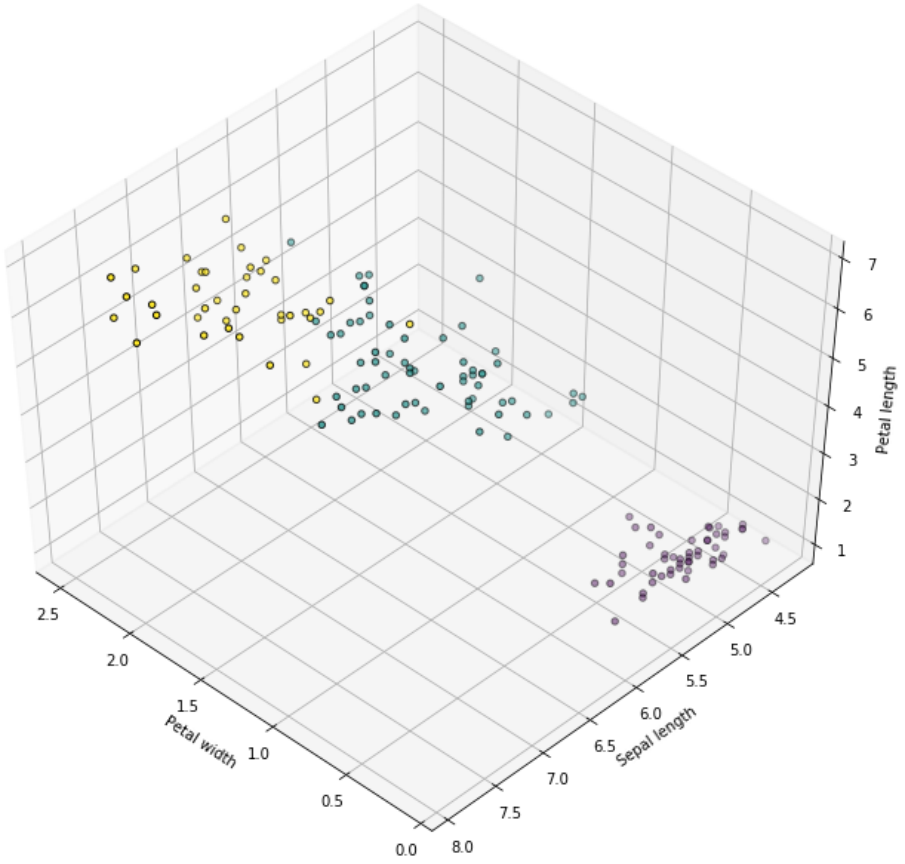

Figure 1: Plotting the results of clustering with K-means on the *Iris* dataset with Matplotlib

Question 3 Consider the same data set with two attributes provided in the first question. We want to cluster the observations using the hierarchical agglomerative method.

Here is the euclidean distance between each pair of observations.

	Data1	Data2	Data3	Data4	Data5	Data6	Data7
Data1	0.00	1.12	3.61	7.21	4.72	5.32	4.30
Data2	1.12	0.00	2.50	6.10	3.61	4.24	3.20
Data3	3.61	2.50	0.00	3.61	1.12	1.80	0.71
Data4	7.21	6.10	3.61	0.00	2.50	2.06	2.92
Data5	4.72	3.61	1.12	2.50	0.00	1.00	0.50
Data6	5.32	4.24	1.80	2.06	1.00	0.00	1.12
Data7	4.30	3.20	0.71	2.92	0.50	1.12	0.00

Figure 2: Distance between every pair of observations.

(a) How many clusters are formed in the first iteration? Which ones?

(b) How many clusters are formed in the second iteration using the single linkage method? Which ones?

(c) How will the clusters of the second iteration change if we use complete (farthest) and centroid linkage ?

(d) How many clusters are formed in the third iteration using the single linkage method? Which ones?

(e) How will the clusters of the second iteration change if we use complete (farthest) and centroid linkage ?

Question 4 Now, let's implement it in scikit-learn at it has an implementation for hierarchical clustering.[3].

```
dataset = np.array([
[1.0, 1.0],
[1.5, 2.0],
[3.0, 4.0],
[5.0, 7.0],
[3.5, 5.0],
[4.5, 5.0],
[3.5, 4.5]])
```

Let's fit the dataset:

```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(distance_threshold=0, n_clusters=None, linkage='singl
clustering = clustering.fit(dataset)
```

To see the dendroegram, run this code:

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)
```

[3]See https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

```
    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

We can now plot the dendroegram:

```
plot_dendrogram(clustering)
```

Note that since array in Python starts at zero, the first observation (Data1) will have index 0 and the last observation (Data7) index 6.