DAT565/DIT407 Assignment 4

Connell Hagen
connellh@student.chalmers.se

Måns Redin
mansre@student.chalmers.se

2024-09-30

1 Problem 1: Pre-processing the Dataset

The given dataset of different wheat kernels, was loaded into Python with pandas. Since the dataset had features with varying normal ranges, the variables were normalized using a Z-score method. This method uses the deviation from the mean of each set of variables to generate normalized values.

2 Problem 2: Determining the appropriate number of clusters

To determine the right amount of clusters in this dataset, not using the labels obtained from the dataset, the ”elbow-method” was used. Firstly, the inertia, or the sum of squared residual error, was calculated for each k, the number of clusters between one and seven. To do this a K-Means clustering algorithm had to be applied to the dataset. The K-Means algorithm initializes k different centroids and splits the data points into subsets, defined by which of these centroids is the closest one. Then the location of these centroid points is adjusted to the location where the sum of squared error is the least. After that, it re-classifies the data points according to which centroid, is the closest. This method is repeated until the algorithm converges to stable clusters. This was then plotted to find the ”elbow”-point, which indicates where the decrease of inertia is not drastically improved from one k to another. By visualizing the plot and ”eye-balling” the right amount of clusters, it came out to be three, which also happened to be the amount of labels in the dataset. Table 2 shows the inertia of different k-values and figure 1 shows the results of the inertia calculations.

3 Problem 3: Visualizing the classes

To reduce dimension of this problem, some different methods was used to easier visualize this multidimensional problem.

3.1 Scatter plot between each pair of features

The first method that was pursued was to plot each pair of features, and to decide which pairs a clustering process would be easy to do by observing the

k	Inertia
1	1610.000
2	767.365
3	451.986
4	401.153
5	366.219
6	325.865
7	291.217

Table 1: The Mean squared error and coefficient of determination between the predicted variable and the test dataset. Note that the Coefficient of Determination is 0.844 as before but it is another one rounded to the same value.

plot. In these plots, a structure could be found to some extent for a few, but not trivially linearly separable for any pair. The outcome of this method was that plotting feature 1 and feature 7 provided an easily clusterable plot. The plot of feature 1 and 7 is shown in figure 2.

3.2 Gaussian random projection

Secondly, the Gaussian random projection algorithm was carried out. This method reduces dimensionality, without eliminating some features and makes it easier to visualize. This algorithm uses randomness to project multiple features down to a two-dimensional set of points. The clusters in this picture was immensely hard to tell apart. Figure 3 conveys the plot of the dimension reduced data points.

3.3 Uniform Manifold Approximation

Another way to reduce dimension, is to use the Uniform Manifold Approximation algorithm. This algorithm projects the dataset onto a hypothesized Riemann Manifold. To proceed this algorithm the UMAP library in Python was used. Using this algorithm, the clusters were much more separated. A visualization of this dimension reduction can be visualized in figure 4.

3.4 Conclusions of visualization

As seen in the plots, the dataset does not look strictly linearly separable, especially not in the first two. The third plot makes the clusters much more separable, but a complete dissection still can not be obtained. Most of the data points would have been correctly classified. The main difficulty is to cluster the red and the blue data points into different clusters.

4 Problem 4: Evaluating clustering

The real number of clusters within our data was 3, so we did 3-means clustering on our data. After that, to measure the accuracy of 3-means, we first found the Rand index compared to the original data. The Rand index is the fraction of

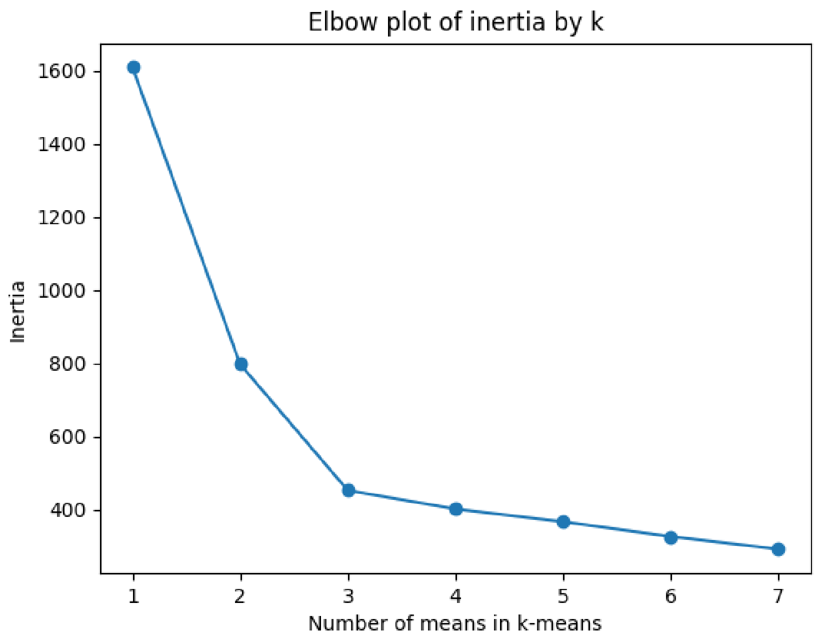

Figure 1: The number of means in the K-means in relationship to the inertia, to determine the "elbow"

points in the dataset that are in the same cluster in both clustering methods, or in different clusters in both clustering methods. We calculated a rand index of 0.956. We determined which clusters were equivalent between the original data and the 3-means method by determining the permutation of cluster labels that would have the greatest accuracy (we assumed that our model would be better than guessing randomly in order to assume that this would work).

Next, we calculated the accuracy of the 3-means clustering, which is the percentage of data points that were assigned to the correct cluster. We did this using the same assumed labels as before, and found an accuracy of 0.967.

5 Problem 5: Agglomerative clustering

We calculated hierarchical clusters using euclidean distance with the linkage options of: ward, single, complete, average, weighted and centroid. All options except for complete and weighted gave the incorrect number of clusters. Between complete and weighted, we found that complete had a greater accuracy when comparing the correct labels that were grouped together in each cluster. Figure 5 shows a dendrogram of this hierarchical clustering, and the correct labels for each leaf.

The complete linkage, defined by the furthest distance between two data points within the dataset gave the best result in hierarchical clusters. It measures and tries to minimize of the size of the cluster, making it a good way to

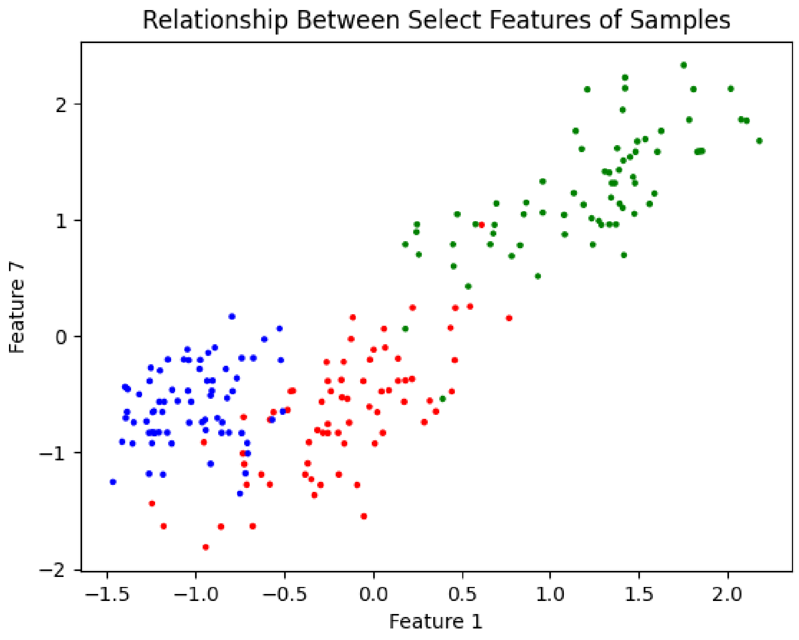

Figure 2: The relationship between the selected features with the strongest clusters

maximize the separation between clusters and comes up with clusters based on how compact a subset of data points is. Single linkage was tied for the worst method in terms of the number of clusters that it created. This is probably because single linkage merges clusters that are the closest to each other. Since there were 2 groups that were usually very closely intertwined in our dimension reduction plots, it seems likely that these groups would get merged together during the clustering process and not be separated very well.

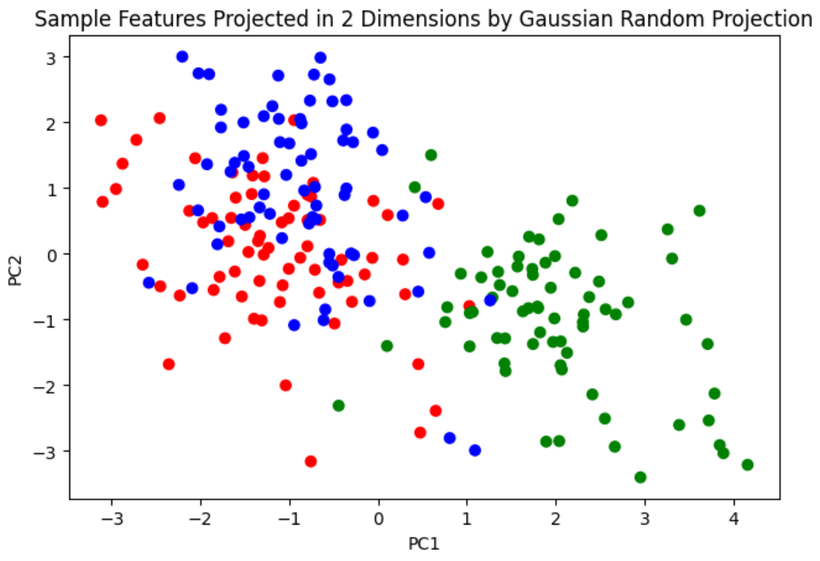

Figure 3: A scatter plot of a dimension reduced version of the dataset using the Gaussian Random Projection

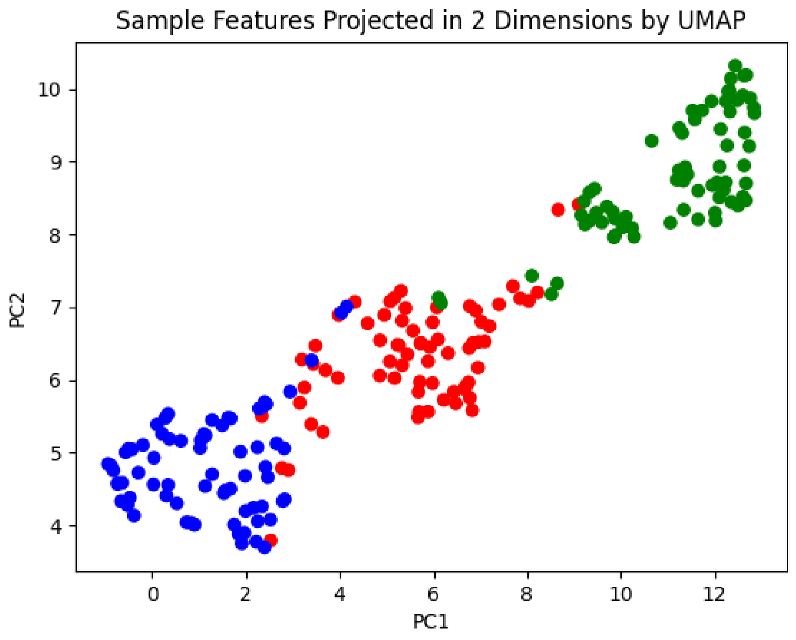

Figure 4: A scatter plot of a dimension reduced version of the dataset using the Uniform Manifold Approximation

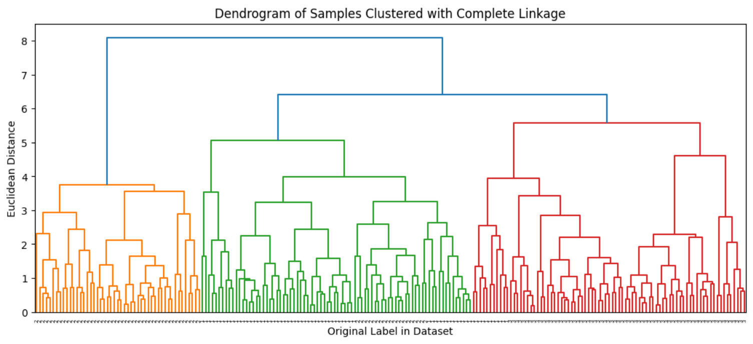

Figure 5: A scatter plot of a dimension reduced version of the dataset using the Uniform Manifold Approximation

A Code

```python

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.random_projection import GaussianRandomProjection
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
import umap

SEED = 98532

df_read = pd.read_table("assignment_5/seeds.tsv", header=None)

# normalize to z-scores
scalar = StandardScaler()
df = pd.DataFrame(scalar.fit_transform(df_read.iloc[:, :7]))

# add labels column back in
df = df.assign(**{"label": df_read.iloc[:, 7]})
df.columns = df.columns.astype(str)


inertia = []
for i in range(1,8):
    kmeans = KMeans(n_clusters=i, random_state=SEED).fit(df)
    inertia += [kmeans.inertia_]

# elbow plot
plt.scatter(range(1, len(inertia) + 1), inertia)
plt.plot(range(1, len(inertia) + 1), inertia)
plt.xlabel("Number␣of␣Means␣in␣k-means")
plt.ylabel("Inertia")
plt.title("\"Elbow␣Plot\"␣of␣Inertia␣by␣k")
plt.show()

color_map = {
    1: "red",
    2: "green",
    3: "blue"
}

```

```python
for i in range(1, 7 + 1):
    for j in range(1, 7 + 1):
        if (j <= i):
            continue
        plt.subplot(7, 7, (i - 1) * 7 + j)
        plt.scatter(df.iloc[:, i - 1], df.iloc[:, j -
            1], c=df["label"].map(color_map), s=1)
plt.show()

# i = 1, j = 7 was the best
plt.scatter(df.iloc[:, 0], df.iloc[:, 6], c=df["label"
    ].map(color_map), s=5)
plt.xlabel("Feature␣1")
plt.ylabel("Feature␣7")
plt.title("Relationship␣Between␣Select␣Features␣of␣
    Samples")
plt.show()

transformer = GaussianRandomProjection(n_components=2,
     random_state=SEED)
df_random = transformer.fit_transform(df.iloc[:, :7])

# the green is pretty clustered, but the red and blue
    are mixing significantly
# when not seeding, there was an occasional good one,
    but most were pretty mixed up
plt.scatter(df_random[:, 0], df_random[:, 1], c=df["
    label"].map(color_map))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Sample␣Features␣Projected␣in␣2␣Dimensions␣
    by␣Gaussian␣Random␣Projection")
plt.show()

reducer = umap.UMAP(random_state=SEED)
df_umap = reducer.fit_transform(df.iloc[:, :7])
plt.scatter(df_umap[:, 0], df_umap[:, 1], c=df["label"
    ].map(color_map))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Sample␣Features␣Projected␣in␣2␣Dimensions␣
    by␣UMAP")
plt.show()

kmeans = KMeans(n_clusters=3, random_state=SEED).fit(
    df)
k_labels = kmeans.labels_

same_cluster = np.zeros((df.shape[0], df.shape[0]))

```

```python
for i in range(0, same_cluster.shape[0]):
    for j in range(0, same_cluster.shape[1]):
        if (i >= j):
            continue

        if (k_labels[i] == k_labels[j]):
            same_cluster[i][j] += 1
        if (df.iloc[i, :]["label"] == df.iloc[j, :]["label"]):
            same_cluster[i][j] += 1

same = 0
for i in range(0, same_cluster.shape[0]):
    for j in range(0, same_cluster.shape[1]):
        if (i >= j):
            continue

        if (same_cluster[i][j] != 1):
            same += 1

# finding the correct clustering by taking the clusters that are most commonly correct -- assumes that the model doesnt completely suck
correct_cluster = np.zeros((3, 3))
for i in range(0, same_cluster.shape[0]):
    correct_cluster[k_labels[i]][(df.iloc[i, :]["label"] - 1).astype(np.int32)] += 1
num_correct_cluster = 0
for i in range(0, correct_cluster.shape[0]):
    largest = correct_cluster[i][0]
    for j in range(1, correct_cluster[i].shape[0]):
        if correct_cluster[i][j] > largest:
            largest = correct_cluster[i][j]
    num_correct_cluster += largest

num_items = same_cluster.shape[0]

rand_index = same / ((pow(num_items, 2) - num_items) / 2)
accuracy = num_correct_cluster / num_items


labels_arr = np.array(df["label"])
linkage_matrix = linkage(df, method="complete", metric="euclidean")
dendrogram(linkage_matrix, labels=labels_arr)
plt.title("Dendrogram␣of␣Samples␣Clustered␣with␣Complete␣Linkage")
plt.xlabel("Original␣Label␣in␣Dataset")
plt.ylabel("Euclidean␣Distance")
plt.show()
```