DAT565/DIT407 Assignment 5

Henning Anevik Joar Forsberg

2024-10-09

Problem 1: Preprocessing the dataset

We tried both standard and min-max normalization. We saw min-max normalization made the clusters a bit more apparent when plotting pairs of features in second task. And so, in the end, we opted for min-max scaling B. This was easily done by calling on functions from the sklearn library.

Problem 2: Determining the appropriate number of clusters

From the inspecting the graph we see that three is an appropriate number of clusters, which can be seen in Figure 1. This is known as the "elbow method", which we were taught about in the lectures C.

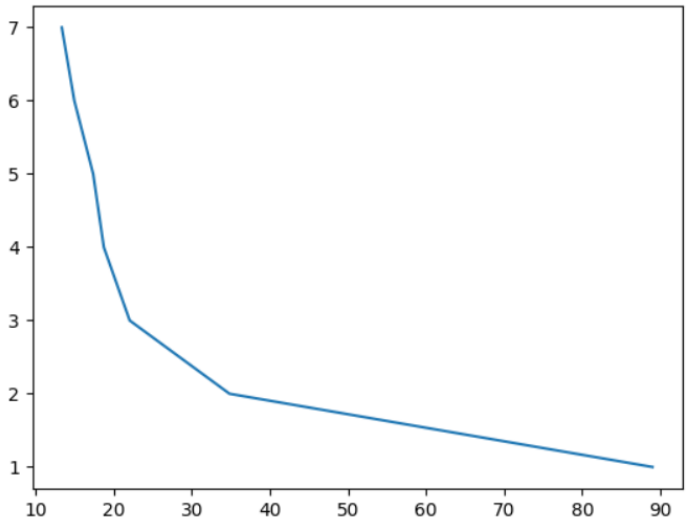

Figure 1: Elbow method for appropriate number of clusters

Problem 3: Visualizing the classes

a)

We see many plots of pairs of features result in straight lines. This makes a lot of sense since many features are highly correlated. For example the plot of kernel length vs. kernel groove length is almost a straight line, which makes a lot of sense seen in Figure 2.

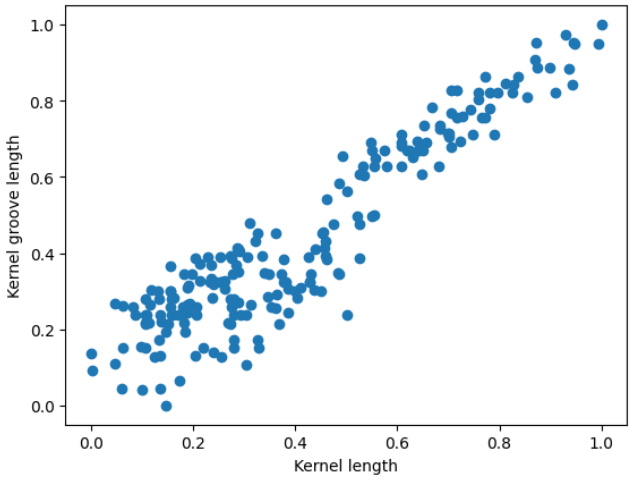

Figure 2: kernel length vs. kernel groove length

More interesting is are the graphs in which we can make out some semblance of clustering - where we are able to, somewhat, make out the different classes. The graph which displays the highest degree of clustering is the one of compactness and kernel groove length seen in Figure 3. We can see a pretty clear delineation between two of the classes and a less distinct one of the third class D.

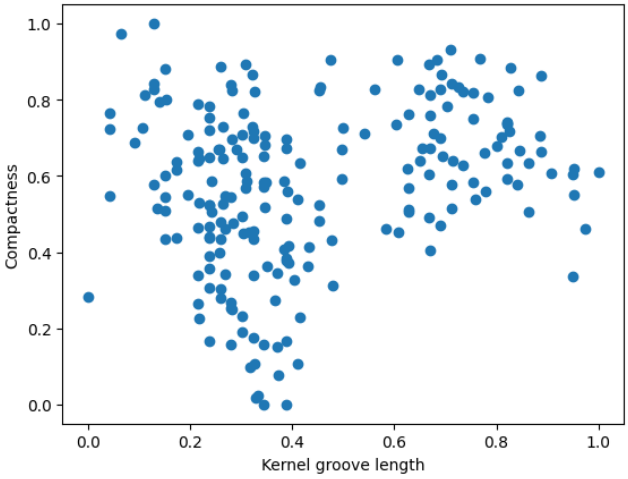

Figure 3: Compactness vs Kernel groove lenght

b)

The data seem to have a linear relationship in the two-dimensional plot. The Figure 4 suggests that the data might contain patterns in the random projection. But since Gaussian Random Projection involves randomness, repeating this can result in slightly different results due to the randomness, which was stated in the assignment E.

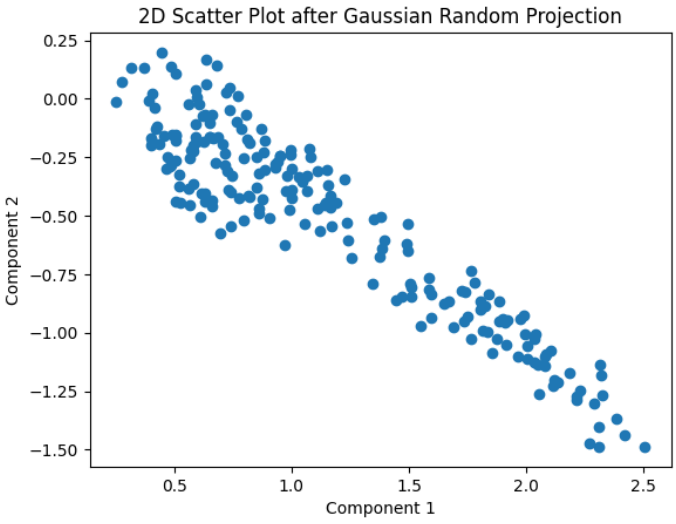

Figure 4: 2D Scatter Plot after Gaussian Random Projection

c)

The code for c) can be seen in F.

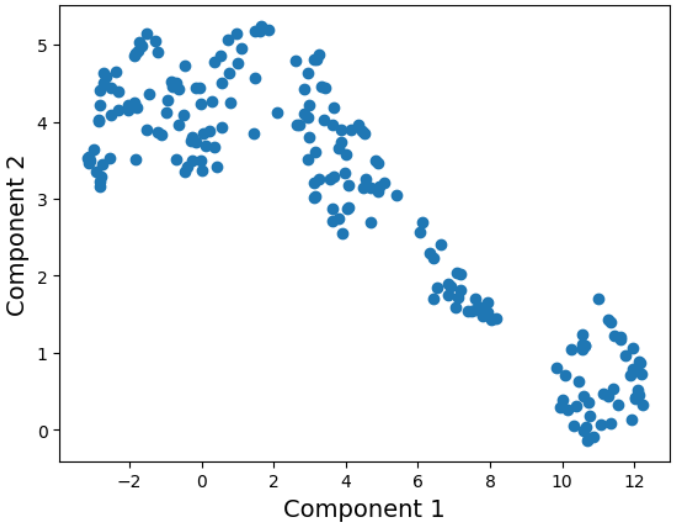

Figure 5: UMAP Riemannian manifold 2D

d)

It's not hard to imagine lines being drawn that separates the datapoints into three different sets. This is especially clear when looking at the graph we obtained by applying UMAP which displays in Figure 5 a high degree of delineation between all three clusters and in which the imagined lines become more apparent.

Problem 4: Evaluating clustering

Running G yield the following. A Rand index of 0.8641604010025062 and the highest accuracy and its associated mapping is (0.8857142857142857, 0: 2, 1: 3, 2: 1)

Problem 5: Agglomerative clustering

We see that the average linkage method works best H. This is a more "robust" linkage method since it considers all the points in a pair of clusters and not just the closest points, unlike single or complete linkage.

We also have well-behaved data; the classes are all roughly the same size and there are no particularly outlying datapoints that will have a disproporionate impact on the average. A plot a dendrogram of the hierarchical clustering can be seen in 6.

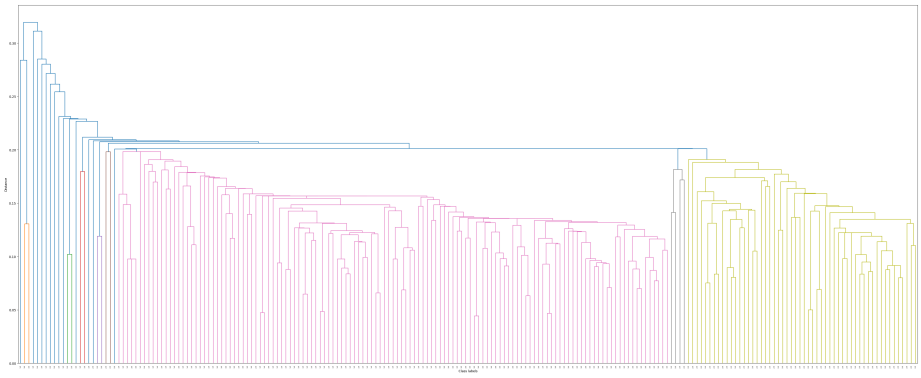

Figure 6: Plot of a dendrogram of the hierarchical clusterin

Its a bit hard to see in the Figure 6 but it showcase three different clusters one cluster to the right, one in the middle and lastly a cluster to the left.

A Libraries

```
!pip install umap-learn
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error,
    ↪ rand_score
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import itertools
from sklearn.random_projection import
    ↪ GaussianRandomProjection
import umap
from sklearn.preprocessing import StandardScaler,
    ↪ MinMaxScaler
from scipy.cluster.hierarchy import dendrogram,
    ↪ linkage
from sklearn.cluster import AgglomerativeClustering

num_clusters = 10
```

B Preprocessing the dataset

```
df = pd.read_csv("/content/seeds.tsv", delimiter='\t')
mapper = {'15.26' : "Area", # 1
          '14.84' : "Perimeter", # 2
          '0.871' : "Compactness", # 3
```

```
            '5.763' : "Kernel length", # 4
            '3.312' : "Kernel width", # 5
            '2.221' : "Asymmetry coefficient", # 6
            '5.22'  : "Kernel groove length" , # 7
            '1'     : "Numerical class label"} # 8

first_row = pd.DataFrame({v: [k] for k, v in mapper.items()})

df = df.rename(columns=mapper)
df = pd.concat([first_row, df], ignore_index=True) # [209 rows x 8 columns]
df = df.astype("float")
print(df)

scaler = MinMaxScaler()
df_features = df.drop("Numerical class label", axis=1)
class_labels = df["Numerical class label"]
normalized_df = pd.DataFrame(scaler.fit_transform(df_features), columns=df_features.columns)
df = pd.concat([normalized_df, class_labels], axis=1)
print("\n")
print(df)
```

C Determining the appropriate number of clusters

```
features = df.iloc[:, :-1].to_numpy()
inertias = []
for n in range(7):
      kmeans = KMeans(n_clusters=n + 1, random_state=0).fit(features)
      inertias.append(kmeans.inertia_)

plt.plot(inertias, range(1, 8))
plt.show()
```

D Problem 3 a)

```
features = df.columns[: -1]
combinations = list(itertools.product(features, features))
cleaned_combinations = [tup for tup in combinations if tup[0] != tup[1]]

```

```
for pair in cleaned_combinations:
  plt.scatter(df[pair[0]], df[pair[1]])
  plt.xlabel(pair[0])
  plt.ylabel(pair[1])
  plt.show()
```

E Problem 3 b)

```
features = df.iloc[:, :-1].to_numpy()
reducer = GaussianRandomProjection(n_components=2,
    ↪ random_state=42)
embedding = reducer.fit_transform(features)
print(len(features[0])) # 7 x 210

plt.scatter(embedding[:, 0], embedding[:, 1])
plt.xlabel('Component␣1')
plt.ylabel('Component␣2')
plt.title('2D␣Scatter␣Plot␣after␣Gaussian␣Random␣
    ↪ Projection')
plt.show()
```

F Problem 3 c)

```
reducer = umap.UMAP(random_state=42)
embedding = reducer.fit_transform(features)

plt.scatter(embedding[:, 0], embedding[:, 1])
plt.xlabel('Component␣1', fontsize=14)
plt.ylabel('Component␣2', fontsize=14)
plt.show()
```

G Evaluating clustering

```
features = df.iloc[:, :-1].to_numpy()
kmeans = KMeans(n_clusters=3,
    ↪ random_state=0).fit(features)
print("Rand␣index:", rand_score(df["Numerical␣class␣
    ↪ label"], kmeans.labels_))

unique_mappings = [dict(zip([0, 1, 2], perm)) for
    ↪ perm in itertools.permutations([1, 2, 3])] #
    ↪ clusters are keys
#print(unique_mappings)

accuracy_list = []
```

```
for mapping in unique_mappings: # {0: 2, 1: 3, 2: 1}
  sum = 0
  c1_class = mapping[0]        # {2 : 0, 3 : 1, 1 : 2}
  c2_class = mapping[1]
  c3_class = mapping[2]
  for i in range(210):
    if kmeans.labels_[i] == 0:
      if int(df.iloc[i, -1]) == c1_class:
        sum = sum + 1
    if kmeans.labels_[i] == 1:
      if int(df.iloc[i, -1]) == c2_class:
        sum = sum + 1
    if kmeans.labels_[i] == 2:
      if int(df.iloc[i, -1]) == c3_class:
        sum = sum + 1
  accuracy_list.append((float(sum/210), mapping))

max_accuracy = max(accuracy_list, key=lambda x: x[0])
print("The highest accuracy and its associated
   ↪ mapping:", max_accuracy)
```

H Agglomerative clustering

```
def accuracy(model):
  unique_mappings = [dict(zip([0, 1, 2], perm)) for
     ↪ perm in itertools.permutations([1, 2, 3])] #
     ↪ clusters are keys

  accuracy_list = []
  for mapping in unique_mappings:
    sum = 0
    c1_class = mapping[0]
    c2_class = mapping[1]
    c3_class = mapping[2]
    for i in range(210):
      if model.labels_[i] == 0:
        if int(df.iloc[i, -1]) == c1_class:
          sum = sum + 1
      if model.labels_[i] == 1:
        if int(df.iloc[i, -1]) == c2_class:
          sum = sum + 1
      if model.labels_[i] == 2:
        if int(df.iloc[i, -1]) == c3_class:
          sum = sum + 1

    accuracy_list.append((float(sum/210), mapping))
  return max(accuracy_list, key=lambda x: x[0])
features = df.iloc[:, :-1].to_numpy()
```

```
clustering_ward = AgglomerativeClustering(n_clusters=3, linkage="ward").fit(features)
clustering_complete = AgglomerativeClustering(n_clusters=3, linkage="complete").fit(features)
clustering_average = AgglomerativeClustering(n_clusters=3, linkage="average").fit(features)
clustering_single = AgglomerativeClustering(n_clusters=3, linkage="single").fit(features)

clustering_ward_acc = accuracy(clustering_ward)
clustering_complete_acc = accuracy(clustering_complete)
clustering_average_acc = accuracy(clustering_average)
clustering_single_acc = accuracy(clustering_single)

print("Ward acc :", clustering_ward_acc)
print("Complete acc :", clustering_complete_acc)
print("Average acc :", clustering_average_acc) # {0: 1, 1: 2, 2: 3})
print("Single acc :", clustering_single_acc) # The highest accuracy and its associated mapping: (0.8857142857142857, {0: 2, 1: 3, 2: 1})

# Plotting dendrogram for task 5

linkage_matrix = linkage(features)
fig = plt.figure(figsize=(50, 20))
labels = df["Numerical class label"].astype(str).tolist()
dn = dendrogram(linkage_matrix, labels=labels, color_threshold=0.2)
plt.xlabel("Class labels")
plt.ylabel("Distance")
plt.show()
```