DAT565/DIT407 Assignment 5

Louis PRUNIER
prunier.louis@icloud.com

Marco SPEZIALE
speziale@student.chalmers.se

2024-05-02

1 Load and Visualizing the classes in the data

1.1 A

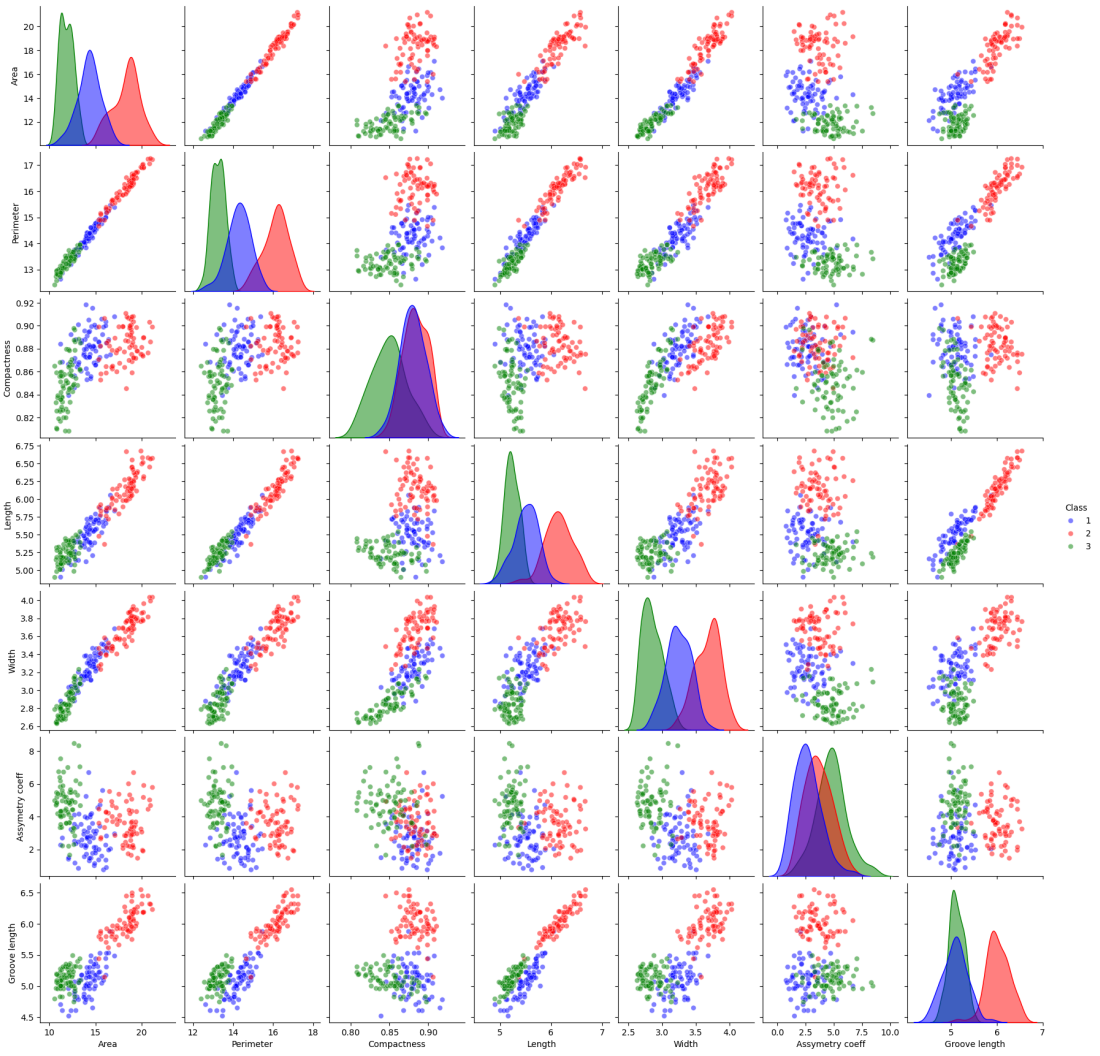

Figure 1: Scatter plot between each pair of features

The pair-plotting 1 shows a fairly consistent structure on the data. The samples of the third class (green) are consistently smaller than samples from the other two seed classes when it comes to any metric that has to do with size (area/perimeter/length/width etc.), whereas samples from the second class (red) are generally bigger. This suggests that there is a clear distinction in general size between the three seed species. Overall, the majority of the feature-pair plots clearly tell the different classes apart, which shows that the different seed species generally occupy different ranges of feature values that are unique to them. This makes it easier to predict which species a given sample belongs to.

As for the interesting feature-pair plot 2, we chose the length vs groove length plot, because even though the third class (green) seed species seems to be generally shorter than the first class (blue) species, their groove lengths are equal. The third class (green) species therefore seems to have a longer groove length in relation to its overall length than at least the first class (blue) species, which is interesting.

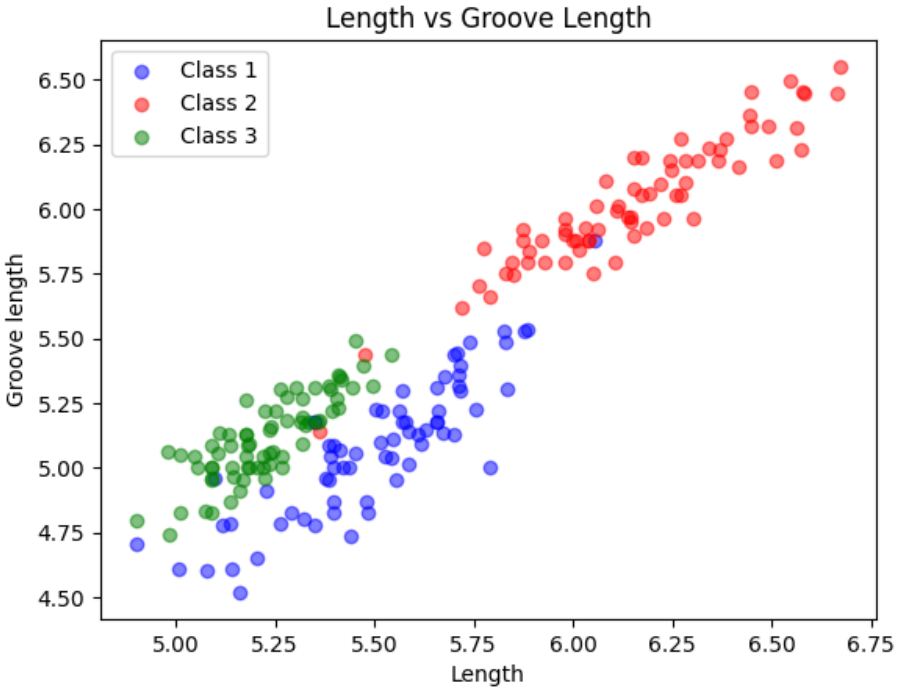

Figure 2: Scatter Plot of Length and Groove Length

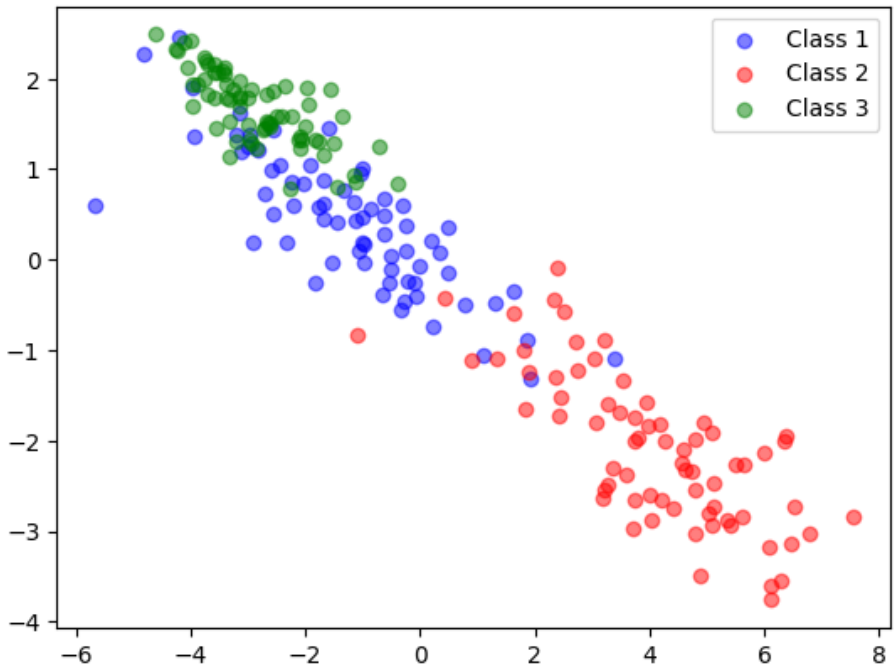

Figure 3: Scatter plot of data reduced to 2 dimensions using Gaussian Random Projection

1.2 B

The data in the plot 3 looks well separated in between classes. The third class (green) seems to stand out from the other two classes by having a range of higher values regarding some feature aspects while having a range of lower values regarding the other feature aspects. Presumably, the majority of the features that the x-axis consists of regard overall size of the seeds, while the y-axis probably contains the asymmetry aspect which the third class (green) showed to have higher levels of when assessing the pair-plots.

1.3 C

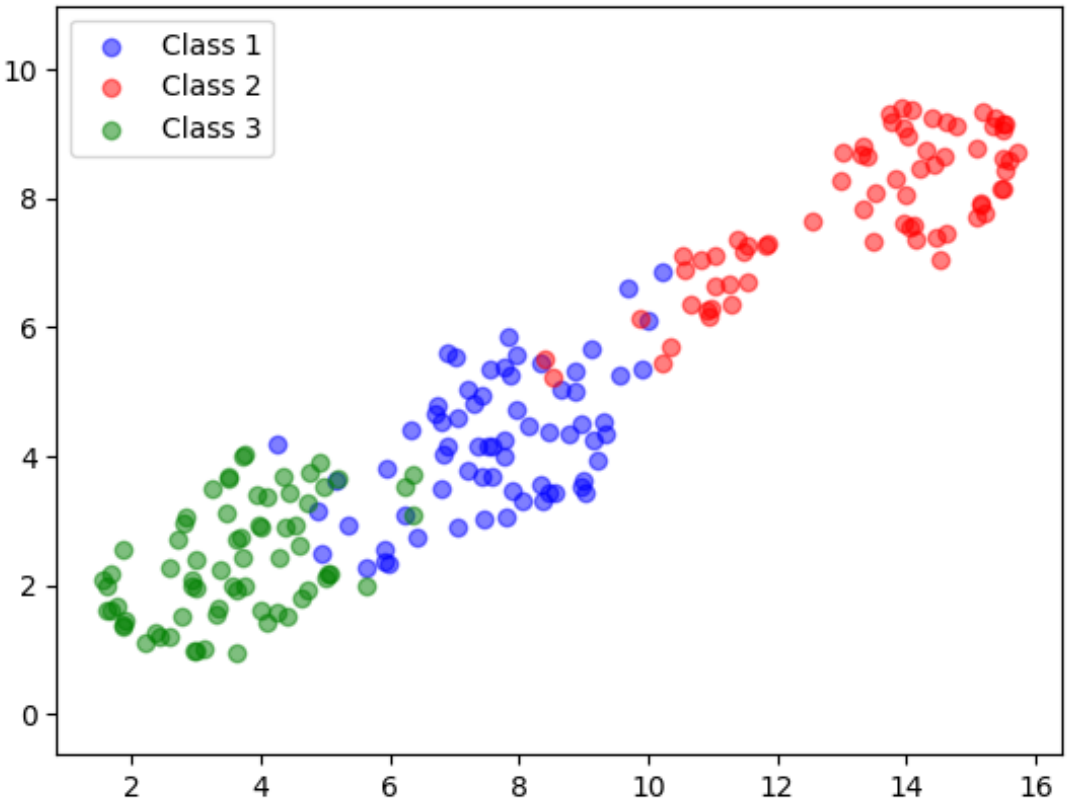

Figure 4: Scatter plot of data reduced to 2 dimensions using UMAP

1.4 D

The data does look linearly separable in the majority of the plots, as you can separate the majority of the points representing one class with the majority of the points representing another with a straight line. For clustering, this means that we probably are going to be able to divide the data into clusters that each represent a given class.

2 Unsupervised Learning: Determining the appropriate number of clusters

2.1 A

To normalize the data, we turned each value into their Z-score in relation to the feature they belong to, using ScikitLearn's StandardScaler.

2.2 B

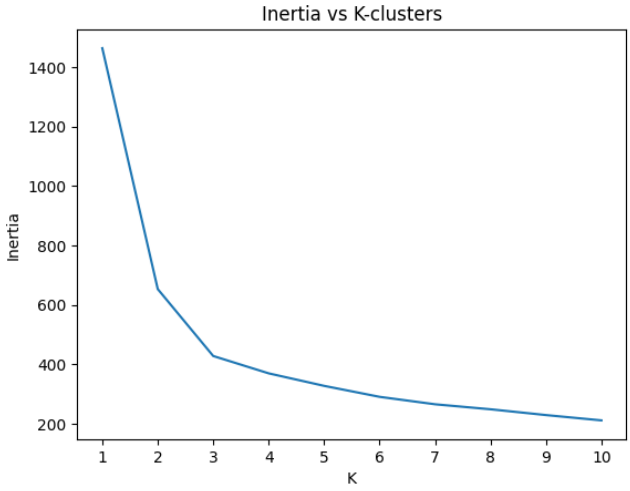

Figure 5: Elbow Method for optimal k

The appropriate number of clusters according to the plot 5 would be 3, because that's where the rate of decline smooths out, or stops decreasing significantly.

3 Evaluating clustering

With k = 3:

Rand Index: 0.773
Accuracy: 0.919

4 Supervised Learning: K-nearest neighbour

4.1 A

The approach we would take would be by trying out different values for k, and then compare the obtained confusion matrices with each other to see which value of k gave the most accurate one.

4.2 B

Chosen value of k was 1, which gave the following confusion matrix:

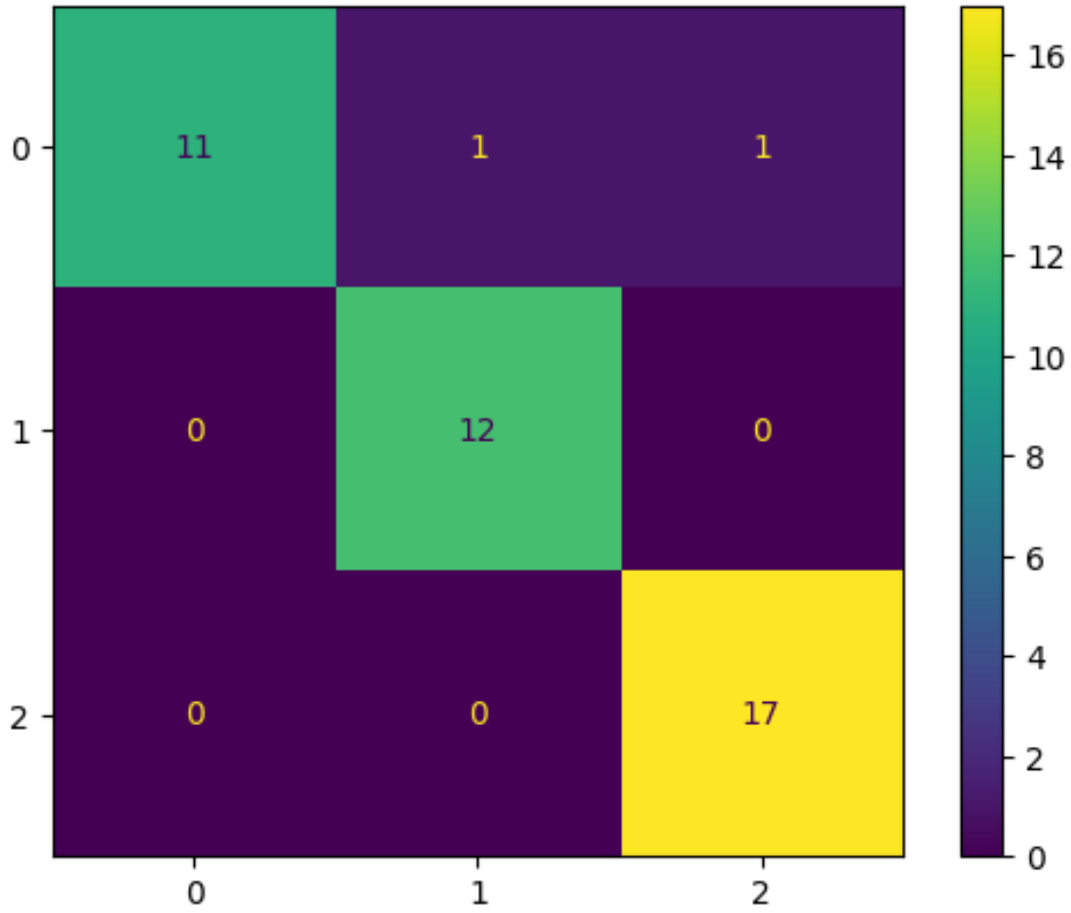

Figure 6: Confusion matrix of predicted class values against actual class values

4.3 C

Accuracy: 0.952

4.4 D

Our guess is that if we change the split so that the training-set is too small, the classifier is not going to have enough information about which regions clearly belong to which clusters, and so a new query point in such region would probably need to be compared to more than one nearest neighbour to increase its chances of being assigned the correct cluster. The accuracy, regardless, would decrease since less training points, and less diversity in positions, means that the classifier has less "determined space" to help it correctly classify which region belongs to which clustering/class.

5 Appendix

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.random_projection import
    GaussianRandomProjection
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,
    adjusted_rand_score, accuracy_score,
    ConfusionMatrixDisplay
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import umap


# read the data and add column names
data = pd.read_table("seeds.tsv")
data.columns = ["Area", "Perimeter", "Compactness", "
    Length", "Width", "Assymetry coeff", "Groove length
    ", "Class"]
data


## Visualizing the data

# Feature pairs:

# scatter between all pairs of features
colors = {1: "blue", 2: "red", 3: "green"}
sns.pairplot(data, hue="Class", palette=colors,
    plot_kws={"alpha": 0.5}, diag_kws={'alpha': 0.5})

# plot the interesting one
length = data[["Length", "Class"]]
grv_leangth = data[["Groove length", "Class"]]

lengthclass1 = length[length["Class"] == 1]
lengthclass2 = length[length["Class"] == 2]
lengthclass3 = length[length["Class"] == 3]

grv_leangthclass1 = grv_leangth[grv_leangth["Class"]
    == 1]
```

```
grv_leangthclass2 = grv_leangth[grv_leangth["Class"] == 2]
grv_leangthclass3 = grv_leangth[grv_leangth["Class"] == 3]

plt.scatter(lengthclass1["Length"], grv_leangthclass1["Groove length"], color="blue", alpha=0.5, label="Class 1")
plt.scatter(lengthclass2["Length"], grv_leangthclass2["Groove length"], color="red", alpha=0.5, label="Class 2")
plt.scatter(lengthclass3["Length"], grv_leangthclass3["Groove length"], color="green", alpha=0.5, label="Class 3")
plt.xlabel("Length")
plt.ylabel("Groove length")
plt.title("Length vs Groove Length")
plt.legend()
plt.show()


# Dimensionality reduction:

# extract the class values and then remove the class labels from the dataset
class1 = data[data["Class"] == 1]
class2 = data[data["Class"] == 2]
class3 = data[data["Class"] == 3]

class1 = class1.drop("Class", axis=1)
class2 = class2.drop("Class", axis=1)
class3 = class3.drop("Class", axis=1)

# normalize features to their Z-scores
data_no_classcol = data.drop("Class", axis=1)
scaler = StandardScaler()
scaler.fit(data_no_classcol)
scaled_data = scaler.transform(data_no_classcol)
class1 = scaler.transform(class1)
class2 = scaler.transform(class2)
class3 = scaler.transform(class3)

# reduce data with gaussian random projection
transformer = GaussianRandomProjection(n_components=2)
transformer.fit(scaled_data)

```

```
74  class1_2d = transformer.transform(class1)
75  class2_2d = transformer.transform(class2)
76  class3_2d = transformer.transform(class3)
77
78  plt.scatter(class1_2d[:, 0], class1_2d[:, 1], color="
        blue", alpha=0.5, label="Class␣1")
79  plt.scatter(class2_2d[:, 0], class2_2d[:, 1], color="
        red", alpha=0.5, label="Class␣2")
80  plt.scatter(class3_2d[:, 0], class3_2d[:, 1], color="
        green", alpha=0.5, label="Class␣3")
81  plt.legend()
82
83  # reduce data with umap
84  reducer = umap.UMAP()
85  embedding = reducer.fit(scaled_data)
86
87  class1_2d = reducer.transform(class1)
88  class2_2d = reducer.transform(class2)
89  class3_2d = reducer.transform(class3)
90
91  plt.scatter(class1_2d[:, 0], class1_2d[:, 1], color="
        blue", alpha=0.5, label="Class␣1")
92  plt.scatter(class2_2d[:, 0], class2_2d[:, 1], color="
        red", alpha=0.5, label="Class␣2")
93  plt.scatter(class3_2d[:, 0], class3_2d[:, 1], color="
        green", alpha=0.5, label="Class␣3")
94  plt.gca().set_aspect('equal', 'datalim')
95  plt.legend()
96
97
98  ## Unsupervised Learning
99
100 # normalize dataset (done already using Z-scores)
101 scaled_data
102
103 # perform k-means clustering on the data for k = 1, 2,
        3, ... , 10
104 inertia_values_k10 = []
105
106 for i in range(10):
107     kmeans = KMeans(n_clusters=i+1)
108     kmeans.fit(scaled_data)
109
110     clusters = kmeans.predict(scaled_data)
111     inertia = kmeans.inertia_
112     inertia_values_k10.append(inertia)
```

```
113     print("K = " + str(i+1) + ":")
114     print("Clusters:")
115     print(clusters)
116     print("Inertia:")
117     print(inertia)
118     print()
119
120
121 # plot inertia against k-levels
122 K = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
123 plt.plot(K, inertia_values_k10)
124 plt.xticks(K)
125 plt.xlabel("K")
126 plt.ylabel("Inertia")
127 plt.title("Inertia vs K-clusters")
128 plt.show()
129
130
131 # evaluate clustering
132 kmeans = KMeans(n_clusters=3)
133 kmeans.fit(scaled_data)
134 clusters = kmeans.predict(scaled_data)
135
136 correct_clustering = data["Class"] - 1
137
138 # reorder the cluster labels to match the class labels
139 print(clusters)
140 cm = confusion_matrix(correct_clustering, clusters)
141 print(cm)
142 cm_argmax = cm.argmax(axis=0)
143 clusters_ = np.array([cm_argmax[i] for i in clusters])
144 cm = confusion_matrix(correct_clustering, clusters_)
145 print(cm)
146 print(clusters_)
147
148 # rand index
149 rand_index = adjusted_rand_score(correct_clustering,
        clusters_)
150 print("Rand Index: " + str(rand_index))
151
152 # accuracy:
153 accuracy = accuracy_score(correct_clustering,
        clusters_)
154 print("Accuracy: ", accuracy)
155
156
```

```
157 ## Supervised Learning
158
159 # split the data into train and test sets
160 X = scaled_data
161 y = correct_clustering
162 X_train, X_test, y_train, y_test = train_test_split(X,
        y, test_size=0.2, random_state=5)
163
164 # run the KNeighbour classifier for different values
       of k and evaluate the best value for k
165 best_k = 1
166 highest_acc = 0
167
168 for k in range(50):
169     knc = KNeighborsClassifier(n_neighbors=(k+1))
170     knc.fit(X_train, y_train)
171     y_pred = knc.predict(X_test)
172     accuracy = accuracy_score(y_test, y_pred)
173     if (accuracy > highest_acc):
174         highest_acc = accuracy
175         best_k = (k+1)
176 print(best_k)
177
178 # confusion matrix for k = 1
179 knc = KNeighborsClassifier(n_neighbors=(1))
180 knc.fit(X_train, y_train)
181 y_pred = knc.predict(X_test)
182
183 disp = ConfusionMatrixDisplay.from_estimator(knc,
       X_test, y_test, display_labels="class")
184 print(disp.confusion_matrix)
185 print("Accuracy:␣", accuracy_score(y_test, y_pred))
```