

# DAT565/DIT407 Assignment 5

Giacomo Guidotto  
gusguigi@student.gu.se

Leong Jia Yi, Janna  
gusleoji@student.gu.se

2024-10-10

## 1 Preprocessing the dataset

The dataset we're analyzing contains different features of wheat kernels with varying ranges of values. We normalized them using the **StandardScaler** class from sklearn to make them as uniform as possible.

## 2 Determining the appropriate number of clusters

After implementing the k-means clustering algorithm on the dataset, Figure 1 displays the inertia values corresponding to different numbers of clusters.

Based on this plot, we can assume that the optimal number of clusters,  $k$ , is **3**, as this choice offers a good balance between a relatively low cluster count and an acceptable level of inertia.

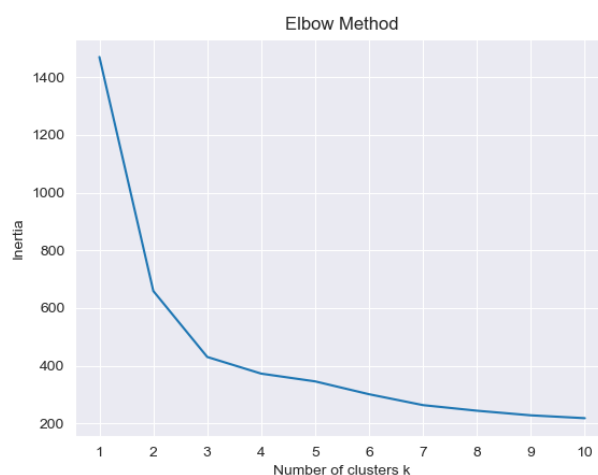


Figure 1: Clustering inertia for different given numbers of clusters, plotting an elbow-shaped curve

### 3 Visualizing the classes

Figure 2 illustrates the pairwise relationships among the dataset's features.

Numerous positive relationships are observed among dimensional features, indicating that as one dimension increases, others tend to grow as well. However, while the area generally increases with length, the relationship may not be strictly linear due to geometric properties.

Moreover, features like the asymmetry coefficient show a low correlation with dimensional measurements, suggesting they may capture aspects of the data unrelated to size, such as shape irregularities.

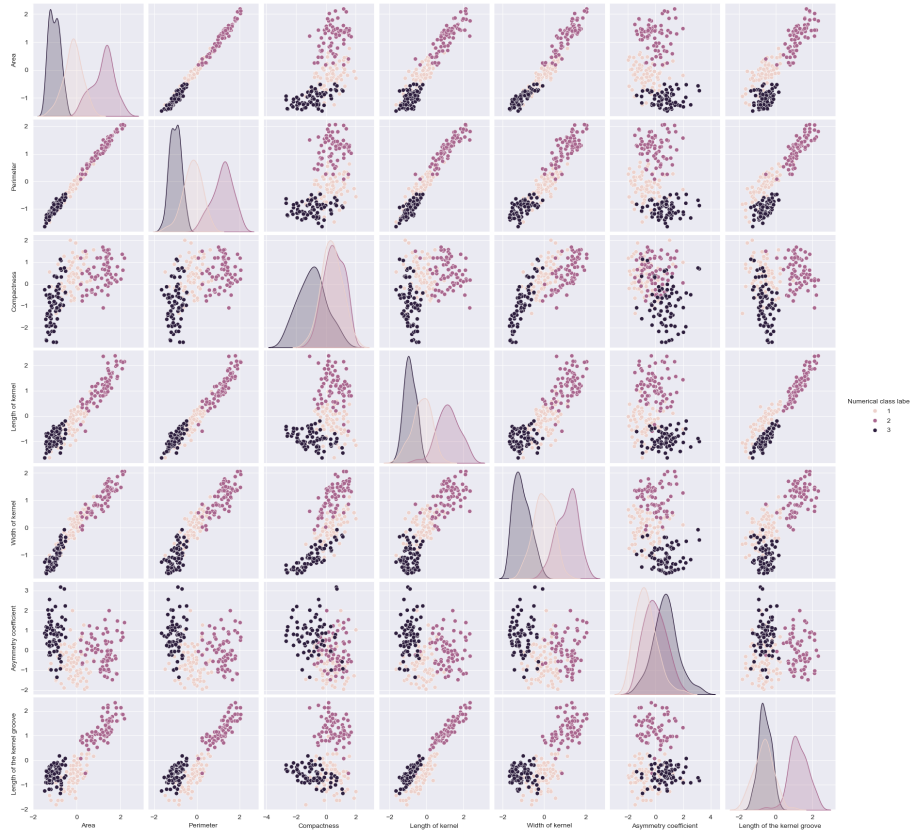


Figure 2: The dataset features plotted against each other

After applying a two-dimensional Gaussian Random Projection on the seven-feature dataset, Figure 3 displays data that seem to have a negative correlation and clusters that appear to be separated.

While following an application of a two-dimensional UMAP, Figure 4 doesn't show a distinct relationship but appears to make the cluster more distinct.

Upon considering all this visualization we can conclude that the data presents a linear separation due to the distinction between data of different clusters. This will enable the clustering algorithm to perform well with the given data.

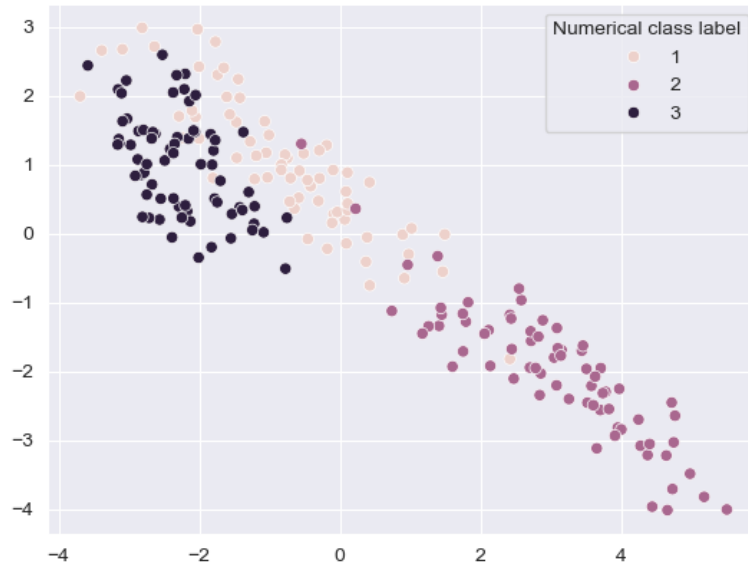


Figure 3: Gaussian random projection of the dataset

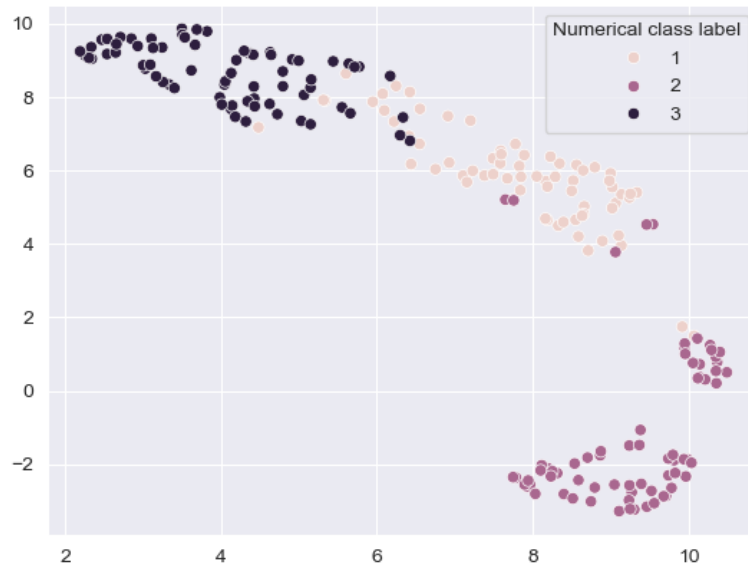


Figure 4: UMAP projection of the dataset

## 4 Evaluating clustering

Utilizing the k-means algorithm on the dataset, with the number of clusters set to match the number of given labels, we obtained a *Rand Index* of **0.900** and an *Accuracy* of **0.605**.

## 5 Agglomerative clustering

In the concluding phase of our analysis, we applied hierarchical clustering to the dataset. Testing various linkage methods resulted in the following accuracies:

- *ward* method: **0.619**
- *complete* method: **0.543**
- *average* method: **0.586**
- *single* method: **0.338**

The *single* linkage method yielded the lowest performance, whereas the *ward* method achieved the highest.

This is because the *ward* linkage method excels in scenarios where clusters are expected to be compact, spherical, and well-separated. Its focus on minimizing within-cluster variance leads to more accurate and meaningful clustering outcomes at the presumed correct number of clusters. (missing citation)

Figure 5 illustrates the dendrogram of this linkage method.

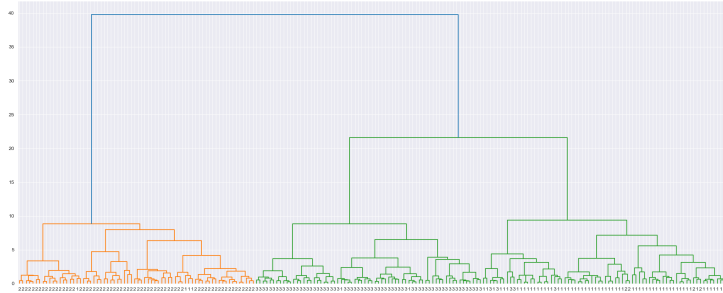


Figure 5: Dendrogram of the *ward* hierarchical clustering

## A Code

Following is the Python code that we used to extract and visualize the information presented in this report:

## A.1 Problem 1: Preprocessing the dataset

Read the dataset:

```
1 import itertools
2
3 import numpy as np
4 import pandas as pd
5 from sklearn.preprocessing import StandardScaler
6
7 column_labels = [
8     'Area',
9     'Perimeter',
10    'Compactness',
11    'Length_of_kernel',
12    'Width_of_kernel',
13    'Asymmetry_coefficient',
14    'Length_of_the_kernel_groove',
15    'Numerical_class_label',
16 ]
17 df = pd.read_csv(
18     'data/seeds.tsv',
19     sep='\t',
20     header=None,
21     names=column_labels
22 )
23 df.head()
```

Scale the dataset using a normalizer:

```
1 scaler = StandardScaler()
2
3 scaled_data = pd.DataFrame(
4     scaler.fit_transform(
5         df.drop(columns="Numerical_class_label")
6     ),
7     columns=column_labels[:-1]
8 )
9 scaled_df = pd.concat(
10    [scaled_data, df["Numerical_class_label"]],
11    axis=1
12 )
13 scaled_df.head()
```

## A.2 Problem 2: Determining the appropriate number of clusters

Iterating over a range of cluster numbers, determine the optimal number of clusters using the elbow method:

```
1 from sklearn.cluster import KMeans
2 import matplotlib.pyplot as plt
```

```

3
4 k = range(1, 11)
5 inertia = []
6 for i in k:
7     kmeans = KMeans(
8         n_clusters=i,
9         random_state=0,
10        n_init="auto"
11    ).fit(scaled_data)
12    inertia.append(kmeans.inertia_)
13
14 plt.plot(k, inertia)
15 plt.title('Elbow Method')
16 plt.xlabel('Number of clusters k')
17 plt.ylabel('Inertia')
18 plt.xticks(k)
19 plt.savefig("figures/elbow_method.png")

```

We can see that the optimal number of clusters is 3.

### A.3 Problem 3: Visualizing the classes

Scatterplot the pairs of features, coloring the points according to the cluster they belong to:

```

1 import seaborn as sns
2
3 sns.pairplot(
4     scaled_df,
5     hue="Numerical_class_label"
6 )
7 plt.savefig("figures/features_pairplot.png")

```

Scatterplot the Gaussian random projections of the dataset, coloring the points according to the cluster they belong to:

```

1 from sklearn.random_projection import \
2     GaussianRandomProjection
3
4 embedding = GaussianRandomProjection(
5     random_state=42, n_components=2
6 ).fit_transform(scaled_data)
7
8 sns.scatterplot(
9     x=embedding[:, 0],
10    y=embedding[:, 1],
11    hue=df["Numerical_class_label"]
12 )
13 plt.savefig(
14     "figures/gaussian_random_projection.png"
15 )

```

Scatterplot the UMAP projections of the dataset, coloring the points according to the cluster they belong to:

```
1 from umap import UMAP
2
3 embedding = UMAP(
4     random_state=42, n_components=2
5 ).fit_transform(scaled_data)
6
7 sns.scatterplot(
8     x=embedding[:, 0],
9     y=embedding[:, 1],
10    hue=df["Numerical_class_label"]
11 )
12 plt.savefig("figures/umap_projection.png")
```

#### A.4 Problem 4: Evaluating clustering

Cluster the dataset using the optimal number of clusters:

```
1 k = scaled_df["Numerical_class_label"].nunique()
2
3 kmeans = KMeans(
4     n_clusters=k,
5     random_state=0,
6     n_init="auto"
7 ).fit(scaled_data)
```

Compute the rand index:

```
1 from sklearn.metrics import rand_score, \
2     accuracy_score
3
4 rand_index = rand_score(
5     scaled_df["Numerical_class_label"],
6     kmeans.labels_
7 )
8 print(f"Rand_index: {rand_index}")
```

Compute the accuracy by finding the maximum accuracy on all the cluster labels permutations:

```
1 def compute_accuracy(y_true, y_pred):
2     max_accuracy = float('-inf')
3     combinations = list(
4         itertools.permutations(range(0, k))
5     )
6     for combination in combinations:
7         transformed_labels = np.array(
8             [combination[label] for label in y_pred]
9         )
10        max_accuracy = max(
11            max_accuracy,
```

```

12         accuracy_score(y_true, transformed_labels)
13     )
14     return max_accuracy
15
16
17 accuracy = compute_accuracy(
18     scaled_df["Numerical_class_label"],
19     kmeans.labels_
20 )
21 print(f"Accuracy: {accuracy}")

```

## A.5 Problem 5: Agglomerative clustering

Hierarchically cluster the dataset with all the linkage methods:

```

1 from sklearn.cluster import \
2     AgglomerativeClustering
3
4 best_method = None
5 best_accuracy = float('-inf')
6 linkage_methods = [
7     "ward",
8     "complete",
9     "average",
10    "single"
11 ]
12 for method in linkage_methods:
13     cluster = AgglomerativeClustering(
14         n_clusters=k,
15         linkage=method
16     ).fit(scaled_data)
17     accuracy = compute_accuracy(
18         scaled_df["Numerical_class_label"],
19         cluster.labels_
20     )
21
22     if accuracy > best_accuracy:
23         best_accuracy = accuracy
24         best_method = method
25
26     print(
27         f"Accuracy for linkage method {method}: {accuracy}"
28     )
29
30 print(f"Best linkage method: {best_method}")

```

Plot the dendrogram for the best linkage method:

```

1 from scipy.cluster.hierarchy import dendrogram, \
2     linkage
3

```



```
4 plt.figure(figsize=(25, 10))
5 dendrogram(
6     linkage(scaled_data, method=best_method),
7     labels=scaled_df[
8         "Numerical_class_label"
9     ].values,
10    leaf_font_size=10,
11    leaf_rotation=0
12 )
13 plt.savefig("figures/dendrogram.png")
```