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 which removes the mean and scales the data to unit variance.

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 is located at the ”elbow”, offering a good balance between a relatively low cluster count and an acceptable level of inertia.

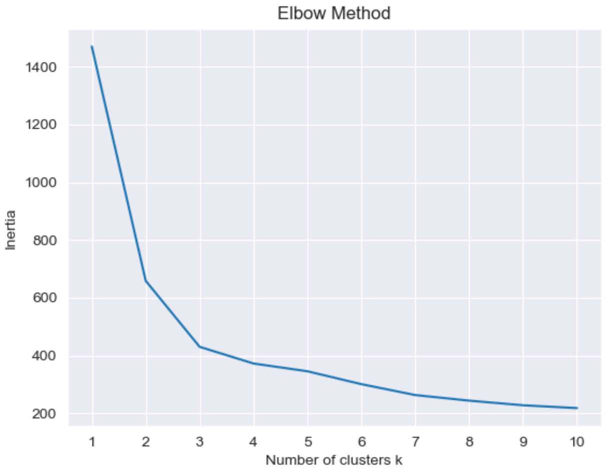

Figure 1: Clustering inertia for different given numbers of clusters, plotting an elbow-shaped curve

3 Visualizing the classes

While computing the pairwise relationships among the dataset's features, numerous positive relationships are observed among dimensional features, indicating that as one dimension increases, others also tend to grow. 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, in particular, displays the relationship between the perimeter and the area. It is interesting because the features have a solid linear relationship, pointing to a straight line with few outliers. We see a very distinct segregation between the classes. Thus, these classes allow us to distinguish between classes.

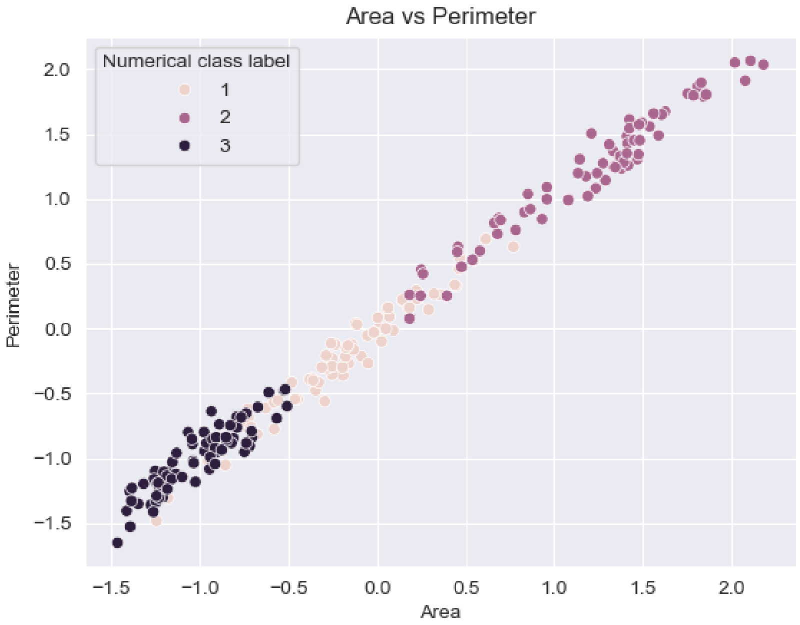

Figure 2: The linear relationship between area and perimeter in the dataset

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.

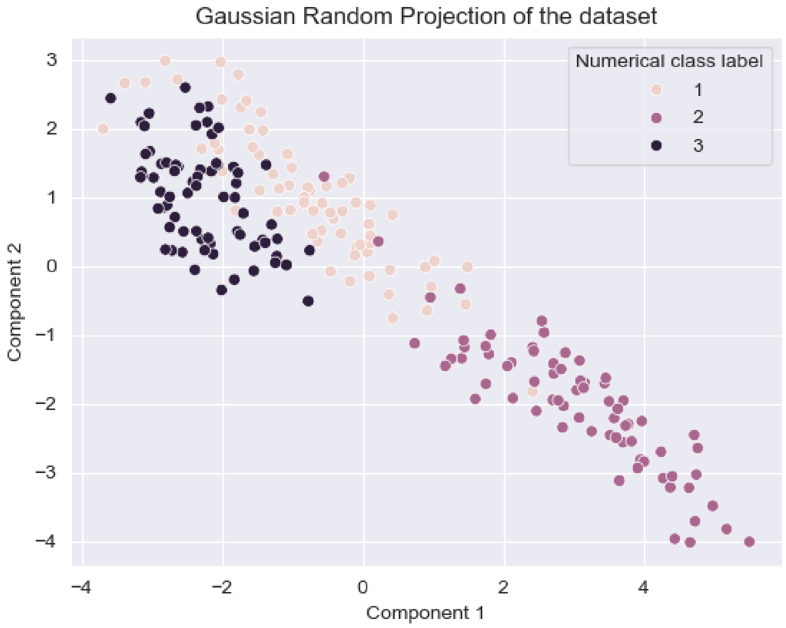

Figure 3: Gaussian random projection of the dataset

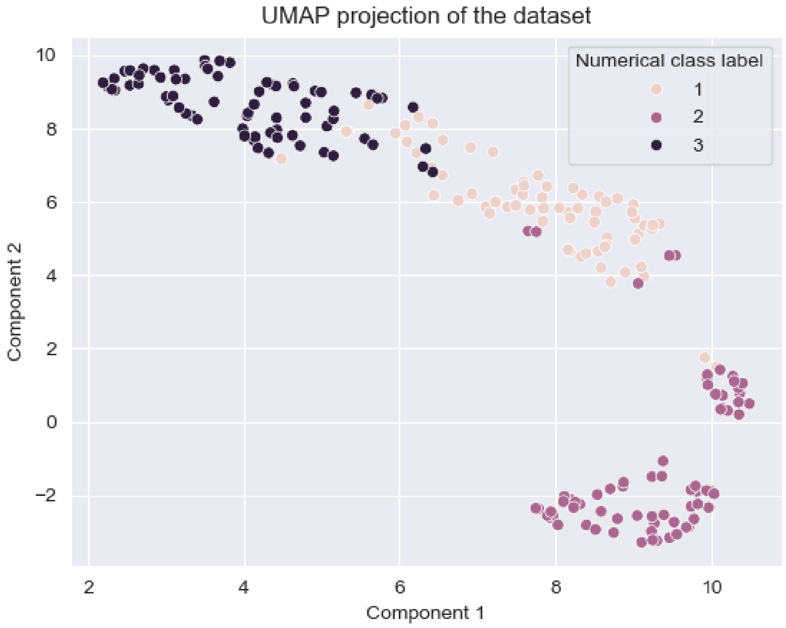

Figure 4: UMAP projection of the dataset

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 **0.919**.

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: **0.929**
- *complete* method: **0.876**
- *average* method: **0.881**
- *single* method: **0.348**

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 data is numerical and clusters are spherical[1]. As all features in this dataset are numeric and clusters are somewhat spherical as seen in Figure 4, the ward linkage method works the best by minimizing within-cluster variance, leading to more accurate and meaningful clustering outcomes at the presumed correct number of clusters. On the other hand, *single* linkage method does not work well on spherical clusters[2], explaining its poor performance on this dataset.

Figure 5 illustrates the dendrogram of this linkage method.

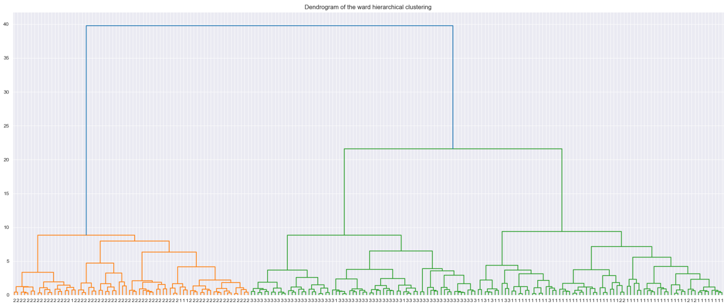

Figure 5: Dendogram of the *ward* hierarchical clustering

References

[1] Bodjam. *Agglomerative Hierarchical Clustering with Ward's Linkage*. Retrieved 2023-10-28. 2024. URL: https://medium.com/@malamine_58760/agglomerative-hierarchical-clustering-with-wards-linkage-305a0436c1a7#:~:text=Difficult%20to%20handle%20categorical%20data,specifically%20designed%20for%20categorical%20data..

[2] Scikit-learn. *Comparing different hierarchical linkage methods on toy datasets*. Retrieved 2023-09-28. nd. URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html.

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:

```
import itertools

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

column_labels = [
        'Area',
        'Perimeter',
        'Compactness',
        'Length␣of␣kernel',
        'Width␣of␣kernel',
        'Asymmetry␣coefficient',
        'Length␣of␣the␣kernel␣groove',
        'Numerical␣class␣label',
]
df = pd.read_csv(
        'data/seeds.tsv',
        sep='\t',
        header=None,
        names=column_labels
)
df.head()
```

Scale the dataset using a normalizer:

```
scaler = StandardScaler()

scaled_data = pd.DataFrame(
        scaler.fit_transform(
                df.drop(columns="Numerical␣class␣label")
        ),
        columns=column_labels[:-1]
)
scaled_df = pd.concat(
        [scaled_data, df["Numerical␣class␣label"]],
        axis=1
)
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:

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

k = range(1, 11)
inertia = []
for i in k:
        kmeans = KMeans(
                n_clusters=i,
                random_state=0,
                n_init="auto"
        ).fit(scaled_data)
        inertia.append(kmeans.inertia_)

plt.plot(k, inertia)
plt.title('Elbow␣Method')
plt.xlabel('Number␣of␣clusters␣k')
plt.ylabel('Inertia')
plt.xticks(k)
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:

```
import seaborn as sns

sns.pairplot(
        scaled_df,
        hue="Numerical␣class␣label"
)
plt.savefig("figures/features_pairplot.png")
```

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

```
from sklearn.random_projection import \
        GaussianRandomProjection

embedding = GaussianRandomProjection(
        random_state=42, n_components=2
).fit_transform(scaled_data)

sns.scatterplot(
        x=embedding[:, 0],
        y=embedding[:, 1],
        hue=df["Numerical␣class␣label"]
)
plt.savefig(
        "figures/gaussian_random_projection.png"
 )
```

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

```
from umap import UMAP

embedding = UMAP(
        random_state=42, n_components=2
).fit_transform(scaled_data)

sns.scatterplot(
        x=embedding[:, 0],
        y=embedding[:, 1],
        hue=df["Numerical␣class␣label"]
)
plt.savefig("figures/umap_projection.png")
```

A.4 Problem 4: Evaluating clustering

Cluster the dataset using the optimal number of clusters:

```
k = scaled_df["Numerical␣class␣label"].nunique()

kmeans = KMeans(
        n_clusters=k,
        random_state=0,
        n_init="auto"
).fit(scaled_data)
```

Compute the rand index:

```
from sklearn.metrics import rand_score, \
        accuracy_score

rand_index = rand_score(
        scaled_df["Numerical␣class␣label"],
        kmeans.labels_
)
print(f"Rand␣index:␣{rand_index}")
```

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

```
def compute_accuracy(y_true, y_pred):
        max_accuracy = float('-inf')
        combinations = list(
                itertools.permutations(range(0, k))
        )
        for combination in combinations:
                transformed_labels = np.array(
                        [combination[label] for label in y_pred]
                )
                max_accuracy = max(
                        max_accuracy,
                        accuracy_score(y_true, transformed_labels)
                )
        return max_accuracy


accuracy = compute_accuracy(
        scaled_df["Numerical␣class␣label"],
        kmeans.labels_
)
print(f"Accuracy:␣{accuracy}")
```

A.5 Problem 5: Agglomerative clustering

Hierarchically cluster the dataset with all the linkage methods:

```
from sklearn.cluster import \
        AgglomerativeClustering

best_method = None
best_accuracy = float('-inf')
linkage_methods = [
        "ward",
        "complete",
        "average",
        "single"
]
for method in linkage_methods:
        cluster = AgglomerativeClustering(
                n_clusters=k,
                linkage=method
        ).fit(scaled_data)
        accuracy = compute_accuracy(
                scaled_df["Numerical␣class␣label"],
                cluster.labels_
        )

        if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_method = method

        print(
                f"Accuracy for linkage method {method}: {accuracy}"
        )

print(f"Best linkage method: {best_method}")
```

Plot the dendrogram for the best linkage method:

```
from scipy.cluster.hierarchy import dendrogram, \
        linkage

plt.figure(figsize=(25, 10))
dendrogram(
        linkage(scaled_data, method=best_method),
        labels=scaled_df[
                "Numerical class label"
        ].values,
        leaf_font_size=10,
        leaf_rotation=0
)
plt.savefig("figures/dendrogram.png")
```