# DAT565/DIT407 Assignment 5

Saif Sayed
gussayedfa@student.gu.se

Gona Ibrahim Abdulrahman
gusibrgo@student.gu.se

2024-05-02

This is a report for assignment 5 for the course *Introduction to Data Science & AI* from Chalmers and Gothenburg University.

## Problem 1: Loading and Visualizing the classes in the data

Firstly, we read the dataset into Pandas, providing the appropriate delimiter (`\t`) and specifying that there is no header in the file. We then add the column names as per the table in the problem description.

**(A)** In task A, we create scatter plots between each pair of features using nested loops. For each pair of features, we iterate over the unique class labels and scatter the points with the corresponding class label color. We label the x-axis and y-axis with the feature names, add a legend, and provide a title for each plot.

Some scatter plots exhibit a discernible pattern where the data points appear to align in a structured manner, while others display a more scattered distribution. For example, when examining the scatter plot between perimeter and area, the points form a linear pattern, suggesting a strong correlation between the two variables. This indicates that, regardless of the species, the area of the seed is dependent on its perimeter.

The scatter plots incorporate color coding, allowing us to distinguish between different species based on the legend. Notably, the points corresponding to each species tend to cluster together in distinct regions of the graph, effectively separating them from points belonging to other species.

One particularly intriguing scatter plot would be either the scatter plot between Perimeter and Area or Length of Kernel and Perimeter. These plots demonstrate a clear distinction between species, as the points representing each species remain separate and do not overlap with those of other species. This distinct separation of data points by species makes these plots particularly compelling for visualizing the species classification **(see Figure 1)**.

**(B)** We use the Gaussian random projection class from `scikit-learn` to per-

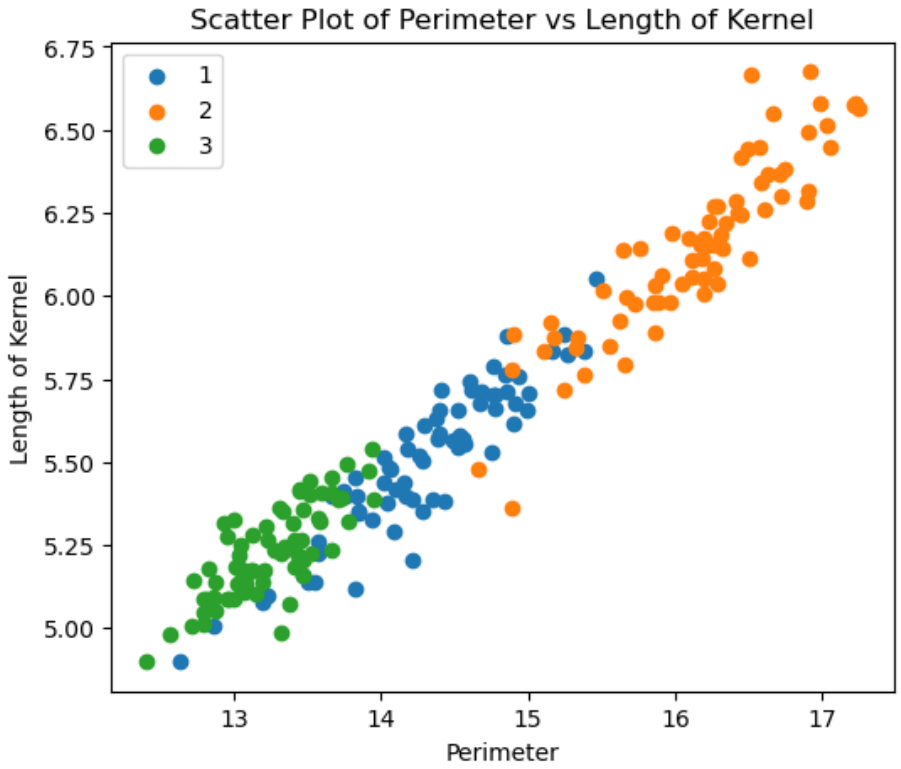


Figure 1: Length of Kernel Vs. Perimeter

form dimensionality reduction and visualize the data. After projecting the data, we create a scatter plot of the projected data. Each point in the scatter plot represents a wheat kernel, and the color of each point represents the class label of the wheat kernel. The scatter plot remains discernible **(see Figure 2)**.

**(C)** We apply UMAP to project the data onto a two-dimensional space using the UMAP class. Then, we create a scatter plot of the projected data, where each point represents a wheat kernel. The color of each point corresponds to the class label of the wheat kernel **(see Figure 3)**.

**(D)** We have observed that not all scatter plots demonstrate a linear relationship. The plots that do exhibit linearity involve pairs of features that are correlated, such as Length of Kernel, Width of Kernel, Perimeter, and Area. These features are directly correlated with each other and display a linear relationship. As each species possesses distinct values for these properties, the plots reveal linearity and distinguishability among seed species. Consequently, clustering algorithms can effectively classify the species based on these specific features. However, for other features that lack separability and exhibit overlapping clusters, the task of classifying the seeds becomes more challenging.

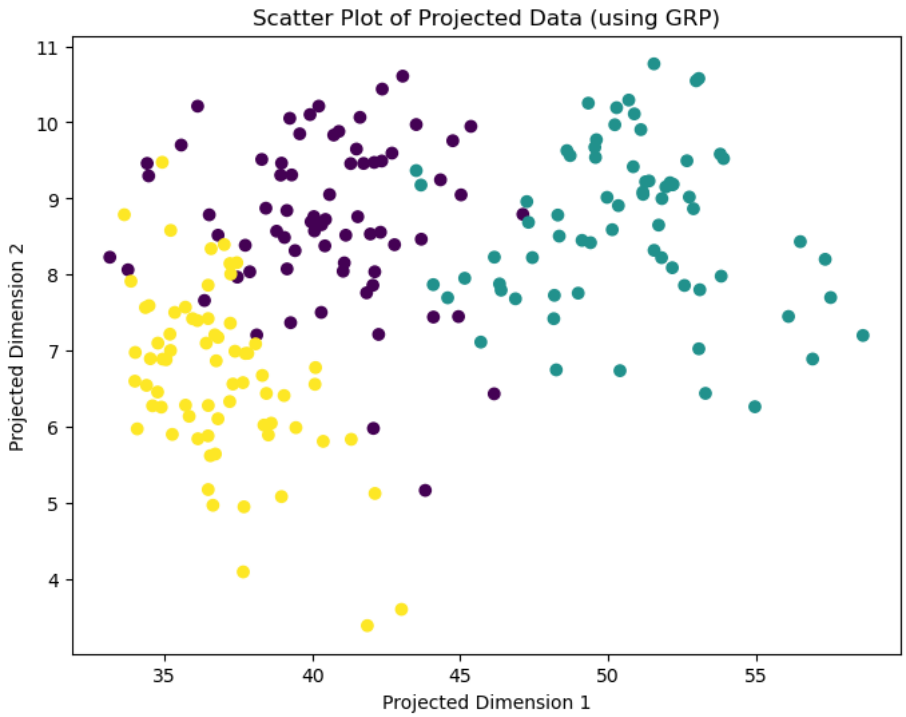


Figure 2: Gaussian Random Projection

Our source code can be found in Appendix A of this document.

## Problem 2: Unsupervised Learning: Determining the appropriate number of clusters

**(A)** To begin, we read the dataset into a pandas DataFrame. Then, we remove the class label column from the DataFrame, as it is not needed for the clustering task. Next, we apply normalization to the remaining features. To normalize the dataset, we use the StandardScaler class from the scikit-learn library. This scaler standardizes the features by subtracting the mean and dividing by the standard deviation. This process transforms the features to have a mean of zero and a standard deviation of one, effectively bringing them to a comparable scale. Finally, we converted the normalized data back to a pandas Dataframe.

**(B)** After applying normalization, we proceed with the clustering task. We use the K-means algorithm for this purpose. K-means is an iterative algorithm that aims to partition the data into K clusters, where K is a predefined number. In our case, we explore different values of K to assess the optimal number of clusters for the dataset. During the clustering process, we calculate the inertia for each value of K. Inertia represents the sum of squared distances of samples to their closest cluster center. It is a measure of how well the data points within

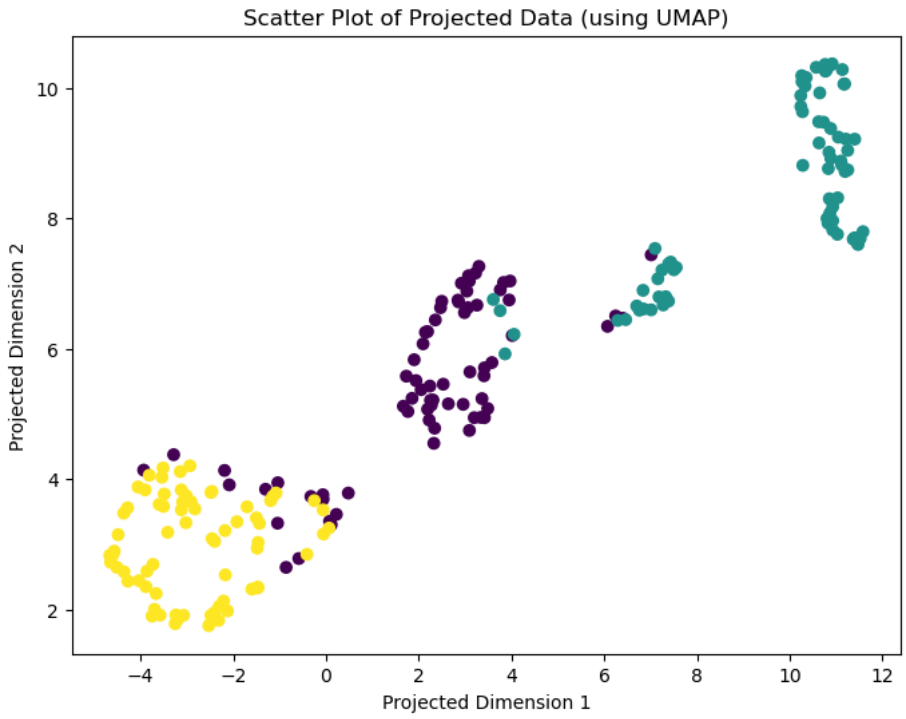


Figure 3: UMAP Projection

each cluster are grouped together. Finally, we plot the inertia values as a function of K to analyze the results **(see Figure 4)**. By visually examining the graph, we can observe a noticeable change in the inertia values starting from $K = 3$, which also corresponds to the elbow point. Based on this observation, we can determine that the appropriate number of clusters is 3.

## Problem 3: Evaluating clustering

In this task, we evaluate the quality of the clustering results obtained using k-means by computing the Rand Index and accuracy. But this time, we applied k-means to our normalised data so that k matches the number of underlying species of wheat.

**Compute Rand Index:** The adjusted_rand_score function from `scikit-learn` is used to calculate the Rand Index between the cluster labels (cluster_labels) obtained from k-means clustering and the original class labels (class_labels). The resulting value of Rand Index is 0.77 (2sf).

**Compute accuracy:** To compute the accuracy of the clustering, our code finds the permutation based on the number of all unique labels. It iterates through each permutation and calculates the accuracy score by assigning the permuted labels to the cluster labels. If the accuracy is higher than the current

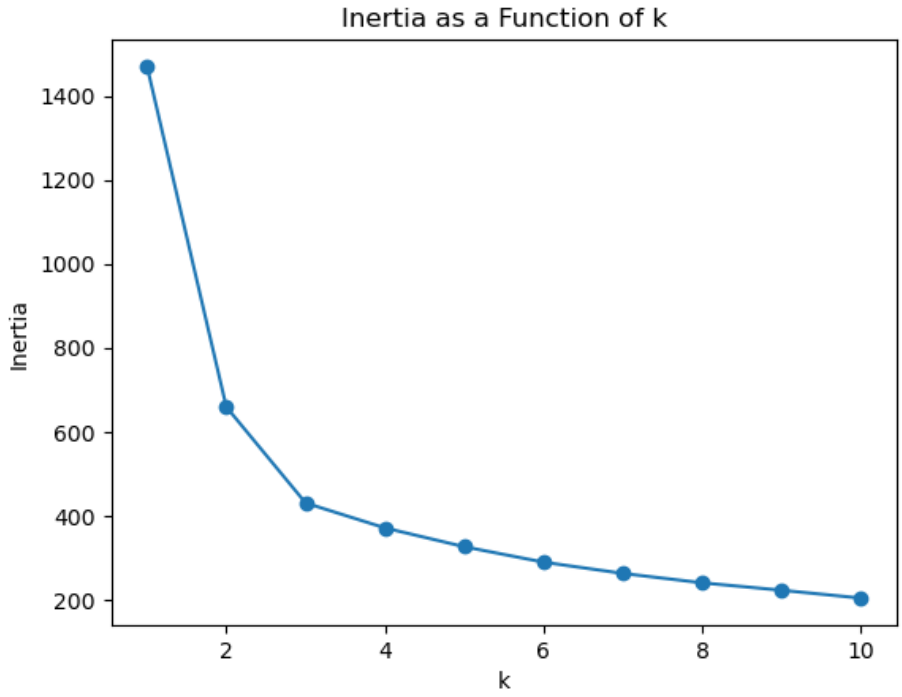


Figure 4: Inertia as a function of k.

best accuracy, the variable best_accuracy is updated. The resulting accuracy value is 0.92 (2sf).

## Problem 4: Supervised Learning: K-nearest neighbour

**(A)** When choosing a value, our approach would be to start with a small value and gradually increase it. While doing so, we also plan to calculate the accuracy score against the test sets for several expected values of the parameter n_neighbour. Additionally, plotting a graph to visualize this would help us to compare the accuracy score at different values. Our goal is to find a balance between underfitting (too small n_neighbors) and overfitting (too large n_neighbors).

**(B)** Our chosen value for n_neighbour is 5. From our plot, we can determine that at $n\_neighbour = 5$ the accuracy score complies with both the training set and testing set **(see Figure 5)**. We also experimented for values up to 150, although we got similar accuracy scores, we decided to not use a higher value to avoid overfitting our model. The code is included in the Appendix A of this document.

**(C)** For our chosen value, the accuracy on the test set is 0.93 (2sf).

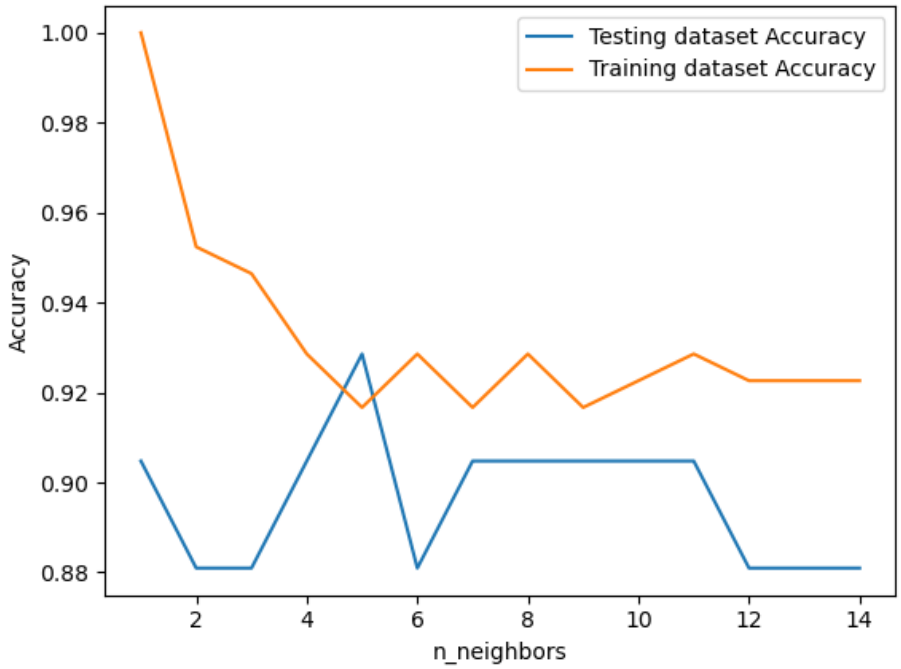


Figure 5: Accuracy Vs. n_neighbors

**(D)** Using a different split for train-test would result in different accuracy values, which can vary from the results obtained with the original split. The randomness in the train-test split introduces variability in the composition of the training and test sets. This variability can lead to different outcomes, particularly if the new training split is imbalanced, meaning it contains a higher proportion of data from a specific seed species. In such cases, the model may become biased towards the overrepresented seed species, resulting in lower accuracy when evaluated on the test set.

# A Python code

```
"""
Assignment 5
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.random_projection import
    GaussianRandomProjection
from umap import UMAP
```

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import adjusted_rand_score
import itertools
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
"""
1. Load and Visualizing the classes in the data
"""
data = pd.read_csv("seeds.tsv", delimiter="\t", header
    =None)

column_names = [
    "Area",
    "Perimeter",
    "Compactness",
    "Length of Kernel",
    "Width of Kernel",
    "Asymmetry Coefficient",
    "Length of Kernel Groove",
    "Class Label"
]

data.columns = column_names
print(data.columns)

# a.
feature_columns = data.columns[:-1]  # Exclude the
    last column (Class Label)
class_labels = data["Class Label"].unique()

for i in range(len(feature_columns)):
    for j in range(i + 1, len(feature_columns)):
        plt.figure(figsize=(6, 5))
        for label in class_labels:
            subset = data[data["Class Label"] == label
                ]
            plt.scatter(subset[feature_columns[i]],
                subset[feature_columns[j]], label=label
                )
        plt.xlabel(feature_columns[i])
        plt.ylabel(feature_columns[j])
        plt.legend()
        plt.title(f"Scatter Plot of {feature_columns[i
            ]} vs {feature_columns[j]}")
        plt.show()
# b.
grp = GaussianRandomProjection(n_components=2)
```

```
projected_data = grp.fit_transform(data.iloc[:, :-1])  # Exclude the last column (Class Label)
plt.figure(figsize=(8, 6))
plt.scatter(projected_data[:, 0], projected_data[:, 1], c=data["Class Label"])
plt.xlabel("Projected Dimension 1")
plt.ylabel("Projected Dimension 2")
plt.title("Scatter Plot of Projected Data (using GRP)")
plt.show()
# c.
umap = UMAP(n_components=2)
projected_data = umap.fit_transform(data.iloc[:, :-1])  # Exclude the last column (Class Label)
# Plot a scatter plot of the projected data
plt.figure(figsize=(8, 6))
plt.scatter(projected_data[:, 0], projected_data[:, 1], c=data["Class Label"])
plt.xlabel("Projected Dimension 1")
plt.ylabel("Projected Dimension 2")
plt.title("Scatter Plot of Projected Data (using UMAP)")
plt.show()
"""
2. Unsupervised Learning: Determining the appropriate number of clusters
"""
# a.
feature_columns = "Class Label"
class_labels = data[feature_columns]
data = data.drop(feature_columns, axis=1)
scaler = StandardScaler()
normalized_data = scaler.fit_transform(data)
normalized_data = pd.DataFrame(normalized_data, columns=data.columns)
# b.
inertia_values = []
k_values = range(1, 11)  # Setting the maximum value of k as 10, but we can adjust it as needed

for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)  # Explicitly setting n_init to 10
    kmeans.fit(normalized_data)
    inertia_values.append(kmeans.inertia_)
    cluster_labels = kmeans.labels_

plt.plot(k_values, inertia_values, marker='o')
plt.xlabel('k')
```

```
plt.ylabel('Inertia')
plt.title('Inertia as a Function of k')
plt.show()
'''
3. Evaluating clustering
'''
class_labels = data["Class Label"]
kmeans = KMeans(n_clusters=len(class_labels.unique()),
    n_init=10, random_state=42)
kmeans.fit(normalized_data)
cluster_labels = kmeans.labels_

rand_index = adjusted_rand_score(class_labels,
   cluster_labels)
print("Rand index:", rand_index)

all_labels = pd.Series(cluster_labels).append(
   class_labels)
perms = itertools.permutations(range(len(all_labels.
   unique())))

best_accuracy = 0
for perm in perms:
    perm_labels = [perm[label] for label in
       cluster_labels]
    accuracy = accuracy_score(perm_labels,
       class_labels)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_permutation = perm

print("Best Accuracy:", best_accuracy)
print("Best Permutation:", best_permutation)
"""
4. Supervised Learning: K-nearest neighbour
"""
X = data.iloc[:, :-1]
X = X.values
y = data.iloc[:, -1]
y = y.values

X_train, X_test, y_train, y_test = train_test_split(X,
    y, test_size=0.2, random_state=42)

neighbors = np.arange(1, 15)
test_accuracy = np.empty(len(neighbors))
train_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
```

```
134     knn.fit(X_train, y_train)
135     test_accuracy[i] = knn.score(X_test, y_test)
136     train_accuracy[i] = knn.score(X_train, y_train)
137 
138 plt.plot(neighbors, test_accuracy, label = 'Testing␣
    dataset␣Accuracy')
139 plt.plot(neighbors, train_accuracy, label = 'Training␣
    dataset␣Accuracy')
140 plt.legend()
141 plt.xlabel('n_neighbors')
142 plt.ylabel('Accuracy')
143 plt.show()
```