DAT565/DIT407 Assignment 5

Ola Bratt
ola.bratt@gmail.com

Patrick Attimont
patrickattimont@gmail.com

2024-02-xx

This paper is addressing the assignment 3 study queries within the *Introduction to Data Science & AI* course, DIT407 at the University of Gothenburg and DAT565 at Chalmers. The main source of information for this project is derived from the lectures and Skiena [1]. Assignment 5 is about distance and network methods.

Problem 1: Preprocessing the dataset

Problem 2: Determining the appropriate number of clusters

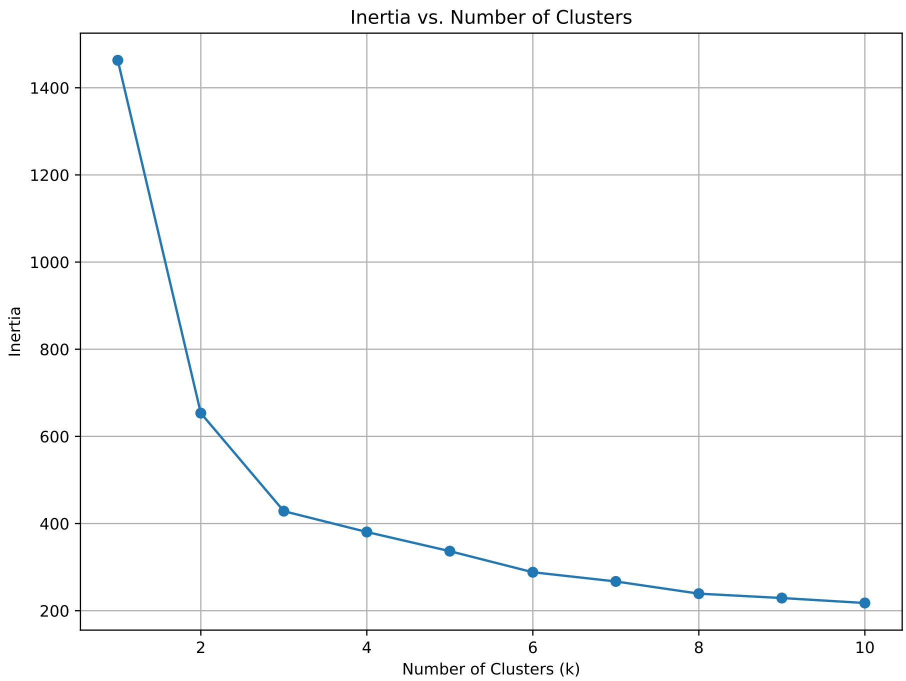

Figure 1: Invertia vs. Number of clusters

Problem 3: Visualizing the classes

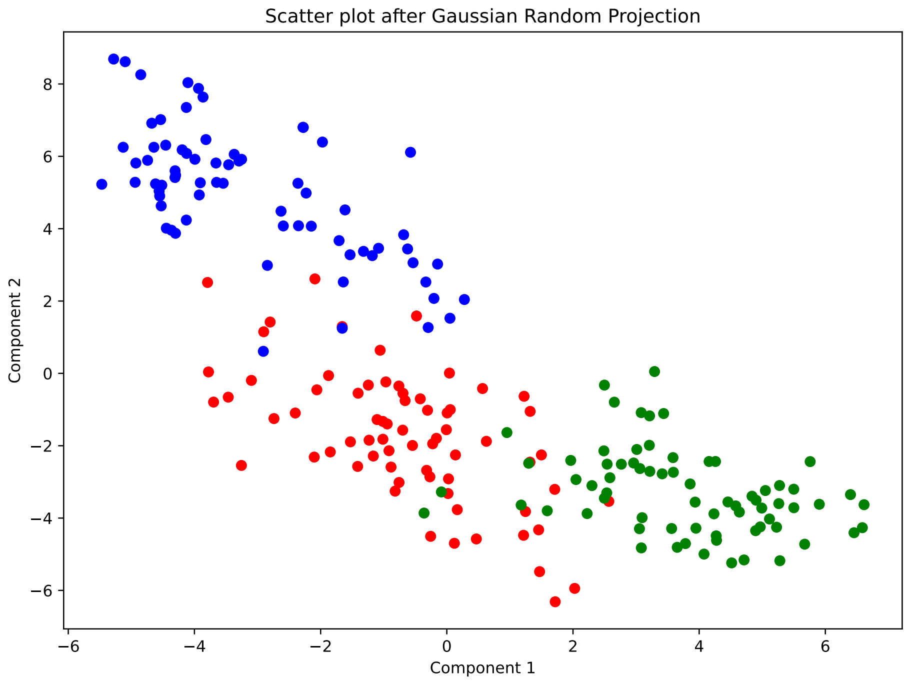

Figure 2: Gaussian random projection

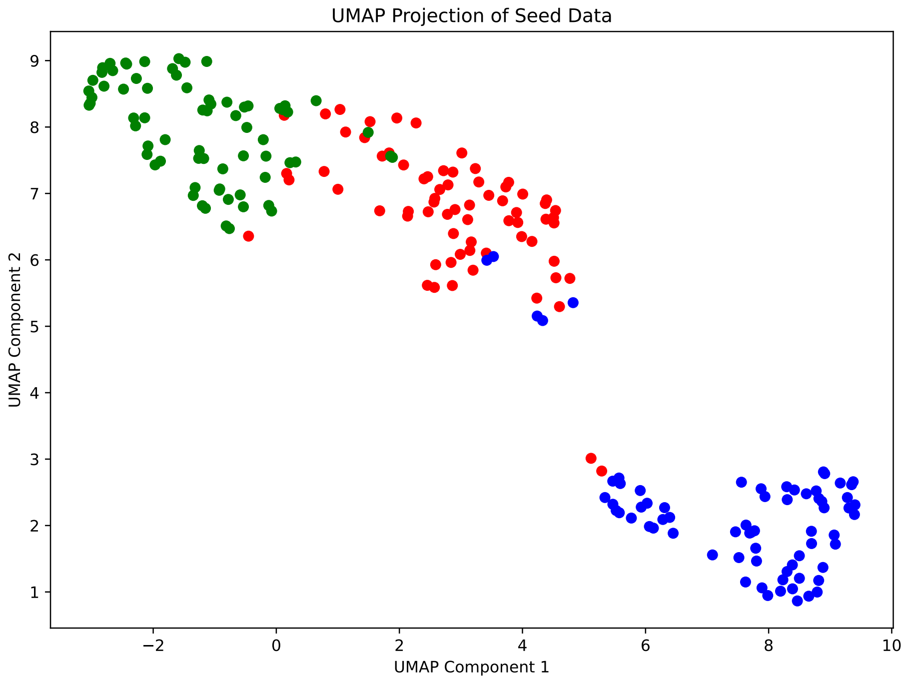

Figure 3: UMAP projection of Seeds

Problem 4: Evaluating clustering

Problem 5: Agglomerative clustering

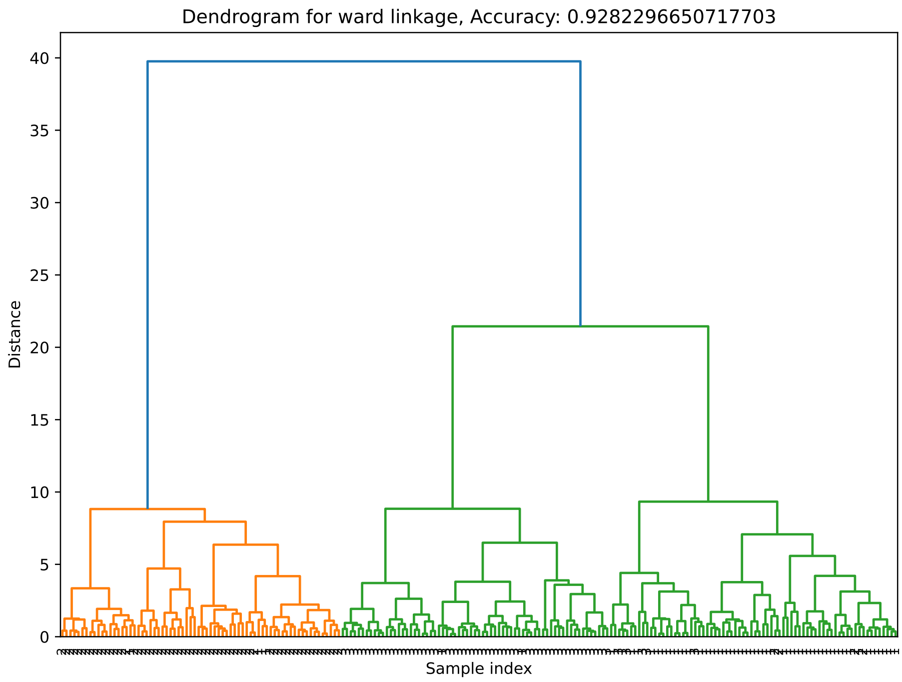

Figure 4: Dendrogram

References

[1] Steven S Skiena. *The Data Science Design Manual*. Retrieved 2024-01-20. 2024. URL: https://ebookcentral.proquest.com/lib/gu/detail.action?docID=6312797.

Appendix: Source Code

```python
from umap import UMAP
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.random_projection import GaussianRandomProjection
from sklearn.metrics import rand_score
import itertools
from sklearn.metrics import accuracy_score
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Load the seeds dataset
random_state = 79
seeds = pd.read_table('Assignment5/seeds.tsv')
seeds.columns = ['area', 'perimeter', 'compactness', 'length', 'width', 'asymmetry', 'groove', 'species']

X = seeds.drop(columns=['species'])  # Features
y = seeds['species']

# Normalize the data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

seeds_normalized = pd.DataFrame(X_normalized, columns=X.columns)
seeds_normalized['species'] = y

X = seeds_normalized.drop(columns=['species'])

def plot_inertia(X):
    inertia_values = []
    for k in range(1, 11):
        kmeans = KMeans(n_clusters=k, random_state=random_state).fit(X)
        inertia_values.append(kmeans.inertia_)

    plt.plot(range(1, 11), inertia_values, marker='o')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Inertia vs. Number of Clusters')
    plt.grid(True)
    plt.show()

def plot_features(features, y, colors):
    num_features = len(features)
    num_rows = num_features - 1
    num_cols = num_features - 1

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 15))

    for i in range(num_rows):
```

```python
        for j in range(num_cols):
            if i != j:
                ax = axes[i, j]
                ax.scatter(X[features[i]], X[features[j]], c=y.map(colors))
                ax.set_xlabel(features[i])
                ax.set_ylabel(features[j])
                ax.set_title(f'Scatter plot between {features[i]} and {features[j]}')

    plt.tight_layout()
    plt.show()

def plot_gaussian_random_projection(X, y, colors):
    grp = GaussianRandomProjection(n_components=2, random_state=random_state)
    projected = grp.fit_transform(X)

    plt.figure(figsize=(8, 6))
    plt.scatter(projected[:, 0], projected[:, 1], c=y.map(colors))
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.title('Scatter plot after Gaussian Random Projection')
    plt.show()

def plot_umap(X, y, colors):
    umap_model = UMAP(n_components=2)
    umap = umap_model.fit_transform(X)

    plt.figure(figsize=(8, 6))
    plt.scatter(umap[:, 0], umap[:, 1], c=y.map(colors))
    plt.xlabel('UMAP Component 1')
    plt.ylabel('UMAP Component 2')
    plt.title('UMAP Projection of Seed Data')
    plt.show()



def find_permutation(n_clusters, true_labels, cluster_labels):
    permutations = itertools.permutations(range(n_clusters))
    best_permutation = None
    best_accuracy = 0
    for permutation in permutations:
        permuted_labels = [permutation[label] for label in cluster_labels]
        accuracy = accuracy_score(permuted_labels, true_labels)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_permutation = permutation
    return best_permutation, best_accuracy


def plot_dendrogram(n_clusters, X, y):
    linkage_options = ['ward', 'complete', 'average', 'single']
    best_accuracy = 0
    best_linkage = None

    for linkage_option in linkage_options:
        clustering = AgglomerativeClustering(n_clusters=len(y.unique()), linkage=linkage_option)
        cluster = clustering.fit(X)
        permutation, accuracy = find_permutation(n_clusters, y,
```

```
cluster.labels_)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_linkage = linkage_option

    Z = linkage(X, method=best_linkage)
    plt.figure(figsize=(12, 6))
    dendrogram(Z, labels=y.values, leaf_rotation=90, leaf_font_size=8)
    plt.title(f"Dendrogram for {best_linkage} linkage, Accuracy: {best_accuracy}")
    plt.xlabel("Sample index")
    plt.ylabel("Distance")
    plt.show()

plot_inertia(X)
colors = {1: 'red', 2: 'blue', 3: 'green'}
features = seeds_normalized.columns
plot_features(features, y, colors)
plot_gaussian_random_projection(X, y, colors)
plot_umap(X, y, colors)


kmeans = KMeans(n_clusters=len(y.unique()), random_state=random_state)
kmeans.fit(X)
kmeans_labels = kmeans.labels_

rand_index = rand_score(y, kmeans_labels)
print("Rand score:", rand_index)

all_labels = pd.Series(kmeans_labels)._append(y)
all_unique_labels = all_labels.unique()

best_permutation, best_accuracy = find_permutation(len(all_unique_labels), y, kmeans_labels)

print("Best Accuracy:", best_accuracy)
print("Best Permutation:", best_permutation)

plot_dendrogram(len(all_unique_labels), X, y)
```