Solution

April 29, 2024

[ ]:
```
import os
import os
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import (
    DBSCAN,
    AgglomerativeClustering,
    BisectingKMeans,
    KMeans,
    SpectralClustering,
)
from sklearn.decomposition import PCA
from sklearn.metrics import fowlkes_mallows_score, silhouette_score
from sklearn.preprocessing import StandardScaler
```

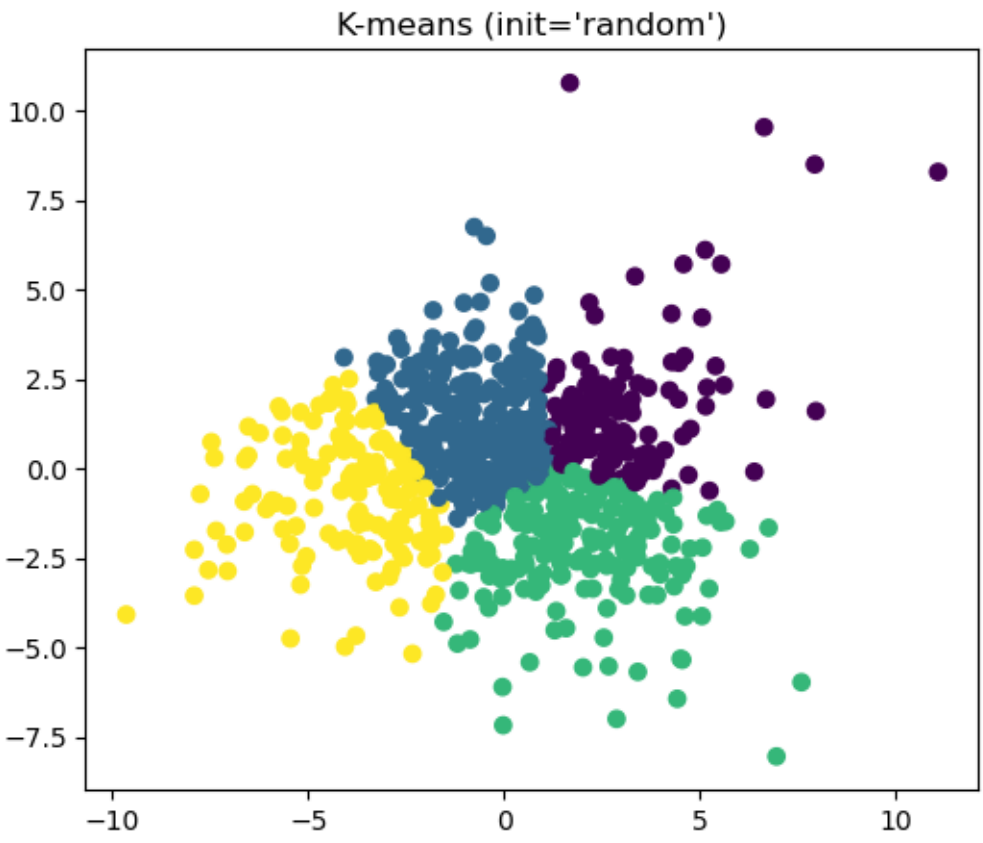
[ ]:
```
def load_histograms_from_directory(directory_path):
    histograms = []
    labels = []
    label_index_mapping = {}
    current_label_index = 0

    for class_name in os.listdir(directory_path):
        class_path = os.path.join(directory_path, class_name)
        if class_name not in label_index_mapping:
            label_index_mapping[class_name] = current_label_index
            current_label_index += 1

        class_histograms, class_labels = load_histograms_from_class(class_path,␣
 ↪class_name, label_index_mapping)
        histograms.extend(class_histograms)
        labels.extend(class_labels)

    return histograms, labels
```

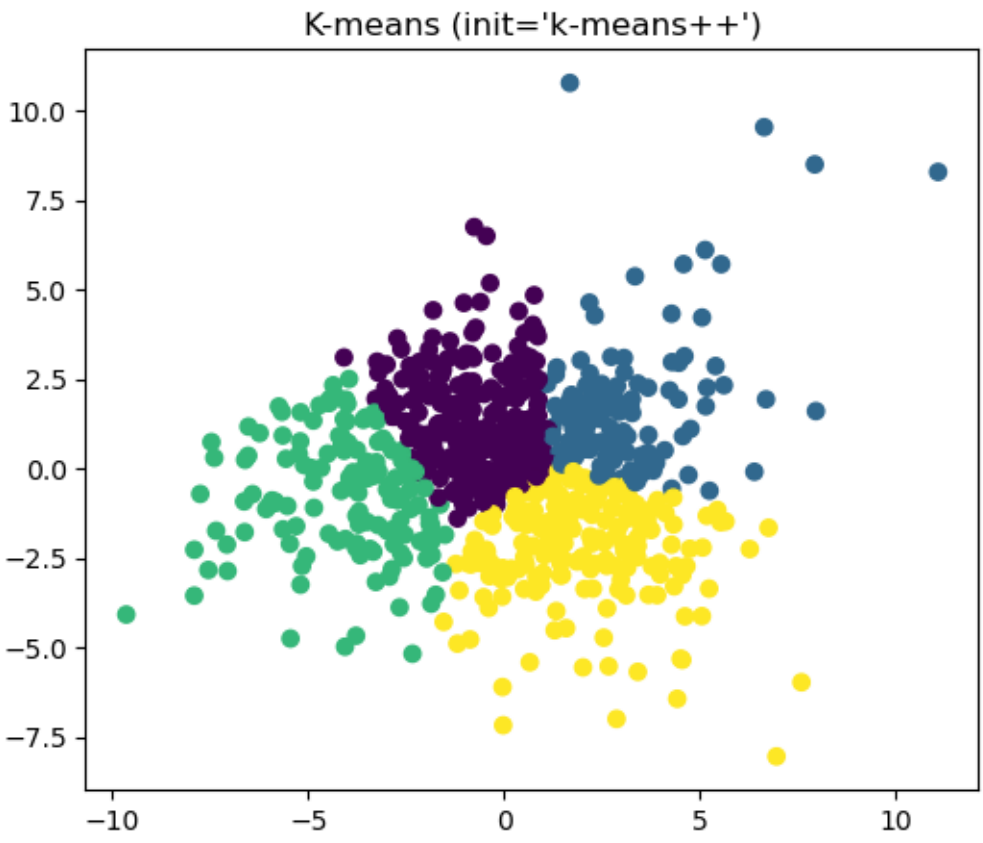
```
def load_histograms_from_class(class_path, class_name, label_index_mapping):
    histograms = []
    labels = []

    for file_name in os.listdir(class_path):
        if file_name.endswith(".npy"):
            histogram_path = os.path.join(class_path, file_name)
            histogram = np.load(histogram_path)
            histograms.append(histogram)
            labels.append(label_index_mapping[class_name])

    return histograms, labels

def normalize_histogram_data(histograms, labels):
    scaler = StandardScaler()
    normalized_histograms = scaler.fit_transform(histograms)
    labels = np.array(labels)
    return normalized_histograms, labels

def reduce_histogram_dimensions(histograms, n_components=2):
    pca = PCA(n_components=n_components)
    reduced_data = pca.fit_transform(histograms)
    return reduced_data

def plot_histogram_data(reduced_data, labels):
    plt.figure(figsize=(10, 8))
    unique_labels = np.unique(labels)
    for label in unique_labels:
        plt.scatter(
            reduced_data[labels == label, 0],
            reduced_data[labels == label, 1],
            label=f"Class {label}"
        )
    plt.title("PCA Reduced Histograms")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.legend()
    plt.grid(True)
    plt.show()


target_directory = "EdgeHistograms"
histograms, labels = load_histograms_from_directory(target_directory)
normalized_histograms, labels = normalize_histogram_data(histograms, labels)
reduced_histograms = reduce_histogram_dimensions(normalized_histograms)
plot_histogram_data(reduced_histograms, labels)
```

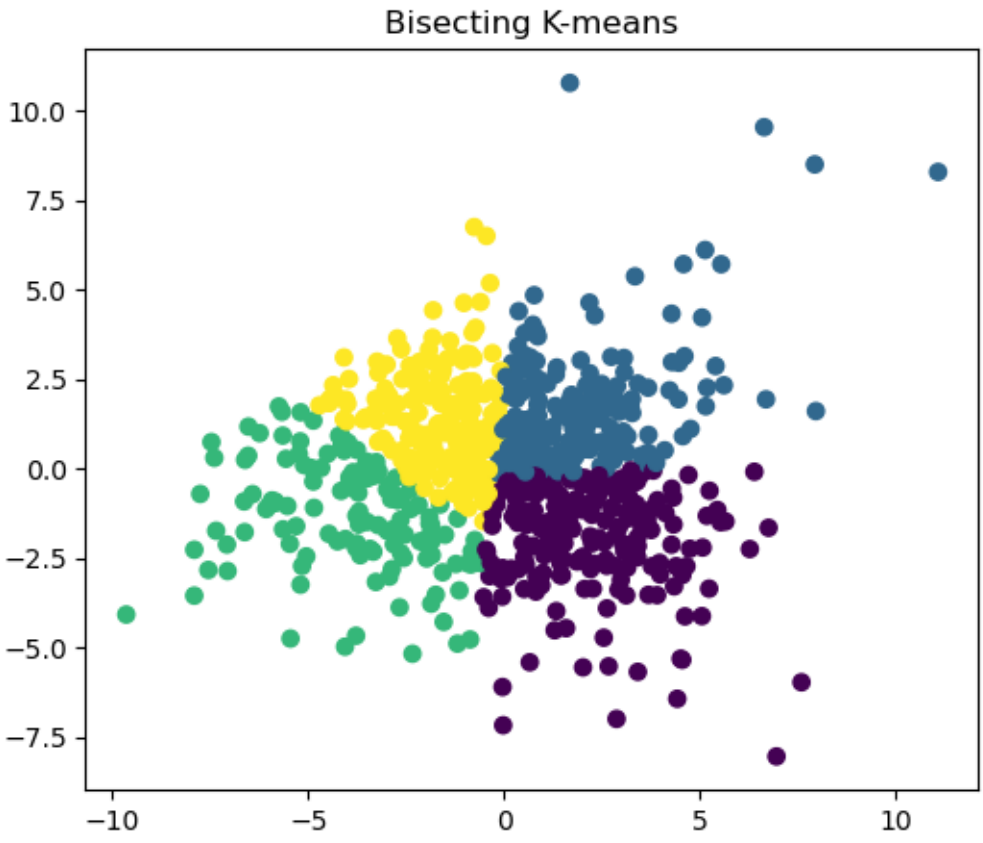
[ ]:
```
def display_cluster_parameters(eps, min_samples, n_clusters):
    print(f"Eps: {eps}")
    print(f"Min samples: {min_samples}")
    print(f"Number of clusters: {n_clusters}")

def perform_kmeans_clustering(data, n_clusters=4, init_method="random",␣
 ↪random_seed=42):
    kmeans = KMeans(
        n_clusters=n_clusters,
        init=init_method,
        n_init=10,
        random_state=random_seed
    )
    return kmeans.fit_predict(data)

def perform_bisecting_kmeans_clustering(data, n_clusters=4,␣
 ↪init_method="random", random_seed=42):
    bisecting_kmeans = BisectingKMeans(
        n_clusters=n_clusters,
        init=init_method,
        n_init=10,
        random_state=random_seed
    )
    return bisecting_kmeans.fit_predict(data)

def perform_spectral_clustering(data, n_clusters=4, random_seed=42):
    spectral = SpectralClustering(
        n_clusters=n_clusters,
        random_state=random_seed
    )
    return spectral.fit_predict(data)

def perform_dbscan_clustering(data, eps, min_samples):
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    return dbscan.fit_predict(data)

def plot_clustered_data(data, labels, title):
    plt.figure(figsize=(6, 5))
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
    plt.title(title)
    plt.show()


eps_value = 2.9
min_samples_value = 2
n_clusters = 4
```

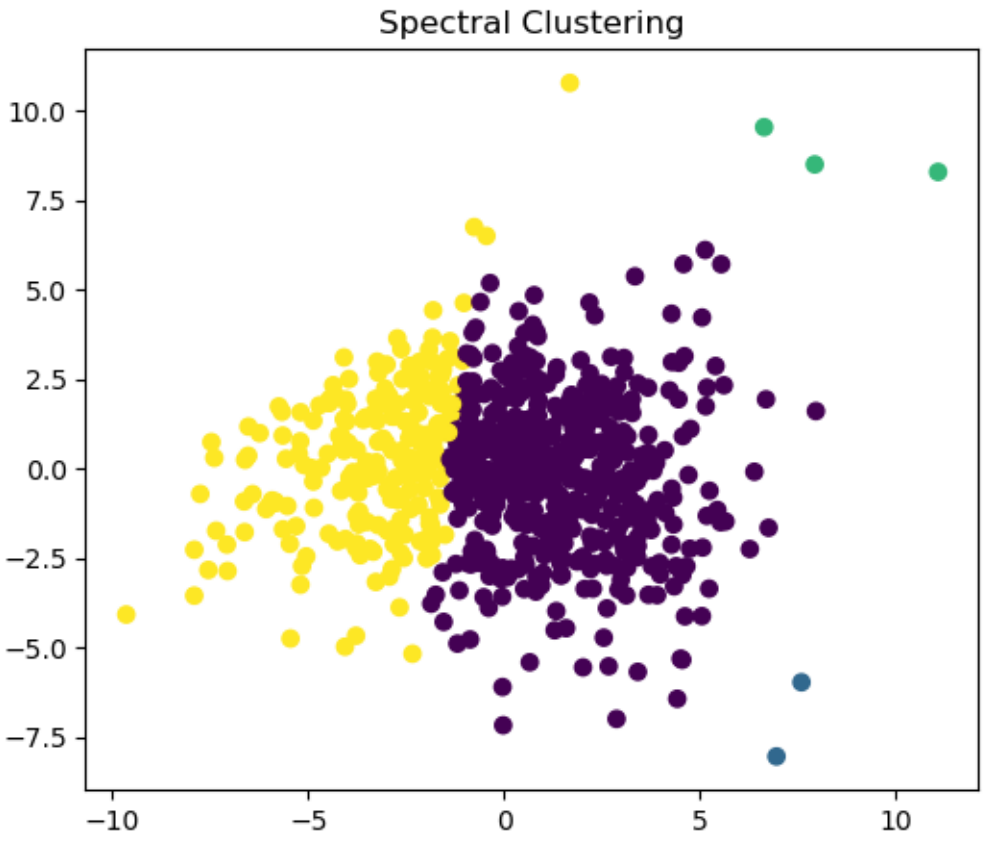
```
display_cluster_parameters(eps_value, min_samples_value, n_clusters)

kmeans_labels_random = perform_kmeans_clustering(reduced_histograms,␣
 ↪init_method="random")
plot_clustered_data(reduced_histograms, kmeans_labels_random, "K-means␣
 ↪(init='random')")

kmeans_labels_plus = perform_kmeans_clustering(reduced_histograms,␣
 ↪init_method="k-means++")
plot_clustered_data(reduced_histograms, kmeans_labels_plus, "K-means␣
 ↪(init='k-means++')")

bisecting_kmeans_labels =␣
 ↪perform_bisecting_kmeans_clustering(reduced_histograms)
plot_clustered_data(reduced_histograms, bisecting_kmeans_labels, "Bisecting␣
 ↪K-means")

spectral_labels = perform_spectral_clustering(reduced_histograms)
plot_clustered_data(reduced_histograms, spectral_labels, "Spectral Clustering")

dbscan_labels = perform_dbscan_clustering(reduced_histograms, eps_value,␣
 ↪min_samples_value)
plot_clustered_data(reduced_histograms, dbscan_labels, "DBSCAN")
```

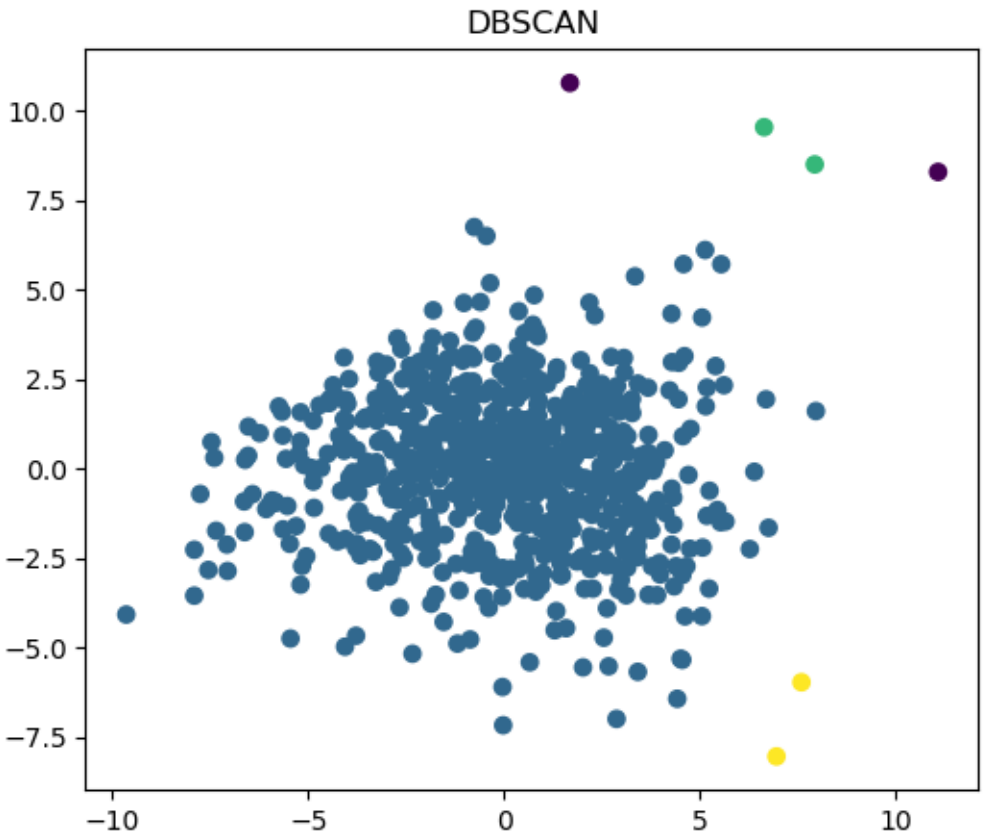
```
Eps: 2.9
Min samples: 2
Number of clusters: 4
```


K-means (init='random')
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10


K-means (init='k-means++')
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10

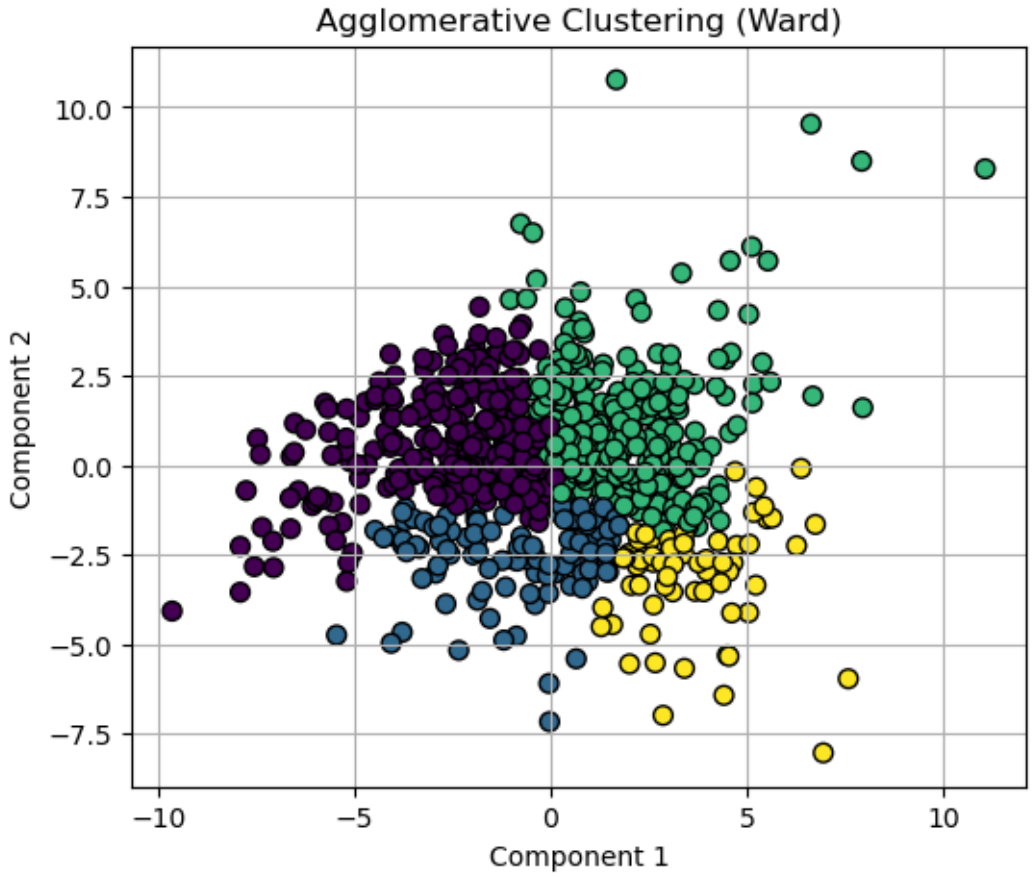

Bisecting K-means
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10

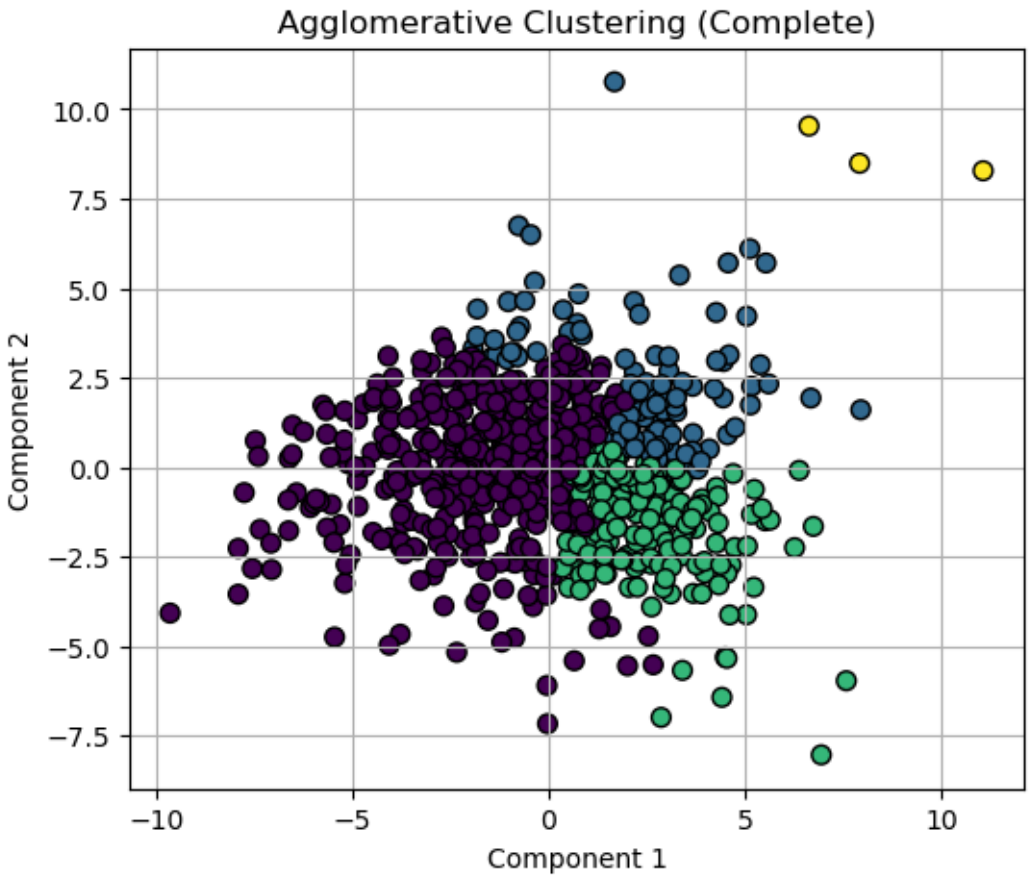

Spectral Clustering
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10

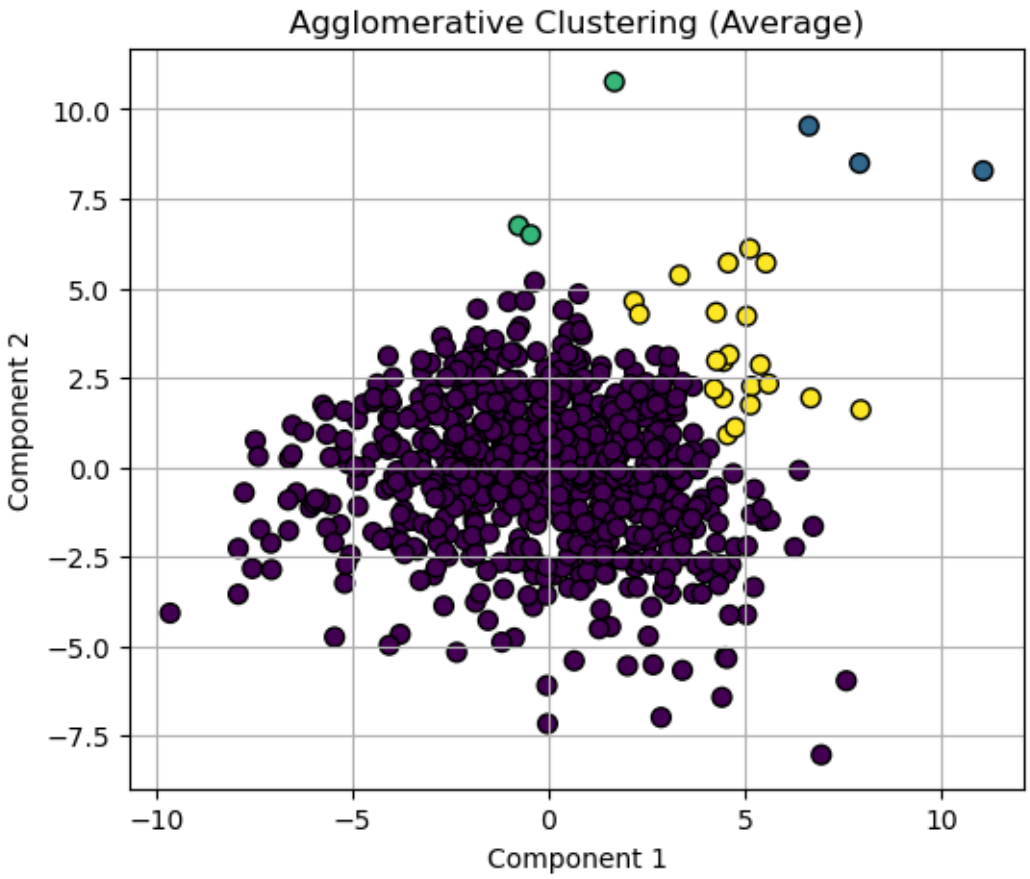


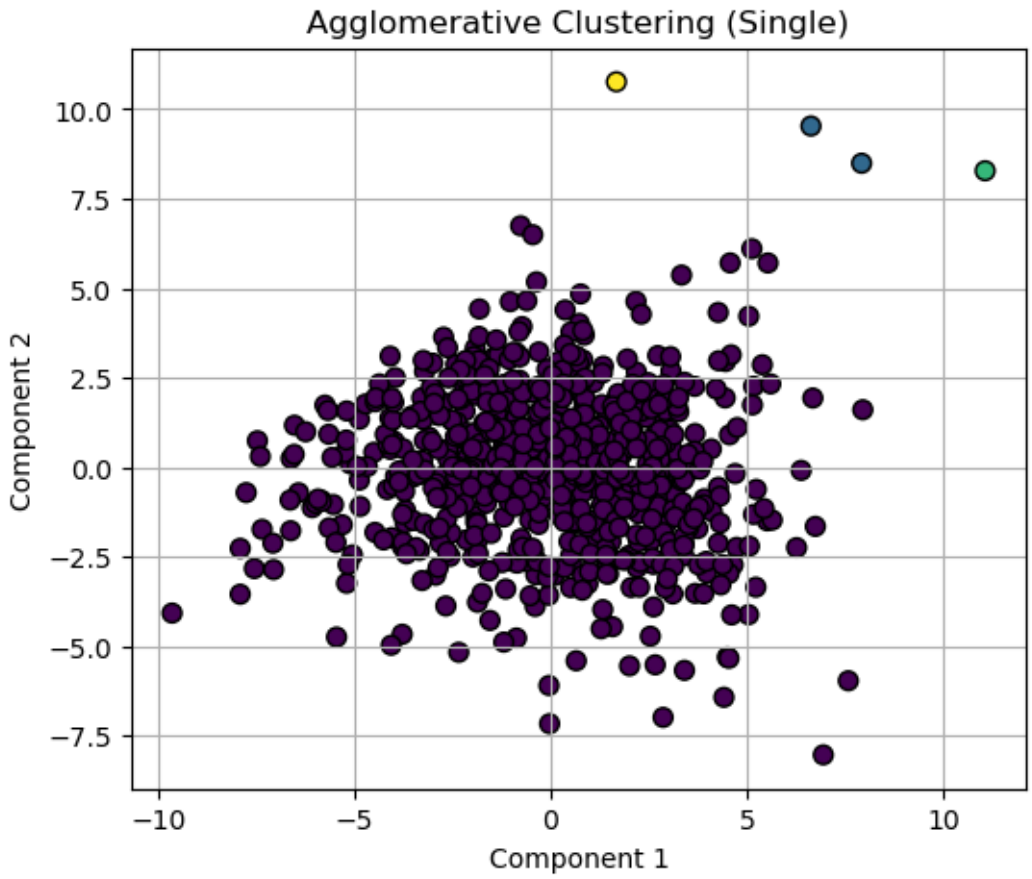
[ ]:
```
def perform_agglomerative_clustering(data, n_clusters=4):
    linkage_methods = ["ward", "complete", "average", "single"]
    for method in linkage_methods:
        plot_agglomerative_clustering(data, n_clusters, method)

def plot_agglomerative_clustering(data, n_clusters, linkage_method):
    clustering = AgglomerativeClustering(n_clusters=n_clusters,␣
 ↪linkage=linkage_method)
    labels = clustering.fit_predict(data)

    plt.figure(figsize=(6, 5))
    plt.scatter(
        data[:, 0],
        data[:, 1],
        c=labels,
        cmap="viridis",
        edgecolor="k",
        s=50
    )
    plt.title(f"Agglomerative Clustering ({linkage_method.capitalize()})")
```

```
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.grid(True)
    plt.show()

perform_agglomerative_clustering(reduced_histograms)
```




Agglomerative Clustering (Complete)
Component 2
Component 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10


Agglomerative Clustering (Average)
Component 2
Component 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10



[ ]:
```
def evaluate_clustering_methods(data, true_labels, eps_value,␣
↪min_samples_values):
   methods = {
        "K-Means (Random)": KMeans(n_clusters=4, init="random", random_state=0,␣
↪n_init=10),
        "K-Means (k-means++)": KMeans(n_clusters=4, init="k-means++",␣
↪random_state=0, n_init=10),
        "Bisecting K-Means": BisectingKMeans(n_clusters=4, random_state=0),
        "Spectral Clustering": SpectralClustering(n_clusters=4, random_state=0),
        "DBSCAN": DBSCAN(eps=eps_value, min_samples=min_samples_values),
        "Agglomerative (Ward)": AgglomerativeClustering(n_clusters=4,␣
↪linkage="ward"),
        "Agglomerative (Complete)": AgglomerativeClustering(n_clusters=4,␣
↪linkage="complete"),
        "Agglomerative (Average)": AgglomerativeClustering(n_clusters=4,␣
↪linkage="average"),
        "Agglomerative (Single)": AgglomerativeClustering(n_clusters=4,␣
↪linkage="single")
```

```
    }

    scores_fm = {}
    scores_silhouette = {}

    for name, model in methods.items():
        labels = model.fit_predict(data)
        if len(set(labels)) > 1:
            fm_score = fowlkes_mallows_score(true_labels, labels)
            silhouette_score_value = silhouette_score(data, labels)
            scores_fm[name] = fm_score
            scores_silhouette[name] = silhouette_score_value
        else:
            scores_fm[name] = None
            scores_silhouette[name] = None

    return scores_fm, scores_silhouette

def print_scores(scores, score_type):
    sorted_scores = sorted(scores.items(), key=lambda item: (item[1] is not␣
 ↪None, item[1]), reverse=True)
    print(f"\n{score_type} (Ranked):\n")
    print(f"{'Rank':<5} {'Method':<30} {'Score':<10}")
    print("-" * 45)
    for index, (method, score) in enumerate(sorted_scores, 1):
        score_display = f"{score:.5f}" if score is not None else "Undefined"
        print(f"{index:<5} {method:<30} {score_display:<10}")

eps_value = 2.9
min_samples_value = 2
scores_fm, scores_silhouette = evaluate_clustering_methods(reduced_histograms,␣
 ↪labels, eps_value, min_samples_value)

print_scores(scores_fm, "Fowlkes-Mallows Index")
print_scores(scores_silhouette, "Silhouette Coefficient")
```

```
Fowlkes-Mallows Index (Ranked):

Rank  Method                         Score
---------------------------------------------
1     Agglomerative (Single)         0.49829
2     DBSCAN                         0.49686
3     Agglomerative (Average)        0.48243
4     Spectral Clustering            0.37305
5     Agglomerative (Complete)       0.34141
6     Agglomerative (Ward)           0.29795
```

```
7     K-Means (Random)              0.27600
8     K-Means (k-means++)           0.27600
9     Bisecting K-Means             0.27129

Silhouette Coefficient (Ranked):

Rank  Method                        Score
-------------------------------------------
1     Agglomerative (Single)        0.58024
2     DBSCAN                        0.51570
3     Spectral Clustering           0.34368
4     K-Means (Random)              0.33140
5     K-Means (k-means++)           0.33140
6     Agglomerative (Average)       0.30693
7     Bisecting K-Means             0.30552
8     Agglomerative (Ward)          0.27935
9     Agglomerative (Complete)      0.26110
```