Fast and Simple Spectral Clustering in Theory and Practice Team Project

Submitted in partial fulfilment of the requirements for the course of

IE 529 Stats of Big Data and Clustering

By:

Professor Carolyn L. Beck

Submitted by:

Prannoy Kathiresan (prannoy2)
and
Selvamani Subramaniam (ss170)

UNIVERISTY OF ILLINOIS URBANA-CHAMPAIGN

MAY 2024

Abstract:

In this project, we have implemented the classical spectral clustering algorithm and the fast & simple spectral clustering algorithm proposed by Macgregor et al. (2023) [1]. Since we require a large dataset, we have chosen HAR (Human Activity Recognition) dataset to perform our analysis. This dataset was fetched from OpenML using scikit-learn library. We ran FASTSPECTRALCLUSTER, CLASSICSPECTRALCLUSTER and computed ARI (Adjusted Rand Index), NMI (Normalized Mutual Information) between actual labels and predicted labels to determine the clustering accuracy. This showed that, for the chosen dataset, FASTSPECTRALCLUSTER algorithm is simple, fast, and effective when compared to CLASSICSPECTRALCLUSTER algorithm with a small trade-off in clustering accuracy.

TABLE OF CONTENTS

1. INTRODUCTION:

1.1 Problem Statement:

Spectral clustering is a popular algorithm for graph clustering, but it can be computationally expensive due to the need to compute eigenvectors of the graph Laplacian matrix.

High-level overview of classic Spectral clustering is as follows:

1. Compute Laplacian matrix to get k eigenvectors and embed the vertices of G into $\mathbb{R}^k$.
2. Apply - means clustering to partition into -clusters

Note that, computing k -eigenvectors for exceptionally large dataset is computationally time-consuming.

1.2 Objective:

The goal is to develop a new spectral clustering algorithm that is fast, effective, and avoids the high computational cost of computing eigenvectors while still achieving excellent clustering performance.

1.3 Solution:

The paper discusses an approach of efficiently approximating a random projection of spectral embedding (using Power method) without computing the spectral embedding itself and thereby showing that projection using Power method is equivalent to projection using Spectral clustering.

High-level overview of the solution is as follows:

1. Using Power method, compute $O(\log(k))$ random vectors.
2. Embed vertices of G into $O(\log(k))$ dimensions.
3. Apply k - means clustering to partition into k -clusters.

2. TECHNICAL BACKGROUND:

Following provides definitions and concepts related to graph theory, particularly focusing on Laplacian matrices, eigenvalues, and partitions of a graph.

1. **Graph Description**: The graph $G = (V, E, w)$ consists of vertices V , edges E , and edge weights W . The degree of a vertex W is the sum of weights of its incident edges $(i.e\ d(v) = \sum_{u \neq v}^{w}(u, v))$. The volume of a subset $S(\subset V)$ of vertices is the sum of degrees of vertices in $S\ \left(i.e\ vol(S) = \sum_{u \in S}^{d}(u)\right)$
2. **Laplacian Matrices**: The Laplacian matrix L of G is computed as $L = D - A$ where D is a diagonal matrix $d(i)$ representing vertex degrees and A is the adjacency matrix. The normalized Laplacian N is derived from $N = D^{\frac{-1}{2}} L\, D^{\frac{-1}{2}}$.
3. **Eigenvalues and Eigenvectors** $\lambda_1 \leq \lambda_2 \leq \lambda_3 \ldots . \leq \lambda_n$ represents the eigenvalues of N , ordered non-decreasingly, with corresponding eigenvectors $f_1, f_2, \ldots . . , f_n$. The signless Laplacian matrix M is defined as $M = I - \left(\frac{1}{2}N\right)$ and $\gamma_1 \geq \gamma_2 \geq \gamma_3 \geq \cdots . \geq \gamma_n$ represents the eigenvalues of M .
4. **Sets and Operations**: Let $[k] = \{1, \ldots . , k\}$ denotes the set of positive integers up to k . The symmetric difference of sets A and B is denoted as $A \Delta B = (A \backslash B) \cup (B \backslash A)$. A k - way partition of V divides V into k non-overlapping subsets.
5. **Notations**: Big-O notation is used to indicate asymptotic behaviour, hiding constants. For example, $l = O(n)$ means l is bounded by a constant times n . $O^{\sim}(n)$ represents $O(n\ \log^{c}(n))$ for some constant c .
6. **Partition Balance**: A partition of V is considered almost-balanced if each subset's volume is proportional to the total volume of V divided by the number of subsets. $\left(i.e\ vol(S_i) = \Theta\left(\frac{vol(v)}{k}\right)\ \forall\ i\ \in\ [k]\right)$
7. **Conductance**: For a graph $G = (V, E)$ and a cluster $S \subset V$, the conductance of S measures its connectivity to the rest of the graph. It's calculated as the ratio of the total weight of edges leaving S to the minimum of the volume of S and $\overline{S}$. $\left(i.e\ \Phi(S) = \frac{w(S,\overline{S})}{\min\{vol(S), vol(\overline{S})\}}\right)$ *where* $w\left(S, \overline{S}\right)$ represents the sum of edge weights between vertices in S .
8. **k-Way Expansion**: The k-way expansion of a graph G is defined as the minimum conductance over all partitions into k subsets. $\left(i.e\ \rho(k) = \min_{C_1, \ldots, C_k} \max_{i} \Phi(C_i)\right)$ A small value of $\rho(k)$ implies that G can be partitioned into k clusters with low conductance.

By invoking Lemma 2.1 (Higher-Order Cheeger Inequality), we can see that G can be partitioned into k -clusters of low conductance and not partitioned into $k+1$ -clusters.

Lemma 2.1 (Higher-Order Cheeger Inequality) -> $\frac{\lambda_k}{2} \leq \rho(k) \leq O(k^3)\sqrt{\lambda_k}$

3. SUMMARY OF MAIN RESULT:

3.1 Problem solution:

We know that in classic spectral clustering, we compute weighted adjacency matrix (W) using any one of similarity graph methods (such as ϵ -neighbourhood graph, k -nearest neighbour graph, fully connected graph). We then compute the Laplacian matrix (either normalized or unnormalized) followed by generation of first k -eigenvectors (k refers to # of clusters). We then k -means algorithm (preferably k -means++ Lloyd's algorithm) to predict the clusters associated with the data.

This is computationally expensive for large datasets, henceforth, the author proposed a new algorithm called FASTSPECTRALCLUSTER which embeds the vertices of the data onto $O(\log(k))$ dimensions instead of $O(k)$ dimensions (for classic spectral algorithm).

FASTSPECTRALCLUSTER uses Power Method to compute the dominant eigenvalues of a matrix which in turn is used to embed the data onto $O(\log(k))$-dimensions as evident from Golub et al. (2000) [2] and Muntz et al. (1913) [3]. It performs repeated multiplication of a vector with the given matrix to obtain the dominant eigenvalue. After projecting the data onto $O(\log(k))$- dimensions, we invoke in-built k -means clustering algorithm to compute the k -clusters.

3.2 Key steps taken:

Two important key steps are taken to solve the problem and those are k -means objective and Power method. Let us first discuss these separately and then move onto the deployment of the algorithm.

3.2.1 k -means Objective:

$$COST_B(A_1, \ldots, A_k) \triangleq \sum_{i=1}^{k} \sum_{u \in A_i} \| B(u,:) - \mu_i \|_2^2$$

where,

$B\ (u,:)$ - u*th* row of B

$$\mu_i = \left(\frac{1}{|A_i|}\right) \sum_{u \in A_i}^{B} (u,:)$$

We know that k -means objective is NP hard we have number of other ways to solve such as polynomial-time approximation algorithm as evident from Kanungo et al. (2002) [4], polynomial approximation scheme in n and d as evident from Feldman et al. (2007) [5] and Kumar et al(2004) [6], $O(\log(n))$-approximation algorithm through k -means++ initialization in Llyod's algorithm as evident from Lloyd (1982) [7].

3.2.2 Power method:

As mentioned in the problem solution, power method performs repeated multiplication of a vector with the given matrix to obtain the dominant eigenvalue. We compute two variables *l*,*t* that are essential to this algorithm which is mentioned in the pseudocode below.

3.3 Pseudocode of main algorithm:

Pseudo code for POWERMETHOD	
	Input: M $\in \mathbb{R}^{n \times n}$, $x_0 \in \mathbb{R}^n$, $t \in \mathbb{Z}_{\geq 0}$
	Output: $x_t \in \mathbb{R}^{n \times n}$
1	**for** $i \in \{1, \ldots, t\}$ **do** *//Run the loop for t times*
2	$x_i = \mathrm{M}\, x_{i-1}$ *// Perform the multiplication*
3	**end**
4	**return** x_t *// Return the result after t-many multiplications*

Pseudo code for FASTSPECTRALCLUSTER method	
	Input: $G = (V, E)$, $k \in \mathbb{Z}_{\geq 0}$
	Output: $A_1, \ldots\ldots, A_k$
1	**M** $\leftarrow I - \left(\frac{1}{2}\right).N_G$ *//Compute signless Laplacian matrix M*
2	l $\leftarrow \Theta(\log(k).\epsilon^{-2})$ *//Compute l for which data projects onto O(log(k)) dimensions*

3	t ⟵ $\Theta\left(\log\left(\frac{n}{\epsilon^2 k}\right)\right)$ *//Compute t which decides the # of iterations for Power method*
4	**for** $i \in \{1,....,l\}$ **do**
5	Let $x_i \in \mathbb{R}^n$ *//random vector from Gaussian distribution N(0,1)*
6	y_i ⟵ POWERMETHOD(M, x_i, t) *// perform Powermethod*
7	**end**
8	**Y** ⟵ $[y_1; \ldots; y_l]$
9	$A_1, \ldots., A_k$ ⟵ KMEANS $\left(D_G^{-\frac{1}{2}} Y, k\right)$ *//perform k-means clustering*
10	**return** $A_1, \ldots., A_k$ *//return the labels*

4. APPLICATION EXAMPLE:

4.1 Data:

We fetched the HAR dataset from OpenML using *scikit-learn* library. We then pre-processed the dataset by computing k-nearest neighbour graph from the data, where k is set to 10. For computing k-nearest neighbour, we used two different methods for mode as 'continuity' and 'distance'. Then the computed adjacency matrix is converted into symmetric graph with vertices and edges based on the number of datapoints and k-nearest neighbours.

For our understanding, we also used the Llyod's algorithm code from CompAssignment-2 for k - means clustering.

4.2 Statistics of application results:

4.2.1 Indicators:

We used the following statistics to assess the performance of the author's algorithm by comparing it with CLASSICSPECTRALCLUSTER algorithm:

1. **NMI (Normalized Mutual Information):**

- NMI measures the similarity between two clustering, considering both the mutual information and the normalization factors.
- It quantifies the amount of information obtained from one clustering about another clustering, normalized by the entropy of the two clustering.
- NMI ranges from 0 to 1, where 0 indicates no mutual information and 1 indicates perfect agreement between the two clustering.

2. **ARI (Adjusted Rand Index):**

- ARI is a measure of the similarity between two clustering, adjusting for chance agreement.
- It computes the proportion of agreements between two clustering while considering the expected agreement by chance.
- ARI ranges from -1 to 1, where a value close to 0 indicates random agreement, a value close to 1 indicates perfect agreement, and negative values suggest disagreement worse than random chance.

3. **Running time of the algorithm.**

4.2.2 Plots:

Since the dataset consist of 10299 rows and 561 columns, PCA was performed on the dataset to identify two principal components. Then the target labels from the original dataset and the clustered assignments from the FASTSIMPLESPECTRALCLUSTER algorithm and CLASSICSPECTRALCLUSTER algorithm was projected onto the principal components for plotting.

HAR DATASET with the target labels and their clusters:

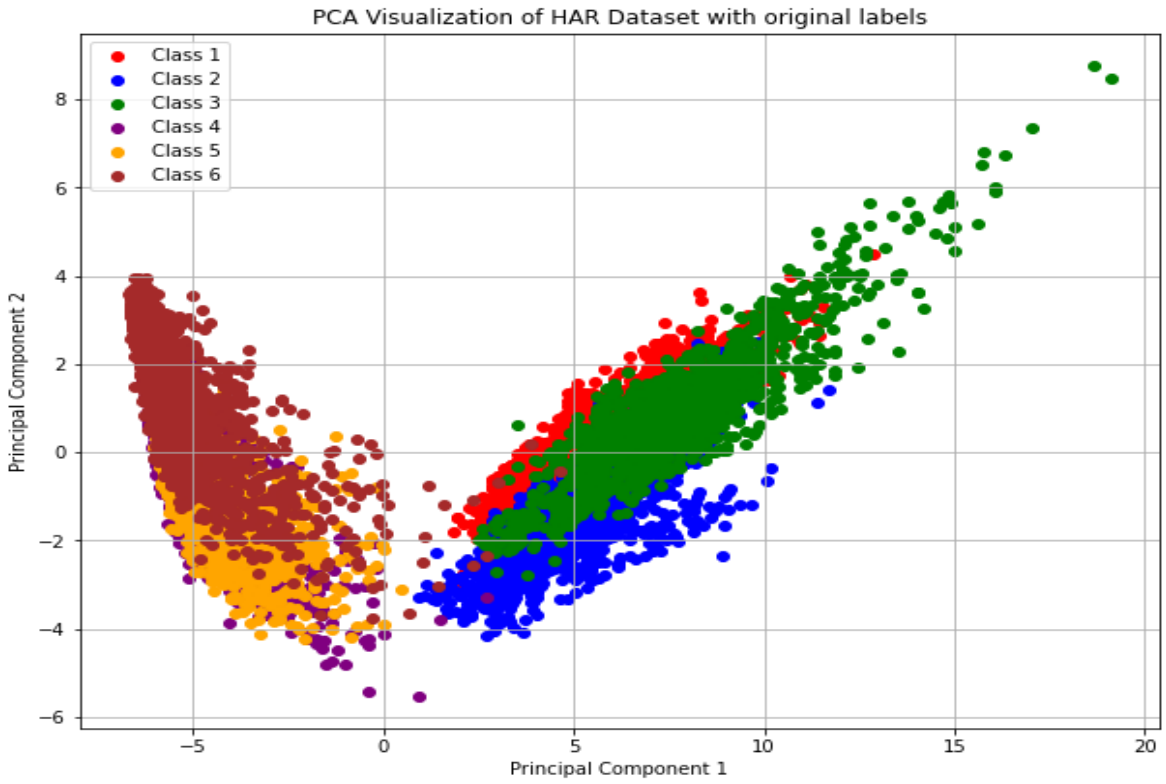


Fig.1

PCA Visualization of HAR Dataset with Clustered labels through Fast and Simple Spectral clustering with 'Connectivity' mode in k-nearest neighbour:

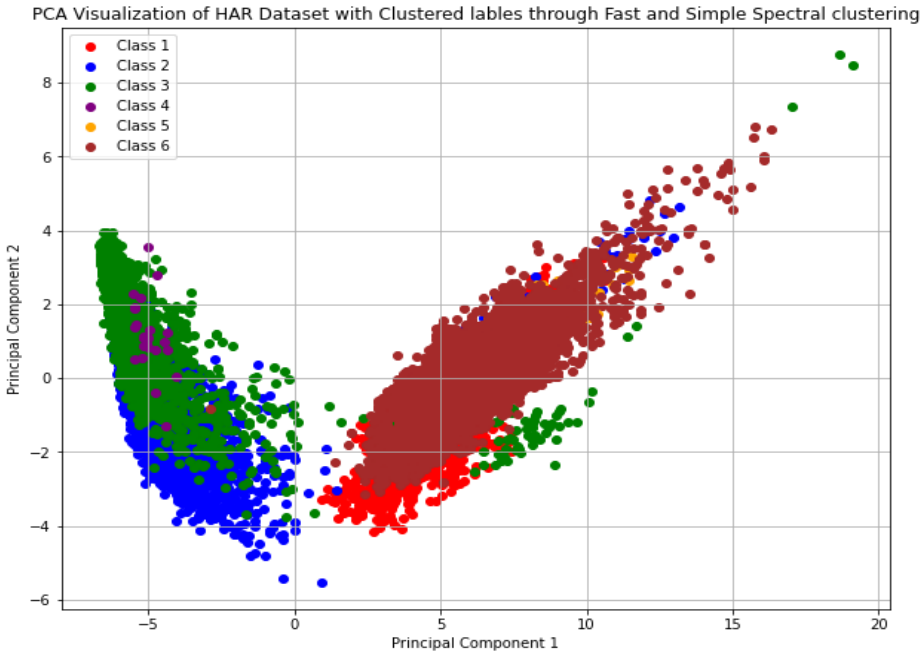


Fig 2.

PCA Visualization of HAR Dataset with Clustered labels through Fast and Simple Spectral clustering with 'Distance' mode in k-nearest neighbour:

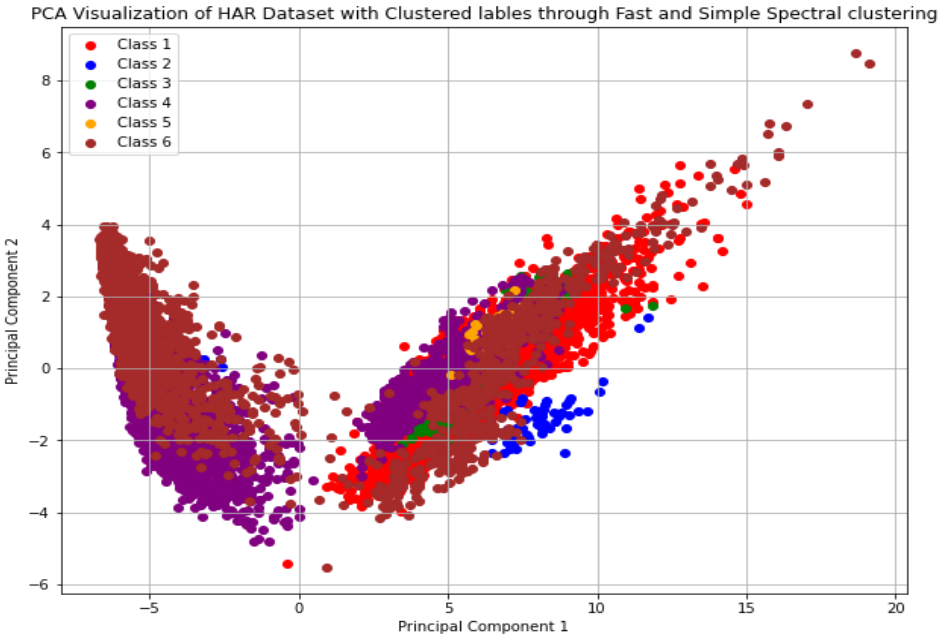


Fig 3.

Classical Spectral Clustering using normalized Laplacian matrix with ‘Connectivity’ mode in k-nearest neighbour:

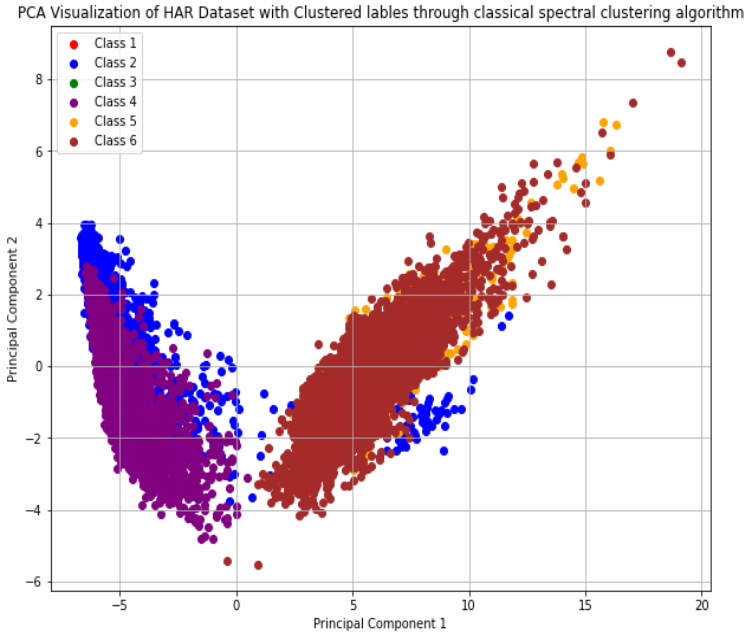


Fig 4.

Classical Spectral Clustering using normalized Laplacian matrix with ‘distance’ mode in k-nearest neighbour:

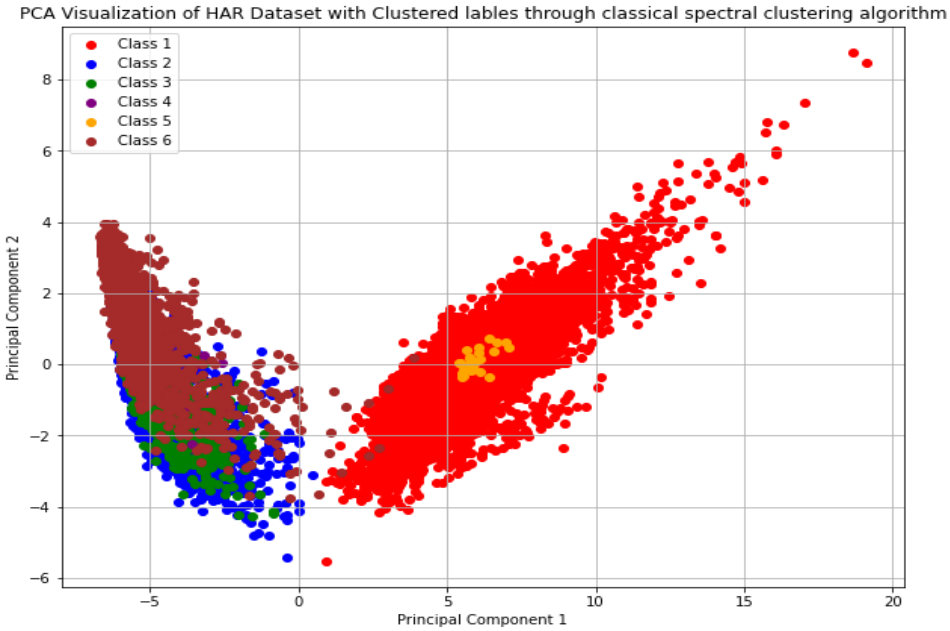


Fig 5.

NRI, AMI and the Running time parameters is computed by running the algorithms 10 times for each of those mentioned above and the average of the values is taken.

```
Results for main_program_fastc:
Average Total Time: 16.1241436958313
Average ARI: 0.36230508644687465
Average NMI: 0.4960793898930381
---------
Results for main_program_fastd:
Average Total Time: 16.062884974479676
Average ARI: 0.3571076531561054
Average NMI: 0.4942517223686824
---------
Results for main_program_classic_c:
Average Total Time: 16.958840894699097
Average ARI: 0.47726896950869657
Average NMI: 0.6838313621031317
---------
Results for main_program_classic_d:
Average Total Time: 16.748224782943726
Average ARI: 0.47451688631082967
Average NMI: 0.6796567746512323
---------
```

Fig 6.

4.3 Hardware specifications:

Coding platform: Python (Jupyter, Google Colab environment)

All experiments are performed in the following two different laptop specifications:

1. HP Laptop; 12th Gen Intel(R) Core (TM) i7-12650H (16 CPUs) @ 2.3GHz processor; 16GB RAM
2. DELL Laptop; 12th Gen Intel(R) Core (TM) i5-12500 (12 CPUs) @ 3.0 GHz processor; 16GB RAM

4.4 Rough Complexity analysis:

Low dimensional embedding follows $O(m.\log(k).t)$ where t is the number of iterations as per power method application.

Clustering with k-means depends on the number of points where it follows.$O(n.k.l.T)$ where T is the number of times the algorithm runs,

Overall complexity $O(m.\log(k).t + n.k.\log(k).T)$.

Space complexity for power method: $O(n.\log(k))$.

Space complexity for k-means: $O(n + k.\log(k))$.

5. SUMMARY AND CONNECTIONS TO THE COURSE:

5.1 Summary:

As we can see from Fig 6, running time for FASTSPECTRALCLUSTER (main_program_fastc, main_program_fastd) is less than that of CLASSICSPECTRALCLUSTER (main_program_classic_c, main_program_classic_d) for a trade-off in accuracy (represented by NMI, ARI)

knn - mode	Indicators	FASTSPECTRALCLUSTER	CLASSICSPECTRALCLUSTER
Connectivity	NMI	0.4960	0.6838 ↑
	ARI	0.3623	0.4772 ↑
	Running time	16.1241 ↓	16.9588
Distance	NMI	0.4942	0.6796 ↑
	ARI	0.3571	0.4745 ↑
	Running time	16.0628 ↓	16.7482

5.2 Connections to the course:

Following topics taught in the class were closely related to the study and analysis of this chosen paper:

1. k -nearest neighbours
2. Clustering algorithms (Spectral Clustering, Lloyd's algorithm, k -means ++)
3. KL-divergence
4. Principal Component Analysis
5. Singular Value Decomposition
6. Model evaluations.

6. REFERENCES:

[1] Peter Macgregor and He Sun. A tighter analysis of spectral clustering, and beyond. In 39th International Conference on Machine Learning (ICML'22), pages 14717–14742, 2022.

[2] Gene H Golub and Henk A Van der Vorst. Eigenvalue computation in the 20th century. Journal of Computational and Applied Mathematics, 123(1-2):35–65, 2000.

[3] Herman Müntz et al. Solution directe de l'équation séculaire et de quelques problemes analogues transcendants. Comptes rendus de l'Académie des Sciences, 156:43–46, 1913.

[4] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In Proceedings of the 18th Annual Symposium on Computational Geometry (SoCG'02), pages 10–18, 2002.
[5] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In Proceedings of the 23rd Annual Symposium on Computational Geometry (SoCG'07), pages 11–18, 2007.
[6] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time (1+ϵ)-approximation algorithm for k-means clustering in any dimension. In 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pages 454–462. IEEE, 2004.
[7] Stuart Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982.

7. APPENDIX:

IE 529 Project Codes

Fast Spectral Clustering using Connectivity as the criteria for the K-nearest neighbors

In [3]:
```
import numpy as np
import scipy.sparse.linalg
from sklearn.cluster import KMeans
from sklearn.neighbors import kneighbors_graph
from sklearn.metrics.pairwise import rbf_kernel
import stag.graph
import stag.stag_internal
import math
import time
import random
import pandas as pd
from sklearn.metrics import normalized_mutual_info_score,adjusted_rand_score
from sklearn.decomposition import PCA
```

In [2]:
```
from typing import List, Set
import pickle
import time
import os.path
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.sparse
from sklearn.metrics import normalized_mutual_info_score
from sklearn.neighbors import kneighbors_graph
from sklearn.datasets import fetch_openml
from scipy.spatial.distance import pdist, squareform
import stag.random
import stag.graph
import stag.cluster
import stag.graphio
```

In [51]:
```
# mnist = fetch_openml('har')
```

In [50]:
```
# mnist.data
```

In [52]:
```
# mnist.target
```

In [7]:
```
# y=mnist.target
```

In [8]:
```
# y.shape
```

In [4]:
```
def preprocess_openml_data_connectivity(dataset_name: str):
    # Load the graph
    mnist = fetch_openml(dataset_name)
    replace_dict = {chr(i): i-96 for i in range(97, 107)}
    X = np.array(mnist.data.replace(replace_dict))
    target_to_label = {}
    gt_labels = []
    next_label = 0
    for l in list(mnist.target):
        if l not in target_to_label:
            target_to_label[l] = next_label
            next_label += 1
        gt_labels.append(target_to_label[l])
    knn_graph = kneighbors_graph(X, n_neighbors=10, mode='connectivity',
                                 include_self=False)
    new_adj = scipy.sparse.lil_matrix(knn_graph.shape)
    for i, j in zip(*knn_graph.nonzero()):
        new_adj[i, j] = 1
        new_adj[j, i] = 1

    return new_adj,gt_labels
#     with open(f"data/{dataset_name}.pickle", 'wb') as fout:
#         pickle.dump((new_adj, gt_labels), fout)
#     with open(f"data/{dataset_name}_data.pickle", 'wb') as fout:
#         pickle.dump(X, fout)
```

In [5]:
```
def fast_spectral_cluster(g: stag.graph.Graph, k: int, t_const=10):
    l = min(k, math.ceil(math.log(k, 2)))
    t = t_const * math.ceil(math.log(g.number_of_vertices() / k, 2))
    M = g.normalised_signless_laplacian()
    Y = np.random.normal(size=(g.number_of_vertices(), l))
    for _ in range(t):
        Y = M @ Y
    labels, cl_center = kmeans(Y, k)
    return labels,cl_center
```

In [6]:
```
def kmeans(data, k):
    """
    Apply the kmeans algorithm to the given data, and return the labels.
    """
    kmeans_obj = KMeans(n_clusters=k,n_init=1,init='random',max_iter=1000, tol=1e-5)
    kmeans_obj.fit(data)
    return [int(x) for x in list(kmeans_obj.labels_)], kmeans_obj.cluster_centers_
```

In [8]:
```
def main_program_fastc():
    mnist = fetch_openml('har')
    y=mnist.target
    start=time.time()
    adj,gt_labels=preprocess_openml_data_connectivity('har')
    g = stag.graph.Graph(adj)
    K=6
    y_pred,clu=fast_spectral_cluster(g,K)
    end=time.time()
    tot_time=end-start
    y_pred_adjusted = [label + 1 for label in y_pred]
    ari=adjusted_rand_score(y,y_pred_adjusted )
    nmi=normalized_mutual_info_score(y,y_pred_adjusted )
#     print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
#     print(ari)
#     print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
#     print(nmi)
    y1=y.tolist()
    y1=[int(num) for num in y1]
    return y_pred_adjusted,clu,mnist.data,y1,tot_time,ari,nmi
```

In [9]:
```
y_pred_1,clu_1,data_1,y_1,tot_time_1,ari_1,nmi_1=main_program_fastc()
```

In [10]:
```
print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
print(ari_1)
print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
print(nmi_1)
```

```
The adjusted_rand_score between the assigned clusters and the actual clusters:
0.49170618201702704
The normalized_mutual_info_score between the assigned clusters and the actual clusters:
0.6309219262311886
```

In [149]:
```
# np.unique(y_1)
```

In [ ]:
```
# np.unique(_)
```

In [11]:
```
# np.unique(y_pred_1)
```

In [12]:
```
# np.unique(y_1)
```

In [13]:
```
# len(y_pred_1)
```

In [14]:
```
print("the total time taken to run the fast spectral clustering with connectivity as the mode: ")
print(tot_time_1)
```

```
the total time taken to run the fast spectral clustering with connectivity as the mode:
16.58402705192566
```

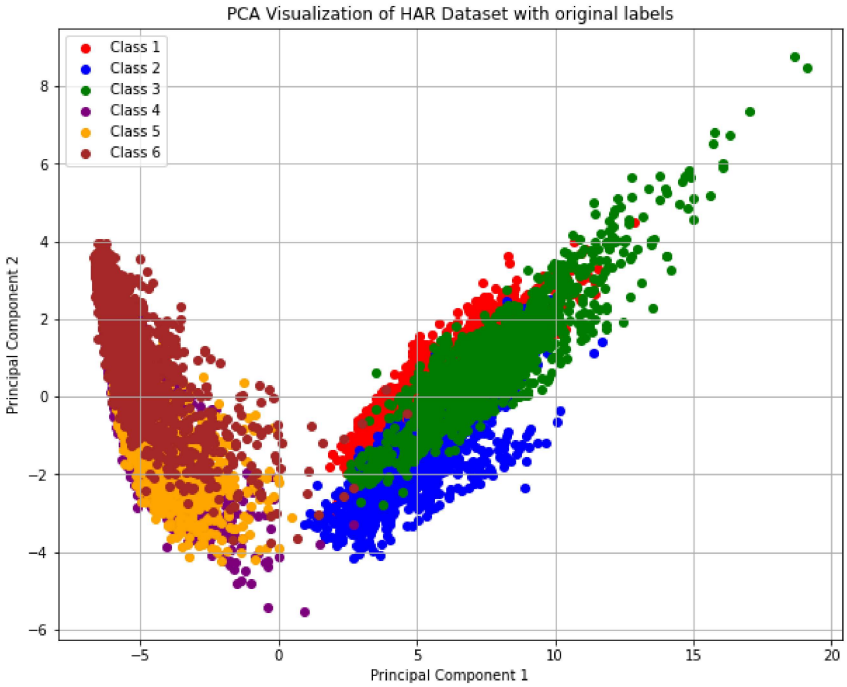
Plot by performing PCA with the data(selecting two principal components for representation) through

In [27]:
```
# import matplotlib.pyplot as plt
# from sklearn.decomposition import PCA

# # Perform dimensionality reduction using PCA
# pca = PCA(n_components=2, random_state=42)
# X_pca = pca.fit_transform(data_1)

# # Plot the data points with different colors based on cluster labels
# plt.figure(figsize=(10, 8))
# plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_1,cmap='viridis', s=20)
# plt.colorbar(label='Cluster')
# plt.title('PCA Visualization of HAR Dataset with Fast Spectral Clustering')
# plt.xlabel('Principal Component 1')
# plt.ylabel('Principal Component 2')
# plt.show()
```

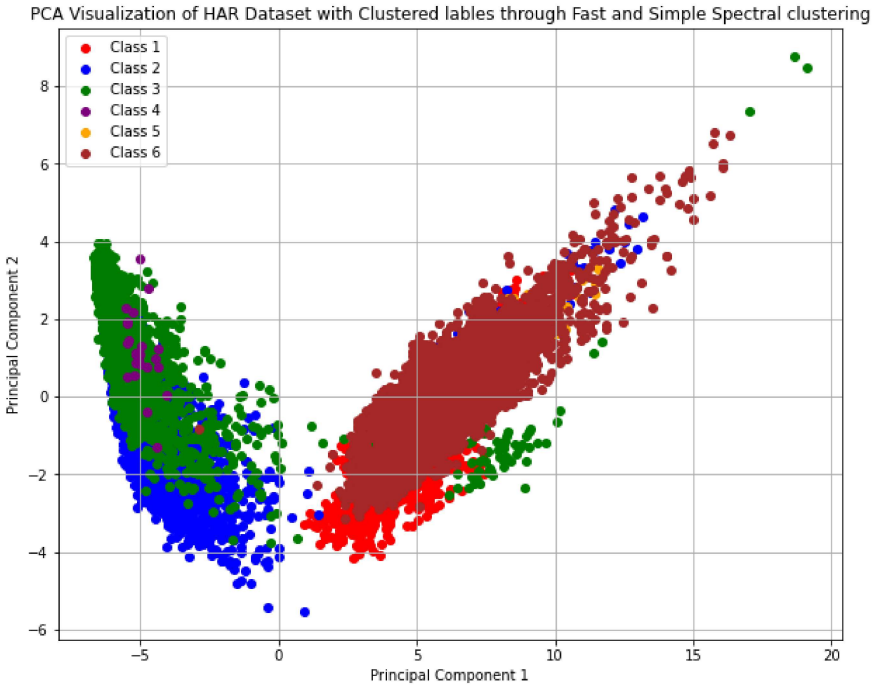
In [24]:
```
# for i, target in enumerate(np.unique(y_pred_1)):
#     print(i,target)
```

In [15]:
```
pca = PCA(n_components=2, random_state=42)  # Choose 2 principal components for visualization
X_pca = pca.fit_transform(data_1)
X_pca=np.array(X_pca)
```

In [16]:
```
# Plot the PCA results
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_1)):
    plt.scatter(X_pca[y_1== target, 0], X_pca[y_1 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with original labels")
plt.legend()
plt.grid(True)
plt.show()
```



In [17]:
```
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_pred_1)):
    plt.scatter(X_pca[y_pred_1== target, 0], X_pca[y_pred_1 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with Clustered lables through Fast and Simple Spectral clustering")
plt.legend()
plt.grid(True)
plt.show()
```



In [34]:
```
# import matplotlib.pyplot as plt
# num_clusters=6
# # pca = PCA(n_components=2, random_state=42)
# # X_pca = pca.fit_transform(data)
# def figure_plot_spectral(x, p_ij, num_clusters, data):
#   colors = ['r', 'g', 'b', 'y', 'c', 'm', 'k', 'orange', 'purple', 'pink', 'brown', 'lime', 'teal', 'olive', 'navy
#   plt.figure(figsize=(10, 8))
#   data1=np.array(data)
#   for i in range(num_clusters):
#     points = data1[p_ij == i]
#     # plt.scatter(points[:, 0], points[:, 1], s=50, c=colors[i], label=f'Cluster {i+1}')
#     plt.scatter(X_pca[:, 0], X_pca[:, 1], c=p_ij, cmap='viridis', s=40)
#   # plt.scatter(x[:, 0], x[:, 1], s=100, c='black', marker='*', label='Centroids')
#   plt.colorbar(label='Cluster')
#   plt.title('Fast Spectral Clustering')
#   plt.xlabel('Feature 1')
#   plt.ylabel('Feature 2')
#   plt.legend()
#   plt.grid(True)
#   return plt.show()

# figure_plot_spectral(clu_1, y_pred_1,num_clusters,data_1)
```

Fast Spectral Clustering using distance as the criteria as the criteria for the K-nearest neighbors

In [18]:
```
def preprocess_openml_data_distance(dataset_name: str):
    # Load the dataset
    mnist = fetch_openml(dataset_name)
    replace_dict = {chr(i): i-96 for i in range(97, 107)}
    X = np.array(mnist.data.replace(replace_dict))
    target_to_label = {}
    gt_labels = []
    next_label = 0
    for l in list(mnist.target):
        if l not in target_to_label:
            target_to_label[l] = next_label
            next_label += 1
        gt_labels.append(target_to_label[l])

    # Create the k-nearest neighbors graph in 'distance' mode
    knn_graph = kneighbors_graph(X, n_neighbors=10, mode='distance', include_self=False)

    # Convert the graph to a lil_matrix
    distance_matrix = scipy.sparse.lil_matrix(knn_graph)

    # Make the matrix symmetric
    for i, j in zip(*distance_matrix.nonzero()):
        distance_matrix[j, i] = distance_matrix[i, j]  # Ensure the matrix is symmetric

    return distance_matrix, gt_labels
```

In [20]:
```
def main_program_fastd():
    mnist = fetch_openml('har')
    y=mnist.target
    start = time.time()
    adj,gt_labels=preprocess_openml_data_distance('har')
    g = stag.graph.Graph(adj)
    K=6
    y_pred,clu=fast_spectral_cluster(g,K)
    end=time.time()
    tot_time=end-start
    y_pred_adjusted = [label + 1 for label in y_pred]
    ari=adjusted_rand_score(y,y_pred_adjusted )
    nmi=normalized_mutual_info_score(y,y_pred_adjusted )
#     print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
#     print(ari)
#     print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
#     print(nmi)
    y1=y.tolist()
    y1=[int(num) for num in y1]
    return y_pred_adjusted,clu,mnist.data,y1,tot_time,ari,nmi
```

In [21]:
```
y_pred_2,clu_2,data_2,y_2,tot_time_2,ari_2,nmi_2=main_program_fastd()
```

In [23]:
```
print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
print(ari_2)
print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
print(nmi_2)
```

```
The adjusted_rand_score between the assigned clusters and the actual clusters:
0.40131418251826434
The normalized_mutual_info_score between the assigned clusters and the actual clusters:
0.5161058567517116
```

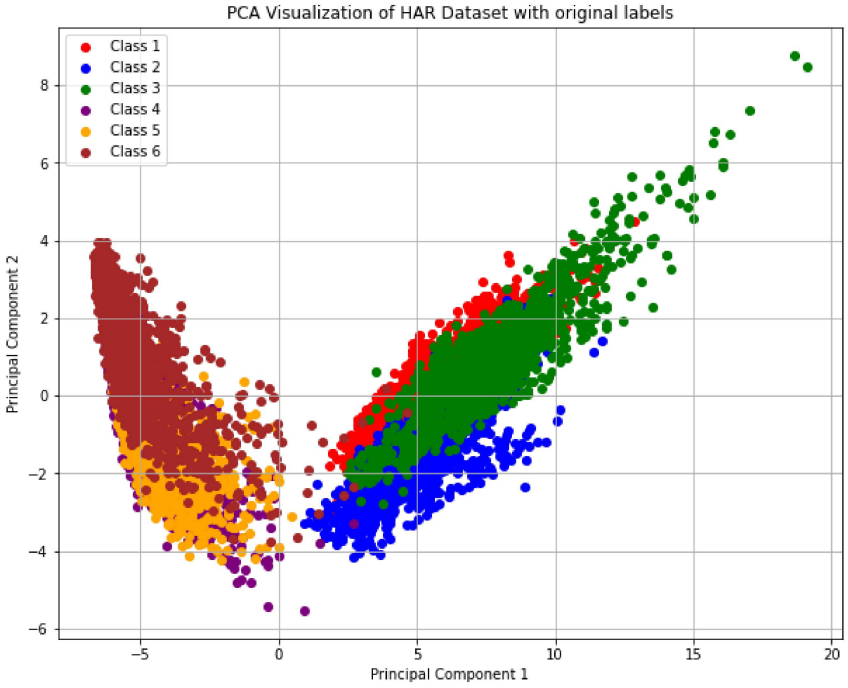
In [24]:
```
tot_time_2
```

Out[24]:
```
16.097012042999268
```

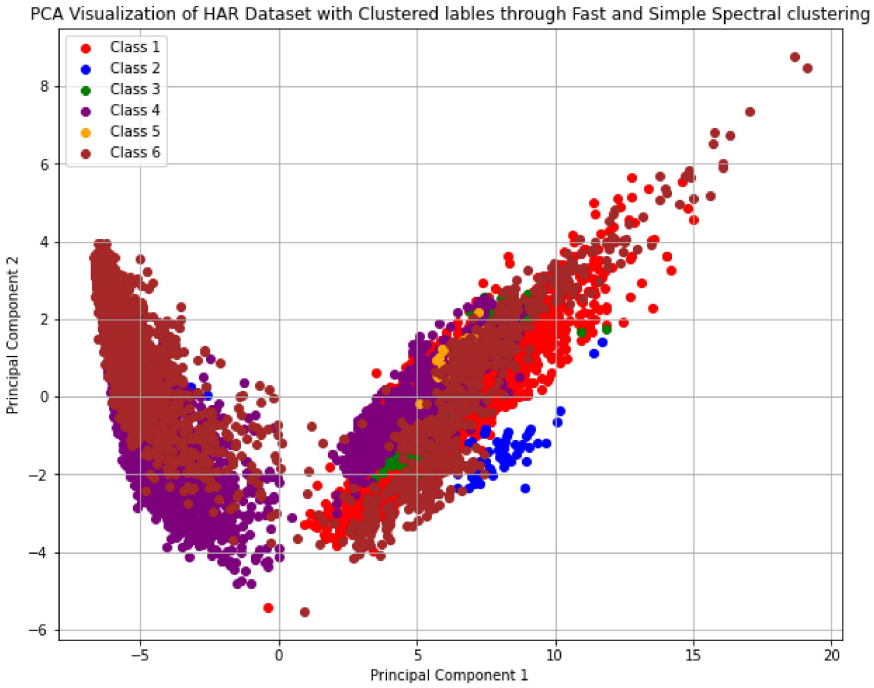
In [91]:
```
# v=target_2.tolist()
# target_2=[int(num) for num in v]
```

In [25]:
```
pca = PCA(n_components=2, random_state=42)  # Choose 2 principal components for visualization
X_pca = pca.fit_transform(data_2)
X_pca=np.array(X_pca)
```

In [26]:
```
# Plot the PCA results
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_2)):
    plt.scatter(X_pca[y_2== target, 0], X_pca[y_2 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with original labels")
plt.legend()
plt.grid(True)
plt.show()
```



In [27]:
```
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_pred_2)):
    plt.scatter(X_pca[y_pred_2== target, 0], X_pca[y_pred_2 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with Clustered lables through Fast and Simple Spectral clustering")
plt.legend()
plt.grid(True)
plt.show()
```



Classical Spectral Clustering using normalized Lapalacian matrix with the the Lloyd's algorithm

Using Connectivity as the criteria for the K-nearest neighbors

In [28]:
```
def spectral_cluster(g: stag.graph.Graph, k: int):
    lap_mat = g.normalised_laplacian()
    _, eigenvectors = scipy.sparse.linalg.eigsh(lap_mat, k, which='SM')
    labels,cluster_center = kmeans(eigenvectors, k)
    return labels,cluster_center
```

In [29]:
```
def main_program_classic_c():
    mnist = fetch_openml('har')
    y=mnist.target
    start=time.time()
    adj,gt_labels=preprocess_openml_data_connectivity('har')
    g = stag.graph.Graph(adj)
    K=6
    y_pred,clu=spectral_cluster(g,K)
    end=time.time()
    tot_time=end-start
    y_pred_adjusted = [label + 1 for label in y_pred]
    ari=adjusted_rand_score(y,y_pred_adjusted )
    nmi=normalized_mutual_info_score(y,y_pred_adjusted )
#     print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
#     print(ari)
#     print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
#     print(nmi)
    y1=y.tolist()
    y1=[int(num) for num in y1]
    return y_pred_adjusted,clu,mnist.data,y1,tot_time,ari,nmi
```

In [30]:
```
y_pred_3,clu_3,data_3,y_3,tot_time_3,ari_3,nmi_3=main_program_classic_c()
```

In [31]:
```
print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
print(ari_3)
print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
print(nmi_3)
```

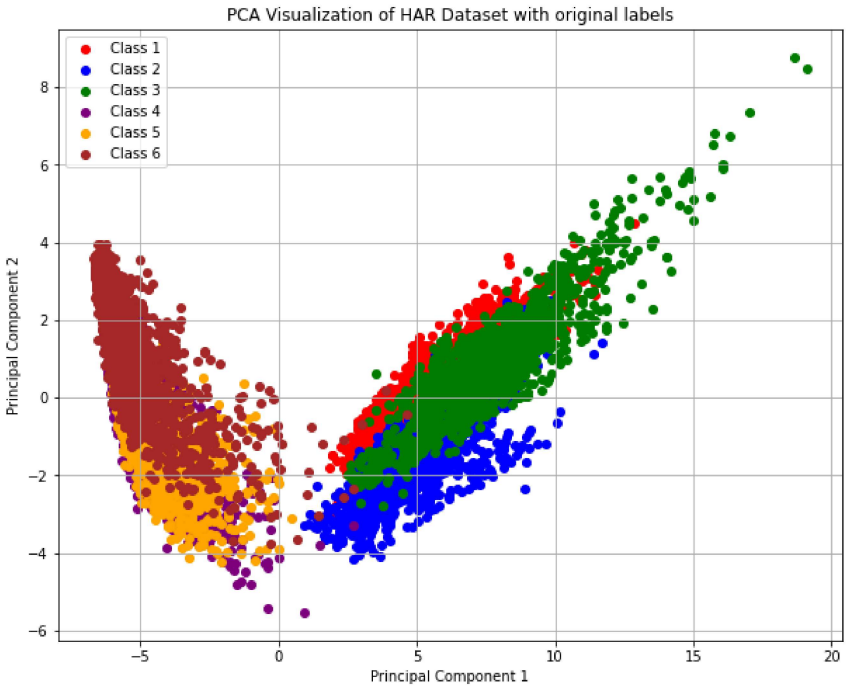
```
The adjusted_rand_score between the assigned clusters and the actual clusters:
0.513052428127109
The normalized_mutual_info_score between the assigned clusters and the actual clusters:
0.6529800450429591
```

In [32]:
```
tot_time_3
```

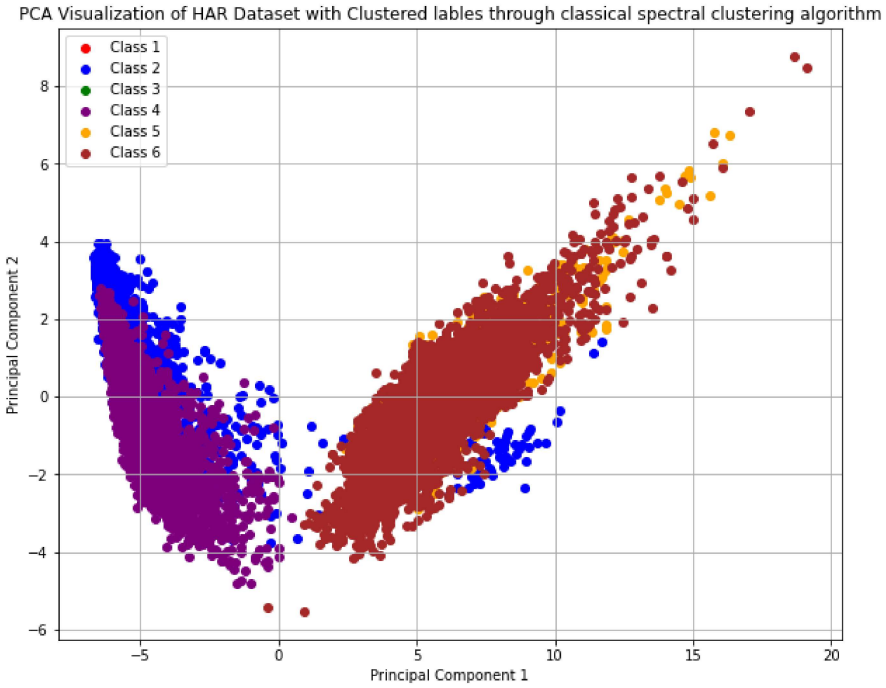
Out[32]:
```
21.61950159072876
```

In [33]:
```
pca = PCA(n_components=2, random_state=42)  # Choose 2 principal components for visualization
X_pca = pca.fit_transform(data_3)
X_pca=np.array(X_pca)
```

In [34]:
```
# Plot the PCA results
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_3)):
    plt.scatter(X_pca[y_3== target, 0], X_pca[y_3 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with original labels")
plt.legend()
plt.grid(True)
plt.show()
```



In [35]:
```
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_pred_3)):
    plt.scatter(X_pca[y_pred_3== target, 0], X_pca[y_pred_3 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with Clustered lables through classical spectral clustering algorithm")
plt.legend()
plt.grid(True)
plt.show()
```



Using distance as the criteria for the K-nearest neighbors

In [37]:
```
def main_program_classic_d():
    mnist = fetch_openml('har')
    y=mnist.target
    start=time.time()
    adj,gt_labels=preprocess_openml_data_distance('har')
    g = stag.graph.Graph(adj)
    K=6
    y_pred,clu=spectral_cluster(g,K)
    end=time.time()
    tot_time=end-start
    y_pred_adjusted = [label + 1 for label in y_pred]
    ari=adjusted_rand_score(y,y_pred_adjusted )
    nmi=normalized_mutual_info_score(y,y_pred_adjusted )
#     print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
#     print(ari)
#     print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
#     print(nmi)
    y1=y.tolist()
    y1=[int(num) for num in y1]
    return y_pred_adjusted,clu,mnist.data,y1,tot_time,ari,nmi
```

In [38]:
```
y_pred_4,clu_4,data_4,y_4,tot_time_4,ari_4,nmi_4=main_program_classic_d()
```

In [39]:
```
print("The adjusted_rand_score between the assigned clusters and the actual clusters: ")
print(ari_4)
print("The normalized_mutual_info_score between the assigned clusters and the actual clusters: ")
print(nmi_4)
```

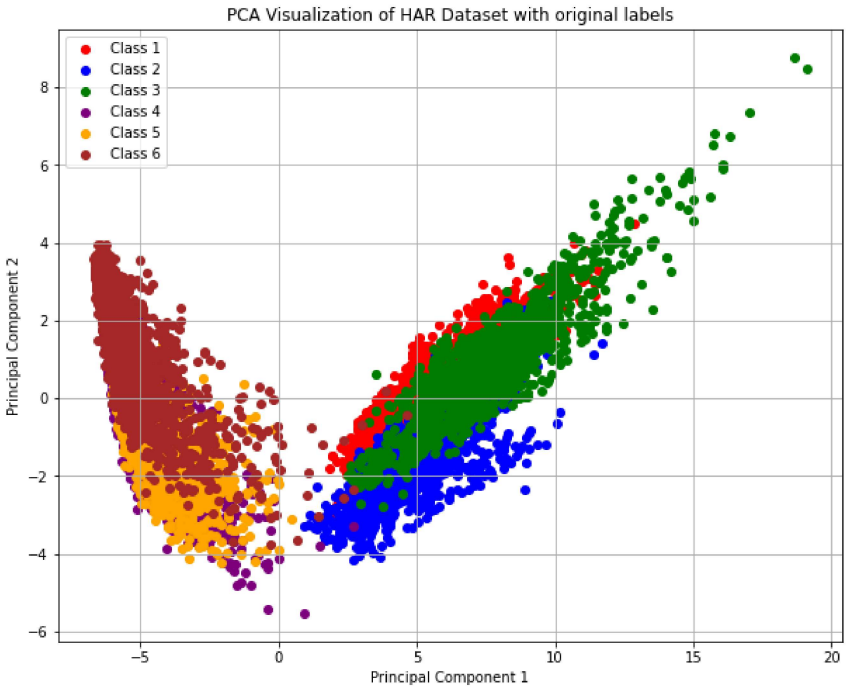
```
The adjusted_rand_score between the assigned clusters and the actual clusters:
0.4724847134037955
The normalized_mutual_info_score between the assigned clusters and the actual clusters:
0.6823109070905495
```

In [40]:
```
tot_time_4
```

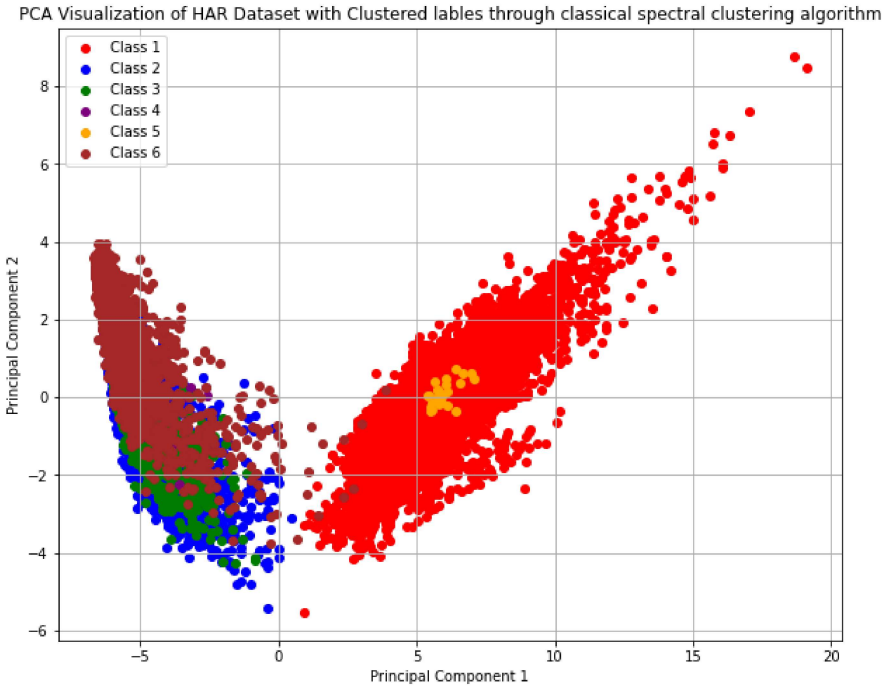
Out[40]:
```
17.576104402542114
```

In [41]:
```
pca = PCA(n_components=2, random_state=42)  # Choose 2 principal components for visualization
X_pca = pca.fit_transform(data_4)
X_pca=np.array(X_pca)
```

In [42]:
```
# Plot the PCA results
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_4)):
    plt.scatter(X_pca[y_4== target, 0], X_pca[y_4 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with original labels")
plt.legend()
plt.grid(True)
plt.show()
```



In [43]:
```
plt.figure(figsize=(10, 8))
colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown']  # Colors for each class
for i, target in enumerate(np.unique(y_pred_4)):
    plt.scatter(X_pca[y_pred_4== target, 0], X_pca[y_pred_4 == target, 1], c=colors[i], label=f"Class {target}")

#plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of HAR Dataset with Clustered lables through classical spectral clustering algorithm")
plt.legend()
plt.grid(True)
plt.show()
```



Calculating the average_time, average ari,and average ami

In [44]:
```
functions = [main_program_fastc, main_program_fastd, main_program_classic_c,main_program_classic_d]
```

In [46]:
```
for func in functions:
    total_times = []
    aris = []
    nmis = []

    for _ in range(10):
        y_pred, clu, data, y, tot_time, ari, nmi = func()
        total_times.append(tot_time)
        aris.append(ari)
        nmis.append(nmi)

    # Compute the averages
    average_time = np.mean(total_times)
    average_ari = np.mean(aris)
    average_nmi = np.mean(nmis)

    # Print the results for each function
    print(f"Results for {func.__name__}:")
    print("Average Total Time:", average_time)
    print("Average ARI:", average_ari)
    print("Average NMI:", average_nmi)
    print("---------")
```

```
Results for main_program_fastc:
Average Total Time: 16.1241436958313
Average ARI: 0.36230508644687465
Average NMI: 0.4960793898930381
---------
Results for main_program_fastd:
Average Total Time: 16.062884974479676
Average ARI: 0.3571076531561054
Average NMI: 0.4942517223686824
---------
Results for main_program_classic_c:
Average Total Time: 16.958840894699097
Average ARI: 0.47726896950869657
Average NMI: 0.6838313621031317
---------
Results for main_program_classic_d:
Average Total Time: 16.748224782943726
Average ARI: 0.47451688631082967
Average NMI: 0.6796567746512323
---------
```

In [ ]:

Computing Running time, ARI, NMI through un-normalized Laplacian matrix for our understanding

In [1]:
```
from sklearn.datasets import fetch_openml
from sklearn.neighbors import kneighbors_graph
from sklearn.cluster import SpectralClustering, KMeans
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import time
```

In [2]:
```
def Lloyds_algo(data, num_clusters, tolerance, max_iterations=1000):
  data= np.array(data)
  random_choice = np.random.choice(data.shape[0], num_clusters)
  x = data[random_choice]
  prev_distortion = None
  iterations = 0

  for i in range(max_iterations):
    diff= data - x.reshape(x.shape[0],1,x.shape[1])
    diff_squared = diff **2
    diff_squared_sum = np.sum(diff_squared, axis=2)
    euclid= np.sqrt(diff_squared_sum)
    p_ij = np.argmin(euclid, axis=0)

    x_new = np.array([data[p_ij == i].mean(axis=0) for i in range(num_clusters)])

    distortion = 0
    for j in range(num_clusters):
      distortion += np.sum((data[p_ij == j] - x[j]) ** 2) #/data1[p_ij == j].shape[0] #np.sum(data1[p_ij == j] ** 2)

    if np.linalg.norm(x_new - x) < tolerance or (prev_distortion is not None and np.linalg.norm(distortion - prev_distortion) < tolerance):
      break

    x = x_new
    prev_distortion = distortion
    iterations+=1

  return x, p_ij, distortion, num_clusters
```

In [3]:
```
def kmeans_algorithm(data, num_clusters, tolerance, max_iterations=1000):
    kmeans = KMeans(n_clusters=num_clusters, init='random', n_init=1, max_iter=max_iterations, tol=tolerance)
    kmeans.fit(data)
    centroids = kmeans.cluster_centers_
    labels = kmeans.labels_
    distortion = kmeans.inertia_
    running_time = kmeans.n_iter_

    return centroids, labels, distortion, num_clusters
```

In [3]:
```
def Spectral_clustering_knn_connectivity(data,K,k):
  A=kneighbors_graph(data,k, mode='connectivity')
  A=A.toarray()
  d_mat_k=np.zeros_like(A)
  for i in range(len(A)):
    d_mat_k[i][i]=np.sum(A[i])
  L_mat_k = d_mat_k - A
  eigenvalues_k, eigenvectors_k = np.linalg.eig(L_mat_k)
  sorted_indices_k = np.argsort(eigenvalues_k)#[::-1]
  sorted_eigenvalues_k = eigenvalues_k[sorted_indices_k]
  sorted_eigenvectors_k = eigenvectors_k[:, sorted_indices_k]
  first_k_eigenvectors_k = sorted_eigenvectors_k[:,:K]
  # first_k_eigenvectors_k
  # sorted_eigenvectors_k
  x, p_ij, distortion, num_clusters = Lloyds_algo(first_k_eigenvectors_k,K,1e-5)
  # x, p_ij, distortion, num_clusters = kmeans_algorithm(first_k_eigenvectors_k,K,1e-5)

  return x, p_ij, distortion, num_clusters, A, first_k_eigenvectors_k
```

In [4]:
```
def Spectral_clustering_knn_distance(data,K,k):
  A=kneighbors_graph(data,k, mode='distance')
  A=A.toarray()
  d_mat_k=np.zeros_like(A)
  for i in range(len(A)):
    d_mat_k[i][i]=np.sum(A[i])
  L_mat_k = d_mat_k - A
  eigenvalues_k, eigenvectors_k = np.linalg.eig(L_mat_k)
  sorted_indices_k = np.argsort(eigenvalues_k)#[::-1]
  sorted_eigenvalues_k = eigenvalues_k[sorted_indices_k]
  sorted_eigenvectors_k = eigenvectors_k[:, sorted_indices_k]
  first_k_eigenvectors_k = sorted_eigenvectors_k[:,:K]
  # first_k_eigenvectors_k
  # sorted_eigenvectors_k
  x, p_ij, distortion, num_clusters = Lloyds_algo(first_k_eigenvectors_k,K,1e-5)
  # x, p_ij, distortion, num_clusters = kmeans_algorithm(first_k_eigenvectors_k,K,1e-5)

  return x, p_ij, distortion, num_clusters, A, first_k_eigenvectors_k
```

In [5]:
```
# Load the HAR dataset (assuming it's available in scikit-learn)
har = fetch_openml(name='har')

# Extract features (X) and labels (y)
X = har.data
y = har.target
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will change from
``'liac-arff'`` to ``'auto'`` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportError` will be raised from
1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Sect
ion in fetch_openml's API doc for details.
  warn(
```

In [6]:
```
start_conn = time.time()
x_knn_conn, p_ij_knn_conn, distortion_knn_conn, num_clusters_knn_conn, adj_knn_conn, first_k_eigenvectors_conn = Spectral_clustering_knn_c
end_conn = time.time()
running_time_conn = end_conn - start_conn

ari_conn = adjusted_rand_score(y, p_ij_knn_conn)
nmi_conn = normalized_mutual_info_score(y, p_ij_knn_conn)
```

In [7]:
```
start_dist = time.time()
x_knn_dist, p_ij_knn_dist, distortion_knn_dist, num_clusters_knn_dist, adj_knn_dist, first_k_eigenvectors_dist = Spectral_clustering_knn_c
end_dist = time.time()
running_time_dist = end_dist - start_dist

ari_dist = adjusted_rand_score(y, p_ij_knn_dist)
nmi_dist = normalized_mutual_info_score(y, p_ij_knn_dist)
```

In [10]:
```
print("-----------------------------------------------------------------------------------------")
print("Spectral Clustering using k-eigenvectors | Mode = connectivity")
print("-----------------------------------------------------------------------------------------")
print("Adjusted Rand Index (ARI):", ari_conn)
print("Normalized Mutual Information (NMI):", nmi_conn)
print("Running time for normal Spectral Clustering (k-nearest neighbors) is: ", running_time_conn)
print("\n")
print("-----------------------------------------------------------------------------------------")
print("Spectral Clustering using k-eigenvectors | Mode = distance")
print("-----------------------------------------------------------------------------------------")
print("Adjusted Rand Index (ARI):", ari_dist)
print("Normalized Mutual Information (NMI):", nmi_dist)
print("Running time for normal Spectral Clustering (k-nearest neighbors) is: ", running_time_dist)
print("-----------------------------------------------------------------------------------------")
```

```
-----------------------------------------------------------------------------------------
Spectral Clustering using k-eigenvectors | Mode = connectivity
-----------------------------------------------------------------------------------------
Adjusted Rand Index (ARI): 0.49861484933924655
Normalized Mutual Information (NMI): 0.6971583494219175
Running time for normal Spectral Clustering (k-nearest neighbors) is:  1189.0399975776672


-----------------------------------------------------------------------------------------
Spectral Clustering using k-eigenvectors | Mode = distance
-----------------------------------------------------------------------------------------
Adjusted Rand Index (ARI): 0.33226710176178137
Normalized Mutual Information (NMI): 0.5473060829884699
Running time for normal Spectral Clustering (k-nearest neighbors) is:  1078.832494020462
-----------------------------------------------------------------------------------------
```