wuicpnd8q

March 16, 2024

[1]:
```
# always import
import sys
from time import time

# numpy & scipy
import numpy as np
import scipy

# sklearn
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import pairwise_distances_argmin, pairwise_distances
from sklearn.metrics.cluster import homogeneity_score, completeness_score,␣
↪v_measure_score, adjusted_rand_score, adjusted_mutual_info_score
from sklearn import neighbors
from sklearn.model_selection import train_test_split

# Hungarian algorithm
from munkres import Munkres
from scipy.optimize import linear_sum_assignment

# visuals
import matplotlib.pyplot as plt
from matplotlib import offsetbox
from sklearn.manifold import Isomap, TSNE

# maybe
from numba import jit
```

[2]:
```
# load MNIST data and normalization
from sklearn.datasets import fetch_openml
X, y = fetch_openml('mnist_784', version=1, return_X_y=True, data_home='mnist/')
y = np.asarray(list(map(int, y)))
X = np.asarray(X.astype(float))
```

```
X = scale(X)
n_digits = len(np.unique(y))
```

[3]:
```
# TODO: Hungarian algorithm
def calculate_hungarian_cost_matrix(y_true, y_pred, K) :
    conf_mat = confusion_matrix(y_true, y_pred, labels=range(K))
    #print(conf_mat.max())
    cost_matrix = (conf_mat.max() - conf_mat)/conf_mat.max()
    return cost_matrix

def accuracy(pred, y):
    dataset_size = y.size
    correct_predictions = 0
    #print(pred)
    #print(y)
    for i in range(dataset_size):
        if y[i] == (pred[i]):
            correct_predictions += 1
    precision = (correct_predictions/dataset_size)*100
    return precision
```

[4]:
```
# TODO: Kmeans
def get_distance_matrix(X, C):
    return (np.sqrt(((X[:,np.newaxis,:] - C) ** 2).sum(axis=2)))

def calculate_cost(centroids, X, min_index_matrix):
    return ((X - centroids[min_index_matrix]) ** 2).sum()

def run_kmeans(K, initial_centroids, X):
    num_samples, num_features = X.shape
    #print(centroids[0])
    #print(centroids.shape)
    centroids = initial_centroids.copy()
    distance_matrix = get_distance_matrix(X, centroids)
    max_iter = 300
    epsilon = 0.00001
    iter = 0
    old_cost = 0
    min_index_matrix = np.argmin(distance_matrix, axis = 1)
    new_cost = calculate_cost(centroids, X, min_index_matrix)
    #print("New Cost : " + str(new_cost))
    while iter <= max_iter and abs(new_cost-old_cost) >= epsilon :
        #print("Iteration : " + str(iter))
        #print(centroids)
        for j in range(K):
            jcluster_indices = np.array(min_index_matrix == j)
            jcluster = X[jcluster_indices, :]
```

```
            if jcluster.shape[0] > 0 :
                centroids[j,:] = jcluster.sum(axis=0)/jcluster.shape[0]
        distance_matrix = get_distance_matrix(X, centroids)
        min_index_matrix = np.argmin(distance_matrix, axis = 1)
        old_cost = new_cost
        new_cost = calculate_cost(centroids, X, min_index_matrix)
        iter = iter + 1
    return [centroids, new_cost]


#print ("NOTE : THESE VALUES DO NOT CORRESPOND TO ANY ANSWER AND WERE SIMPLY␣
 ↪USED FOR TESTING K-MEANS")
num_samples, num_features = X.shape
K = 10
'''
#Using the Forgy seed method for initialisation : take K random datapoints from␣
 ↪the dataset
initial_centroid_indices = np.random.choice(num_samples, K, replace=False)
centroids = X[initial_centroid_indices][:]
#print(initial_centroid_indices)
[final_kmeans_centroids, new_cost] = run_kmeans(K, centroids, X)
#print(final_kmeans_solution[0])
distance_matrix = get_distance_matrix(X, final_kmeans_centroids)
y_pred = np.argmin(distance_matrix, axis = 1)
homogeneity_score = metrics.homogeneity_score(y, y_pred)
print("Homogeneity Score (using Forgy method): " + str(homogeneity_score))
completeness_score = metrics.completeness_score(y, y_pred)
print("Completeness Score (using Forgy method): " + str(completeness_score))
vmeasure_score = metrics.v_measure_score(y, y_pred)
print("V-Measure Score (using Forgy method): " + str(vmeasure_score))
adjusted_rand_score = metrics.adjusted_rand_score(y, y_pred)
print("Adjusted Rand Score (using Forgy method): " + str(adjusted_rand_score))
adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(y, y_pred)
print("Adjusted Rand Score (using Forgy method): " +␣
 ↪str(adjusted_mutual_info_score))
cost_matrix = calculate_hungarian_cost_matrix(y, y_pred, K)
#print(cost_matrix)
cluster_to_class_matching = {}
row_ind, col_ind = linear_sum_assignment(cost_matrix)
for row, column in zip(row_ind, col_ind):
    print(f"Cluster {row} assigned to Class {column} ")
    cluster_to_class_matching[row] = column
for original_value, new_value in cluster_to_class_matching.items():
    y_pred[y_pred == original_value] = new_value + 10
y_pred = y_pred - 10
cm = confusion_matrix(y, y_pred, labels=range(K))
print(cm)
```

```
acc = accuracy(y_pred, y)
print("Accuracy : " + str(acc))
print ("NOTE : THESE VALUES DO NOT CORRESPOND TO ANY ANSWER AND WERE SIMPLY␣
↪USED FOR TESTING K-MEANS")
'''
```

[4]: '\n#Using the Forgy seed method for initialisation : take K random datapoints from the dataset\ninitial_centroid_indices = np.random.choice(num_samples, K, replace=False)\ncentroids = X[initial_centroid_indices][:]\n#print(initial_centroid_indices)\n[final_kmeans_centroids, new_cost] = run_kmeans(K, centroids, X)\n#print(final_kmeans_solution[0])\ndistance_matrix = get_distance_matrix(X, final_kmeans_centroids)\ny_pred = np.argmin(distance_matrix, axis = 1)\nhomogeneity_score = metrics.homogeneity_score(y, y_pred)\nprint("Homogeneity Score (using Forgy method): " + str(homogeneity_score))\ncompleteness_score = metrics.completeness_score(y, y_pred)\nprint("Completeness Score (using Forgy method): " + str(completeness_score))\nvmeasure_score = metrics.v_measure_score(y, y_pred)\nprint("V-Measure Score (using Forgy method): " + str(vmeasure_score))\nadjusted_rand_score = metrics.adjusted_rand_score(y, y_pred)\nprint("Adjusted Rand Score (using Forgy method): " + str(adjusted_rand_score))\nadjusted_mutual_info_score = metrics.adjusted_mutual_info_score(y, y_pred)\nprint("Adjusted Rand Score (using Forgy method): " + str(adjusted_mutual_info_score))\ncost_matrix = calculate_hungarian_cost_matrix(y, y_pred, K)\n#print(cost_matrix)\ncluster_to_class_matching = {}\nrow_ind, col_ind = linear_sum_assignment(cost_matrix)\nfor row, column in zip(row_ind, col_ind):\n print(f"Cluster {row} assigned to Class {column} ")\n cluster_to_class_matching[row] = column\nfor original_value, new_value in cluster_to_class_matching.items():\n   y_pred[y_pred == original_value] = new_value + 10\ny_pred = y_pred - 10\ncm = confusion_matrix(y, y_pred, labels=range(K))\nprint(cm)\nacc = accuracy(y_pred, y)\nprint("Accuracy : " + str(acc))\nprint ("NOTE : THESE VALUES DO NOT CORRESPOND TO ANY ANSWER AND WERE SIMPLY USED FOR TESTING K-MEANS")\n'

[5]:
```
print("Section 2 a/b) - I")
pca = PCA(n_components=K)
pca.fit(X)
centroids = pca.components_
[final_kmeans_centroids, new_cost] = run_kmeans(K, centroids, X)
#print(final_kmeans_solution[0])
distance_matrix = get_distance_matrix(X, final_kmeans_centroids)
y_pred = np.argmin(distance_matrix, axis = 1)
homogeneity_score = metrics.homogeneity_score(y, y_pred)
print("Homogeneity Score (using top K principal eigenvectors of PCA): " +␣
↪str(homogeneity_score))
completeness_score = metrics.completeness_score(y, y_pred)
print("Completeness Score (using top K principal eigenvectors of PCA): " +␣
↪str(completeness_score))
```

```
vmeasure_score = metrics.v_measure_score(y, y_pred)
print("V-Measure Score (using top K principal eigenvectors of PCA): " +␣
 ↪str(vmeasure_score))
adjusted_rand_score = metrics.adjusted_rand_score(y, y_pred)
print("Adjusted Rand Score (using top K principal eigenvectors of PCA): " +␣
 ↪str(adjusted_rand_score))
adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(y, y_pred)
print("Adjusted Rand Score (using top K principal eigenvectors of PCA): " +␣
 ↪str(adjusted_mutual_info_score))
cost_matrix = calculate_hungarian_cost_matrix(y, y_pred, K)
#print(cost_matrix)
cluster_to_class_matching = {}
row_ind, col_ind = linear_sum_assignment(cost_matrix)
print("Class to Cluster Assignments based on Hungarian Algorithm")
for row, column in zip(row_ind, col_ind):
    print(f"Class {row} assigned to Cluster {column} ")
    cluster_to_class_matching[column] = row
for original_value, new_value in cluster_to_class_matching.items():
    y_pred[y_pred == original_value] = new_value + 10
y_pred = y_pred - 10
print("Confusion Matrix on predictions post matching : ")
cm = confusion_matrix(y, y_pred, labels=range(K))
print(cm)
acc = accuracy(y_pred, y)
print("Accuracy : " + str(acc))
```

```
Section 2 a/b) - I
Homogeneity Score (using top K principal eigenvectors of PCA):
0.4197975269152666
Completeness Score (using top K principal eigenvectors of PCA):
0.4419190420959369
V-Measure Score (using top K principal eigenvectors of PCA): 0.43057433880263696
Adjusted Rand Score (using top K principal eigenvectors of PCA):
0.32075816979535976
Adjusted Rand Score (using top K principal eigenvectors of PCA):
0.4304273983543338
Class to Cluster Assignments based on Hungarian Algorithm
Class 0 assigned to Cluster 0
Class 1 assigned to Cluster 4
Class 2 assigned to Cluster 2
Class 3 assigned to Cluster 5
Class 4 assigned to Cluster 9
Class 5 assigned to Cluster 8
Class 6 assigned to Cluster 3
Class 7 assigned to Cluster 7
Class 8 assigned to Cluster 6
Class 9 assigned to Cluster 1
```

```
Confusion Matrix on predictions post matching :
[[3876   35  118 1356   13  658  351    6  480   10]
 [   0 7638   13   27    8  159   16    5   10    1]
 [  40  827 2380  726  185   85  862   29 1815   41]
 [  10  577  734 4097  200  118   88   95 1117  105]
 [  55  536   73    5 3968  948  128  741   27  343]
 [  33  467  246 2131  311 2709  102   93  158   63]
 [ 282  561  423  109   27  133 5323    1   12    5]
 [  20  516   16    9 1580  134    3 4086   13  916]
 [  45 1171  205 2589  355 2096   31  182   78   73]
 [  44  331   23  124 3487  131    5 2368   14  431]]
Accuracy : 49.40857142857143
```

[6]:

```
print("Section 2 a/b) - II")
d = 30
pca = PCA(n_components=d)
X_pca = pca.fit_transform(X)
#print(X_pca.shape)
no_of_runs = 10
cost = 100000000
final_centroids = np.zeros((K, d))
for i in range(no_of_runs) :
    #Using the Forgy seed method for initialisation : take K random datapoints␣
 ↪from PCA-applied dataset
    rs = 5*i
    np.random.seed(rs)
    initial_centroid_indices = np.random.choice(num_samples, K, replace=False)
    centroids = X_pca[initial_centroid_indices][:]
    #print(centroids.shape)
    [final_kmeans_centroids, new_cost] = run_kmeans(K, centroids, X_pca)
    if cost > new_cost :
        cost = new_cost
        final_centroids = final_kmeans_centroids.copy()
distance_matrix = get_distance_matrix(X_pca, final_centroids)
y_pred = np.argmin(distance_matrix, axis = 1)
homogeneity_score = metrics.homogeneity_score(y, y_pred)
print("Homogeneity Score (using PCA for dimensionality reduction to d = 30): "␣
 ↪+ str(homogeneity_score))
completeness_score = metrics.completeness_score(y, y_pred)
print("Completeness Score (using PCA for dimensionality reduction to d = 30): "␣
 ↪+ str(completeness_score))
vmeasure_score = metrics.v_measure_score(y, y_pred)
print("V-Measure Score (using PCA for dimensionality reduction to d = 30): " +␣
 ↪str(vmeasure_score))
adjusted_rand_score = metrics.adjusted_rand_score(y, y_pred)
print("Adjusted Rand Score (using PCA for dimensionality reduction to d = 30):␣
 ↪" + str(adjusted_rand_score))
```

```
adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(y, y_pred)
print("Adjusted Rand Score (using PCA for dimensionality reduction to d = 30):
 ↪" + str(adjusted_mutual_info_score))
cost_matrix = calculate_hungarian_cost_matrix(y, y_pred, K)
#print(cost_matrix)
cluster_to_class_matching = {}
row_ind, col_ind = linear_sum_assignment(cost_matrix)
print("Class to Cluster Assignments based on Hungarian Algorithm")
for row, column in zip(row_ind, col_ind):
    print(f"Class {row} assigned to Cluster {column} ")
    cluster_to_class_matching[column] = row
for original_value, new_value in cluster_to_class_matching.items():
    y_pred[y_pred == original_value] = new_value + 10
y_pred = y_pred - 10
cm = confusion_matrix(y, y_pred, labels=range(K))
print("Confusion Matrix on predictions post matching : ")
print(cm)
acc = accuracy(y_pred, y)
print("Accuracy : " + str(acc))
```

```
Section 2 a/b) - II
Homogeneity Score (using PCA for dimensionality reduction to d = 30):
0.4222313136385659
Completeness Score (using PCA for dimensionality reduction to d = 30):
0.4439863964506075
V-Measure Score (using PCA for dimensionality reduction to d = 30):
0.43283566527794587
Adjusted Rand Score (using PCA for dimensionality reduction to d = 30):
0.32192715104290953
Adjusted Rand Score (using PCA for dimensionality reduction to d = 30):
0.43268938916389627
Class to Cluster Assignments based on Hungarian Algorithm
Class 0 assigned to Cluster 7
Class 1 assigned to Cluster 1
Class 2 assigned to Cluster 5
Class 3 assigned to Cluster 6
Class 4 assigned to Cluster 0
Class 5 assigned to Cluster 2
Class 6 assigned to Cluster 4
Class 7 assigned to Cluster 9
Class 8 assigned to Cluster 8
Class 9 assigned to Cluster 3
Confusion Matrix on predictions post matching :
[[3965   35   82  942   20  658  763    7  418   13]
 [   0 7621   13   29    5  173   16    7   12    1]
 [  42  839 2588  775  165   92  673   42 1717   57]
 [  11  678  219 4440  187  152   97   91 1141  125]
```

```
 [  50  543   49    3 3938  905  159  677   25  475]
 [  35  529  128 2061  326 2798  129   70  157   80]
 [ 240  504  774   53   37  132 5120    1    9    6]
 [  19  552    7   14 1584  114    3 4072   17  911]
 [  48 1348   96 2220  398 2297   53  180   78  107]
 [  42  361   15  106 3502  118    6 2275   15  518]]
Accuracy : 50.197142857142865
```

[7]:

```python
# TODO: Spectral clustering
def spectral_clustering_analysis(X_pca):
    # k is number of nearest neighbours in the KNN model, K is the number of␣
 ↪clusters in k_means
    k = 500
    H = neighbors.NearestNeighbors(n_neighbors=k, algorithm='kd_tree',␣
 ↪metric='euclidean').fit(X_pca).kneighbors_graph(mode='distance')
    #print(E.shape)
    sigma = H.sum()/H.nnz
    H = H.power(2)
    E = -H.multiply(1.0/(2*sigma*sigma))
    #print("norm done")
    #print(E)
    #print(scipy.sparse.issparse(E))
    E.data = np.exp(E.data)
    #print("exp done")
    #print(E)
    E.setdiag(values=0)
    #print(E)
    #total_sum = E.sum()
    #E = E.multiply(1.0/total_sum)
    row_sums = E.sum(axis=1)
    row_sums = np.ravel(row_sums)
    row_sums_reciprocal = np.reciprocal(row_sums, where=row_sums!=0)
    E = E.multiply(row_sums_reciprocal[:, np.newaxis])
    E.setdiag(values=1)
    #print(E)
    E_transpose = E.transpose()
    E = (E + E_transpose)
    E = E.multiply(0.5)
    #print(E)
    D = scipy.sparse.csr_array((E.shape))
    D.setdiag(E.sum(axis=1))
    #print(D)
    D_L = D.sqrt()
    D_L = scipy.sparse.linalg.inv(D_L)
    L = -D_L @ E @ D_L
    L_r, L_c = L.shape
    L = scipy.sparse.identity(L_r, format='csr') + L
```

```
    #print(L.shape)
    #print(L)
    no_of_eigenvectors = 20
    #print("Start Eigen Analysis")
    eigenvalues, eigenvectors = scipy.sparse.linalg.eigs(L,␣
 ↪k=no_of_eigenvectors, which='SR')
    #print(eigenvectors)
    idx = np.argsort(eigenvalues.real)
    sorted_eigen_values = eigenvalues[idx]
    sorted_eigen_vectors = eigenvectors.real[:,idx]
    k_eigen_vectors = sorted_eigen_vectors[:,1:]
    return [sorted_eigen_values, k_eigen_vectors]

print("Spectral Clustering Results :")
[sorted_eigen_values, k_eigen_vectors] = spectral_clustering_analysis(X_pca)
y_pred = KMeans(init='k-means++', n_clusters=10, n_init=10).
 ↪fit_predict(k_eigen_vectors)
homogeneity_score = metrics.homogeneity_score(y, y_pred)
print("Homogeneity Score (using spectral clustering): " +␣
 ↪str(homogeneity_score))
completeness_score = metrics.completeness_score(y, y_pred)
print("Completeness Score (using spectral clustering): " +␣
 ↪str(completeness_score))
vmeasure_score = metrics.v_measure_score(y, y_pred)
print("V-Measure Score (using spectral clustering): " + str(vmeasure_score))
adjusted_rand_score = metrics.adjusted_rand_score(y, y_pred)
print("Adjusted Rand Score (using spectral clustering): " +␣
 ↪str(adjusted_rand_score))
adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(y, y_pred)
print("Adjusted Rand Score (using spectral clustering): " +␣
 ↪str(adjusted_mutual_info_score))
cost_matrix = calculate_hungarian_cost_matrix(y, y_pred, 10)
#print(cost_matrix)
#print(sorted_eigen_values)
#print(k_eigen_vectors.shape)
cluster_to_class_matching = {}
row_ind, col_ind = linear_sum_assignment(cost_matrix)
print("Class to Cluster Assignments based on Hungarian Algorithm")
for row, column in zip(row_ind, col_ind):
    print(f"Class {row} assigned to Cluster {column} ")
    cluster_to_class_matching[column] = row
for original_value, new_value in cluster_to_class_matching.items():
    y_pred[y_pred == original_value] = new_value + 10
y_pred = y_pred - 10
print("Confusion Matrix on predictions post matching : ")
cm = confusion_matrix(y, y_pred, labels=range(10))
```

```python
print(cm)
acc = accuracy(y_pred, y)
print("Accuracy : " + str(acc))
```

```
Spectral Clustering Results :
```

```
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/_index.py:146: SparseEfficiencyWarning: Changing the
sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_arrayXarray(i, j, x)
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/linalg/_dsolve/linsolve.py:394: SparseEfficiencyWarning:
splu converted its input to CSC format
  warn('splu converted its input to CSC format', SparseEfficiencyWarning)
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/linalg/_dsolve/linsolve.py:285: SparseEfficiencyWarning:
spsolve is more efficient when sparse b is in the CSC matrix format
  warn('spsolve is more efficient when sparse b '
```

```
Homogeneity Score (using spectral clustering): 0.5987127729924084
Completeness Score (using spectral clustering): 0.6694485180155375
V-Measure Score (using spectral clustering): 0.6321078894912106
Adjusted Rand Score (using spectral clustering): 0.4677415369830309
Adjusted Rand Score (using spectral clustering): 0.6320129377410119
Class to Cluster Assignments based on Hungarian Algorithm
Class 0 assigned to Cluster 9
Class 1 assigned to Cluster 6
Class 2 assigned to Cluster 1
Class 3 assigned to Cluster 0
Class 4 assigned to Cluster 7
Class 5 assigned to Cluster 3
Class 6 assigned to Cluster 8
Class 7 assigned to Cluster 2
Class 8 assigned to Cluster 4
Class 9 assigned to Cluster 5
Confusion Matrix on predictions post matching :
[[6475    3   70   70    0    0   22    1  233   29]
 [   0 4396   28   21 3333    0    7    1   62   29]
 [  57   35 5861  362   12    0   44   38  473  108]
 [  11   10  185 6049   53    0    1   47  603  182]
 [   3   28  116   23   29    1   11   10   50 6553]
 [  44    4 1542 1803    8    0   19    6 2631  256]
 [ 104   12   95   29    7    0 3639    0 2939   51]
 [  10   88   44   38   43    1    0 4150   29 2890]
 [  47   40  351  945   28    0    3    6 5149  256]
 [  20   10   32   97   31    0    0   96   70 6602]]
Accuracy : 60.5
```

[8]:

```python
# TODO: Clustering + KNN
def euclidean_distance(row1, row2):
    distance = np.linalg.norm(row1 - row2)
    return distance

def knn(X_train, y_train, X_test, kvalue):
    y_pred = []
    for x in X_test :
        distances = [euclidean_distance(x, x_train) for x_train in X_train]
        k_indices = np.argsort(distances)[:kvalue]
        k_nearest_labels = [y_train[i] for i in k_indices]
        most_common = np.argmax(np.bincount(k_nearest_labels))
        y_pred.append(most_common)
    return np.array(y_pred)

no_of_runs = 10
cost = 100000000
K_d_cluster_size=100
k_values = [1,3,5]
final_centroids = np.zeros((K, d))
for i in range(no_of_runs) :
    #Using the Forgy seed method for initialisation : take K random datapoints␣
 ↪from PCA-applied dataset
    rs = 5*i
    np.random.seed(rs)
    initial_centroid_indices = np.random.choice(num_samples, K_d_cluster_size,␣
 ↪replace=False)
    centroids = X_pca[initial_centroid_indices][:]
    #print(centroids.shape)
    [final_kmeans_centroids, new_cost] = run_kmeans(K_d_cluster_size,␣
 ↪centroids, X_pca)
    #print(cost)
    if cost > new_cost :
        cost = new_cost
        final_centroids = final_kmeans_centroids.copy()
distance_matrix = get_distance_matrix(X_pca, final_centroids)
min_row_indices = np.argmin(distance_matrix, axis=0)
final_knn_train_X = X_pca[min_row_indices][:]
final_knn_train_y = y[min_row_indices][:]
final_knn_test_X = np.delete(X_pca, min_row_indices, axis=0)
final_knn_test_y = np.delete(y, min_row_indices, axis=0)

train_accuracies = []
test_accuracies = []
# We loop over the different values of k and calculate the accuracy for each␣
 ↪value of k.
for kvalue in k_values:
```

```
    # Predict labels for train set and test set
    y_pred_train = knn(final_knn_train_X, final_knn_train_y, final_knn_train_X,␣
↪kvalue)
    y_pred_test = knn(final_knn_train_X, final_knn_train_y, final_knn_test_X,␣
↪kvalue)
    # Calculate accuracy
    train_accuracies.append(accuracy(final_knn_train_y, y_pred_train))
    test_accuracies.append(accuracy(final_knn_test_y, y_pred_test))

print("Kmeans + kNN Results :")
print("Training Accuracy for Kmeans+knn for k = 1 : " +␣
↪str(train_accuracies[0]))
print("Testing Accuracy for Kmeans+knn for k = 1 : " + str(test_accuracies[0]))
print("Training Accuracy for Kmeans+knn for k = 3 : " +␣
↪str(train_accuracies[1]))
print("Testing Accuracy for Kmeans+knn for k = 3 : " + str(test_accuracies[1]))
print("Training Accuracy for Kmeans+knn for k = 5 : " +␣
↪str(train_accuracies[2]))
print("Testing Accuracy for Kmeans+knn for k = 5 : " + str(test_accuracies[2]))
```

```
Kmeans + kNN Results :
Training Accuracy for Kmeans+knn for k = 1 : 100.0
Testing Accuracy for Kmeans+knn for k = 1 : 82.00286123032903
Training Accuracy for Kmeans+knn for k = 3 : 93.0
Testing Accuracy for Kmeans+knn for k = 3 : 77.81545064377683
Training Accuracy for Kmeans+knn for k = 5 : 88.0
Testing Accuracy for Kmeans+knn for k = 5 : 76.03290414878397
```

[12]:
```
cost = 100000000
[sorted_eigen_values, k_eigen_vectors] = spectral_clustering_analysis(X_pca)
for i in range(no_of_runs) :
    #Using the Forgy seed method for initialisation : take K random datapoints␣
↪from PCA-applied dataset
    rs = 5*i
    np.random.seed(rs)
    initial_centroid_indices = np.random.choice(num_samples, K_d_cluster_size,␣
↪replace=False)
    centroids = k_eigen_vectors[initial_centroid_indices][:]
    #print(centroids.shape)
    [final_kmeans_centroids, new_cost] = run_kmeans(K_d_cluster_size,␣
↪centroids, k_eigen_vectors)
    #print(cost)
    if cost > new_cost :
        cost = new_cost
        final_centroids = final_kmeans_centroids.copy()
distance_matrix = get_distance_matrix(k_eigen_vectors, final_centroids)
```

```python
min_row_indices = np.argmin(distance_matrix, axis=0)
final_knn_train_X = k_eigen_vectors[min_row_indices][:]
final_knn_train_y = y[min_row_indices][:]
final_knn_test_X = np.delete(k_eigen_vectors, min_row_indices, axis=0)
final_knn_test_y = np.delete(y, min_row_indices, axis=0)

train_accuracies = []
test_accuracies = []
# We loop over the different values of k and calculate the accuracy for each␣
 ↪value of k.
for kvalue in k_values:
    # Predict labels for train set and test set
    y_pred_train = knn(final_knn_train_X, final_knn_train_y, final_knn_train_X,␣
 ↪kvalue)
    y_pred_test = knn(final_knn_train_X, final_knn_train_y, final_knn_test_X,␣
 ↪kvalue)
    # Calculate accuracy
    train_accuracies.append(accuracy(final_knn_train_y, y_pred_train))
    test_accuracies.append(accuracy(final_knn_test_y, y_pred_test))

print("Spectral Clustering + kNN Results :")
print("Training Accuracy for Spectral+Kmeans+knn for k = 1 : " +␣
 ↪str(train_accuracies[0]))
print("Testing Accuracy for Spectral+Kmeans+knn for k = 1 : " +␣
 ↪str(test_accuracies[0]))
print("Training Accuracy for Spectral+Kmeans+knn for k = 3 : " +␣
 ↪str(train_accuracies[1]))
print("Testing Accuracy for Spectral+Kmeans+knn for k = 3 : " +␣
 ↪str(test_accuracies[1]))
print("Training Accuracy for Spectral+Kmeans+knn for k = 5 : " +␣
 ↪str(train_accuracies[2]))
print("Testing Accuracy for Spectral+Kmeans+knn for k = 5 : " +␣
 ↪str(test_accuracies[2]))
```

```
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/_index.py:146: SparseEfficiencyWarning: Changing the
sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_arrayXarray(i, j, x)
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/linalg/_dsolve/linsolve.py:394: SparseEfficiencyWarning:
splu converted its input to CSC format
  warn('splu converted its input to CSC format', SparseEfficiencyWarning)
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/scipy/sparse/linalg/_dsolve/linsolve.py:285: SparseEfficiencyWarning:
spsolve is more efficient when sparse b is in the CSC matrix format
  warn('spsolve is more efficient when sparse b '
```

```
Spectral Clustering + kNN Results :
Training Accuracy for Spectral+Kmeans+knn for k = 1 : 100.0
Testing Accuracy for Spectral+Kmeans+knn for k = 1 : 78.8483547925608
Training Accuracy for Spectral+Kmeans+knn for k = 3 : 81.0
Testing Accuracy for Spectral+Kmeans+knn for k = 3 : 77.46638054363376
Training Accuracy for Spectral+Kmeans+knn for k = 5 : 80.0
Testing Accuracy for Spectral+Kmeans+knn for k = 5 : 76.13590844062948
```

[10]:

```
row_indices = np.random.choice(num_samples, K_d_cluster_size, replace=False)
final_knn_train_X = X_pca[row_indices][:]
final_knn_train_y = y[row_indices][:]
final_knn_test_X = np.delete(X_pca, row_indices, axis=0)
final_knn_test_y =np.delete(y, row_indices, axis=0)

train_accuracies = []
test_accuracies = []
# We loop over the different values of k and calculate the accuracy for each␣
 ↪value of k.
for kvalue in k_values:
    # Predict labels for train set and test set
    y_pred_train = knn(final_knn_train_X, final_knn_train_y, final_knn_train_X,␣
 ↪kvalue)
    y_pred_test = knn(final_knn_train_X, final_knn_train_y, final_knn_test_X,␣
 ↪kvalue)
    # Calculate accuracy
    train_accuracies.append(accuracy(final_knn_train_y, y_pred_train))
    test_accuracies.append(accuracy(final_knn_test_y, y_pred_test))

print("Random Sampling + kNN Results :")
print("Training Accuracy for RS+knn for k = 1 : " + str(train_accuracies[0]))
print("Testing Accuracy for RS+knn for k = 1 : " + str(test_accuracies[0]))
print("Training Accuracy for RS+knn for k = 3 : " + str(train_accuracies[1]))
print("Testing Accuracy for RS+knn for k = 3 : " + str(test_accuracies[1]))
print("Training Accuracy for RS+knn for k = 5 : " + str(train_accuracies[2]))
print("Testing Accuracy for RS+knn for k = 5 : " + str(test_accuracies[2]))
```

```
Random Sampling + kNN Results :
Training Accuracy for RS+knn for k = 1 : 100.0
Testing Accuracy for RS+knn for k = 1 : 73.03719599427754
Training Accuracy for RS+knn for k = 3 : 87.0
Testing Accuracy for RS+knn for k = 3 : 68.42346208869814
Training Accuracy for RS+knn for k = 5 : 83.0
Testing Accuracy for RS+knn for k = 5 : 67.6752503576538
```