Bangladesh University of Engineering and Technology

EEE 6608

Machine Learning & Pattern Recognition

Final assignement: Implementing K-means clustering alogrithm from scratch

Submitted by,
Name: Md. Al-Imran Abir
Student ID: 0421062549

Date of Submission: April 17, 2022

1 Code:

```
# Libraries

import numpy as np
import pandas as pd

from sklearn.datasets import load_digits
from sklearn.metrics import calinski_harabasz_score, silhouette_score
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

#---------- Load data--------------

digits = load_digits()
features = digits.data
labels = digits.target
print(f"Feauters -> type:{type(features)}; shape:{features.shape}")
print(f"Labels   -> type:{type(labels)}; shape:{labels.shape}")

#---------------------------------------
#----------------PCA-------------------
#---------------------------------------

# sample average
mu = np.mean(features, axis=0)
print(f"Shape of mean:{mu.shape}")

# mean centered sample
X = features - mu
# print(np.mean(X, axis=0))

# the covariance matrix
A = np.cov(X, rowvar=False)
print(f"shape of covariance matrix:{A.shape}")
# eigenvalues and right eigenvectors
eig_value, eig_vect = np.linalg.eig(A)

print(f"eigenValues: {eig_value.shape}; type: {type(eig_value)}")
print(f"eigenValues: {eig_vect.shape}; type: {type(eig_vect)}")


# find the value of K for which 95% of energy is retained
k = 1
```

```python
energy = 0
while energy <= 0.95:
    eig_value_k = eig_value[:k]
    energy = eig_value_k.sum()/eig_value.sum()
    k += 1
#     print(energy)
print(f"k:{k} -> energy:{energy}")

# choose the first k eigen vectors
U_k = eig_vect[:, :k]
print(f"shape of projection matrix: {U_k.shape}")

# Projection
X_proj = X.dot(U_k)
print(f"projected matrix shape: {X_proj.shape}")


#----------------------------------------------------
#----------------- K-means clustering------------------
#----------------------------------------------------

def compute_cost(X, centroids, cluster):
    m = X.shape[0]
    sum = 0
    for i in range(m):
        sum += (X[i] - centroids[int(cluster[i])]) @ (X[i] - centroids[int(cluster[i])]).T
    cost = sum/m
    return cost

def my_kmeans(X, k, seed=41):
    diff = 1
    np.random.seed(seed)
    cluster_labels = np.zeros(X.shape[0]) #cluster label for each sample
    rand_idx = np.random.choice(len(X), k, replace=False)
    #Randomly choosing Centroids
    centroids = X[rand_idx, :] #Step 1

    while diff:
        for i, row in enumerate(X):
            min_dist = float('inf')
            for idx, centroid in enumerate(centroids):
                d = np.sqrt((row - centroid) @ (row-centroid).T)
                if min_dist > d:
                    min_dist = d
                    cluster_labels[i] = idx
        new_centroids = pd.DataFrame(X).groupby(by=cluster_labels).mean().values
```

```

        if np.count_nonzero(centroids-new_centroids) == 0:
            diff = 0
        else:
            centroids = new_centroids
    cost = compute_cost(X, centroids, cluster_labels)
    cal_score = calinski_harabasz_score(X, cluster_labels)
    return centroids, cluster_labels, cost, cal_score

#---------------------------------------------------
#------------------- With PCA ---------------------
#---------------------------------------------------

def plot_cost(cost, k):
    plt.plot(range(2, k), cost)
    plt.show()

# #### Variation wrt No of Clusters

K = 15
cost = np.zeros(K-2)
sil_score = np.zeros(K-2)
cal_score = np.zeros(K-2)
centroids_lst = []
labels = np.zeros((K-1, X.shape[0]))

for k in range(2, K):
    centroids, labels[k-2], cost[k-2], cal_score[k-2] = my_kmeans(X_proj, k)
    centroids_lst.append(centroids)
    sil_score[k-2] = silhouette_score(X_proj, labels[k-2])
    print(f"k={k} -> cost:{cost[k-2]:.3f};
        calinski:{cal_score[k-2]:.3f}; sil:{sil_score[k-2]:.3f}")

# print(f"silhouette_score:{sil_score}\nCost: {cost}\ncalinski_harabasz_score: {cal_score}")
plot_cost(cost, K)

### Plot cost vs clusters
plt.figure()
plt.plot(range(2, K), cost)
plt.savefig("pca_cost_vs_clusters.png")
plt.xlabel("No of clusters")
plt.ylabel("Cost")
plt.show()

### Plot Calinski-Harabasz Index vs clusters
plt.figure()
```

```python
plt.plot(range(2, K), cal_score)
plt.xlabel("No of clusters")
plt.ylabel("Calinski-Harabasz Index")
plt.savefig("pca_cal_score_vs_clusters.png")
plt.show()

### Plot Silhouette Coefficient vs clusters
plt.figure()
plt.plot(range(2, K), sil_score)
plt.xlabel("No of clusters")
plt.ylabel("Silhouette Coefficient")
plt.savefig("pca_cost_vs_sil_score.png")
plt.show()


# Variation wrt No of Samples---------------------


ki = 10
per = np.linspace(0.1, 1, num=10)
z = per.shape[0]
cost_sam = np.zeros(z)
sil_score_sam = np.zeros(z)
cal_score_sam = np.zeros(z)
centroids_lst_sam = []
labels_lst_sam = []
for i, j in enumerate(per):
    idx = np.random.choice(range(X_proj.shape[0]), int(j*X_proj.shape[0]), replace=False)
    X_per = X_proj[idx,:]
#     print(i, j, X_per.shape[0])
    centroids_sam, labels_sam, cost_sam[i], cal_score_sam[i] = my_kmeans(X_per, ki)
    centroids_lst_sam.append(centroids_sam)
    labels_lst_sam.append(labels_sam)
    sil_score_sam[i] = silhouette_score(X_per, labels_lst_sam[i])
    print(f"j={j:.2f} -> cost:{cost_sam[i]:.3f};
        calinski:{cal_score_sam[i]:.3f}; sil:{sil_score_sam[i]:.3f}")

plt.plot(per, sil_score_sam)

### Plot cost vs sample size
plt.figure()
plt.plot(per, cost_sam)
plt.savefig("pca_cost_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Cost")
```

```
plt.show()

### Plot Calinski-Harabasz Index vs sample size
plt.figure()
plt.plot(per, cal_score_sam)
plt.savefig("pca_cal_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Calinski-Harabasz Index")
plt.show()

### Plot Silhouette Coefficient vs smaple size
plt.figure()
plt.plot(per, sil_score_sam)
plt.savefig("pca_sim_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Silhouette Coefficient")
plt.show()

# Different Initialization----------------------

seeds = [21, 75, 84, 12, 51]
ls = len(seeds)
cost_in = np.zeros(ls)
cal_in = np.zeros(ls)
sil_in = np.zeros(ls)
for ij, seed in enumerate(seeds):
    labels_in, cost_in[ij], cal_in[ij] = my_kmeans(X_proj, 10, seed=seed)[1:4]
    sil_in[ij] = silhouette_score(X_proj, labels_in)
    print(f" {seed} -> cost:{cost_in[ij]:.3f};
        cal:{cal_in[ij]:.3f}; sil:{sil_in[ij]:.3f}")

#---------------------------------------------------
#-------------------- Without PCA -------------------
#---------------------------------------------------

# Variation wrt No of Clusters ----------------------

K_w = 15
cost_w = np.zeros(K_w-2)
sil_score_w = np.zeros(K_w-2)
cal_score_w = np.zeros(K_w-2)
centroids_lst_w = []
labels_w = np.zeros((K_w-1, X.shape[0]))

for k in range(2, K_w):
```

```
    centroids_w, labels_w[k-2], cost_w[k-2], cal_score_w[k-2] = my_kmeans(X, k)
    centroids_lst_w.append(centroids_w)
    sil_score_w[k-2] = silhouette_score(X, labels_w[k-2])
    print(f"k={k} -> cost:{cost_w[k-2]}; calinski:{cal_score_w[k-2]};
        sil:{sil_score_w[k-2]}")

# print(f"silhouette_score:{sil_score}\nCost: {cost}\ncalinski_harabasz_score: {cal_score}")
plot_cost(cost_w, K)

plt.figure()
plt.plot(range(2, K_w), cost_w)
plt.savefig("no_pca_cost_vs_clusters.png")
plt.xlabel("No of clusters")
plt.ylabel("Cost")
plt.show()

plt.figure()
plt.plot(range(2, K_w), cal_score_w)
plt.savefig("no_pca_cal_vs_clusters.png")
plt.xlabel("% of total sample")
plt.ylabel("Calinski-Harabasz Index")
plt.show()

plt.figure()
plt.plot(range(2, K_w), sil_score_w)
plt.savefig("no_pca_sil_vs_clusters.png")
plt.xlabel("% of total sample")
plt.ylabel("Silhouette Coefficient")
plt.show()

# Variation wrt No of Samples-----------------------

ki_w = 10
per_w = np.linspace(0.1, 1, num=10)
z_w = per_w.shape[0]
cost_sam_w = np.zeros(z_w)
sil_score_sam_w = np.zeros(z_w)
cal_score_sam_w = np.zeros(z_w)
centroids_lst_sam_w = []
labels_lst_sam_w = []
for i, j in enumerate(per_w):
    idx_w = np.random.choice(range(X.shape[0]), int(j*X.shape[0]), replace=False)
    X_per_w = X[idx_w,:]
#     print(i, j, X_per.shape[0])
    centroids_sam_w, labels_sam_w, cost_sam_w[i], cal_score_sam_w[i] = my_kmeans(X_per_w, ki_w)
```

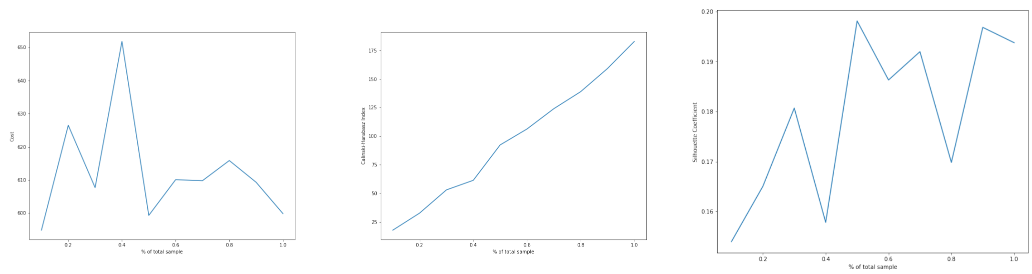
```
    centroids_lst_sam_w.append(centroids_sam_w)
    labels_lst_sam_w.append(labels_sam_w)
    sil_score_sam_w[i] = silhouette_score(X_per_w, labels_lst_sam_w[i])
    print(f"j={j:.2f} -> cost:{cost_sam_w[i]:.3f};
        calinski:{cal_score_sam_w[i]:.3f}; sil:{sil_score_sam_w[i]:.3f}")

plt.figure()
plt.plot(per_w, cost_sam_w)
plt.savefig("no_pca_cost_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Cost")
plt.show()

plt.figure()
plt.plot(per_w, cal_score_sam_w)
plt.savefig("no_pca_cal_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Calinski-Harabasz Index")
plt.show()

plt.figure()
plt.plot(per_w, sil_score_sam_w)
plt.savefig("no_pca_sil_vs_per_sample.png")
plt.xlabel("% of total sample")
plt.ylabel("Silhouette Coefficient")
plt.show()

# Different Initialization---------------------------

seeds_w = [48, 14, 97, 62, 53]
ls_w = len(seeds_w)
cost_in_w = np.zeros(ls_w)
cal_in_w = np.zeros(ls_w)
sil_in_w = np.zeros(ls_w)
for ij, seed in enumerate(seeds_w):
    labels_in_w, cost_in_w[ij], cal_in_w[ij] = my_kmeans(X, 10, seed=seed)[1:4]
    sil_in_w[ij] = silhouette_score(X, labels_in_w)
    print(f" {seed} -> cost:{cost_in_w[ij]:.3f};
        cal:{cal_in_w[ij]:.3f}; sil:{sil_in_w[ij]:.3f}")
```

2 Results

2.1 With PCA

2.1.1 Variation of clustering performance with number of samples

Cost, Calinski-Harabasz Index, and Silhouette Coefficients while varying the sample size are shown in fig. 1a, fig. 1b, and fig. 1c respectively.

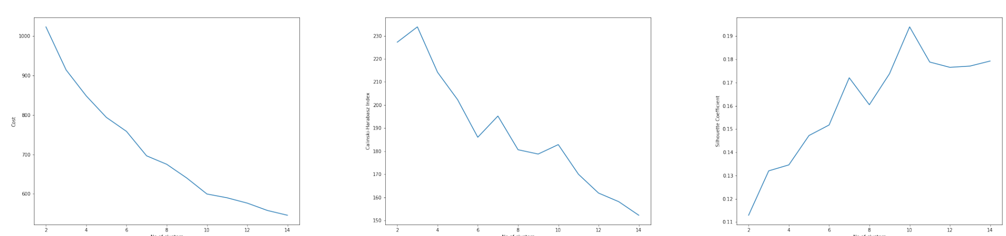


(a) Cost vs no of sample plot (b) Calinski-Harabasz Index vs no of sample (c) Silhouette Coefficient vs no of samples plot

Figure 1: Variation of clustering performance with number of samples (with PCA)

2.1.2 Variation of clustering performance with number of clusters K

Cost, Calinski-Harabasz Index, and Silhouette Coefficients while varying the no of clusters are shown in fig. 2a, fig. 2b, and fig. 2c respectively.



(a) Cost vs no of clusters plot (b) Calinski-Harabasz Index vs no of clusters (c) Silhouette Coefficient vs no of clusters

Figure 2: Variation of clustering performance with number of clusters K (with PCA)

2.1.3 Variation of clustering performance with different initializations of the cluster centers

Different initialization was performed by setting the seed of NumPy pseudo-random number generator (`numpy.random.seed(seed)`) to different numbers. The corresponding performance metrics are shown in table 1.

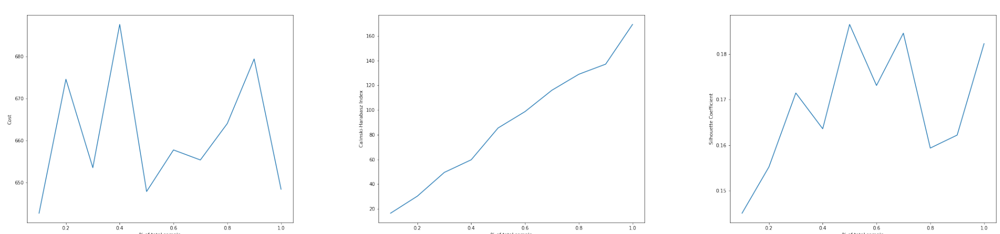
Table 1: Variation of clustering performance with different initializations of the cluster centers (with PCA)

Seed	Cost	Calinski-Harabasz Index	Silhouette Coefficient
21	603.331	180.672	0.195
75	619.554	170.742	0.182
84	643.371	157.071	0.160
12	624.812	167.634	0.179
51	602.507	181.190	0.198

2.2 Without PCA

2.2.1 Variation of clustering performance with number of samples

Cost, Calinski-Harabasz Index, and Silhouette Coefficients while varying the sample size are shown in fig. 3a, fig. 3b, and fig. 3c respectively.



(a) Cost vs no of samples plot (b) Calinski-Harabasz Index vs no of samples (c) Silhouette Coefficient vs no of smaples

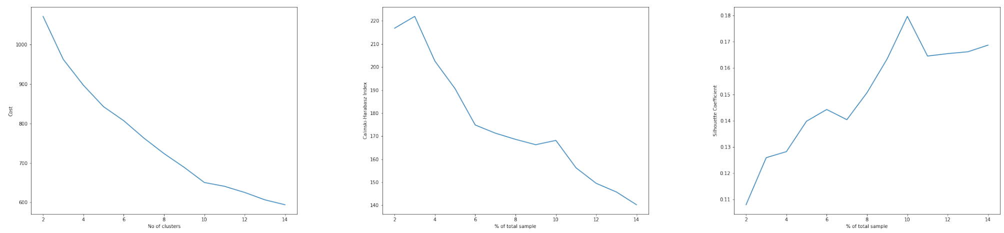
Figure 3: Variation of clustering performance with number of samples (without PCA)

2.2.2 Variation of clustering performance with number of clusters K

Cost, Calinski-Harabasz Index, and Silhouette Coefficients while varying the no of clusters are shown in fig. 4a, fig. 4b, and fig. 4c respectively.

2.2.3 Variation of clustering performance with different initializations of the cluster centers

Different initialization was performed by setting the seed of NumPy pseudo-random number generator (`numpy.random.seed(seed)`) to different numbers. The corresponding performance metrics are shown in table 2.



(a) Cost vs no of clusters plot (b) Calinski-Harabasz Index vs no of clusters (c) Silhouette Coefficient vs no of clusters

Figure 4: Variation of clustering performance with number of clusters K (without PCA)

Table 2: Variation of clustering performance with different initializations of the cluster centers (without PCA)

Seed	**Cost**	**Calinski-Harabasz Index**	**Silhouette Coefficient**
48	650.183	168.357	0.188
14	650.176	168.361	0.188
97	682.802	150.829	0.163
62	650.852	167.980	0.187
53	659.397	163.230	0.173

3 Discussion

In this project, the K-means clustering algorithm was implemented from scratch. Also, principal component analysis (PCA) was used for dimensionality reduction. We investigated the variation in the performances of the K-means algorithm with respect to different aspects.

3.1 Principal Component Analysis

We implemented PCA using the `numpy.linalg.eig()` function which computes the eigenvalues and right eigenvectors of a square matrix. As per the instruction, we also retained 95% energy in the reduced "k" dimensional space. The energy can be calculated by summing the corresponding eigenvalues.

The original dataset had 64 features. To capture 95% of the original energy, we needed only 30 principal features. These 30 features retained 95.48% of the total energy.

3.2 K-Means Clustering

Then we performed K-means clustering on both the given data samples and the reduced featured samples. We varied the no of clusters, sample size and observed three metrics, namely the loss function, Calinski-Harabasz Index and Silhouette Coefficient. We also observed the effect of initialization.

3.2.1 With PCA

At first, we performed K-means clustering on the reduced dimensional space. As we varied the size of the datasize (i.e. no of samples), we found that Calinski-Harabasz Index increases monotonically with no of samples (fig. 1b) and Silhouette Coefficient has an increasing trend (fig. 1c). But the cost function has no regular shape with respect to the no of sample (fig. 1a).

With the increase in no of clusters, we found that the cost function decreases monotonically as expected (fig. 2a). But there is no clear "knee" in the plot from which we may reach a conclusion about the optimal no of clusters. However, the Silhouette Coefficient (fig. 2c) is highest for 10 clusters and also the Calinski-Harabasz Index (fig. 2b) has a higher value at 10 than the neighbors. So, we may conclude that there are 10 clusters in the dataset. This is also supported by our prior knowledge about the dataset (the dataset has 10 different classes).

We also observe the effect of different initialization on the performance metrics (table 1). Depending on the initialization the performance metrics could vary in a large scale. However for our case, the performance metrics didn't vary that much.

3.2.2 Without PCA

Using the full feature space, we found also more or less the same results as with PCA (reduced dimensional space, fig. 3, fig. 4). The performance metrics didn't vary that much. This proves the fact that using PCA, we can achieve similar performances as using the main dataset with the added benefit of reduced computational resources.