# Assignment 6: Unsupervised Learning

***CS 589 - ML***

Shubham Shetty (shubhamshett@umass.edu)
Brinda Murulidhara (bmurulidhara@umass.edu)
Adarsh Kolya (akolya@umass.edu)

*May 2021*

# Preface

Before running our codes, the following packages are imported and test & training data are assigned to variables. Also, some reusable functions are defined -

```
import os

import matplotlib.image as mpimg
import numpy as np

face_data = []
for i in range(100):
    img = mpimg.imread(os.path.join('faces', f'face_{i}.png'))
    img_vector = img.flatten()
    face_data.append(img_vector)

face_data = np.array(face_data, dtype=np.float64)


# Helper function to print table
# Input variable table is a numpy array and headers is an array of
    column headers
def prettyPrintTable(table, headers) :
    for i, d in enumerate(table):
        if i == 0 :
            line = '|'.join(str(x).ljust(30) for x in headers)
            print(line)
            print('-' * len(line))

        line = '|'.join(str(x).ljust(30) for x in d)
        print(line)
    print()
```

# Answer 1

Let,

$$f = \frac{1}{N} \sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2$$

Given $w$ is fixed and we have to find value of $a$ which minimizes the reconstruction error $f$.

To find optimum $a$, we differentiate f and equate it to 0 -

$$\nabla f = \frac{1}{N}(-2w^\top) \sum_{n=1}^{N} (x^{(n)} - a_n w) = 0$$

$$\implies (-w^\top) \sum_{n=1}^{N} (x^{(n)} - a_n w) = 0$$

As each term of sum should equal to 0,

$$\implies a_n w^\top w = w^\top x^{(n)}$$

$$\implies a_n = \frac{w \cdot x^{(n)}}{\|w\|^2}$$

Hence the optimal value of a should be $\frac{w \cdot x^{(n)}}{\|w\|^2}$ for all values of $n = 1, 2, ..., N$.

# Answer 2

Replacing optimal value of $a$ in reconstruction error,

$$f^* = \frac{1}{N}\sum_{n=1}^{N} \|x^{(n)} - (\frac{w \cdot x^{(n)}}{\|w\|^2})w\|^2$$

$$\implies \frac{1}{N}\sum_{n=1}^{N} (x^{(n)} - (\frac{w \cdot x^{(n)}}{\|w\|^2})w)^\top (x^{(n)} - (\frac{w \cdot x^{(n)}}{\|w\|^2})w)$$

# Answer 3

For singular value decomposition (SVD) of $X = USV^\top$, we know that best possible encoder and decoder are of the form -

$$\text{Decoder: } s(\lambda) = V_k\lambda$$

$$\text{Encoder: } t(x) = V_k^\top x$$

where $V_k$ is first k columns of V. Required dimensionality is k = 1. For given reconstruction error we have,

$$\text{Decoder: } s(a_n) = wa_n$$

$$\text{Encoder: } t(x^{(n)}) = w^\top x^{(n)}$$

Therefore comparing above corresponding equations, $w = V_1$.

Covariance matrix given by,

$$\begin{aligned} C &= \frac{1}{N}\sum_{n=1}^{N} x^{(n)}x^{(n)\top} \\ &\implies \frac{1}{N}X^\top X \\ &\implies \frac{1}{N}(USV^\top)^\top(USV^\top) \\ &\implies \frac{1}{N}VS^\top U^\top USV^\top \\ &\implies V\frac{S^2}{N}V^\top \end{aligned}$$

This is equivalent to SVD form for C, $C = V\Sigma V^\top$ where $\Sigma$ is a diagonal matrix where the diagonal elements are the corresponding eigenvalues $\Sigma_{ii} = \sigma_i$. From properties of SVD, we know that $V_1$ corresponds to first eigenvector of C, and $\sigma_1$ corresponds to largest eigenvalue. Therefore the optimal vector w which minimizes the reconstruction error is the eigenvector corresponding to largest eigenvalue of covariance matrix C.

# Answer 4

The minimum number of dimensions we will need in our code such that the data can be reconstructed without error is - D/2.

As half of the data is being modified based on other half, half of the columns are linearly dependent. Hence it can be compressed to D/2 dimensions without losing any information.

# Answer 5

```
import os

import matplotlib
from matplotlib import cm
from sklearn.decomposition import PCA

from preface import face_data


def pca_dim_reduction(data, k):
    pca = PCA(n_components=k)
    pca.fit(data)

    dim_reduced_face_data = pca.transform(data)
    restored_face_data = pca.inverse_transform(dim_reduced_face_data)

    for i in range(100):
        matplotlib.image.imsave(os.path.join('output', f'face_{i}_restored_{k}.png'),
                                restored_face_data[i].reshape(50, 50), cmap=cm.Greys_r)
    return dim_reduced_face_data, pca.components_


for k in [3, 5, 10, 30, 50, 100]:
    pca_dim_reduction(face_data, k)
```

Input -

Generated output -

Figure 1: k = 3

Figure 2: k = 5

Figure 3: k = 10

Figure 4: k = 30

Figure 5: k = 50

Figure 6: k = 100

# Answer 6

```
import numpy as np

from question_5 import pca_dim_reduction
from preface import face_data, prettyPrintTable


def compute_compression_rate(data):
    compression_rate = []
    for k in [3, 5, 10, 30, 50, 100]:
        dim_reduced_face_data, eigen_vectors = pca_dim_reduction(data, k)
        original_data_size = data.size * data.itemsize
        compressed_data_size = dim_reduced_face_data.size * dim_reduced_face_data.itemsize + eigen_vectors.size * eigen_vectors.itemsize
        compression_rate.append((k, compressed_data_size / original_data_size))
    return compression_rate

print(f"Table showing the compression rate for each value of k using PCA")
prettyPrintTable(np.array(compute_compression_rate(face_data)), ['k', 'Compression rate'])
```

| k | Compression rate |
|---|---|
| 3 | 0.0312 |
| 5 | 0.052 |
| 10 | 0.104 |
| 30 | 0.312 |
| 50 | 0.52 |
| 100 | 1.04 |

Table 1: Compression rate

# Answer 7

Run k-means clustering on the dataset for a range of values of k (number of clusters). For each value of k, calculate the sum of squared error. Plot the sum of squared error vs k. The graph resembles an arm and the "elbow" region (point where there is a sharp change) gives the best value of k for the dataset. As k increases, the sum of squared error decreases and becomes 0 when there is one data point in each cluster. The idea behind the elbow method is to choose an optimal value of k that has a low sum of squared error. Increasing k beyond this point would have diminishing returns.

# Answer 8

The K-means++ algorithm is as follows:

- Randomly select the first centroid from the given data points.
- Iterate through the data points and compute the distance of each point from the nearest centroid (chosen in the previous step).
- Select the next centroid such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest centroid. Therefore, the point having maximum distance from the nearest centroid is most likely to be chosen as the next centroid.
- Repeat steps 2 and 3 until you have k centroids

In this approach, centroids that are far away from one another are chosen which increases the chances of choosing centroids that lie in different clusters. Since the centroids are distributed over the data it is more likely to have less within cluster sum of squared error as compared to random initialization (KMeans uses random initialization where results differ based on the initial set of centroids)

## Answer 9

Figure 7: Reconstructed image after applying K-means clustering with k=2

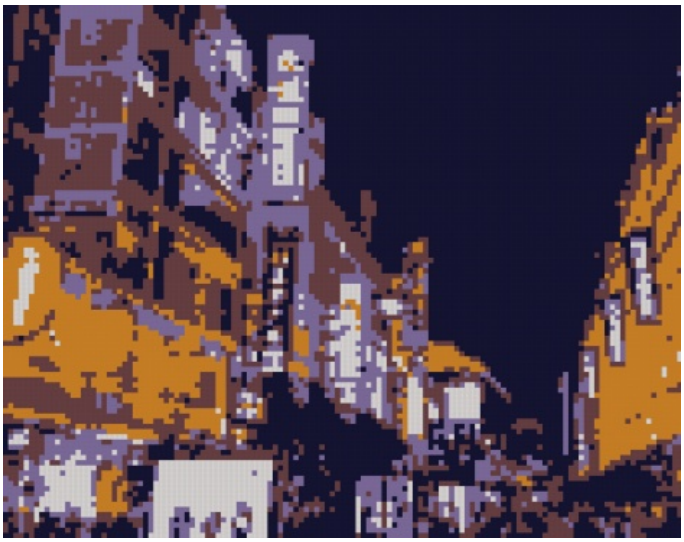

Figure 8: Reconstructed image after applying K-means clustering with k=5

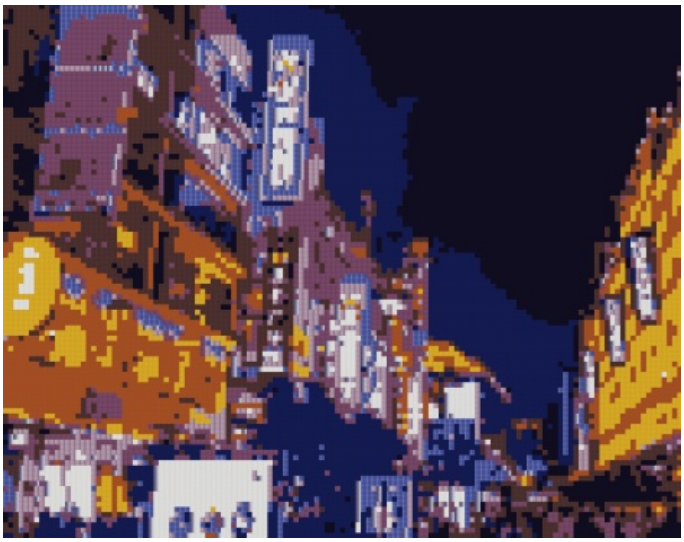

Figure 9: Reconstructed image after applying K-means clustering with k=10

Figure 10: Reconstructed image after applying K-means clustering with k=25

Figure 11: Reconstructed image after applying K-means clustering with k=50

Figure 12: Reconstructed image after applying K-means clustering with k=100

Figure 13: Reconstructed image after applying K-means clustering with k=200

Figure 14: Reconstructed image after applying K-means clustering with k=1000

# Answer 10

| Number of clusters | Reconstruction error |
|---|---|
| 2 | 2634.905822 |
| 5 | 1512.731589 |
| 10 | 1158.213087 |
| 25 | 859.560694 |
| 50 | 708.930815 |
| 100 | 588.945486 |
| 200 | 498.413334 |
| 1000 | 319.706627 |

Table 2: Mean squared difference in intensity, averaged over all pixels, and color channels

# Answer 11

$$\begin{aligned}\text{Total numbers in the compressed representation} &= 27 \times \text{Number of centroids} \\ &= 27 \times \text{Number of clusters}\end{aligned}$$

| **Number of clusters** | **Total numbers in the compressed representation** |
|---|---|
| 2 | 54 |
| 5 | 135 |
| 10 | 270 |
| 25 | 675 |
| 50 | 1350 |
| 100 | 2700 |
| 200 | 5400 |
| 1000 | 27000 |

Table 3: Total numbers in the compressed representation

# Answer 12

$$\text{Compression ratio} = \frac{\text{Total numbers in the compressed representation}}{\text{Total numbers in the original}}$$

| Number of clusters | Compression ratio |
|---|---|
| 2 | 0.0001482250055584377 |
| 5 | 0.0003705625138960942 5 |
| 10 | 0.000741125027792188 5 |
| 25 | 0.001852812569480471 4 |
| 50 | 0.003705625138960942 7 |
| 100 | 0.007411250277921885 5 |
| 200 | 0.014822500555843771 |
| 1000 | 0.07411250277921885 |

Table 4: Compression ratio