ECE M146 - HW 7

May 28, 2020

John Rapp Farnes | 405461225

1

1.1 (a)

As J is a sum of squares, we have $J \geq 0$. For $K = N$, we can choose $r_{ii} = 1 \, \forall i$ and $r_{ij} = 0 \, j \neq i$, and $\mu_i = x_i$. We then get

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||_2^2 = \sum_{n=1}^{N} || \underbrace{x_n - \mu_n}_{=0} ||_2^2 = 0$$

which is the minimum value of J in this case.

1.2 (b)

As J is convex in μ_k, we will obtain the minimum by solving for a stationary point $\frac{\partial J}{\partial \mu_k} = 0$:

$$\begin{aligned} \frac{\partial}{\partial \mu_k} \lambda ||\mu_k||_2^2 + \sum_{n=1}^{N} r_{nk} ||x_n - \mu_k||_2^2 = 0 \quad &\Leftrightarrow \\ 2\lambda \mu_k - 2 \sum_{n=1}^{N} r_{nk} (x_n - \mu_k) = 0 \quad &\Leftrightarrow \\ \mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\lambda + \sum_{n=1}^{N} r_{nk}} \end{aligned}$$

For $\lambda = 0$ we get the original result of μ_k being the sample mean. For $\lambda > 0$, the denominator increases, and hence μ_k will be closer to the origin the larger λ is. In the limit, where $\lambda \to \infty$, all $\mu_k = 0$.

2

2.1 (a)

For $x \neq 0$, $f(x)$ is differentiable, and we have:

$$f(x) = |x| = \begin{cases} x, x > 0 \\ -x, x < 0 \end{cases} \implies$$
$$f'(x) = \begin{cases} 1, x > 0 \\ -1, x < 0 \end{cases} = \text{sign}(x)$$

For $x = 0$, we need to find a g such that $|z| \geq \underbrace{f(0)}_{=0} + gz \; \forall z \in \mathbb{R}$. For $z > 0$, we can write the condition as $g \leq \frac{|z|}{z} = \text{sign}(z) = 1$, and for $z < 0$ we can write the condition as $g \geq \frac{|z|}{z} = \text{sign}(z) = -1$, for $z = 0$ the condition is met by all z. As such, we get $-1 \leq g \leq 1$.

In summary, we have $\partial f(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ -1 \text{ if } x \leq 0 \end{cases}$ and $\partial f(x) \in [-1, 1]$ if $x = 0$

2.2 (b)

For $f(\bar{y})$, we have from (a) that the subgradient at $y_{(N_k+1)/2}$, as y_i are ordered:

$$\begin{aligned}
\partial f(y_{(N_k+1)/2}) &= \partial \Big[\sum_{j=1}^{N_k} |y_j - \bar{y}| \Big] (y_{(N_k+1)/2}) \\
&= \sum_{j=1}^{N_k} \partial \Big[|y_j - \bar{y}| \Big] (y_{(N_k+1)/2}) \\
&= \underbrace{\underbrace{\sum_{j=1}^{(N_k+1)/2-1} \underbrace{\partial \Big[|y_j - \bar{y}| \Big] (y_{(N_k+1)/2})}_{=-1}}_{=-((N_k+1)/2-1)} + \underbrace{\sum_{j=(N_k+1)/2+1}^{N_k} \underbrace{\partial \Big[|y_j - \bar{y}| \Big] (y_{(N_k+1)/2})}_{=1}}_{=((N_k+1)/2-1)}}_{=0} \\
&\quad + \underbrace{\partial \Big[|y_{(N_k+1)/2} - \bar{y}| \Big] (y_{(N_k+1)/2})}_{g \in [-1,1]} \Leftrightarrow \\
g &\in [-1, 1]
\end{aligned}$$

As such, we have that $g = 0$ is a subgradient to $f(\bar{y})$ at $\bar{y} = y_{(N_k+1)/2}$. As such, by the theorem, we have that $y_{(N_k+1)/2}$, or the median, minimizes $f(\bar{y})$

2.3 (c)

The new algorithm becomes:

Choose initial prototypes.

Step 1: Assign each point to the cluster of the prototypes that it has the shortest manhattan distance to.

Step 2: Re-assign the prototypes as the median of the points of the cluster.

Since this algorithm is based on the median rather than the average, it may perform better e.g. when some points, such as outliers, are very spread apart from most of the other points. With traditional *K*-means, these points may end up being their own cluster, whereas they will belong to the closest cluster with this approach.

3

3.1 (a)

The eigenvalues λ must solve the characteristic equation:

$$\begin{aligned}
\det(A - \lambda I) = 0 \quad &\Leftrightarrow \\
-\lambda(3-\lambda) - 4 = 0 &\Leftrightarrow \\
\lambda - 3\lambda - 4 = 0 &\Leftrightarrow \\
(\lambda - \frac{3}{2})^2 - \frac{3}{2}^2 - \frac{16}{4} = 0 &\Leftrightarrow \\
\lambda = \frac{3}{2} \pm \sqrt{\frac{5^2}{2^2}} &\Leftrightarrow \\
\lambda \in \{\underbrace{-1}_{\lambda_1}, \underbrace{4}_{\lambda_2}\}&
\end{aligned}$$

We can then find the corresponding eigenvalues by:

$$\begin{aligned}
Ax_1 = \lambda_1 x_1 \quad &\Leftrightarrow \\
\begin{cases} 4x + 2y = 0 \\ 2x + y = 0 \end{cases} \quad &\Leftrightarrow \\
\begin{cases} y = -2x \\ 0 = 0 \end{cases} \quad &\Leftrightarrow \\
\begin{cases} x = t \\ y = -2t \end{cases} \quad &, t \in \mathbb{R}
\end{aligned}$$

we have $|x_1| = t\sqrt{5}$, as such we get a normalized vector by $x_1 = \frac{1}{\sqrt{5}}(1, -2)^T$
Similarly for λ_2:

$$\begin{aligned}
Ax_2 = \lambda_2 x_2 \quad &\Leftrightarrow \\
\begin{cases} -x + 2y = 0 \\ 2x - 4y = 0 \end{cases} \quad &\Leftrightarrow \\
\begin{cases} x = 2y \\ 0 = 0 \end{cases} \quad &\Leftrightarrow \\
\begin{cases} x = 2t \\ y = t \end{cases} \quad &, t \in \mathbb{R}
\end{aligned}$$

we have $|x_2| = t\sqrt{5}$, as such we get a normalized vector by $x_2 = \frac{1}{\sqrt{5}}(2, -2)^T$

3.2 (b)

Let $\Lambda = \begin{bmatrix} -1 & 0 \\ 0 & 4 \end{bmatrix}$ and $Q = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}$ given by the eigenvalues and corresponding eigenvectors of A.

We then have $A = Q\Lambda Q^{-1}$. Since A is symmetric and orthogonal, we have $Q^{-1} = Q^T$.

As such we have the eigenvalue decomposition:

$$\begin{aligned} A &= \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{-2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \\ &= \frac{1}{5} \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} \\ &= \frac{1}{5} \begin{bmatrix} 15 & 10 \\ 10 & 0 \end{bmatrix} \\ &= A \end{aligned}$$

4

4.1 (a)

We have:

$$\begin{aligned} z^T A z > 0 \quad &\Leftrightarrow \\ 9z_1^2 + 12z_1z_2 + az_2^2 > 0 \quad &\Leftrightarrow \\ 9(z_1 + \frac{12}{9\cdot 2}z_2)^2 - \frac{12^2}{9\cdot 2^2}z_2^2 + az_2^2 > 0 \quad &\Leftrightarrow \\ \underbrace{9(z_1 + \frac{2}{3}z_2)^2}_{\geq 0} + (a-4)z_2^2 > 0 \quad &\Leftrightarrow \\ a > 4 \end{aligned}$$

4.2 (b)

Assume B^{-1} not positive definite. This implies that there exists a $z_0 \neq 0$ such that (using B symmetric and $a^T = a$ if a is scalar):

$$\begin{aligned} \underbrace{z_0^T B^{-1} z_0}_{=z_0(B^{-1})^T z_0^T} \leq 0 \quad &\Leftrightarrow \\ z_0(B^{-1})^T z_0^T \cdot \underbrace{z_0 B^T z_0^T}_{>0} \leq 0 \quad &\Leftrightarrow \\ z_0(B^{-1})^T \underbrace{z_0^T z_0}_{=|z_0|} B^T z_0^T \leq 0 \quad &\Leftrightarrow \\ |z_0| \cdot z_0 B^{-1} B z_0^T \leq 0 \quad &\Leftrightarrow \\ \underbrace{|z_0|}_{>0} \cdot \underbrace{z_0 I z_0^T}_{=z_0^T I z_0 > 0} \leq 0 \quad &\Leftrightarrow \end{aligned}$$

but this is a contradiction as I is positive definite, as such B^{-1} must be positive definite.

4.3 (c)

We have $S = \frac{1}{N}\sum_{i=1}^{N} \tilde{x}_n \tilde{x}_n^T$, hence for $z \neq 0$:

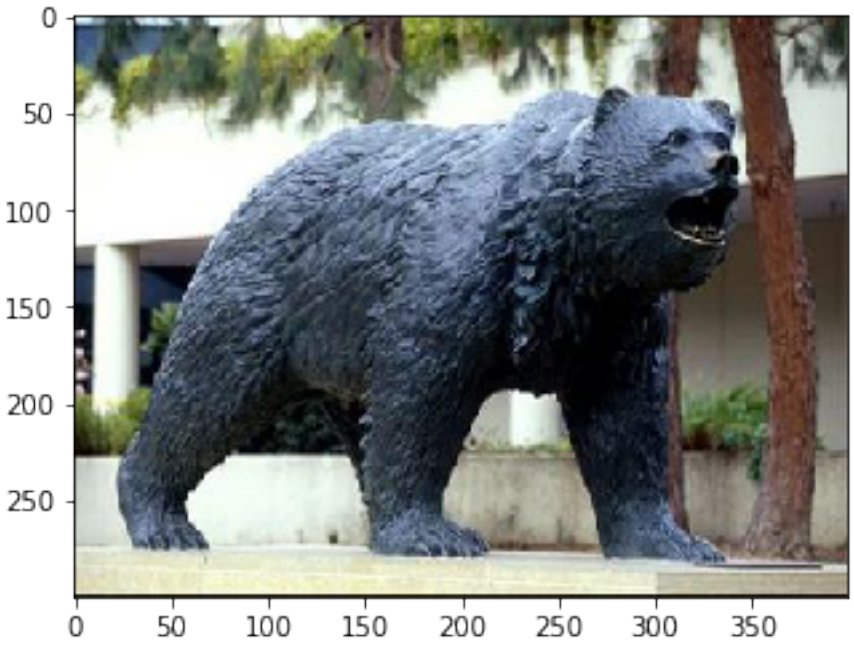
$$\begin{aligned} z^T S z &= z^T \frac{1}{N} \sum_{i=1}^{N} \tilde{x}_n \tilde{x}_n^T z \\ &= \frac{1}{N} \sum_{i=1}^{N} z^T \tilde{x}_n \underbrace{\tilde{x}_n^T z}_{=z^T \tilde{x}_n} \\ &= \frac{1}{N} \sum_{i=1}^{N} (z^T \tilde{x}_n)^2 \\ &\geq 0 \end{aligned}$$

As such, S is positive semi-definite.

5

5.1 (a)

[6]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('UCLA_Bruin.jpg')

plt.imshow(img)
plt.show()
```



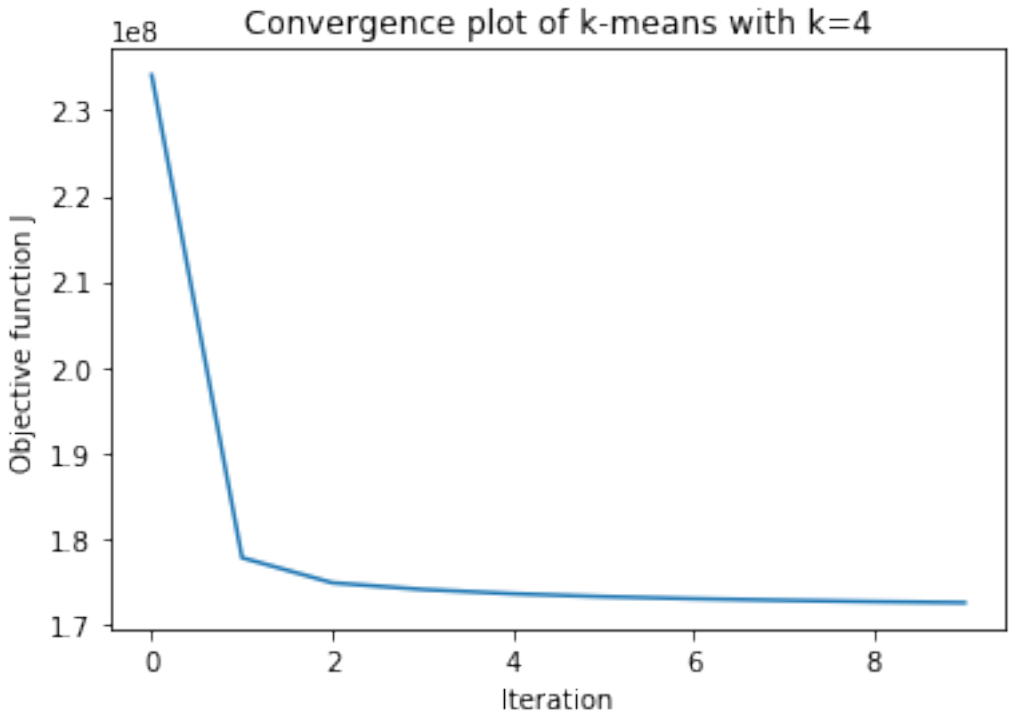
5.2 (b)

[7]:
```
h, w, channels = img.shape

def img_to_array(img):
    return img.reshape(w * h, channels)

def array_to_img(img_data):
    return img_data.reshape(h, w, channels)

img_data = img_to_array(img)
```

[8]:
```
def dist_squared(x, y):
    return (x - y).dot(x - y)

def dists(X, y):
    return [dist_squared(x, y) for x in X]

INITIAL_CENTER = [229, 249, 250]
def initial_centers(data, K):
    centers = [INITIAL_CENTER]

    for k in range(1, K):
        cluster_dists = [dists(img_data, c) for c in centers]
        min_cluster_dists = np.amin(cluster_dists, axis=0)
        next_center = data[np.argmax(min_cluster_dists)]
        centers = np.vstack([centers, next_center])

    return centers
```

[9]:
```
def J(data, centers, clusters):
    return sum([x.dot(x) for x in (data - centers[clusters])])

def k_means_clustering(data, centers, iterations = 10):
    Js = []
    for i in range(iterations):
        # Step 1
        cluster_dists = [dists(img_data, c) for c in centers]
        clusters = np.argmin(cluster_dists, axis=0)

        # Step 2
        centers = np.array([np.mean(img_data[clusters == c], axis=0) for c in␣
 ↪range(len(centers))])

        Js.append(J(data, centers, clusters))

    return centers, clusters, Js
```

```
centers = initial_centers(img_data, 4)
centers, clusters, Js = k_means_clustering(img_data, centers)

plt.plot(Js)
plt.xlabel('Iteration')
plt.ylabel('Objective function J')
plt.title('Convergence plot of k-means with k=4')
plt.show()
```

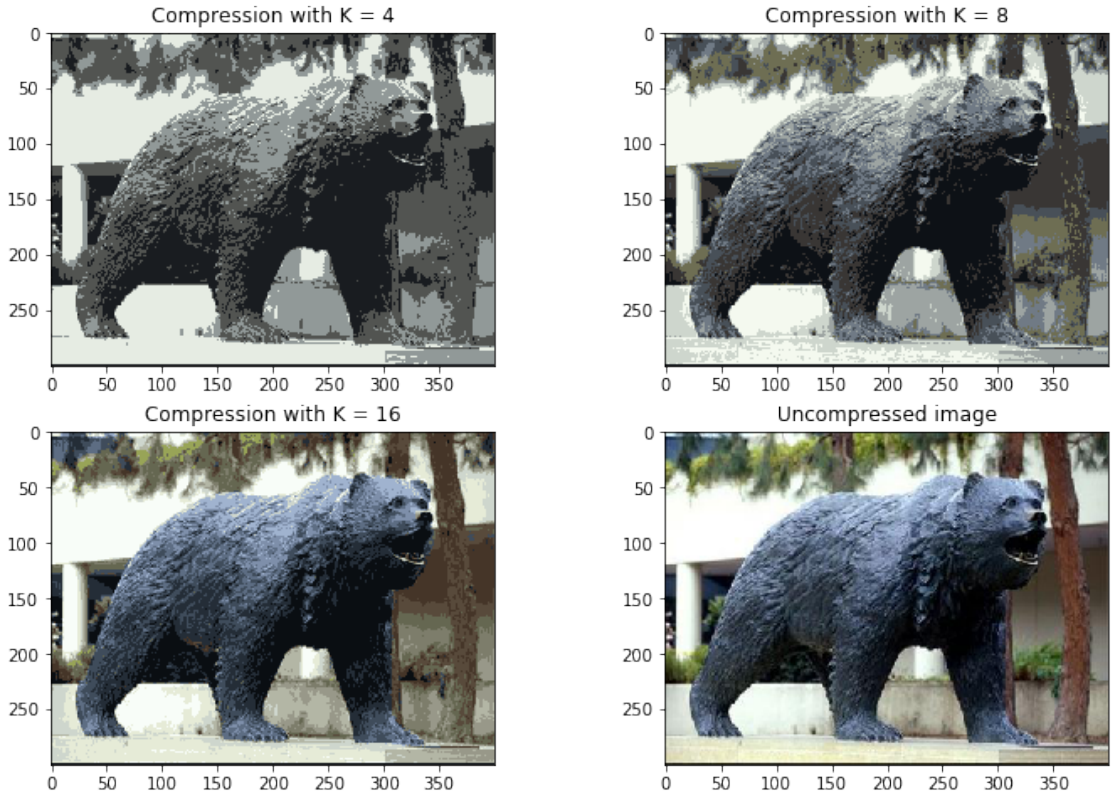


We can see that the *K*-means algorithm has not fully converged in the 10 iterations, however it has reached a lower value of J fairly quickly.

5.3 (c)

[15]:
```
Ks = [4, 8, 16]
i = 0

plt.figure(figsize = (12, 8))
for K in Ks:
    centers = initial_centers(img_data, K)
    centers, clusters, Js = k_means_clustering(img_data, centers)

    print(f'Last J for K = {K} was {Js[-1]}')

    compressed = centers[clusters]

    i = i + 1
    plt.subplot(2, 2, i)
    plt.imshow(array_to_img(np.round(compressed).astype(int)))
    plt.title(f'Compression with K = {K}')

plt.subplot(2, 2, i + 1)
plt.imshow(array_to_img(img_data))
plt.title(f'Uncompressed image')

plt.show()
```

```
Last J for K = 4 was 172523926.2813242
Last J for K = 8 was 81145816.14453024
Last J for K = 16 was 37778195.209877506
```



We can see that a higher K increases the quality of the compressed image, with the best result for $K = 16$. For $K = 16$ the compressed image quality is fairly good and highly resembles the originial uncompressed image in most areas, except for some parts of the background.

5.4 (d)

For the original image we need 8 bits per channel times 3 colors times 300x400 pixels = $8 * 3 * 300 * 400 = 2880000$ bits.

For $K = 4$ we need 8 bits per channel times 3 colors, per the 4 cluster centers. In addition, we can store a value 0-3 for which cluster each belongs to with 2 bits, times 300x400 pixels = $8 * 3 * 4 + 2 * 300 * 400 = 240096$ bits, which gives compression ratio $\frac{240096}{2880000} \approx 8.3\%$.

For $K = 8$ we need 3 bits to store which cluster each pixel belongs to, as such we need $8 * 3 * 8 + 3 * 300 * 400 = 360192$ bits, which gives compression ratio $\frac{360192}{2880000} \approx 12.5\%$.

For $K = 16$ we need 4 bits to store which cluster each pixel belongs to, as such we need $8 * 3 * 16 + 4 * 300 * 400 = 480384$ bits, which gives compression ratio $\frac{480384}{2880000} \approx 16.7\%$.