Hae Sung Jeong
005 146 648

Homework 7

1. (a) $f(x) = |x|$ for $x < 0$, $x > 0$, and $x = 0$.

Since $|x|$ is not differentiable everywhere, we need to use the subgradient.

At first, let $f_1(x) = x$ and $f_2(x) = -x$. Both functions are differentiable, meaning that $f(x) = |x|$ is differentiable where $x > 0$ and $x < 0$.

Therefore, the subgradient of $f(x)$ is equal to its gradient. At $x = 0$, the subgradient of f should satisfy $|z| \geq 0 + g(z - 0) \Rightarrow g \cdot z \leq |z|$

Therefore, $g \in [-1, 1]$

$$f(x) = |x|, \qquad \partial f(x) = \begin{cases} \{1\} & \text{if } x > 0 \\ \{-1\} & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \end{cases}$$

(b) $\bar{y}^*$ is the median of $\{y_1, \dots, y_{N_k}\}$ that minimizes $\sum_{j=1}^{N_k} |y_j - \bar{y}|$

In order to find $\bar{y}^*$ that minimizes $\sum_{j=1}^{N_k} |y_j - \bar{y}|$, we have to take the derivative and set it equal to 0.

However, since the function is the absolute function, we have the similar result as part (a). For this problem, we have $y_1 < y_2 < y_3 < \dots < y_{N_k}$.

Therefore, we can have $\partial f(y) = \partial f_1(y) + \partial f_2(y) + \dots + \partial f_{N_k}(y)$. So if we choose the median of $\{y_1, \dots, y_{N_k}\}$ as $\bar{y}$, then we can have exactly same number of 1 and -1 because the set is already sorted in increasing order and there are odd number of elements in the set. So, when we add those numbers, the results of $\partial f(y)$, we would get 0, which is the minimum value that we can get from this problem. Therefore, $\bar{y}^*$ is the median of $\{y_1, \dots, y_{N_k}\}$ that minimizes $\sum_{j=1}^{N_k} |y_j - \bar{y}|$.

(c) In the k-means algorithm that minimizes J, we first find the u_k and find the closest point so that we can minimize $x_n - u_k$.

In order to find the algorithm that optimize the function $f(u_k) = \sum_{n \in C_k} \|x_n - u_k\|$, we can find x_n that is closest to u_k so that we can find the minimum value as the next step.

2.

(a) $A = \begin{bmatrix} 3 & 2 \\ 2 & 0 \end{bmatrix}$

$\lambda I = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$

$A - \lambda I = \begin{bmatrix} 3 & 2 \\ 2 & 0 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$

$= \begin{bmatrix} 3-\lambda & 2 \\ 2 & -\lambda \end{bmatrix}$

$\det \begin{bmatrix} 3-\lambda & 2 \\ 2 & -\lambda \end{bmatrix}$

$= (3-\lambda)(-\lambda) - 4$

$= \lambda^2 - 3\lambda - 4$

$= (\lambda - 4)(\lambda + 1) = 0$

$\lambda = 4, \quad \lambda = -1$

$\lambda = 4$: $\begin{bmatrix} 3-4 & 2 \\ 2 & -4 \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 2 & -4 \end{bmatrix}$

$\begin{bmatrix} -1 & 2 \\ 2 & -4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\left[\begin{array}{cc|c} -1 & 2 & 0 \\ 2 & -4 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} -1 & 2 & 0 \\ 0 & 0 & 0 \end{array}\right]$ $\quad -v_1 + 2v_2 = 0$

$2R_1 + R_2 \rightarrow R_2$

$v_1 = 2v_2$

Eigenvector: $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

$\|\vec{v}\| = \sqrt{1+4}$

$\vec{u} = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix}$

$\lambda = -1$: $\begin{bmatrix} 3-(-1) & 2 \\ 2 & 0-(-1) \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}$

$\begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\left[\begin{array}{cc|c} 4 & 2 & 0 \\ 2 & 1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 0 & 0 & 0 \\ 2 & 1 & 0 \end{array}\right]$ $\quad 2v_1 + v_2 = 0$

$2R_2 - R_1 \rightarrow R_1$

$2v_1 = -v_2$

Eigenvector: $\begin{bmatrix} 1 \\ -2 \end{bmatrix}$

$\|\vec{v}\| = \sqrt{1+4}$

$\vec{u} = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} \end{bmatrix}$

(b) $A = U \Lambda U^{-1}$ where U is the matrix that each column is eigenvector of A and Λ is the diagonal matrix where diagonal elements are eigenvalues

$U^{-1} = \det \begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} -2 & -1 \\ -1 & 2 \end{bmatrix}$

$\det U = -4 - 1 = -5$

$U^{-1} = \begin{bmatrix} \frac{2}{5} & \frac{1}{5} \\ 1/5 & -2/5 \end{bmatrix}$

$A = \begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix}$

$= \begin{bmatrix} 8 & -1 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 1/3 & 1/6 \\ 1/6 & -1/3 \end{bmatrix}$

$= \begin{bmatrix} 3 & 2 \\ 2 & 0 \end{bmatrix}$

3. (a)

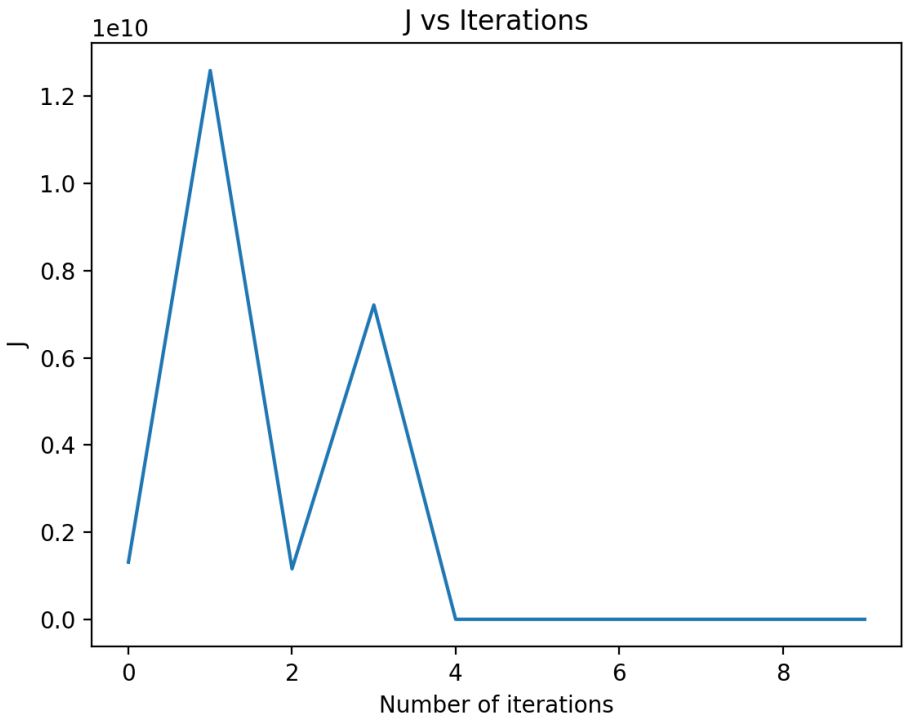
```python
# (a)
import cv2
img = cv2.imread('UCLA_Bruin.jpg')
cv2.imshow('Visualization', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(b)

```
import numpy as np
import cv2
img = cv2.imread('UCLA_Bruin.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

k = 16
u = np.zeros((k, 3))
u[0] = [147, 200, 250]
new_img = img
for i in range(1, k):
    maxD = 0
    max_x = 0
    max_y = 0
    for j in range(len(img)):
        for t in range(len(img[0])):
            diff = []
            for z in range(0, i):
                diff.append(np.linalg.norm(img[j][t] - u[z]))
            if np.min(diff) > maxD:
                maxD = np.min(diff)
                max_x = t
                max_y = j
    u[i] = img[max_y][max_x]

r = np.zeros((300, 400, k))
jj = np.zeros(10)
for it in range(10):
    for j in range(len(img)):
        for t in range(len(img[0])):
            diff = []
            for z in range(k):
                diff.append(np.linalg.norm(img[j][t] - u[z]))
            n = np.argmin(diff)
            r[j][t][n] = 1
    n = np.zeros(3)
    d = 0
    for i in range(k):
        for j in range(len(img)):
            for t in range(len(img[0])):
                if r[j][t][i] == 1:
                    n += img[j][t]
                    d += 1
        u[i] = n / d
    for i in range(k):
        for j in range(len(img)):
            for t in range(len(img[0])):
                if r[j][t][i] == 1:
                    jj[it] += np.linalg.norm((img[j][t] - u[i]), ord=2) ** 2
```

```
# (b)
# u_init = [[147. 200. 250.]
#         [  0.   0.   0.]
#         [137. 141.  57.]
#         [ 31.  70. 125.]]
#
# u = [[203.05552686 216.13657329 216.7668869 ]
#      [149.15837804 158.20136669 155.55279833]
#      [144.18720077 151.69300358 146.28917722]
#      [113.9861382  120.20466861 113.34953944]]
#
# J = [1.31125081e+09 1.25912859e+10 1.15830990e+09 7.21232640e+09
#      0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
#      0.00000000e+00 0.00000000e+00]
```

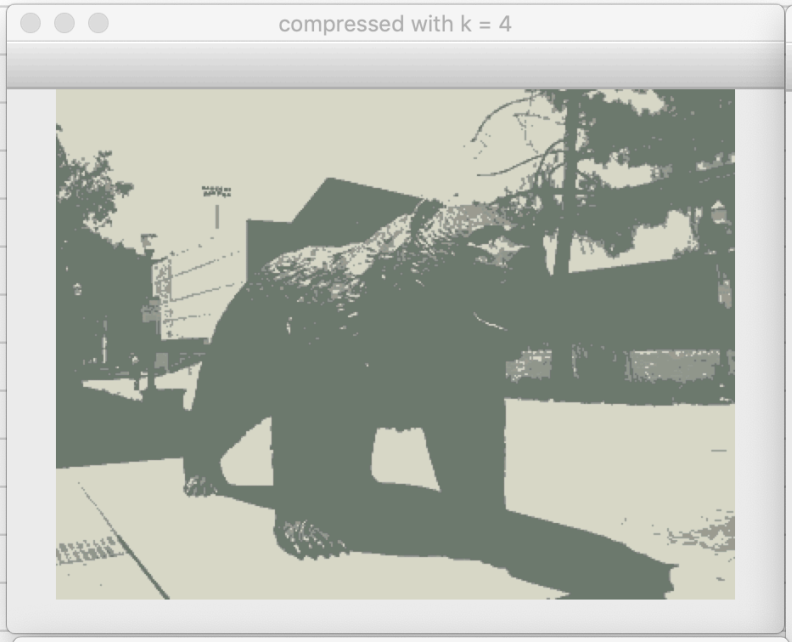
```
print(r)
print(u)
print(jj)
plt.plot(jj)
plt.ylabel("J")
plt.xlabel("Number of iterations")
plt.title("J vs Iterations")
plt.show()
```

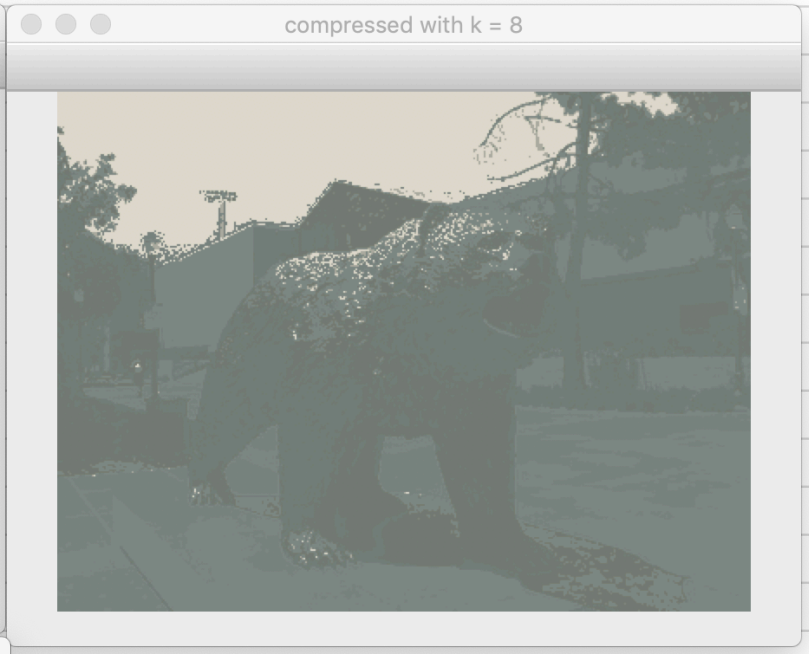


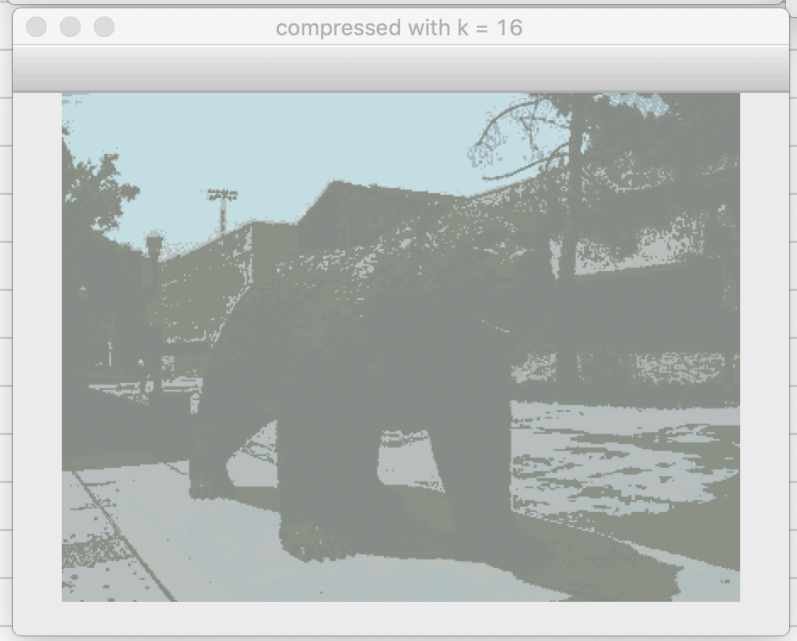
As the number of iterations increases, the k-means objective function converges to 0. We can conclude that from by looking at the graph above. This is because we minimize the function.

(c)

```python
import numpy as np
import cv2
img = cv2.imread('UCLA_Bruin.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

k = 16
u = np.zeros((k, 3))
u[0] = [147, 200, 250]
new_img = img
for i in range(1, k):
    maxD = 0
    max_x = 0
    max_y = 0
    for j in range(len(img)):
        for t in range(len(img[0])):
            diff = []
            for z in range(0, i):
                diff.append(np.linalg.norm(img[j][t] - u[z]))
            if np.min(diff) > maxD:
                maxD = np.min(diff)
                max_x = t
                max_y = j
    u[i] = img[max_y][max_x]

r = np.zeros((300, 400, k))
jj = np.zeros(10)
for it in range(10):
    for j in range(len(img)):
        for t in range(len(img[0])):
            diff = []
            for z in range(k):
                diff.append(np.linalg.norm(img[j][t] - u[z]))
            n = np.argmin(diff)
            r[j][t][n] = 1
    n = np.zeros(3)
    d = 0
    for i in range(k):
        for j in range(len(img)):
            for t in range(len(img[0])):
                if r[j][t][i] == 1:
                    n += img[j][t]
                    d += 1
        u[i] = n / d
    for i in range(k):
        for j in range(len(img)):
            for t in range(len(img[0])):
                if r[j][t][i] == 1:
                    jj[it] += np.linalg.norm((img[j][t] - u[i]), ord=2) ** 2

for i in range(k):
    for j in range(len(img)):
        for t in range(len(img[0])):
            diff = []
            if r[j][t][i] == 1:
                new_img[j][t] = u[i]

print(jj)
new_img = cv2.cvtColor(new_img, cv2.COLOR_RGB2BGR)
cv2.imshow("compressed with k = 16", new_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```







As the K increases, the quality of image gets better. Since the greater number of K can take more colors, the quality of image is getting better as the K increases

(d) In order to store the original image, we only need 351 kB of storage $(300 \times 400) \cdot \frac{3}{1024}$ as 1024 bytes = 1 kB, 1 byte = 8 bit.

For $k = 4$, we will have 2 bits per pixel, thus we would get the size of $(300 * 400 * 2) \frac{1}{8} * \frac{1}{1024} = 29$ kB

For $k = 8$, 3 bits per pixel $\Rightarrow (300 * 400 * 3) \frac{1}{8} * \frac{1}{1024} = 44$ kB

For $k = 16$, 4 bits per pixel $\Rightarrow (300 * 400 * 4) \frac{1}{8} * \frac{1}{1024} = 58$ kB

Compression ratio:

$k = 4$: $\frac{351 \text{ kB}}{29 \text{ kB}} = 12.1$

$k = 8$: $\frac{351 \text{ kB}}{44 \text{ kB}} = 7.98$

$k = 16$: $\frac{351 \text{ kB}}{58 \text{ kB}} = 6.05$