CMSC 25300: Mathematical Foundations of ML

Problem Set 5

Hung Le Tran

11 Nov 2023

Problem 5.1 (Problem 1)

Solution

(a)

$$\begin{aligned}
\mathbf{X} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\
\mathbf{X} &= \sum_{i=1}^{k} \mathbf{u}_i \sigma_i \mathbf{v}_i^T \\
\mathbf{X}^T &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
\mathbf{X}\mathbf{X}^T &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T) \\
&= \mathbf{U}(\mathbf{\Sigma}\mathbf{\Sigma}^T)\mathbf{U}^T \\
\mathbf{X}^T\mathbf{X} &= (\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \\
&= \mathbf{V}(\mathbf{\Sigma}^T\mathbf{\Sigma})\mathbf{V}^T
\end{aligned}$$

(b)(i)

$$\mathbf{X} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 6 \\ 2 \\ 1 \\ 2 \end{bmatrix}$$

The squared error is minimized when

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$
$$\begin{bmatrix} 18 \\ 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 9 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{w}$$

so

$$18 = 9\mathbf{w}_1, 2 = w_2 \Rightarrow \mathbf{w}_1 = \mathbf{w}_2 = 2$$

Therefore the set of $\mathbf{w}$ that minimizes the squared error is

$$W = \left\{ \begin{bmatrix} 2 \\ 2 \\ a \\ b \end{bmatrix} : a, b \in \mathbb{R} \right\}$$

Clearly, the one with the smallest norm has $a = b = 0$, i.e.

$$\hat{\mathbf{w}} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

(b)(ii)

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

The $(n - k + 1)$ last right singular vectors, i.e., $\mathbf{v}_{k+1}, \mathbf{v}_{k+2}, \ldots, \mathbf{v}_n$ form a basis for the null space of $\mathbf{X}$.

We have:

$$\begin{aligned} \mathbf{X}\mathbf{w} &= \sum_{i=1}^{k} \mathbf{u}_i \sigma_i \mathbf{v}_i^T \mathbf{w} \\ &= \sum_{i=1}^{k} \mathbf{u}_i \sigma_i (\mathbf{v}_i^T \mathbf{w}) \end{aligned}$$

Notice that if for any i, $\mathbf{v}_i^T\mathbf{w}$ is non-zero, then since $\{\mathbf{u}_i\}$ are linearly independent, $\mathbf{X}\mathbf{w}$ would then be non-zero.

Therefore $\mathbf{v}_i^T\mathbf{w} = 0 \ \forall \ 1 \leq i \leq k$. The basis of the space of $\mathbf{w}$ that satisfies this is $\{\mathbf{v}_{k+1}, \mathbf{v}_{k+2}, \dots, \mathbf{v}_n\}$, since $\{\mathbf{v}_i\}$ are orthonormal vectors.

(b)(iii)

$$\begin{aligned}
\|\tilde{\mathbf{y}} - \mathbf{\Sigma}\tilde{\mathbf{w}}\|^2 &= \tilde{\mathbf{y}}^T\tilde{\mathbf{y}} - 2\tilde{\mathbf{y}}^T\mathbf{\Sigma}\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T\mathbf{\Sigma}^T\mathbf{\Sigma}\tilde{\mathbf{w}} \\
&= \mathbf{y}^T\mathbf{U}^T\mathbf{U}\mathbf{y} - 2\mathbf{y}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{w} + \mathbf{w}^T\mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T\mathbf{w} \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{w} \\
&= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + (\mathbf{X}\mathbf{w})^T(\mathbf{X}\mathbf{w}) \\
&= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2
\end{aligned}$$

Therefore the set of $\tilde{\mathbf{w}}$ that minimizes the norm above is

$$A = \left\{ \mathbf{V}^T \begin{bmatrix} 2 \\ 2 \\ a \\ b \end{bmatrix} : a, b \in \mathbb{R} \right\}$$

The SVD of our particular X is

$$\mathbf{X} = I \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} I$$

so

$$A = \left\{ I \begin{bmatrix} 2 \\ 2 \\ a \\ b \end{bmatrix} : a, b \in \mathbb{R} \right\} = \left\{ \begin{bmatrix} 2 \\ 2 \\ a \\ b \end{bmatrix} : a, b \in \mathbb{R} \right\}$$

the one with the smallest norm is $\begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$

(b)(iv) Given

$$\tilde{\mathbf{w}} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{w} = I^{-1}\tilde{\mathbf{w}} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

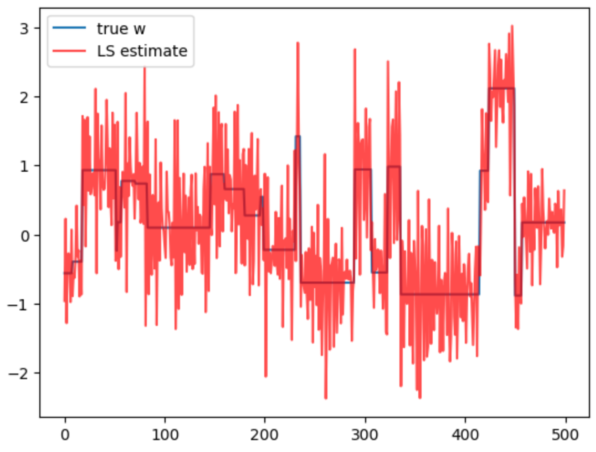
(c)

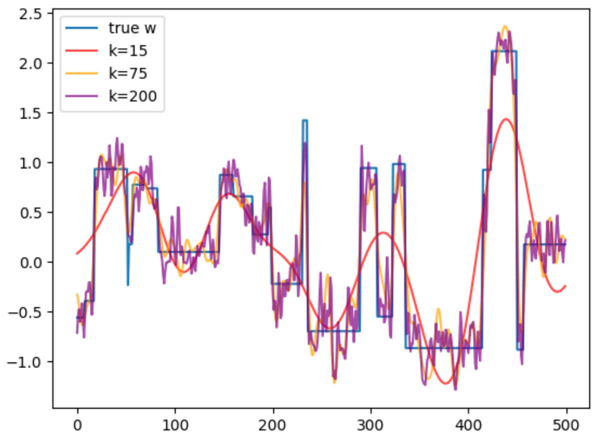
```
import numpy as np

import matplotlib.pyplot as plt

data = np.load("blurring.npz")
X = data['X']
y = data['y']

U, S, Vt = np.linalg.svd(X, full_matrices=False)

V = Vt.T
Sm = np.diag(S)
w_LS = V @ np.linalg.inv(Sm.T @ Sm) @ Sm.T @ U.T @ y

Spseudo = (S > 0) * (1/S)

w_15 = V[:, :15] @ np.diag(Spseudo[:15]) @ U[:, :15].T @ y
w_75 = V[:, :75] @ np.diag(Spseudo[:75]) @ U[:, :75].T @ y
w_150 = V[:, :150] @ np.diag(Spseudo[:150]) @ U[:, :150].T @ y

plt.plot(data['w'], label='true w')
plt.plot(w_LS, alpha=0.7, c='red', label='LS estimate')
plt.legend()
plt.show()

plt.plot(data['w'], label = 'true w')
plt.plot(w_15, alpha=0.7, c='red', label = 'k=15')
plt.plot(w_75, alpha=0.7, c='orange', label = 'k=75')
plt.plot(w_150, alpha=0.7, c='purple', label = 'k=150')

plt.legend()
plt.show()

print(np.linalg.norm(w_LS - data['w']))
print(np.linalg.norm(w_15 - data['w']))
print(np.linalg.norm(w_75 - data['w']))
print(np.linalg.norm(w_150 - data['w']))
```

which prints

```
14.840043525210662
12.003106141836785
6.390503358052758
5.102851020292181
```

and following plots:





(i), (ii) The impact of the added noise is tremendous in the least squares estimation, with high variance in small time increments.

The code printed the following 2-norm between $\mathbf{w}_{LS}$ and true $\mathbf{w}$: 14.840043525210662, which is relatively high when compared to the approximations produced by truncated SVD: 12.003106141836785, 6.390503358052758, 5.102851020292181 respectively. The higher values of k produced better approximates for the true value of $\mathbf{w}$, as seen in the red line for $k = 15$, there wasn't much variation and flexibility, while for $k = 200$, the purple line showed more flexibility and tracked the true value of $\mathbf{w}$ more closely. $\square$

Problem 5.2 (Problem 2)

Solution

(a)

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

and

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

(b)

```
pi = np.array([0.25, 0.25, 0.25, 0.25])

A = np.array([0, 0, 0, 1, 1, 0 , 0, 0, 0, 0.5, 0, 0, 0, 0.5, 1, 0]).
   reshape((4, 4))

for i in range(10000):
    pi = A @ pi
print(pi)
```

which prints

```
[0.28571429 0.28571429 0.14285714 0.28571429]
```

(c)

```
alpha = 0.8

G = alpha * A + (1 - alpha)/4 * np.ones((4, 4))

G = G/G.sum(axis=0)
pi = np.array([0.25, 0.25, 0.25, 0.25])
for i in range(10000):
    pi = G @ pi

print(pi)
```

which prints

```
[0.27967359 0.27373887 0.15949555 0.28709199]
```

(d)

$$M = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1/2 & 1/2 & \cdots & 1/2 \\ 1/2 & 1/2 & \cdots & 1/2 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

so

$$G = \begin{bmatrix} \alpha/2 + (1-\alpha)/n & \alpha/2 + (1-\alpha)/n & \cdots & \alpha/2 + (1-\alpha)/n \\ \alpha/2 + (1-\alpha)/n & \alpha/2 + (1-\alpha)/n & \cdots & \alpha/2 + (1-\alpha)/n \\ (1-\alpha)/n & (1-\alpha)/n & \cdots & (1-\alpha)/n \\ \vdots & \vdots & \vdots & \vdots \\ (1-\alpha)/n & (1-\alpha)/n & \cdots & (1-\alpha)/n \end{bmatrix}$$

To find the PageRank, let $\pi = \begin{bmatrix} x \\ x \\ y \\ \vdots \\ y \end{bmatrix}$ (since the system is symmetric between YouTube and Wikipedia, and between all other links).

We want

$$\pi = G\pi$$

which implies

$$x = \left(\frac{\alpha}{2} + \frac{1-\alpha}{n}\right)(2x + (n-2)y) = \left(\frac{\alpha}{2} + \frac{1-\alpha}{n}\right)$$
$$y = \left(\frac{1-\alpha}{n}\right)(2x + (n-2)y) = \left(\frac{1-\alpha}{n}\right)$$

since π represents a probability distribution, so $2x + (n-2)y = 1$. □

Problem 5.3 (Problem 3)

Solution

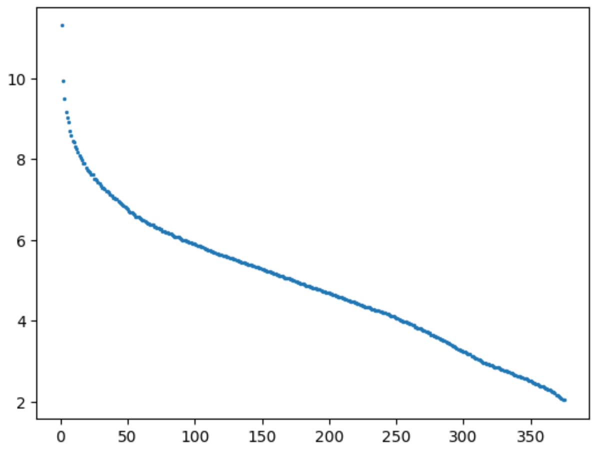
(a)

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image

img = Image.open("disaster-girl.jpg", mode = "r")
img = np.array(img).astype(np.int32)

print(img.shape)
plt.imshow(img)
plt.title("Original Picture")
plt.show()

imgstack = img.reshape((img.shape[0], -1))

U, S, Vt = np.linalg.svd(imgstack)

print(np.log(S)[:10])
plt.scatter(np.arange(len(S)) + 1, np.log(S), s = 2)
```

which prints the top 10 singular values (in log scale)

```
[11.30075264  9.94090064  9.48820164  9.1664354   9.04031758  8.9255915
8.69222982  8.58369855  8.45757526  8.41247316]
```

and plots

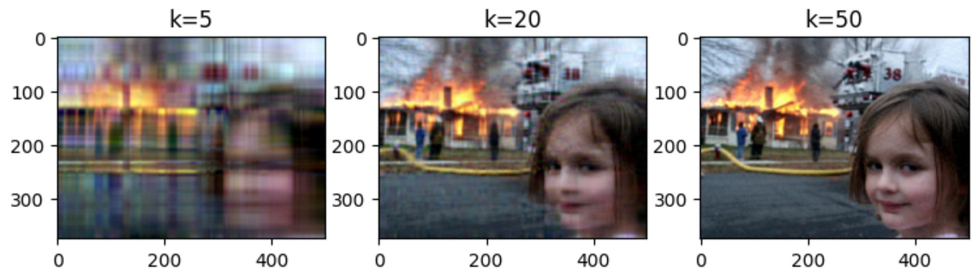


(b)

```
def compress (image, k):
"""
Perform SVD and truncate it, using k singular values/vectors

Params:
image (np.array)

k (int): approximation rank

Returns:
```

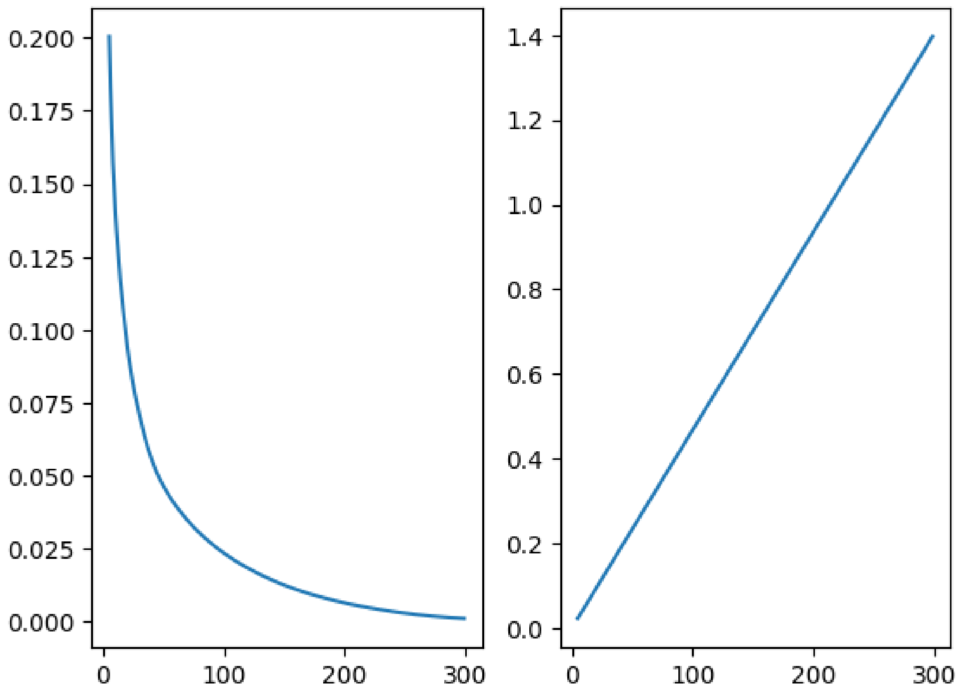
```
reconst_matrix: reconstructed matrix (tensor in colourful case)

s (np.array): array of singular values
"""
reconst_matrix = []
s = []
if (image.ndim == 2):
    print("BW image")
    U, S, Vt = np.linalg.svd(image)
    V = Vt.T
    Uk = U[:, :k]
    Sk = S[:k]
    Vk = V[:, :k]
    reconst_matrix = Uk @ np.diag(Sk) @ Vk.T
    s = Sk
elif (image.ndim == 3):
    r, g, b = image[:, :, 0], image[:, :, 1], image[:, :, 2]

    U, S, Vt = np.linalg.svd(r)
    V = Vt.T
    rr = U[:, :k] @ np.diag(S[:k]) @ V[:, :k].T
    rr = rr.astype(np.int32)
    s.append(S[:k])

    U, S, Vt = np.linalg.svd(g)
    V = Vt.T
    rg = U[:, :k] @ np.diag(S[:k]) @ V[:, :k].T
    rg = rg.astype(np.int32)
    s.append(S[:k])


    U, S, Vt = np.linalg.svd(b)
    V = Vt.T
    rb = U[:, :k] @ np.diag(S[:k]) @ V[:, :k].T
    rb = rb.astype(np.int32)
    s.append(S[:k])

    s = np.array(s).flatten()
    reconst_matrix = np.dstack((rr, rg, rb))

return reconst_matrix, s

figure = plt.figure(figsize=(9, 20))
ax1 = figure.add_subplot(131)
ax2 = figure.add_subplot(132)
ax3 = figure.add_subplot(133)
img5, s5 = compress(img, 5)
img20, s20 = compress(img, 20)
img50, s50 = compress(img, 50)

ax1.imshow(img5)
ax1.set_title('k=5')
ax2.imshow(img20)
ax2.set_title('k=20')
ax3.imshow(img50)
ax3.set_title('k=50')
```

which plots



(c)(i, ii) Working only on the first channel.

```
X = img[:, :, 0]

n, p = X.shape
k_vals = np.arange(5, 300, 1)
err_k = np.zeros(k_vals.shape)
U, S, Vt = np.linalg.svd(X)
V = Vt.T

for i, k in enumerate(k_vals):
    Uk = U[:, :k]
    Sk = S[:k]
    Vk = V[:, :k]
    Xhat = Uk @ np.diag(Sk) @ Vk.T
    err_k[i]= np.linalg.norm(X - Xhat)/np.linalg.norm(X)

figure = plt.figure()
ax1 = figure.add_subplot(121)
ax2 = figure.add_subplot(122)
ax1.plot(k_vals, err_k)
ax2.plot(k_vals, ((n + p + 1) * k_vals)/(n * p))
```



For a certain k, the compression stores $n \times k + k + k \times p = (n + p + 1)k$.

The relative error decreases and plateaus similar to $1/k$ as k increases, while the compression rate increases linearly as k increases. I would choose $k = 150$, to still get around 0.6 compression rate, while keeping relative error relatively low (after the drastic slope in the beginning). $\square$