Homework 2 Exercise Write-up

Jingxuan Sun, js3422
Yue Wang, yw986

October 20, 2018

1 Writing Exercises

1.1 HTF Exercise 4.1

Q:
Show how to solve the generalized eigenvalue problem $\max a^T Ba$ subject to $a^T Wa = 1$ by transforming it to a standard eigenvalue problem.
A:
According to Lagrange multipliers, we have:

$$\mathcal{L}(a) = a^T Ba - \lambda(a^T Wa - 1)$$

Take derivative of a:

$$\frac{d\mathcal{L}}{da} = 2a^T B^T - 2\lambda a^T W^T = 0$$

which leads to:

$$Ba = \lambda Wa$$

This looks like a eigen decomposition question, so we want to transform it into an exact eigen decomposition equation. Let $W = D^T D$, then we have:

$$Ba = \lambda D^T Da$$

which leads to $D^{-T}Ba = D^{-T}D^T\lambda Da$, thus:

$$D^{-T}Ba = \lambda Da$$

Also, let $C = D^{-1}BD^{-1}$, $y = Da$, we finally get:

$$Cy = \lambda y$$

1.2 HTF Exercise 4.2

1.2.1 a

Apply Bayes' theorem to express the posterior probability for class k as follows:

$$Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_l(x)\pi_l}$$

Assume we only have class 1 and class 2, the log odd for class 1 is:

$$\log \frac{Pr(G=1|X=x)}{Pr(G=2|X=x)} = \log \frac{f_1(x)}{f_2(x)} + \log \frac{\pi_1}{\pi_2}$$

Meantime, log odd for class 2 is:

$$\log \frac{Pr(G=2|X=x)}{Pr(G=1|X=x)} = -\log \frac{f_1(x)}{f_2(x)} - \log \frac{\pi_1}{\pi_2} \tag{1}$$

Also, according to Gaussian density function:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)} \tag{2}$$

Thus plug in (2) into (1), we got the log odd of class 2 is:

$$\log \frac{Pr(G=2|X=x)}{Pr(G=1|X=x)} = -\log \frac{\pi_1}{\pi_2} - \frac{1}{2}(\mu_1+\mu_2)^\top \Sigma^{-1}(\mu_2-\mu_1) + x^\top \Sigma^{-1}(\mu_2-\mu_1)$$

If this point is assigned to class 2, then the log odd should be bigger than 0:

$$x^\top \Sigma^{-1}(\mu_2-\mu_1) > \log \frac{\pi_1}{\pi_2} + \frac{1}{2}(\mu_1+\mu_2)^\top \Sigma^{-1}(\mu_2-\mu_1)$$

For the right part:

$$\log \frac{\pi_1}{\pi_2} + \frac{1}{2}(\mu 1+\mu_2)^\top \Sigma^{-1}(\mu_2-\mu_1) = \log \frac{N1}{N} - \log \frac{N2}{N} + \frac{1}{2}(\mu_1+\mu_2)^\top \Sigma^{-1}(\mu_2-\mu_1)$$

$$\log \frac{\pi_1}{\pi_2} + \frac{1}{2}(\mu 1+\mu_2)^\top \Sigma^{-1}(\mu_2-\mu_1) = -\log \frac{N2}{N1} + \frac{1}{2}(\mu_2+\mu_1)^\top \Sigma^{-1}(\mu_2-\mu_1)$$

Thus, for class 2:

$$x^\top \Sigma^{-1}(\mu_2-\mu_1) > -\log \frac{N2}{N1} + \frac{1}{2}(\mu_2+\mu_1)^\top \Sigma^{-1}(\mu_2-\mu_1)$$

Same for class 1, which we got:

$$x^\top \Sigma^{-1}(\mu_2-\mu_1) < -\log \frac{N2}{N1} + \frac{1}{2}(\mu_2+\mu_1)^\top \Sigma^{-1}(\mu_2-\mu_1)$$

1.2.2 b

To minimize least square, we start from the normal equation

$Y = X\beta$ where X = $\begin{bmatrix} 1 & x_1 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$ and $\beta = \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix}$

$$X^T Y = X^T X\beta$$

$$\mathrm{X}^T X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^T & x_2^T & \dots & x_n^T \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^N x_i^T \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i x_i^T \end{bmatrix}$$

$$\mathrm{X}^T Y = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^T & x_2^T & \dots & x_n^T \end{bmatrix} \begin{bmatrix} -\frac{N}{N_1} \\ -\frac{N}{N_1} \\ \vdots \\ -\frac{N}{N_2} \end{bmatrix}$$
$$= \begin{bmatrix} N_1\left(-\frac{N}{N_1}\right) + N_2\left(\frac{N}{N_2}\right) \\ \sum_{i=1}^{N_1} x_i^T\left(-\frac{N}{N_1}\right) + \sum_{i=1}^{N_2} x_i^T\left(\frac{N}{N_2}\right) \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix}$$

From the definition of covariance matrix:
$\hat{\sum} = \frac{1}{N-K}\sum_{k=1}^{K}\sum_{g_i=k}(x_i-\mu_k)(x_i-\mu_k)^T$
$= \frac{1}{N-2}\left(\sum_{i=1}^{N}(x_i-\mu_1)(x_i-\mu_1)^T + (x_i-\mu_2)(x_i-\mu_2)^T\right)$
$= \frac{1}{N-2}\left(\sum_{i=1}^{N}\left(x_i x_i^T + \mu_1\mu_1^T\right) + \sum_{i=1}^{N}\left(x_i x_i^T + \mu_2\mu_2^T\right)\right)$
$= \frac{1}{N-2}\left(\sum_{i=1}^{N} x_i x_i^T - \left(N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T\right)\right)$

We get Equation (1):
$\sum_{i=1}^{N} x_i x_i^T = (N-2)\hat{\sum} + N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T$

Plug into the matrix derived from the original model:
$$\begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix} = \begin{bmatrix} N & N_1\mu_1^T + N_2\mu_2^T \\ N\mu_1 + N\mu_2 & (1) \end{bmatrix}$$
$$\left(N_1\mu_1^T + N_2\mu_2^T\right)\left(-\frac{N}{N_1}\mu_1 - \frac{N}{N_2}\mu_2\right)\beta + (1)\beta = N(\mu_2 - \mu_1)$$

From the coefficient of , we get:
$(N-2)\hat{\sum} + \frac{N_1N_2}{N}\left(\mu_1\mu_1^T - 2\mu_1\mu_2 - \mu_1\mu_2^T\right)$
$= \frac{N_1N_2}{N}(\mu_1-\mu_2)(\mu_1-\mu_2)^T$
$= \frac{N_1N_2}{N}\hat{\sum}\beta$

Thus:
$\left[(N-2)\hat{\sum} + \frac{N_1N_2}{N}\hat{\sum}\beta\right]\beta = N(\mu_2-\mu_1)$

1.2.3 c

$\hat{\sum}\beta = (\mu_2-\mu_1)(\mu_2-\mu_1)^T\beta$
and the product $(\mu_2-\mu_1)^T\beta$ is a scalar. Therefore the vector direction of $\hat{\sum}\beta$ is given by $\mu_2-\mu_1$. Thus in the final equation as both the right-hand-side and the term $\frac{N_1N_2}{N}\hat{\sum}\beta$ are in the direction of $\mu_2-\mu_1$ the solution must be in the direction (i.e. proportional to) $\hat{\sum}^{-1}\beta(\mu_2-\mu_1)$.

1.3 SVD of Rank Deficient Matrix

Please refer to Jupyter Notebook output.
$M^\top M$ and $MM^\top$ were calculated by dot product.
Normalized eigenvalues and eigenvectors of both matrices were calculated with np.linalg.eigh function. The column v[:, i] is the normalized eigenvector corre-

sponding to the eigenvalue w[i].

In [1]:
```
import numpy as np

M = np.asarray([[1,0,3],[3,7,2],[2,-2,8],[0,-1,1],[5,8,7]])
MTM = np.dot(M.T, M)
MMT = np.dot(M, M.T)

print(MTM)
print(MMT)
```

```
[[ 39  57  60]
 [ 57 118  53]
 [ 60  53 127]]
[[ 10   9  26   3  26]
 [  9  62   8  -5  85]
 [ 26   8  72  10  50]
 [  3  -5  10   2  -1]
 [ 26  85  50  -1 138]]
```

In [2]:
```
# w = eigenvalues, v = eigenvectors
w1, v1 = np.linalg.eigh(MTM)
w2, v2 = np.linalg.eigh(MMT)
print(w1)
print(v1)
print(w2)
print(v2)
```

```
[8.12918346e-15 6.93295108e+01 2.14670489e+02]
[[ 0.90453403  0.01460404  0.42615127]
 [-0.30151134  0.72859799  0.61500884]
 [-0.30151134 -0.68478587  0.66344497]]
[-2.86858803e-14  1.28575407e-15  7.86020103e-15  6.93295108e+01
  2.14670489e+02]
[[ 0.95187257 -0.02656868  0.07758291  0.24497323 -0.16492942]
 [ 0.09661006  0.01964755 -0.74989503 -0.45330644 -0.47164732]
 [-0.24036664  0.20388528 -0.315392    0.82943965 -0.33647055]
 [-0.06054744 -0.97446897 -0.13386333  0.16974659 -0.00330585]
 [-0.1521939  -0.0880289   0.56057723 -0.13310656 -0.79820031]]
```

From the above step we can observe the two largest eigenvalues are 6.93295108e+01 and 2.14670489e+02, thus the diagonal matrix s = [[6.93295108e+01 0], [0 2.14670489e+02]].
We sliced out the corresponding columns from v1 and v2 to construct u and v thus done the SVD.

In [4]:
```
# The column v[:, i] is the normalized eigenvector corresponding to the eigenvalue w[i]
# from the above step we can observe:
# s = [[6.93295108e+01 0]
#      [0 2.14670489e+02]]

# u = v2
# v = v1
# s = w1

s = np.diag(np.sqrt(w1[1:]))
u = v2[:, 3:]
vt = v1[:, 1:].T

print(u)
print(s)
print(vt)
```

```
[[ 0.24497323 -0.16492942]
 [-0.45330644 -0.47164732]
 [ 0.82943965 -0.33647055]
 [ 0.16974659 -0.00330585]
 [-0.13310656 -0.79820031]]
[[ 8.32643446  0.        ]
 [ 0.         14.65163776]]
[[ 0.01460404  0.72859799 -0.68478587]
 [ 0.42615127  0.61500884  0.66344497]]
```

Keeping the largest eigenvalue, we further reduced the singular value matrices to rank-1 and reconstructed the rank-1 approximation xr of original matrix M.
Because the eigenvalues and eigenvectors were normalized in former steps, we normalized the output and orginal matrix too for easy comparison. There's still great disparity between the approximation and the original matrix.

In [5]:
```
# set s[0] = 0

xr = np.linalg.multi_dot((u[:, 1:], s[1:, 1:], vt[1:, :]))
print(xr)
```

```
[[-1.02978864 -1.48616035 -1.60320558]
 [-2.94487812 -4.24996055 -4.58467382]
 [-2.10085952 -3.031898   -3.27068057]
 [-0.02064112 -0.02978864 -0.0321347 ]
 [-4.9838143  -7.19249261 -7.75895028]]
```

In [6]:
```
from sklearn.preprocessing import normalize
# M_norm = normalize(M, axis=1, norm='l1')
# xr_norm = normalize(xr, axis=1, norm='l1')
M_norm = M / np.linalg.norm(M, axis=-1)[:, np.newaxis]
xr_norm = xr / np.linalg.norm(xr, axis=-1)[:, np.newaxis]
print(M_norm)
print(xr_norm)
```

```
[[ 0.31622777  0.          0.9486833 ]
 [ 0.38100038  0.88900089  0.25400025]
 [ 0.23570226 -0.23570226  0.94280904]
 [ 0.         -0.70710678  0.70710678]
 [ 0.42562827  0.68100522  0.59587957]]
[[-0.42615127 -0.61500884 -0.66344497]
 [-0.42615127 -0.61500884 -0.66344497]
 [-0.42615127 -0.61500884 -0.66344497]
 [-0.42615127 -0.61500884 -0.66344497]
 [-0.42615127 -0.61500884 -0.66344497]]
```