1. EECS 545 Homework 2. Yuan Yin

1) (a) Note that log-likelihood of λ in Gamma pdf is:

$$l(\lambda) = \sum_{i=1}^{n} \log\left(\frac{1}{\Gamma(\alpha)}\lambda^{\alpha} x_i^{\alpha-1} \exp(-\lambda x_i)\right)$$

Suppose $a_i := \frac{1}{\Gamma(\alpha)} x_i^{\alpha-1}$ (α is known.)

$$\Rightarrow l(\lambda) = \sum_{i=1}^{n} \log(a_i \cancel{*} \lambda^{\alpha} e^{-\lambda x_i})$$

$$= \sum_{i=1}^{n} (\log a_i + \alpha \log\lambda - \lambda x_i).$$

$\frac{\partial l(\lambda)}{\partial \lambda} \overset{!}{=} 0$ to maximize $l(\lambda)$, $\forall\lambda$

$$\Rightarrow \frac{\partial l(\lambda)}{\partial \lambda} = \frac{n\alpha}{\lambda} - \sum_{i=1}^{n} x_i \overset{!}{=} 0.$$

$\Rightarrow$. $\lambda = \frac{n\alpha}{\sum_{i=1}^{n} x_i}$ is the maximum likelihood estimate of λ

(b). suppose $a := \frac{1}{\sqrt{(2\pi)^d |\Sigma|}}$ is known

$$l(\vec{\mu}) = \sum_{i=1}^{n} \log f(\vec{x}_i; \vec{\mu}, \Sigma)$$

$$= \sum_{i=1}^{n} \left[\log a + \left(-\frac{1}{2}(\vec{x}_i - \vec{\mu})^T \Sigma^{-1} (\vec{x}_i - \vec{\mu})\right)\right]$$

$\frac{\partial l(\vec{\mu})}{\partial \vec{\mu}} \overset{!}{=} 0$ to maximize $l(\vec{\mu})$. $\forall\vec{\mu}$

$$\Rightarrow \frac{\partial l(\vec{\mu})}{\partial \vec{\mu}} = \frac{\partial\left(-\frac{1}{2}\sum_{i=1}^{n}\left[(\vec{x}_i - \vec{\mu})^T \Sigma^{-1} (\vec{x}_i - \vec{\mu})\right]\right)}{\partial \vec{\mu}}$$

$$= \Sigma^{-1}\left(\sum_{i=1}^{n} (\vec{x}_i - \vec{\mu})\right) \overset{!}{=} 0.$$

$\Rightarrow$. $\vec{\mu} = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i$ is the maximum likelihood estimate of $\cancel{\vec{x}}\ \vec{\mu}$

2) is coding problem (submitted with .py file)

3). $J(\vec{\theta}) = -l(\vec{\theta}) + \lambda \|\vec{\theta}\|^2$

Compute $\dfrac{\partial l(\vec{\theta})}{\partial \theta_j} = \sum_{i=1}^{n} \left[y_i \left(-\log(1+e^{-\vec{\theta}^T \tilde{x}_i})\right)' + [1-y_i]\left(-\vec{\theta}^T \tilde{x}_i - \log(1+e^{-\vec{\theta}^T \tilde{x}_i})\right)' \right]$

$$= \sum_{i=1}^{n} \left[y_i \cdot \frac{e^{-\vec{\theta}^T \tilde{x}_i} \cdot x_i^{(j)}}{1+e^{-\vec{\theta}^T \tilde{x}_i}} + (1-y_i)\left(-\tilde{x}_i^{(j)} + \frac{e^{-\vec{\theta}^T \tilde{x}_i} \cdot \tilde{x}_i^{(j)}}{1+e^{-\vec{\theta}^T \tilde{x}_i}}\right) \right]$$

$$= \sum_{i=1}^{n} \left[y_i \tilde{x}_i^{(j)} - \tilde{x}_i^{(j)} \cdot \frac{1}{1+e^{-\vec{\theta}^T \tilde{x}_i}} \right]$$

Compute $\dfrac{\partial^2 l(\vec{\theta})}{\partial \theta_j \partial \theta_k} = -\sum_{i=1}^{n} \left[\tilde{x}_i^{(j)} \cdot \dfrac{e^{-\vec{\theta}^T \tilde{x}_i}}{(1+e^{-\vec{\theta}^T \tilde{x}_i})^2} \cdot \tilde{x}_i^{(k)} \right]$

(notice $j \neq k$, $j = k$)

Thus. $\nabla J(\vec{\theta}) = \dfrac{-\partial l(\vec{\theta})}{\partial \vec{\theta}} + 2\lambda \vec{\theta}$ where $\dfrac{\partial l(\vec{\theta})}{\partial \theta_j}$ is as above

$\nabla^2 J(\vec{\theta}) = -\dfrac{\partial l^2(\vec{\theta})}{\partial \vec{\theta} \partial \vec{\theta}} + 2\lambda I$ where $\dfrac{\partial^2 l(\vec{\theta})}{\partial \theta_j \partial \theta_k}$ is as above

Since we know

$J(\vec{\theta})$ is convex $\Leftrightarrow$ $\nabla^2 J(\vec{\theta})$ is PSD

$J(\vec{\theta})$ is strictly convex $\Leftrightarrow$ $\nabla^2 J(\vec{\theta})$ is PD

all we need to show is $\nabla^2 J(\vec{\theta})$ is PD or PSD

for $\forall \vec{z} \geq \vec{0}$ $\vec{z} \in \mathbb{R}^{d+1}$ ~~$\hat{X}$ is $n \times (d+1)$ matrix $\eta(\hat{x})$ is $1 \times n$ matrix~~

$$z^T \nabla^2 J(\vec{\theta}) z = z^T \left[\tilde{X}^T \eta(\tilde{x})^T (1-\eta(\tilde{x})) \hat{X} \right] z + n\lambda z^T z$$

(where $\tilde{X}$ is $n \times (d+1)$ matrix, $\eta(\hat{x})$ is $1 \times n$ matrix, $\eta(\hat{x}) = \dfrac{1}{1+e^{-\vec{\theta}^T \hat{x}}}$)

$$= (\hat{X} z)^T \begin{pmatrix} \eta_1(1-\eta_1) & & 0 \\ & \ddots & \\ 0 & & \eta_n(1-\eta_n) \end{pmatrix} (\hat{X} z) + n\lambda z^T z.$$

Suppose $A := \tilde{X} z$ is a $n \times 1$ matrix.

Then $z^T \nabla^2 J(\vec{\theta}) z = \sum_{i=1}^{n} \eta_i (1-\eta_i) A_i^2 + n\lambda \sum_{j=1}^{n} z_j^2$

since $\eta_i \in (0,1)$. $\Rightarrow \eta_i(1-\eta_i) > 0$.

$\Rightarrow z^T \nabla^2 J(\vec{\theta}) z \geq 0$, ($\lambda > 0$ then $z^T \nabla^2 J(\vec{\theta}) z > 0$ PD)

is PSD

2.

5). First claim that

$$X_{n\times(d+1)} = \begin{bmatrix} 1 & x_{1}^{(1)} & \cdots & x_{1}^{(d)} \\ 1 & \vdots & & \\ \vdots & \vdots & & \\ 1 & x_{n}^{(1)} & \cdots & x_{n}^{(d)} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\vec{\theta} = \begin{bmatrix} b \\ \vec{w} \end{bmatrix} \in \mathbb{R}^{d+1}. \quad C = \begin{bmatrix} C_1 & & 0 \\ & \ddots & \\ 0 & & C_n \end{bmatrix}$$

Thus, the obj. fun becomes:

$$\sum_{i=1}^{n} C_i (y_i - X_i\theta)^2$$

$$= (\vec{y} - X\theta)^T C (\vec{y} - X\theta).$$

$$= (\vec{y}^T C - (X\theta)^T C)(\vec{y} - X\theta).$$

$$= \vec{y}^T C \vec{y} - \underbrace{\vec{y}^T C (X\theta)} - \underbrace{(X\theta)^T C \vec{y}} + (X\theta)^T C X\theta$$

scalar. equal.

$$= \vec{y}^T C \vec{y} - 2(X\theta)^T C \vec{y} + (X\theta)^T C X \theta$$

$$= \theta^T X^T C X \theta - 2\vec{y}^T C X \theta + \vec{y}^T C \vec{y}.$$

Suppose $A = 2X^T C X \quad r = -2X^T C \vec{y} \quad d = \vec{y}^T C \vec{y}.$

Then $J(\theta) = \frac{1}{2}\theta^T A \theta + r^T \theta + d.$ which is the same form with OLS,

The alternate solution:

$$\vec{\theta} = (X^T C X)^{-1} X^T C \vec{y}$$

Sanity check. if $C = I$. then $\vec{\theta} = (X^T X)^{-1} X^T \vec{y}$

EECS 545 Homework 2

Yuan Yin

October 4, 2018

Problem 2

Code and result is as follows:

```
import numpy as np

# # Process the data
z = np.genfromtxt('spambase.data', dtype = float, delimiter = ',')
np.random.seed(0) # Seed the random number generator
rp = np.random.permutation(z.shape[0]) # random permutation of indices
z = z[rp,:] # shuffle the rows of z
x = z[:,:-1]
y = z[:,-1]

# Quantize variables with option 2 where values equal to the median to 1
row_num,col_num = x.shape; y = list(y)
x_train = x[0:2000,:]
y_train = y[0:2000]
x_test = x[2000:row_num,:]
y_test= y[2000:row_num]

mid_train = np.median(x_train,axis=0)
for i in range(2000):
    for j in range(col_num):
        if x_train[i,j] > mid_train[j]:
            x_train[i,j] = 2
        else:
            x_train[i,j] = 1

# Build up Naive Bayes Classifier
## compute the probability of y=0,1 and the conditional probability of x_j=1,2 given y=0,1
n_1 = sum(y_train); n_0 = len(y_train) - n_1
pi_1 = n_1/len(y_train); pi_0 = 1-pi_1 # the pmf of y
count_x_equ_1_y_equ_0 = np.zeros(col_num)
```

```
count_x_equ_1_y_equ_1 = np.zeros(col_num)
count_x_equ_2_y_equ_0 = np.zeros(col_num)
count_x_equ_2_y_equ_1 = np.zeros(col_num)
for l in range(len(y_train)):
    if y_train[l] == 0:
        for m in range(col_num):
            if x_train[l,m] == 1:
                count_x_equ_1_y_equ_0[m] += 1
            elif x_train[l,m] == 2:
                count_x_equ_2_y_equ_0[m] += 1
    elif y_train[l] == 1:
        for n in range(col_num):
            if x_train[l,n] == 1:
                count_x_equ_1_y_equ_1[n] += 1
            elif x_train[l,n] == 2:
                count_x_equ_2_y_equ_1[n] += 1
p_yto0_j_xto1 = np.zeros(col_num)
p_yto1_j_xto1 = np.zeros(col_num)
p_yto0_j_xto2 = np.zeros(col_num)
p_yto1_j_xto2 = np.zeros(col_num)
for i in range(col_num):
    p_yto0_j_xto1[i] = count_x_equ_1_y_equ_0[i]/n_0
    p_yto0_j_xto2[i] = count_x_equ_2_y_equ_0[i]/n_0
for i in range(col_num):
    p_yto1_j_xto1[i] = count_x_equ_1_y_equ_1[i]/(len(y_train)-n_0)
    p_yto1_j_xto2[i] = count_x_equ_2_y_equ_1[i]/(len(y_train)-n_0)

# Test data
## quantize the test data with median of training data, and compute the test result
for i in range(x_test.shape[0]):
    for j in range(x_test.shape[1]):
        if x_test[i,j] > mid_train[j]:
            x_test[i,j] = 2
        else:
            x_test[i,j] = 1

y_test_result = np.zeros(x_test.shape[0])
for i in range(x_test.shape[0]):
    y0 = 1; y1 = 1
    for j in range(x_test.shape[1]):
        if x_test[i,j] == 1:
            y0 = y0*p_yto0_j_xto1[j]
            y1 = y1*p_yto1_j_xto1[j]
        elif x_test[i,j] == 2:
```

```
            y0 = y0*p_yto0_j_xto2[j]
            y1 = y1*p_yto1_j_xto2[j]
    if (pi_0*y0) >= (pi_1*y1):
        y_test_result[i] = 0
    else:
        y_test_result[i] = 1

## Test error
error = 0
for i in range(len(y_test)):
    if y_test[i] != y_test_result[i]:
        error += 1
print("The test error of spam emails by Naive Bayes classifier is %f." %(error/len(y_test)) )

## Sanity check
major = 1-y_train.count(1)/len(y_train) # In the training data, the major class for emails
    is "not spam", so we assume to predict all emails are not spam emails.
error_sanity = y_test.count(1)/len(y_test)
print("The sanity check error is %f." %(error_sanity))
```

the result is:

```
The test error of spam emails by Naive Bayes classifier is 0.105344.
The sanity check error is 0.386774.
```

Problem 4

the code and the result is as follows:

```
from numpy import *
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt

# Import the data
mnist_49_3000 = sio.loadmat('mnist_49_3000.mat')
x = mnist_49_3000['x']
y = mnist_49_3000['y']
d,n = x.shape
y += (y < 0) * 1

# Process the data, we divide the data into training part and test part
added = np.ones(n)
```

```
x_original = x; y_original = y
x = mat(np.vstack((added,x))); y = mat(y)
x_train = x[:,:2000]; y_train = y[:,:2000]
x_test = x[:,2000:]; y_test= y[:,2000:]

# Initialize parameters
lamda = 10
theta = mat(np.zeros(d + 1))
dJ = mat(np.zeros(d + 1))
d2J = mat(np.zeros([d + 1,d + 1]))

# Iteration with Newton's Method:
## Notice here the stop condition for iteration is when the change of theta is less than 1%
N = 0; error0 = 10
while (error0 > 0.01):
   z = 1 / (1 + np.exp(-theta*x_train))
   dJ = x_train*(z-y_train).T+2*lamda*theta.T
   d2J = x_train * mat(diag(multiply(z, (1-z)).getA()[0])) * x_train.T +
       2*lamda*mat(np.eye(d+1,d+1))
   error0 = np.sqrt(((d2J.I * dJ).T * (d2J.I * dJ))[0,0])
   theta = theta - (d2J.I * dJ).T
   y_train_result = theta * x_train
   for l in range(2000):
      if y_train_result[0,l] < 0:
         y_train_result[0,l] = 0
      else:
         y_train_result[0,l] = 1
   N = N+1
print("Iteration times: ", N)
log_like = 0
for l in range(2000):
   z = 1 / (1 + np.exp(-theta * x_train[:,l]))
   log_like += y_train[0,l]*np.log(z)+(1-y_train[0,l])*np.log(1-z)
J = -log_like + lamda*theta*theta.T
print("Value of objective function is: ", J[0,0])

# Test data
y_test_result = theta * x_test
eta_test = 1/(1+np.exp(-y_test_result))
false = []
for l in range(1000):
   if y_test_result[0, l] < 0:
      y_test_result[0, l] = 0
   else:
```

```
        y_test_result[0, l] = 1
error1 = 0
for m in range(1000):
    if y_test_result[0, m] != y_test[0, m]:
        error1 += 1
        false.append(m)
print("Test error is: ", error1/1000)
prob = np.zeros(1000)
x_false = x_test[1:,false]; x_false = x_false.getA()
y_false = y_test[:,false]
y_false_result = y_test_result[:,false]
for l in range(1000):
    if y_test_result[0,l] == 0:
        prob[l] = 1-eta_test[0,l]
    else:
        prob[l] = eta_test[0,l]
confidence = prob[false]
indx = argsort(confidence)[28:]
x_false20 = np.zeros((d,20)); y_real20 = np.zeros(20); y_pre20 = np.zeros(20)
for l in range(20):
    x_false20[:,l] = x_false[:,int(indx[l])]
    y_real20[l] = y_false[:,int(indx[l])]
    y_pre20[l] = y_false_result[:,int(indx[l])]

## Plot the picture of 20 most confident missclassified pictures
fig = plt.figure(num='missclassified',figsize=(8,8))
fig.suptitle("\"True\" represents real result, \"Pre\" represents predicted result\n")
for l in range(20):
    plt.subplot(4,5,1+l)
    if y_real20[l] == 0:
        true_title = 4
        pre_title = 9
    else:
        true_title = 9
        pre_title = 4
    plt.title('True: %s, Pre: %s' %(str(true_title), str(pre_title)))
    plt.imshow(np.reshape(x_false20[:,l], (int(np.sqrt(d)), int(np.sqrt(d)))))
plt.show()
plt.close()
```

the result is:

```
Iteration times: 6
```

```
Value of objective function is: 307.65546352880267
Test error is: 0.048
```

"True" represents real result, "Pre" represents predicted result