STA414 Homework 1

Yubo Wang 1002138377

1.

1.1

a.

X and Y are independent, so $E[XY^T] = E(X)E(Y^T)$

$cov(X,Y) = E\left[\left(X - E(X)\right)\left(Y - E(Y)\right)^T\right] = E[XY^T] - E[X]E[Y^T] = E[X]E[Y^T] - E[X]E[Y^T] = 0$

b.

For each value x_i, y_i in X and Y

$E(x_i + y_i) = E(x_i) + E(y_i)$ for all integer $i \in [1, m]$

We can get $E(X+Y) = E(X) + E(Y)$

For a mxm matrix A, j-th number for AX is $\sum_{i=1}^{m} A_{ji} x_i$

$E\left(\sum_{i=1}^{m} A_{ji} x_i\right) = \sum_{i=1}^{m} A_{ji}\, E(x_i)$

We can get $E(AX) = AE(X)$

$Let\ W = AX(AX)^T$

$W_{ij} = \sum_{k=1}^{m} \sum_{l=1}^{m} A_{il} x_l x_k A_{kj}$

$E(W) = AE(x_l x_k)A^T$

We can conclude that $E(X + AY) = E(x) + AE(Y)$

$Var(X+AY) = Var(X) + Var(AY) + 2Cov(X, AY) = Var(X) + E(AYY^TA^T) - E(AY)E(AY)^T$

$= Var(X) + AE(YY^T)A^T - AE(Y)E(Y)^TA^T = Var(X) + AVar(Y)A^T$

We can conclude that $Var(X + AY) = Var(X) + AVar(Y)A^T$

c.

AX is normally distributed since X is normally distributed

Then we have $E(AX) = AE(X)$

$Var(AX) = AVar(X)A$

$X \sim N(\mu, \Sigma)$ implies $AX \sim N(A\mu, A\Sigma \mathrm{A}^{\mathrm{T}})$

1.2

a.

Yes, the uniform distribution on the interval [0,0.5] has probability density f(x)=2 for 0⩽x⩽0.5 and f(x)=0 elsewhere.

b.

$f(x) = \frac{1}{\sqrt{\frac{\pi}{50}}} e^{-50x^2}$

c.

At 0 , the value of this pdf is $\frac{10}{\sqrt{2\pi}}$

d.

$P(x = 0) = \int_0^0 f(x)dx = 0$

1.3

a.

$\frac{\partial}{\partial \mathrm{x}} = x^T y = \mathrm{y}^{\mathrm{T}}$

b.

$\frac{\partial}{\partial \mathrm{x}} = 2x^T$

c.

$\frac{\partial}{\partial \mathrm{x}} = x^T(A^T + A)$

d.

$\frac{\partial}{\partial \mathrm{x}} = A$

2.

2.1

a.

$\hat{\beta} = (X^T X)^{-1} X^T Y$

$\mathrm{Y}|\mathrm{X}, \beta \sim \mathrm{N}(\mathrm{X}\beta, \sigma^2 I)$

$\hat{\beta}$ is normally distributed

$\mathrm{E}(\hat{\beta}) = E((X^T X)^{-1} X^T Y) = (\mathrm{X}^{\mathrm{T}} X)^{-1} X^T E(Y) = (X^T X)^{-1} X^T X\beta = \beta$

$\mathrm{Var}(\hat{\beta}) = Var((X^T X)^{-1} X^T Y) = (X^T X)^{-1} X^T Var(Y)((X^T X)^{-1} X^T)^T = \sigma^2 (X^T X)^{-1}$

b.

$\mathrm{L}(\mathrm{Y}|\mathrm{X}, \beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - x_i\beta)^2}{2\sigma^2}} = (2\pi\sigma^2)^{-\frac{n}{2}} e^{\sum_{i=1}^{n} -\frac{(y_i - x_i\beta)^2}{2\sigma^2}}$

$\mathrm{l}(\mathrm{Y}|\mathrm{X}, \beta) = \log(\mathrm{L}) = -\frac{\mathrm{n}}{2}\log(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{2\sigma^2}$

$\frac{\partial}{\partial\beta}\mathrm{l}(\mathrm{Y}|\mathrm{X}, \beta) = \frac{\sum_{i=1}^{n} x_i^T y_i - x_i^T x_i \beta}{\sigma^2} = \frac{\mathrm{X}^{\mathrm{T}} Y - X^T X\beta}{\sigma^2}$

c.

$\mathrm{P}(|\widehat{\beta_\iota} - \beta_i| \le \varepsilon) = P(\beta_i - \varepsilon \le \widehat{\beta_\iota} \le \varepsilon + \beta_i) = \mathrm{F}(\varepsilon + \beta_i) - \mathrm{F}(\beta_i - \varepsilon)$ where F(t) is the cdf of $\hat{\beta}$

2.2

a.

$\mathrm{P}(\beta|\mathrm{Y}) = \frac{\mathrm{P}(\mathrm{Y}|\beta)\mathrm{P}(\beta)}{\mathrm{P}(\mathrm{Y})}$ based on bayes rule

$\mathrm{P}(\mathrm{Y}|\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - x_i\beta)^2}{2\sigma^2}}$ where x_I is the i-th row of x

$\mathrm{P}(\beta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\beta)^2}{2\tau^2}}$

Since $\mathrm{P}(\beta|\mathrm{Y}) \propto \mathrm{P}(\mathrm{Y}|\beta)\mathrm{P}(\beta)$

We have $\mathrm{argmax}(\mathrm{P}(\beta|\mathrm{Y})) = \mathrm{argmax}(\mathrm{P}(\mathrm{Y}|\beta)\mathrm{P}(\beta))$

By expanding $\mathrm{P}(\mathrm{Y}|\beta)\mathrm{P}(\beta)$ we have

$(2\pi\sigma^2)^{-\frac{n}{2}} * (2\pi\sigma^2)^{-\frac{1}{2}} e^{-\sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{2\sigma^2} - \frac{(\beta)^2}{2\tau^2}}$

$\mathrm{argmax}(\mathrm{P}(\mathrm{Y}|\beta)\mathrm{P}(\beta)) = \mathrm{argmin}\left(\sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{2\sigma^2} + \frac{(\beta)^2}{2\tau^2}\right)$

To get result of argmin, we set derivative to 0

$\frac{\partial}{\partial\beta} = \sum_{i=1}^{n} -x_i^T \frac{y_i - x_i\beta}{\sigma^2} + \frac{\beta}{\tau^2} = 0$

$\sum_{i=1}^{n} -x_i^T \frac{y_i - x_i\beta}{\sigma^2} = -\frac{\beta}{\tau^2}$
$\sum_{i=1}^{n} x_i^T (y_i - x_i\beta) = \frac{\sigma^2\beta}{\tau^2}$
$\sum_{i=1}^{n} x_i^T (y_i - x_i\beta) = \lambda\beta$
$X^T Y - X^T X\beta = \lambda\beta$
$(X^T X + \lambda I)\beta = X^T Y$
$\beta = \left(X^T X + \lambda I\right)^{-1} X^T Y$

b.

We have $X^* = \begin{bmatrix} & X & \\ \sqrt{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda} \end{bmatrix}$ $Y^* = \begin{bmatrix} & Y & \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$
$X^{*T} X^* = X^T X + \lambda I$
$X^{*T} Y^* = X^T Y$
MLE of β
$P(Y|\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - x_i\beta)^2}{2\sigma^2}}$
Same as part a, we can try to minimize $\sum_{i=1}^{n} (y_i - x_i\beta)^2$
$\frac{\partial}{\partial\beta} = \sum_{i=1}^{n} -2x_i^T (y_i - x_i\beta) = 0$
$-X^T Y + X^T X\beta = 0$
$\beta = (X^T X)^{-1} X^T Y$

Substitute X, Y with X^*, Y^*, we have
$\beta = (X^{*T} X^*)^{-1} X^{*T} Y^* = (X^T X + \lambda I)^{-1} X^T Y$
which is same as $\hat{\beta}$ in ridge regression.

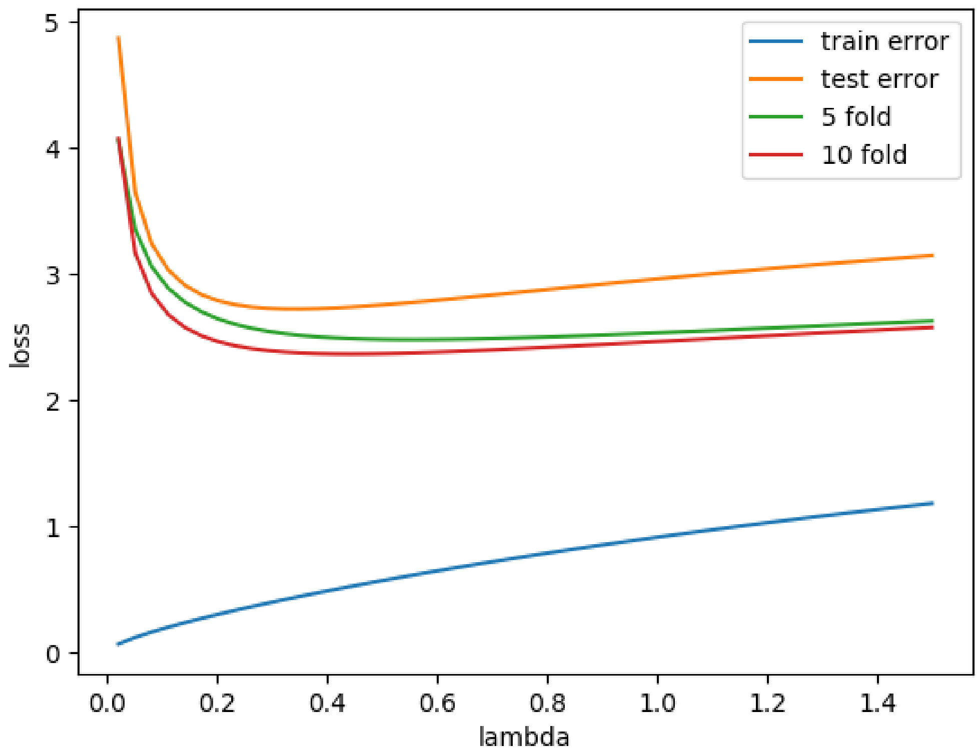
2.3
Code:

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

# part a  load in mat data
dataset = sio.loadmat('./dataset.mat')
data_train_X = dataset['data_train_X']
data_train_y = dataset['data_train_y'][0]
data_test_X = dataset['data_test_X']
data_test_y = dataset['data_test_y'][0]


# use random.shuffle to get random permutation of data
def shuffle_data(data):
    temp = data[:]
    random.shuffle(temp)
    return temp


def split_data(data, num_folds, fold):
```

```
    # get each split's size
    size = int(len(data) / num_folds)
    total = []
    index = 0
    for i in range(num_folds):
        total.append(data[index:(index+size)])
        index += size
    # get specified data_fold based on index fold
    data_fold = total[fold-1]
    total.pop(fold-1)
    data_rest = []
    for i in total:
        data_rest.extend(i)
    return data_fold, data_rest


def train_model(data, lambd):
    # create correct size matrices for X and Y
    y, x = np.empty([1, len(data)]), np.empty([len(data), 400])
    for i in range(len(data)):
        y[0][i] = data[i][0]
        x[i] = data[i][1]
    I = np.identity(400)
    beta = np.dot(np.dot(np.linalg.inv((np.dot(x.transpose(), x) + np.dot(lambd, I))), x.transpose()), y.transpose())
    return beta


def predict_model(data, model):
    res = []
    for d in data:
        res.append(np.dot(d[1].reshape([1, 400]), model.reshape([400, 1])))
    return res


# get summation of all differences then divided by number of value
def loss(data, model):
    prediction = predict_model(data, model)
    real, diff = [], []
    for i in data:
        real.append(i[0])
    for i in range(len(real)):
        diff.append((real[i] - prediction[i]) ** 2)
    return sum(diff)/len(diff)


def cross_validation(data, num_folds, lambd_seq):
    data = shuffle_data(data)
    cv_error = []
    for i in range(50):
        lambd = lambd_seq[i]
        cv_loss_lmd = 0
        for fold in range(1, num_folds+1):
            val_cv, train_cv = split_data(data, num_folds, fold)
            model = train_model(train_cv, lambd)
```

```
            cv_loss_lmd += loss(val_cv, model)
        cv_error.append(cv_loss_lmd / num_folds)
    return cv_error


# part b
total_train = []
for i in range(len(data_train_X)):
    total_train.append((data_train_y[i], data_train_X[i]))

total_test = []
for i in range(len(data_test_X)):
    total_test.append((data_test_y[i], data_test_X[i]))

lambd_seq = np.linspace(0.02, 1.5, 50)
cv_5_err = cross_validation(total_train, 5, lambd_seq)
cv_10_err = cross_validation(total_train, 10, lambd_seq)


# part c
def compute_loss(train_data, test_data, lambd_seq):
    train_loss, test_loss  = [], []
    for i in lambd_seq:
        model = train_model(train_data, i)
        train_loss.append(loss(train_data, model))
        test_loss.append(loss(test_data, model))
    return train_loss, test_loss


train_error, test_error = compute_loss(total_train, total_test, lambd_seq)

# part d


def clip_data(data):
    res = []
    for i in data:
        res.append(i[0][0])
    return res

def plot_graph():
    x = np.linspace(0.02, 1.5, 50)
    plt.plot(x, clip_data(train_error), label="train error")
    plt.plot(x, clip_data(test_error), label="test error")
    plt.plot(x, clip_data(cv_5_err), label="5 fold")
    plt.plot(x, clip_data(cv_10_err), label="10 fold")
    plt.xlabel('lambda')
    plt.ylabel('loss')
    plt.legend()
    plt.show()

plot_graph()
```

Graph:


train error
test error
5 fold
10 fold
loss
lambda
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4