STA414 Assignment 1

Zikun Chen

January 29, 2019

1. Probability and Calculus

1.1 Variance and Covariance

(a) Let $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$ and $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$

Since they are independent, $\mathbb{E}(x_i y_j) = \mathbb{E}(x_i)\,\mathbb{E}(y_j)$

By definition,

$$\begin{aligned} Cov(X, Y) &= \mathbb{E}(XY^T) - \mathbb{E}(X)\,\mathbb{E}(Y)^T \\ &= \mathbb{E}(\sum_{i=1}^{m} x_i y_i) - \sum_{i=1}^{m} \mathbb{E}(x_i)\,\mathbb{E}(x_j) \\ &= \sum_{i=1}^{m} \mathbb{E}(x_i y_i) - \sum_{i=1}^{m} \mathbb{E}(x_i)\,\mathbb{E}(x_j) \\ &= \sum_{i=1}^{m} \mathbb{E}(x_i)\,\mathbb{E}(y_i) - \sum_{i=1}^{m} \mathbb{E}(x_i)\,\mathbb{E}(x_j) \\ &= 0 \end{aligned}$$

(b) Let $A = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix}$

$$\mathbb{E}(X+AY) = \mathbb{E}(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix})$$

$$= \mathbb{E}(\begin{bmatrix} x_1 + \sum_{j=1}^m y_1 a_{1j} \\ \vdots \\ x_i + \sum_{j=1}^m y_i a_{ij} \\ \vdots \\ x_m + \sum_{j=1}^m y_m a_{im} \end{bmatrix}) = \mathbb{E}(X) + \mathbb{E}(\begin{bmatrix} \sum_{j=1}^m a_{1j}\, \mathbb{E}(y_1) \\ \vdots \\ \sum_{j=1}^m a_{ij}\, \mathbb{E}(y_i) \\ \vdots \\ \sum_{j=1}^m y_m a_{im}\, \mathbb{E}(y_m) \end{bmatrix})$$

$$= \mathbb{E}(X) + A\,\mathbb{E}(Y)$$

Since X and Y independent and A is constant

$$\begin{aligned} Var(X+AY) &= Var(X) + Var(AY) \\ &= Var(X) + Cov(AY, AY) \\ &= Var(X) + \mathbb{E}(AY[AY]^T) - E(AY)E(AY)^T \\ &= Var(X) + \mathbb{E}(AYY^TA^T) - AE(Y)(AE(Y))^T \\ &= Var(X) + A\,\mathbb{E}(YY^T)A^T - AE(Y)E(Y)^TA^T \\ &= Var(X) + A(\mathbb{E}(YY^T) - E(Y)E(Y)^T)A^T \\ &= Var(X) + ACov(Y,Y)A^T \\ &= Var(X) + AVar(Y)A^T \end{aligned}$$

(c) Since AX is a linear transformation of X. AX is a Gaussian distribution with mean and covariance matrix as follows:
Since $\mathbb{E}(X) = \mu$ and $Var(X) = \Sigma$
By part b), $\mathbb{E}(AX) = AE(X) = A\mu$ and $Var(AX) = A\Sigma A^T$
Therefore, $AX \sim \mathcal{N}(A\mu, A\Sigma A^T)$

1.2 Densities

(a) It is certainly possible that a probability density function takes values greater than 1.

(b) Normal distribution density function with mean μ and variance σ^2:

$$\frac{1}{\sigma\sqrt{2\pi}} exp\{-\frac{x-\mu^2}{2\sigma^2}\}$$

X be a uni-variate normally distributed random variable with mean 0 and variance 1/100. It's density is of the form:

$$p_X(x) = \frac{1}{\frac{1}{10}\sqrt{2\pi}} exp\{-\frac{x^2}{2\frac{1}{100}}\} = \frac{10}{\sqrt{2\pi}} e^{-50x^2} = \sqrt{\frac{50}{\pi}} e^{-50x^2}$$

(c) $p_X(0) = \sqrt{\frac{50}{\pi}}$

(d) $P(X = 0) = \int_0^0 p_X(x)dx = 0$

1.3 Calculus

(a)

$$x^T y = \sum_{i=1}^{m} x_i y_i$$

$$\frac{\partial x^T y}{\partial x} = (\frac{\partial x^T y}{\partial x_1}, \cdots, \frac{\partial x^T y}{\partial x_m})^T = (y_1, \cdots, y_m)^T = y$$

(b)

$$x^T x = \sum_{i=1}^{m} x_i^2$$

$$\frac{\partial x^T x}{\partial x} = (\frac{\partial x^T x}{\partial x_1}, \cdots, \frac{\partial x^T x}{\partial x_m})^T = (2x_1, \cdots, 2x_m)^T = 2x$$

(c)

$$x^T A x = \sum_{i=1}^{m} \sum_{j=1}^{m} a_{ij} x_i x_j$$

For a particular $k \in \mathbb{Z}, k \in [1, m]$

$$\frac{\partial x^T A x}{\partial x_k} = \sum_{i=1}^{m} a_{ik} x_i x_k + \sum_{j=1}^{m} a_{kj} x_k x_j$$

$$\frac{\partial x^T A x}{\partial x} = (\frac{\partial x^T A x}{\partial x_1}, \cdots, \frac{\partial x^T A x}{\partial x_m})^T = (\sum_{i=1}^{m}(a_{1i}+a_{i1})x_i, \cdots, \sum_{i=1}^{m}(a_{mi}+a_{im})x_i)^T = (A+A^T)x$$

(d)

$$Ax = \begin{bmatrix} \sum_{i=1}^{m} a_{1i} x_i \\ \vdots \\ \sum_{i=1}^{m} a_{mi} x_i \end{bmatrix}$$

$$\frac{\partial Ax}{\partial x} = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix} = A$$

2. Regression

2.1 Linear Regression

(a) **Distribution:** Since $(X^TX)^{-1}X^T$ is a matrix, $\hat{\beta} = (X^TX)^{-1}X^TY$ is a linear transformation of Y. Therefore, $\hat{\beta}$ follows Normal distribution with the following mean and variance.

Mean:

$$\mathbb{E}(\hat{\beta}) = \mathbb{E}((X^TX)^{-1}X^TY) = (X^TX)^{-1}X^T\,\mathbb{E}(Y) = (X^TX)^{-1}X^TX\beta = \beta$$

Variance:

Let $A = (X^TX)^{-1}X^T$

$$\begin{aligned}
Var(\hat{\beta}) &= Var((X^TX)^{-1}X^TY) = Var(AY) = AVar(Y)A^T \\
&= (X^TX)^{-1}X^T\sigma^2 I((X^TX)^{-1}X^T)^T \\
&= \sigma^2(X^TX)^{-1}X^T(X((X^TX)^{-1})^T) \\
&= \sigma^2(X^TX)^{-1}X^TX((X^TX)^T)^{-1} \\
&= \sigma^2(X^TX)^{-1}X^TX((X^TX))^{-1} \\
&= \sigma^2(X^TX)^{-1}
\end{aligned}$$

Therefore, $\hat{\beta} \sim \mathcal{N}(\beta, \sigma^2(X^TX)^{-1})$

(b) Likelihood Function:

$$\mathcal{L}(\beta|Y) = -\frac{m}{2}ln(2\pi) - \frac{1}{2}ln(|\Sigma|) - \frac{\sigma^2}{2}(Y - X\beta)^T(Y - X\beta)$$

Set $\frac{\partial log(\mathcal{L}(\beta|Y)}{\partial\beta}) = 0$

$$\frac{\partial log(\mathcal{L}(\beta|Y)}{\partial\beta} = \frac{\partial[-\frac{\sigma^2}{2}(Y^TY - 2\beta^TX^TY + \beta^TX^TX\beta)]}{\partial\beta} = -\frac{\sigma^2}{2}[-2X^TY+2(X^TX)\beta] = 0$$

$$\begin{aligned}
-X^TY + (X^TX)\beta &= 0 \\
\hat{\beta}_{ML} &= (X^TX)^{-1}X^TY
\end{aligned}$$

(c) For each i from 1 to m, the marginal distribution of $\hat{\beta}_i$ is:

$$\hat{\beta}_i \sim \mathcal{N}(\beta_i, \sigma^2 C_{ii})$$

where C_{ii} is the ith diagonal element of $(X^TX)^{-1}$

Consequently,

$$\frac{\hat{\beta}_i - \beta_i}{\sigma\sqrt{C_{ii}}} \sim \mathcal{N}(0,1)$$

$$\mathbb{P}(|\hat{\beta}_i - \beta_i| \leq \epsilon) = \mathbb{P}(\beta_i - \epsilon \leq \hat{\beta}_i \leq \beta_i + \epsilon)$$

$$= \mathbb{P}(\frac{-\epsilon}{\sigma\sqrt{ii}} \leq \frac{\hat{\beta}_i - \beta_i}{\sigma\sqrt{C_{ii}}} \leq \frac{\epsilon}{\sigma\sqrt{C_{ii}}}) = \Phi(\frac{\epsilon}{\sigma\sqrt{C_{ii}}}) - \Phi(\frac{-\epsilon}{\sigma\sqrt{C_{ii}}})$$

2.2 Ridge Regression

(a) $\hat{\beta}_{MAP}$ estimator maximize the posterior $p(\beta|X,Y) \propto p(X,Y|\beta)P(\beta)$

$$p(X,Y|\beta) \propto \mathcal{L}(\beta|Y) \propto exp\{-\frac{\sigma^2}{2}(Y - X\beta)^T(Y - X\beta)\}$$

$$P(\beta) \propto exp\{\frac{\tau^2}{2}\beta^T\beta\}$$

Set

$$\frac{\partial log(p(X,Y|\beta)p(\beta))}{\partial\beta} = 0$$

$$log[p(X,Y|\beta)] + log[p(\beta))] \propto -\frac{\sigma^2}{2}(Y^TY - 2\beta^TX^TY + \beta^TX^TX\beta) - \frac{\tau^2}{2}\beta^T\beta$$

$$\frac{\partial log(p(X,Y|\beta)p(\beta))}{\partial\beta} = \frac{\sigma^2}{2}[-2X^TY + 2(X^TX)\beta] + \tau^2\beta = -\sigma^2X^TY + \sigma(X^TX)\beta + \tau^2\beta = 0$$

$$\sigma^2(X^TX)\beta + \tau^2\beta = \sigma^2X^TY$$

$$(X^TX)\beta + \frac{\tau^2}{\sigma^2}\beta = X^TY$$

$$(X^TX + \frac{\tau^2}{\sigma^2}I)\beta = X^TY$$

$$\hat{\beta}_{MAP} = (X^TX + \lambda I)^{-1}X^TY$$

(b) Let $X^* = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mm} \\ \sqrt{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda} \end{bmatrix}$

Then $X^{*T}X^* = \begin{bmatrix} x_{11} & \dots & x_{n1} & \sqrt{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{1m} & \dots & x_{nm} & 0 & \cdots & \sqrt{\lambda} \end{bmatrix} \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \\ \sqrt{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda} \end{bmatrix}$

$= \begin{bmatrix} \sum_{i=1}^m x_{i1}^2 + \lambda & \dots & \sum_{j=1}^m x_{i1}x_{im} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^m x_{im}x_{i1} & \dots & \sum_{j=1}^m x_{im}x_{im} + \lambda \end{bmatrix} = X^TX + \lambda I$

$X^{*T}Y = \begin{bmatrix} x_{11} & \dots & x_{n1} & \sqrt{\lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{1m} & \dots & x_{nm} & 0 & \cdots & \sqrt{\lambda} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \\ 0 \\ \vdots \\ 0 \end{bmatrix} = X^TY$

Therefore, MLE estimate of β based on X^* and Y^* is

$$\beta^*_{ML} = (X^{*T}X^*)^{-1}X^{*T}Y = (X^TX + \lambda I)^{-1}X^TY$$

2.3 Cross Validation

(a) See code in Appendix

```
### Part (a) ###
# Load data from dataset.mat
dataset = ...
```

(b) See 6 functions in Appendix

```
### Part (b) ###
def shuffle_data(data):
    ...
```

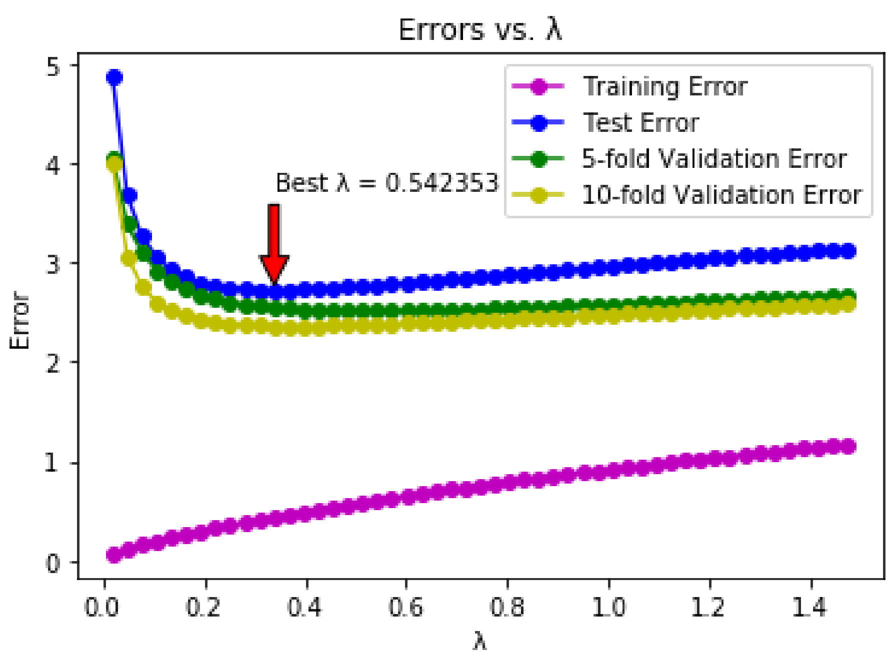
(c) See code in Appendix

```
### Part (c) ###
def lambd_train_test(train_data, test_data, lambd_seq)
    ...
```

(d) In the program, for reprehensibility purposes, I set the random seed to 2019. As we observe in the following graph, rolling through different λ in the cross validation process. The best choice of λ is obtained at $\lambda^* = 0.542353$, which minimizes the test error, 5-fold validation error and 10-fold validation error.

The training error increases as we increase the value of λ. The test error curve, 5-fold validation error curve, and 10-fold validation error curve have all have steep drops as we initially increase the value of lambda starting from 0.02. The curves then decrease in a slower pace, eventually hitting the best λ. The curves slope upwards after the best λ.

Additionally, 10-fold validation error achieved better error rate than 5-fold validation.




```
Test error is minimized when  = 0.542353
5-fold validation error is minimized when  = 0.542353
10-fold validation error is minimized when  = 0.542353
```

Appendix: Python Code

```
"""
STA414 Assignment 1
Due Jan 29, 2019

@name: cross_validation.py
@author: zikunchen
"""

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

### Part (b) ###

def shuffle_data(data):
    """Returns randomly uniformly permuted version of data along the samples.
    Note that y and X need to be permuted the
    same way preserving the target-feature pairs."""
    indices = np.arange(len(data['y']))
    np.random.seed(SEED)

    indices = np.random.permutation(indices)

    return {'y': np.array(data['y'])[indices].tolist(), \
            'X': np.array(data['X'])[indices].tolist()}

def split_data(data, num_folds, fold):
    """
    - Attributes -
    data:
        (y, X) pair in the form of dictionary
    num_folds:
        number of folds in total
    fold:
        specific fold number to be used as the training set

    - Returns -
    data_fold:
        (y, X) pair used for validation
    data_rest:
        rest of the data used for training
    """

    N = len(data['y'])
```

```
    fold_size = int(N//num_folds)

    fold_indices = np.arange((fold - 1) * fold_size, fold * fold_size)
    data_fold = {'y': np.array(data['y'])[fold_indices].tolist(), \
                 'X': np.array(data['X'])[fold_indices].tolist()}

    rest_indices = np.append(np.arange(0, (fold - 1) * fold_size), \
                             np.arange(fold * fold_size, N), axis = 0)
    data_rest = {'y': np.array(data['y'])[rest_indices].tolist(), \
                 'X': np.array(data['X'])[rest_indices].tolist()}

    return data_fold, data_rest


def train_model(data, lambd):
    """ Returns the coefficients of ridge regression with penalty level ."""

    N = len(data['y'])

    X = np.array(data['X'])
    y = np.array(data['y']).reshape(N,1)

    M = X.shape[1]

    XTX = np.dot(X.T, X)

    model = np.dot(np.dot(np.linalg.inv(XTX + lambd*np.identity(M)), X.T), y)

    return np.squeeze(model).tolist()


def predict(data, model):
    """ Returns the predictions based on data and model."""

    X = np.array(data['X'])
    M = X.shape[1]
    beta = np.array(model).reshape(M,1)

    return np.squeeze(np.dot(X, beta)).tolist()

def loss(data, model):
    """Returns the average squared error loss based on model."""

    N = len(data['y'])
    y = np.array(data['y']).reshape(N,1)
```

```python
    predictions = np.array(predict(data, model)).reshape(N, 1)

    error = np.sum(np.squeeze((y - predictions)**2)) / N
    return error

def cross_validation(data, num_folds, lambd_seq):
    """Returns the cross validation error across all lambdas in lambd_seq"""
    data = shuffle_data(data)
    cv_error = [0] * len(lambd_seq)

    for i in range(len(lambd_seq)):
        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[i] = cv_loss_lmd/num_folds

    return cv_error

### Part (c) ###

def lambd_train_test(train_data, test_data, lambd_seq):
    """Returns the training and test error across all lambdas in lambd_seq"""

    train_error = [0] * len(lambd_seq)
    test_error = [0] * len(lambd_seq)

    for i in range(len(lambd_seq)):
        lambd = lambd_seq[i]
        model = train_model(train_data, lambd)
        train_error[i] = loss(train_data, model)
        test_error[i] = loss(test_data, model)

    return train_error, test_error

### Helper Function ###

def find_min_lambd(error_list):
    """Find the best lambda based on the minimum error achieved in error_list"""
    min_error, index =  min((val, idx) for (idx, val) in enumerate(five_fold_cv))
    return lambd_seq[index]
```

```
if __name__ == '__main__':

    ### Part (a) ###
    dataset = sio.loadmat("/Users/zikunchen/Desktop/STA414/A1/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]

    # Set Random Seed to reproduce results
    SEED = 2019

    # Construct datasets
    train_data = {'y': data_train_y, 'X': data_train_X}
    test_data = {'y': data_test_y, 'X': data_test_X}

    # Construct Lambda values
    space = (1.5-0.02)/(50-1+2)
    lambd_seq = np.arange(0.02, 1.5, space).tolist()

    # Compute training, validation and test errors
    train_error, test_error = lambd_train_test(train_data, test_data, lambd_seq)
    five_fold_cv = cross_validation(train_data, 5, lambd_seq)
    ten_fold_cv = cross_validation(train_data, 10, lambd_seq)

    #Find the lambda with the least error
    lambd_test = find_min_lambd(test_error)
    lambd_five_fold= find_min_lambd(five_fold_cv)
    lambd_ten_fold = find_min_lambd(ten_fold_cv)

    # Plot 'Training Error', 'Test Error',
    # '5-fold Validation Error', and '10-fold Validation Error'
    # curves against Lambda values
    fig, ax = plt.subplots()

    test_min = min(test_error)
    xpos = test_error.index(test_min)
    xmin = lambd_seq[xpos]

    ax.annotate('Best lambda = %f' % lambd_test, xy=(xmin, test_min), \
                xytext=(xmin, test_min+1), \
                arrowprops=dict(facecolor='red', shrink=0.05),)

    plt.plot(lambd_seq, train_error, 'mo-')
    plt.plot(lambd_seq, test_error, 'bo-')
    plt.plot(lambd_seq, five_fold_cv, 'go-')
```

```
plt.plot(lambd_seq, ten_fold_cv, 'yo-')

plt.title('Errors vs. Lambda')
plt.ylabel('Error')
plt.xlabel('')
plt.legend(['Training Error', 'Test Error', '5-fold Validation Error', \
            '10-fold Validation Error'], loc='upper right')

plt.show()

print("Test error is minimized when lambda = %f" % lambd_test)
print("5-fold validation error is minimized when lambda = %f" % lambd_five_fold)
print("10-fold validation error is minimized when lambda = %f" % lambd_ten_fold)
```