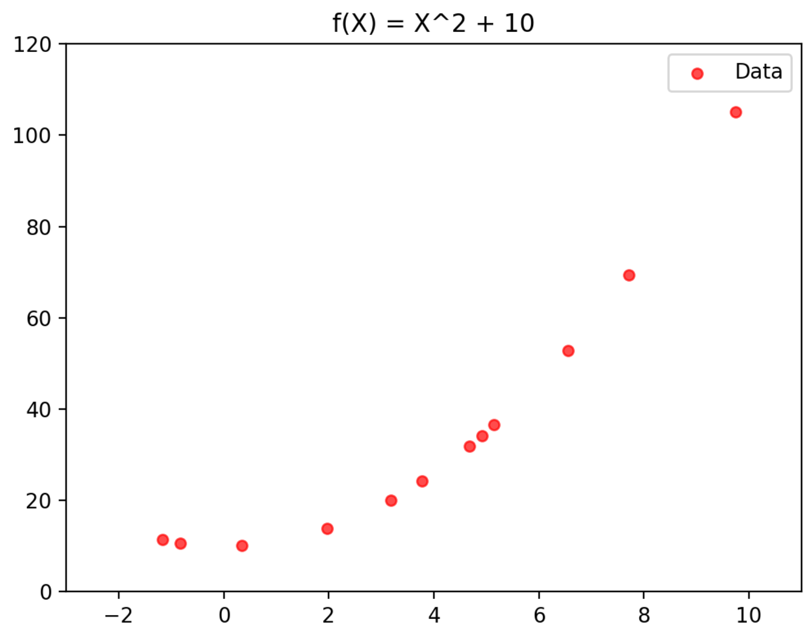
Christopher Thompson
CS 4340
Project 5

Analysis of Regularization on Linear Regression

1. $X = [1.97\ \ 3.78\ \ 4.68\ \ -0.82\ \ 9.75\ \ 4.92\ \ 5.15\ \ 3.18\ \ 6.55\ \ 7.71\ \ 0.35\ \ -1.16]$

$Y = [3.88\ \ 24.29\ \ 31.9\ \ 10.67\ \ 105.06\ \ 34.21\ \ 36.52\ \ 20.11\ \ 52.9\ 69.44\ \ 10.12\ \ 11.35]$



2. Linear Regression Without Regularization:
   a. Equation of Line: y = f(x) = 7.79 x + 5.15
   b. In-Sample Error Rate: 123.704
   c. Root Mean Squared Error: 11.122
3. Cross-Validation Results:
   a. Lambda = 0.1

Line	In-Sample Error	Cross-Validation Error
y = f(x) = 9.02 x + -2.17	134.28	567.87
y = f(x) = 8.8 x + -0.15	127.99	457.53
y = f(x) = 8.23 x + 4.91	131.9	319.58

b. Lambda = 1

Line	In-Sample Error	Cross-Validation Error
y = f(x) = 8.83 x + -1.26	134.68	455.46
y = f(x) = 8.71 x + 0.28	128.08	301.76
y = f(x) = 8.32 x + 4.04	132.28	283.56

c. Lambda = 10

Line	In-Sample Error	Cross-Validation Error
y = f(x) = 8.15 x + 0.6	141.24	369.54
y = f(x) = 8.34 x + 0.98	131.44	230.15
y = f(x) = 8.32 x + 2.03	138.48	244.48

d. Lambda = 100

Line	In-Sample Error	Cross-Validation Error
y = f(x) = 5.65 x + 0.86	340.55	695.06
y = f(x) = 6.38 x + 0.91	309.48	503.65
y = f(x) = 6.34 x + 1.01	331.16	546.69

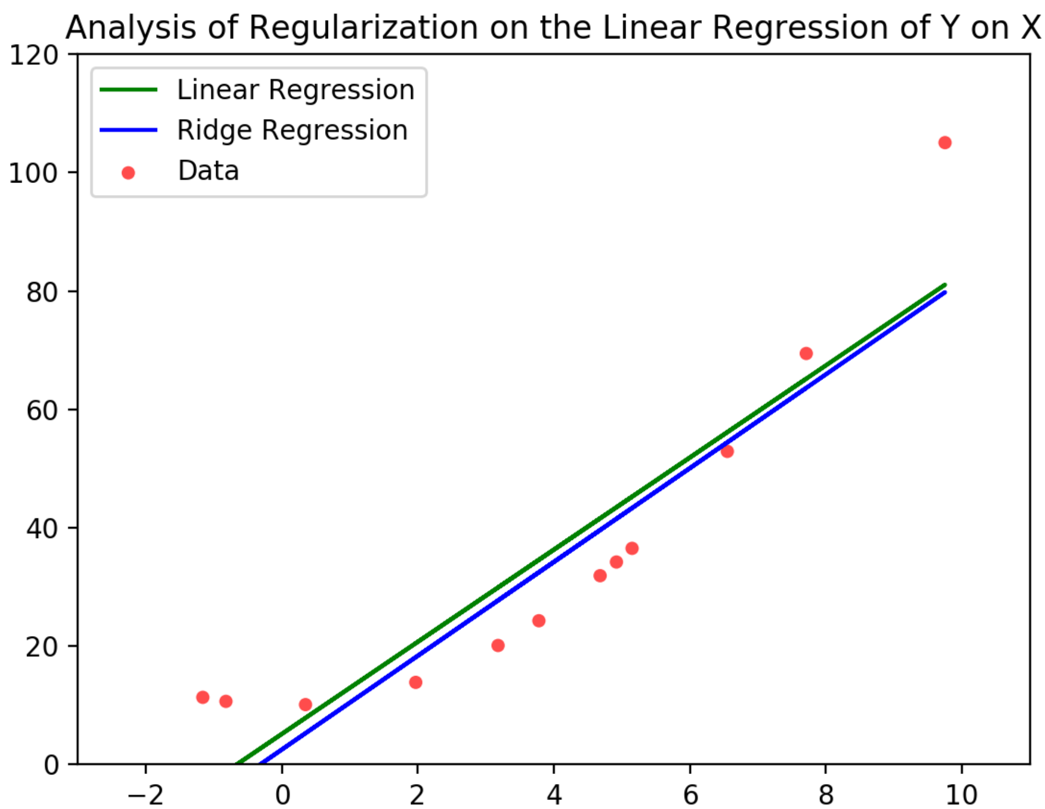
4. $Final\ \lambda = 10$

    a. This may seem odd and a bit high, but I ran this experiment for 40 trials and out of those 40, $\lambda = 10$ had the lowest average minimum cross-validation error.

5. Final equation of Ridge Regression line: y = f(x) = 7.926 x + 2.518

6. Final In-Sample Error: 128.34

7. Final plot of data with both regression lines:



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import random
import decimal

'''This function creates the training data. Returns a pandas dataframe of random
uniform samples'''
def createXData():
    '''create a random uniform sample for X'''
    x_ = []
    X = []

    '''loop will append X_0, which always equals 1, and a random uniform double to the
array'''
    for i in range(12):
        x_.append(np.round(random.uniform(-2, 10), 2))
        X.append([1, x_[i]])

    X = np.array(X)

    return X

'''This function will create the y data'''
def createYData(X):
    '''f(x) = x^2 + 10'''
    y = [ np.round(np.power(x, 2) + 10, 2) for x in X ]

    y = np.array(y)
    return y

'''This function will compute weights for linear regression without regularization and
return the weight vector'''
def linearRegression(X, y):
    x_sword = pseudoInverse(X) #compute the pseudo inverse of the matrix X
    weights = np.dot(x_sword, y) #compute the weight vector
    return weights

'''This function will compute yHat and return the yHat (predictions) vector for linear
regression without regularization'''
def yHatLinear(X, linearWeights):
    y_hat = np.dot(X, linearWeights) #compute the dot product of the matrix X and the
linear weights
    return y_hat #return the predictions


def ridgeRegression(X, y, lamda):
    xTx = np.dot(X.T, X) # get the dot product of the X transpose and X
    lambda_I = np.dot(lamda, np.identity(2)) # get the dot product of lambda and the
Identity matrix

    '''let K = xTx + lambda * I'''
    K = np.add(xTx, lambda_I) # compute K
    inverse_K = np.linalg.inv(K) # get the inverse of K
    xTy = np.dot(X.T, y) # get the dot product of X transpose and y
    weights = np.dot(inverse_K, xTy) # compute the weight vector
    return weights # return the weights

'''This function will compute yHat and return the yHat (predictions) vector for ridge
regression'''
```

```
def yHatRidge(X, weights):
    yHat = np.dot(X, weights) # compute the dot product of the matrix X and the and
the ridge regression weights
    return yHat # return the predictions

'''This function will compute and return the pseudo inverse of the matrix X'''
def pseudoInverse(X):
    xTx = np.dot(X.T, X)
    xTx_inv = np.linalg.inv(xTx)
    Xsword = np.dot(xTx_inv, X.T)
    return Xsword

'''This function will conduct  3 fold cross validation'''
def crossValidation(X, y, iteration, lmbda):

    if iteration == 0:
        Xtrain = X[0:8:1] #get the first 8 values for the 2/3 training data
        y_observed = y[0:8:1]

    elif iteration == 1:
        Xtrain = X[2:10:1] #get some middle values, leave out 2 from the head and tail
        y_observed = y[2:10:1]

    else:
        Xtrain = X[4:12:1] #get the last 2/3 of the training set
        y_observed = y[4:12:1]

    weights_reg = ridgeRegression(Xtrain, y_observed, lmbda)
    yHat = yHatRidge(Xtrain, weights_reg)

    '''let Ecv = cross-validation error estimate AND let Ein = the in-sample error
estimate'''
    Ecv = crossValidationError(Xtrain, yHat, y_observed, lmbda)
    Ein = inSampleError(y_observed, yHat)
    rootError = RMSE(y_observed, yHat)


    '''Print some stats about the iteration'''
    print("\nIteration: ", iteration + 1)
    print("Lambda: ", lmbda)
    Xsub1Col = Xtrain[:, 1] # this just selects the column with the 'real' X values
    slope, intercept = np.round(np.polyfit(Xsub1Col, yHat, 1), 2)
    print("Equation of Line: ", "y = f(x) =", slope, "x +", intercept)
    print("In-Sample Error: ", np.round(Ein, 2))
    print("Cross-Validation Error: ", np.round(Ecv, 2))
    print("Root Mean Squared Error: ", np.round(rootError, 2))

    return Ecv, rootError


'''This function will compute and return H(lambda)'''
def H_of_lambda(X, lmbda):
    '''H(lambda) = X(xTx + lambda * I) ^ -1 * xT'''
    xTx = np.dot(X.T, X)
    lambda_I = np.dot(lmbda, np.identity(2))

    '''let K = xTx + lambda * I'''
    K = xTx + lambda_I  # compute K
    inverse_K = np.linalg.inv(K)  # get the inverse of K
    inverse_K_mul_X = np.dot(X, inverse_K) # get the dot product of K^-1 and X
    H_lambda = np.dot(inverse_K_mul_X, X.T) # get the dot product o
```

```python
    return  H_lambda # return H(lambda)

'''This function will compute the in-sample error estimate'''
def inSampleError(y, yHat):
    total = 0
    for i in range(len(y)):
        total += (yHat[i] - y[i]) ** 2
    Ein = total / len(y)
    return Ein


'''This function will compute the compute the cross validation error'''
def crossValidationError(X, yHat, y, lamda):
    h_of_lambda = H_of_lambda(X, lamda) # get the H of lambda matrix

    total = 0
    for i in range(len(y)):
        numerator = yHat[i] - y[i]  # compute the numerator
        denominator = 1 - h_of_lambda[i, i] # compute the denominator
        total += np.power(numerator/denominator, 2) # compute the total

    Ecv = total / len(y) # compute the cross validation error
    return Ecv # return the cross validation error


'''This function will compute the Root Mean Squared Error for model evaluation and
return the Root Mean Squared Error'''
def RMSE(y, y_hat):

    total = 0;
    for i in range(len(y)):
        yhat_Minus_y = y_hat[i] - y[i] # y_hat = predictions - observed
        total += np.power(yhat_Minus_y, 2) # y_hat ^ 2
    RMSE = np.sqrt(total/len(y)) #compute the root mean squared error

    return RMSE # return RMSE


def plotData(X, y):
    fig = plt.figure()
    plt.title("f(X) = X^2 + 10")
    ax1 = fig.add_subplot(1,1,1)
    ax1.scatter(X, y, alpha=0.7, c='red', s=25, label="Data")
    plt.xlim(-3, 11)
    plt.ylim(0, 120)
    plt.legend()
    plt.show()

'''This function will plot the data along with the specified lines. Note: you must
supply a 1-D numpy array as the first parameter'''
def plotDataWithLines(X, y, yHat1, yHat2):
    fig = plt.figure()
    plt.title("Analysis of Regularization on the Linear Regression of Y on X")
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.scatter(X, y, alpha=0.7, c='red', edgecolors='none', s=25, label="Data")
    plt.xlim(-3, 11)
    plt.ylim(0, 120)

    '''plot the linear line'''
    m, b = np.polyfit(X, yHat1, 1)
    Line1 = [ m * x + b for x in X ]
```

```
    ax2 = fig.add_subplot(1, 1, 1)
    ax2.plot(X, Line1, color='green', label="Linear Regression")

    '''plot the ridge line'''
    m1, b1 = np.polyfit(X, yHat2, 1)
    line2 = [m1*x + b1 for x in X]
    ax3 = fig.add_subplot(1, 1, 1)
    ax3.plot(X, line2, color='blue', label="Ridge Regression")

    plt.legend()
    plt.show()

def main():

    lamdas = [0.1, 1, 10, 100] #list for the lambda values

    '''make the test data'''
    X = createXData()
    Xsub1Col = X[:, 1]  # crucial to have the correct calculations *** learned the
hard way, hence why this is late ***
    y = createYData(Xsub1Col)

    '''print the test data'''
    print("X-matrix:")
    print(X[:, 1]) #only print the X_sub1 column
    print("\nY-matrix:")
    print(y)
    print("\n")

    '''show the data without regression lines'''
    plotData(Xsub1Col, y) # you have to feed this function a 1-d numpy array as the
first parameter

    '''Compute the linear regression w/o regularization weights '''
    weightsLin = linearRegression(X, y) # fit the model
    yHatLin = yHatLinear(X, weightsLin) # get a vector of predictions
    Ein = inSampleError(y, yHatLin) # get the in-sample error rate
    root_error = RMSE(y, yHatLin) # root mean squared error
    slope, intercept = np.round(np.polyfit(Xsub1Col, yHatLin, 1), 2)
    print("Linear Regression Without Regularization: ", )
    print("Equation of Line:", "y = f(x) =", slope, "x +", intercept)
    print("In-Sample Error Rate:", np.round(Ein, 3))
    print("Root Mean Squared Error:", np.round(root_error, 3))


    avgEcvForLambdaSelection = []
    correspondingLambda = []
    '''Do three-fold cross validation'''
    for lam in lamdas:
        Ecv_total = 0
        rootErrorTotal = 0
        for i in range(3):
            Ecv, rootError = crossValidation(X, y, i, lam)
            Ecv_total += Ecv
            rootErrorTotal += rootError

        avgEcvForLambdaSelection.append((Ecv_total/3))
        correspondingLambda.append(lam)


    minEcvIndex = np.argmin(avgEcvForLambdaSelection) #get the index of the minimum
```

```
cross-validation error

    print("\nThe average minimum Cross-Validation Error is: ",
np.min(avgEcvForLambdaSelection))
    print("The corresponding lambda is: ", correspondingLambda[minEcvIndex])

    selectedLambda = correspondingLambda[minEcvIndex] #select the corresponding lambda

    weightsReg = ridgeRegression(X, y, selectedLambda) # train the ridge regression
weights
    validated_yHat = yHatRidge(X, weightsReg) # get the predications from the cross-
validated model
    finalInSampleError = inSampleError(y, validated_yHat) # compute the final in-
sample error
    slope, intercept = np.round(np.polyfit(Xsub1Col, validated_yHat, 1), 3) # compute
the final ridge regression line equation

    print("\nFinal equation of Ridge Regression line: ", "y = f(x) =", slope, "x +",
intercept)
    print("Final In-Sample Error: ", finalInSampleError)
    plotDataWithLines(Xsub1Col, y, yHatLin, validated_yHat)


main()
```