CSE 151: Programming Assignment #3

Due on Monday, May 9, 2016

Mangione-Tran,Carmine 9AM

Kyle Lee
A01614951

Homework Code

```
import numpy as np
import csv
import random
import math
import operator
from decimal import *
from numpy import genfromtxt

##### METHODS #####
#method for generating hits
#@param:x => our data
#       :testSize => hit rate
#       :genList => list of # of hits per index
#@return:genList => updated list with hits per index
def count (x, testSize, genList):
    size = len(x)
    expectedDraws = int(round(size*testSize))
    x1 = Decimal(expectedDraws)/Decimal(size)

    j = 0

    for i in range(0, size):
        # Compare with random uniform
        x2 = random.uniform(0,1)
        if x2 < x1:
            genList[i] = genList[i] + 1
            j = j+1
        # Update x1 to new conditional probability
        x1 = Decimal(expectedDraws-j)/Decimal(size-i)
    return genList

#method for separating input data based on counter
#@param: counter => a vector that stores the indices of test/train sets
#@param: inputData => original data
#@return: trainingSet - Training Set based on data
#@return: testSet - Test Set based on data
def separateSet(counter,inputData):
    trainingSet = []
    testSet = []
    size = len(inputData)
    for i in range(size):
        if counter[i] == 1:
            trainingSet.append(inputData1[i])
        else:
            testSet.append(inputData1[i])
    return trainingSet, testSet

#method to initialize QR Decomposition on a matrix X
#@param: X - the matrix we want to decompose. We note that the
#            X is not an augmented matrix
def QRdecompose(X):
```

```
    # Copy
    R = X.copy()
    # Store the shape of A
    [m, n] = X.shape
    # Create identity matrix of size m
    Q = np.identity(m)

    # Applying method 1. Recursively define H
    for i in range(n - (m == n)):

        # Empty householder identity matrix
        H = np.identity(m)

        # Create householder matrix
        H[i:, i:] = householdervk(R[i:, i])

        # Recreate Q,R
        Q = np.dot(Q, H)
        R = np.dot(H, R)
    return [Q,R]

def householdervk(X):

    # Determine shift
    vk = X / (X[0] + np.copysign(np.linalg.norm(X), X[0]))

    # e1
    vk[0] = 1

    # Take the shape of each R[i:,i]
    H = np.identity(X.shape[0])

    # Create householder matrix
    H = H - (2 / np.dot(vk, vk)) * np.dot(vk[:, None], vk[None, :])
    return H

#method for back substitution a matrix A with b
#@param: R - Upper right triangular matrix already formatted
#@param: b - A nx1 vector
def backsolve(R,b):
    n = np.size(b)
    x = np.zeros((n,1))

    # Start at the end, solve accordingly
    for i in range(n-1,-1,-1):
        x[i] = (b[i] - np.dot(R[i,:],x))/R[i,i]

    return x


#method for adjusting to method 2. Remove (m-n) rows and adjust
#sizes accordingly
```

```
#@param: Q - a matrix
#@param: R - an upper right triangular matrix with rows of 0's
def simplifyQR(Q,R):
   [Rm,Rn] = R.shape
   # Remove rows of zeros and adjust Q matrix to match dimensions
   # for np.dot
   if Rm != Rn:
       R = R[0:Rn,:]
       Q = Q[:,0:Rn]
   return Q,R

#method for calculating the root mean squared error
#@param: YActual - the correct vector
#@param: YEstimated - the estimated vector after QR Decomposition
def rmse(Y_actual,Y_estimated):
    return np.sqrt(np.mean((Y_actual-Y_estimated)**2))

###########################
# Set a random seed
np.random.seed(123451)

#################### Test regression-.05 ##################
# Read CSV Files
inputData1 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho

# Initiaize test Size
trainingSize = 0.6

# Length of the data
size1 = len(inputData1)

# Initialize counter
counter1 = [] #stores hit rates on index

# initialize counter array
for x in range(size1):
    counter1.append(0)

# Find test size
for i in range(1,2):
    counter1 = count(inputData1, trainingSize, counter1)

# Create training and testing sets
[trainingSet1, testSet1] = separateSet(counter1, inputData1)
# Change to numeric arrays
trainingSet1 = np.array(trainingSet1)
testSet1 = np.array(testSet1)

# Separate into X and Y
X_train1 = trainingSet1[:,:-1]
Y_train1= trainingSet1[:,-1]

X_test1 = testSet1[:,:-1]
```

```
Y_test1 = testSet1[:,-1]
# Fix dimension
Y_test1 = Y_test1[:,np.newaxis]

# Perform QR Decomposition
Q1,R1 = QRdecompose(X_train1)

# Check if QR decomposition was successful
np.dot(Q1,R1)
X_train1

# Fixing and backsolving
Q1,R1 = simplifyQR(Q1,R1)
Z1 = np.dot(Q1.T,Y_train1)
Z1 = Z1[:,np.newaxis]
beta1 = backsolve(R1,Z1)

RMSE1 = rmse(np.dot(X_test1,beta1), Y_test1)

################# Test regression-A #################
# Read CSV Files
inputData2 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho

# Initiaize test Size
trainingSize = 0.6

# Length of the data
size2 = len(inputData2)

# Initialize counter
counter2 = [] #stores hit rates on index
# initialize counter array
for x in range(size2):
    counter2.append(0)

# Find test size
for i in range(1,2):
    counter2 = count(inputData2, trainingSize, counter2)

# Create training and testing sets
[trainingSet2, testSet2] = separateSet(counter2, inputData2)
# Change to numeric arrays
trainingSet2 = np.array(trainingSet2)
testSet2 = np.array(testSet2)

X_train2 = trainingSet2[:,:-1]
Y_train2= trainingSet2[:,-1]

X_test2 = testSet2[:,:-1]
Y_test2 = testSet2[:,-1]
# Fix dimension
Y_test2 = Y_test2[:,np.newaxis]
```

```
# Perform QR Decomposition
Q2,R2 = QRdecompose(X_train2)

# Check if QR decomposition was successful
np.dot(Q2,R2)
X_train2

# Fixing and backsolving
Q2,R2 = simplifyQR(Q2,R2)
Z2 = np.dot(Q2.T,Y_train2)
beta2 = backsolve(R2,Z2)

RMSE2 = rmse(np.dot(X_test2,beta2), Y_test2)

################# Test regression-B #################
# Read CSV Files
inputData3 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho

# Initiaize test Size
trainingSize = 0.6

# Length of the data
size3 = len(inputData3)

# Initialize counter
counter3 = [] #stores hit rates on index
# initialize counter array
for x in range(size3):
    counter3.append(0)

# Find test size
for i in range(1,2):
    counter3 = count(inputData3, trainingSize, counter3)

# Create training and testing sets
[trainingSet3, testSet3] = separateSet(counter3, inputData3)
# Change to numeric arrays
trainingSet3 = np.array(trainingSet3)
testSet3 = np.array(testSet3)

X_train3 = trainingSet3[:,:-1]
Y_train3= trainingSet3[:,-1]

X_test3 = testSet3[:,:-1]
Y_test3 = testSet3[:,-1]
# Fix dimension
Y_test3 = Y_test3[:,np.newaxis]

# Perform QR Decomposition and backsolving
Q3,R3 = QRdecompose(X_train3)

# Check if QR decomposition was successful
np.dot(Q3,R3)
```

```
X_train3

# Fixing and linear regression using Householders
Q3,R3 = simplifyQR(Q3,R3)
Z3 = np.dot(Q3.T,Y_train3)
beta3 = backsolve(R3,Z3)

RMSE3 = rmse(np.dot(X_test3,beta3), Y_test3)

################# Test regression-C #################
# Read CSV Files
inputData4 = genfromtxt(r'C:\Users\Kyle Lee\Google Drive\School Doc. 2015-2016\CSE 151\Programming Ho

# Initiaize test Size
trainingSize = 0.6

# Length of the data
size4 = len(inputData4)

# Initialize counter
counter4 = [] #stores hit rates on index
# initialize counter array
for x in range(size4):
    counter4.append(0)

# Find test size
for i in range(1,2):
    counter4 = count(inputData4, trainingSize, counter4)

# Create training and testing sets
[trainingSet4, testSet4] = separateSet(counter4, inputData4)
# Change to numeric arrays
trainingSet4 = np.array(trainingSet4)
testSet4 = np.array(testSet4)

X_train4 = trainingSet4[:,:-1]
Y_train4= trainingSet4[:,-1]

X_test4 = testSet4[:,:-1]
Y_test4 = testSet4[:,-1]
# Fix dimension
Y_test4 = Y_test4[:,np.newaxis]

# Perform QR Decomposition
Q4,R4 = QRdecompose(X_train4)

# Check if QR decomposition was successful
np.dot(Q4,R4)
X_train1

# Fixing and backsolving
Q4,R4 = simplifyQR(Q4,R4)
Z4 = np.dot(Q4.T,Y_train4)
```

```
beta4 = backsolve(R4,Z4)

RMSE4 = rmse(np.dot(X_test4,beta4), Y_test4)

################# ABALONE DATA SET #################
# Read CSV Files
inputFile5 = open('abalone.data')
inputReader5 = csv.reader(inputFile5)
inputData5 = list(inputReader5)   #inputData = list of our data (which is in lists)


# Initiaize test Size
trainingSize = 0.6

# Length of the data
size5 = len(inputData5)

# Initialize counter
counter5 = [] #stores hit rates on index

# initialize counter array
for x in range(size5):
    counter5.append(0)

# Find test size
for i in range(1,2):
    counter5 = count(inputData5, trainingSize, counter5)

for i in range(size5):
    if inputData5[i][0] == 'M':
        inputData5[i][0] = 0
    elif inputData5[i][0] == 'F':
        inputData5[i][0] = 1
    elif inputData5[i][0] == 'I':
        inputData5[i][0] = 2

# Add 3 columns for classification of sex
proxiedData = np.zeros((size5,len(inputData5[0])+3))
proxiedData[:,:-3] = inputData5
# Fix data for Euclidean distance
for i in range(size5):
    if proxiedData[i][0] == 0:
        proxiedData[i][9] = 1
    elif proxiedData[i][0] == 1:
        proxiedData[i][10] = 1
    elif proxiedData[i][0] == 2:
        proxiedData[i][11] = 1

# Remove first column
proxiedData = proxiedData[:,1:12]

# Swap the actual to the predictions
proxiedData[:,[7,8,9,10]] = proxiedData[:,[8,9,10,7]]
```

```

# Keep track of counter and create training and testing sets
trainIndex5 = []
testIndex5 = []
trainingSet5 = []
testSet5 = []

# Create training and testing sets
for i in range(size5):
    if counter5[i] == 1:
        trainIndex5.append(i)
        trainingSet5.append(proxiedData[i])
    else:
        testIndex5.append(i)
        testSet5.append(proxiedData[i])


# Change to numeric arrays
trainingSet5 = np.array(trainingSet5)
testSet5 = np.array(testSet5)

X_train5 = trainingSet5[:,:-1]
Y_train5= trainingSet5[:,-1]

X_test5 = testSet5[:,:-1]
Y_test5 = testSet5[:,-1]
# Fix dimension
Y_test5 = Y_test5[:,np.newaxis]

# Perform QR Decomposition and backsolving
Q5,R5 = QRdecompose(X_train5)

# Check if QR decomposition was successful
np.dot(Q5,R5)
X_train5

# Fixing and backsolving
Q5,R5 = simplifyQR(Q5,R5)
Z5 = np.dot(Q5.T,Y_train5)
beta5 = backsolve(R5,Z5)

RMSE5 = rmse(np.dot(X_test5,beta5), Y_test5)
print('RMSE for regression-0.05 Dataset: ' + str(RMSE1))
print('RMSE for regression-A Dataset: ' + str(RMSE2))
print('RMSE for regression-B: Dataset: ' + str(RMSE3))
print('RMSE for regression-C: Dataset: ' + str(RMSE4))
print('RMSE for Abalone Dataset: ' + str(RMSE5))

# End
```

Console Output

```
>>> print('RMSE for regression-0.05 Dataset: ' + str(RMSE1))
RMSE for regression-0.05 Dataset: 0.0500339924867

>>> print('RMSE for regression-A Dataset: ' + str(RMSE2))
RMSE for regression-A Dataset: 0.0506783852395

>>> print('RMSE for regression-B: Dataset: ' + str(RMSE3))
RMSE for regression-B: Dataset: 0.0498675149061

>>> print('RMSE for regression-C: Dataset: ' + str(RMSE4))
RMSE for regression-C: Dataset: 0.0501509759907

>>> print('RMSE for Abalone Dataset: ' + str(RMSE5))
RMSE for Abalone Dataset: 2.32236830404
```