LMSE

September 29, 2021

[ ]:
```
# importing modules
import numpy
import pandas
from IPython.display import display
```

[ ]:
```
# reading data
myData = pandas.read_csv("ques1Data.csv")
myData2 = pandas.read_csv("ques1Data.csv")

display(myData)
```

	Y	squareFeet
0	245	1400
1	312	1600
2	279	1700
3	308	1875
4	199	1100
5	219	1550
6	405	2350
7	324	2450
8	319	1425
9	255	1700

[ ]:
```
# testing .iteritems()
for columnName, columnData in myData.iteritems():
    print(columnName)
    for i in columnData:
        print(i)
```

```
Y
245
312
279
308
199
219
405
324
```

```
319
255
squareFeet
1400
1600
1700
1875
1100
1550
2350
2450
1425
1700
```

[ ]:
```
# normalize data using min max normalization
for columnName, columnData in myData.iteritems():
    maxI = max(columnData)
    minI = min(columnData)

    tempList = []

    for j in range(len(columnData)):
        xStar = ( ((columnData[j] - minI) / (maxI - minI)) * (1 - 0) ) + 0
        tempList.append(xStar)

    myData[columnName] = tempList

display(myData)
```

	Y	squareFeet
0	0.223301	0.222222
1	0.548544	0.370370
2	0.388350	0.444444
3	0.529126	0.574074
4	0.000000	0.000000
5	0.097087	0.333333
6	1.000000	0.925926
7	0.606796	1.000000
8	0.582524	0.240741
9	0.271845	0.444444

[ ]:
```
# convert the X to matrix
# [
#     1 X11        X1k
#     1 X21        X2k
#     1 X22        X3k
```

```
#      1 Xn1         Xnk
# ]

xMat = []

for i in myData.index:
    templist = [1]

    for j in myData.iloc[i][1:]:
        templist.append(j)

    xMat.append(templist)


xMat = numpy.array(xMat)

display(xMat)
```

```
array([[1.        , 0.22222222],
       [1.        , 0.37037037],
       [1.        , 0.44444444],
       [1.        , 0.57407407],
       [1.        , 0.        ],
       [1.        , 0.33333333],
       [1.        , 0.92592593],
       [1.        , 1.        ],
       [1.        , 0.24074074],
       [1.        , 0.44444444]])
```

[ ]:
```
# transpose X
xMat_transpose = xMat.transpose()

display(xMat_transpose)
```

```
array([[1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.22222222, 0.37037037, 0.44444444, 0.57407407, 0.        ,
        0.33333333, 0.92592593, 1.        , 0.24074074, 0.44444444]])
```

[ ]:
```
# beta = ( (X_transpose*X) ^ -1 ) * X_transpose * y

# inverse ( X.T * X )
inversed = numpy.linalg.inv(numpy.dot(xMat_transpose,xMat))

display(inversed)
```

```
array([[ 0.3406777 , -0.52831689],
```

```
        [-0.52831689,  1.15972001]])
```

[ ]:

```
# intermidiate beta
tempBeta = numpy.dot(inversed , xMat_transpose)

display(tempBeta)
```

```
array([[ 0.22327394,  0.14500477,  0.10587019,  0.03738466,  0.3406777 ,
         0.16457206, -0.14850461, -0.1876392 ,  0.2134903 ,  0.10587019],
       [-0.27060134, -0.09879096, -0.01288578,  0.1374483 , -0.52831689,
        -0.14174356,  0.54549793,  0.63140312, -0.24912504, -0.01288578]])
```

[ ]:

```
# beta
y = numpy.array(myData["Y"])

beta = numpy.dot(tempBeta , y)
display(beta)
```

```
array([0.09705263, 0.71935168])
```

[ ]:

```
# function to scale by the y predicted value from normalized form to simple form
def scaleBackYPredicted(originalY , ypredicted):
    maxOriginalY = max(originalY)
    minOriginalY = min(originalY)

    tempList = []

    for j in range(len(ypredicted)):
        yScaled = ( ((ypredicted[j] - 0) / (1 - 0)) * (maxOriginalY -␣
 ↪minOriginalY) ) + minOriginalY
        tempList.append(yScaled)

    return tempList



# function to predict the y based on test data x
# x is the new input to predict y
# x should be a pandas data frame type
def hypothesisFunction(beta ,  x):

    x = numpy.array(x)

    # y = b0 + b1*x1 + b2*x2 +          + bk*xk

    yPredicted = beta[0]

    for i in range(1 , len(beta)):
```

```
        yPredicted = yPredicted + ( beta[i] * x[i-1] )

    return yPredicted
```

[ ]:
```
# function to normalize the new test data based on original data
# here original data min max are used to normalize the data
def returnNormalisedTestData(originalData , testData):

    for columnName, columnData in testData.iteritems():

        maxI = max(originalData[columnName])
        minI = min(originalData[columnName])

        tempList = []

        for j in range(len(columnData)):
            xStar = ( ((columnData[j] - minI) / (maxI - minI)) * (1 - 0) ) + 0
            tempList.append(xStar)

        testData[columnName] = tempList

    return testData
```

[ ]:
```
# test data
testData = [[[3000]] , [[2000]] , [[1500]]]



# convert test data to data frame
testData = pandas.DataFrame([[3000] , [2000] , [1500]] , columns =␣
 ↪["squareFeet"])

display(testData)



# normalize testData
testDataNormal = returnNormalisedTestData(myData2 , testData)

display(testData)



yPredicted = []

# predict y using hypothesis function
for i in testDataNormal.index:
    yPredicted.append(hypothesisFunction(beta , testDataNormal.iloc[i]))

display(yPredicted)
```

```
# scale back the predicted value
yPredicted = scaleBackYPredicted(myData2["Y"] , yPredicted)

for i in range(len(yPredicted)):
    print("price for plot {} = {}".format(testData.iloc[i].tolist() ,␣
 ↪yPredicted[i]))
```

```
   squareFeet
0        3000
1        2000
2        1500

   squareFeet
0    1.407407
1    0.666667
2    0.296296

[1.1094735102508577, 0.5766204139882432, 0.31019386585693587]

price for plot [1.4074074074074074] = 427.55154311167666
price for plot [0.6666666666666666] = 317.7838052815781
price for plot [0.2962962962962963] = 262.8999363665288
```

[ ]:
```
# Code by harshnative
# Email - harshnative@gmail.com
# Github - https://github.com/harshnative
```

[ ]:
```

```