gradientDesent

September 29, 2021

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

[ ]:
```
# read 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

[ ]:
```
# 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]])
```

[ ]:
```
# number of beta's
kValue = len(xMat[0])

print(kValue)

# number of rows in data
nValue = len(xMat)

print(nValue)


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

display(y)
```

```
2
10
```

```
array([0.22330097, 0.54854369, 0.38834951, 0.52912621, 0.        ,
       0.09708738, 1.        , 0.60679612, 0.58252427, 0.27184466])
```

[ ]:
```
# iterate over and change value of bk's
# as
# tempj = bj - aplha *  [( 1 / n ) * submission of from i=1 to n(( b0 + b1*xi1
↪+ b2*xi2          + bk*xik - yi) * xij)]
# bj = tempj

alpha = 0.01

# max iterations to find the beta
maxIteration = 50000

# accuracy of beta needed
betaErrorTolerance = 10

# init bk's as zero
# init tempk's as zero
betaKs = [0 for _ in range(kValue)]
tempKs = [0 for _ in range(kValue)]
slopeKs = [0 for _ in range(kValue)]


for iteration in range(maxIteration):

    breakLoop = False

    for j in range(kValue):
```

```
        slope = 1 / nValue
        submission = 0
        k = 1


        for i in range(nValue):
            innerSubmission = betaKs[0]

            for k in range(1 , kValue):
                innerSubmission = innerSubmission + (betaKs[k] * xMat[i][k])

            innerSubmission = innerSubmission - y[i]

            innerSubmission = innerSubmission * xMat[i][j]

            submission = submission + innerSubmission

        slope = submission * slope

        tempKs[j] = betaKs[j] - (alpha * slope)

        slopeKs[j] = slope


    for a,b in zip(betaKs , tempKs):
        if(round(a , betaErrorTolerance) == round(b , betaErrorTolerance)):
            breakLoop = True
        else:
            breakLoop = False


    if(breakLoop):
        print("break on" , iteration)
        break


    # assign bj = tempj
    for j in range(kValue):
        betaKs[j] = tempKs[j]

print(betaKs)
```

```
break on 21544
[0.09705269439985972, 0.7193515431506073]
```

[ ]:
```
# function to scale back the ypredicted using normalized x vector
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 y based on new x input
# x is the new input to predict y
# x must be a 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]]]

testData = pandas.DataFrame([[3000] , [2000] , [1500]] , columns =␣
↪["squareFeet"])

display(testData)


# normalize testData
testDataNormal = returnNormalisedTestData(myData2 , testData)

display(testData)


# predict y based on new data
yPredicted = []

for i in testDataNormal.index:
    yPredicted.append(hypothesisFunction(betaKs , testDataNormal.iloc[i]))

display(yPredicted)

# scale up y
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
```

```
[1.1094733847599738, 0.576620389833598, 0.31019389237041006]
```

```
price for plot [1.4074074074074074] = 427.55151726055465
price for plot [0.6666666666666666] = 317.7838003057212
price for plot [0.2962962962962963] = 262.89994182830446
```