Name:Vishal Chauhan

Prn:2020BTECS00090

SUB:Ai-Ml

Ass-8

1. Implement simple linear regression for below data. Also calculate accuracy of predictive model.

Year	GDP(X)	4wheeler_passengar_vehicle_sale(in lakhs)
2011	6.2	26.3
2012	6.5	26.6
2013	5.4	25
2014	6.5	26
2015	7.1	27.9
2016	7.9	30.4
2017	8.5	35.4
2018	8.9	38.5
2019	9.5	42.6
2020	10.6	48.3

```
from sklearn.metrics import mean_squared_error
import numpy as np


class Simple_Linear_Regression():

    """
        Class to calculate simple regression
    """

    def __init__(self, X, Y):
```

```
        """
            Constructor:
                Args :
                X (list) : input value of feature
                Y (list) : corresponding output values of feature.

        """
        self.input = np.array(X)
        self.output = np.array(Y)

        self.mean_x = sum(self.input)/len(self.input)
        self.mean_y = sum(self.output)/len(self.output)

        self.mean_sq_x = np.dot(self.input, self.input)/len(self.input)
        self.mean_xy = np.dot(self.input, self.output)/len(self.input)

        self.a1 = (self.mean_xy - self.mean_x*self.mean_y) / \
            ((self.mean_sq_x) - (self.mean_x)**2)

        self.a0 = self.mean_y - self.a1*self.mean_x

    def calculate(self, x):
        """
        Main Method:
            Args:
            x (float) : value of feature
            Return:
            output value of class y(float)
        """
        return round((self.a1 * x + self.a0), 2)

# inputs = [6.2, 6.5, 5.4, 6.5, 7.1, 7.9, 8.5, 8.9, 9.5, 10.6]
# outputs = [26.3, 26.6, 25, 26, 27.9, 30.4, 35.4, 38.5, 42.6, 48.3]


inputs = [1, 2, 3, 4, 5]
outputs = [1.2, 1.8, 2.6, 3.2, 3.8]

x = 3
slr = Simple_Linear_Regression(inputs, outputs)
print("Predicted value of :", x, " is =", slr.calculate(x))

predicted_output = [slr.calculate(inputs[i]) for i in range(len(inputs))]
# mean_squared_error(Y_true,Y_pred)
print("error:", mean_squared_error(outputs, predicted_output))
print("accuracy:", 100-(mean_squared_error(outputs, predicted_output)*100))
```

OUTPUT:

Predicted value of : 3 is = 2.52

error: 0.0024000000000000004

accuracy: 99.76

2. Implement multiple linear regression model for below data, calculate accuracy using R^2 method.

X1	X2	Y
1	2	3
2	3	4
3	1	6
4	5	8
5	4	10
6	3	11
7	6	12
8	4	15
9	8	16

10	6	19

```
import numpy as np


class multiple_linear_regression():

    def __init__(self, X, Y):

        self.input = np.array(X)
        self.output = np.array(Y)

        new_inputs = np.insert(self.input, 0, 1, axis=1)

        self.theta = np.matmul(np.matmul(np.linalg.pinv(np.matmul(np.transpose(
            new_inputs), new_inputs)), np.transpose(new_inputs)), self.output)
        print(self.theta)

    def calculate(self, x1, x2):
```

```
        answer = self.theta[0] + self.theta[1]*x1 + self.theta[2]*x2
        print(round(answer))


inputs = [[1, 2], [2, 3], [3, 1], [4, 5], [
    5, 4], [6, 3], [7, 6], [8, 4], [9, 8]]
outputs = [3, 4, 6, 8, 10, 11, 12, 15, 16]

mlr = multiple_linear_regression(inputs, outputs)
mlr.calculate(9, 8)

OUTPUT:
[ 1.20380273  1.71657754 -0.0855615 ]
16
```

3. Implement a polynomial regression to identify the best-fit line for the below dataset.

X	Y
1	1
2	4
3	9
4	15
5	23
6	35
7	46
8	62
9	75
10	97

```
# Here is a more elegant and scalable solution, imo. It'll work for any nxn matrix and
you may find use for the other methods.
# Note that getMatrixInverse(m) takes in an array of arrays as input. Please feel free to
ask any questions
import numpy as np
import matplotlib.pyplot as plt
import sys
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures


def zeros_matrix(rows, cols):
    A = []
    for i in range(rows):
        A.append([])
        for j in range(cols):
            A[-1].append(0.0)

    return A
# # Taking a 3 * 3 matrix
# A = np.array([[6, 1, 1],
#               [4, -2, 5],
#               [2, 8, 7]])


def mulAB(A, B, result):
    # iterating by row of A
    for i in range(len(A)):
        # print("i", A[i])

        # iterating by column by B
        for j in range(len(B[0])):
            # print("j", B[j])

            # iterating by rows of B
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]

    return result


x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# x = x.reshape(-1, 1)
# y = np.array((1, 4, 9, 15, 23, 35, 46, 62, 75, 97), ndmin=2)
y = np.array([1, 4, 9, 15, 23, 35, 46, 62, 75, 97])
# y = y.reshape(10)


Matrix_X = zeros_matrix(3, 3)
row = 3
col = 3
for i in range(row):
    for j in range(col):
        Matrix_X[i][j] = 0  # firsly initilised as zero
        if i == 0 and j == 0:
            Matrix_X[i][j] = len(x)
        else:
```

```
            power_of_x = i+j
            summation_x_to_power_of_i_plus_j = 0
            for val in x:
                summation_x_to_power_of_i_plus_j = summation_x_to_power_of_i_plus_j + \
                    pow(val, power_of_x)
            Matrix_X[i][j] = summation_x_to_power_of_i_plus_j

# Calculating the inverse of the matrix
# print(np.linalg.inv(A))
Inv_Matrix_X = np.linalg.inv(Matrix_X)

# for i in range(row):
#     for j in range(col):
#         print(Matrix_X[row][col])

Matrix_Y = zeros_matrix(3, 1)


def summation_y(y):
    sum = 0
    for val in y:
        sum += val
    return sum


def summation_xy(x, y):
    sum = 0
    for i in range(len(x)):
        sum = sum+(y[i]*x[i])
    return sum


def summation_x_square_y(x, y):
    sum = 0
    for i in range(len(x)):
        sum = sum+(y[i]*(pow(x[i], 2)))
    return sum


Matrix_Y[0][0] = summation_y(y)
Matrix_Y[1][0] = summation_xy(x, y)
Matrix_Y[2][0] = summation_x_square_y(x, y)

# --------------------------------
result = [[0],
          [0],
          [0]]

Ans_as_all_coefficient = mulAB(Inv_Matrix_X, Matrix_Y, result)
# print(Matrix_X)
# print(Inv_Matrix_X)
```

```
# print(Matrix_Y)
print(Ans_as_all_coefficient)

# for i in range(len(Ans_as_all_coefficient)):
#     for j in range(len(Ans_as_all_coefficient[0])):
#         print(Ans_as_all_coefficient[i][j])
print("best fit line will be:[a0+a1x+a2x^2]:", str(Ans_as_all_coefficient[0][0]) +
      " + " + str(Ans_as_all_coefficient[1][0])+" *x  +
"+str(Ans_as_all_coefficient[2][0])+" *x^2")


OUTPUT:
[[0.6666666666671972], [-0.34242424242438574], [0.9848484848484915]]
best fit line will be:[a0+a1x+a2x^2]: 0.6666666666671972 + -0.34242424242438574 *x  +
0.9848484848484915 *x^2
```

4. Implement logistic regression for below dataset.

Amount in savings	Loan defaulter
0.5	0
1.0	0
1.25	0
2.5	0
3.0	0
1.75	1
4.0	1
4.25	1
4.75	1
5.0	1

****

```
import numpy as np
import math


class Logistic_Regression():

    """
        Class to calculate simple regression
```

```
    """

    def __init__(self, X, Y):
        """
            Constructor:
                Args :
                X (list) : input value of feature
                Y (list) : corresponding output values of feature.

        """
        self.input = np.array(X)
        self.output = np.array(Y)

        self.mean_x = sum(self.input)/len(self.input)
        self.mean_y = sum(self.output)/len(self.output)

        self.mean_sq_x = np.dot(self.input, self.input)/len(self.input)
        self.mean_xy = np.dot(self.input, self.output)/len(self.input)

        self.a1 = (self.mean_xy - self.mean_x*self.mean_y) / \
            ((self.mean_sq_x) - (self.mean_x)**2)

        self.a0 = self.mean_y - self.a1*self.mean_x

    def sigmoid(self, x):
        return 1/(1 + math.e**(-(self.a0 + self.a1*x)))

    def calculate(self, x):
        if (self.sigmoid(x) >= 0.5):
            return 1
        else:
            return 0

    def log_fun(self, input):
        # squares = [x*x for x in range(11)]

        # lst = [self.calculate(val) for val in self.input]
        lst = []

        for value in input:
            lst.append(self.calculate(value))
        return lst


input = np.array([0.5, 1.0, 1.25, 2.5, 3.0, 1.75, 4.0, 4.25, 4.75, 5.0])
output = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])


def my_accuracy(lst1, lst2):
    count_yes = 0
```

```
    for i in range(len(lst1)):
        if lst1[i] == lst2[i]:
            count_yes = count_yes+1
    return (count_yes)/len(lst1)

# model.score(input, output)
# print("score:", model.score(input, output))


x = 7
myObj = Logistic_Regression(input, output)
# print("Predicted value of ", x, " is ", slr.calculate(x))
pred_ans = myObj.log_fun(input)
print(myObj.log_fun(input))
print("accuracy:", my_accuracy(pred_ans, output))

OUTPUT:
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
accuracy: 0.6
```