Assignment 03

EE-527 Machine Learning Laboratory

Group Members

RAJENDRA KUJUR (214161008)
ROHIT RAJ SINGH CHAUHARN (214161009)

Problem 1

Write the following function in python to generate n number of points around the line y = ax + b [y outlier , y noisy , y actual] = generateDataSet(a, b, xmin, xmax, n, α, σ) where x ∈ [x min , x max] , σ is the standard deviation of additive white noise and α is the fraction of outliers present in the data (α ∈ (0, 1)). The output of the function is obtained as follows y actual (i) = ax(i) + b (1) y noisy (i) = y actual (i) + σN (0, 1) y outlier = outlierCorruption(y noisy , α) Display the scatter plot of the dataset. Plot the inliers in BLUE and outliers in RED.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import random

def line(a, b, x):
    return a*x + b

def generateDataSet(a, b, x_min, x_max, n, alpha, sigma):
    x = (np.array([(np.random.random()-0.5)*10 for i in range(n)])).reshape(n,1)

    # Since alpha is fraction of outliers present
    number_of_outliers = int(alpha*n)
    # assumed 50% points more than outliers are noise
    number_of_noise = int((alpha + 0.5)*n)
    # remaining points
    number_of_inliners = n - number_of_outliers - number_of_noise

    # generate y_actual valeus
    y_actual = line(a, b, x[:number_of_inliners])

    # generate y_noise values
    y_noise = line(a, b, x[number_of_inliners:number_of_inliners + number_of_noise]

    sequence = [-20, 20]
    # generate y_outliers
    y_outlier = line(a, b, x[number_of_inliners + number_of_noise:]) + sigma*np.ran

    # Plot all the points
    plt.plot(x[:number_of_inliners,:],y_actual,'bo',x[number_of_inliners:number_of_
    plt.title(f'Points at alpha = {alpha}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

    return x, y_outlier, y_noise, y_actual

x, y_outlier, y_noise, y_actual = generateDataSet(2, 3, -5, 5, 100, 0.2, 1)
```

Problem 2

Perform Regression Diagnostics and display the line obtained in each iteration. Please note that the outliers detected in each iteration should be marked in red color. Experiment with different values of α.

In [ ]:

```
def secondProblem(x, y_outlier, y_noise, y_actual):
    y = np.concatenate((y_actual,y_noise,y_outlier))

    import copy
    real_x = copy.deepcopy(x)
    real_y = copy.deepcopy(y)

    from numpy.lib.type_check import real
    from sklearn.linear_model import LinearRegression
    l_list=[2.5]

    for l in l_list:
        x=copy.deepcopy(real_x)
        y=copy.deepcopy(real_y)
        print("value of lembda : ",l)

        for epoch in range(10):
            reg = LinearRegression().fit(x, y)
            # reg.score(x, y)

            y_pred=reg.predict(x)
            error=y-y_pred
            mean=np.mean(error)
            var=np.var(error)
            std=var**(1/2)
            new_X=[]
            new_y=[]
            out_X=[]
            out_y=[]
            for i in range(len(error)):
                if(abs(error[i])>=mean+l*std):
                    out_X.append(x[i])
                    out_y.append(y[i])
                else:
                    new_X.append(x[i])
                    new_y.append(y[i])

            print("Iteration number :",epoch+1,"   number of outlier:",len(out_X),"
            plt.plot(new_X,new_y,'bo',out_X,out_y,'ro')
            x_line = np.linspace(-5,5,100)
            y_line= reg.coef_[0]*x_line+reg.intercept_
            plt.plot(x_line, y_line, 'black')
            plt.xlabel('x')
            plt.ylabel('y')
            x=new_X
            y=new_y
            plt.show()


alpha_values = [0.12, 0.18, 0.2]
```

For alpha = 0.12

In [ ]:

```
x, y_outlier, y_noise, y_actual = generateDataSet(2, 3, -5, 5, 100, alpha_values[0]
secondProblem(x, y_outlier, y_noise, y_actual)
```

For alpha = 0.18

In [ ]:

```
x, y_outlier, y_noise, y_actual = generateDataSet(2, 3, -5, 5, 100, alpha_values[1]
secondProblem(x, y_outlier, y_noise, y_actual)
```

For alpha 0.2

In [ ]:

```
x, y_outlier, y_noise, y_actual = generateDataSet(2, 3, -5, 5, 100, alpha_values[2]
secondProblem(x, y_outlier, y_noise, y_actual)
```

Problem 3

Perform RANSAC on the above set of points and plot the output of each trial. Identify and plot the final line.

In [ ]:

```
# First concatenate all of the co-ordinates
y = np.concatenate((y_actual,y_noise,y_outlier))

# faltten x and y
x = (x).flatten()
y = (y).flatten()

# returns the y value from the given line Ax + By + C = 0
def line(A, B, C, x):
    return (-C/B) - (A/B)*x

# Randomly generates two points indices
def randomIndexGenerator():
    first_point_index = random.randint(0, 99)
    second_point_index = random.randint(0, 99)

    while first_point_index == second_point_index:
        second_point_index = random.randint(0, 99)
    return x[first_point_index], y[first_point_index], x[second_point_index], y[sec


iterations = 100
tolerance = 1
count_inliers = []
point_x = []
point_y = []


for _ in range(iterations):
    x1, y1, x2, y2 = randomIndexGenerator()

    # line equation is (y1-y2)*x + (x2-x1)*y + (x1*y2 -x2*y1) = 0 in the form of Ax
    A = y1 - y2
    B = x2 - x1
    C = x1*y2 - x2*y1

    # Now lets find each points distance from the line forming by choosing these tw
    # from formula |Ax + By + C| ((A**2 + B**2)**0.5)
    inliers = 0
    for index in range(len(x)):
        distance = abs(A*x[index] + B*y[index] + C)/ ((A**2 + B**2) ** 0.5)
        if distance < tolerance:
            inliers += 1

    print(f'Iteration number : {_+1}',end=' ')
    print(f'Number of inliers : {inliers}')

    # store each iteration's inliers count and x and y co ordinates for last
    count_inliers.append(inliers)
    point_x.append([x1, x2])
    point_y.append([y1, y2])

    # To store the current points co ordinates an make a line joining them
    line_x_coordinate = [x1, x2]
    line_y_coordinate = [y1, y2]

    # Plot the graph at each iteration
    # for all the points make scatter graph
    plt.scatter(x,y)
```

```
    # for the generated points make a line joining both the points
    plt.plot(line_x_coordinate, line_y_coordinate, 'red', linewidth = 2.0)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('')
    plt.show()

    print('**********************************************************************

max_in = max(count_inliers)
print(f'Highest number of inliers : {max_in}')

# Find the index which is having highest number of inliners so that corresponding l
index = count_inliers.index(max_in)

# fetch x and y coordinates of both points
x1, x2 = point_x[index][0], point_x[index][1]
y1, y2 = point_y[index][0], point_y[index][1]

# Now form the equation of line
# line equation is (y1-y2)*x + (x2-x1)*y + (x1*y2 -x2*y1) = 0 in the form of Ax + B
A = y1 - y2
B = x2 - x1
C = x1*y2 - x2*y1

# make equally spaced x points
line_x = np.linspace(-5, 5, 100)
# for every x calculate y values
line_y = [line(A, B, C, x) for x in line_x]

# plot the line from x_min ot x_max with the calcuated line equation
plt.plot(line_x, line_y, 'red')
plt.scatter(x, y)
plt.title(f'Plot with Highest number of inliers in {iterations} iterations')
plt.xlabel('x')
plt.ylabel(f'y')
plt.show()
```