Jay Ellis

Math 364

Project Report

4/25/2024

LIGO Sensor Formula

Preface:

I started work on this project on April 10, 2024 and have completed a sensor reading formula to determine whether or not the gravitational waves detected are from space or not. I was given a set of training data and a set of test data, along side the answers of whether or not it was a false reading or a positive reading. The training data contained 20 entries and the testing data contained 20 entries. I have some concerns on whether or not my equation will work in all circumstances due to this, so I will include the code I used to generate my linear program that created the equation I got as a result. This is so if you happen to have a more exhaustive data set than the one I was provided, you can change the equation to be more accurate for those fringe cases not given originally. The code I used has been attached.

Method:

I used python code with the numpy, pandas, and scipy.optimize libraries to create the resulting equation 9.0016309 + 0.8472103 * x_1 < x_2. The equation gives a true (1) or false (0) depending if x_2 is above the line or below the line. Additionally, the way it has been set up is to give a default false or 0 should the data point be on the line.

I created the linear program from the given training data by putting it into two categories: true and false. The true category was made from the subset of the data where the 'answer' equals 1, and the false category was made from the subset of the data where the 'answer' equals 0. The true category had a simple modification where the 'varience' was set to 1. The false category had the entire data set to their negative value and afterwards the 'varience' was set to 1. After that I recombined the two subsets of the data into one whole data set again. This results in the whole data set being ready to be subdivided into two arrays for the linear program. Matrix A is created by creating a copy of the complete data set and dropping 'answer' and x_2 columns and finally changing the dataset to a numpy array. Matrix B is the copy of the complete data set's x_2 column which is then changed into a numpy array. Finally I set my c array to have the 'varience' term be equal to 1 and the x_1 term be equal to 0, in order to have the equation properly maximize the resulting line's distance from the closest points.

Verification:

After I have the resulting['x'] from the scipy.optimize.linprog() function I then used two functions to test my results. I used flag_data() where it will go through the mathematical steps of 'varience' + slope * x_1_test and see if that sum is less than x_2_test and return 1 if true and 0 otherwise. I then made test_data_flag() which automized going through the two test files I was given and putting said data into flag_data() and printing out the result of flag_data() along side the expected result from the test data answers file. I have attached the output I had, and I don't see any problems with my results according to the test data due to scoring 100% correct with the test data.

Python Code Used:

```
import numpy as np
import pandas as pd
import scipy.optimize as opt
import matplotlib.pyplot as plt

print("Start math364 Project")

##### these are to be used to check your test data #####
test_data = np.genfromtxt("2d_test_data.txt", delimiter=None, dtype=float,
missing_values=None)
#print(test_data) #ok works properly #pre sorted/split
#2d_true_classification.txt
true_classification_data = np.genfromtxt("2d_true_classification.txt",
delimiter=None, dtype=float, missing_values=None)
#print(true_classification_data) #ok works properly #pre sorted/split

training_data = np.genfromtxt("2d_training_data.txt", delimiter=None,
dtype=float, missing_values=None)
#print(training_data) #ok works properly #also in the proper form for my
program #need to split into two data sets

df = pd.DataFrame(training_data, columns = ["answer", "x_1", "x_2"])
df.insert(0, "varience", 0)
#df.insert(0, "baseline", 1)
#print(df)
##### BASELINE COMPLETE #####

def negate(num):
    return -1*num

def create_true(df):
    data_true = df.copy()
    data_true = data_true.loc[df['answer'] == 1]
    #equation form needed
    #x_1 + baseline + varience <= x_2
    data_true['varience'] = 1
    return data_true

def create_false(df):
    data_false = df.copy()
    data_false = data_false.loc[df['answer'] == 0]
    #equation form needed
    #x_1 + baseline - varience >= x_2
    #-x_1 - baseline + varience <= -x_2
    data_false = data_false.apply(negate)
    data_false['varience'] = 1
    return data_false

def get_matrixA(df):
    temp = df.copy()
    temp.drop('answer', inplace = True, axis=1)
    temp.drop('x_2', inplace = True, axis=1)
    temp = temp.to_numpy()
    return temp
```

```python
def get_matrixB(df):
    temp = df['x_2']
    temp = temp.to_numpy()
    return temp

def fuse_true_false(data_true, data_false):
    data_full = pd.concat([data_true, data_false])
    return data_full

##### BASIC FUNCTIONS COMPLETE #####

data_true = create_true(df)
data_false = create_false(df)
data_full = fuse_true_false(data_true, data_false)
##### CREATING BASELINE MATRIX COMPLETE #####

A = get_matrixA(data_full)
B = get_matrixB(data_full)
#layout of A
#baseline, varience, x_1
c = np.array([1,0]) #what you maximize/minimize
res=opt.linprog(-c,A,B,None,None,bounds = (0,None)) #currently maximized
#print(res)
##### LP processing complete #####

"""
next need to plug x array into an equation where
varience + x_1 < x_2 therefore y = 0
varience + x_1 > x_2 therefore y = 1
THE LINE < VALUE Y = 0
THE LINE > VALUE Y = 1
"""

def flag_data(line_array, data_array):
    answer = 0
    #put my equation here
    x = line_array[0]
    x += line_array[1] * data_array[0]
    if(x < data_array[1]):
        answer = 1
    return answer

line = res['x']
# data = [51.2,95.0]
# print(flag_data(line, data))

##### POST PROCESSING #####

def test_data_flag(line, test_data, test_classification):
    file = open("math364_output.txt", "a")
    file.write("Jay Ellis\nMath364\nProject\n4/28/2024\n")
    print("\nEquation: [delta, x_1]\nEquation: ", line, "\n", file=file)
    #print("\nEquation: [delta, x_1]\nEquation: ", line, "\n")
    print("\nEquation: [x_1, x_2]", file=file)
    #print("[x_1, x_2]")
    y = 0
    for x in test_data:
```

```
        print(x, "result:", flag_data(line, x), "expected:",
test_classification[y], file=file)
        #print(x, "result:", flag_data(line, x), "expected:",
test_classification[y])
        #file.write(x, "result:", flag_data(line, x), "expected:",
test_classification[y],"\n")
        y += 1
    file.close()
    return

def plot_data(line, test_data):
    print("start plot")
    base = line[0]
    slope = line[1]
    start = 0
    stop = 100
    resolution = 50
    xs = np.linspace(start, stop, resolution)
    ys = base + slope*xs
    plt.plot(xs,ys)

    y = 0
    for x in test_data:
        #print(x, "result:", flag_data(line, x), "expected:")
        if(flag_data(line, x) == 0):
            plt.scatter(x[0], x[1], color = 'red')
        else:
            plt.scatter(x[0], x[1], color = 'green')
        y += 1
    plt.show()
    print("end plot")
    return

test_data_flag(line, test_data, true_classification_data)
plot_data(line, test_data)


def get_plotable_coordinates(df):
    temp = df.copy()
    temp.drop('answer', inplace = True, axis=1)
    temp.drop('varience', inplace = True, axis=1)
    temp = temp.to_numpy()
    return temp

plot_data(line, get_plotable_coordinates(df))
print("End of Program")
```

Test Output From Code:

Equation: [delta, x_1]

Equation: [9.0016309 0.8472103]

Equation: [x_1, x_2]

[64.1 85.3] result: 1 expected: 1.0

[59.3 26.] result: 0 expected: 0.0

[84. 50.9] result: 0 expected: 0.0

[51.1 75.3] result: 1 expected: 1.0

[14.8 82.] result: 1 expected: 1.0

[68.3 78.7] result: 1 expected: 1.0

[19.2 80.2] result: 1 expected: 1.0

[19.1 8.2] result: 0 expected: 0.0

[85.5 86.1] result: 1 expected: 1.0

[87.7 47.2] result: 0 expected: 0.0

[27.4 0.7] result: 0 expected: 0.0

[64.6 72.] result: 1 expected: 1.0

[83.6 28.2] result: 0 expected: 0.0

[21.5 63.9] result: 1 expected: 1.0

[80.5 96.4] result: 1 expected: 1.0

[15.1 48.2] result: 1 expected: 1.0

[89.5 42.3] result: 0 expected: 0.0

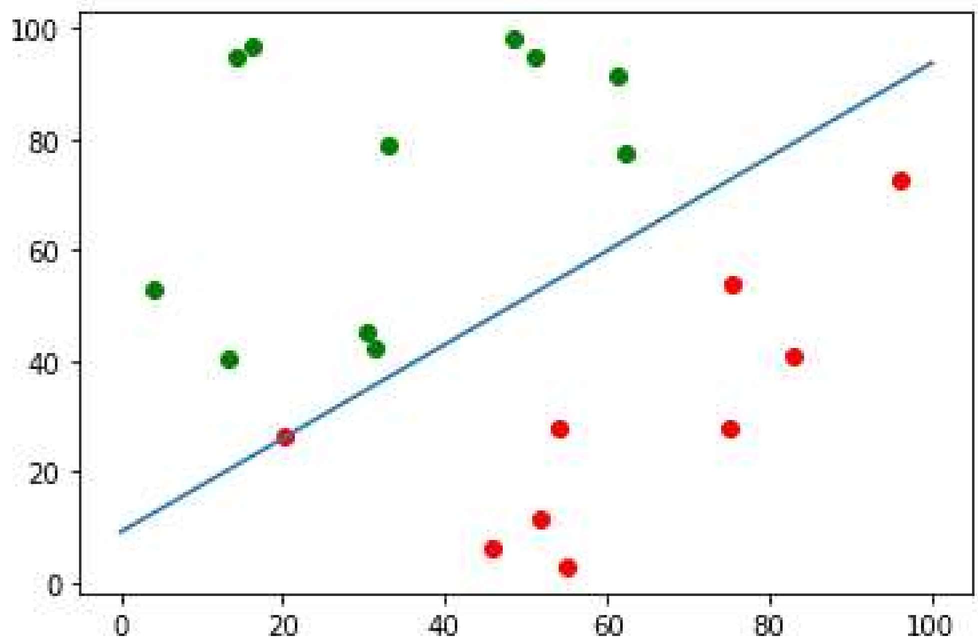
[59.   2.4] result: 0 expected: 0.0

[67.3 91.9] result: 1 expected: 1.0

[82.7 88.6] result: 1 expected: 1.0

Graphs:

Training Data:



Test Data: