**Symbolic Regression using Random Search**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **tqdm.notebook** **import** tqdm

**from** **sklearn.metrics** **import** mean\_squared\_error

**import** **operator**

**import** **random**

**from** **random** **import** randint,seed

data = np.genfromtxt("data.txt",delimiter=",")

df = pd.DataFrame(data,columns=["x","y"])

feature\_names = ['x']

target\_name = 'y'

X = df[feature\_names]

y = df[target\_name]

**def** **div**(a, b):

**return** a / b **if** b **else** a

**def** **cos**(a):

**return** np.cos(a)

**def** **sin**(a):

**return** np.sin(a)

**def** **exp2**(a):

**return** a\*\***2**

**def** **exp3**(a):

**return** a\*\***3**

**def** **generate\_function**(depth):

**if** randint(**0**, **10**) >= depth\***2**:

oper = operations[randint(**0**, len(operations) - **1**)]

**return** {

"func": oper["func"],

"children": [generate\_function(depth + **1**) **for** \_ **in** range(oper["arg\_count"])],

"format\_str": oper["format\_str"],

}

**else**:

**return** {"feature\_name": features[randint(**0**, len(features) - **1**)]}

**def** **string\_of\_function**(node):

**if** "children" **not** **in** node:

**return** node["feature\_name"]

**return** node["format\_str"].format(\*[string\_of\_function(c) **for** c **in** node["children"]])

operations = (

{"func": operator.add, "arg\_count": **2**, "format\_str": "({} + {})"},

{"func": operator.sub, "arg\_count": **2**, "format\_str": "({} - {})"},

{"func": operator.mul, "arg\_count": **2**, "format\_str": "({} \* {})"},

{"func": div, "arg\_count": **2**, "format\_str": "({} / {})"},

{"func": cos, "arg\_count": **1**, "format\_str": "np.cos({})"},

{"func": sin, "arg\_count": **1**, "format\_str": "np.sin({})"},

{"func": exp2, "arg\_count": **1**, "format\_str": "({} \*\* 2)"},

{"func": exp3, "arg\_count": **1**, "format\_str": "({} \*\* 3)"}

)

features = ['x',**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**,**10**]

#Random Search

#First we generate a random function using the given set of operators and variables

#Then we fit the x values in that function to obtain a list of y values

#We find the Mean Squared Error between calculated y and iven y values

#We obtain the fitness using 1/MSE

#Check if fitness has improved and if so, append

best\_fitness = **0**

fitness\_evolution\_list = []

**for** epoch **in** tqdm(range(**1000**)):

eq = str(string\_of\_function(generate\_function(**0**))).replace(" ","")

y\_pred = []

X.reset\_index(drop=True)

**for** i **in** range(len(X)):

x = X.iloc[i]

**try**:

pred = float(eval(eq))

**except** (**SyntaxError**, **NameError**, **TypeError**, **ZeroDivisionError**):

pred = **0.0**

**if** type(pred)!=float:

pred = pred['x']

**if** pred == np.inf **or** pred == -np.inf **or** np.isnan(pred):

pred = **0**

y\_pred.append(pred)

fitness = **1**/mean\_squared\_error(y, y\_pred)

**if** fitness > best\_fitness:

best\_fitness = fitness

best\_fit\_function = eq

fitness\_evolution\_list.append((epoch,eq,fitness,y\_pred))

**print**("Epoch "+str(epoch)+": "+str(best\_fit\_function)+"**\n**Fitness :"+str(fitness),end= "**\r**")

#Plotting the new curve along side the original one

plt.plot(data[:,**0**],fitness\_evolution\_list[-**1**][**3**])

plt.plot(data[:,**0**],data[:,**1**])

plt.show()