**2) Non - Polynomial Regression –**

**Description:** If the data shows a curvy trend, then linear regression will not produce very accurate results when compared to a non-linear regression because, as the name implies, linear regression presumes that the data is linear. Let's learn about nonlinear regressions and apply an example on python. In this notebook, we fit a non-linear model to the data points corresponding to China's GDP from 1960 to 2014.

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

**# import data**

df = pd.read\_csv("china\_gdp.csv")

x\_data, y\_data = (df["Year"].values, df["Value"].values)

**# build model**

def sigmoid(x, Beta\_1, Beta\_2):

y = 1 / (1 + np.exp(-Beta\_1\*(x-Beta\_2)))

return y

beta\_1 = 0.10

beta\_2 = 1990.0

#logistic function

Y\_pred = sigmoid(x\_data, beta\_1 , beta\_2)

**#plot initial prediction against datapoints**

plt.plot(x\_data, Y\_pred\*15000000000000.)

plt.plot(x\_data, y\_data, 'ro')

# Our task here is to find the best parameters for our model. Lets first normalize our x and y:

# Lets normalize our data

xdata =x\_data/max(x\_data)

ydata =y\_data/max(y\_data)

**# using curve fit**

from scipy.optimize import curve\_fit

popt, pcov = curve\_fit(sigmoid, xdata, ydata)

#print the final parameters

print(" beta\_1 = %f, beta\_2 = %f" % (popt[0], popt[1]))

**# plot new fit**

x = np.linspace(1960, 2015, 55)

x = x/max(x)

plt.figure(figsize=(8,5))

y = sigmoid(x, \*popt)

plt.plot(xdata, ydata, 'ro', label='data')

plt.plot(x,y, linewidth=3.0, label='fit')

plt.legend(loc='best')

plt.ylabel('GDP')

plt.xlabel('Year')

plt.show()

**# Evaluation**

# split data into train/test

msk = np.random.rand(len(df)) < 0.8

train\_x = xdata[msk]

test\_x = xdata[~msk]

train\_y = ydata[msk]

test\_y = ydata[~msk]

# build the model using train set

popt, pcov = curve\_fit(sigmoid, train\_x, train\_y)

# predict using test set

y\_hat = sigmoid(test\_x, \*popt)

# evaluation

print("Mean absolute error: %.2f" % np.mean(np.absolute(y\_hat - test\_y)))

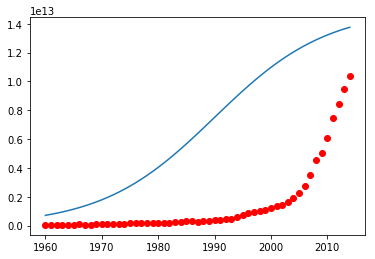
print("Residual sum of squares (MSE): %.2f" % np.mean((y\_hat - test\_y) \*\* 2))

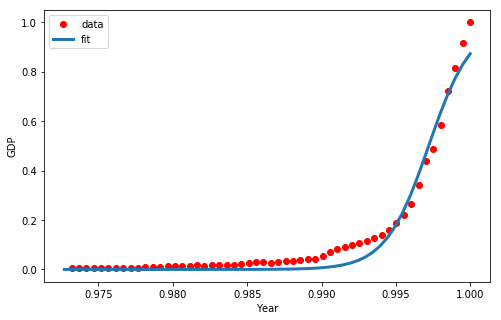
from sklearn.metrics import r2\_score

print("R2-score: %.2f" % r2\_score(y\_hat , test\_y) )

**Output:**

beta\_1 = 690.451710, beta\_2 = 0.997207





Mean absolute error: 0.03

Residual sum of squares (MSE): 0.00

R2-score: 0.97