**Non Linear Regression Analysis**

Estimated time needed: **20** minutes

**Objectives**

After completing this lab you will be able to:

* Differentiate between linear and non-linear regression
* Use non-linear regression model in Python

If the data shows a curvy trend, then linear regression will not produce very accurate results when compared to a non-linear regression since linear regression presumes that the data is linear. Let's learn about non linear regressions and apply an example in python. In this notebook, we fit a non-linear model to the datapoints corrensponding to China's GDP from 1960 to 2014.

**Importing required libraries**

[ ]:



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

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

**%**matplotlib inline

Although linear regression can do a great job at modeling some datasets, it cannot be used for all datasets. First recall how linear regression, models a dataset. It models the linear relationship between a dependent variable y and the independent variables x. It has a simple equation, of degree 1, for example y = 2𝑥2x + 3.

[ ]:



x **=** np.arange(**-**5.0, 5.0, 0.1)

​

*##You can adjust the slope and intercept to verify the changes in the graph*

y **=** 2**\***(x) **+** 3

y\_noise **=** 2 **\*** np.random.normal(size**=**x.size)

ydata **=** y **+** y\_noise

*#plt.figure(figsize=(8,6))*

plt.plot(x, ydata, 'bo')

plt.plot(x,y, 'r')

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

Non-linear regression is a method to model the non-linear relationship between the independent variables 𝑥x and the dependent variable 𝑦y. Essentially any relationship that is not linear can be termed as non-linear, and is usually represented by the polynomial of 𝑘k degrees (maximum power of 𝑥x). For example:

 𝑦=𝑎𝑥3+𝑏𝑥2+𝑐𝑥+𝑑  y=ax3+bx2+cx+d

Non-linear functions can have elements like exponentials, logarithms, fractions, and so on. For example:

𝑦=log(𝑥)y=log⁡(x)

We can have a function that's even more complicated such as :

𝑦=log(𝑎𝑥3+𝑏𝑥2+𝑐𝑥+𝑑)y=log⁡(ax3+bx2+cx+d)

Let's take a look at a cubic function's graph.

[ ]:



x **=** np.arange(**-**5.0, 5.0, 0.1)

​

*##You can adjust the slope and intercept to verify the changes in the graph*

y **=** 1**\***(x**\*\***3) **+** 1**\***(x**\*\***2) **+** 1**\***x **+** 3

y\_noise **=** 20 **\*** np.random.normal(size**=**x.size)

ydata **=** y **+** y\_noise

plt.plot(x, ydata, 'bo')

plt.plot(x,y, 'r')

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

As you can see, this function has 𝑥3x3 and 𝑥2x2 as independent variables. Also, the graphic of this function is not a straight line over the 2D plane. So this is a non-linear function.

Some other types of non-linear functions are:

**Quadratic**

𝑌=𝑋2Y=X2

[ ]:



x **=** np.arange(**-**5.0, 5.0, 0.1)

​

*##You can adjust the slope and intercept to verify the changes in the graph*

​

y **=** np.power(x,2)

y\_noise **=** 2 **\*** np.random.normal(size**=**x.size)

ydata **=** y **+** y\_noise

plt.plot(x, ydata, 'bo')

plt.plot(x,y, 'r')

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

**Exponential**

An exponential function with base c is defined by

𝑌=𝑎+𝑏𝑐𝑋Y=a+bcX

where b ≠0, c > 0 , c ≠1, and x is any real number. The base, c, is constant and the exponent, x, is a variable.

[ ]:



X **=** np.arange(**-**5.0, 5.0, 0.1)

​

*##You can adjust the slope and intercept to verify the changes in the graph*

​

Y**=** np.exp(X)

​

plt.plot(X,Y)

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

**Logarithmic**

The response 𝑦y is a results of applying the logarithmic map from the input 𝑥x to the output 𝑦y. It is one of the simplest form of **log()**: i.e.

𝑦=log(𝑥)y=log⁡(x)

Please consider that instead of 𝑥x, we can use 𝑋X, which can be a polynomial representation of the 𝑥x values. In general form it would be written as

𝑦=log(𝑋)y=log⁡(X)

[ ]:



X **=** np.arange(**-**5.0, 5.0, 0.1)

​

Y **=** np.log(X)

​

plt.plot(X,Y)

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

**Sigmoidal/Logistic**

𝑌=𝑎+𝑏1+(𝑋−𝑑)Y=a+b1+c(X−d)

[ ]:



X **=** np.arange(**-**5.0, 5.0, 0.1)

​

​

Y **=** 1**-**4**/**(1**+**np.power(3, X**-**2))

​

plt.plot(X,Y)

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

**Non-Linear Regression example**

For an example, we're going to try and fit a non-linear model to the datapoints corresponding to China's GDP from 1960 to 2014. We download a dataset with two columns, the first, a year between 1960 and 2014, the second, China's corresponding annual gross domestic income in US dollars for that year.

[ ]:



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

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

​

*#downloading dataset*

**!**wget **-**nv **-**O china\_gdp.csv https:**//**cf**-**courses**-**data.s3.us.cloud**-**object**-**storage.appdomain.cloud**/**IBMDeveloperSkillsNetwork**-**ML0101EN**-**SkillsNetwork**/**labs**/**Module**%**202**/**data**/**china\_gdp.csv

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

df.head(10)

**Did you know?** When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](http://cocl.us/ML0101EN-IBM-Offer-CC)

**Plotting the Dataset**

This is what the datapoints look like. It kind of looks like an either logistic or exponential function. The growth starts off slow, then from 2005 on forward, the growth is very significant. And finally, it decelerate slightly in the 2010s.

**Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in.**[**Learn more here.**](https://cocl.us/ibm_watson_studio_infobox)

[ ]:



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

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

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

plt.ylabel('GDP')

plt.xlabel('Year')

plt.show()

**Choosing a model**

From an initial look at the plot, we determine that the logistic function could be a good approximation, since it has the property of starting with a slow growth, increasing growth in the middle, and then decreasing again at the end; as illustrated below:

[ ]:



X **=** np.arange(**-**5.0, 5.0, 0.1)

Y **=** 1.0 **/** (1.0 **+** np.exp(**-**X))

​

plt.plot(X,Y)

plt.ylabel('Dependent Variable')

plt.xlabel('Independent Variable')

plt.show()

The formula for the logistic function is the following:

𝑌̂ =11+𝑒𝛽\_1(𝑋−𝛽\_2)Y^=11+eβ\_1(X−β\_2)

𝛽\_1β\_1: Controls the curve's steepness,

𝛽\_2β\_2: Slides the curve on the x-axis.

**Building The Model**

Now, let's build our regression model and initialize its parameters.

[ ]:



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

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

**return** y

Lets look at a sample sigmoid line that might fit with the data:

[ ]:



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)

**How we find the best parameters for our fit line?**

we can use **curve\_fit** which uses non-linear least squares to fit our sigmoid function, to data. Optimal values for the parameters so that the sum of the squared residuals of sigmoid(xdata, \*popt) - ydata is minimized.

popt are our optimized parameters.

[ ]:



**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]))

Now we plot our resulting regression model.

[ ]:



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()

**Practice**

Can you calculate what is the accuracy of our model?

[ ]:



*# write your code here*

​

​

​

Click here for the solution

*# 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) )

**Want to learn more?**

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: [SPSS Modeler](https://www.ibm.com/analytics/spss-statistics-software?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at [Watson Studio](https://www.ibm.com/cloud/watson-studio?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

**Thank you for completing this lab!**

**Author**

Saeed Aghabozorgi

**Other Contributors**

[Joseph Santarcangelo](https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

**Change Log**

| **Date (YYYY-MM-DD)** | **Version** | **Changed By** | **Change Description** |
| --- | --- | --- | --- |
| 2020-11-03 | 2.1 | Lakshmi | Made changes in URL |
| 2020-08-27 | 2.0 | Lavanya | Moved lab to course repo in GitLab |
|  |  |  |  |
|  |  |  |  |

**© IBM Corporation 2020. All rights reserved.**

Simple

0

4

Python | Idle

Mem: 342.05 / 6144.00 MB

Saving completed

ML0101EN-Reg-NoneLinearRegression.ipynb

English (American)

Ln 1, Col 1

Mode: Command