Tutorial

Linear Regression in Python Using NumPy

Dr. Walter Filgueira de Azevedo Jr.

walter@azevedolab.net

Abstract

In this tutorial, it is shown how to program a simple linear regression analysis using *polyfit()* function available in the NumPy library. The code is straightforward and has been posted for educational purposes. The program reads a csv file and selects two columns of this file to carry out a linear regression analysis. The program shows the results on the screen and generates a plot file. The code itself is not optimized and is provided as it is with no guarantees (GNU license). To run the program described here; it is necessary to have Python 3 installed. We also need the NumPy and Matplotlib libraries.

Keywords: Linear regression; python; NumPy; csv file; *polyfit()*

Python libraries to be used in this tutorial.

In the development of a machine-learning model to predict any response variable for a given system, the goal is to determine the relative weights (γ_j) of the explanatory variables, to bring the predicted values (f_i) close to the experimental values (y_i). In equation (1), we expressed the response variable (f) as a function of the explanatory variables (x_j),

$$f(x_1, \ldots, x_N) = \gamma_0 + \sum_{j=1}^{N} \gamma_j x_j \quad \text{(Eq. 1)}$$

Where N indicates the number of explanatory variables and γ_0 represents the regression constant.

Python libraries to be used in this tutorial.

Among the supervised machine learning techniques, the oldest method is the ordinary linear regression method. The first statement of the ordinary linear regression method come out in the form of an appendix entitled “Sur la Méthode des moindres quarrés” in Legendre’s Nouvelles méthodes pour la détermination des orbites des comètes, Paris 1805 [1]. Legendre originally proposed this method in 1805 in a study of orbits of comets. The significant progress in the research of celestial mechanics that occurred during the early years of the nineteenth century was mainly due to the development of the ordinary linear regression method. The basic idea behind the ordinary linear regression method is to minimize the cost function known as the residual sum of squares (RSS).

Python libraries to be used in this tutorial.

Some authors call this cost function the sum of squared residuals (SSR) [2, 3]. Below we have the equation for RSS,

$$RSS = \sum_{i=1}^{M} \left(y_i - f(x_1, \ldots, x_N)\right)^2 \text{(Eq. 2)}$$

In the above equation, M is the number of observations, y_i is the experimental value, and f_i is the predicted value. RSS is the sum of the differences between the experimental value (y_i) and the predicted value (f_i). The regression method optimizes the weights (γ_j) in the equation (2) to minimize the RSS.

Python libraries to be used in this tutorial.

In this tutorial, we describe all lines of the code *linear_regression.py*. Since it is a short program, anyone with some basic knowledge of Python can easily follow the description using the comments in the code itself. Below we have the main function.

```
def main():
    # Set attributes of the LinReg class
    file_in = "data1.csv"            # Input csv file
    plot_file = "scatter_plot.png"  # Output png file
    col_1 = 6   # Number of the columns in the csv file with reponse variable
    col_2 = 5   # Number of the columns in the csv file with explanatory variable
    deg_in =1   # Degree of the polynomial equation

    # Instantiate an object of the LinReg class
    model = LinReg(file_in,plot_file,col_1,col_2,deg_in)

    # Call read_csv() method
    model.read_csv()

    # Call best_fit() method
    model.best_fit()

    # Call plot_it() method
    model.plot_it()
main()
```

In the first lines of the *main()* function, we assign values to the variables that will define the input file, the plot file, the numbers of the columns for the response and explanatory variables, and finally the degree of the polynomial equation to approximate the data.

```
def main():
    # Set attributes of the LinReg class
    file_in = "data1.csv"            # Input csv file
    plot_file = "scatter_plot.png"  # Output png file
    col_1 = 6   # Number of the column in the csv file with response variable
    col_2 = 5   # Number of the column in the csv file with explanatory variable
    deg_in =1   # Degree of the polynomial equation

    # Instantiate an object of the LinReg class
    model = LinReg(file_in,plot_file,col_1,col_2,deg_in)

    # Call read_csv() method
    model.read_csv()

    # Call best_fit() method
    model.best_fit()

    # Call plot_it() method
    model.plot_it()
main()
```

In the sequence, we instantiate an object of the class *LinReg()* and call the methods to read a csv file (*read_csv()*), to carry out regression analysis (*best_fit()*), and to generate the plot (*plot_it()*). Then we call the *main()* function.

```
def main():
    # Set attributes of the LinReg class
    file_in = "data1.csv"            # Input csv file
    plot_file = "scatter_plot.png"  # Output png file
    col_1 = 6   # Number of the column in the csv file with response variable
    col_2 = 5   # Number of the column in the csv file with explanatory variable
    deg_in =1   # Degree of the polynomial equation

    # Instantiate an object of the LinReg class
    model = LinReg(file_in,plot_file,col_1,col_2,deg_in)

    # Call read_csv() method
    model.read_csv()

    # Call best_fit() method
    model.best_fit()

    # Call plot_it() method
    model.plot_it()
main()
```

Below, we have the definition of the *LinReg()* class, followed by the code of the constructor method. In this method, we define the attributes.

```
# Class for linear regression
class LinReg(object):
    """Class to carry out linear regression for a two-dimensional data set"""

    # Constructor method
    def __init__(self,file_in,plot_file,col_1,col_2,deg_in):

        # Set up attributes
        self.file_in = file_in
        self.plot_file = plot_file
        self.col_1 = col_1
        self.col_2 = col_2
        self.deg_in = deg_in
```

The following method of the *LinReg()* class deals with the csv file; it reads the file and assigns the columns to the explanatory and response variables to the attributes *self.x* and *self.y*, respectively. These variables will be used later in the *best_fit()* method to calculate the regression equation.

```
    # Method to read csv file
    def read_csv(self):
        """Method to read csv file"""

        # Import library
        import numpy as np

        # Read csv file and get the headers
        headers_0 = np.genfromtxt(self.file_in, dtype=None, delimiter=',',
names=True)

        # Assign headers_0 to a string
        self.headers_in = str(headers_0.dtype.names)

        # Read CSV file to get predicted and experimental data
        my_csv = np.genfromtxt (self.file_in, delimiter=",", skip_header = 1)

        # Get selected columns (self.col_1 and self.col_2) from CSV file
        self.y = my_csv[:,self.col_1]   # Response variable
        self.x = my_csv[:,self.col_2]   # Explanatory variable
```

To carry out linear regression is straight forward with the function *polyfit()* of the NumPy library. This function needs the arrays (*self.x* and *self.y*) and information about the degree of the polynomial equation *(self.deg_in).* The regression model is assigned to the variable *self.z.*

```
# Method to carry out linear regression
   def best_fit(self):
       """Method to carry out linear regression analysis using np.polyfit()"""

       # Import library
       import numpy as np

       # Least-squares polynomial fitting
       self.z = np.polyfit(self.x,self.y,self.deg_in)
       self.p = np.poly1d(self.z)

       # Equation y = ax + b
       # z array has the coefficients a = z[0] and b = z[1]
       print("a = ",self.z[0])
       print("b = ",self.z[1])
       print("Best fit polynomial equation: ",self.p)
```

In the sequence, we use the model assigned to the variable *self.z* to generate the polynomial equation. To do so, we use the function *poly1d()* of the NumPy library, as indicated below.

```
# Method to carry out linear regression
   def best_fit(self):
       """Method to carry out linear regression analysis using np.polyfit()"""

       # Import library
       import numpy as np

       # Least-squares polynomial fitting
       self.z = np.polyfit(self.x,self.y,self.deg_in)
       self.p = np.poly1d(self.z)

       # Equation y = ax + b
       # z array has the coefficients a = z[0] and b = z[1]
       print("a = ",self.z[0])
       print("b = ",self.z[1])
       print("Best fit polynomial equation: ",self.p)
```

Now we may show the constant of the regression (*self.z[0]*) and the relative weight (*self.z[1]*) of the explanatory variable.

```
# Method to carry out linear regression
   def best_fit(self):
       """Method to carry out linear regression analysis using np.polyfit()"""

       # Import library
       import numpy as np

       # Least-squares polynomial fitting
       self.z = np.polyfit(self.x,self.y,self.deg_in)
       self.p = np.poly1d(self.z)

       # Equation y = ax + b
       # z array has the coefficients a = z[0] and b = z[1]
       print("a = ",self.z[0])
       print("b = ",self.z[1])
       print("Best fit polynomial equation: ",self.p)
```

The regression analysis is done; we may now show the scatter plot of the data and the best file line using Matplotlib. We import the *plt* from the matplotlib, as shown below. Then we generate the scatter plot, with the command *plt.scatter(self.x,self.y).*

```
# Method to generate plot
def plot_it(self):
    """Method to generate scatter plot"""

    # Import library
    import matplotlib.pyplot as plt

    # Generates plot
    plt.scatter(self.x,self.y)

    # Generate plot
    plt.plot(self.x, self.p(self.x), '-')

    # Show plot
    plt.show()

    # Save plot file
    plt.savefig(self.plot_file)
```

In the sequence, we plot the linear equation. Finally, we show the results on the screen (plt.show()) and save the png file.

```
# Method to generate plot
def plot_it(self):
    """Method to generate scatter plot"""

    # Import library
    import matplotlib.pyplot as plt

    # Generates plot
    plt.scatter(self.x,self.y)

    # Generate plot
    plt.plot(self.x, self.p(self.x), '-')

    # Show plot
    plt.show()

    # Save plot file
    plt.savefig(self.plot_file)
```

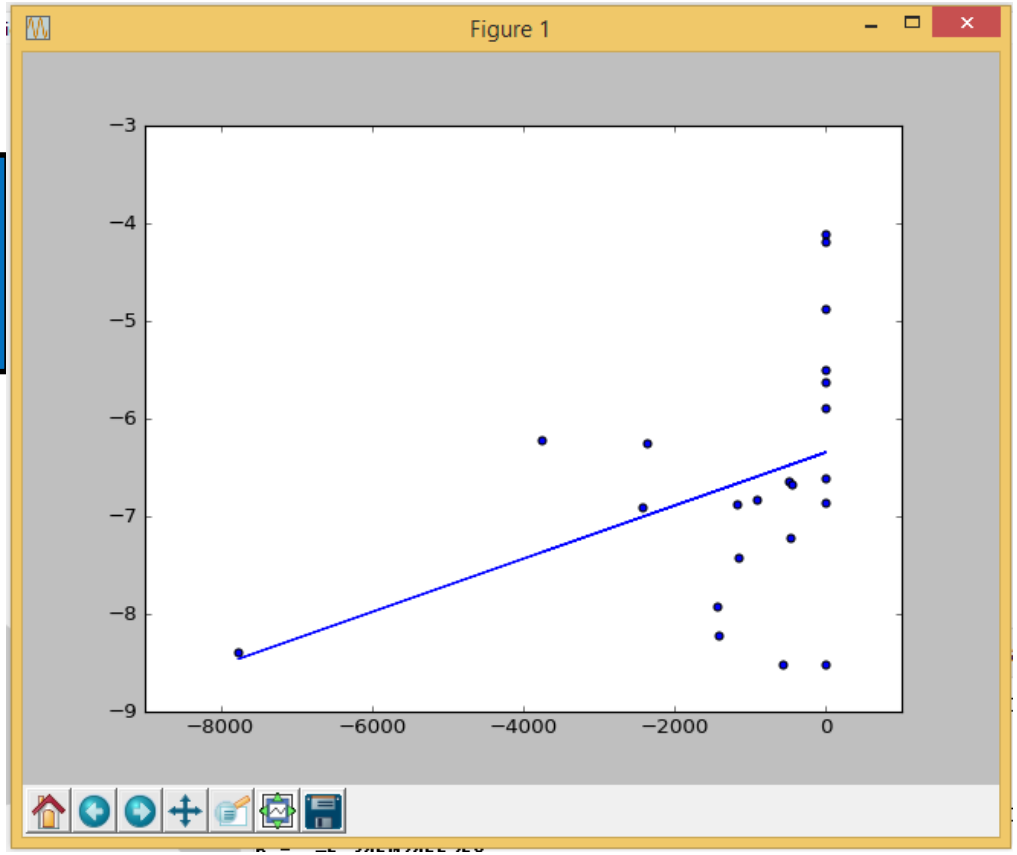
The data we use in this tutorial is in the file *data1.csv*. We are interested in columns 5 and 6, as indicated below.

					Column 5	**Column 6**
PDB	Ligand	dCS	dCF	dOS	Predicted	Experimental
1e1x	NW1	0	0	0	-6.64815	-5.88606
1h1s	4SP	7.1435	0	6.7592	-1419.64	-8.22185
1ogu	ST8	0	0	0	-6.64815	-5.61979
1pxm	CK5	7.0755	0	7.484	-469.065	-7.22185
1pxp	CK8	7.287	0	7.333	-494.269	-6.63764
2clx	F18	0	0	0	-6.64815	-4.87615
2exm	ZIP	0	0	0	-6.64815	-4.10791
2fvd	LIA	7.2072	6.9792	7.0846	-578.412	-8.52288
3blr	CPB	0	0	0	-6.64815	-8.52288
3ddq	RRC	0	0	0	-6.64815	-6.60206
3lfn	A27	0	0	0	-6.64815	-5.50031
3my5	RFZ	0	0	0	-6.64815	-4.18709
4acm	7YG	7.2549	0	6.8624	-444.847	-6.67778
4bck	T3E	7.3344	0	7.4593	-7766.76	-8.39794
4bcm	T7Z	7.2895	0	7.3211	-2433.71	-6.91009
4bcn	T9N	7.4018	0	7.6267	-1440.1	-7.92082
4bco	T6Q	7.2869	0	7.6529	-1175.44	-6.88273
4bcp	T3C	7.338	0	7.6348	-2363.4	-6.24565
4bcq	TJF	7.5638	0	7.7566	-914.491	-6.83268
4eop	1RO	7.1385	0	7.0481	-3753.54	-6.21112
4nj3	2KD	0	0	0	-6.64815	-6.85387
5d1j	56H	7.1215	0	7.6146	-1158.44	-7.42022

Running *linear_regression.py*

To run the program *linear_regression.py*, we have to go to the directory where all files are and type: *python linear_regression.py*. The results are shown below.

```
a =  0.000272509016497
b =  -6.34502455758
Best fit polynomial equation:
0.0002725 x - 6.345
```

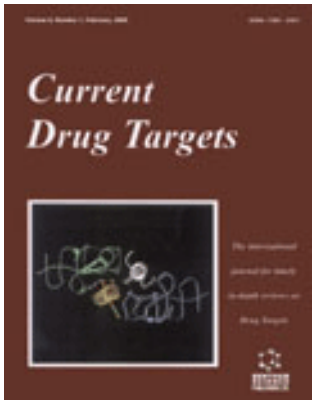


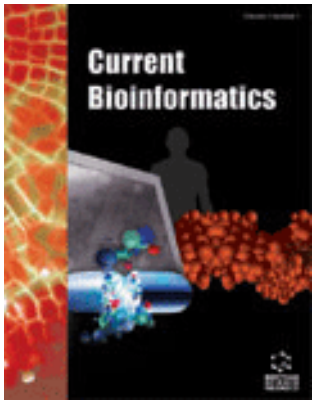
Colophon

This tutorial was produced in a PC ruuning windows 8 with 4GB of memory, a 700 GB hard disk, and an Intel® Core® i3-2120 CPU @ 3.30 GHz running Windows 8. Text and layout were generated using PowerPoint 2013 and the plot on slide 17 was generated using Matplotlib. This tutorial uses Arial font.

Author

I graduated in Physics (BSc in Physics) from the University of São Paulo (USP) in 1990. I completed a Master Degree in Applied Physics also at USP (1992), working under the supervision of Prof. Yvonne P. Mascarenhas, the founder of crystallography in Brazil. My dissertation was about X-ray crystallography applied to organometallics compounds (De Azevedo *et al.*, 1995). During my Ph.D., I worked under the supervision of Prof. Sung-Hou Kim (University of California, Berkeley. Department of Chemistry), on a split Ph.D. program with a fellowship from Brazilian Research Council (CNPq)(1993-1996). My Ph.D. was about the crystallographic structure of CDK2 (Cyclin-Dependent Kinase 2) (De Azevedo *et al.*, 1996). In 1996, I returned to Brazil. In April 1997, I finished my Ph.D. and moved to Sao Jose do Rio Preto (SP, Brazil) (UNESP) and worked there from 1997 to 2005. In 1997, I started the Laboratory of Biomolecular Systems- Department of Physics-UNESP - São Paulo State University. In 2005, I moved to Porto Alegre/RS (Brazil), where I am now. My current position is the coordinator of the Laboratory of Computational Systems Biology at Pontifical Catholic University of Rio Grande do Sul (PUCRS). My research interests are interdisciplinary with two major emphases: Bio-inspired computing and Computational Systems Biology. I published over 170 scientific papers about protein structures and computer simulation methods applied to the study of biological systems (H-index: 38). These publications have over 5000 citations.





References

[1] Legendre AM. Nouvelle méthodes pour la déterminiation des orbites des comètes, Courcier, Paris. 1805.

[2] Bell, J. Machine Learning. Hands-On for Developers and Technical Professionals; John Wiley and Sons: Indianapolis, 2015.

[3] Bruce, P.; Bruce, A. Practical Statistics for Data Scientists. 50 Essential Concepts; O'Reilly Media: Sebastopol, 2017.

Last update on January 27, 2019