Aldo Hernández

MULTIPLE LINEAR REGRESSION WITH PYTHON

Abstract *Multiple linear regression is a very common model used in machine learning, although it may not be the best option in the vast majority of cases. Nonetheless, it remains fairly useful when we need to determine a tendency in a large dataset based on certain characteristics (or labels) of interest. This document serves as an introduction to multiple linear regression and its applications with the programming language Python. In order to do this we will explore definitions, mathematical formulas, and pieces of code to make predictions on a dataset. Conclusions indicate that even though the model is not trustworthy, we can see a tendency in our set of data given one or two labels that may indicate a relationship between them and our outcome of interest. This suggests that for a small dataset where underfitting is very common, multiple linear regression should be used only for finding tendencies, instead of making our predictions on this type of model.*

Keywords multiple linear regression, python, model, tendency, machine learning

1. Introduction

Multiple linear regression is a vastly used algorithm in statistics and machine learning to make predictions and show tendencies given a certain dataset. This algorithm *finds* a linear function that indicates the tendency —usually thought as a straight line in two dimensions— and makes predictions from it. [1]

Said function is represented as it follows

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where β_i is the coefficient —also known as weight— for the i-th independent variable x that represents a characteristic –or label– that we are analyzing in the model.

If the reader is already familiarized with some concepts of machine learning, it is easy to see why linear regression is considered as a **supervised algorithm**. We should remember that the task of supervised learning is:

> "Given a training set of N example input-output pairs
> $(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$
> where each y_i was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f." [2]

So since the dataset is already labeled with information we want to take into account to predict a certain outcome, and we have our N input-output pairs given our specific tags.

It remains obvious that this definition can be applied to multiple linear regression if we change the training set from pairs to an n-tuple with $n-1$ tags and one output.

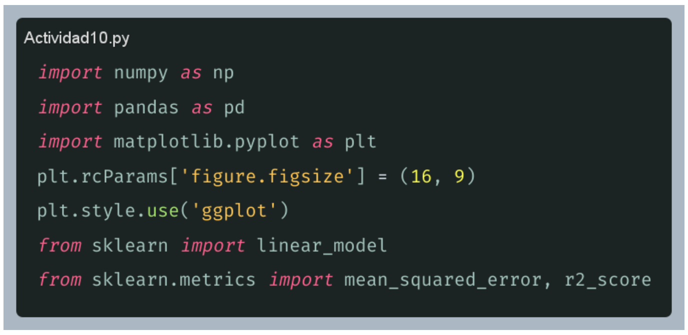
2. Methodology

2.1. Before typing code

Before we start the activity, we must download the following .csv file [1] to use the same dataset. Also, we have to import the following Python packages:

- Numpy
- Pandas
- Matplotlib
- Scikit-learn

After doing this, we are ready to import all of the needed packages to our Python main file:



Actividad10.py

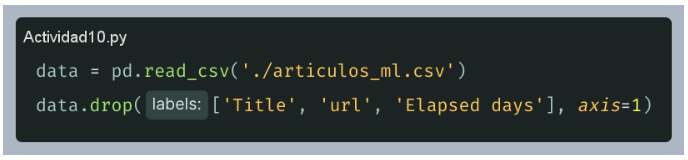
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

Figure 1. Necessary imports and starting configurations for the activity.

As the reader can see, we also made some specific configuration for the program plots, such as size (in line 4) and style (in line 5); this may be changed according to the reader's preferences.

2.2. Data processing

Next, we will read the data from the .csv file using pandas to transform it into a dataframe. This is necessary since it will help us to easily remove all the columns that we will not use for the analysis, as shown in the figure:



Actividad10.py

```python
data = pd.read_csv('./articulos_ml.csv')
data.drop( labels: ['Title', 'url', 'Elapsed days'], axis=1)
```

Figure 2. Reading the data and removing unimportant columns from the dataframe.

2.3. Filtering data

Now that we have all of our essential data, we need to filter it in order to remove random anomalies in the dataset. In order to do this, we will take a look into the dataset information with the data.describe() method, but only to our two main columns:

Table 1
Column information

	Word count	# Shares
Count	161	161
Mean	1808.26	27948.348
Std	1141.919	43408.007

As we can see in table 1, there are no null values in the columns, and our values are not very concentrated. Since we can't do really much with this information, we will graph a histogram to see where our data is really concentrated using the hist() method right after dropping the unnecessary columns:

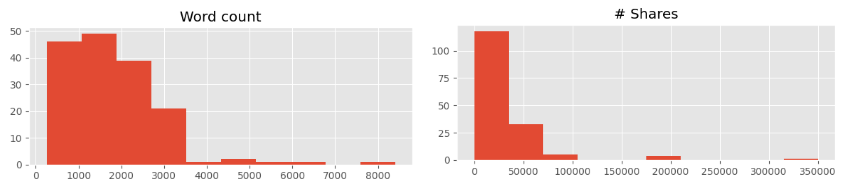
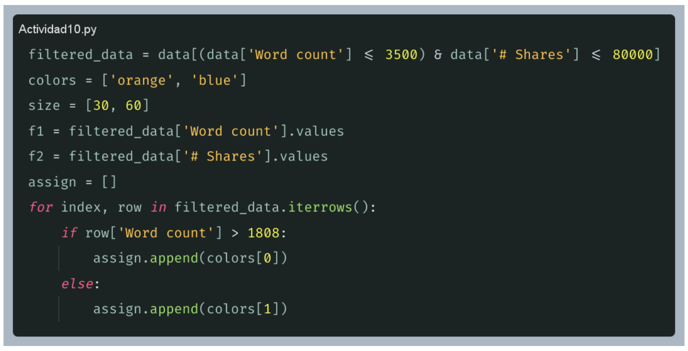


Figure 3. Data concentration histogram of the Word count and # Shares columns.

Thanks to these graphics, we can now filter the data to remove anomalies. To do so, we will simply remove all data with more than 3500 words and data with more than 80000 shares with the following code:


Actividad10.py

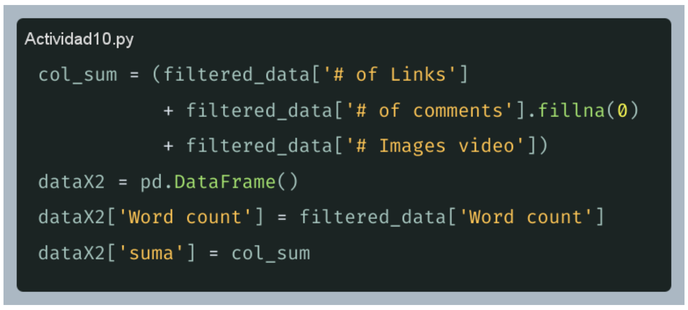
```python
filtered_data = data[(data['Word count'] ⩽ 3500) & data['# Shares'] ⩽ 80000]
colors = ['orange', 'blue']
size = [30, 60]
f1 = filtered_data['Word count'].values
f2 = filtered_data['# Shares'].values
assign = []
for index, row in filtered_data.iterrows():
    if row['Word count'] > 1808:
        assign.append(colors[0])
    else:
        assign.append(colors[1])
```

Figure 4. Data filtering from the original dataset

We also colored data with orange if they are over the word count mean, or blue otherwise.

2.4. Creating a new dataframe

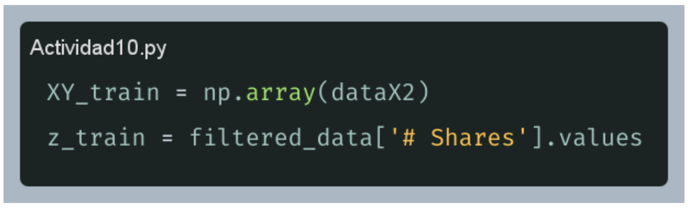
To create our multiple regression model, we will have to create a new dataframe with the columns that we will use as inputs. In this case, we will keep word count, but the other column will be the sum of links, comments, and images or videos.



Actividad10.py
```python
col_sum = (filtered_data['# of Links']
           + filtered_data['# of comments'].fillna(0)
           + filtered_data['# Images video'])
dataX2 = pd.DataFrame()
dataX2['Word count'] = filtered_data['Word count']
dataX2['suma'] = col_sum
```

Figure 5. Creating a new dataframe with x_1 and x_2 as our input columns.

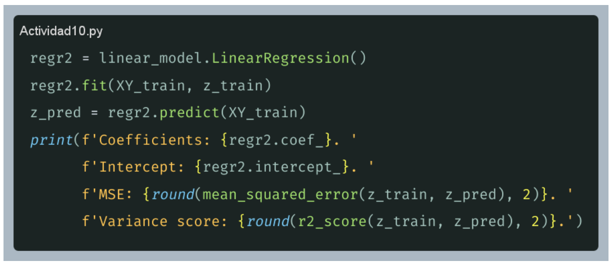
We then create our training set as a NumPy array for x, y, and z, which are our 3-tuples (x_1, y_1, z_1)



Actividad10.py
```python
XY_train = np.array(dataX2)
z_train = filtered_data['# Shares'].values
```

Figure 6. Training set for the multiple linear regression model.

Finally, we create our model and fit it to the training data.



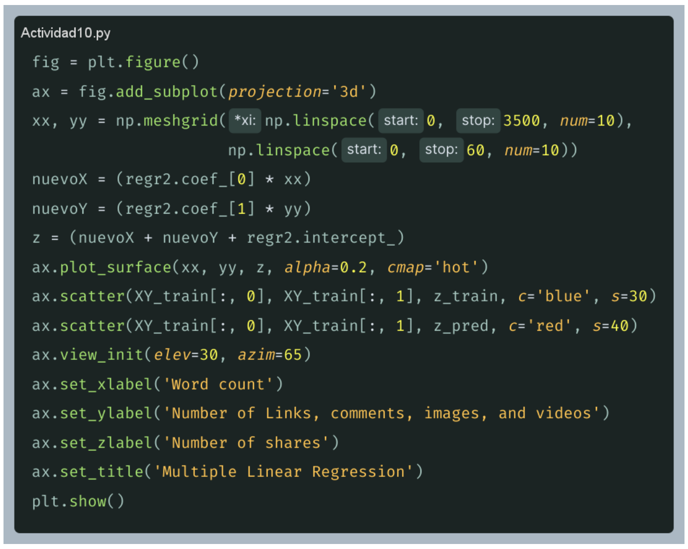
Actividad10.py
```python
regr2 = linear_model.LinearRegression()
regr2.fit(XY_train, z_train)
z_pred = regr2.predict(XY_train)
print(f'Coefficients: {regr2.coef_}. '
      f'Intercept: {regr2.intercept_}. '
      f'MSE: {round(mean_squared_error(z_train, z_pred), 2)}. '
      f'Variance score: {round(r2_score(z_train, z_pred), 2)}.')
```

Figure 7. Multiple linear regression model.

The reader may notice that we also print the coefficients $\{\beta_0, \beta_1, \beta_2\}$ of the linear function used for the model and some metrics to check the effectiveness of the model. These metrics are the *Mean Squared Error*, and the *Variance score*, where in ideal cases the MSE should be low and the variance score near one, which is not the case.

2.5. Making a 3D graph with Matplotlib.pyplot

Finally, we create a 3D plot using our new dataframe. We will start by creating a meshgrid where all of our x and y data will be distributed, and then make all of our plot values of a certain color (blue if from original set, red if it comes from a prediction using the training set).


Actividad10.py

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
xx, yy = np.meshgrid( *xi: np.linspace( start: 0,  stop: 3500, num=10),
                     np.linspace( start: 0,  stop: 60, num=10))
nuevoX = (regr2.coef_[0] * xx)
nuevoY = (regr2.coef_[1] * yy)
z = (nuevoX + nuevoY + regr2.intercept_)
ax.plot_surface(xx, yy, z, alpha=0.2, cmap='hot')
ax.scatter(XY_train[:, 0], XY_train[:, 1], z_train, c='blue', s=30)
ax.scatter(XY_train[:, 0], XY_train[:, 1], z_pred, c='red', s=40)
ax.view_init(elev=30, azim=65)
ax.set_xlabel('Word count')
ax.set_ylabel('Number of Links, comments, images, and videos')
ax.set_zlabel('Number of shares')
ax.set_title('Multiple Linear Regression')
plt.show()
```

Figure 8. Creating the 3D plot.

The reader may wonder why the plot contains a big 2D plane, and it is as simple as that a line in 2D is a plane in 3D. With that stated, it is quite easy to visualize that said plane is our model, represented by the linear function

$$z = 21140.97 + 0.493x + 289.3253y$$

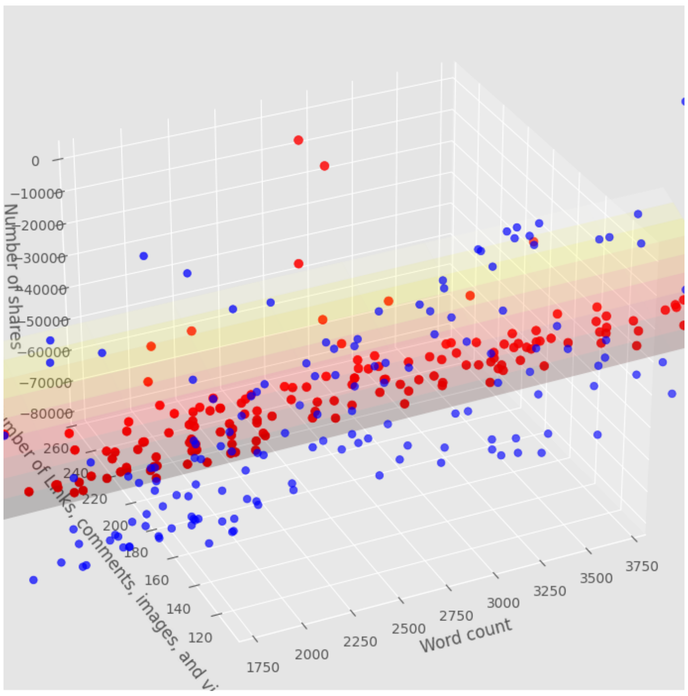


Figure 9. Final 3D plot.

3. Results

Even though our model is not trustworthy, we can still make some predictions with it. For example, let's suppose we have an article with 2000 words, 10 links, 4 comments, and 6 images, according to our model, we would have around 27913 shares:

$$z = 21140.97 + 0.493(2000) + 289.3253(10 + 4 + 6) \approx 27913$$

4. Conclusions

Guided by the used metrics, it turned out our model was not trustworthy given the dataset. Although it seems like a failure, the model helped to find a tendency: the more words, links, comments, and images an article has, the more shares it will have.

Of course it is not as simple as that, but given these inputs, the model suggests that this is true. We could prove it by reducing dimensions using an algorithm like Principal Component Analysis, but that is out of the purpose of this article.

Therefore, we conclude that the model is not suitable for small and dispersed datasets since it will probably lead to an underfitted model that will make bad predictions, but it remains useful to find a tendency early on during a larger investigation.

References

[1] Bagnato J.I.: *Aprende Machine Learning en Español.* Leanpub, 1.5st ed., 2020.

[2] Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd ed., 2010.

Affiliations

Aldo Hernández
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
aldo.hernandezt@uanl.edu.mx

Received: ???
Revised: ???
Accepted: ???