Aldo Hernández

LINEAR REGRESSION WITH PYTHON

Abstract *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 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 label that may indicate a relationship between it and our outcome of interest. This suggests that for a small dataset where underfitting is very common, univariate linear regression should be used only for finding tendencies, instead of making our predictions on this type of model.*

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

1. Introduction

Linear regression is a fairly common algorithm in statistics and machine learning to make predictions and show tendencies given a certain dataset. This algorithm *finds* a straight line—also known as "fitting" a line—that indicates a tendency and makes predictions from it [1]. This model is widely used as an activation function on neural networks for some specific tasks but it is required to be combined with other non-linear functions in order to recognize more complex patterns.

The straight line is represented as

$$h_w(x) = w_1 x + w_0$$

Where $\begin{bmatrix} w_0 & w_1 \end{bmatrix}$ is known as the weight vector and are the coefficients to be learned—or "found"—. The task of finding the h_w that fits best this data is called **linear regression**. To do this, we simply have to find the values of the weight vector that minimize the empirical loss. It is traditional to use the squared loss function L_2 summed all over the training set [2]

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

We would like to find $\mathbf{w^*} = \text{argmin}_w \, Loss(h_w)$. To minimize this function we have to partially differentiate with respect to each weight and set those equations to zero, since they will be at a minimum when their partial derivative is zero:

$$\begin{cases} \frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 \\ \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 \end{cases}$$

This equation system has a unique solution for the weight vector given by:

$$w_1 = \frac{N(\Sigma x_j y_j) - (\Sigma x_j)(\Sigma y_j)}{N(\Sigma x_j^2) - (\Sigma x_j)^2} \; ; \; w_0 = \frac{\Sigma y_j - w_1 \Sigma x_j}{N}$$

Since in this way of learning we want to minimize a loss, it is useful to have a mental picture of what is going on in the **weight space**—the space defined by all possible settings of the weights. For univariate linear regression, this space is two-dimensional, hence we can graph the loss as a function of w_0 and w_1 in a 3D plot as seen in figure 1.

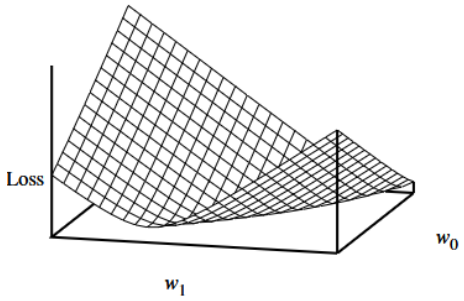

Figure 1. Example loss function taken from [2]

As we can see, this function is **convex**, and this is true for *every* linear regression problem with an L_2 loss function, it also implies that there are no *local minima*. [2]

When applying this concept in machine learning, it is important to note that we need a **learning rate** α—also known as step size—that can be a fixed constant or decay over time, in purpose of minimizing the *Loss* function.

For N training samples, we want to minimize the sum of individual losses for each pair in the set, so we have

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j)); \;\; w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) \times x_j$$

These updates constitute the **batch gradient descent** learning rule for univariate linear regression [2] that is a machine learning algorithm that uses the whole training dataset to optimize a training model. Convergence to a unique global minimum is guaranteed—as long as α is small enough—but may be vary slow since we have to cycle through all the training data for every step, and there may be a lot of steps [2]. This is essentially what linear regression models use to fit best the training dataset.

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:

- Pandas
- Matplotlib
- Scikit-learn

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

Actividad 9.py

```
import math
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 2. Essential imports and styling.

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:

Actividad 9.py

```
data = pd.read_csv('./articulos_ml.csv')
print(data.shape)
print(data.head())
print(data.describe())
data.drop( labels: ['Title', 'url', 'Elapsed days'], axis=1).hist()
plt.show()
```

Figure 3. Importing data and displaying useful information.

Using this information, we can see that the dataset contains 161 rows of data with 8 columns. Also, we can focus our interest in the columns "Word count" and "# Shares", they will be our input and output for this model since we will try to predict the number of shares based on the word count of an article, as shown in table 1.

Table 1
Column information

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

We can also create two histograms to look where is our dataframe more concentrated, as following

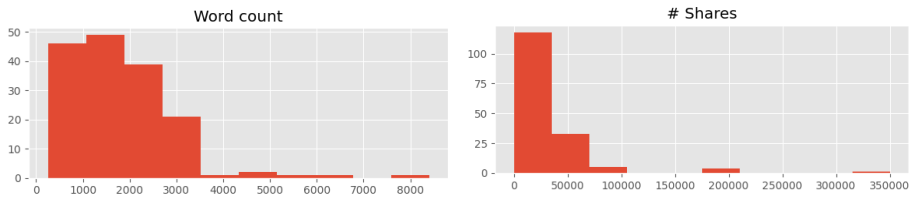

Figure 4. Data concentration in both columns.

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.

Thanks to the histograms shown in Figure 4, 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:

Actividad 9.py

```
filtered_data = data[(data['Word count'] ⩽ 3500) & (data['# Shares'] ⩽ 80000)]
f1 = filtered_data['Word count'].values
f2 = filtered_data['# Shares'].values
colored_set = []
for index, row in filtered_data.iterrows():
    if row['Word count'] > 1808:
        colored_set.append('red')
    else:
        colored_set.append('blue')
```

Figure 5. Filtering and coloring data.

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

2.4. Creating the model

After filtering our dataset, all we have left to do is create our linear regression model. For this, we will fit our model—or line—to the dataset and make predictions with the original data. Next, we will output some useful messages:

```
Actividad 9.py
X_train = filtered_data[['Word count']].values
y_train = filtered_data['# Shares'].values
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
char = '+' if model.intercept_ ⩾ 0 else '-'
print(f'Function: y = {round(model.coef_[0], 2)}x {char} {round(model.intercept_, 2)}',
      f'Mean Squared Error: {round(mean_squared_error(y_train, y_pred), 2)}',
      f'Variance score: {round(r2_score(y_train, y_pred), 2)}',
      sep='\n')
```

Figure 6. Training the linear regression model.

The piece of code shown in Figure 6 outputs the model's linear function $y \approx 5.7x + 11200.3$ with a very large mean squared error and small variance score, which are pretty bad results for said metrics.

2.5. Making a graph

Finally, we can plot our dataset in a 2D graph to visualize the straight line that represents our model:

```
Actividad 9.py
plt.scatter(f1, f2, c=colored_set, s=30)
plt.axline( xy1: (0, model.intercept_), slope=model.coef_[0])
plt.show()
```

Figure 7. Plotting a graph

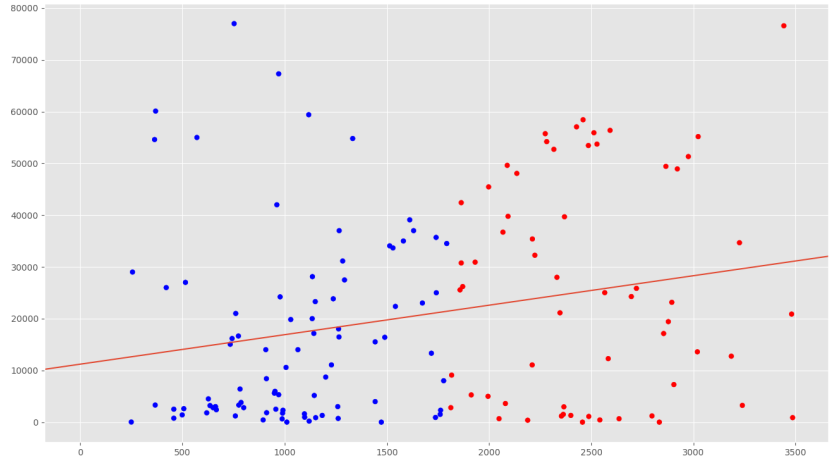

Figure 8. Word count vs Number of shares

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, according to our model, we would have around 22595 shares:

$$y \approx 5.7(2000) + 11200.3 \approx 22595$$

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 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: ???