Aldo Hernández

LOGISTIC REGRESSION WITH PYTHON

Abstract *Logistic regression is a supervised algorithm that classifies data into given classes based on their information, it uses the L_2 function for minimizing loss and converges to a minimum-error solution even if data is not linearly separabale. This document is an introduction to said process while analyzing a real dataset using Python. To do this, we explore the mathematical definitions around this method, use Python code to analyze data and create a logistic regression model to make predictions based on this data. Our conclusions indicate that this type of model is suitable for classification of any dataset using a cross validation method, since it removes the overfitting problem that is naturally found in this type of classification.*

Keywords logistic regression, python, model, machine learning

1. Introduction

Linear functions can be used not only for regression, but also for **classification**. In this case, the line—or surface—is called a *decision boundary*—also known as linear separator—and the data that admits this separator are called *linearly separable*. [2] In order to use these functions for classification, the first approach was to use the *threshold* function as the classification hypothesis

$$h_w(x) = Threshold(w \cdot x) \text{ where } Threshold(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise}$$

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times x_i \tag{1}$$

But this method has some pretty big disadvantages: even though it is *guaranteed* to converge with linearly separable data, it may take too many steps to do so, also, data may not be linearly separable always, so the algorithm (weight update rule shown in Equation 1) will fail to converge for a fixed learning rate α, unless it decays as $O(1/t)$ where t is the iteration number, then it is shown to converge to a minimum-error solution. [2]

Also, the hypothesis is not differentiable and is a discontinuous function of its inputs and weights, which makes learning with the **perceptron rule** shown in Equation 1 pretty complicated. Besides, this function will always return a confident prediction of 1 or 0, even if examples are too close to the boundary; in a lot of situations, this is not optimal since we will need more gradated predictions. [2]

It's because of these issues that *logistic regression* surges as a better alternative. We define a function—known as logistic or sigmoid—as follows

$$h_w(x) = Logistic(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$$

This hypothesis gives a probability of 0.5 for every input at the center, and approaches 0 or 1 as we move away from the boundary [2]. The process of fitting the weights to this model to minimize the loss on a dataset is called **logistic regression** [2], there is no easy closed-form solution to find the optimal value of w but gradient descent computation is straightforward. Partially differentiating the L_2 function we get the following

$$\frac{\partial}{\partial w_i}(y - h_w(x))^2 = -2(y - h_w(x)) \times h_w(x)(1 - h_w(x)) \times x_i$$

Thus, the weight update for minimizing the loss is

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times h_w(x)(1 - h_w(x)) \times x_i$$

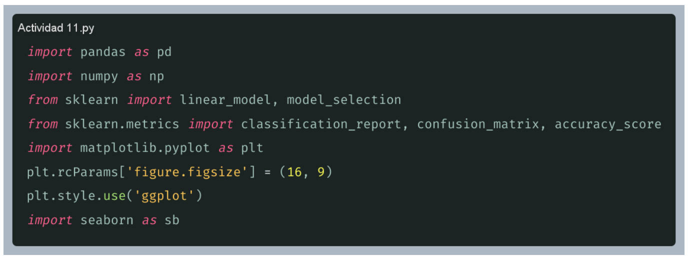
This process is a **supervised algorithm** that can classify data into two states—binary, as shown earlier—or multiple tags [1]. Some common use cases are:

- Classify mail into spam or not.
- Classify tumors into benign or malignant.
- Classify the content of an article.

2. Methodology

2.1. Before typing code

First of all, we need to download this .csv file [1] from where we will obtain our dataset, this file has information about users that visit a certain website, such as: visit duration, viewed pages, total actions made, action value sum, and operating system—0 for Windows, 1 for MacOS, and 2 for Linux—. Next, we will import some necessary libraries to our script



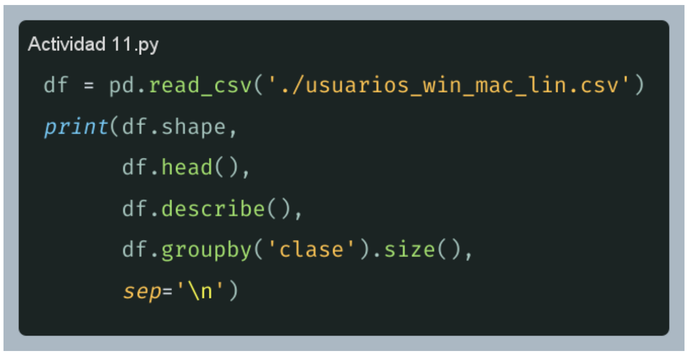
Actividad 11.py

```
import pandas as pd
import numpy as np
from sklearn import linear_model, model_selection
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
import seaborn as sb
```

Figure 1. Necessary library imports.

2.2. Data analysis

Next, let's analyze our data, we will parse it from the .csv file we downloaded in Subsection 2.1, and then print out some useful information



Actividad 11.py

```
df = pd.read_csv('./usuarios_win_mac_lin.csv')
print(df.shape,
      df.head(),
      df.describe(),
      df.groupby('clase').size(),
      sep='\n')
```

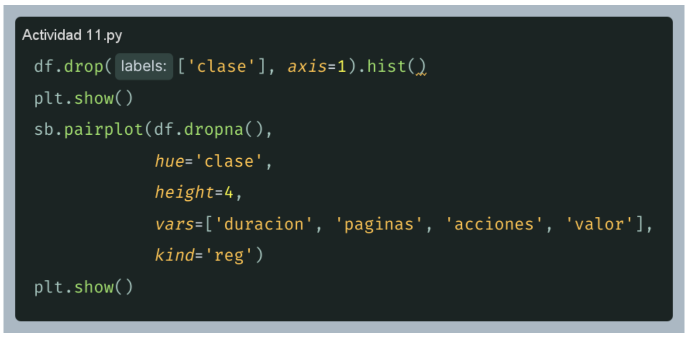
Figure 2. Displaying information about the dataset.

Thanks to these lines of code, we know that there are 170 rows of data along 5 columns, where 86 rows are Windows users, 40 are MacOS users, and 44 are Linux users. Besides, we may find the following information useful for further analysis

Table 1
Column information

	Duration	Pages	Actions	Value
Count	170	170	170	170
Mean	111.08	2.04	8.72	35.68
Std	202.45	1.50	9.14	44.75

After this quick review, we will create a lot of graphs in order to understand the information


Actividad 11.py

```
df.drop( labels: ['clase'], axis=1).hist()
plt.show()
sb.pairplot(df.dropna(),
            hue='clase',
            height=4,
            vars=['duracion', 'paginas', 'acciones', 'valor'],
            kind='reg')
plt.show()
```

Figure 3. Creating histograms and regression plots.

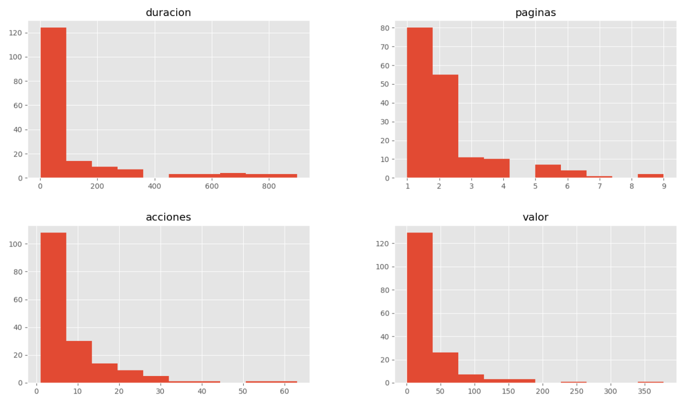


Figure 4. Number of rows per column value histograms.

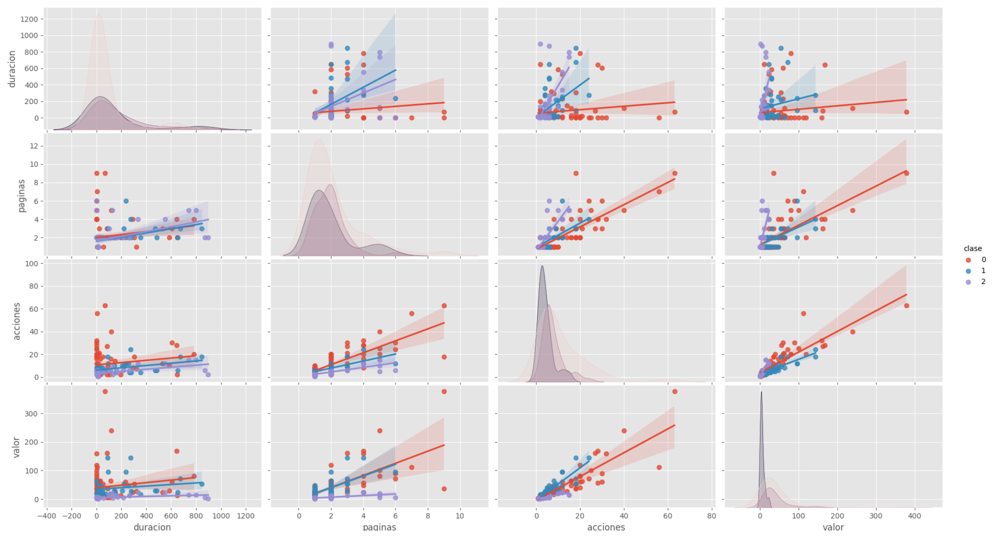
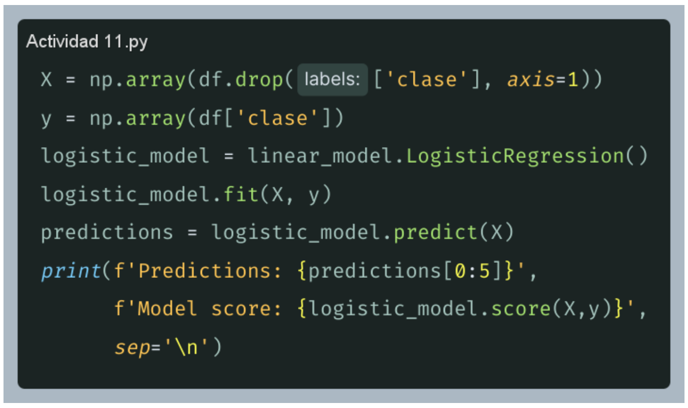


Figure 5. Data regression plots.

From Figure 4 we can see that duration is mostly concentrated around 0 ms to 100 ms, pages between 1 and 2, actions around 0 to 10, and sum value between 0 to 50. We can also see in Figure 5 the lineal relationships between inputs, i. e. MacOS and Linux users do more actions when they spend more time in the website, which may hint that the website has relation with UNIX-like operating systems.

2.3. Creating the model

Next, we will create our logistic regression model and print some predictions using training data along with scoring


Actividad 11.py

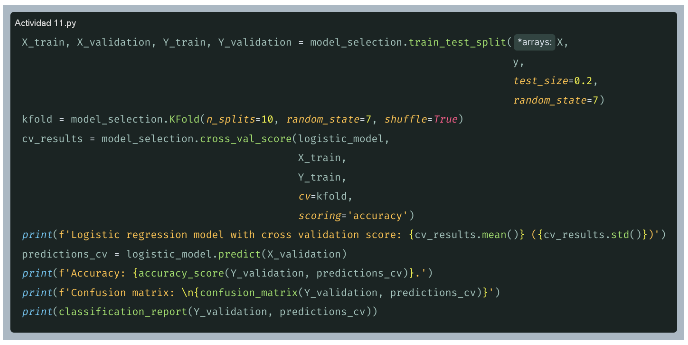
```
X = np.array(df.drop( labels: ['clase'], axis=1))
y = np.array(df['clase'])
logistic_model = linear_model.LogisticRegression()
logistic_model.fit(X, y)
predictions = logistic_model.predict(X)
print(f'Predictions: {predictions[0:5]}',
      f'Model score: {logistic_model.score(X,y)}',
      sep='\n')
```

Figure 6. Model creation.

Turns out that the mean accuracy between predictions and real values is around 71%, which is a pretty good score.

2.4. Model validation

A good practice in machine learning is to divide the dataset into two sets: a training set and a validation set, since this will avoid overfitting problems. [1] In order to do this, we will use 80% of the dataset for training and 20% for validation, said rows will be randomly selected.



```
Actividad 11.py
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split( *arrays: X,
                                                                                y,
                                                                                test_size=0.2,
                                                                                random_state=7)
kfold = model_selection.KFold(n_splits=10, random_state=7, shuffle=True)
cv_results = model_selection.cross_val_score(logistic_model,
                                             X_train,
                                             Y_train,
                                             cv=kfold,
                                             scoring='accuracy')
print(f'Logistic regression model with cross validation score: {cv_results.mean()} ({cv_results.std()})')
predictions_cv = logistic_model.predict(X_validation)
print(f'Accuracy: {accuracy_score(Y_validation, predictions_cv)}.')
print(f'Confusion matrix: \n{confusion_matrix(Y_validation, predictions_cv)}')
print(classification_report(Y_validation, predictions_cv))
```

Figure 7. Re-compiling the model using only the training set (80%).

Now that we recompiled the model using cross-validation, we make new predictions to test the accuracy, resulting in an 85.29% accuracy. We also print out the confusion matrix

$$\begin{pmatrix} 16 & 0 & 2 \\ 3 & 3 & 0 \\ 0 & 0 & 10 \end{pmatrix}$$

This matrix tells us that in our predictions, there were 3 real MacOS users predicted as Windows users, and 2 real Windows users predicted as Linux users. We can also see the classification report with our validation set in Table 2, that tells us that i. e. MacOS classes had 3 bad predictions and 3 good predictions our of 6 cases (support), the model obtained a 84% in f1-score, which is a very good value for said metric.

Table 2
Classification report

	Precision	Recall	f1-score	Support
0	0.84	0.89	0.86	18
1	1.00	0.50	0.67	6
2	0.83	1.00	0.91	10
Weighted avg	0.87	0.85	0.84	34

3. Results

Finally, we can make some predictions with this model. For example, let's say a new user visited the website with the following data:

- Had a duration of 10 seconds.
- Visited 3 pages.
- Had 5 actions while navigating.
- Had a value of 9.

Then, the model classifies it as a Linux user.

4. Conclusions

After using cross validation to improve our logistic regression model, we obtained a high score even when our sample was small. We conclude that logistic regression models classification is very good, even if the dataset is not so big. Cross-validation improved the model because it removed the overfitting problem we had when first using the whole dataset as training set.

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: ???