University of Carthage
National School of Sciences and Advanced Technologies
Borj Cedria

Machine learning project:

Linear regression-
A mathematical overview

Carried out by:

- Boughdiri Hafedh.
- Ben Selem Behir.
- Sghaier Radhi.

2 EAN.

2020/2021

I. Introduction

A. Machine learning:

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on building applications that learn from data in order to improve their accuracy over time.[1] In the case of supervised learning, ML is applied when these three conditions are met:

- A pattern exists.
- We can not pin the pattern down mathematically.
- We have data.

It is to mention that the first two conditions are not mandatory when applying ML, since these imply that applying machine learning will not be the most efficient option, whereas if the third option is not met, we can not do learning whatsoever.
In fact, ML aims to approximate an unknown target function $f(x) = y$ by picking a hypothesis $h(x)$ from a hypothesis set H. In order to ensure that $h(x)$would actually approximate $f(x)$, we need an error measure function $E(f, h)$ that would quantify this approximation.
By applying $E(f, h)$we get:

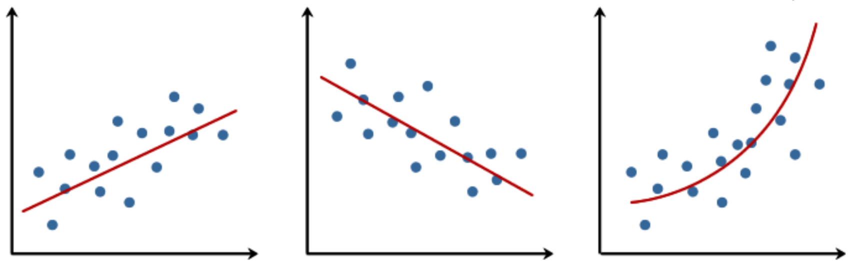
- $E_{in}(f, h)$: The in-sample error, i.e. how much does $h(x)$ approximate $f(x)$ in the training data.
- $E_{out}(f, h)$: The out-of-sample error, i.e. how much does $h(x)$ approximate $f(x)$ in the testing data.

To ensure the generalization of the model, i.e. the feasibility of learning, **Wassily Hoeffding** proposed in 1963 the Hoeffding Inequality[3] which states that the probability that E_{in} deviates from E_{out} by more than a prescribed tolerance ε is less than or equal to a quantity that is proportional to ε^2 and the sample size N. From this, we conclude that learning is feasible, but only in a probabilistic sense.

$$Hoeffding's\ inequality:\ P[|E_{in} - E_{out}| > \varepsilon] \le 2e^{-2\varepsilon^2 N} \quad (1)$$

B. Linear regression:

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (output) and one or more explanatory variables (input variables). The case of one explanatory variable is called simple linear regression; for more than one variable, the process is called multiple linear regression.[2]



A- Positively correlated linear data. B- Negatively correlated linear data. C- Non-linear data.

Figure1: A. B. Linear vs C. non-linear data.

The general formula for linear regression is of the form:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \epsilon \qquad (2)$$

Where: **β_0** is known as the intercept.

β_1 to **β_n** are known as coefficients.

x_1 to **x_n** are the features of the data set.

ε is the residual term.

It is to note that linear regression only works on linear data, so in Figure1, linear regression would be the red line illustrated in Figure1. A and Figure1. B.

II. Mathematical overview of linear regression:

A. Simple linear regression:

In the case of simple linear regression, we opted for two fitment methods. We first went with the ordinary least squares method (OLS) in its simple format to find the best fitting slope (coefficient), then we implemented a gradient descent (GD) algorithm to find the best slope.

- OLS: We can estimate the slope (coefficient) by calculating the covariance between X (input) and Y (output) and dividing it by the variance of X.

$$coefficient = \frac{Cov(X, Y)}{Var(X)} = \frac{\sum_{i=0}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=0}^{n}(X_i - \overline{X})^2} \qquad (3)$$

Where: X_i, Y_i are the values of X, and Y at index i.

$\overline{X}$, $\overline{Y}$(X bar, Y bar) are the averages of X, and Y.

Then to estimate the intercept, we have:

$$Intercept = \overline{Y} - coefficient \times \overline{X} \qquad (4)$$

- GD: Is an optimization algorithm that's used to find the optimal set of parameters given a training dataset. GD minimizes the loss function in order to find the optimal value of slope "m" and constant (bias) "b".

  Gradient descent iteratively calculates the gradients of the loss function with respect to the parameters and keeps on updating the parameters until we reach the minima.

  1st step: Initialize the values for the parameters, m = b = 0.

  2nd step: Calculate the partial derivatives in respect to the parameters:

$$\frac{\partial}{\partial m} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b)) \qquad (5)$$

$$\frac{\partial}{\partial b} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b)) \qquad (6)$$

Where N is the number of points in the input.

3rd step: Updating the parameters' values:

$$m = m - Lr \times \frac{\partial L}{\partial m} \quad (7)$$

$$b = b - Lr \times \frac{\partial L}{\partial b} \quad (8)$$

Where: Lr is the learning rate of the algorithm.
L is the loss (error value).

B. Multiple linear regression:

Multiple linear regression is a model capable of capturing a linear relationship between multiple variables and features, in case there is actually a relationship to capture. Equation (2) showcased the general formula for linear regression, which could be represented in vector notation as:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} \mathbf{x}_0^\mathsf{T} \\ \mathbf{x}_1^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Figure2: A vector representation of the linear regression's parameters.

And now we get this final formula:

$$y = \beta X + \varepsilon \quad (9)$$

Linear least squares (LLS) is the algorithm that's mainly used for estimating the coefficients (β) in equation (9). We used its most popular variant, which is OLS (previously used in the simple linear regression section).
Without diving deep into the mathematics of OLS, we are able to obtain the optimal coefficients values by applying this formula:

$$\widehat{\beta} = (X^T X)^{-1} X^T y \quad (10)$$

Where: $\widehat{\beta}$ is a vector containing all the coefficients that we can use to make predictions.
$(X^T X)^{-1} X^T$ is called the pseudo-inverse of X.

C. Evaluation: R squared

In order to evaluate our model and get a way for us to quantify its accuracy, we used the r squared (R^2) metric. R^2 could be obtained using this formula:

$$R^2 = 1 - \frac{\sum_{i=0}^{n} (y_i - \widehat{y})^2}{\sum_{i=0}^{n} (y_i - \bar{y})^2}$$

Where: y_i is a value of Y at index i.

$\hat{y}$ (y hat) is the vector of predicted values of Y.

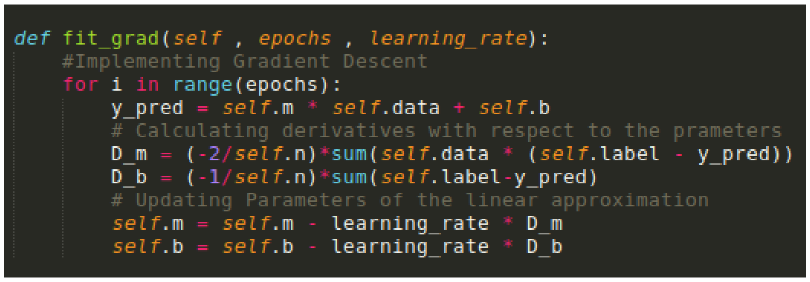
$\bar{y}$(y bar) is the average of Y.

D. Implementation:

We implemented two classes, one for the simple linear regression, which we called **TwoD_LinearRegression (TwoD)**, and one for the multiple linear regression, which we called **MultiD_LinearRegression (MultiD)**.

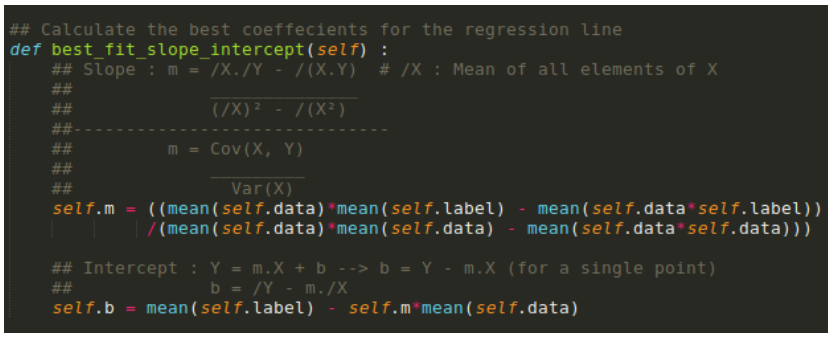
In the TwoD calss, we implemented two fitment methods, Gradient Descent (GD) and Ordinary Least Squares (OSL) and a predict method, whereas in the MultiD class, we only implemented one fitment method which is the OSL along with the predict method. Note that both classes share the same evaluation method, which is R^2.

Figures 3, 4, 5, and 6 showcases parts of the implementations of the used classes in our full code-base[4].



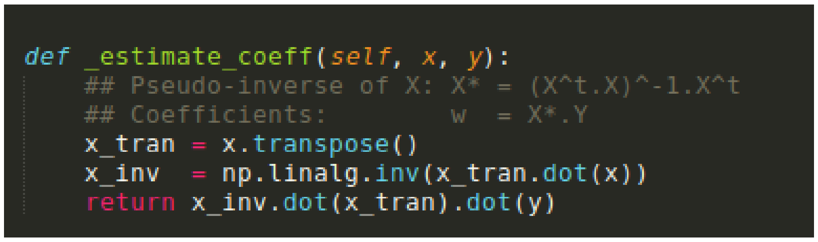
```python
def fit_grad(self , epochs , learning_rate):
    #Implementing Gradient Descent
    for i in range(epochs):
        y_pred = self.m * self.data + self.b
        # Calculating derivatives with respect to the prameters
        D_m = (-2/self.n)*sum(self.data * (self.label - y_pred))
        D_b = (-1/self.n)*sum(self.label-y_pred)
        # Updating Parameters of the linear approximation
        self.m = self.m - learning_rate * D_m
        self.b = self.b - learning_rate * D_b
```

Figure3: GD method in the TwoD class.



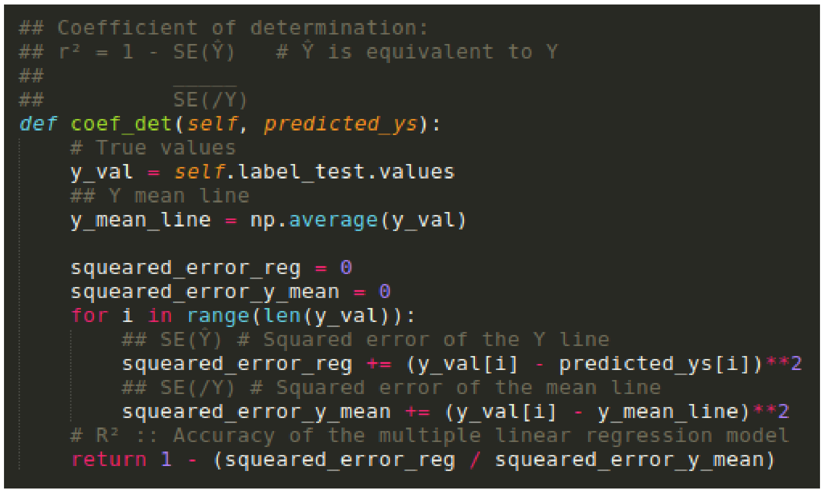
```python
## Calculate the best coeffecients for the regression line
def best_fit_slope_intercept(self) :
    ## Slope : m = /X./Y - /(X.Y)  # /X : Mean of all elements of X
    ##              ______________
    ##              (/X)² - /(X²)
    ##------------------------------
    ##          m = Cov(X, Y)
    ##              _________
    ##                Var(X)
    self.m = ((mean(self.data)*mean(self.label) - mean(self.data*self.label))
              /(mean(self.data)*mean(self.data) - mean(self.data*self.data)))

    ## Intercept : Y = m.X + b --> b = Y - m.X (for a single point)
    ##              b = /Y - m./X
    self.b = mean(self.label) - self.m*mean(self.data)
```

Figure4: OSL method in the TwoD class.



```python
def _estimate_coeff(self, x, y):
    ## Pseudo-inverse of X: X* = (X^t.X)^-1.X^t
    ## Coefficients:        w  = X*.Y
    x_tran = x.transpose()
    x_inv  = np.linalg.inv(x_tran.dot(x))
    return x_inv.dot(x_tran).dot(y)
```

Figure5: Coefficients determination of the OSL method in the MultiD class.



```
## Coefficient of determination:
## r² = 1 - SE(Ŷ)   # Ŷ is equivalent to Y
##
##          _____
##          SE(/Y)
def coef_det(self, predicted_ys):
    # True values
    y_val = self.label_test.values
    ## Y mean line
    y_mean_line = np.average(y_val)

    squared_error_reg = 0
    squared_error_y_mean = 0
    for i in range(len(y_val)):
        ## SE(Ŷ) # Squared error of the Y line
        squared_error_reg += (y_val[i] - predicted_ys[i])**2
        ## SE(/Y) # Squared error of the mean line
        squared_error_y_mean += (y_val[i] - y_mean_line)**2
    # R² :: Accuracy of the multiple linear regression model
    return 1 - (squared_error_reg / squared_error_y_mean)
```

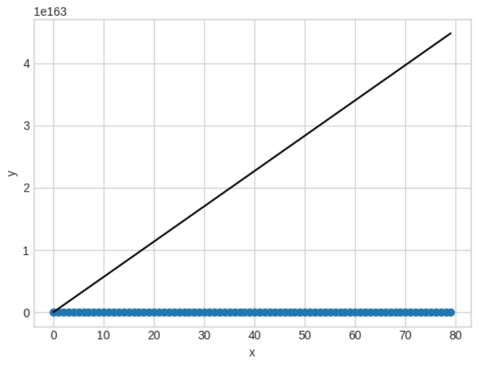
Figure6: R^2 function in the MultiD class.

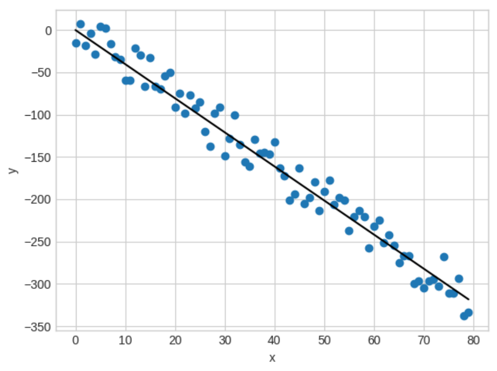
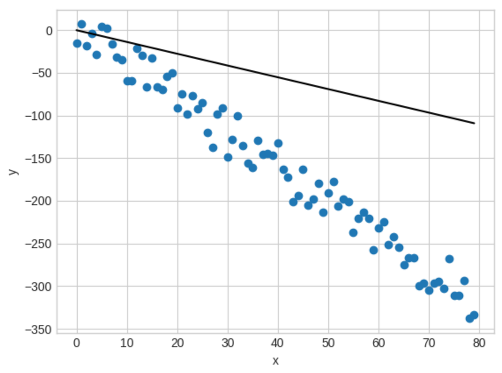
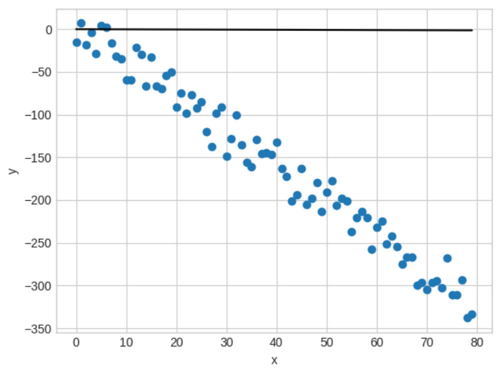
III. Results:

A. Gradient descent with different learning rates:

This is a comparison between different learning rates of the gradient descent optimizer with a fixed number of iterations (100) for simple linear regression on a negatively correlated, randomly generated linear data.

Table1: Outputs of different learning rates for the gradient descent with their corresponding R^2 scores.

LR (Learning rate)	R^2	Comment
0.001 	$-\infty$	Our data is negatively correlated, whereas the slope is still positive due to the big learning rate. → The learning rate should be smaller.

0.0001 	0.97109971529 70419	This is the optimal learning rate since it gives us the best slope, and it follows the data perfectly.
0.000001 	-0.58781823567 21813	This learning rate is too small that we didn't obtain the optimal slope. → The number of iterations must be significantly increased in order to get good fitness, but this will increase the runtime.
0.00000001 	-2.60732318223 0777	This learning rate is infinitely small that the slope didn't even slightly change. → The number of iterations must be significantly increased in order to get good fitness, but this will increase the runtime.

B. GD VS OLS:

This is a comparison between an iterative method, being gradient descent, and a direct method, being OLS, for simple linear regression on a negatively correlated, randomly generated linear data.

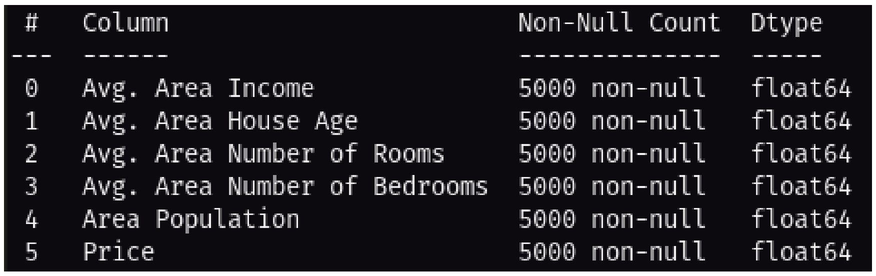
Table2: GD with its optimal parameters VS the OSL method based on the R² metric.

Method	R²	Comment
Gradient descent (100 iterations, 0.00001 LR)	0.9710997152970419	This is the best model from the previous comparison.
OSL	0.972101802237473	This is an instant method with no parameters to tweak.

Results presented in Table2 showcase that GD and OLS have almost the same performance.

C. Multiple linear regression: Our model VS Sklearn's

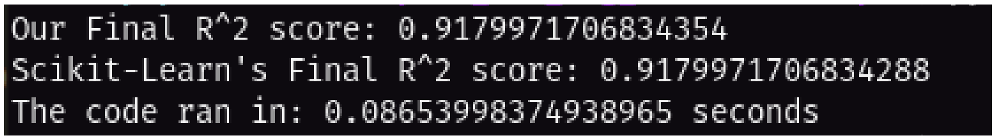
In this section, we use a USA housing prices dataset[4] with the goal being to predict a house's price based on five input features which are shown in Figure7.
The dataset has 5000 examples, which we split into training data, and testing data.



```
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
```

Figure7: General info about the dataset.

After running both our OSL model, and Sklearn's LinearRegression model on the same data, and obtaining its respective R² scores, we get almost the same results as shown in Figure8.



```
Our Final R^2 score: 0.9179971706834354
Scikit-Learn's Final R^2 score: 0.9179971706834288
The code ran in: 0.08653998374938965 seconds
```

Figure8: R² scores of our multiple linear regression model and Sklearn's.

IV. Conclusion:

Linear regression is a powerful model considering its implementation simplicity, which makes it an easily maintainable tool, as well as a great benchmark for fast testing.
This report discusses the basics of linear regression, but a lot is left to be discussed in this topic, to site a couple:

- Classification of linearly separable data with linear regression.
- Classification of non-linearly separable data with linear regression through non-linear transformations.

References:

[1] Machine learning, by IBM Cloud Education: https://www.ibm.com/cloud/learn/machine-learning

[2] Linear regression from scratch with python:
https://www.askpython.com/python/examples/linear-regression-from-scratch

[3] Hoeffding's inequality: https://en.wikipedia.org/wiki/Hoeffding%27s_inequality

[4] Source code & dataset: https://github.com/Rad-hi/Linear_Reg_From_Scratch

[5] Colaboratory code:
https://colab.research.google.com/drive/10CZT-rRKvJPTtAxUVGByr8ueTFJy4BE7?usp=sharing