Multiple Linear Regression

Multiple Linear Regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to observed data.

Clearly, it is nothing but an extension of *Simple Linear Regression*.

Consider a dataset with p *features* (or *independent variables:* X) and one *response* (or *dependent variable:* y). Also, the dataset contains n rows/observations.

We define:

X (**feature matrix**) = a matrix of shape (n,p) where x_{ij} denotes the values of j th feature for i th observation.

So,

$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \vdots & x_{np} \end{pmatrix}$$

and

y (**response vector**) = a vector of size n where y_i denotes the value of response for i th observation.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

The **regression line** for **p** features is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

where $h(x_i)$ is **predicted response value** for i th observation and $b_{0,} b_{1,} \ldots, b_p$ are the **regression coefficients**.

Also, we can write:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i$$
$$or$$
$$y_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

where e_i represents **residual error** in i th observation.

We can generalize our linear model a little bit more by representing feature matrix X as:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

So now, the linear model can be expressed in terms of matrices as:

$$y = X\beta + \varepsilon$$

where,

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_p \end{bmatrix}$$

and

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

Now, we determine **estimate of b**, i.e. *b'* using **Least Squares** method.

As already explained, Least Squares method tends to determine *b'* for which total residual error is minimized.

We present the result directly here:

$$\hat{\beta} = (X'X)^{-1}X'y$$

where ' represents the transpose of the matrix while -1 represents the matrix inverse.

Knowing the least square estimates, *b'* , the multiple linear regression model can now be estimated as:

$$\hat{y} = X\hat{\beta}$$

where *y'* is **estimated response vector**.

Note: The complete derivation for obtaining least square estimates in multiple linear regression can be found here.

Given below is the implementation of multiple linear regression technique on the Boston house pricing dataset using Scikit-learn.

در مثال زیر، روش‌های مزکور را روی یکی از دیتاست های آماده ماژول *sklearn* اجرا می کنیم.

```
#https://www.geeksforgeeks.org/linear-regression-python-implementation/

import numpy as np
from sklearn import datasets, linear_model, metrics

# load the boston dataset
boston = datasets.load_boston(return_X_y=False)

# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.4, random_state=1)

# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: \n', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

خروجی آن به قرار زیر است:

```
Coefficients:
[ -8.80740828e-02   6.72507352e-02   5.10280463e-02   2.18879172e+00
-1.72283734e+01   3.62985243e+00   2.13933641e-03  -1.36531300e+00
2.88788067e-01  -1.22618657e-02  -8.36014969e-01   9.53058061e-03
-5.05036163e-01]
Variance score: 0.720898784611
```

ملاحظه می شود که 13 تا $\hat{\beta}$ پیدا کرد. چون دیتاست 13 تا ستون داشت. یعنی برای هر ستون *Train* معادله خطی به صورت

$$\hat{y} = X\hat{\beta}$$

پیدا کرد. آن را روی *Test* اجرا می کنیم. و مقادیر حاصل را با *y* واقعی مقایسه می کند. دقت مدل (*Score*) را محاسبه می کند. در اینجا 72 درصد شده. یعنی این مدل دقت مطلوبی دارد و قابل اطمینان است.

فرمول محاسبه *Score* به صورت زیر است:

$$1 - \frac{Var(y - y')}{Var(y)}$$

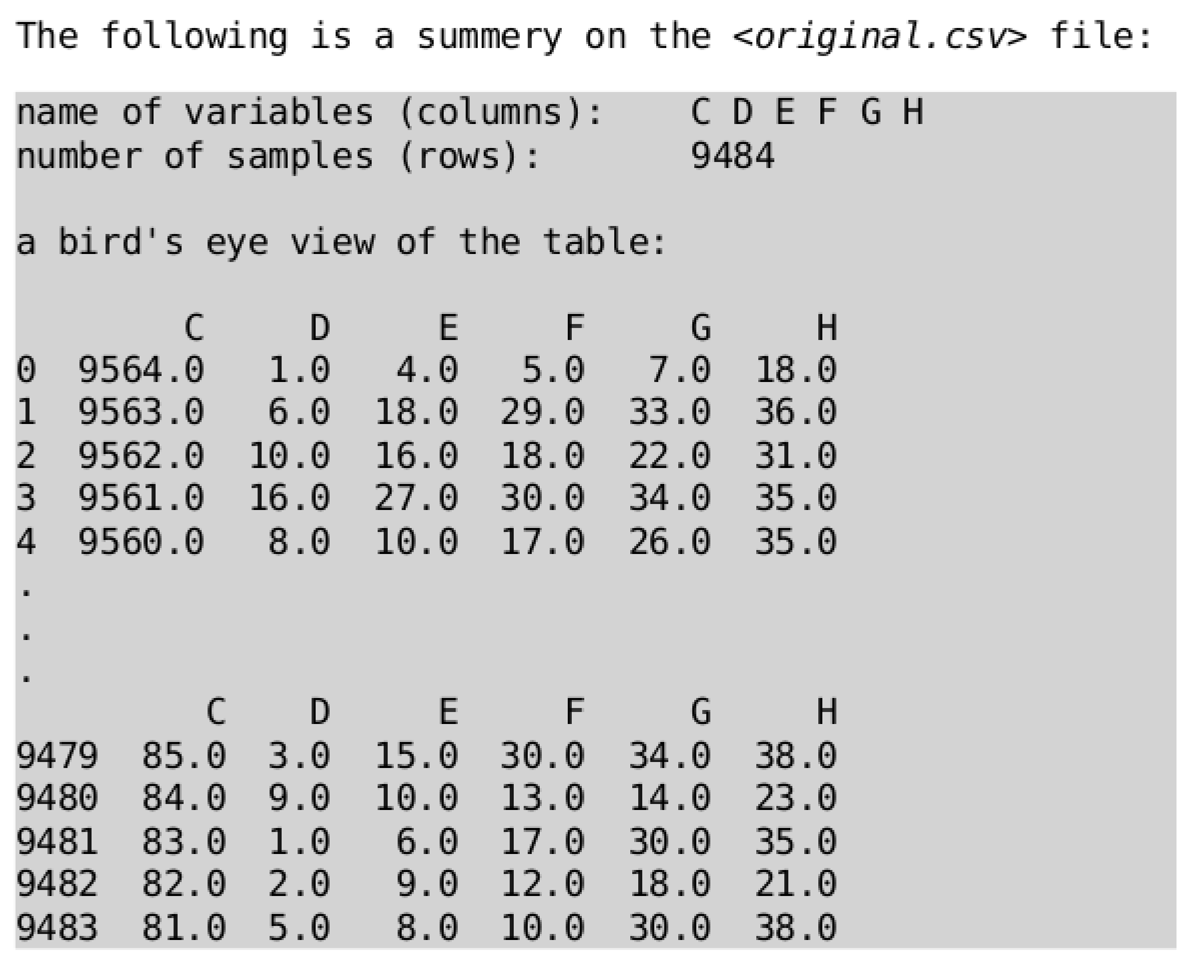
In above example, we determine accuracy score using **Explained Variance Score:**

1 – Var{y – y'}/Var{y}

where *y'* is the estimated target output, *y* the corresponding (correct) target output, and *Var* is Variance, the square of the standard deviation.

The best possible score is 1.0 , lower values are worse.

Providing the *original.csv* data-set, with 5 columns and 9484 rows:



```
The following is a summery on the <original.csv> file:

name of variables (columns):    C D E F G H
number of samples (rows):       9484

a bird's eye view of the table:

        C     D     E     F     G     H
0  9564.0   1.0   4.0   5.0   7.0  18.0
1  9563.0   6.0  18.0  29.0  33.0  36.0
2  9562.0  10.0  16.0  18.0  22.0  31.0
3  9561.0  16.0  27.0  30.0  34.0  35.0
4  9560.0   8.0  10.0  17.0  26.0  35.0
.
.
.
        C    D     E     F     G     H
9479  85.0  3.0  15.0  30.0  34.0  38.0
9480  84.0  9.0  10.0  13.0  14.0  23.0
9481  83.0  1.0   6.0  17.0  30.0  35.0
9482  82.0  2.0   9.0  12.0  18.0  21.0
9483  81.0  5.0   8.0  10.0  30.0  38.0
```

We ready to apply the M**ultiple Linear Regression** on them variables:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn import datasets, linear_model, metrics

// omitted ...

# 'H'

X = v[['C', 'D', 'E', 'F', 'G']]

X = np.array(X)

y = v['H']

y = np.array(y)
```

The Scores for each of them variables are:

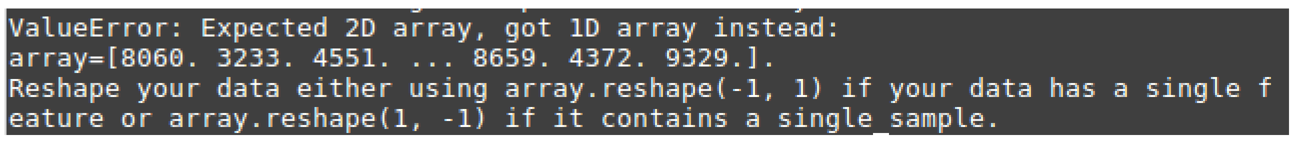
'D':	Variance score: 0.425745578495
'E':	Variance score: 0.64434924678
'F':	Variance score: 0.671096989866
'G':	Variance score: 0.621778826067
'H':	Variance score: 0.413719006021

let's consider one example enlisting for 40 instances of the D column:

```
score: 0.67 %

y^    int(y^)      y  HIT

-----------------------------------

24.16  (23, 24, 25) 26 no

14.45  (13, 14, 15) 13 yes

15.65  (14, 15, 16) 14 yes

28.75  (27, 28, 29) 28 yes

21.87  (20, 21, 22) 11 no

12.82  (11, 12, 13) 10 no

13.16  (12, 13, 14) 12 yes

18.17  (17, 18, 19) 18 yes

…

22.49  (21, 22, 23) 24 no

12.18  (11, 12, 13) 13 yes
```

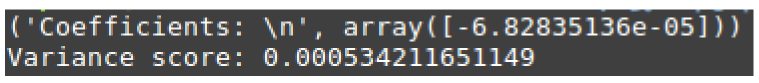
Now, this is time to predict the unseen $\hat{y}$ values for the C column, using the achieved model. This is NOT feasible for X is a 1-dim array.

However, we could separately apply the *multi linear regression* on each column. The $\hat{y}$ For C column:



```
ValueError: Expected 2D array, got 1D array instead:
array=[8060. 3233. 4551. ... 8659. 4372. 9329.].
Reshape your data either using array.reshape(-1, 1) if your data has a single f
eature or array.reshape(1, -1) if it contains a single sample.
```

The result is not promising as expected:



```
('Coefficients: \n', array([-6.82835136e-05]))
Variance score: 0.000534211651149
```

The score is very close to 0!