Adecuación del metodo manual de regresión Logística

1. Implementar (adecuar) el mètodo "manual" descrito en: https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#id13. Con los datos en el csv de clasificación en teams en la carpeta semana 6 . Puede descargar el código también en el github del autor, pero por favor leer primero en su totalidad el contenido en el link de arriba. el github del autor es este: https://github.com/bfortuner/ml-glossary

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
```

```
data= pd.read_csv('data_classification.csv', sep = ';')
data.head()
X = data[['suenio','estudio']].values
y = data['pasan'].values
```

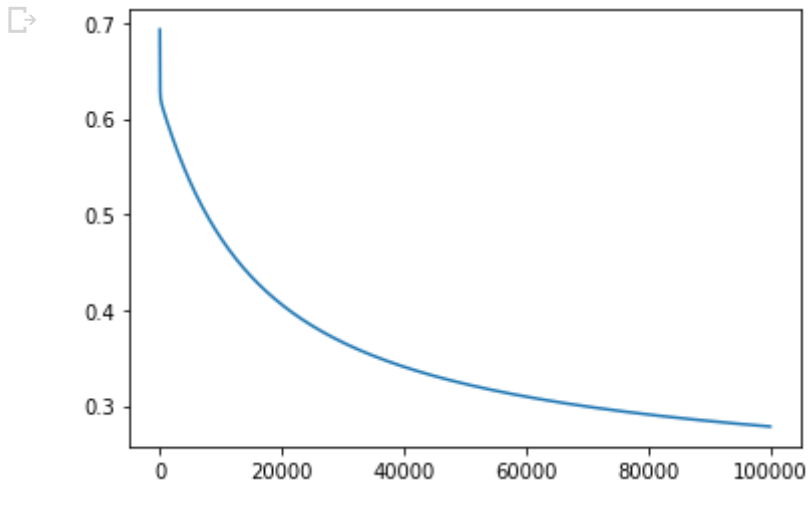
```
X = X.T
y = y.reshape(1, X.shape[1])

print("Shape of X : ", X.shape)
print("Shape of Y : ", y.shape)
```

```
     Shape of X :  (2, 100)
     Shape of Y :  (1, 100)
```

```
def sigmoid(x):
    # Activation function used to map any real value between 0 and 1
    return 1 / (1 + np.exp(-x))


def model(X, Y, learning_rate, iterations):

    m = X.shape[1]
    n = X.shape[0]

    W = np.zeros((n,1))
    B = 0

    cost_list = []

    for i in range(iterations):

        Z = np.dot(W.T, X) + B
        A = sigmoid(Z)

        # cost function
        cost = -(1/m)*np.sum( Y*np.log(A) + (1-Y)*np.log(1-A))

        # Gradient Descent
        dW = (1/m)*np.dot(A-Y, X.T)
        dB = (1/m)*np.sum(A - Y)

        W = W - learning_rate*dW.T
        B = B - learning_rate*dB

        # Keeping track of our cost function value
        cost_list.append(cost)

        if(i%(iterations/10) == 0):
            print("cost after ", i, "iteration is : ", cost)

    return W, B, cost_list


iterations = 100000
learning_rate = 0.0015
W, B, cost_list = model(X, y, learning_rate, iterations)
```

```
cost after  0 iteration is :  0.6931471805599453
cost after  10000 iteration is :  0.4753572235137952
cost after  20000 iteration is :  0.4057350332678727
cost after  30000 iteration is :  0.36626140508651517
cost after  40000 iteration is :  0.3408319955194374
cost after  50000 iteration is :  0.32297772027584126
cost after  60000 iteration is :  0.3096737022266117
cost after  70000 iteration is :  0.2993262973910783
cost after  80000 iteration is :  0.2910162986028703
cost after  90000 iteration is :  0.2841752434417335
```

```
plt.plot(np.arange(iterations), cost_list)
plt.show()
```




```
def accuracy(X, Y, W, B):

    Z = np.dot(W.T, X) + B
    A = sigmoid(Z)

    A = A > 0.5

    A = np.array(A, dtype = 'int64')

    acc = (1 - np.sum(np.absolute(A - Y))/Y.shape[1])*100

    print("Accuracy of the model is : ", round(acc, 2), "%")
```

```
accuracy(X, y, W, B)
```

```
 Accuracy of the model is :  87.0 %
```

✓ 0 s completado a las 14:07

Implementación Regresión Logística con librerias sklearn

2. Correr el ejemplo con los datos del csv en la carpeta semana 6 con la función de sklearn: •https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html y comparar.

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sb
```

```
data= pd.read_csv('data_classification.csv', sep = ';')
data.head(10)
X = data[['suenio','estudio']].values
y = data['pasan'].values
```

```
clf=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=200,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=0, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
clf.fit(X, y)
```

```
     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=200,
                        multi_class='auto', n_jobs=None, penalty='none',
                        random_state=0, solver='newton-cg', tol=0.0001, verbose=0,
                        warm_start=False)
```

```
clf.predict(X[:3, :])
```

```
     array([1, 0, 0])
```

```
clf.predict_proba(X[:3, :])
```

```
     array([[1.30069687e-03, 9.98699303e-01],
            [7.34580981e-01, 2.65419019e-01],
            [9.99775985e-01, 2.24015273e-04]])
```

```
clf.score(X, y)
```

```
0.89
```

Cómo podemos observar los resultados son muy similares, lo cuál indica ques muy válido usas cualquiera de las dos formas, aún así hacer uso de las librerias de scikit learn brindan maypr facilidad a la hora de desarrolar los ejercicios y dejan a la primerqa implementación como un ejercicio netamente académico.

✓ 0 s completado a las 13:05

Tarea #3 Inteligencia Artificial | 3 | 10 | 21

3. R/=

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{si } y = 1 \\ -\log(1 - h_\theta(x)) & \text{si } y = 0 \end{cases}$$

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y)\log(1 - h_\theta(x))$$

$$h_\theta = \frac{1}{1 + e^{-\theta^T x}}$$

$$\sigma = \frac{1}{1+e^{-x}}$$

$$\frac{d(\sigma(x))}{dx} = \frac{0 \cdot (1+e^{-x}) - (1) \cdot (e^{-x} \cdot (-1))}{(1+e^{-x})^2}$$

$$\frac{d(\sigma(x))}{dx} = \frac{(e^{-x})}{(1+e^{-x})^2} = \frac{|-1 + (e^{-x})}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

$$\frac{d(\sigma(x))}{dx} = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial(J(\theta))}{\partial(\theta_j)} = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} * \frac{1}{h_\theta(x^{(i)})} \cdot \frac{\partial(h_\theta(x^{(i)}))}{\partial(\theta_j)} \right)$$

$$+ \sum_{i=1}^{n} \left((1 - y^{(i)}) \cdot \frac{1}{(1 - h_\theta(x^{(i)}))} \cdot \frac{\partial(1 - h_\theta(x^{(i)}))}{\partial(\theta_j)} \right)$$

$$\frac{\partial(J(\theta))}{\partial(\theta_j)} = -\frac{1}{n} \cdot \left(\sum_{i=1}^{m} \left(y^{(i)} \cdot \frac{1}{h_\theta(x^{(i)})} \cdot \sigma(z)(1 - \sigma(z)) \cdot \frac{\partial(\theta^T x)}{\partial(\theta_j)} \right) \right.$$

$$\left. + \sum_{i=1}^{m} \left(1 - y^{(i)} \cdot \frac{1}{(1 - h_\theta(x^{(i)}))} \cdot (-\sigma(z)(1 - \sigma(z)) \cdot \frac{\partial(\theta^T x)}{\partial(\theta_j)} \right) \right)$$

marfil

$$\frac{\partial(J(\theta))}{\partial(\theta_j)} = -\frac{1}{n}\cdot\left(\sum_{i=1}^{n}\left(y^{(i)}\cdot\frac{1}{h_\theta(x^{(i)})}\cdot\sigma(z)(1-\sigma(z))\cdot\frac{\partial(\theta^T x)}{\partial(\theta_j)}\right)\right.$$

$$\left.+\sum_{i=1}^{n}\left((1-y^{(i)})\cdot\frac{1}{(1-h_\theta(x^{(i)}))}\cdot(-\sigma(z)(1-\sigma(z))\cdot\frac{\partial(\theta^T x)}{\partial(\theta_j)}\right)\right)$$

$$= -\frac{1}{n}\cdot\left(\sum_{i=1}^{n}\left(y^{(i)}\cdot\frac{1}{h_\theta(x^{(i)})}h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\cdot x_j^{(i)}\right)\right.$$

$$\left.+\sum_{i=1}^{n}\left((1-y^{(i)})\cdot\frac{1}{(1-h_\theta(x^{(i)}))}\cdot(-h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\cdot x_j^{(i)}\right)\right)$$

$$= -\frac{1}{n}\left(\sum_{i=1}^{n}\left(y^{(i)}\cdot(1-h_\theta(x^{(i)}))\cdot x_j^{(i)} - (1-y^{(i)})\cdot h_\theta(x^{(i)})\cdot x_j^{(i)}\right)\right)$$

$$= -\frac{1}{n}\left(\sum_{i=1}^{n}\left(y^{(i)} - y^{(i)}\cdot h_\theta(x^{(i)}) - h_\theta(x^{(i)}) + y^{(i)}\cdot h_\theta(x^{(i)})\right)\cdot x_j^{(i)}\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - h_\theta(x^{(i)})\right)\cdot x_j^{(i)}$$