LogisticRegression

April 12, 2023

1) Correr el ejemplo multiclase al final de:https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, concuya y explique cuales son los hiper parámetros por defecto.
2) Encontrar la derivada de J para un theta arbitratio en la Regresión Logística. Ver la presentaciòn de regresiòn logistica para encontrar las web de ayuda al respecto. Este punto debe hacerse a mano y escanear en PDF.
3) Implementar (adecuar) los dos métodos descritos en: https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#id13. Con los datos en el csv en teams carpeta semana 6 (data_classification.csv).

Puede descargar el código en:https://github.com/bfortuner/ml-glossary

Se espera que usted use estas funciones para implementar una clasificaciòn simple. Se le suministran las funciones se espera que usted las use, para este punto NO se debe usar una implementaciòn de la Regresión Logística en sklearn u otra librerìa.

1 Ejemplo multiclase 'Iris dataset' de sklearn

[1]:
```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```

[2]:
```
X, y = load_iris(return_X_y=True)
```

[3]:
```
clf = LogisticRegression(max_iter=1000,random_state=0).fit(X, y)
```

[4]:
```
print('Las etiquetas predichas son: ', clf.predict(X[:2, :]))
print('Las probabilidades para cada clase son: ',clf.predict_proba(X[:2, :]))
print('Puntaje de evaluar el dataset: ', clf.score(X, y))
```

```
Las etiquetas predichas son:  [0 0]
Las probabilidades para cada clase son:  [[9.81583570e-01 1.84164159e-02
1.44983478e-08]
 [9.71340098e-01 2.86598715e-02 3.01821901e-08]]
Puntaje de evaluar el dataset:  0.9733333333333334
```

Se trata de un ejemplo de clasificación multiclase (3 clases) del famoso dataset 'Iris'. Utilizamos la clase 'LogisticRegression' del modulo 'linear_model' de la libreria sklearn. Se hace una regresión logística por defecto pero inmediatemente se reporta un error.

Al correr el ejemplo tal como esta escrito en la pagina indicada obtenemos el siguiente mensaje de error: "458: ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT". Además, se nos indica "Increase the number of iterations (max_iter) or scale the data", entre otras sugerencias.

Esto fue corregido modificando el parametro max_iter=100 que está por defecto a max_iter=1000.

Para conocer los parametros por defecto de esta función podemos observarlos en la pagina online sugerida en el encabezado, se muestran a continuación:

class sklearn.linear_model.LogisticRegression(

```
penalty='l2',
*,
dual=False,
tol=0.0001,
C=1.0,
fit_intercept=True,
intercept_scaling=1,
class_weight=None,
random_state=None,
solver='lbfgs',
max_iter=100,
multi_class='auto',
verbose=0,
warm_start=False,
n_jobs=None,
l1_ratio=None)
```

Al instanciar un objeto 'clf' de la clase LogisticRegression heredamos sus métodos. Podemos ver en el ejemplo que se utiliza .predict() para predecir la clase de las 2 primeras filas del dataset y este método retorna la etiqueta predicha. Con el método .predict_proba() se obtienen las probabilidades dadas para cada instancia por cada clase posible del dataset. Y por últimmo con el método .score() estámos obteniendo el porcentaje de instancias correctamente clasificadas por el el modelo con respecto a todo el dataset, que en este ejemplo fue el mismo conjunto de datos en el que se entreno el modelo.

2 Derivada funcion de costo J de la Regresión Logística

Mirar el anexo al final del pdf, se realizo el procedimiento a mano.

3 Clasificador Logístico sin librerias

[5]:
```
import pandas as pd
import numpy as np
```

EXPLORACIÓN DEL DATASET:

[12]:
```
data= pd.read_csv('data_classification.csv')
```

[13]:
```
data.head()
```

[13]:

	suenio	estudio	pasan
0	4.855064	9.639962	1
1	8.625440	0.058927	0
2	3.828192	0.723199	0
3	7.150955	3.899420	1
4	6.477900	8.198181	1

[14]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   suenio   100 non-null    float64
 1   estudio  100 non-null    float64
 2   pasan    100 non-null    int64
dtypes: float64(2), int64(1)
memory usage: 2.5 KB
```

[15]:
```
data.describe()
```

[15]:

	suenio	estudio	pasan
count	100.000000	100.000000	100.00
mean	5.534528	4.836464	0.55
std	3.027270	3.121408	0.50
min	0.022280	0.024946	0.00
25%	2.974605	2.085163	0.00
50%	6.469398	4.203772	1.00
75%	7.866485	7.733342	1.00
max	9.947841	9.945176	1.00

SELECCIÓN DE CARACTERÍSTICAS Y ETIQUETA

[21]:
```
X=data[['suenio', 'estudio']]
Y=data[['pasan']]
```

[47]:
```
import math
import numpy

def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))
def sigmoid_prime(z):
    return sigmoid(z) * (1-sigmoid(z))
```

```
def predict(features, weights):
    z = np.dot(features, weights)
    return sigmoid(z)


def cost_function(features, labels, weights):
    observations = len(labels)

    predictions = predict(features, weights)

    #Take the error when label=1
    class1_cost = -labels*np.log(predictions)

    #Take the error when label=0
    class2_cost = (1-labels)*np.log(1-predictions)

    #Take the sum of both costs
    cost = class1_cost - class2_cost

    #Take the average cost
    cost = cost.sum() / observations

    return cost


def update_weights(features, labels, weights, lr):
    '''
    Vectorized Gradient Descent
    Features:(200, 3)
    Labels: (200, 1)
    Weights:(3, 1)
    '''
    N = len(features)

    #1 - Get Predictions
    predictions = predict(features, weights)

    #2 Transpose features from (200, 3) to (3, 200)
    # So we can multiply w the (200,1)  cost matrix.
    # Returns a (3,1) matrix holding 3 partial derivatives --
    # one for each feature -- representing the aggregate
    # slope of the cost function across all observations
    gradient = np.dot(features.T,  predictions - labels)

    #3 Take the average cost derivative for each feature
    gradient /= N
```

```
    #4 - Multiply the gradient by our learning rate
    gradient *= lr

    #5 - Subtract from our weights to minimize cost
    weights -= gradient

    return weights


def decision_boundary(prob):
    return (prob >= 0.5).astype(int)


def classify(predictions):
    decision_boundary = np.vectorize(decision_boundary)
    return decision_boundary(predictions).flatten()


def train(features, labels, weights, lr, iters):
    cost_history = []

    for i in range(iters):
        weights = update_weights(features, labels, weights, lr)

        #Calculate error for auditing purposes
        cost = cost_function(features, labels, weights)
        cost_history.append(cost)

        # Log Progress
        if i % 1000 == 0:
            pass

    return weights, cost_history


def accuracy(predicted_labels, actual_labels):
    diff = predicted_labels - actual_labels
    return 1.0 - (float(np.count_nonzero(diff)) / len(diff))


def plot_decision_boundary(trues, falses):
    fig = plt.figure()
    ax = fig.add_subplot(111)

    no_of_preds = len(trues) + len(falses)
```

```
    ax.scatter([i for i in range(len(trues))], trues, s=25, c='b', marker="o",␣
↪label='Trues')
    ax.scatter([i for i in range(len(falses))], falses, s=25, c='r',␣
↪marker="s", label='Falses')

    plt.legend(loc='upper right');
    ax.set_title("Decision Boundary")
    ax.set_xlabel('N/2')
    ax.set_ylabel('Predicted Probability')
    plt.axhline(.5, color='black')
    plt.show()
```

[52]:
```
lr = 0.001
iters=1000

weights = np.random.normal(loc=0.0, scale=1.0, size=(2, 1))
print('Los pesos antes del entrenamiento: ', weights)
weights , ch = train(X, Y, weights, lr, iters)
print('Los pesos después del entrenamiento: ', weights)
```

```
Los pesos antes del entrenamiento:  [[-0.48718359]
 [-0.80467342]]
Los pesos después del entrenamiento:  [[ 0.14265135]
 [-0.01000105]]
```

[53]:
```
h= predict(X, weights)
h = decision_boundary(h)
h
```

[53]:
```
array([[1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
```

```
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[0],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[0],
[1],
[1],
[1],
[1],
[1],
```

```
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1]])
```

[54]:
```
accuracy(h,Y)
```

[54]:
```
0.5800000000000001
```

El modelo es muy simple y no cuenta con regularización, intercepto (bias), y demás parametros que la libreria sklearn ya tiene implementados.

Se obtuvo una exactitud final del 58%

FUNCIÓN DE COSTO J(θ) DE LA REGRESIÓN LOGÍSTICA

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))]$$

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

Simplificamos:

$$\rightarrow \log(h_\theta(x^{(i)}) = \log\left(\frac{1}{1+e^{-x\theta}}\right) \quad \log\left(\frac{1}{a}\right) = -\log(a)$$

$$= -\log(1 + e^{-x\theta})$$

$$\rightarrow \log\left(1 - \frac{1}{1+e^{-x\theta}}\right) = \log\left(\frac{1+e^{-x\theta} - 1}{1+e^{-x\theta}}\right) = \log\left(\frac{e^{-x\theta}}{1+e^{-x\theta}}\right)$$

$$= \log(e^{-x\theta}) - \log(1 + e^{-x\theta})$$

$$= -x\theta - \log(1 + e^{-x\theta})$$

Reemplazando:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} [-y^{(i)} \cdot \log(1 + e^{-x^{(i)}\theta}) + (1 - y^{(i)}) \cdot (-x^{(i)}\theta - \log(1 + e^{-x^{(i)}\theta})$$

Simplificamos:

$$a := \log(1 + e^{-x^{(i)}\theta})$$

Reemplazamos:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} [-y \cdot a + (1-y)(-x\theta - a)]$$

$$J(\theta) = \frac{-1}{m} \sum_{i}^{m} [-ya - x\theta - a + yx\theta + ya]$$

$$J(\theta) = \frac{-1}{m} \sum_{i}^{m} [-x\theta - a + yx\theta]$$

$$J(\theta) = \frac{-1}{m} \sum_{i}^{m} [-x\theta - \log(1+e^{-x\theta}) + 4x\theta]$$

Simplificando

$$J(\theta) = \frac{-1}{m} \sum [-(x\theta + \log(1+e^{-x\theta})) + 4x\theta]$$

$$J(\theta) = \frac{-1}{m} \sum [-(\log(e^{x\theta}) + \log(1+e^{-x\theta})) + 4x\theta]$$

$$J(\theta) = \frac{-1}{m} \sum [-(\log(e^{x\theta}(1+e^{-x\theta})) + 4x\theta]$$

$$J(\theta) = \frac{-1}{m} \sum [-(\log(e^{x\theta}+1)) + 4x\theta]$$

$$J(\theta) = \frac{1}{m} \sum [\log(e^{x\theta}+1) - 4x\theta]$$

Derivando:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \sum \left[\frac{\partial \log(e^{x\theta}+1)}{\partial \theta} - \frac{\partial (4x\theta)}{\partial \theta} \right]$$

$$\rightarrow \frac{\partial (4x\theta)}{\partial \theta} = 4x$$

$$\rightarrow \frac{\partial (\log(e^{x\theta}+1))}{\partial \theta} = \frac{1}{e^{x\theta}+1} \cdot x e^{x\theta} = \frac{x}{(e^{x\theta}+1)e^{-x\theta}} = \frac{x}{1+e^{-x\theta}}$$

$$= x \cdot h_\theta(x)$$

Reemplazando:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \sum_{i=0}^{m} [x \cdot h_\theta(x) - xy]$$

$$\boxed{\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \sum_{i=0}^{m} [x^{(i)} (h_\theta(x) - y)]}$$