Regresión Logística con Python

José Luis Higuera Caraveo

February 28 2022

1 Regresión Logística

La regresión Logística es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica, es decir, utilizado para clasificar un conjunto de datos de acuerdo a las posibles categorías dadas por la variable a predecir.

1.0.1 Hipótesis

El algoritmo predice la probabilidad de que, cierto ejemplo, pertenezca a una determinada categoría. Pero, para cada caso de estudio se tendrá que especificar un umbral de aprobación, es un número del 0 al 1, dado por el usuario, el cual va a determinar que, si la probabilidad es mayor a este umbral entonces este ejemplo pertenece a cierta categoría.

Teniendo esto claro, nuestra hipótesis queda de la siguiente manera:

Vamos a usar un umbral de 0.5, por lo tanto:

- Si $h_0(x) \geq 0.5$, la predicción será "$y = 1$"
- Si $h_0(x) \leq 0.5$, la predicción será "$y = 0$"

Se debe tener en cuenta que $0 \leq h_0 \leq 1$.

Y como se obtiene h_0:

$$h_0(x) = g(\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n) = g(z)$$

donde:

- $g(z) = \frac{1}{1+e^{-\theta^T \cdot X}}$

Nos detenemos a analizar las fórmulas. Primero, se obtiene el valor de $\theta^T \cdot X$ y a este resultado se aplica una función la cual nos pueda convertir el resultado de $\theta^T \cdot X$ a uno que vaya entre el rango de 0 a 1, y así, dado el umbral podamos decidir si ese ejemplo lo podamos catalogar como "$y = 1$" o "$y = 0$".

Esta función es la llamada función sigmoide:

$$Sigmoide = \frac{1}{1 + e^{-X}}$$

1.0.2 Función de coste

La función de coste para una regresión lineal es la siguiente:

$$J(\theta_0,\ \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_0(x^i) - y^i)^2$$

Para la regresión logística no podemos utilizar esta misma, ya que esto nos dará como resultado una función no convexa, por lo que nunca se lograría encontrar un mínimo global óptimo para minimizar esta misma.

Tenemos que modificar esta función para tomar en cuenta los dos posibles resultados "$y = 1$" y "$y = 0$". Nuestra función de costo quedaría de la siguiente manera:

$$Cost(h_0(x), y) = \begin{cases} -log(h_0(x)) & if\ y = 1 \\ -log(1 - h_0(x)) & if\ y = 0 \end{cases}$$

Pero lo anterior pueda que sea difícil de comprender, para ello simplificamos la función para que nos quede de la siguiente manera:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_0(x_i), y_i)$$

Tomando como referencia las ecuaciones para "$y = 1$" y "$y = 0$", tenemos una fórmula que engloba los dos posibles resultados:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i log(h_0(x_i)) + (1 - y_i) log(1 - h_0(x_i))\right]$$

Para hacer la predicción de un nuevo elemento x:

$$Output\ h_0(x) = \frac{1}{1 + e^{-\theta^T x}}$$

1.0.3 Descenso del gradiente

Como obtenemos los θ óptimos?

Como en el algoritmo de regresión lineal, usaremos el descenso del gradiente para ayudarnos a encontrar estos parámetros. El algoritmo es el siguiente:

Repetir hasta converger {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_0(x^i) - y^i) \cdot x_j^i$$

}

donde: * α es la tasa de aprendizaje.

Notamos que el algoritmo luce idéntico al de regresión lineal, pero hay que dejar algo claro, ahora, la función para calcular $h_0(x)$ es distinta, se tiene que usar:

$$h_0(x) = \frac{1}{1 + e^{-\theta^T x}}$$

1.0.4 Implementación en código

Un dataset de ejemplo que nos ayuda a entender como el algoritmo de regresión logística es usado es el dataset de predicción de tumores Malignos o Benignos con base a ciertas características.

El dataset que se utilizará nos proporciona características de los tumores como

- identificación
- diagnóstico
- radio_promedio
- textura_media
- perímetro_medio
- área_media
- suavidad_media
- compacidad_media media_concavidad
- puntos cóncavos_media
- simetría_media

Entre otros datos importantes. Para esto, intentaremos predecir el diagnóstico (M = Maligno, B = Benigno) según las especificaciones de cada tumor.

Los datos se encuentran en el siguiente link: https://www.kaggle.com/yasserh/breast-cancer-dataset

[2]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
```

[3]:
```
data = pd.read_csv('../data/breast-cancer.csv')
data.head()
```

[3]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
```

```
3        0.14250          0.28390         0.2414              0.10520
4        0.10030          0.13280         0.1980              0.10430

   …  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  …         25.38          17.33           184.60      2019.0
1  …         24.99          23.41           158.80      1956.0
2  …         23.57          25.53           152.50      1709.0
3  …         14.91          26.50            98.87       567.7
4  …         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

Contamos con 569 ejemplos y 30 características, no tomamos en cuenta el ID, ni el diagnóstico.

[4]:
```
data['diagnosis'][data['diagnosis'] == 'M'].count(), \
data['diagnosis'][data['diagnosis'] == 'B'].count()
```

[4]:
```
(212, 357)
```

El dataset se conforma de 212 casos de tumores Malignos y 357 casos de tumores Benignos.

Cuando se va aplicar un algoritmo de regresión, es importante verificar que los datos con los que se va a trabajar sean numéricos. De lo contrario, se tendrá que realizar un trabajo de transformación que nos ayude a convertir las variables categóricas a numéricas.

[57]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
```

```
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Todos nuestros datos son numéricos, por lo que no es necesaria una transformación antes de aplicar el algoritmo, tampoco tenemos datos nulos.

[5]:
```
# Variables que se usarán para el algoritmo
X = data.iloc[:, 2:]
m = len(X)
y = data['diagnosis']
```

La variable a predecir es categórica, por lo que se tiene que transformar a numérica. Normalmente se utiliza una técnica de encoding, pero en este caso de estudio tenemos dos posibles resultados, por lo tanto:

- $y = 1$ cuando el diagnóstico sea "M".
- $y = 0$ cuando el diagnóstico sea "B".

[6]:
```
y = y.apply(lambda x: 0 if x == 'B' else 1)
```

Para poder manejar una solución vectorizada, es necesario agregar una columna de unos al inicio de las variables X, esto nos ayuda a que θ_0 no sea modificado.

[7]:
```
ones = [1] * len(X)

X.insert(0, 'ones', ones)
X = X.values
```

[9]:
```
import warnings
warnings.filterwarnings('ignore')
```

[11]:
```
# Función Sigmoide
def sigmoid(x):
    return 1/(1 + np.e**(-x))

# Función de predicción
def get_y_pred(x, thetas):
    return sigmoid(x.dot(thetas.T))

# Función de coste
def get_cost(y, y_pred):
    cost = 0
    for i in range(m):
        cost += (y[i] * np.log(y_pred[i])) + ((1-y[i]) * np.log(1-y_pred[i]))
    return -1 * (1/m) * cost

# Gradiente Descendente
def get_gradient(x, y, n_iter, thetas, alpha=0.01):
    for i in range(n_iter):
        y_pred = get_y_pred(x, thetas)

        thetas = thetas - (alpha * ((1 / m)*((y_pred - y).T.dot(x))))

    return thetas


# Obteniendo los theta óptimos
initial_thetas = np.zeros([X.shape[1]]).T
n_iter = 100000
thetas = get_gradient(X, y, n_iter, initial_thetas)

# Obteniendo las predicciones de acuerdo a los theta óptimos.
y_pred = get_y_pred(X, thetas)

# Obtener 0 o 1 de acuerdo al threshold.
threshold = 0.5
y_pred2 = [1 if pred > threshold else 0 for pred in y_pred]
```

La mejor manera de verificar que tan bien está prediciendo nuestro modelo es usar una matriz de confusión, esta nos ayuda a verificar, verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

La librería de Sklearn nos proporciona una función para obtener esta matriz. La cual da como resultado una matriz como sigue:

$$\begin{pmatrix} Verdaderos\ Positivos & Falsos\ Positivos \\ Falsos\ Negativos & Verdaderos\ Negativos \end{pmatrix}$$

Esta matriz también nos ayuda a calcular:

- Accuracy: De las predicciones, cual es la proporción que el algoritmo predice correctamente.

$$Accuracy = \frac{Verdaderos\ Positivos\ +\ Verdaderos\ Negativos}{m}$$

- Precisión: Del total de predicciones positivas, que proporción realmente es verdadero positivo.

$$Precisin = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos\ +\ Falsos\ Positivos}$$

- Recall: Del total de predicciones positivas, que proporción realmente predice como verdadero positivo.

$$Recall = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos\ +\ Falsos\ Negativos}$$

[13]:
```
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y, y_pred2)

VP = conf_matrix[0][0]
FP = conf_matrix[0][1]
VN = conf_matrix[1][1]
FN = conf_matrix[1][0]

accuracy = (VP + VN) / m
presicion = VP / (VP + FP)
recall = VP / (VP + FN)

print('Accuracy: {}, Precisión: {}, Recall: {}'.format(accuracy, presicion,␣
 ↪recall))
```

```
Accuracy: 0.9121265377855887, Precisión: 0.9971988795518207, Recall:
0.8790123456790123
```

Nuestro algoritmo obtiene una precisión del casi 100%, con una precisión y recall altos. Por lo que se concluye que este algoritmo ayuda a predecir casi con 100% de exactitud un tumor Maligno o Benigno.

1.0.5 Algoritmo con Sklearn

[24]:
```
from sklearn.linear_model import LogisticRegression

X = data.iloc[:, 2:].values
m = len(X)
y = data['diagnosis']

# Convirtiendo los datos categóricos a numéricos
y = y.apply(lambda x: 0 if x == 'B' else 1)

# Entrenando el modelo
model = LogisticRegression()
model.fit(X, y)

# Obteniendo predicciones
y_pred = model.predict(X)

# Matriz de confusión
conf_matrix = confusion_matrix(y, y_pred2)

VP = conf_matrix[0][0]
FP = conf_matrix[0][1]
VN = conf_matrix[1][1]
FN = conf_matrix[1][0]

accuracy = (VP + VN) / m
presicion = VP / (VP + FP)
recall = VP / (VP + FN)

print('Accuracy: {}, Precisión: {}, Recall: {}'.format(accuracy, presicion,␣
 ↪recall))
```

```
Accuracy: 0.9121265377855887, Precisión: 0.9971988795518207, Recall:
0.8790123456790123
```

Podemos Observar como obtenemos los mismos resultados, pero con la ventaja de que ahora el procedimiento se aplica con pocas líneas de código.