Fundamentos de Inteligencia Artificial

Regresión Logística

Oliver Fernando Cuate González

20 de septiembre de 2022

Contenido

Contenido de la presentación

El problema de clasificación

- Supongamos que tenemos dos clases o categorías C_1 y C_2, cada una de ellas con ciertas **características** que pueden representarse mediante valores reales.
- Denotemos a n como la cantidad de muestras y a m como el número de características.
- Ya que las características pueden representarse mediante valores reales, entonces es posible representarlas en un espacio m-dimensional.
- Supongamos también que es posible dividir las clases mediante un hiperplano.
- Es decir, buscamos una combinación lineal de las características que represente la división entre clases.
- Si logramos obtener dicha combinación lineal, entonces podremos clasificar cada nueva observación.

Contenido de la presentación

Regresión Logística

Regresión Logística

La **regressión logística** es un algoritmo de clasificación que aprende una función que aproxima $P[Y|X]$, es decir, se puede interpretar como la *"probabilidad"* de pertenencia a una clase dado un vector de características.

Regresión Logística

Regresión Logística

La **regressión logística** es un algoritmo de clasificación que aprende una función que aproxima $P[Y|X]$, es decir, se puede interpretar como la *"probabilidad"* de pertenencia a una clase dado un vector de características.

En la práctica, se supone que se puede aproximar como una función **sigmoide** que se aplica a una combinación lineal de características de entrada.

Regresión Logística

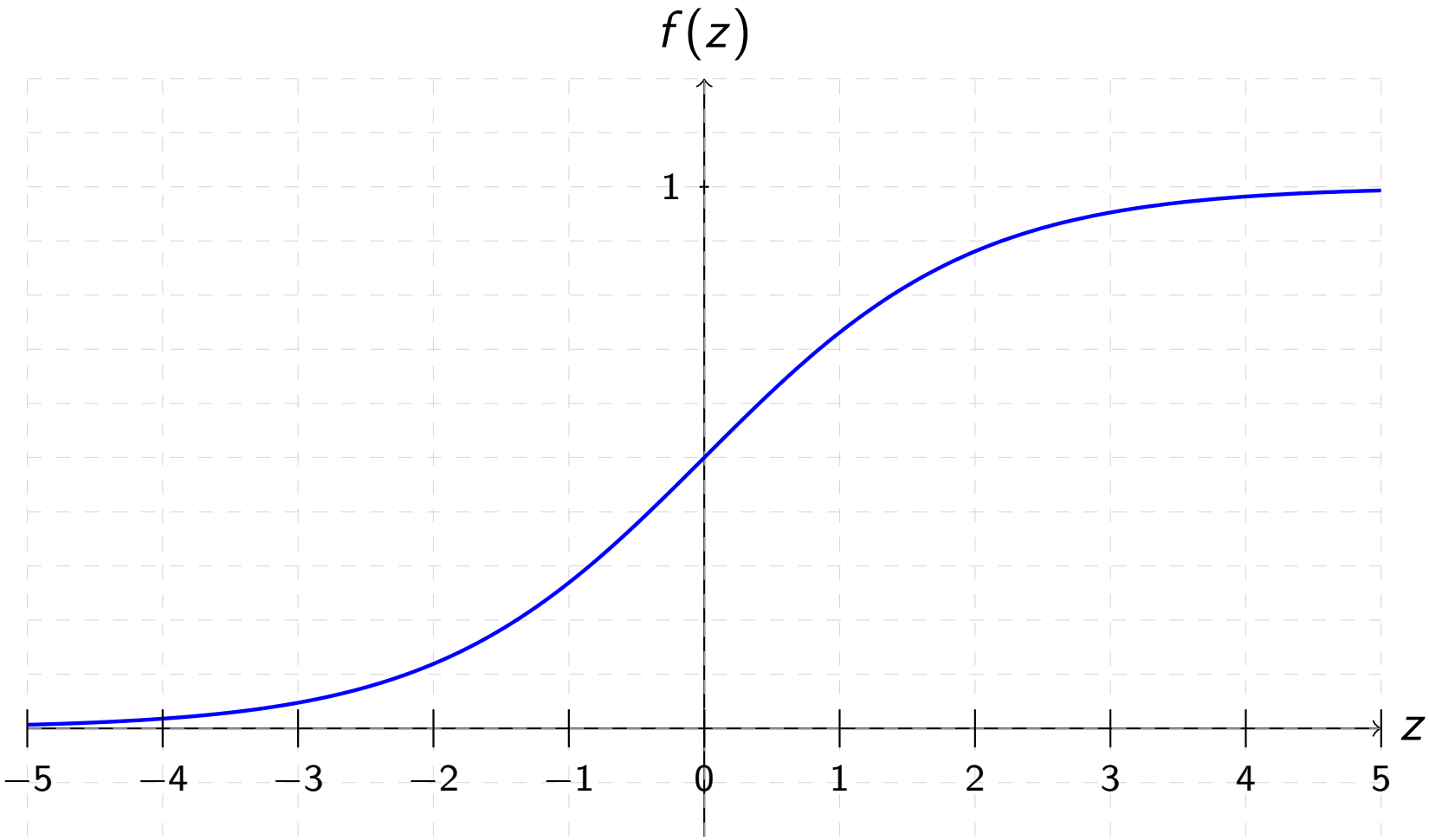
Regresión Logística

La **regressión logística** es un algoritmo de clasificación que aprende una función que aproxima $P[Y|X]$, es decir, se puede interpretar como la *"probabilidad"* de pertenencia a una clase dado un vector de características.

En la práctica, se supone que se puede aproximar como una función **sigmoide** que se aplica a una combinación lineal de características de entrada.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Función sigmoide



Deduciendo la función de costo

Consideremos el caso de una variable dependiente x_1 (característica) y una variable dependiente y, de tal forma que:

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-w_0 - w_1 x_1}} \tag{2}$$

Deduciendo la función de costo

Consideremos el caso de una variable dependiente x_1 (característica) y una variable dependiente y, de tal forma que:

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-w_0 - w_1 x_1}} \tag{2}$$

Necesitamos penalizar valores que difieran del valor real de las muestras $y^{(i)} \in \{0, 1\}$; además, recordemos que la imagen de la función sigmoide está en el intervalo $(0, 1)$. Es decir, el valor de $y - f(W)$ es a lo más uno y eso significa un gran error que debe ser penalizado.

Deduciendo la función de costo

Consideremos el caso de una variable dependiente x_1 (característica) y una variable dependiente y, de tal forma que:

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-w_0 - w_1 x_1}} \tag{2}$$

Necesitamos penalizar valores que difieran del valor real de las muestras $y^{(i)} \in \{0, 1\}$; además, recordemos que la imagen de la función sigmoide está en el intervalo $(0, 1)$. Es decir, el valor de $y - f(W)$ es a lo más uno y eso significa un gran error que debe ser penalizado.

Una alternativa es usar la función $\ln(x)$, que tiende a infinito cuando $x \to 0$ y es cero cunando $x = 1$. De este modo:

$$r(z) = y \ln\left(f\left(z\right)\right) + (1 - y) \ln\left(1 - f\left(z\right)\right) \tag{3}$$

Ejemplo

Ejemplo: Vamos a asignar a nuestras clases C_1 y C_2 los valores de 1 y 0, respectivamente.
Entonces, tendremos lo siguiente para algunas muestras:

$y^{(i)}$	$f(z^{(i)})$	Error: $r(z^{(i)})$
0	0.99	4.60
0	0.01	0.01
1	0.99	0.01
1	0.01	460

Ejemplo

Ejemplo: Vamos a asignar a nuestras clases C_1 y C_2 los valores de 1 y 0, respectivamente.
Entonces, tendremos lo siguiente para algunas muestras:

$y^{(i)}$	$f(z^{(i)})$	Error: $r(z^{(i)})$
0	0.99	4.60
0	0.01	0.01
1	0.99	0.01
1	0.01	460

Para el caso de n muestras tenemos entonces que minimizar la función:

$$R(Z) = \sum_{i=1}^{n} - \left(y^{(i)} \ln \left(f\left(z^{(i)} \right) \right) + \left(1 - y^{(i)} \right) \ln \left(1 - f\left(z^{(i)} \right) \right) \right) \qquad (4)$$

Obteniendo el gradiente

Notemos que:

$$\frac{d}{dz} f(z) = f(z)(1 - f(z)) \tag{5}$$

Obteniendo el gradiente

Notemos que:

$$\frac{d}{dz} f(z) = f(z)(1 - f(z)) \tag{5}$$

Además:

$$\begin{aligned} \frac{d}{dz} r(z) &= \frac{y}{f(z)} - \frac{1-y}{1-f(z)} = \frac{y(1-f(z)) - (1-y)f(z)}{f(z)(1-f(z))} \\ &= \frac{y(1-f(y))}{f(z)(1-f(z))} \end{aligned} \tag{6}$$

Obteniendo el gradiente

Notemos que:

$$\frac{d}{dz} f(z) = f(z)(1 - f(z)) \tag{5}$$

Además:

$$\begin{aligned} \frac{d}{dz} r(z) &= \frac{y}{f(z)} - \frac{1-y}{1-f(z)} = \frac{y(1-f(z)) - (1-y)f(z)}{f(z)(1-f(z))} \\ &= \frac{y(1-f(y))}{f(z)(1-f(z))} \end{aligned} \tag{6}$$

Así:

$$\frac{\delta r(z)}{\delta w_j} = \frac{\delta r}{\delta f} \frac{\delta f}{\delta z} \frac{\delta z}{\delta w_j} = (y - f(z)) x_j \tag{7}$$

Gradiente para regresión logística

$$\nabla_W J(W) = \frac{1}{n} \begin{pmatrix} \sum_{i=1}^{n} \left(-y^{(i)} + w_0 + w_1 x_1^{(i)} + \ldots + w_1 x_n^{(i)} \right) \cdot 1 \\ \sum_{i=1}^{n} \left(-y^{(i)} + w_0 + w_1 x_1^{(i)} + \ldots + w_1 x_n^{(i)} \right) \cdot x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{n} \left(-y^{(i)} + w_0 + w_1 x_1^{(i)} + \ldots + w_1 x_n^{(i)} \right) \cdot x_m^{(i)} \end{pmatrix} \tag{8}$$

Contenido de la presentación

Funciones básicas necesarias

Importar bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
```

Función de activación:

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

Función de costo:

```
def costo(y, f, n):
  return (1/n)*np.sum(-(y*np.log(f)+(1-y)*np.log(1-f)))
```

Funciones para el descenso del gradiente

Función de propagación hacia adelante

```
def forwardPropagation(x,weights):
  return sigmoid(x.dot(weights))
```

Función de entrenamiento:

```
def training( x,y,epochs,learning_rate ):
  w = np.random.uniform(1,1,( x.shape [1],1))
  n = len(y)
  cost = np.zeros((epochs,1))

  for epoch in range(epochs):
    f = forwardPropagation(x,weights)
    cost[epoch]= costo(y,f,n)
    w = w-(1/n)*learning_rate)*x.T.dot(-y+f))

  return weights,cost
```

Función de predicción y datos

Función de predicción

```
def predict(x,weights):
  res = forwardPropagation(x,weights)
  predicted = np.ones( (len(x),1))
  predicted[res<0.5]=0
  return predicted
```

Preparación de datos:

```
x_orig = np.array([[0.5,0.7],[0.6,0.8],[0.1,0.2],[0.15,0.2],[0.1,0.1],[0.6,0.6]])
y = np.array([[0],[0],[1],[1],[1],[0]])
print( x_orig.shape )
print(y.shape )
x = np.column_stack((np.ones(len(x_orig)), x_orig)) #Se agrega un uno para el bias
```

Ejecutar

Entrenar

```
epochs=100
w,l = training(x,y,epochs,0.5)
plt.plot(np.arange(0,epochs,1),l)
plt.show()
```

Predecir

```
print(predict(x,w))
```

Contenido de la presentación

Datos del dataset del Iris

El dataset iris está incluído en la biblioteca sklearn

```
from sklearn import datasets
iris = datasets.load_iris()
```

Explorar el dataset

```
dir(iris)

print(iris.target)
print(iris.target_names)

print(iris.data)
print(iris.DESCR)
```

Preparar el Dataset

```
def dropOneClass(dataset,classToRemove):
  newX = []
  newY = []
  for i in range(len(dataset.data)):
    if iris.target[i] != classToRemove:
      newX.append(iris.data[i])
      newY.append(iris.target[i])
  newX = np.array(newX)
  newY = np.array(newY,ndmin = 2)
  return newX,newY.T

x_orig,y = dropOneClass(iris,0)
print(x_orig.shape)

y[y==1] = 0
y[y==2] = 1
print(y.shape)

x = np.column_stack((np.ones(len(x_orig)), x_orig ))
print(x.shape)
```

¿Preguntas?