Aprendizaje Automático y Big Data, Práctica 2

Romain Contrain y Guillermo Martín Sánchez

Marzo 2020

1 Objetivo

El objetivo de esta práctica es implementar el algoritmo de regresión logística. Se trata de un algoritmo de aprendizaje supervisado de clasificación que busca encontrar la ecuación logística que clasifique mejor ciertos puntos de forma binaria. En concreto, buscamos el vector θ tal que la función $h_\theta = \frac{1}{1+e^{-\theta^T x}}$ minimice la función coste

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(h_\theta(1-x^{(i)}))] \tag{1}$$

Intuitivamente, es encontrar una función que nos diga cómo de probable es que un punto dado pertenezca a la clase positiva ($y = 1$). Así, esperamos que con este modelo si $h_\theta(x) \geq 0.5$ lo clasificaremos correctamente como perteneciente a la clase positiva.

Por otro lado, también hemos visto técnicas como la regularización (para evitar el *overfitting*) y la expansión polinómica de los atributos (para conseguir curvas de separación del conjunto de datos que no sean necesariamente rectas)

2 Método y Resultados

Hemos usado el método *fmin_tnc* de la librería *scipy.optimize* para calcular el mínimo de la función de coste. Para ello hemos implementado la función de coste en Eq 1 y su gradiente:

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \tag{2}$$

y se lo hemos pasado como parámetros junto a un valor inicial de $\theta_0 = (0, ..., 0)$.

Primero analizamos el caso de un dataset donde los elementos son estudiantes con dos atributos: la nota que sacaron en un primer examen y la nota que sacaron en un segundo (sobre 100). La variable a estimar y es si fueron admitidos ($y = 1$) o rechazados ($y = 0$) en la institución. En la Figura 1 vemos estos datos representados.

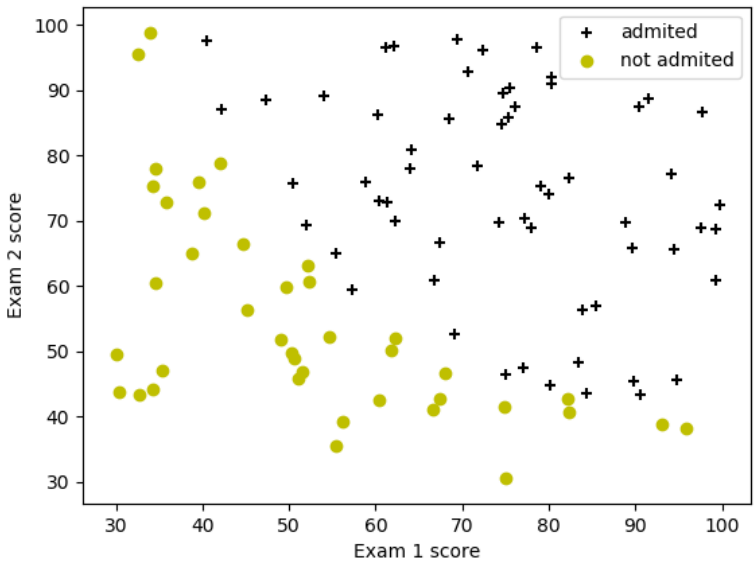


Figure 1: Puntos del dataset según sus resultados en los exámenes y si fueron admitidos o no

En la Figura 2 imprimimos la frontera de decisión que hemos obtenido, es decir, los puntos x tales que $h_\theta(x) = 0.5$. Como podemos ver, separa correctamente las clases.

En concreto, hemos calculado con los propios datos de entrenamiento la precisión del método. Para ello calculamos para cada estudiante $x^{(i)}$ el valor del modelo $y' = h_\theta(x^{(i)})$. Si este valor $p^{(i)}$ cumple que $p^{(i)} \geq 0.5$ predecimos que el estudiante fue admitido y, si no, que fue rechazado. Luego comparamos la predicción con el resultado actual $y^{(i)}$ y vemos que porcentaje de acierto hemos obtenido.

En este caso obtenemos un porcentaje de acierto del 89%. Es importante recalcar que al haber sido calculado este porcentaje de acierto sobre los datos de entrenamiento y no sobre unos datos de test, esta precisión es mejor de la que cabe esperar con datos aún no vistos por el modelo.

En el segundo apartado hemos utilizado la regresión logística para clasificar microchips en aquellos que pasarán el control de calidad ($y = 1$) y los que no ($y = 0$) dependiendo del resultados de dos tests a los que se ven sometidos.

Como se puede ver en la Figura 3 los datos ahora no son linealmente separables y, por lo tanto, hace falta usar una frontera de decisión que no sea una recta. Por ello, combinamos los atributos originales para extender cada punto con términos polinómicos de x_1 y x_2 hasta la sexta potencia. Utilizamos para

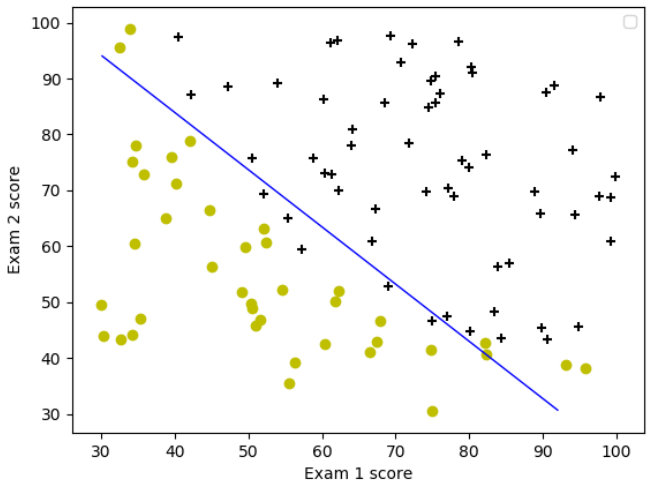


Figure 2: Puntos del dataset y frontera de decisión calculada

ello *PolynomialFeatures* de la librería *sklearn.preprocessing*.

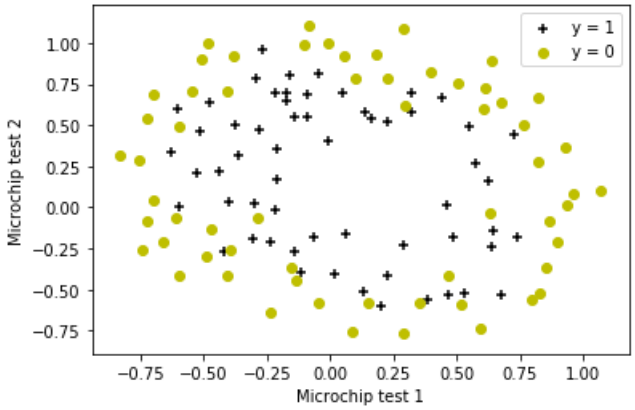


Figure 3: Puntos del dataset según sus resultados en los tests y si pasaron el control o no

Usamos regularización para estudiar y mitigar el *overfitting* del modelo a los datos de entrenamiento. Para ello añadimos un parámetro λ al cálculo del coste:

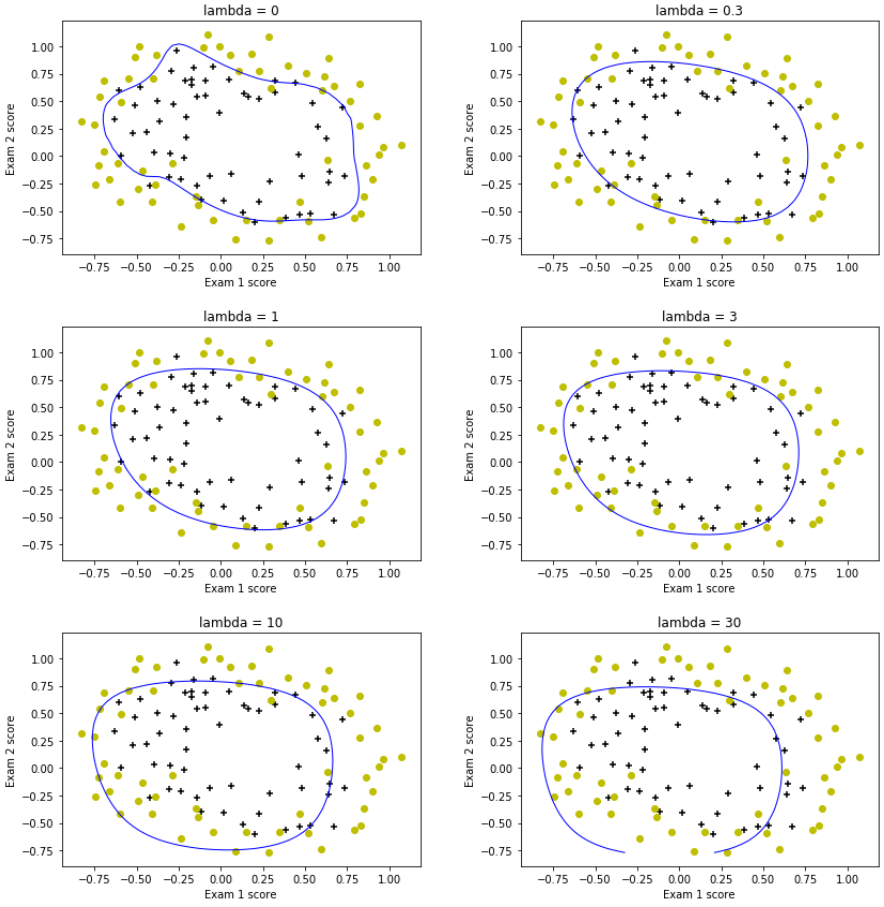
$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \tag{3}$$

y al cálculo del gradiente:

$$\begin{cases} \frac{\delta J(\theta)}{\delta \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} & j = 0 \\ \frac{\delta J(\theta)}{\delta \theta_j} = (\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}) + \frac{\lambda}{m} \theta_j & j \geq 1 \end{cases}$$

Estudiamos cómo cambia la frontera de decisión y la precisión sobre los datos de entrenamiento con $\lambda \in \{0, 0.3, 1, 3, 10, 30\}$.

Como vemos en la Figura 2, para un valor de $\lambda = 0$ obtenemos un sobreajuste del modelo. Es en esta caso cuando la precisión a los datos de entrenamiento es máxima con un valor de 0.873 (de nuevo recordemos que esto no es necesariamente lo mejor ya que el sobreajuste suele implicar una peor precisión en datos nuevos). Posteriormente con $\lambda \in \{0.3, 1\}$ obtenemos modelos que tiene peor precisión pero que parecen sufrir menos de *overfitting*. Es posible que estos modelos sean mejores para predecir nuevos datos. Finalmente, para $\lambda \in \{3, 10, 30\}$ los modelos se ajustan demasiado poco a los datos.



3 Conclusiones

La regresión logística es una técnica de clasificación de aprendizaje supervisado que calcula una buena frontera de decisión para los datasets estudiados. Para casos en los que la frontera tiene que ser diferente a una recta podemos usar, gracias al truco de expandir los atributos con relaciones polinómicas, el mismo método, lo cuál hace que sea una técnica de aprendizaje muy versátil. Además hemos visto como la regularización puede ayudar a arreglar el problema de *overfitting* y cómo es importante acertar con el valor correcto de λ para no hacerlo demasiado grande y generar un modelo con *underfitting*

4 Código

```
# -*- coding: utf-8 -*-
"""
Coursework 2: Logistical regression
"""

#%%
"""
Imports and definitions
"""



import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
import scipy.optimize as opt

def carga_csv(file_name):
    """
    carga el fichero csv especificado y lo
    devuelve en un array de numpy
    """
    valores = read_csv(file_name,header=None).values
    #suponemos que siempre trabajaremos con float
    return valores.astype(float)

def sigmoid(z):
    """
    sigmoid function
    can be applied on numbers and on numpy arrays of any dimensions
    """
    return 1 / (1 + np.exp(-z))

def coste(theta, X, Y):
    """
    cost function
    computes J(theta) for a given dataset
    """
    m = np.shape(X)[0]
    H = sigmoid((np.dot(X,theta)))
    J = -1/m * ( np.log(H).transpose().dot(Y)
                 + np.log(1-H).transpose().dot(1-Y))
    return J
```

```
def gradiente(theta, X, Y):
    """
    gradient function
    computes the gradient of J at a given theta for a given dataset
    """
    m = np.shape(X)[0]
    H = sigmoid(np.dot(X,theta))
    G = 1/m * X.transpose().dot(H - Y)
    return G

def coste_reg(theta, X, Y, lamb):
    """
    cost function with regularization
    computes J(theta) for a given dataset
    with a regularization factor of lambda
    """
    m = np.shape(X)[0]
    H = sigmoid((np.dot(X,theta)))
    J = -1/m * ( np.log(H).transpose().dot(Y)
                 + np.log(1-H).transpose().dot(1-Y))
    reg = lamb/(2*m)*np.sum(theta[1:]**2)
    return J + reg

def gradiente_reg(theta, X, Y, lamb):
    """
    gradient function with regularization
    computes the gradient of J at a given theta for a given dataset
    with a regularization factor of lambda
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    H = sigmoid(np.dot(X,theta))
    G = 1/m * X.transpose().dot(H - Y)
    reg = np.ones(n,)
    reg[0] = 0
    reg = (lamb/m)*reg*theta
    return G + reg

def precision(theta,X,Y):
    """
    accuracy function
    computes the accuracy of the logistic model theta on X with true target variable Y
    """
    m = np.shape(X)[0]
    H = sigmoid(np.dot(X,theta))
    H[H >= 0.5] = 1
```

```
    H[H < 0.5] = 0
    return np.sum(H == Y)/m

def pinta_frontera_recta(X, Y, theta):
    """
    draws straight decision borders
    """
    plt.figure()
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
               np.linspace(x2_min, x2_max))
    h = sigmoid(np.c_[np.ones((xx1.ravel().shape[0], 1)),
                      xx1.ravel(),
                      xx2.ravel()].dot(theta))
    h = h.reshape(xx1.shape)
    # el cuarto parámetro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    # Add the data to the graph
    plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k')
    plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y')
    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')
    plt.show()

def pinta_frontera_curva(X, Y, theta, poly, lamb):
    """
    draws polynomial decision borders
    adds the value of the regularization parameter for wich it was computed
    """
    plt.figure()
    x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
    x2_min, x2_max = X[:, 1].min(), X[:, 1].max()
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max),
                           np.linspace(x2_min, x2_max))
    h = sigmoid(poly.fit_transform(np.c_[xx1.ravel(),
                            xx2.ravel()]).dot(theta))
    h = h.reshape(xx1.shape)
    # el cuarto parámetro es el valor de z cuya frontera se
    # quiere pintar
    plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
    # add the data to the graph
    plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k')
    plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y')
    plt.title('lambda = ' + str(lamb))
```

```
    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')
    plt.show()


#%%
"""
1 - Logistical regression
"""


# loading the data
datos = carga_csv('ex2data1.csv')
X = datos[:, :-1]
Y = datos[:, -1]
m = np.shape(X)[0]
n = np.shape(X)[1]

# Visualizing the data
pos = np.where(Y == 1)
not_pos = np.where(Y == 0)
plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k', label = 'admited')
plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y', label = 'not admited')
plt.legend()
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')

# Add the column of 1s
X = np.hstack([np.ones([m,1]),X])

theta = np.zeros((n+1,))
# Compute the best value for theta
result = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente, args=(X,Y), messages=0)
theta_opt = result[0]

# Display the border
pinta_frontera_recta(X[:,1:], Y, theta_opt)

# Display the accuracy of the model
p = precision(theta_opt,X,Y)
print('Model accuracy : {0:1.3g}'.format(p))


#%%
"""
2 - Regularized logistical regression
```

```
"""


datos = carga_csv('ex2data2.csv')
X = datos[:, :-1]
Y = datos[:, -1]
m = np.shape(X)[0]

# Visualizing the data
pos = np.where(Y == 1)
not_pos = np.where(Y == 0)
plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k', label = 'y = 1')
plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y', label = 'y = 0')
plt.legend()
plt.xlabel('Microchip test 1')
plt.ylabel('Microchip test 2')

# Adding new attributes
poly = PolynomialFeatures(6)
X_pol = poly.fit_transform(X)
n = np.shape(X_pol)[1]

theta = np.zeros((n,))
lamb_arr = [0,0.3,1,3,10,30]
# Compute the optimal value for theta using various values of lambda
for lamb in lamb_arr:
    # Compute the best value for theta
    result = opt.fmin_tnc(func=coste_reg, x0=theta, fprime=gradiente_reg,
                                      args=(X_pol,Y,lamb), messages = 0)
    theta_opt = result[0]
    # Display the border
    pinta_frontera_curva(X_pol[:,1:], Y, theta_opt,poly,lamb)
    # Display the accuracy of the model
    p = precision(theta_opt,X_pol,Y)
    print('With lambda = ' + str(lamb) + ' we get an accuracy of {0:1.3g}'.format(p))
```