Práctica 3 (Clasificador)

¿Cómo he planteado la práctica?

Como es lógico, primero necesitamos los datos, en este caso leemos el .MAT 'ex3data1', y separamos los datos en X e Y, como está en un diccionario, accedemos como un array en el que, en lugar de posiciones, tenemos el nombre de las columnas: data['X'] y data['Y'].

Además, por comodidad, he añadido una columna de 1's al principio del eje X para que, al multiplicar las Thetas, se pueda hacer de forma vectorizada.

También he implementado las funciones de coste y gradiente regularizadas, que tienen como parámetros:

- Theta
- X
- Y
- lam

En el siguiente paso, como pide el enunciado, he calculado el valor óptimo en la función 'oneVsAll' de las θ's a través de la función FMIN_TNC de Scipy para cada etiqueta, dichos resultados los he devuelto como una matriz.

Finalmente he predicho los resultados, multiplicando cada ejemplo X_i por cada etiqueta para θ (10 distintas), y he calculado su precisión con $\frac{aciertos}{total}$.

La clasificación ha arrojado unos resultados cercanos al 95%.

Código comentado y gráficas

1. Importar las librerías

```
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
```

2. Obtener los datos del CSV:

```
data = loadmat('ex3data1.mat')
Y = data['y'].ravel()
X = data['X']

m = X.shape[0]
```

Es interesante destacar que en la Y necesitamos hacer ravel(), para que la dimensión sea de (5000,) y no de (5000, 1), ya que, si no lo hacemos, la función FMIN_TNC suele dar problemas.

```
sample = np.random.choice(X.shape[0], 10)
plt.imshow(X[sample, :].reshape(-1, 20).T)
plt.axis('off')

X = np.hstack([np.ones([m, 1]), X])
n = X.shape[1]
theta = np.zeros(n)
```

Como he comentado anteriormente, he añadido con hstack una columna de unos por comodidad y, además, he inicializado las θ's.

3. Ahora creamos las funciones:
   - Sigmoide

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

   - De coste: en la que divido los dos términos (positivo y negativo) y posteriormente los sumo. Tras ello, multiplico por $\frac{-1}{m}$
     Además, añado regularización.

```python
def coste(theta, X, Y, lam):
    m = X.shape[0]
    n = X.shape[1]


    h_theta = np.dot(X, theta)
    sig = sigmoid(h_theta)
    positive = np.dot(np.log(sig).T, Y)
    negative = np.dot(np.log(1 - sig).T, 1 - Y)
    J_theta = (-1 / m) * (positive + negative)

    # Regularizacion
    reg = (lam /(2 * m)) * np.sum(np.square(theta))

    # Coste Regularizado
    J_theta += reg

    return J_theta
```

   - De gradiente: además añado regularización.

```python
def gradiente(theta, X, Y, lam):
    m = X.shape[0]
    n = X.shape[1]

    h_theta = np.dot(X, theta.T)
    sig = sigmoid(h_theta)
    gradient = (1/m) * np.dot(sig.T - Y, X)

    # Regularizacion
    reg = (lam / m) * theta

    # Gradiente Regularizada
    gradient += reg

    return gradient
```

4. En este paso obtenemos las θ's óptimas de todas las etiquetas gracias a la función FMIN_TNC de Scipy, para ello, en cada iteración nos quedamos con la etiqueta que nos interesa, es decir, en la iteración 1 entrenaremos el clasificador para la etiqueta 1, y en la iteración 10, para la etiqueta 0 (Por eso va desde 1 a num_etiquetas + 1).
   En #13 lo que hacemos es quedarnos con las etiquetas que coinciden con la iteración (True), y el resto lo dejamos a False.
   Obtenemos las $\theta_{OPTIMAS}$ con FMIN_TNC
   Finalmente devolvemos los resultados como un array.

```
1  def oneVsAll(X, Y, num_etiquetas, lam):
2      """
3      oneVsAll entrena varios clasificadores por regresión logistica
4      con término de regularización 'reg' y devuelve el resultado en
5      una matriz, donde la fila i-ésima corresponde al clasificador
6      de la etiqueta i-ésima
7      """
8
9      resultado = []
10
11     for i in range(1, num_etiquetas + 1):
12
13         Y_aux = (Y == i) * 1
14
15         theta_optima, _, _ = opt.fmin_tnc(
16             func=coste,
17             x0 = theta,
18             fprime=gradiente,
19             args=(X, Y_aux, lam)
20         )
21
22         resultado.append(theta_optima)
23
24     return np.array(resultado).T
25
26
27 theta_opt = oneVsAll(X, Y, 10, 0.1)
```

NOTA: Esta función solo puede utilizarse cuando, a la función de coste y a la función de gradiente, le pasamos primero el parámetro θ y después 'X','Y' y 'λ'

5. Terminando esta parte, lo que hacemos es crear una función 'predecir', lo que hace dicha función es quedarse con la posición + 1 para obtener el resultado, para ello, multiplicamos cada ejemplo (X_i) por las distintas θ's en cada etiqueta. Además, nos quedamos con la posición y le sumamos 1. Devolvemos un array con las predicciones.

```
def predecir(X, Y, theta):
    Y_hat = []
    for i in range(X.shape[0]):
        ejemplo = X[i]
        resultados = np.dot(ejemplo.T, theta)
        num = np.argmax(resultados) + 1    # Va de 0 a 9, no de 1 a 10 -> +1
        Y_hat.append(num)

    return np.array(Y_hat)
```

6. Finalmente, para evaluar la precisión de nuestra clasificación, debemos calcular los aciertos respecto a los resultados que tenemos. Es decir , comparamos nuestras predicciones 'Y_hat con la 'Y' y sumamos todos los aciertos.
   Finalmente aplicamos la fórmula de $\frac{aciertos}{total} * 100$ que nos devolverá un porcentaje de aciertos.

```
def precision(X, Y, theta):
    Y_hat = predecir(X, Y, theta)

    return np.round(
        np.sum(Y_hat == Y) / m * 100,
        decimals = 2
    )
```

```
print("La precisión del Clasificador es de aproximadamente un: {}%"
      .format(precision(X, Y, theta_opt)))
```

```
La precisión del Clasificador es de aproximadamente un: 96.5%
```

Práctica 3 (Redes Neuronales)

¿Cómo he planteado la práctica?

Como es lógico, primero necesitamos los datos, en este caso leemos el CSV 'ex2data2', y separamos los datos en X e Y, siendo Y la última columna, y X las anteriores.

Posteriormente he dibujado los ejemplos positivos (o) y negativos(x), esto nos servirá para posteriormente verificar que la frontera de decisión tiene sentido.

También he implementado las funciones de coste y gradiente regularizados, que tienen como parámetros:

- Theta
- X
- Y
- Lambda

En el siguiente paso, como pide el enunciado, he calculado el valor óptimo de las θ's a través de la función FMIN_TNC de Scipy.

Finalmente he dibujado la frontera de decisión y experimentado con diferentes valores para λ y, por ende, diferentes θ's. Además, he evaluado la fiabilidad que nos ofrecen dichos valores.

En todos los pasos, he verificado el correcto cálculo de las funciones.

¿Cómo he planteado la práctica?

Como es lógico, primero necesitamos los datos, en este caso leemos el .MAT 'ex3data1', y separamos los datos en X e Y, análogo a la parte anterior, además, he cargado en 'w' el archivo .MAT 'ex3weights', al ser un diccionario he preferido dejarlo como está en lugar de dividirlo en 'Theta1' y 'Theta2'.

También he implementado una función forward_prop, a la que le paso los parámetros:

- X
- Y
- W

Posteriormente he predicho y calculado la precisión de los resultados, de forma análoga a la parte anterior.

Código comentado y gráficas

1. Importar las librerías

```
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
```

2. Obtener los datos del CSV:

```
data = loadmat('ex3data1.mat')
Y = data['y'].ravel()
X = data['X']
w = loadmat('ex3weights.mat')


```

3. Implementamos la función de Forward Propagation, para ello hay un patrón:

$$a^1 = x \qquad \text{ó} \qquad a^i = g(z^i)$$
$$a_i = [1, a_i]$$
$$z_2 = \theta^{i-1} * a^{i-1}$$
$$a^i = g(z^i)$$
$$\dots$$

Y devolvemos la última 'a' conseguida, en este caso 'a^3'

```
def forward_prop(X, Y, w):
    a1 = X
    a1 = np.hstack([np.ones([X.shape[0], 1]), a1])

    z2 = np.dot(w['Theta1'], a1.T)
    a2 = sigmoid(z2).T
    a2 = np.hstack([np.ones([a2.shape[0], 1]), a2])

    z3 = np.dot(w['Theta2'], a2.T)
    a3 = sigmoid(z3).T

    return a3
```

4. Predecimos los resultados al igual que en la anterior parte, en este caso, en lugar de llamar a oneVsAll, llamamos a forward_prop

```
def predecir_nn(X, Y, w):
    Y_hat = []

    pred = forward_prop(X, Y, w)

    for i in range(pred.shape[0]):
        ejemplo = pred[i]
        num = np.argmax(ejemplo) + 1    # Va de 0 a 9, no de 1 a 10 -> +1
        Y_hat.append(num)

    Y_hat = np.array(Y_hat)

    return Y_hat
```

5. Además, calculamos la precisión, de la misma forma que en la parte anterior.

```
def precision_nn(X, Y, w):
    Y_hat = predecir_nn(X, Y, w)

    return np.round(
        np.sum(Y_hat == Y) / m * 100,
        decimals = 2
    )
```

6. Y finalmente, mostramos la precisión conseguida:

```
print("La precisión de la Red Neuronal es de aproximadamente un: {}%"
      .format(precision_nn(X, Y, w)))
```

```
La precisión de la Red Neuronal es de aproximadamente un: 97.52%
```