3regresionLogisticaMultiClase

June 19, 2019

```
In [2]: #José Antonio Garrido SUaldea
```

```
In [3]: ###############################
        #REGRESIÓN LOGÍSTICA MULTICLASE
        ###############################
```

```
In [4]: # Cargamos los datos:
        # en y se guardarán las etiquetas de los números (qué número es)
        # en X se guardarán las imágenes en sí mismas (los valores de los píxeles)
        from scipy.io import loadmat
        data = loadmat ('ex3data1.mat')
        # se pueden consultar las claves con data.keys()
        y = data['y']
        X = data['X']
        # almacena los datos leídos en X, y
```

```
In [5]: # Selecciona aleatoriamente 10 ejemplos y los pinta
        import numpy as np
        import matplotlib.pyplot as plt
        sample = np.random.choice(X.shape[0] ,10)
        plt.imshow(X[sample, :].reshape(-1, 20).T)
        plt.axis('off')
        plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

```
In [6]: # Número de píxeles por imagen (cada píxel equivale a una feature)
        #, 400 por cada imagen
        m = X.shape[1]
        o = np.zeros(m) # Theta inicializado con ceros
        reg = 0.01
```

```
In [7]: # Etiquetamos cada número diciendo si es o no un 0 en este caso
        y_0 = np.array([1 if i == 10 else 0 for i in y])
```

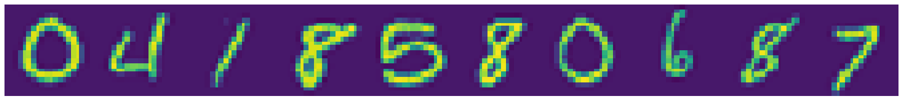
```
In [8]: # Hipótesis (probabilidad de que sea o no sea la etiqueta que
        #se quiere predecir o la probabilidad de que sea 0 en este caso)
```

```
        def ho(X, o):
            ##valoracion o estimacion de acierto para cada imagen
            return 1 / (1+np.exp(-X.dot(o.transpose())))

In [9]: def coste(o):
            return (1/m*((-y*np.log(ho(X, o)) - (1 - y_n)*np.log(1 - ho(X, o))).sum())
                   + (reg/(2*m)) * o.dot(o)
                  )

In [10]: def gradiente(o):
             return (1/m*(((ho(X, o) - y_n)*X.transpose()).transpose().sum(axis=0))
                    + (reg/m) * o[1:].sum()
                   )

In [11]: # Método iterativo de optimización de Theta mediante
         #las funciones de coste y gradiente
         import scipy.optimize as opt
         y_n = y_0
         (final_o, nfval, rc) = opt.fmin_tnc(func=coste, x0=o, fprime=gradiente)

In [12]: # Utilizando la Theta calculada con el método de optimización,
         #predecimos la probabilidad de que sea 0 cada imagen
         prediccion = ho(X, final_o)

In [13]: # Muestra el resultado de la predicción de una imagen en cuestión
         def resultado_prediccion(i):
             decision = 1 if prediccion[i] > 0.5 else 0
             return '{} [{}]: [{}] {}'.format(y[i], y_0[i], decision, prediccion[i])

In [14]: fp = 0 # falsos positivos
         fn = 0 # falsos negtivos
         for i in range(0, len(y)):
             decision = 1 if prediccion[i] > 0.5 else 0
             #print(resultado_prediccion(i))
             if y_0[i] == 1 and y_0[i] != decision:
                 fn += 1
             if y_0[i] == 0 and y_0[i] != decision:
                 fp += 1

         print('F.N.: {}%'.format((100. * fn) / len(y)))
         print('F.P.: {}%'.format((100. * fp) / len(y)))
         print('errores: {}%'.format((100. * (fn+fp)) / len(y)))
```

```
F.N.: 1.44%
F.P.: 0.5%
errores: 1.94%
```

```
In [15]: import numpy as np
         import matplotlib.pyplot as plt
```

```python
sample = np.random.choice(X.shape[0] ,10)
plt.imshow(X[sample, :].reshape(-1, 20).T)
plt.axis('off')
plt.show()
for i in sample:
    print(resultado_prediccion(i))
```



```
[10] [1]: [1] 0.9999837596956113
[4] [0]: [0] 0.015012134616994225
[1] [0]: [0] 8.791374022560749e-05
[8] [0]: [0] 0.00013310697554062145
[5] [0]: [0] 0.1305430386664507
[8] [0]: [0] 1.6113922319555307e-06
[10] [1]: [1] 0.9835760724601739
[6] [0]: [0] 1.3883134873328833e-05
[8] [0]: [0] 6.902829706863012e-05
[7] [0]: [0] 0.00017293090945978346
```

In [16]:
```python
#realizamos un clasificador logístico para cada numero del 0-9
O = []
for n in range(1,11): # números del 1 al 10 (el 10 es el 0)
    y_n = np.array([1 if i == n else 0 for i in y])
    (final_o, nfval, rc) = opt.fmin_tnc(func=coste, x0=o, fprime=gradiente)
    O.append(final_o)

O = [O[-1]] + O[0:-1]
```

In [17]:
```python
predicciones = []
for o_n in O:
    predicciones.append(ho(X, o_n))
```

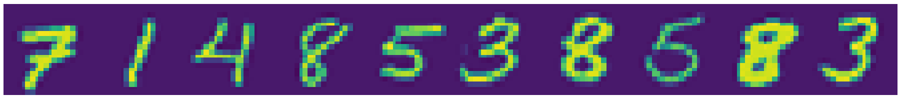
In [18]:
```python
def tomar_decision(i):
    decision = 0
    mejor_probabilidad = predicciones[0][i]
    for n in range(1, len(O)):
        if predicciones[n][i] > mejor_probabilidad:
            decision = n
            mejor_probabilidad = predicciones[n][i]
    return decision
```

```
In [19]: def resultado_prediccion_numero(i):
             decision = tomar_decision(i)
             return '{}: {} [{}] {}'.format(
                 y[i], y[i]==
                 (decision if decision != 0 else 10),
                 decision,
                 predicciones[decision][i]
                 )
```

```
In [20]: errores = 0
         for i in range(0, len(y)):
             decision = tomar_decision(i)
             #print(resultado_prediccion_numero(i))
             if y[i] != (decision if decision != 0 else 10):
                 errores += 1

         print('errores: {}%'.format((100. * errores) / len(y)))
```

```
errores: 22.54%
```

```
In [21]: import numpy as np
         import matplotlib.pyplot as plt
         sample = np.random.choice(X.shape[0] ,10)
         plt.imshow(X[sample, :].reshape(-1, 20).T)
         plt.axis('off')
         plt.show()
         for i in sample:
             print(resultado_prediccion_numero(i))
```



```
[7]: [ True] [7] 0.039225933842703745
[1]: [ True] [1] 0.9143262852707484
[4]: [ True] [4] 0.9697123446710744
[8]: [ True] [8] 0.9999802452465724
[5]: [ True] [5] 0.9999676653487249
[3]: [ True] [3] 0.6886871334482835
[8]: [ True] [8] 0.6311480146144287
[5]: [ True] [5] 0.9562037287644387
[8]: [ True] [8] 0.9926149750350398
[3]: [ True] [3] 0.9103984787446625
```

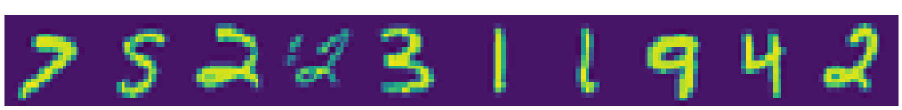
```
In [22]: ##################################################
         #REGRESIÓN LOGÍSTICA MULTICLASE CON REDES NEURONALES
         ##################################################
```

```
In [23]: from scipy.io import loadmat
         weights = loadmat('ex3weights.mat')
         theta1, theta2 = weights['Theta1'], weights['Theta2']

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

```
In [24]: def g(z):
             return 1/(1+np.exp(-z))
```

```
In [25]: def clasificarNumero(x, theta1, theta2):
             (nOcultas, nEntrada_) = theta1.shape
             (nSalida, nOcultas_) = theta2.shape
             entradas = np.insert(x, 0, 1.)
             ocultas = np.empty(nOcultas_)
             ocultas[0] = 1
             for o in range(nOcultas):
                 z = entradas.dot(theta1[o, :])
                 ocultas[o + 1] = g(z)
             salidas = np.empty(nSalida)
             for s in range(nSalida):
                 z = ocultas.dot(theta2[s, :])
                 salidas[s] = g(z)
             return salidas
```

```
In [26]: import numpy as np
         import matplotlib.pyplot as plt
         sample = np.random.choice(X.shape[0] ,10)
         plt.imshow(X[sample, :].reshape(-1, 20).T)
         plt.axis('off')
         plt.show()
         for i in sample:
             print('{}: {}'.format(
                 y[i],
                 np.argmax(clasificarNumero(X[i, :], theta1, theta2)) + 1
             ))
```



```
[7]: 7
[5]: 5
[2]: 2
[2]: 2
[3]: 3
[1]: 1
[1]: 1
[9]: 9
[4]: 4
[2]: 2
```

```
In [27]: (n, m) = X.shape
         nAciertos = 0
         nTotal = 0
         for i in range(n):
             nTotal += 1
             if y[i] == np.argmax(clasificarNumero(X[i, :], theta1, theta2)) + 1:
                 nAciertos += 1
         print(100*nAciertos/nTotal)
```

```
97.52
```

```
In [28]: def ho(X, theta1, theta2):
             _X = np.append(X, np.ones((n, 1)), axis=1)
             (n_hidden, n_input) = theta1.shape
             a_h = []
             for h in range(n_hidden):
                 a = g(_X.dot(theta1[h].transpose()))
                 a_h.append(a)
             a_h = np.concatenate([a_h], axis=1).transpose()
             a_h = np.append(a_h, np.ones((n, 1)), axis=1)
             (n_output, n_hidden) = theta2.shape
             a_o = []
             for o in range(n_output):
                 a_o.append(g(a_h.dot(theta2[o])))
             return np.array(a_o).transpose()
```

```
In [29]: def ho(X, theta1, theta2):
             resultado = []
             for i in range(n):
                 x = X[i, :]
                 (nOcultas, nEntrada_) = theta1.shape
                 (nSalida, nOcultas_) = theta2.shape
                 entradas = np.insert(x, 0, 1.)
                 ocultas = np.empty(nOcultas_)
                 ocultas[0] = 1
                 for o in range(nOcultas):
```

```
                z = entradas.dot(theta1[o, :])
                ocultas[o + 1] = g(z)
            salidas = np.empty(nSalida)
            for s in range(nSalida):
                z = ocultas.dot(theta2[s, :])
                salidas[s] = g(z)
            resultado.append(salidas)
        return np.array(resultado)
```

```
In [30]: prediccion = ho(X, theta1, theta2)
         (n, m) = X.shape
         nAciertos = 0
         nTotal = 0
         for i in range(n):
             nTotal += 1
             if y[i] == np.argmax(prediccion[i]) + 1:
                 nAciertos += 1
         print(100*nAciertos/nTotal)
```

```
97.52
```

```
In [31]: for i in sample:
             print('{}: {}'.format(
                 y[i],
                 np.argmax(prediccion[i]) + 1
             ))
```

```
[7]: 7
[5]: 5
[2]: 2
[2]: 2
[3]: 3
[1]: 1
[1]: 1
[9]: 9
[4]: 4
[2]: 2
```