Javier Cordero Calvo y Juan Ruiz Jiménez

Aprendizaje Automático y Minería de Datos – Práctica 3

Parte 1:

Para comenzar, cargaremos los datos con la función proporcionada para ello y los guardaremos en sus correspondientes matrices, así como estableceremos el valor de lambda para el futuro a 1.

```
datos = loadmat ("ex3data1.mat")
y = datos["y"]
X = datos["X"]
yaux = np.ravel(y)

landa = 1
```

Tras ello, llamamos a nuestras dos funciones para resolver el problema, calcAciertos() y oneVsAll(), con los parámetros necesarios y “parseado” a “string” para mostrar después los datos obtenidos.

```
result = str(calcAciertos(X, yaux, oneVsAll(X, yaux, 10, landa))) + "% de
 acierto"
```

A la función calcAciertos() hay que pasarle las matrices de X e Y, así como las de thetas la cual se calcula mediante la función oneVsAll().

```
def calcAciertos(X, Y, t):
    cont = 0
    aciertos = 0
    totales = len(Y)
    dimThetas = len(t)
    valores = np.zeros(dimThetas)

    for i in X:
        p = 0
        for x in range(dimThetas):
            valores[p] = sigmoide(np.dot(i, t[x]))
            p+=1

        r = np.argmax(valores)
        if r == 0:
            r = 10
        if(r==Y[cont]):
            aciertos+=1

        cont += 1

    porcentaje = aciertos / totales * 100
    return porcentaje
```

A su vez, la función oneVsAll()) necesita de otra función para apoyarse, calcOptTheta().

```
def oneVsAll(X, y, num_etiquetas, reg):

    ThetasMatriz = np.zeros((num_etiquetas, X.shape[1]))

    i = 0
    while i < num_etiquetas:
        if i == 0:
            aux = 10
        else: aux = i

        auxY = (y == aux).astype(int)
        ThetasMatriz[i, :] = calcOptTheta(auxY)

        i += 1

    return ThetasMatriz
```

La cual implementa también las funciones gradiante(), coste() y sigmoide().

```
def sigmoide(x):
    s = 1 / (1 + np.exp(-x))
    return s

def coste(theta, X, Y, landa):
    H = sigmoide(np.matmul(X, theta))
    m = len(X)
    cost = ((- 1 / m) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1
- H)))) + ((landa / (2 * m)) * (np.sum(np.power(theta, 2))))

    return cost

def gradiente(theta, XX, Y, landa):
    H = sigmoide(np.matmul(XX, theta))
    m=len(Y)
    grad = (1 / m) * np.matmul(XX.T, H - Y)

    aux=np.r_[[0],theta[1:]]

    firstPart = grad+(landa*aux/m)
    thetaAux = theta
    thetaAux[0] = 0

    result = firstPart + (landa / m * thetaAux)
    return result
```

```
def calcOptTheta(Y):
    result = opt.fmin_tnc(func=coste, x0=np.zeros(X.shape[1]), fprime=gra
diente, args=(X, Y, landa))
    return result[0]
```

Finalmente, tras todas estas llamadas obtenemos que el valor total de 93.44% de aciertos de nuestro sistema, el cual se encuentra cerca del valor aproximado de 95% que se nos pedía para corroborar que el resultado era correcto.

```
def pinta_aleatorio(X):
    sample = np.random.choice(X.shape[0], 10)
    aux = X[sample, :].reshape(-1, 20)
    plt.imshow(aux.T)
    plt.axis("off")
```

Mención a la función pinta_aleatorio() la cual muestra por pantalla un ejemplo aleatorio de todas las muestras del ejercicio.

Parte 2:

En esta parte, para la carga de datos necesitamos algunos valores más, como lo son las thetas intermedias. Por lo que nuestra carga de datos cambia relativamente en comparación a la parte 1, añadiendo los pesos y las thetas a nuestra carga.

```
datos = loadmat ("ex3data1.mat")

#almacenamos los datos leídos en X e y
X = datos["X"]
Y = datos["y"]

yaux = np.ravel(Y)

weights = loadmat("ex3weights.mat")
theta1, theta2 = weights["Theta1"], weights["Theta2"]
```

Tras hacer esto, llamaremos a nuestra función propagación_hacia_delante() que nos calculará mediante las thetas y la entrada X del problema la salida óptima del sistema llamada “h”.

```
h = propagacion_hacia_delante(X, theta1, theta2)
```

```
def propagacion_hacia_delante(X, theta1, theta2):
    m = X.shape[0]
    a1 = np.hstack([np.ones([m, 1]), X])
    z2 = np.dot(a1, theta1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoide(z2)])
    z3 = np.dot(a2, theta2.T)
    h = sigmoide(z3)
    #return a1, z2, a2, z3, h
    return h
```

Acto seguido, utilizamos dicha “h” para calcular el número de aciertos que contiene nuestro sistema de aprendizaje. Para ello, llamaremos a la función calcAciertos().

```
result = calcAciertos(X, yaux, h)

def calcAciertos(X, Y, t):
    aciertos = 0
    totales = len(Y)
    dimThetas = len(t)

    for i in range(dimThetas):
        r = np.argmax(t[i]) + 1
        if(r==Y[i]):
            aciertos+=1

    porcentaje = aciertos / totales * 100
    return porcentaje
```

Esta función simplemente compara el índice de la theta máxima de cada una de las filas de thetas con los ejemplos de salida (Y). Si son iguales, contaremos un acierto.

Destacamos también, como en la parte 1, las funciones sigmoide() y pinta_aleatorio().

Tras realizar todas las operaciones, el sistema nos da una tasa de éxito del 97.52%, que es el resultado esperado por nuestro sistema.