Aprendizaje Automático y minería de datos: Práctica 3

Autores: Tatiana Duarte Balvís, Miguel Mur Cortés

1. Objetivo:

El objetivo de la práctica es aplicar un algoritmo de recocimiento de números escritos a mano. Este problema se enfrentará de dos maneras distintas. Primero, deberemos aplicar un algoritmo de regresión logística multi-clase. En segundo lugar, se aplicará una red neuronal para resolver el problema.

2. Regresión Logística Multi-clase:

El fichero ex3data1.mat contiene 5000 ejemplos de entrenamiento. Obtenemos los datos proporcionados y pintamos aleatoriamente 10 ejemplos.

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

    sample = np.random.choice(X.shape[0], 10)
    plt.figure()
    plt.imshow(X[sample, :].reshape(-1, 20).T)
    plt.axis("off")
    plt.savefig("fig1.png")
```

Con lo que se consigue esta figura:

A continuación, entrenamos un clasificador por regresión logística para cada una de las 10 clases del conjunto de datos.

```
def oneVsAll(X, y, n_labels, reg):
    Theta = np.zeros([n_labels, len(X[0])])

    for k in range(n_labels):
        aux = (y == k+1)*1
        Theta[k] = opt.fmin_tnc(func=coste_reg, x0=Theta[k],
fprime=gradiente_reg, args=(X, aux, reg), messages=0)[0]

    return Theta
```

Para calcular el gradiente, utilizamos estas funciones auxiliares:

```
def sigmoide(z): #g(z)
    return 1 / (1 + np.exp(-z))


def coste(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    op1 = np.dot(np.log(H), y)
    op2 = np.dot(np.log(1 - H), (1 - y))
    return -(op1 + op2) / len(x)


def coste_reg(theta, x, y, lamda):
    op1 = coste(theta, x, y)
    op2 = lamda * np.sum(theta**2) / (2*len(x))
    return op1 + op2


def gradiente(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    return np.dot((H - y), x) / len(y)


def gradiente_reg(theta, x, y, lamda):
    op1 = gradiente(theta, x, y)
    op2 = lamda * theta / len(y)

    aux = op1[0]
    result = op1 + op2
    result[0] = aux
    return result
```

Una vez entrenado el clasificador, lo utilizamos para hacer predicciones y comprobar cuántos de los ejemplos de entrenamiento clasifica correctamente.

```
def evaluacion_regresion(X, y, Theta):
    resultado = np.empty(len(X))

    for k in range(len(X)):
        H = sigmoide(np.dot(X[k], Theta.T))
        resultado[k] = np.argmax(H) + 1

    return np.mean(resultado == y)


evaluacion = evaluacion_regresion(X_aux, y, Theta)
```

```
    print("Evaluación de la regresión logística multiclase: " +
str(evaluacion*100)[:5] + "%")
```

Evaluación de la regresión logística multiclase: 96.46%

3. Redes neuronales:

El objetivo de esta parte de la práctica es utilizar los pesos proporcionados para una red neuronal ya entrenada sobre los ejemplos para evaluar su precisión sobre esos mismos ejemplos.

El fichero ex3weights.mat contiene las matrices **Θ**(1) y **Θ**(2) con el resultado de haber entrenado la red neuronal.

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

Una vez obtenidos los datos, implementamos una propagación hacia delante para computar el valor de $h\theta(x^{(i)})$ de cada ejemplo i.

```
# Capas
a1 = X
z2 = np.dot(Theta1, a1.T)
a2 = sigmoide(z2)
a2 = np.vstack([np.ones(len(a2[0])), a2])
z3 = np.dot(Theta2, a2)
a3 = sigmoide(z3)

resultado = np.empty(len(X))
for k in range(len(X)):
    resultado[k] = np.argmax(a3[:,k]) + 1

return np.mean(resultado == y)
```

Con estos datos, determinamos la precisión de la red neuronal.

```
evaluacion_neuronal = red_neuronal(X_aux, y)
print("Evaluación de la red neuronal: " + str(evaluacion_neuronal*100)[:5] +
"%")
```

Evaluación de la red neuronal: 97.52%

4. Código

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


def sigmoide(z): #g(z)
    return 1 / (1 + np.exp(-z))


def coste(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    op1 = np.dot(np.log(H), y)
    op2 = np.dot(np.log(1 - H), (1 - y))
    return -(op1 + op2) / len(x)


def coste_reg(theta, x, y, lamda):
    op1 = coste(theta, x, y)
    op2 = lamda * np.sum(theta**2) / (2*len(x))
    return op1 + op2


def gradiente(theta, x, y):
    H = sigmoide(np.dot(x, theta))
    return np.dot((H - y), x) / len(y)


def gradiente_reg(theta, x, y, lamda):
    op1 = gradiente(theta, x, y)
    op2 = lamda * theta / len(y)

    aux = op1[0]
    result = op1 + op2
    result[0] = aux
    return result


def oneVsAll(X, y, n_labels, reg):
    Theta = np.zeros([n_labels, len(X[0])])

    for k in range(n_labels):
        aux = (y == k+1)*1
        Theta[k] = opt.fmin_tnc(func=coste_reg, x0=Theta[k],
fprime=gradiente_reg, args=(X, aux, reg), messages=0)[0]

    return Theta
```

```python
def evaluacion_regresion(X, y, Theta):
    resultado = np.empty(len(X))

    for k in range(len(X)):
        H = sigmoide(np.dot(X[k], Theta.T))
        resultado[k] = np.argmax(H) + 1

    return np.mean(resultado == y)


def red_neuronal(X, y):
    weights = loadmat("ex3weights.mat")
    Theta1 = weights["Theta1"]
    Theta2 = weights["Theta2"]

    # Capas
    a1 = X
    z2 = np.dot(Theta1, a1.T)
    a2 = sigmoide(z2)
    a2 = np.vstack([np.ones(len(a2[0])), a2])
    z3 = np.dot(Theta2, a2)
    a3 = sigmoide(z3)

    resultado = np.empty(len(X))
    for k in range(len(X)):
        resultado[k] = np.argmax(a3[:,k]) + 1

    return np.mean(resultado == y)


def regresion_logistica_multiclase():
    datos = loadmat("ex3data1.mat")
    X = datos["X"]
    y = datos["y"]
    y = np.ravel(y)

    sample = np.random.choice(X.shape[0], 10)
    plt.figure()
    plt.imshow(X[sample, :].reshape(-1, 20).T)
    plt.axis("off")
    plt.savefig("fig1.png")

    n_labels = 10
    reg = 0.1
    X_aux = np.hstack([np.ones([np.shape(X)[0], 1]), X])
    Theta = oneVsAll(X_aux, y, n_labels, reg)
    evaluacion = evaluacion_regresion(X_aux, y, Theta)
```

```
    print("Evaluación de la regresión logística multiclase: " +
str(evaluacion*100)[:5] + "%")

    evaluacion_neuronal = red_neuronal(X_aux, y)
    print("Evaluación de la red neuronal: " +
str(evaluacion_neuronal*100)[:5] + "%")
```