Práctica 3: Regresión logística multi-clase y redes neuronales

Material proporcionado:

Fichero	Explicación
`ex3data1.mat`	Datos de entrenamiento con imágenes de números escritos a mano.
`ex3weights.mat`	Pesos iniciales para el ejercicio con redes neuronales.

1. Regresión logística multi-clase

El objetivo de esta primera parte de la práctica es aplicar regresión logística multi-clase al reconocimiento de imágenes que representan números escritos a mano.

1.1. Visualización de los datos

El fichero `ex3data1.mat` contiene 5000 ejemplos de entrenamiento en el formato nativo para matrices de Octave/Matlab[1]. El fichero se carga con la función `scipy.io.loadmat` que devuelve un diccionario del que podemos extraer las matrices X e y:

```
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
```

Cada ejemplo de entrenamiento es una imagen de 20×20 píxeles donde cada píxel está representado por un número real que indica la intensidad en escala de grises de ese punto. Cada matriz de 20×20 se ha desplegado para formar un vector de 400 componentes que ocupa una fila de la matriz X. De esta forma, X es una matriz de 5000×400 donde cada fila representa la imagen de un número escrito a mano:

[1] Una parte del conjunto de datos `http://yann.lecun.com/exdb/mnist/`

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix}$$

El vector y es un vector de 5000 componentes que representan las etiquetas de los ejemplos de entrenamiento. El "0" se ha etiquetado como "10", manteniendo las etiquetas naturales del "1" al "9" para el resto de los números.

Una vez cargados los datos, puedes ejecutar el siguiente código para visualizar una selección aleatoria de 10 ejemplos de entrenamiento

```
# Selecciona aleatoriamente 10 ejemplos y los pinta

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

para obtener una imagen similar a esta:

1.2. Vectorización de la regresión logística

Para aplicar regresión logística al reconocimiento de dígitos tendrás que entrenar 10 clasificadores logísticos diferentes, uno para cada posible valor del dígito a reconocer. Para que este proceso sea eficiente es importante utilizar una versión vectorizada porque de lo contrario el proceso será extremadamente lento.

Vectorización de la función de coste

Recuerda que el valor de la función de coste (sin regularizar) en regresión logística viene dado por la expresión:

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

Para computar cada elemento del sumatorio, es necesario calcular $h_\theta(x^{(i)})$ para cada ejemplo i, donde $h_\theta(x^{(i)}) = g(\theta^T x^{(i)})$ y $g(z) = \frac{1}{1+e^{-z}}$. Utilizando multiplicación entre matrices, es posible calcular rápidamente ese término para todos los ejemplos de entrenamiento. Dados:

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \quad y \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

el producto $X\theta$ resulta en

$$X\theta = \begin{bmatrix} (x^{(1)})^T\theta \\ (x^{(2)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} = \begin{bmatrix} \theta^T(x^{(1)}) \\ \theta^T(x^{(2)}) \\ \vdots \\ \theta^T(x^{(m)}) \end{bmatrix}$$

a partir de lo cual es sencillo obtener el valor de la función de coste sin necesidad de bucles, ayudados de la multiplicación elemento a elemento y la función sumatorio (`sum`).

Vectorización del gradiente

Recuerda que el gradiente de la función de coste (sin regularizar) es un vector de la misma longitud que θ donde la componente j (para $j = 0, 1, \ldots, n$) viene dada por la expresión:

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

Para vectorizar esta operación escribimos el gradiente en forma de vector:

$$\begin{aligned} \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} &= \frac{1}{m}\begin{bmatrix} \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_n^{(i)} \end{bmatrix} \\ &= \frac{1}{m}\sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x^{(i)}\right) \\ &= \frac{1}{m}X^T(h_\theta(x) - y) \end{aligned}$$

donde:

$$h_\theta(x) - y = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}$$

Donde $x^{(i)}$ es un vector mientras que $(h_\theta(x^{(i)}) - y^{(i)})$ es un número.

De esta forma, es posible obtener el vector gradiente sin necesidad de realizar ningún bucle. Si tienes una versión previa que utiliza bucles, será buena idea que compruebes que genera el mismo resultado que la versión vectorizada.

Vectorización de la versión regularizada

Por último, añade el término de regularización a tu versión vectorizada de la función de coste y el gradiente de la regresión logística. Recuerda que la función de coste se define como:

$$J(\theta) = \left[\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right]\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

y el gradiente de la función de coste es un vector de la misma longitud que θ donde la componente j se calcula como:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_0} &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} && \text{para } j = 0 \\ \frac{\partial J(\theta)}{\partial \theta_j} &= \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j && \text{para } j \geq 1 \end{aligned}$$

1.3. Clasificación de uno frente a todos

A continuación debes entrenar un clasificador por regresión logística para cada una de las 10 clases del conjunto de datos. Debes implementarlo como una función que devuelva una matriz $\Theta \in \mathbb{R}^{K\times(N+1)}$ donde cada fila de Θ corresponde a los parámetros aprendidos para el clasificador de una de las clases. Implementa para ello una función con esta cabecera:

```
def oneVsAll(X, y, num_etiquetas, reg):
    """
    oneVsAll entrena varios clasificadores por regresión logística con término
    de regularización 'reg' y devuelve el resultado en una matriz, donde
    la fila i-ésima corresponde al clasificador de la etiqueta i-ésima
    """
```

Recuerda que el argumento y es un vector con etiquetas de 1 a 10, donde el dígito "0" se ha hecho corresponder con la etiqueta 10. Por otra parte, cuando entrenes al clasificador para la clase $k \in \{1, \ldots, K\}$, tendrás que obtener un vector m-dimensional de etiquetas y donde $y_j \in 0, 1$ indica si el ejemplo de entrenamiento j-ésimo pertenece a la clase k ($y_j = 1$) o a otra clase ($y_j = 0$). Para ello, te será útil saber que en Python `True * 1` es igual a 1 y `False * 1` es igual a 0.

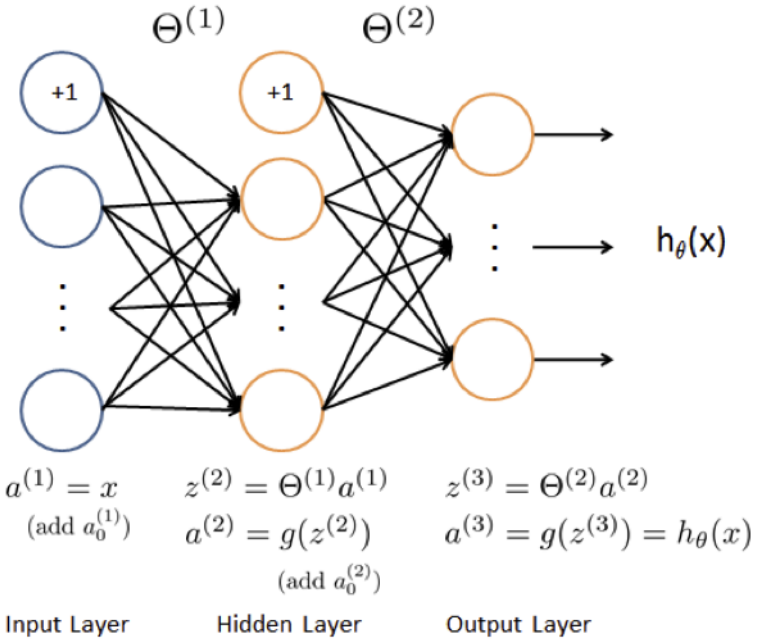
Para el entrenamiento puedes utilizar la función `scipy.optimize.fmin_tnc`

Por último, una vez que hayas entrenado al clasificador, podrás utilizarlo para hacer predicciones y comprobar cuántos de los ejemplos de entrenamiento clasifica correctamente. Para ello, debes calcular para cada ejemplo de entrenamiento cuál es la "probabilidad" de que pertenezca a cada una de las clases, asignándole la etiqueta (1, 2, ..., o K) para la que se obtenga el valor máximo. Para un valor del término de regularización de 0,1 el resultado debería estar en torno al 95 %.

2. 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.

La red neuronal tiene la siguiente estructura:



formada por tres capas, con 400 unidades en la primera capa (además de la primera fijada siempre a +1), 25 en la capa oculta y 10 en la capa de salida.

El fichero `ex3weights.mat` contiene las matrices $\Theta^{(1)}$ y $\Theta^{(2)}$ con el resultado de haber entrenado la red neuronal y que cargaremos con la función `scipy.io.loadmat`

```
weights = loadmat('ex3weights.mat')
theta1, theta2 = weights['Theta1'], weights['Theta2']
# Theta1 es de dimensión 25 x 401
# Theta2 es de dimensión 10 x 26
```

Debes implementar la propagación hacia adelante para computar el valor de $h_\theta(x^{(i)})$ para cada ejemplo i. De la misma forma que en la regresión logística, interpretaremos que la clase asignada por la red neuronal a un ejemplo es la correspondiente a la salida de la red con el máximo valor (te puede resultar útil la función `argmax` de numpy). Deberías obtener que la precisión de la red neuronal está en torno al 97.5 %.

3. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicadas en la cabecera de la práctica.

Se entregará un único fichero en formato pdf que contenga la memoria de la práctica, incluyendo el código desarrollado y los comentarios y gráficas que se estimen más adecuados para explicar los resultados obtenidos.