Práctica 3

Pablo Arranz Ropero
Juan Alberto Camino Sáez
Grupo 2

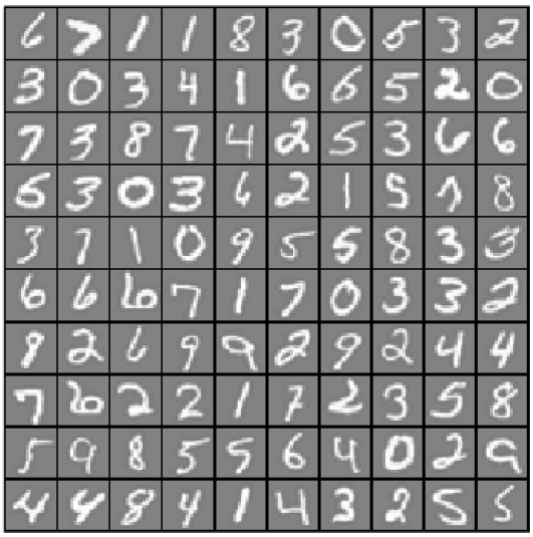
Práctica 3: Regresión logística multi-clase y redes neuronales

En esta práctica, que se divide en dos apartados, se trata de aplicar regresión logística multi-clase y una red neuronal a un conjunto de datos consistente en 5000 imágenes de 20x20 píxeles (que serán nuestros "atributos" donde cada píxel está representado por un número que representa la intensidad de ese píxel en escala de grises. Los datos están en el fichero ex3data1.mat que cargan directamente los datos en X e y, guardándose en y el número al que pertenece ese mapa de píxeles (del 0 al 9, representando el 0 como un 10).

Regresión logística multi-clase

Primeramente, hemos visualizado 100 datos aleatorios en pantalla con el siguiente código.

```
1. m = size(X, 1);
2.
3. rand_indices = randperm(m);
4. sel = X(rand_indices(1:100), :);
5.
6. displayData(sel);
```



Para el cálculo de las hipótesis, hemos utilizado la función sigmoide, aunque hemos realizado una mejora sobre la función sigmoide de la anterior práctica para realizar un cálculo vectorizado sin bucles.

```
1. function [sigmoide] = sigmoide(z)
2.
3.     sigmoide = 1./(1+(e.^-z));
4.
5. endfunction
```

Crearemos un fichero llamado *lrCostFunction.m* que se encargará de calcular la función de coste y el valor del gradiente para unas determinadas thetas y un determinado valor de regularización lambda. Esta función es la que pasaremos a la función *fmincg* para que realice el descenso de gradiente. Utilizaremos esta función en lugar de *fminunc* porque es más apropiada para un número tan grande de atributos.

```
1. function[J, grad] = lrCostFunction(theta, X, y, lambda)
2.
3.   valhipotesis = sigmoide(X*theta);
4.   m = rows(X);
5.
6.   valory0 = (-y' * log(valhipotesis));
7.   valory1 = ((1 - y)' * log(1-valhipotesis));
8.
9.   Jsinreg = (1/m)*(valory0-valory1);
10.
11.        regularizacion
   = (lambda/(2*m)) * sum(theta(2:rows(theta)).^2);
12.
13.        J = Jsinreg + regularizacion;
14.
15.        grad = ((1/m) * X' * (valhipotesis - y));
16.
17.        regularizacion = ((lambda/m) *
   theta(2:rows(theta),:));
18.
19.        grad(2:rows(theta),1) = grad(2:rows(theta),1) +
   regularizacion;
20.
21.      endfunction
```

Con 0s como thetas iniciales, la función de coste tendrá un valor de 0.69315.

*Aclaración:

Debemos añadir X_0 a nuestros datos con las siguientes líneas de código:

```
1. X(:,2:columns(X)+1) = X;
2. X(:,1) = ones(rows(X),1);
```

Para hacer la clasificación multi-clase usaremos el método one-vs-all que para cada posible clase nos dirá la probabilidad de que el elemento de entrada pertenezca o no a esa clase. Por lo tanto, existirá un vector de thetas para cada clase, es decir, que theta será una matriz $\Theta \epsilon \Re^{K \times (N+1)}$.

Para el cálculo de las thetas crearemos una función llamada *oneVsAll* con el siguiente código:

```
1. function [all_theta] = oneVsAll(X, y, num_etiquetas, lambda)
2.
3.   initial_theta = zeros(columns(X),1);
4.   options = optimset('GradObj', 'on', 'MaxIter', 50);
5.
6.   for i = 1:num_etiquetas
```

```
7.
8.      all_theta(i,:) = fmincg(@(t)(lrCostFunction(t, X, (y
   == i), lambda)), initial_theta, options);
9.
10.        endfor
11.
12.      endfunction
```

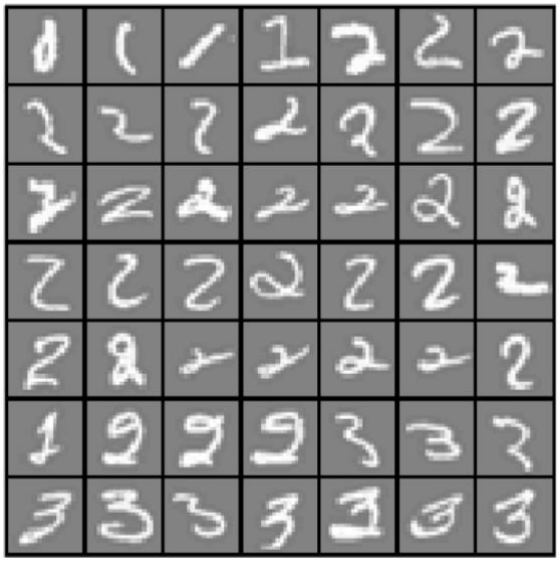
Con un lambda de 0.1, para los dos primeros ejemplos de la imagen en la página 1, cuyos índices son el 3145 y el 3675 obtenemos que son el 6 y el 7 con un 99.8% y un 99.9% de probabilidades respectivamente. Para obtener estas probabilidades hemos creado esta función que con una entrada y las thetas devuelvas por oneVsAll devuelve la clase a la que tiene más probabilidades de pertenecer y la probabilidad.

```
1. function [class, probs] = guessClass(X, theta)
2.
3.      [probs, class] = max(sigmoide(X*theta'));
4.
5. endfunction
```

También hemos creado una función que aplica la clasificación a todos los ejemplos de entrenamiento para comprobar cuantos clasifica incorrectamente:

```
1. function [num_wrongs] = checkTrainingCases(X, theta, y)
2.      num_wrongs = 0;
3.      for i = 1:rows(X)
4.          [probs, class] = max(sigmoide(X(i,:)*theta'));
5.          if (class == y(i))
6.              printf("class: %d , probs: %f \t &#10003; \n",
   class, probs);
7.          else
8.              printf("class: %d , probs: %f , should be: %d
   (index: %d) \n", class, probs, y(i), i);
9.              num_wrongs += 1;
10.               endif
11.           endfor
12.       endfunction
```

En nuestro caso, clasifica incorrectamente 244 elementos, entre muchos otros, los índices:
143, 562, 676, 766, 1009, 1026, 1027, 1046, 1058, 1063, 1082, 1088, 1091, 1098, 1104, 1113, 1131, 1146, 1154, 1188, 1190, 1208, 1214, 1232, 1293, 1312, 1363, 1364, 1384, 1400, 1409, 1411, 1413, 1414, 1424, 1441, 1485, 1489, 1499, 1525, 1528, 1551, 1557, 1565, 1589, 1594, 1600, 1603, 1608
Que si los pasamos a la función displayData tienen la siguiente forma:



Nuestro algoritmo de regresión logística tiene una precisión de
(5000 – 244) / 5000 ≈ 95%

Redes neuronales

Para probar la eficacia de las redes neuronales hemos creado una función que realiza el forward propagation:

```
1. function [probs, class] = forwardprop(X, Theta1, Theta2)
2.   #Suponemos que viene añadida la bias unit
3.   z2 = Theta1 * X';
4.   a2 = sigmoide(z2);
5.
6.   #añadimos bias unit
7.   a2(2:rows(a2)+1,:) = a2;
8.   a2(1,:) = ones(columns(a2),1);
9.
10.        z3 = Theta2 * a2;
11.        h = sigmoide(z3);
12.
13.        [probs, class] = max(h);
14.
15.     endfunction
```

Esto nos devuelve un vector de m columnas con las probabilidades de cada caso i de ser lo predicho en la columna i del vector class.

Para comprobar el ajuste a los casos de entrenamiento hemos modificado la función checkTrainingCases usada en el apartado anterior:

```
1. function [num_wrongs] = checkTrainingCasesNN(X, Theta1,
   Theta2, y)
2.     num_wrongs = 0;
3.     [probs, class] = forwardprop(X, Theta1, Theta2);
4.     for i = 1:rows(X)
5.         if (class(1,i) == y(i))
6.             printf("class: %d , probs: %f \t &#10003; \n",
   class(1, i), probs(1, i));
7.         else
8.             printf("class: %d , probs: %f , should be: %d
   (index: %d) \n", class(1, i), probs(1, i), y(i), i);
9.             num_wrongs += 1;
10.              endif
11.          endfor
12.      endfunction
```

En nuestro caso, clasifica incorrectamente 124 elementos, es decir, que tiene una precisión de
(5000 – 124) / 5000 ≈ 97.5%