Regresión Logística Multi-Clase

Práctica 4

César Borja

21/03/2022

Aprendizaje Automático

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

1. Objetivo

El objetivo de la práctica es resolver mediante regresión logística multiclase el reconocimiento de dígitos manuscritos. Se utilizará una versión reducida del conjunto de datos MNIST.

2. Estudio previo

```
options = [];
options.display = 'final';
options.method = 'newton';
lambda = logspace(-6,2);

best_model = 0;
best_errV = inf;

for  model = 1:length(lambda)
    etr = 0; ecv = 0;
    theta = zeros(N_pixels+1,1);
    for  i = 1:10
        y_clasif = (ytr == i);  % obtener salidas binarias para cada clasificador
        th = minFunc(@costeLogReg, zeros(N_pixels+1,1), options, Xtr, y_clasif,
            lambda(model);
        theta = [theta th] % concatenar columna de thetas de clasificador binario i.
    end
    ytr_pred = pred_sigmoid(Xtr, theta);
    etr = tasa_error(ytr_pred, ytr);
    ycv_pred = pred_sigmoid(Xcv, theta);
    ecv = tasa_error(ycv_pred, ycv);
    if ecv < best_errV
       best_errV = ecv;
       best_model = model;
end  end
```

Figura 1: Estudio previo. Algoritmo de entrenamiento y clasificación multiclase utilizando regresión logística regularizada.

Se ha utilizado el método One-vs-all para desarrollar el algoritmo. Este método consiste en entrenar un clasificador binario para cada clase (diez, uno para cada dígito) que estime la probabilidad de pertenencia a dicha clase.

Para hacer esto, se programa un bucle en el que se entrenará un clasificador por iteración. Dentro de cada iteración se convierte el vector de salidas de los datos de entrenamiento en un vector de salidas, que dice si pertenece (1) o no (0) a la clase i. A continuación se entrena el clasificador con la función minFunc y el parámetro de regularización *lambda*. Para poder elegir el mejor valor de *lambda* se repite este bucle para todos los valores distintos que se quieran probar.

Nótese que en las opciones de la función de entrenamiento minFunc se ha utilizado el método "pase de newton", sin embargo, dada la magnitud de los atributos de los datos de entrada (400 niveles de intensidad de píxeles por imagen), el rendimiento utilizando este método es muy bajo ya que requiere el cálculo del Hessiano. Por esto, en la versión implementada en MATLAB, se ha utilizado el método "lbfgs", más ligero.

El código del algoritmo utilizado se encuentra en el Anexo I.

3. Regresión logística regularizada

Se pide resolver la regresión logística regularizada utilizando el algoritmo escrito en la sección 2, separando un 20 % de los datos para validación. También se pide dibujar la gráfica de tasa de errores en función del parámetro *lambda*.

Tras resolver la regresión, se obtiene que el mejor factor de regularización es $lambda = 8.6851e\text{-}04$. En la Figura 2 se muestra la evolución de las tasas de errores en función del factor de regularización. Se puede ver, como en toda regularización, que para valores muy bajos de lambda se produce sobreajuste (modelo demasiado complejo) y para valores muy grandes se produce subajuste (modelo demasiado simple).

4. Matriz de confusión y Precisión/Recall

Con el mejor modelo obtenido en el apartado anterior ($lambda = 8.6851e\text{-}04$) se re-entrena con todos los datos (sin datos de validación) y se calcula

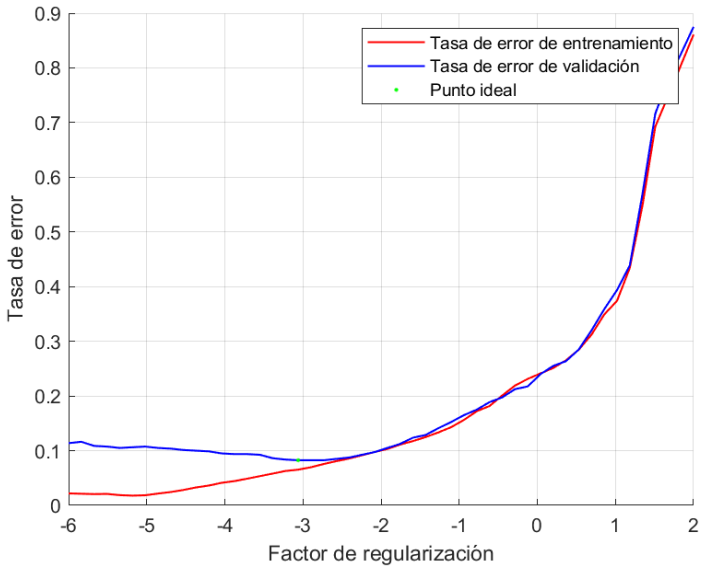

Figura 2: Evolución de las tasas de error con datos de entrenamiento y datos de validación

la matriz de confusión utilizando los datos de test.

La matriz de confusión se muestra en la Figura 3. Las filas representan las clases predichas mientras que en las columnas se representan las clases reales. Además, se pide calcular los valores de precisión y recall para cada dígito. Dichos valores se pueden ver en el Cuadro 1.

Dígito	Precisión	Recall
1	0.8899	0.9700
2	0.8763	0.8500
3	0.8485	0.8400
4	0.8900	0.8900
5	0.8673	0.8500
6	0.9159	0.9800
7	0.9293	0.9200
8	0.8367	0.8200
9	0.9053	0.8600
0	0.9796	0.9600

Cuadro 1: Precisión y recall para cada dígito utilizando los datos de test

$$\begin{pmatrix} 97 & 2 & 1 & 1 & 1 & 0 & 2 & 4 & 1 & 0 \\ 1 & 85 & 5 & 3 & 0 & 0 & 2 & 1 & 0 & 0 \\ 0 & 3 & 84 & 0 & 6 & 0 & 0 & 3 & 3 & 0 \\ 0 & 1 & 0 & 89 & 2 & 0 & 2 & 2 & 3 & 1 \\ 0 & 1 & 6 & 0 & 85 & 0 & 0 & 3 & 1 & 2 \\ 0 & 2 & 0 & 2 & 2 & 98 & 0 & 2 & 0 & 1 \\ 0 & 2 & 1 & 0 & 0 & 0 & 92 & 0 & 4 & 0 \\ 2 & 3 & 2 & 2 & 4 & 1 & 0 & 82 & 2 & 0 \\ 0 & 0 & 0 & 3 & 0 & 1 & 2 & 3 & 86 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 96 \end{pmatrix}$$

Figura 3: Matriz de confusión obtenida utilizando los datos de test para el conjunto de datos MNIST

Por último, en la Figura 4 se visualizan algunas de las confusiones obtenidas con la función verConfusiones() administrada como material de la práctica.

¿Qué dígitos son los más problemáticos?

En el cuadro 1 el '8' destaca como el dígito con menor precisión y recall, seguido por el '3' y el '5'. En concreto estos dos últimos se confunden con frecuencia entre sí (ver Figura 3).

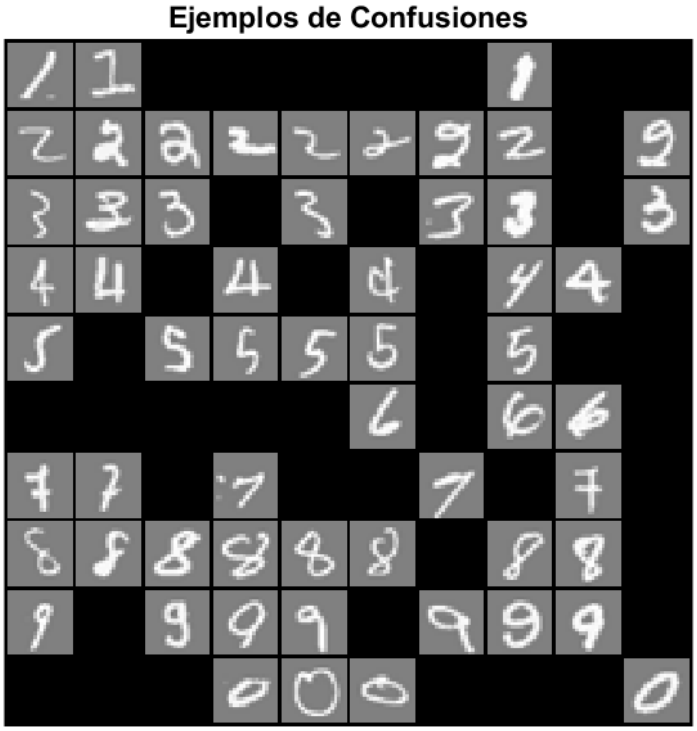

Figura 4: Ejemplos de las confusiones habidas

Anexo I. Algoritmo de entrenamiento

```
Xtr = [ones(height(ytr),1) Xtr];
Xcv = [ones(height(ycv),1) Xcv];

options = [];
options.display = 'final';
options.method = 'lbfgs';
lambda = logspace(-6, 2);

Etr = [];
Ecv = [];

best_model = 0;
best_errV = inf;

for model = 1:length(lambda)
    etr = 0; ecv = 0;
    theta = zeros(N_pixels+1,1);
    for i = 1:10
        y_clasif = (ytr == i);
        th = minFunc(@costeLogReg, zeros(N_pixels+1,1), options, Xtr, ...
            y_clasif, lambda(model));
        theta = [theta th];
    end

    ytr_pred = pred_sigmoid(Xtr,theta);
    etr = tasa_error(ytr_pred,ytr);

    ycv_pred = pred_sigmoid(Xcv,theta);
    ecv = tasa_error(ycv_pred,ycv);

    Etr = [Etr;etr];
    Ecv = [Ecv;ecv];

    if ecv < best_errV
        best_errV = ecv;
        best_model = model;
    end
end
```