Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Aprendizaje automático

Practica 4: Regresión logística Multi-Case

Autora:
Nerea Gallego Sánchez (801950)

7 de diciembre de 2022

Índice

1. Introducción y objetivos

Esta práctica consiste en aplicar regresión logística regularizada en un problema real de clasificación multi-clase: el reconocimiento de dígitos manuscritos.
Se utiliza como datos de entrada una versión reducida del conjunto de datos MNIST.
Cada muestra es una imagen de 20x20 píxeles. Como atributos para la clasificación se utilizan directamente los niveles de intensidad de los 400 píxeles.

Se van a utilizar técnicas de regresión logística básica para predecir a qué número corresponde la imagen.
Se utilizan también técnicas de regularización para determinar una mejor predicción.
Por último se calcula la matriz de confusión y tanto el valor de predicción como el de recall para cada una de las clases.

2. Estudio previo

```
y = {10, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Se elige best-of-all

function t = entrenaRegularizacionMulticase(X, y, λ)
    options = []
    options.display = 'final'
    options.methods = 'newton'
    theta_ini = zeros(size(X,2),1)
    t = []
    for i = 1:10
        t = [t minFunc(@CosteLogReg, theta_ini, options, X, (y==i), lambda)
    end
end
```

```
function err = errorRegularizacionMulticase(X, y, t, λ)
    ypred = 1./(1+exp(-(X*t)))
    [M, I] = max(ypred, [], 2)
    err = sum(y ~= I) / size(y,1)
end
```

Figura 1: Estudio previo a la práctica.

3. Regresión logística regularizada

Se pide encontrar el mejor parámetro de lambda para la regularización de manera que se obtenga una buena predicción de los números del conjunto de datos MINST.

Para realizar este ejercicio, se han separado el 20 % de los datos de entrenamiento para realizar la validación con cada uno de los parámetros que se van a probar.
Se ha creado la función separar que dado un conjunto de datos, el porcentaje de datos que se quieren utilizar para entrenamiento, las columnas en las que se encuentran los atributos y las columnas en las que se encuentra la salida, devuelve una matriz con el conjunto de datos que se van a utilizar de entrenamiento, con sus respectivas salidas y los datos que se van a utilizar como test, con sus respectivas salidas.
En este caso se utilizarán el 80 % de los datos para entrenar el modelo y el 20 % restante para realizar la validación.
La función creada para separar los datos es:

```
% Dado el conjunto de datos completo, el porcentaje de datos que se quieren
% usar como datos de entrenamiento, las columnas de los atributos y las
% columnas de las salidas esperadas, devuelve el conjunto de datos de
% entrenamiento con sus salidas esperadas y el conjunto de datos de
% validacion con sus salidas esperadas.
function [Xtr, Ytr, Xtst, Ytst] = separar(data, porc, col_x, col_y)
    % Realiza una permutacion de los datos para que se escojan de manera
    % aleatoria.
    randOrd = randperm(size(data,1));
    perm = data( randOrd,:);
    % Establece el limite entre los datos de entrenamiento y los de
    % validacion.
    lim = int32(size(data,1)*porc);
    Xtr = perm((1:lim),col_x);
    Ytr = perm(1:lim,col_y);
    Xtst = perm(lim+1:end,col_x);
    Ytst = perm(lim+1:end, col_y);
end
```

Una vez separados los conjuntos de datos que se van a utilizar, se ha usado la función *logspace* para crear un vector con los posibles valores de lambda. En este caso se han utilizado valores desde 10^{-6} hasta 10.

A continuación, se ha utilizado una sección de código, que realiza diversos experimentos de manera que para cada modelo de regularización (cada posible valor que puede adquirir el parámetro λ), se calculan los errores obtenidos, tanto con los datos de entrenamiento como con los datos separados para la validación.

Para obtener el vector de pesos se han creado una función entrenaRegularizacionMulticlase que dados los datos de entrada, las salidas esperadas, el factor de regularización y la cantidad de clases que hay en los datos, devuelve el vector de pesos entrenando con la función minFunc.

```
% Dado el conjunto de datos de entrada, las salidas esperadas, el parametro
% de regularizacion y la cantidad de clases, devuelve la matriz de pesos
% para predecir cada una de las clases.
function t = entrenaRegularizacionMulticase(X,y,lambda,clases)
    % Establece los parametros de la funcion minFunc
    options = [];
    options.display = 'final';
    options.methos = 'lbfgs';
    % Establece el vector inicial de pesos
    theta_ini = zeros(size(X,2),1);
    t = [];
    % para cada una de las clases, a ade a la matriz de pesos, los pesos
    % obtenidos para la clase i
    for i = 1 : clases
        t = [t minFunc(@CosteLogReg, theta_ini, options, X, (y == i), lambda)];
    end
end
```

La función establece los parámetros de la función *minFunc* para calcular el vector de pesos. En este caso se ha utilizado el método 'lbfgs' en lugar del método de 'newton' ya que el segundo era mucho más lento.

Se crea un vector de pesos inicial, con valor de 0's para todos los atributos.
Para cada una de las clases, se obtiene una columna en la matriz de los pesos, siendo la columna i de la matriz de pesos, el vector de pesos calculado para predecir dicha clase.
La función devuelve la matriz de pesos calculada para predecir todas las posibles clases.

Se ha creado una función que devuelve el error obtenido dada la entrada, la salida esperada y la matriz de pesos.
La función errorRegularizacionMulticase devuelve el error medio obtenido con esa matriz de pesos:

```
% Dada la entrada, la salida esperada y la matriz de pesos, devuelve la
% media de casos erroneos que se obtienen.
function err = errorRegularizacionMulticase(X,y,t)
    % Calcula la prediccion obtenida con la matriz de pesos
    ypred = prediccionMulticlase(X,y,t);
    % Se suma la cantidad de filas que son distntas de las que hay en el
    % vector de salidas esperadas, y se calcula la media de error.
    err = sum(y ~= ypred) / size(y,1);
end
```

La función calcula la predicción del modelo mediante la función prediccionMulticlase y devuelve el error medio calculando el porcentaje de fallos que produce el modelo, por el total de muestras que se examinan.
La función prediccionMulticlase devuelve en un vector cuáles son las salidas predichas.

```
% Dada la entrada y la matriz de pesos, devuelve la salida predicha.
function ypred = prediccionMulticlase(X,t)
    % Calcula la matriz de prediccion para cada clase mediante la funcion
    % sigmoidal.
    yp = 1./(1+exp(-(X*t)));
    % La salida predicha es aquella que ha obtenido una mayor probabilidad
    % en la salida predicha.
    % Se devuelve el indice de la columna que tiene el maximo valor.
    [M, ypred] = max(yp,[],2);
end
```

Esta función calcula la matriz de salidas predichas realizando la multiplicación de las entradas por la matriz de pesos. Para determinar cuál es la salida predicha, se busca aquella columna que tiene mayor probabilidad de ser la acertada. Es decir, la que obtiene un mayor valor en la salida predicha.
Finalmente devuelve el vector con las clases que se predicen.

Para calcular cuál es el mejor parámetro de regularización se ha utilizado una sección de código muy similar a la función de $k-fold\ cross-validation$ utilizada en las prácticas anteriores. En este caso se ha utilizado un único $fold$ de manera que solo se utiliza como pliegue de validación los datos ya separados anteriormente.

```
bestModel = 0;
bestErrV = inf;
errT = [];
errV = [];
% para cada valor de lambda
for model = 1:size(lambda,1)
    err_T = 0;
    err_V = 0;
    % se calcula la matriz de pesos
    th = entrenaRegularizacionMulticase(Xtr,Ytr,lambda(model,:),10);
    % se calculan los valores de error
    err_T = errorRegularizacionMulticase(Xtr,Ytr,th);
    err_V = errorRegularizacionMulticase(Xv,Yv,th);
    % se guardan los valores de error obtenidos para posteriormente
    % imprimir una grafica
    errT = [errT err_T];
    errV = [errV err_V];
    % si el modelo es mejor que el anterior, se guarda
    if err_V < bestErrV
        bestModel = model;
        bestErrV = err_V;
    end
end
```

Cuando finaliza de ejecutar esta sección de código se obtiene en el vector errT los errores obtenidos con los datos de entrenamiento en cada ejecución. De la misma manera, se guarda en errV los errores

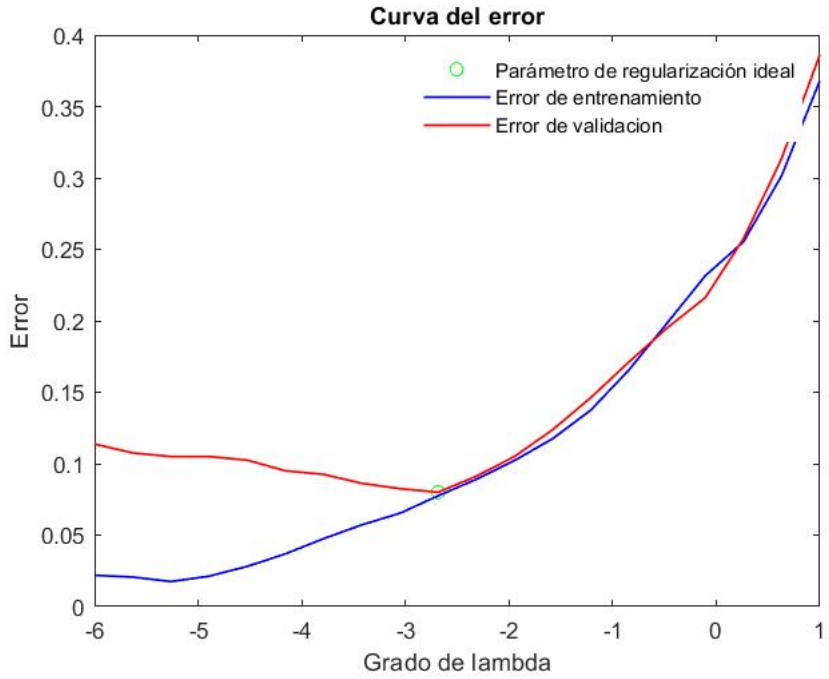

Figura 2: Curva de evolución del error en función de lambda.

obtenidos con los datos de validación en cada ejecución.
Se guarda en bestModel el ínidice del mejor modelo obtenido.

Como se puede observar en la gráfica 2 se obtiene como mejor parámetro de regularización $\lambda = 2{,}069138e-03$. En la gráfica también se muestra como, tener un parámetro de regularización más pequeño produce sobreajuste, ya que el error obtenido con los datos de entrenamiento es muy pequeño, pero el error obtenido con los datos de validación es mucho mayor. Sin embargo, si se aumenta mucho el valor del parámetro λ se obtiene subajuste, ya que se simplifica el modelo y ambos errores comienzan a ser muy grandes.

4. Matriz de confusión y Precisión/Recall

Se pide re-entrenar con todos los datos de entrenamiento para el mejor valor de lambda obtenido y utilizar los datos de test para calcular la matriz de confusión.
Para ello, se han cogido todos los datos de entrenamiento y se ha calculado con ayuda de la función entrenaRegularizacionMulticase una nueva matriz de pesos con el parámetro de regularización $\lambda = 2{,}069138e-03$.

Una vez obtenida la matriz de pesos se ha utilizado la funcion prediccionMulticlase para obtener las salidas predichas con los datos de test.

Para calcular la matriz de confusión se ha utilizado la función matrizConfusion de manera que le pasas la salida predicha, la salida esperada y el número de clases y devuelve la matriz de confusión. Contiene en las filas la clase real y el las columnas la clase predicha.

```
% Dada la salida predicha, la salida esperada y la cantidad de clases,
% devuelve la matriz de confusion de manera que contiene en las filas la
% clase real y en las columnas, la clase predicha.
function matrizConf = matrizConfusion(ypred,ytest,clases)
    matrizConf = zeros(clases,clases);
    for i = 1: clases
        for j = 1:clases
            matrizConf(i,j) = sum(ytest==i & ypred == j);
        end
    end
end
```

La función calcula en un doble bucle anidado, para cada clase esperada, cuántas veces se ha predicho un valor con cada una de las clases existentes.

Cuadro 1: Matriz de confusión.

		Clase predicha									
		1	2	3	4	5	6	7	8	9	0
Clase real	1	97	1	0	0	0	0	0	2	0	0
	2	3	83	3	2	1	2	2	3	0	1
	3	1	5	85	0	5	1	1	2	0	0
	4	1	3	0	88	0	2	0	2	4	0
	5	1	0	6	2	83	2	0	6	0	0
	6	0	0	0	0	0	99	0	0	1	0
	7	2	3	0	2	0	0	91	0	2	0
	8	4	1	3	2	3	2	0	82	3	0
	9	1	0	3	3	1	0	4	1	86	1
	0	0	0	0	1	2	3	0	0	0	94

Como se puede observar en la matriz de confusión, los números que generan mayor conflicto son el 8, seguidos de 2 y el 5.

También se pide calcular la precisión y el recall para cada una de las clases.
Se define como precisión a la fracción de positivos que hay realmente de todos los que fueron clasificados como positivos.
Se calcula de la siguiente manera: $Precision = \frac{TP}{(TP+FP)}$.
Calculando TP y FP con respecto a la clase que se observa en ese instante.
Se define como recall a la cantidad de positivos que se clasificaron como tal, del total de positivos que había.
Se calcula de la siguiente manera: $Recall = \frac{TP}{(TP+FN)}$.

Para obtener tanto la precisión como el recall, se han calculado las medidas TP, FP, TN y FN para cada una de las clases. Por último, se ha calculado las medidas de precisión y recall en dos vectores.

```
TP = [];
TN = [];
FP = [];
FN = [];
for i = 1:10
    TP = [TP sum(ytest == i & ypred == i)];
    TN = [TN sum(ytest ~= i & ypred ~= i)];
    FP = [FP sum(ypred == i & ytest ~= i)];
    FN = [FN sum(ypred ~= i & ytest == i)];
end
Precision = TP ./ (TP + FP)
Recall = TP ./ (TP + FN)
```

En la tabla precision/recall se muestran los resultados obtenidos tanto de precisión como de recall para cada una de las clases.

Como se puede observar, las clases con menor precisión don el 8, seguido del 3 y el 2. Esto es debido a que en numerosas ocasiones se predicen que las entradas son de esta clase, cuando en realidad no lo son. De la misma manera, las clases con peor recall son el 8, 5 y 2. Como ya hemos podido observar anteriormente en la matriz de confusión, son las clases que peor predicción tienen y por consecuente, los número más problemáticos.

Por último, se pide visualizar las confusiones con la función $verConfusiones$.

Como se puede observar en la imagen 3 el modelo confunde algunos dígitos con otros debido a su mala grafía. Se muestran más ejemplos de aquellos números que tienen una peor predicción como pueden ser el 2 o el 8.

Cuadro 2: Tabla de precisión/recall.

	Precision	Recall
1	0.8818	0.97
2	0.8646	0.83
3	0.85	0.85
4	0.88	0.88
5	0.8737	0.83
6	0.8919	0.99
7	0.9286	0.91
8	0.8367	0.82
9	0.8958	0.86
0	0.9792	0.94

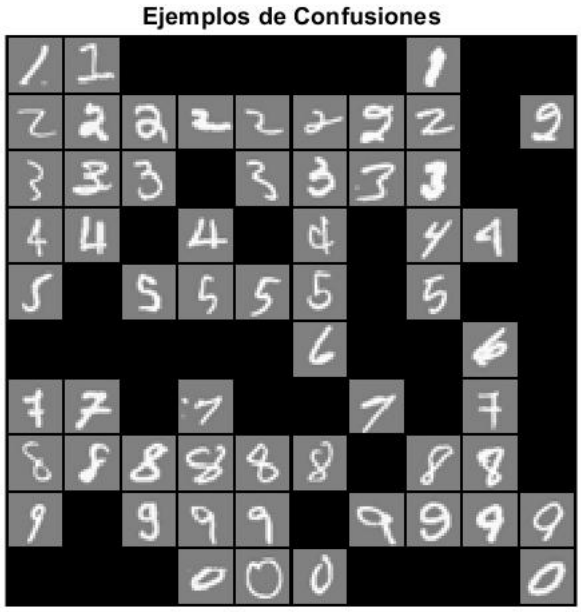

Figura 3: Ejemplos de confusión con los dígitos.

5. Comparación de resultados con redes neuronales

Como ya se comentó en clase, durante el primer semestre se construyó un modelo con redes neuronales en la asignatura de Inteligencia Artificial en la que se utilizaba este mismo conjunto de datos, por lo tanto, se van a comentar los resultados obtenidos en las redes neuronales que se realizaron en dicha asignatura con los resultados obtenidos en la regresión logística.

Para que los resultados sean equiparables, se va a calcular la medida de accuracy del modelo realizado, y de esta manera se evitan repetir cálculos bastante costosos en tiempo con las redes neuronales ya entrenadas.
De la misma manera, las matrices de confusión obtenidas en esta práctica y las matrices que se calcularon en la asignatura de Inteligencia Artificial no son comparables porque en el segundo caso se utilizó un conjunto de datos más amplio. Sin embargo, para realizar esta práctica se ha utilizado un conjunto de datos reducido del conjunto MNIST.
Se define la medidas de *accuracy* como el total de muestras bien clasificadas en modelo. La medida de accuracy se calcula de la siguiente manera: $accuracy = \frac{(TP+TN)}{(TP+TN+FP+FP)}$.
Con el modelo realizado se ha obtenido una medida de $accuracy = 0{,}9776$.

Inicialmente se calculó una red neuronal con una sola capa. El objetivo era encontrar cuál es la mejor función de activación para la capa de salida. Como resultado se obtuvo que la mejor función de activación para la capa de salida era $softplus$. Con esta red la medida obtenida era $accuracy = 0{,}9296$. Por lo tanto, una red neuronal con una sola capa, no supera al modelo entrenado mediante regresión logística regularizada.

El siguiente problema consistía en encontrar un modelo que tuviera una capa oculta, y era necesario averiguar cuántas neuronas se deben colocar en la capa oculta y cuál es la mejor función de activación para la capa oculta. Se realizaron sucesivos experimentos y finalmente se obtuvo que eran necesarias 177 neuronas en la capa oculta y que la mejor función de activación era la función $relu$. Para la capa de salida se usó la función $sigmoidal$ por simplicidad.
Con estas características se obtuvo un accuracy de 0,9826.
En este caso, la red neuronal es muy precisa y es mejor que el modelo calculado con regresión logística. Esto puede ser debido a, como ya se ha comentado antes, que en el modelo de redes neuronales se utilizaron bastantes más datos de entrenamiento.

Por último, se pedía una red neuronal con dos capas ocultas. Igual que en el caso anterior, era necesario averiguar el número exacto de neuronas y las funciones de activación que proporcionan un mejor resultado. En el caso de la primera capa, se obtuvo que eran necesarias 167 neuronas, y la función de activación que proporcionaba mejores resultados era $relu$. En la segunda capa, los mejores resultados se obtuvieron con 67 neuronas, y la función de activación también era $relu$. En la capa de salida se mantuvo como función de activación la función $sigmoidal$.
Con esta red neuronal se obtuvo un resultado algo peor que el anterior, pero mejor que el obtenido con el modelo logístico. El resultado es $accuracy = 0{,}982$.

Como conclusión de esta práctica, cabe destacar que, con una red neuronal más complicada no siempre se van a obtener mejores resultados. Sin embargo, con una buena red neuronal, como ya se ha podido observar, se pueden obtener mejores resultados que con un modelo logístico regularizado.