Regresión Logística

Práctica 3

César Borja

14/03/2022

Aprendizaje Automático

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

1. Estudio previo

```
function [best_model, Etr, Ecv] = kfold_cross_validation(X, y, k, models)
% k-fold para elegir el parametro de regularización
    best_model = 0;
    best_errV = inf;
    options = [];
    options.display = 'final';
    options.method = 'newton';

    Etr = [];
    Ecv = [];
    [rows, ~] = size(models);
    for model = 1:rows
        etr = 0; ecv = 0;
        for fold = 1:k
            [Xcv, ycv, Xtr, ytr] = particion(fold,k,X,y);
            th = minFunc(@costeLogReg, zeros(size(Xtr,2),1), options, Xtr, ytr, models(model));

            h = 1./(1+exp(-(Xtr*th))) >= 0.5;
            etr = etr + tasa_error(h,ytr);

            h = 1./(1+exp(-(Xcv*th))) >= 0.5;
            ecv = ecv + tasa_error(h,ycv);
        end
        etr = etr/k;
        Etr = [Etr;etr];
        ecv = ecv/k;
        Ecv = [Ecv;ecv];
        if ecv < best_errV
            best_errV = ecv;
            best_model = model;
        end
    end
end
```

Figura 1: Algoritmo de k-fold cross-validation para elegir el valor del parámetro de regularización

2. Regresión logística básica

En este apartado se pide predecir los alumnos admitidos en función de la calificación de dos exámenes, a partir de los datos del fichero exam_data.txt.

Para ello se ha utilizado el paquete minFunc y la función de coste logístico mostrada en el Anexo I y se han utilizado el 20 % de los datos como datos de test.

En la Figura 2 se muestra la frontera de decisión obtenida con los pesos resultantes de entrenar el modelo con los datos de entrenamiento. Como se puede ver, con una frontera de decisión lineal es suficiente para conseguir un buen modelo para este caso.

Interpretando la gráfica, podemos concluir que se necesita aproximadamente una nota media de 60 puntos sobre 100 entre los dos exámenes para ser admitido.

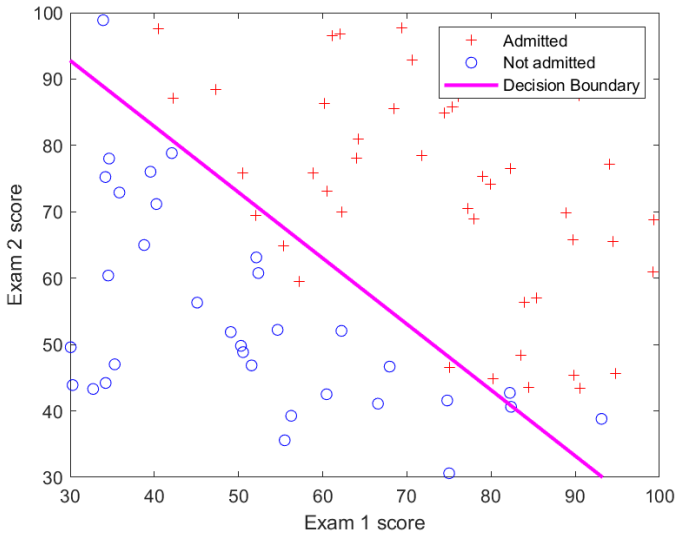

Figura 2: Frontera de decisión del clasificador de alumnos admitidos

Las tasas de error con los datos de entrenamiento y con los datos de test son las siguientes:

$$\begin{aligned} E_tr &= 0{,}1125 \\ E_test &= 0{,}1000 \end{aligned} \tag{1}$$

Para un alumno que ha sacado 45 puntos en el primer examen, la probabilidad de ser admitido en función de la calificación del segundo examen se muestra en la Figura 3. Se puede observar una sigmoide que empieza a

crecer en torno a los 60 puntos de nota del segundo examen. La probabilidad es 0 para notas inferiores a 50 y prácticamente 1 para notas superiores a 85 puntos.

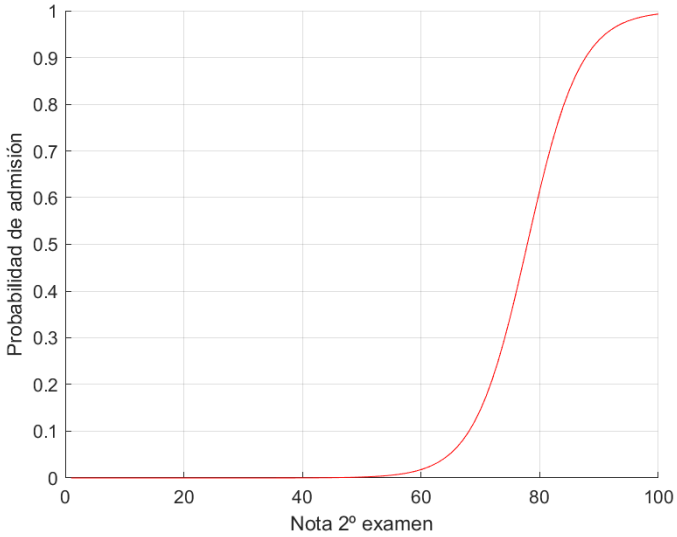

Figura 3: Probabilidad de que el alumno sea admitido en función de la calificación del segundo examen

3. Regularización

Utilizando ahora el conjunto de datos mchip_data.txt, se pide utilizar una regresión logística regularizada con expansión de funciones base para decidir se aceptar o rechazar microchips utilizando solamente dos test de funcionamiento por chip. De nuevo, se separa un 20 % de los datos para test. El parámetro de regularización lambda se elige con el algoritmo k-fold cross-validation realizado en la sección 1. El entrenamiento de cada modelo en el interior del algoritmo se realiza con el paquete minFunc y la función de coste logarítmico regularizado que se encuentra en el Anexo II.

En la Figura 4 se muestran las curvas de evolución de las tasas de error con los datos de entrenamiento y de validación. Se puede apreciar como con valores muy pequeños de lambda, la regularización no tiene apenas presencia y se produce sobreajuste (error de entrenamiento muy bajo pero error de validacióon alto). Por el contrario, con valores muy grandes se produce

subajuste claro, pues tanto la tasa de error de entrenamiento como la de validación crecen. Esto ocurre porque se regulariza el polinomio demasiado y se convierte en un modelo demasiado simple. El punto ideal con el que se consigue la menor tasa de error de validación utiliza $\lambda = 0.00028118$.

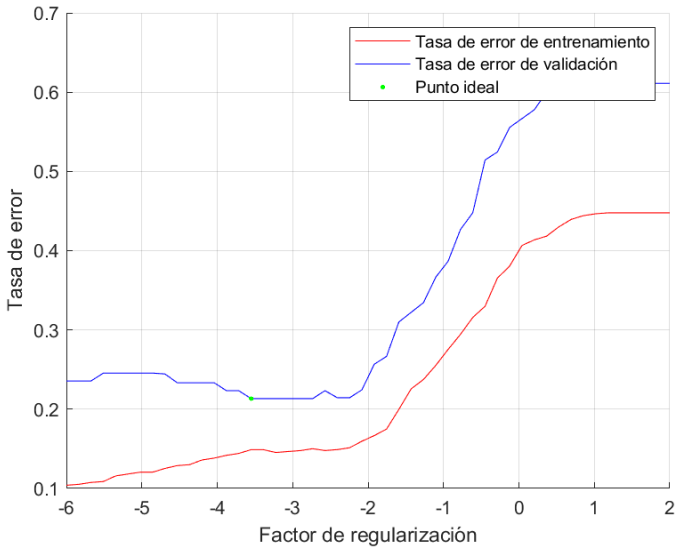

Figura 4: Evolución de las tasas de error con datos de entrenamiento y datos de validación

Una vez tenemos el mejor modelo, se entrenan todos los datos (excepto los de test) con el mejor modelo y con $\lambda = 0$ y se dibujan las superficies de separación. En las Figura 5 se puede ver como la sección está sobreajustada a los datos de entrenamiento, mientras que en la Figura 6 la superficie es más suave y generalizará mejor para los datos de test.

¿Cuál de los dos modelos es mejor?
Observando las superficies ya se puede estimar que el modelo regularizado se comportará mejor con nuevos datos de entrada. Además el mejor modelo encontrado se ha seleccionado teniendo en cuenta el valor de la tasa de error para los datos de validación, por lo que el modelo con $\lambda = 0$ sobreajustado obtiene una tasa de error de validación mayor. (Figura 4).

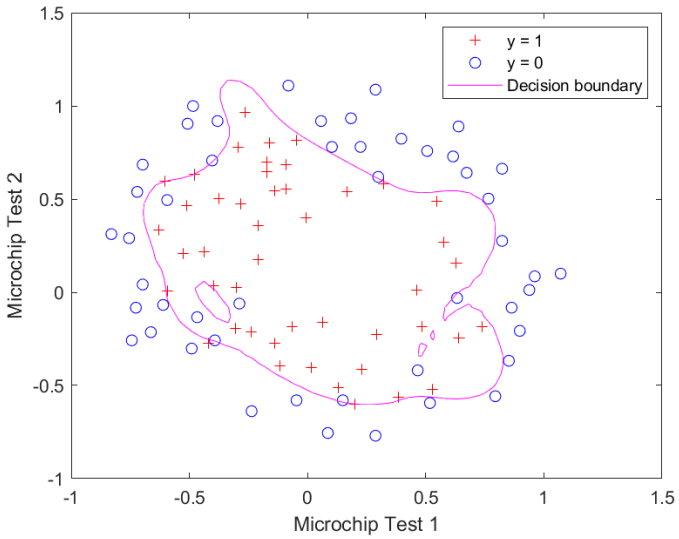

Figura 5: Superficie de separación de modelo con $\lambda = 0$

4. Precisión/Recall

Con el mejor modelo obtenido en el apartado anterior, se pide utilizar los datos de test para calcular la matriz de confusión y los valores de precisión y recall.

La matriz de confusión obtenida es la siguiente:

$$\begin{pmatrix} 8(TP) & 2(FP) \\ 10(TN) & 4(FN) \end{pmatrix}$$

A partir de la matriz de confusión, sacamos la precisión y el recall:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} = 0{,}8 \\ Recall &= \frac{TP}{TP + FN} = 0{,}6\widehat{6} \end{aligned} \tag{2}$$

Para conseguir que el 95 % de los chips aceptados sean buenos, es decir, una precisión de 0.95, debemos ajustar el umbral de decisión. En la Figura 7 se encuentra el fragmento de código usado para encontrar el umbral que proporcione una precisión de 0.95 o más. El umbral obtenido es de 0.84.

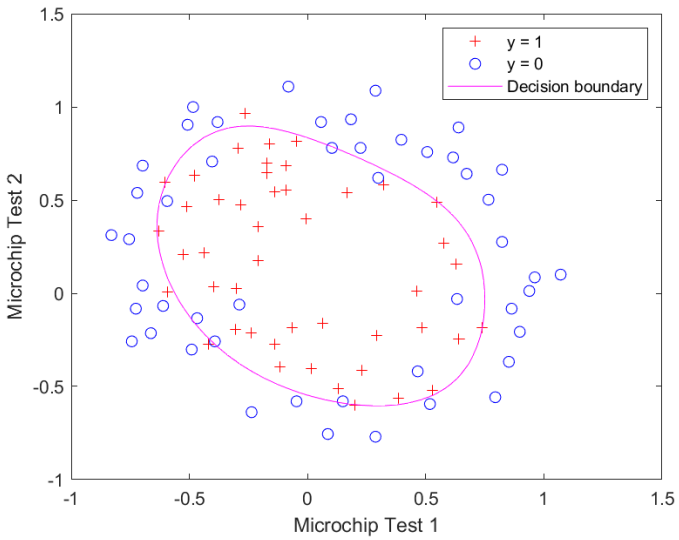

Figura 6: Superficie de separación del mejor modelo encontrado

```
for umbral=0:0.01:1
    ytest_pred = 1./(1+exp(-(Xtest_exp*th1))) >= umbral;
    TP = sum(ytest_pred == 1 & ytest == 1);
    FP = sum(ytest_pred == 1 & ytest == 0);
    P_buscada = TP/(TP+FP);
    if(P_buscada >= 0.95)
        disp(umbral);
        break
    end
end
```

Figura 7: Búsqueda del umbral que proporciona P $\geq$ 0,95

Anexo I. Función de coste logístico

```
function [J,grad,Hess] = CosteLogistico(theta,X,y)
% Calcula el coste logistico, y si se piden,
% su gradiente y su Hessiano
    N = size(X,2);
    h = 1./(1+exp(-(X*theta)));
    J = (-y'*log(h) - (1-y')*log(1-h))/N;
    if nargout > 1
        grad = X'*(h-y)/N;
    end
    if nargout > 2
        R = diag(h.*(1-h));
    Hess = X'*R*X/N;
    end
end
```

Anexo II. Función de coste logístico regularizado

```
function [J,grad,Hess] = CosteLogReg(theta,X,y,lambda)
% Calcula el coste logistico regularizado,
% y si se piden, su gradiente y su Hessiano
    [N, D] = size(X);
    h = 1./(1+exp(-(X*theta)));
    th = theta; th(1)= 0; %Para no penalizar theta_0
    J =(-y'*log(h) - (1-y')*log(1-h))/N +(lambda/2)*(th'*th);
    if nargout > 1
        grad = X'*(h-y)/N + lambda*th;
    end
    if nargout > 2
        R = diag(h.*(1-h));
        Hess = X'*R*X/N + diag([0 lambda*ones(1,D-1)]);
    end
end
```