Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Aprendizaje automático

Practica 3: Regresión logística

Autora:
Nerea Gallego Sánchez (801950)

7 de diciembre de 2022

Índice

1. Introducción y objetivos

Esta práctica consiste en aplicar regresión logística en casos sencillos de clasificación binaria utilizando técnicas de regularización y validación cruzada.

El caso de estudio es la predicción de qué alumnos serán admitidos a una universidad en función de la calificación obtenida en dos exámenes. En este primer caso se aplica regresión logística básica.
El segundo caso de estudio es el control de calidad de una planta de fabricación de microchips. Se quiere decidir si aceptar o rechazar los microchips utilizando únicamente dos tests.

Los distintos casos que se van a probar son: regresión logística básica, regularización y precisión/recall.

2. Estudio previo

```
function [H, errTrain, errV] = K_fold_cross_validation(K, funcTrain, X, y, models, funcErr)
    bestModel = 0;
    bestErrV = inf;
    errT = [];
    errV = [];
    for model = 1:size(models, 1)
        err_T = 0;
        err_V = 0;
        for fold = 1: K
            [x_tset, y_tset, x_tset, y_tset] = particion(fold, K, X, y);
            th = funcTrain(models(model, :), x_tset, y_tset);
            err_T = err_T + funcErr(th, x_tset, y_tset, models(model,:));
            err_V = err_V + funcErr(th, x_tset, y_tset, models(model, :));
        end
        err_T = err_T / K;
        err_V = err_V / K;
        errT = [errT err_T];
        errV = [errV err_V];
        if err_V < bestErrV
            bestModel = model;
            bestErrV = err_V;
        end
    end
    H = bestModel;
end
```

```
function t = entrenaRegularizacion(lambda, X, y)
    options = [];
    options.display = 'final';
    options.method = 'newton';
    theta_ini = zeros(size(x, 2), 1);
    t = minFunc(@costeLogReg, theta_ini, options, X, y, lambda);
end
```

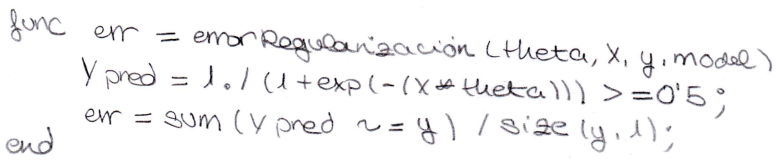

Figura 1: Estudio previo a la práctica.

3. Regresión logística básica

Se pide predecir qué alumnos serán admitidos en una universidad, en función de la clasificación obtenida en dos exámenes.
A partir de un fichero de datos de entrada, es necesario separar un 20 % de los datos de test. Los datos restantes se utilizarán como datos de entrenamiento.

Para empezar, se ha creado una función, separar que dado el conjunto de datos de entrada y, el porcentaje concreto de datos que se quieren separar, devuelven los datos de entrenamiento y test con sus respectivas salidas. El volumen de datos devuelto para entrenamiento es acorde al que se indica en el parámetro de la función.

```
function [Xtr, Ytr, Xtst, Ytst] = separar(data,porc)
    randOrd = randperm(size(data,1));
    perm = data( randOrd,:);
    lim = int32(size(data,1)*porc);
    Xtr = perm((1:lim),[1,2]);
    Ytr = perm(1:lim,3);
    Xtst = perm(lim+1:end,[1,2]);
    Ytst = perm(lim+1:end,3);
end
```

Se asume que la matriz inicial de datos tiene en sus dos primeras columnas los atributos con los que se quiere entrenar. La matriz de datos data, tiene en la tercera columna las salidas.
Los conjuntos de datos devueltos se escogen de manera aleatoria de entre todos los datos que hay en el conjunto inicial. Por lo tanto, ejecuciones sucesivas puede dar como resultado conjuntos de datos distintos.

Una vez se dispone del conjunto de datos que se va a utilizar para entrenar el modelo, se añade la columna de unos para calcular el vector de pesos.

Se pide programar una función de coste y resolver la regresión logística.
Como función de coste de ha utilizado la función proporcionada en clase por los profesores: CosteLogistico.
Se utiliza la función *minFunc* proporcionada por los profesores de la asignatura para calcular el vector de pesos del modelo.

```
function [J,grad,Hess] = CosteLogistico(theta,X,y)
    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
```

Se ha elegido como vector de pesos inicial uno en el que contiene todo 0's, ya que al realizar el descenso de gradiente, no influye demasiado el vector inicial de pesos.
Como opciones de la función se han utilizado las siguientes:

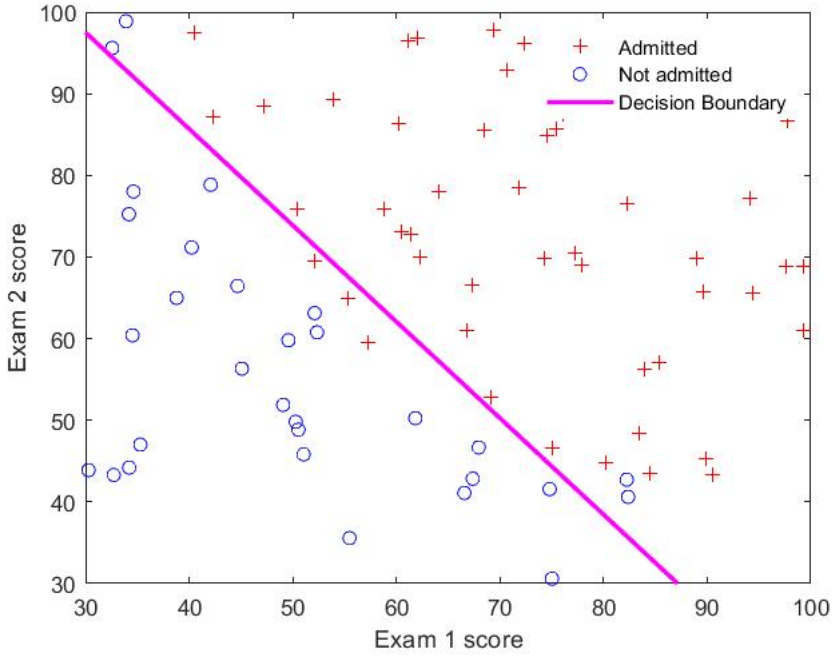

Figura 2: Regresión logística calculada a partir de los datos de entrenamiento.

```
options.display = 'final';
options.method = 'newton';
```

Finalmente, se ha pasado a la función *minFunc* el vector inicial de pesos, los datos de entrenamiento y la salida esperada. La función devuelve el vector de pesos calculado.

Se muestra en la gráfica 2 la regresión calculada mostrando los datos de entrenamiento. Como se puede observar, la línea rosa separa mayoritariamente los puntos azules de los puntos rojos. Por lo tanto, se puede decir que se ha calculado adecuadamente la regresión. Sin embargo, en la frontera se sitúan algunos puntos que no se encuentran clasificados correctamente. Es decir, el modelo no es perfecto, ya que con una línea recta no se pueden separar todos los puntos rojos de todos los puntos azules.
En este caso, se puede ver como la regresión logística se ajusta adecuadamente al modelo que se quiere predecir. Por lo tanto, al tener estos datos de entrada, no es necesario calcular un modelo más complejo y sería suficiente con una regresión logística básica.

La tasa de error se calcula encontrando el porcentaje de predicciones incorrectas con respecto al total de predicciones realizadas.
Para calcular la salida se ha utilizado un umbral de 0.5 de manera que si la salida obtenida por la función es mayor o igual que 0.5, la salida se considera 1, sin embargo, si la salida es menor que 0.5, se considerará que ese dato tiene como salida 0.

La tasa de error con los datos de entrenamiento es $TasaErrorTrain = 0{,}0875$.
La tasa de error con los datos de test es $TasaErrorTest = 0{,}1000$.

Como se puede observar, el error de entrenamiento es algo menor que el error obtenido con los datos de test. Esto es debido a que los datos se ajustan correctamente a un modelo de regresión logística básica. Además, la cantidad de datos de la que se dispone es pequeña, por lo tanto, el error no es muy grande ya que el modelo generaliza bien.

Finalmente, se pide dibujar una gráfica en la que se muestre la probabilidad de que un alumno que haya obtenido una calificación de 45 puntos en el primer examen, sea admitido en la universidad en función de la nota del segundo examen.
Para realizar esta gráfica, se han elegido puntos a lo largo de las posibles notas que puede obtener un alumno en el segundo examen (se ha asumido que son de 0 a 100). A continuación se ha calculado la

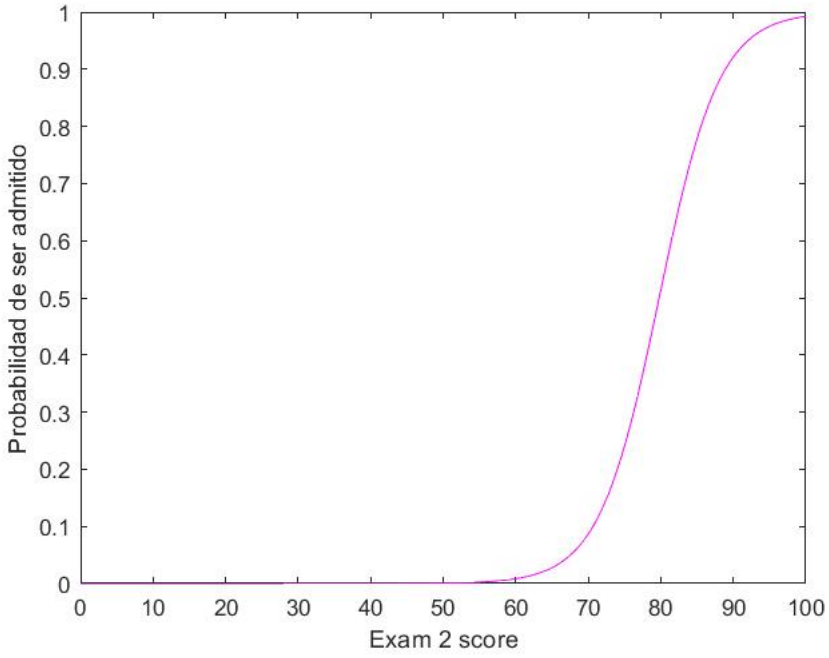

Figura 3: Probabilidad que tiene un alumno de ser admitido en la universidad en función de la nota del segundo examen.

predicción realizada y se ha mostrado el resultado en la gráfica 3.
Como se puede observar en la gráfica, la probabilidad de que sea admitido cuando la nota del segundo examen es menor de 60 puntos, es prácticamente nula, a pesar de que la media de ambos es aprobado. Para que el alumno sea admitido, la nota del segundo examen debe tener una nota mayor que 79 puntos (aproximadamente). De esta manera, la probabilidad obtenida de ser admitido es mayor que 0.5. Por lo tanto, la salida con el umbral elegido sería de 1. Y por consecuente, la salida predicha sería admitido.

4. Regularización

Se pide realizar un modelo para decidir si unos chips son aceptados o rechazados tras el control de calidad de una planta de fabricación de microchips. De cada microchip se pasa una serie de test de funcionamiento que proporcionan resultados numéricos.
El fichero de datos de entrada contiene los resultados obtenidos por un conjunto de microchips y si finalmente fueron aceptados o rechazados.

Para empezar, se ha utilizado la función separar para conseguir dividir los datos en un set de datos de entrenamiento y otro distinto para test. Los datos de entrenamiento son el 80 % del total de los datos y los datos de test corresponden al 20 % restante.

Una vez se dispone del conjunto de datos que se va a utilizar para entrenar el modelo, se añade la columna de unos como primer atributo del conjunto de datos para calcular el vector de pesos. A continuación, se ha realizado la expansión de funciones base mediante la función $mapFeature$ proporcionada por los profesores de la asignatura.

Ha sido necesario encontrar el parámetro de regularización lambda mediante $k-fold\ cross-validation$. Como función de k-fold se ha utilizado la misma función construida que en la práctica anterior:

```
function [H, errT, errV] = k_fold_cross_validation(k, func, X, y, models, funcErr)
    bestModel = 0;
    bestErrV = inf;
    size(models,1)
    errT = [];
    errV = [];
    for model = 1:size(models,1)
        err_T = 0;
        err_V = 0;
        for fold = 1:k
            models(model,:);
            [x_tsset, y_tsset, x_trset, y_trset] = particion(fold,k,X,y);
            th = func(models(model,:),x_trset,y_trset);
            err_T = err_T + funcErr(th,x_trset,y_trset,models(model,:));
            err_V = err_V + funcErr(th, x_tsset, y_tsset,models(model,:));
        end
        err_T = err_T / k;
        err_V = err_V / k;
        errT = [errT err_T];
        errV = [errV err_V];
        if err_V < bestErrV
            bestModel = model;
            bestErrV = err_V;
        end
    end
    H = bestModel;
end
```

Para encontrar el mejor parámetro de regularización se ha utilizado un vector que contiene distintos valores para el parámetro lambda. Estos valores de lambda van desde 10^{-6} hasta 10^{2} en escala exponencial.

A continuación, se han creado dos funciones, una para entrenar el modelo y otra que calcula el error del modelo. Ambas funciones se pasan por parámetro a la función $k - fold\ cross - validation$.
La función de entrenamiento es la siguiente:

```
function t = entrenaRegularizacion(lambda,X,y)
    options = []; options.display = 'final';
    options.method = 'newton'; %por defecto: 'lbfgs'
    theta_ini = zeros(size(X,2),1);
    t = minFunc(@CosteLogReg, theta_ini, options,X, y, lambda);
end
```

Esta función crea el vector de opciones para calcular el vector de pesos. Inicializa un vector de pesos al que le asigna 0's en todas sus componentes. Finalmente, entrena el modelo con con la función *minFunc* y le pasa como parámetros la función CosteLogReg que calcula el coste con regularización, el vector de pesos inicial, las opciones de entrenamiento del modelo, el conjunto de datos inicial, la salida esperada, y el factor de regularización.

```
function [J,grad,Hess] = CosteLogReg(theta,X,y,lambda)
    [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
```

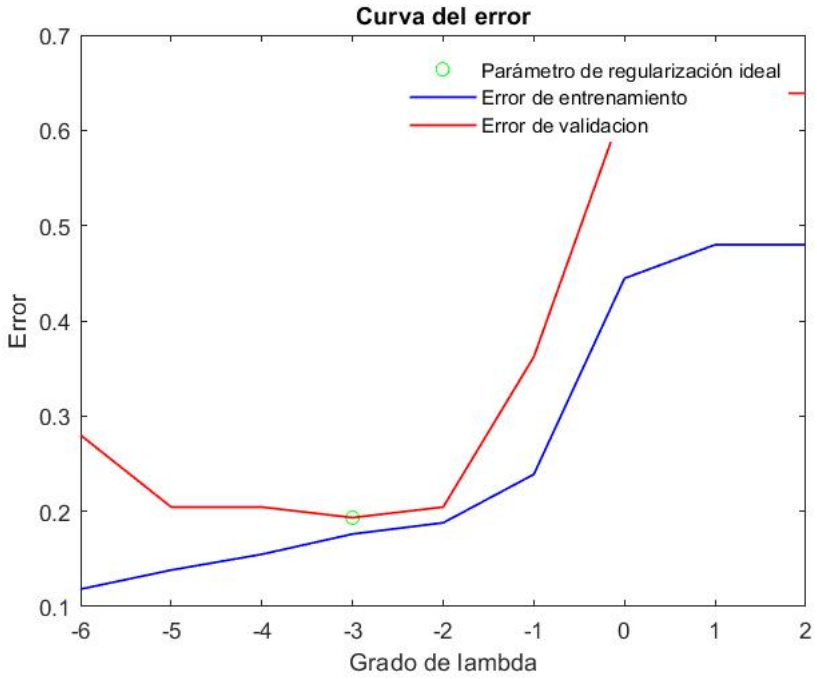

Figura 4: Curva de evolución de las tasas de errores con los datos de entrenamiento y de validación.

La función creada para calcular el error es errorRegularizacion. Esta función calcula los errores medios que se producen con el modelo construido.

```
function err = errorRegularizacion(theta, X, y, model)
    Ypred =  1./(1+exp(-(X*theta))) >= 0.5 ;
    err = sum(Ypred ~= y) / size(y,1);
end
```

La función k-fold cross-validation devuelve el mejor modelo y los errores medios obtenidos, tanto con los datos de entrenamiento como con los datos de validación para cada posible modelo. Como se puede observar en la gráfica 4 al inicio, el error de regularización es muy grande, ya que se está produciendo sobreajuste en el modelo. Sin embargo, cuanto mayor es el parámetro de lambda, se produce un mayor subajuste y por consecuente, los resultados obtenidos son peores. Por lo tanto, el parámetro ideal de lambda es $\lambda = 1{,}0000e-03$ ya que es el valor que proporciona menor error con los datos de validación.

A continuación se pide entrenar el modelo con todos los datos (excepto los separados para test) con el mejor modelo encontrado y con el modelo $\lambda = 0$, y dibujar las correspondientes superficies de separación. Como se puede observar en la gráfica 5 (modelo regularizado con $\lambda = 1{,}0000e-03$) la predicción hace un buen ajuste al modelo, ignorando los datos que son espurios y dejando un pequeño margen de error en la frontera, como es habitual en este tipo de modelos.
Sin embargo, en la gráfica 6 (modelo sin regularización $\lambda = 0$) vemos como el modelo produce sobreajuste a los datos de entrenamiento, dejando la superficie de separación de manera extraña y separando absolutamente todos los datos de entrenamiento de los de test.
Por lo tanto, se puede concluir que el modelo regularizado es mejor que el modelo no regularizado ya que generaliza mejor a nuevos datos.

5. Precisión/Recall

El último apartado pide, con el mejor modelo obtenido, utilizar los datos de test para calcular la matriz de confusión y los valores de precisión y recall.

Se ha calculado la predicción con los datos de test. A continuación se ha calculado cada uno de los valores que debe contener la matriz de confusión:

Como se puede observar en la matriz de confusión el modelo es adecuado, ya que en la mayoría de los casos acierta el resultado correspondiente y se producen pocos falsos positivos y falsos negativos.

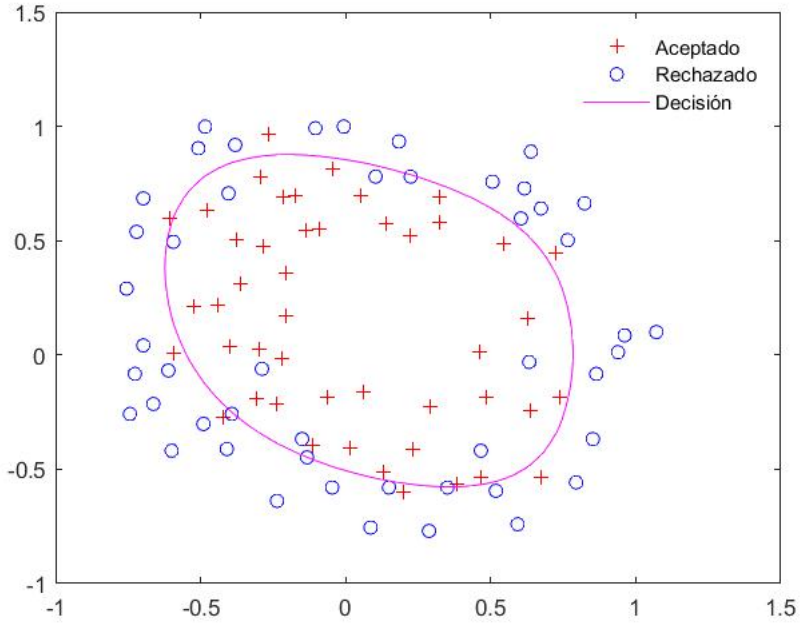

Figura 5: Superficie de separación con el mejor modelo encontrado.

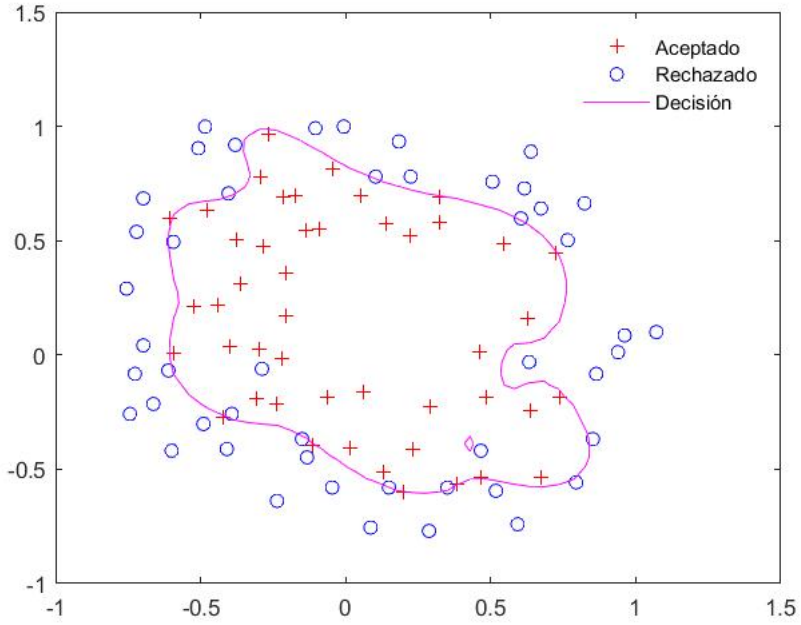

Figura 6: Superficie de separación con el modelo lambda = 0.

Cuadro 1: Matriz de confusión.

	Clase real	
Clase predicha	11	2
	1	10

La precisión se calcula de la siguiente manera $Precision = \frac{TP}{(TP+FP)}$ y el resultado obtenido es $Precision =$ 0,8462.
El recall se calcula de la siguiente manera $Recall = \frac{TP}{(TP+FN)}$ y el resultado obtenido es $Recall = 0{,}9167$.
Se define precisión como la fracción de positivos que hay realmente de todos los que fueron clasificados como positivos. Se define como recall, la cantidad de positivos que se clasificaron como tal, del total de positivos que había.
Los resultados obtenidos tanto de precisión como de recall, son adecuados y de ellos se puede concluir que el modelo elegido es adecuado para realizar nuevas predicciones.

Si queremos que el 95 % de los chips aceptados sean buenos, es necesario modificar el umbral de aceptación del modelo.
Para ello, se ha realizado un experimento en el que se modifica el umbral de aceptación. Los umbrales elegidos son desde 0,01 hasta 1 aumentando cada iteración en 0,01. Para cada uno de los umbrales seleccionados, se realiza la predicción usando dicho umbral. A continuación, se calculan tanto la predicción de ese nuevo modelo como el recall.
Como interesa hallar un modelo de manera que el 95 % de los chips aceptados sean buenos, tenemos que encontrar un modelo que tenga una precisión mayor que 0,95. Finalmente, se ha elegido el modelo que tenga un mejor recall de entre los que tienen una precisión mayor que 0,95.
El umbral obtenido es 0,8200.