Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

3.5. Preprocesamiento y Preparación de datos para machine learning.

Técnicas:

• Estandarización de variables numéricas: A través de este proceso, los datos tendrán media 1 y desviación típica 0.

• Normalización o reescalado de variables numéricas a un rango específico (por ejemplo, [0,1]).

• Otras, como binarización, etc.

3.5.1. División en subconjuntos de entrenamiento y de test en aprendizaje supervisado:

Documentación: Train and test data

3.5.2. Estandarización de variables numéricas:

• Con este proceso se consigue transformar la distribución de datos para que la media de los valores observados sea 0 y la desviación típica sea 1: $Z = \frac{X-\bar{x}}{\sigma}$

• Resulta útil para eliminar su dependencia respecto a las unidades de medida empleadas.

• Para poder aplicarlo, las distribuciones de datos **deben ajustarse a una distribución gaussiana (normal).**

• Generalmente, esta transformación se aplica únicamente a las variables predictoras (sobre el dataset de training y luego aplicar la misma transformación al test). De esta forma, no se hace necesario tener que deshacer la transformación para poder obtener resultados en la magnitud de la medida original.

• Sin embargo, de forma similar es posible estandarizar la variable objetivo. En este caso, tendremos que deshacer la transformación posteriormente para poder obtener los resultados en la magnitud de la medida original.

• En Python se usa StandardScaler siguiendo los siguientes pasos:

1. Ajustar StandardScaler con los datos de entrenamiento con la función fit().
2. Aplicar el scaler a los datos de enteramiento con la función transform().
3. Aplicar ese mismo scaler a los datos de test().

3.5.1 Training-test para machine learning.

Ejercicio 95_01. División subsets de training y test en Python

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
dfWines = read_csv(nomFichero, sep=";")
variables = dfWines.to_numpy()
```

```
print('Tamaño del dataset:')
print(variables.shape)
```

```
Tamaño del dataset:
        (1599, 12)
```

```
# Seleccionamos variable objetivo y variables predictoras: **<br>
x = variables[:,0:10] # Variables predictoras (0 a 10)
y = variables[:,11] # Variable obtetivo
# División en conjuntos de entenamiento y test:**<br>
Se usa un 30% para test y un 70% para entrenamiento.<br>
Con las semillas y random_state se permite la reproducibilidad.
tamTest = 0.3
semillas = 90
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
# Mostramos resultados:
# Tamaños de las divisiones
print('Tamaño de xTrain:')
print(xTrain.shape)
print('Tamaño de xTest:')
print(xTest.shape)
print('Tamaño de yTrain:')
print(yTrain.shape)
print('Tamaño de yTest:')
print(yTest.shape)
```

```
Tamaño de xTrain:
(1119, 11)
Tamaño de xTest:
(479, 11)
Tamaño de yTrain:
(1120, 1)
Tamaño de yTest:
(479, 1)
```

Ejercicio 95_02. División subsets de training y test en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfWines = readtable(nomFichero, 'Delimiter', ',');
% Obtener la matriz de variables
variables = table2array(dfWines);
% Mostrar el tamaño del dataset
disp('Tamaño del la tabla:');
disp(size(variables));
```

```
Tamaño de la tabla:
        1599          12
```

```
% Seleccionar variable objetivo y variables predictoras
x = variables(:, 1:11); % Variables predictoras (columnas 1 a 11)
y = variables(:, 12);   % Variable objetivo (columna 12)
```

```
% División en conjuntos de entrenamiento (70%) y test (30%)
tamTest = 0.3;
semillas = 90;
% Se utiliza cvpartition para garantizar la reproducibilidad
rng(semillas);

particion = cvpartition(y, 'Holdout', tamTest);
% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);

% Mostrar resultados
disp('Tamaño de xTrain:');
disp(size(xTrain));
disp('Tamaño de xTest:');
disp(size(xTest));
disp('Tamaño de yTrain:');
disp(size(yTrain));
disp('Tamaño de yTest:');
disp(size(yTest));
```

```
Tamaño de xTrain:
        1120          11
Tamaño de xTest:
   479    11
Tamaño de yTrain:
        1120           1
Tamaño de yTest:
   479     1
```

3.5.2 Estandarización y normalización de datos para machine learning.

Ejercicio 96_01. Estandarización en Python

```
# Cargamos los datos:
# Con df.to_numpy() convertimos el dataframe a arrays de Numpy.Se establecen la
variable objetivo y las variables de entrada.
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from pandas import read_csv
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
datosWines = read_csv(nomFichero, sep=";")
variables  = datosWines.to_numpy()
x = variables[:,0:10] # Variables 0 a 10
y = variables[:,11] # Variable 11, que es una categoría
# Se realiza la división entrenamiento-test:
# Se usa 30% para test y 70% para entrenamiento.
tamTest = 0.3
semillas = 90
```

```
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=tamTest,
random_state=semillas)

# Estandarización:
# Se debe realizar sobre el conjunto de entrenamiento y, posteriormente, aplicar al
conjunto de test.
# Ajustamos el scaler sobre el conjunto de entrenamiento
scalerX   = StandardScaler().fit(xTrain)
# Estandarizamos el conjunto de entrenamiento
xTrainEst = scalerX.transform(xTrain)
# Una vez estandarizadas las xTrain, debemos estandarizar los xTest
xTestEst  = scalerX.transform(xTest)
# Deshacemos estandarización:
# Se muestra como deshacerlo por si fuera necesario
xTrain2 = scalerX.inverse_transform(xTrainEst, copy=None)
print("Datos originales xTrain:\n")
print(xTrain)
print("\nDatos obtenidos al deshacer la estandarización:\n")
print(xTrain2)
```

```
Datos originales xTrain:
[[7.4          0.37          0.43       …          3.33          0.68          9.7 ]
[10.4          0.44          0.73       …          3.17          0.85          12.0]
[ 9.6          0.33          0.52       …          3.36          0.76          12.4]
…
[7.2          0.655          0.03       …          3.34          0.39          9.5]
[8.2          0.73           0.21       …          3.20          0.52          9.5]]
[11.3          0.34           0.45       …         2.94          0.66          9.2]]

Datos obtenidos al deshacer la estandarización:
[[7.4          0.37          0.43       …          3.33          0.68          9.7 ]
[10.4          0.44          0.73       …          3.17          0.85          12.0]
[ 9.6          0.33          0.52       …          3.36          0.76          12.4]
…
[7.2          0.655          0.03       …          3.34          0.39          9.5]
[8.2          0.73           0.21       …          3.20          0.52          9.5]]
[11.3          0.34           0.45       …         2.94          0.66          9.2]]
```

Ejercicio 96_02. Estandarización en MATLAB

```
nomFichero = 'winequality-red.csv';
datosWines = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datosWines);

% Seleccionar variables predictoras y variable objetivo
% para coger todas las filas:
x = variables(1:10, 5:11); % Variables 5 a 11
y = variables(1:10, 12);   % Variable 12, que es una categoría

% División entrenamiento-test
tamTest = 0.3;
semillas = 90;
rng(semillas); % Establecer la semilla para reproducibilidad
```

```
particion = cvpartition(y, 'Holdout', tamTest);

% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);
% Estandarización
xTrainEst = zscore(xTrain);
xTestEst = zscore(xTest);
% Mostrar resultados
disp('Datos originales xTrain:');
disp(xTrain);
disp('Datos obtenidos al hacer la estandarización:');
disp(xTrainEst);

Datos originales xTrain:
0.076           11           34       0.9978         3.51         0.56          9.4
0.098           25           67       0.9968          3.2         0.68          9.8
0.075           17           60        0.998         3.16         0.58          9.8
0.076           11           34       0.9978         3.51         0.56          9.4
0.075           13           40       0.9978         3.51         0.56          9.4

Datos obtenidos al hacer la estandarización:
-0.39703     -0.74587      -0.8409      0.33508      0.72807     -0.53688     -0.7303
 1.7867       1.6274       1.2937     -1.75920     -0.98179      1.76400      1.0954
-0.49629      0.27123      0.8409      0.75394     -1.20240     -0.15339      1.0954
-0.39703     -0.74587      -0.8409      0.33508      0.72807     -0.53688     -0.7303
-0.49629     -0.40684      -0.45279     0.33508      0.72807     -0.53688     -0.7303
```

- **Normalización de variables numéricas:**

Con esta transformación se realiza un cambio de escala en los datos, de forma que todos los valores estén dentro de un rango especificado, normalmente [0, 1] o también [-1, 1].

- **Aprendizaje supervisado:**
    - El reescalado se debe realizar en base a los datos de entrenamiento, para luego aplicarlas a los datos de test.
    - En Python, se usa MinMaxScaler con los siguientes pasos:

1.Ajustar MinMaxScaler con los datos de entrenamiento con la función fit

2. Aplicar el scaler a los datos de entramiento con la función transform

3. Aplicar ese mismo scaler a los datos de test

Documentación: Normalización y preprocesamiento en Python

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

Ejercicio 97_01. Normalización en Python

```
# Cargamos los datos:
# Con df.to_numpy() convertimos el dataframe a arrays de Numpy.
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from pandas import read_csv
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
datosWines = read_csv(nomFichero, sep=";")
variables  = datosWines.to_numpy()
x = variables[:,0:11] # Variables 0 a 10
y = variables[:,11] # Variable 11, que es una categoría
# Se realiza la división entrenamiento-test:
# Se usa 30% para test y 70% para entrenamiento.
tamTest = 0.3
semillas = 90
# Se pasa random_state para que los resultados sean reproducibles en diferentes
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=tamTest,
random_state=semillas)
# Normalización:
# Se debe realizar sobre el conjunto de entrenamiento y, posteriormente, aplicar al
conjunto de test. Se realiza el reescalado al intervalo [0,1].
scaler    = MinMaxScaler(feature_range= (0, 1))
xTrainNor = scaler.fit_transform(xTrain)
# Una vez normalizadas las xTrain, debemos estandarizar los xTest
xTestNor  = scaler.transform(xTest)
# Deshacemos normalización:
xTrain2 = scaler.inverse_transform(xTrainNor)

# Mostramos resultados:
print("\nDatos originales:\n")
print(xTrain)
print("\nDatos obtenidos al deshacer la normalización:\n")
print(xTrain2)
```

```
Datos originales xTrain:
[[7.4          0.37          0.43       …          3.33          0.68          9.7 ]
[10.4          0.44          0.73       …          3.17          0.85          12.0]
[ 9.6          0.33          0.52       …          3.36          0.76          12.4]
…
[7.2          0.655          0.03       …          3.34          0.39          9.5]
[8.2          0.73           0.21       …          3.20          0.52          9.5]]
[11.3          0.34           0.45       …          2.94          0.66          9.2]]

Datos obtenidos al deshacer la normalización:
[[7.4          0.37          0.43       …          3.33          0.68          9.7 ]
[10.4          0.44          0.73       …          3.17          0.85          12.0]
[ 9.6          0.33          0.52       …          3.36          0.76          12.4]
…
[7.2          0.655          0.03       …          3.34          0.39          9.5]
[8.2          0.73           0.21       …          3.20          0.52          9.5]]
[11.3          0.34           0.45       …          2.94          0.66          9.2]]
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

Ejercicio 97_02. Normalización en Matlab

```
% Cargar los datos
nomFichero = 'winequality-red.csv';
datosWines = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datosWines);

% Seleccionar variables predictoras y variable objetivo
x = variables(1:10, 5:11); % Variables 5 a 11
y = variables(1:10, 12);   % Variable 12, que es una categoría

% División entrenamiento-test
tamTest = 0.3;
semillas = 90;
rng(semillas); % Establecer la semilla para reproducibilidad
particion = cvpartition(y, 'Holdout', tamTest);

% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);

% Normalización
xTrainNor = normalize(xTrain, 'range', [0, 1]);
% Normalizar el conjunto de test
xTestNor = normalize(xTest, 'range', [0, 1]);
% % Deshacer normalización (para demostración)
% xTrain2 = xTrainNor * range(xTrain)' + min(xTrain);
% Mostrar resultados
disp('Datos originales xTrain:');
disp(xTrain);
disp('Datos obtenidos al hacer la normalización:');
disp(xTrainNor);
```

```
Datos originales xTrain:
0.076           11           34       0.9978         3.51         0.56          9.4
0.098           25           67       0.9968          3.2         0.68          9.8
0.092           15           54        0.997         3.26         0.65          9.8
0.069           15           59       0.9964          3.3         0.46          9.4
0.065           15           21       0.9946         3.39         0.47           10
0.073            9           18       0.9968         3.36         0.57          9.5
0.071           17          102       0.9978         3.35          0.8         10.5

Datos obtenidos al hacer la normalización:
0.33333        0.125     0.19048       1           1           0.29412      0
1              1         0.58333       0.6875      0           0.64706      0.36364
0.81818        0.375     0.42857       0.75        0.19355     0.55882      0.36364
0.12121        0.375     0.4881        0.5625      0.32258     0            0
0              0.375     0.035714      0           0.6129      0.029412     0.54545
0.24242        0         0             0.6875      0.51613     0.32353      0.090909
0.18182        0.5       1             1           0.48387     1            1
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

3.5.3. Métricas para problemas de regresión y clasificación.

Principales métricas para problemas de regresión:

• **R^2 (coeficiente de determinación):** Representa la proporción de la varianza de una variable dependiente que es posible explicar mediante una variable independiente o variables en un modelo de regresión.

Documentación: R2 Regression score function

Documentación: Regresión logística

Documentación: Regresión múltiple lineal

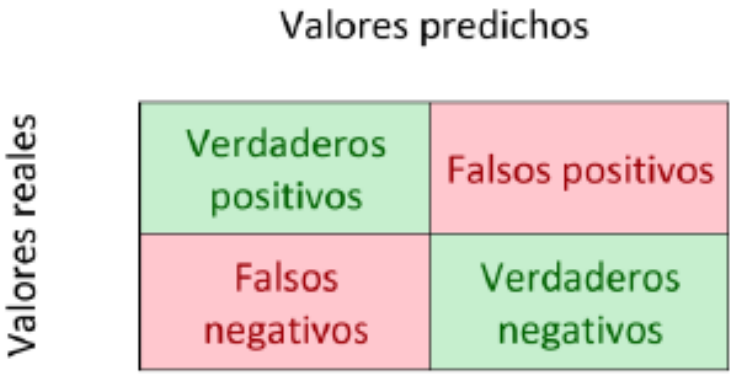
• **MSE:** El error cuadrático medio se calcula como la diferencia cuadrática media entre los valores estimados y los valores reales.

Documentación: MSE Mean squeared error regression loss

Principales métricas para problemas de clasificación:

• **Precisión de la clasificación:** Se puede definir como la ratio de predicciones correctas. Realmente sólo se emplea bien cuando hay el mismo número de observaciones en cada clase y los errores de predicción al equivocarnos al clasificar son igualmente importantes.
• **Área bajo la curva ROC:** Se emplea en problemas de clasificación binarios. Un área de 1 indica que el modelo ha podido clasificar todos los casos correctamente. Un área de 0.5 indica que el modelo es tan bueno como clasificar aleatoriamente.
• **Matriz de confusión:** Está indicado para representar la precisión de un modelo con dos o más clases.

Documentación: Confussion matrix



Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

3.5.4. Validación cruzada *k-Folds* y *leave one out*.

Validación cruzada *kFolds*:

• Se divide el subconjunto de training en un número k de particiones (2, 5, 10…).

• El modelo se entrena usando k-1 particiones, también llamadas *folds*, para training y 1 para test (que algunos autores denominan validación). Esto se repite hasta que todas las particiones han actuado como partición de test exactamente una vez (ver gráfico).

• Los indicadores de calidad del modelo se obtiene como media de esas repeticiones individuales.

• Para problemas de **clasificación** se usa una variante conocida como **validación cruzada estratificada** (por defecto en *Scikit-Learn* para problemas de clasificación) En ella, se dividen los datos de manera que las proporciones entre las clases sean las mismas en cada división que en todo el conjunto de datos original.

Documentación: Validación cruzada

Documentación: Validación cruzada estratificada

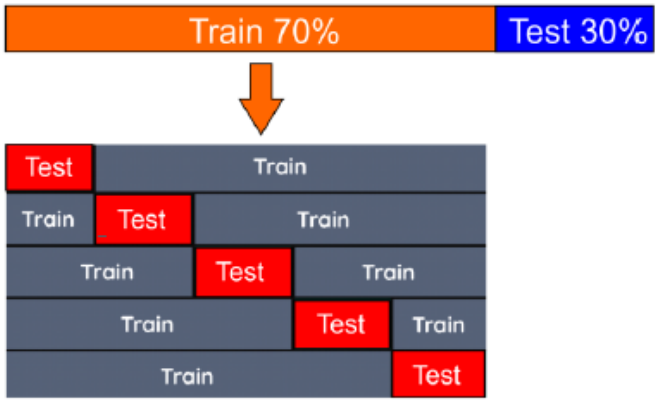
Validación cruzada *leave one out*:

• Similar a la anterior, pero en este caso hay tantas particiones como el número de observaciones en el dataset.

• Es computacionalmente mucho más costoso que la validación cruzada anteriormente expuesta.

• Se suele emplear cuando la cantidad de datos disponibles es pequeña.

Documentación: Validación cruzada *Leave one Out*



Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

Ejercicio 98_01. Validación cruzada con regresión en Python

```
# Este es un ejemplo simple para ilustrar la validación cruzada. Se debería estudiar
si es necesario realizar un pretratamiento de los datos, etc.
from pandas import read_csv
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
# Cargamos los datos:
# La variable objetivo será la estación 14, mientras que el resto de variables será
nuestras variables de entrada.
nomFichero = 'no2.csv'
cols = ['Estacion 1', 'Estacion 2', 'Estacion 3', 'Estacion 4', 'Estacion 5',\
        'Estacion 6', 'Estacion 7', 'Estacion 8', 'Estacion 9', 'Estacion 10',\
        'Estacion 11', 'Estacion 12', 'Estacion 13', 'Estacion 14']
datos = read_csv(nomFichero, names = cols)
variables = datos.to_numpy()
x = variables[:,0:13]
y = variables[:,13]
tamTest = 0.30
semillas = 20
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
# Establecemos el modelo:
# En este ejemplo usaremos regresión múltiple lineal.
modelo = LinearRegression()
# Validación cruzada:
# Usaremos 10-validacion cruzada aleatoria.<br>
# Establecemos el MSE como métrica y el número de hilos a usar para los cálculos.
particiones = 10 # Número de particiones
validación = KFold(n_splits=particiones, random_state = semillas, shuffle = True)
metrica = ('neg_mean_squared_error') # Error cuadrático medio
hilos_cores_a_usar =8; # Podemos modificarlo para usar hilos si disponibles
# Resultados en la etapa de entrenamiento:
print("Resultados de la etapa de entrenamiento:\n")
resultados = cross_val_score(modelo, xTrain, yTrain, cv = particiones,
            scoring = metrica, n_jobs = hilos_cores_a_usar)
# Hay que usar absoluto porque esta métrica en Python la da en negativo
print("MSE: %f - Desviación típica: %f" %(abs(resultados.mean()),
        resultados.std()))
metrica = 'r2'
resultados = cross_val_score(modelo, xTrain, yTrain, cv=particiones,
             scoring=metrica, n_jobs=hilos_cores_a_usar)
print("R2: %f - Desviación típica: %f" % (resultados.mean(),
        resultados.std()))
% # Resultados de la etapa de test:
modelo.fit(xTrain, yTrain)
yPredichos = modelo.predict(xTest)
print("Resultados de la etapa de test:\n")
print("R2:")
```

```
print(r2_score(yTest, yPredichos))
print("MSE:")
print(mean_squared_error(yTest, yPredichos))
```

Ejercicio 98_02. Validación cruzada con regresión en MATLAB

```
% Este es un ejemplo simple para ilustrar la validación cruzada.
% Se debería estudiar si es necesario realizar un pretratamiento de los datos, etc.
% Cargamos los datos:
% La variable objetivo será la estación 14, mientras que el resto de variables será
nuestras variables de entrada.
nomFichero = 'no2_2.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datos);
x = variables(:, 1:13);
y = variables(:, 14);
tamTest = 0.30;
semillas = 20;
% Se pasa 'rng' para que los resultados sean reproducibles en diferentes
% llamadas a 'datasample'
rng(semillas);
% Dividimos los datos en conjunto de entrenamiento y prueba
idx = datasample(1:size(variables, 1), round((1 - tamTest) * size(variables, 1)),
'Replace', false);
xTrain = x(idx, :);
yTrain = y(idx);
xTest = x(setdiff(1:size(variables, 1), idx), :);
yTest = y(setdiff(1:size(variables, 1), idx));
% Establecemos el modelo:
% En este ejemplo usaremos regresión múltiple lineal.
modelo = fitlm(xTrain, yTrain);
% Validación cruzada:
particiones = 10; % Número de particiones
cv = cvpartition(length(yTrain), 'KFold', particiones, 'Stratify', false);
metrica = 'mse'; % Error cuadrático medio
% Resultados en la etapa de entrenamiento:
disp('Resultados de la etapa de entrenamiento:');
% crossval('mcr',X1,X2,X3,y,'Predfun',@classf,'Stratify',y)
resultados = crossval('mse', xTrain, yTrain, 'Predfun', @(xTrain, yTrain, xTest,
yTest) predict(fitlm(xTrain, yTrain), xTest), 'partition', cv);
disp(['MSE: ', num2str(resultados)]);
% Calculamos R^2 manualmente:
resultados_r2 = zeros(particiones, 1);
for i = 1:particiones
    idx_train = training(cv, i);
    idx_test = test(cv, i);
        modelo_temporal = fitlm(xTrain(idx_train, :), yTrain(idx_train));
    yPredichos_temporal = predict(modelo_temporal, xTrain(idx_test, :));
        resultados_r2(i) = 1 - sum((yTrain(idx_test) - yPredichos_temporal).^2) /
sum((yTrain(idx_test) - mean(yTrain(idx_train))).^2);
end
disp(['R^2: ', num2str(mean(resultados_r2))]);
% Resultados de la etapa de test:
yPredichos = predict(modelo, xTest);
r2_test = 1 - sum((yTest - yPredichos).^2) / sum((yTest - mean(yTrain)).^2);
```

```
disp('Resultados de la etapa de test:');
disp(['R^2: ', num2str(r2_test)]);
disp(['MSE: ', num2str(mean((yTest - yPredichos).^2))]);

Resultados de la etapa de entrenamiento:
MSE: 19.2685
R^2: 0.81102
Resultados de la etapa de test:
R^2: 0.8214
MSE: 13.0131
```

Ejercicio 99_01. Clasificación con validación *leave one out* con regresión logística en Python

```
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
# Cargamos los datos:
# La variable objetivo será la variable 8 "clase", mientras que el resto de variables
será nuestras variables de entrada.
nomFichero = 'pima-indians-diabetes.data.csv'
cols = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
datos = read_csv(nomFichero, names=cols)
variables  = datos.to_numpy()
x = variables[:,0:7] # Variables 0 a 7
y = variables[:,8] # Variable 8
# División entrenamiento-test:
# Usaremos 70% para entrenamiento y 30% para test, obtenidos aleatoriamente.
# Usamos semillas y ramdon_state para que los resultados sean reproducibles.
tamTest = 0.3
semillas = 23
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
% # Establecemos el modelo:
% # Al ser un ejemplo de clasificación, usaremos regresión logística.
modelo = LogisticRegression(solver='liblinear')
% # Validación cruzada leave one out:
% # Usaremos 10 particiones.<br>
% # Establecemos el número de hilos a usar para los cálculos.
validacion = LeaveOneOut()
hilos_cores_a_usar = 8; # Podemos modificarlo para usar + hilos si disponibles
particiones = 10;
% # Resultados en la etapa de entrenamiento:
% # Como métricas, usaremos la precisión y el área bajo la curva ROC.
print("Resultados de la etapa de entrenamiento:\n")
metrica = 'accuracy'
resultados = cross_val_score(modelo, xTrain, yTrain, cv = particiones,
             scoring = metrica, n_jobs = hilos_cores_a_usar)
print("Precisión validación cruzada: %f - Desviación típica: %f" %(resultados.mean(),
```

```
        resultados.std()))
metrica = 'roc_auc'
resultados = cross_val_score(modelo, xTrain, yTrain, cv=particiones,
             scoring=metrica, n_job=hilos_cores_a_usar)
print("Área bajo la curva ROC validación cruzada: %f - Desviación típica: %f"
      %(resultados.mean(), resultados.std()))
```

```
Resultados de la etapa de entrenamiento:

Precisión validación cruzada: 0.763452 – Desviación típica: 0.050628
Área bajo la curva ROC validación cruzada: 0.824573 – Desviación típica: 0.040479
```

```
% # Resultados de la etapa de test:
% # Mostramos la precisión, el área bajo la curva ROC y la matriz de confusión.
modelo.fit(xTrain, yTrain)
yPredichos =  modelo.predict(xTest)
print("Resultados de la etapa de test:\n")
precision =  accuracy_score(yTest, yPredichos)
print("Precisión de test: %.f %%" %(precision*100.0))
aRoc = roc_auc_score(yTest, yPredichos)
print("Área bajo la curva ROC test: %f" %(aRoc))
matriz = confusion_matrix(yTest, yPredichos)
print("Matriz de confusión test:")
print(matriz)
```

```
Resultados de la etapa de test:

Precisión de test: 78%
Área bajo la curva ROC test: 0.733190
Matriz de confusión test:
[[135  13]
[37  46]]
```

Ejercicio 99_02. Clasificación con validación *leave one out* con regresión logística en MATLAB

```
% Cargamos los datos:
nomFichero = 'pima-indians-diabetes.data.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datos);
X = variables(:, 1:8); % Variables 1 a 8
y = variables(:, 9);   % Variable 9
% División entrenamiento-test:
tamTest = 0.3;
semillas = 23;
rng(semillas);
[idxTrain, idxTest] = datasample(1:size(variables, 1), round((1 - tamTest) *
size(variables, 1)), 'Replace', false);
xTrain = X(idxTrain, :);
yTrain = y(idxTrain);
xTest = X(idxTest, :);
yTest = y(idxTest);
% Validación Leave-One-Out:
numMuestras = length(yTrain);
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
predicciones = zeros(numMuestras, 1);
% Validación Leave-One-Out
num_muestras = length(y);
for i = 1:num_muestras
    % Dejar una muestra fuera para validación
    X_validacion = X(i, :);
    y_validacion = y(i);
    X_entrenamiento = X([1:i-1, i+1:end], :);
    y_entrenamiento = y([1:i-1, i+1:end]);
    % Ajustar el modelo de regresión logística
    modelo = fitglm(X_entrenamiento, y_entrenamiento, 'Distribution', 'binomial',
'Link', 'logit');
    % Realizar predicciones para la muestra dejada fuera
    y_predicho = predict(modelo, X_validacion);
    predicciones(i) = y_predicho;
    etiquetas_verdaderas(i) = y_validacion;
end
disp(modelo);
% Calcular el área bajo la curva ROC (AUC)
[X_roc, Y_roc, ~, AUC] = perfcurve(etiquetas_verdaderas, predicciones, 1);
disp(['Área bajo la curva ROC: ', num2str(AUC)]);
% Graficar la curva ROC
figure;
plot(X_roc, Y_roc);
xlabel('Tasa de Falsos Positivos');
ylabel('Tasa de Verdaderos Positivos');
title('Curva ROC');
% Realizar predicciones en el conjunto de prueba
yPredichos = predict(modelo, xTest);
% Realizar predicciones en el conjunto de prueba
yPredichos = predict(modelo, xTest);
% Convertir las probabilidades a etiquetas binarias usando un umbral (por ejemplo,
0.5)
umbral = 0.5;
etiquetas_binarias = zeros(size(yPredichos));  % Inicializar un vector de ceros
% Aplicar el umbral y asignar etiquetas binarias
for i = 1:length(yPredichos)
    if yPredichos(i) >= umbral
        etiquetas_binarias(i) = 1;
    end
end
% Calcular la precisión en el conjunto de prueba
precision = sum(etiquetas_binarias == yTest) / length(yTest);
disp(['Precisión de test: ', num2str(precision * 100), '%']);

modelo = Generalized linear regression model:
    logit(y) ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8
    Distribution = Binomial
Estimated Coefficients:
                    Estimate        SE         tStat        pValue
                   ________    _________    ________    __________

   (Intercept)        -8.3979      0.71665     -11.718    1.0286e-31
    x1                0.12302     0.032074      3.8354    0.00012534
    x2               0.035143    0.0037082      9.4771    2.6154e-21
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
    x3              -0.013283    0.0052325     -2.5386      0.011131
    x4             0.00066315    0.0068997    0.096112       0.92343
    x5              -0.0011963    0.0009012     -1.3274       0.18436
    x6               0.089617     0.015085      5.9409     2.835e-09
    x7                 0.9439      0.29908       3.156     0.0015992
    x8               0.014843    0.0093334      1.5903       0.11177
767 observations, 758 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 269, p-value = 1.37e-53

Área bajo la curva ROC: 0.82982
Precisión de test: 76.3941%
```