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

3.8. Regresión con Redes Neuronales Artificiales

Ejercicio 103_01. Regresión con redes neuronales en Python

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from scipy.stats import pearsonr
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Leer el archivo Excel
filename = 'datos.xlsx'  # Reemplaza con el nombre de tu archivo
data = pd.read_excel(filename)

# Suponer que la última columna es la variable dependiente (y)
# y el resto de las columnas son las características de entrada (X)
X = data.iloc[:, :-1].values  # Entradas
y = data.iloc[:, -1].values   # Salidas (variable dependiente)

# Parámetros de la validación cruzada
K_outer = 5  # Número de pliegues para la validación cruzada externa
K_inner = 5  # Número de pliegues para la validación cruzada interna

# Función para construir la red neuronal
def build_model(input_dim, num_neurons, num_layers):
    model = Sequential()
    model.add(Dense(num_neurons, input_dim=input_dim, activation='relu'))
    for _ in range(num_layers - 1):
        model.add(Dense(num_neurons, activation='relu'))
    model.add(Dense(1))  # Salida para regresión (una sola salida)
    model.compile(optimizer='adam', loss='mse')  # Usamos MSE como función de
pérdida
    return model

# Inicializar listas para almacenar los resultados
mse_vals = []
mape_vals = []
r_vals = []

# Validación cruzada externa
outer_cv = KFold(n_splits=K_outer, shuffle=True, random_state=42)

for train_idx, test_idx in outer_cv.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Validación cruzada interna para ajustar el modelo
    inner_cv = KFold(n_splits=K_inner)
    best_mse_inner = np.inf
    best_params = {}
```

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

```
    for num_neurons in [10, 20, 50]:  # Número de neuronas a probar
        for num_layers in [1, 2]:  # Número de capas ocultas a probar
            mse_inner_list = []

            # Validación cruzada interna
            for inner_train_idx, inner_val_idx in inner_cv.split(X_train):
                X_train_inner, X_val_inner = X_train[inner_train_idx],
X_train[inner_val_idx]
                y_train_inner, y_val_inner = y_train[inner_train_idx],
y_train[inner_val_idx]

                # Construir y entrenar el modelo
                model = build_model(X_train.shape[1], num_neurons, num_layers)
                model.fit(X_train_inner, y_train_inner, epochs=100, batch_size=32,
verbose=0)
                # Predecir en el conjunto de validación interno
                y_pred_val = model.predict(X_val_inner).flatten()

                # Calcular MSE en el conjunto de validación interno
                mse_inner = mean_squared_error(y_val_inner, y_pred_val)
                mse_inner_list.append(mse_inner)

            # Calcular MSE promedio en la validación cruzada interna
            mean_mse_inner = np.mean(mse_inner_list)

            # Guardar el mejor modelo basado en MSE
            if mean_mse_inner < best_mse_inner:
                best_mse_inner = mean_mse_inner
                best_params = {'num_neurons': num_neurons, 'num_layers':
num_layers}

    # Entrenar el mejor modelo en el conjunto de entrenamiento externo
    best_model = build_model(X_train.shape[1], best_params['num_neurons'],
best_params['num_layers'])
    best_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

    # Predecir en el conjunto de prueba externo
    y_pred_test = best_model.predict(X_test).flatten()

    # Calcular las métricas: MSE, MAPE y coeficiente de correlación (r)
    mse = mean_squared_error(y_test, y_pred_test)
    mape = mean_absolute_percentage_error(y_test, y_pred_test)
    r, _ = pearsonr(y_test, y_pred_test)

    # Guardar los resultados
    mse_vals.append(mse)
    mape_vals.append(mape)
    r_vals.append(r)

    print(f"Precisión para este pliegue externo - MSE: {mse:.4f}, MAPE: {mape *
100:.2f}%, r: {r:.4f}")

# Mostrar las métricas promedio en todos los pliegues externos
mean_mse = np.mean(mse_vals)
mean_mape = np.mean(mape_vals)
```

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

```
mean_r = np.mean(r_vals)
print(f"MSE promedio: {mean_mse:.4f}")
print(f"MAPE promedio: {mean_mape * 100:.2f}%")
```

Explicación:

1. **Lectura del archivo Excel**:

```
data = pd.read_excel(filename)
X = data.iloc[:, :-1].values  # Entradas
y = data.iloc[:, -1].values   # Salidas (variable dependiente)
```

- Leemos los datos desde un archivo Excel. Las primeras columnas son las características (**X**), y la última columna es la variable dependiente o valor a predecir (**y**).

2. **Definición de la red neuronal**:

```
def build_model(input_dim, num_neurons, num_layers):
    model = Sequential()
    model.add(Dense(num_neurons, input_dim=input_dim, activation='relu'))
    for _ in range(num_layers - 1):
        model.add(Dense(num_neurons, activation='relu'))
    model.add(Dense(1))  # Salida para regresión
    model.compile(optimizer='adam', loss='mse')  # Usamos MSE como función de
    pérdida
    return model
```

- La función **build_model** define una red neuronal completamente conectada con un número variable de neuronas y capas ocultas. La salida tiene una sola neurona para regresión. Utilizamos **adam** como optimizador y **MSE** como función de pérdida.

3. **Validación cruzada externa**:

```
outer_cv = KFold(n_splits=K_outer, shuffle=True, random_state=42)
```

- Usamos **KFold** para crear la validación cruzada externa. Se dividen los datos en K_outer (5) particiones. En cada iteración, entrenamos y evaluamos en diferentes subconjuntos de datos.

4. **Validación cruzada interna**:

- Dentro de cada trozo externo, realizamos una validación cruzada interna con K_inner (5) particiones, donde ajustamos los hiperparámetros del modelo (número de neuronas y capas ocultas):

```
inner_cv = KFold(n_splits=K_inner)
```

- Para cada combinación de **num_neurons** y **num_layers**, se entrena el modelo en los datos de entrenamiento interno y se evalúa en los datos de validación internos.

5. **Entrenamiento y evaluación del mejor modelo**:

```
best_model = build_model(X_train.shape[1], best_params['num_neurons'],
best_params['num_layers'])
best_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

- Después de encontrar la mejor arquitectura, entrenamos el modelo en el conjunto de entrenamiento externo y evaluamos el rendimiento en el conjunto de prueba externo.

6. **Cálculo de las métricas**:

```
mse = mean_squared_error(y_test, y_pred_test)
```

```
mape = mean_absolute_percentage_error(y_test, y_pred_test)
r, _ = pearsonr(y_test, y_pred_test)
```

- Calculamos las métricas de evaluación:
  - **MSE (Error Cuadrático Medio)**: mide el error promedio al cuadrado entre los valores reales y predichos.
  - **MAPE (Error Porcentual Absoluto Medio)**: mide el error promedio en porcentaje entre los valores reales y predichos.
  - **Coeficiente de correlación (r)**: mide la correlación entre los valores reales y predichos

7. **Resultados finales**:

```
mean_mse = np.mean(mse_vals)
mean_mape = np.mean(mape_vals)
mean_r = np.mean(r_vals)
```

Librerías necesarias:
Para ejecutar este código, necesitas instalar las siguientes librerías:

```
pip install pandas scikit-learn tensorflow openpyxl
```

Ejercicio 103_02. Regresión con redes neuronales en MATLAB

```
function [mse_mean, r2_mean, mae_mean] = regresion_lineal_cv(X, y, k)
% REGRESION_LINEAL_CV Realiza regresión lineal múltiple con validación cruzada k-
fold
%
%   Entrada:
%       X: Matriz de características (cada fila es una observación, cada columna
una característica)
%       y: Vector de valores objetivo
%       k: Número de folds para la validación cruzada
%
%   Salida:
%       mse_mean: Error cuadrático medio promedio
%       r2_mean: Coeficiente de determinación promedio
%       mae_mean: Error absoluto medio promedio

    % Crear un objeto de partición k-fold
    cv = cvpartition(size(X,1),'KFold',k);

    % Inicializar vectores para almacenar las métricas
    mse_scores = zeros(k,1);
    r2_scores = zeros(k,1);
    mae_scores = zeros(k,1);

    % Iterar sobre cada fold
    for i = 1:k
        % Obtener índices de entrenamiento y prueba
        idx = cv.test(i);
        X_train = X(~idx,:);
        y_train = y(~idx);
        X_test = X(idx,:);
        y_test = y(idx);
```

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

```
        % Entrenar el modelo
        mdl = fitlm(X_train, y_train);

        % Hacer predicciones
        y_pred = predict(mdl, X_test);

        % Calcular métricas
        mse_scores(i) = mean((y_test - y_pred).^2);
        r2_scores(i) = 1 - sum((y_test - y_pred).^2) / sum((y_test -
        mean(y_test)).^2);
        mae_scores(i) = mean(abs(y_test - y_pred));
    end

    % Calcular los promedios de las métricas
    mse_mean = mean(mse_scores);
    r2_mean = mean(r2_scores);
    mae_mean = mean(mae_scores);
end
```

Cómo usar la función:

1. **Cargar los datos:** Carga tus datos en matrices X (características) e y (variable objetivo).
2. **Definir el número de folds:** Especifica el valor de k.
3. **Llamar a la función:** Llama a la función regresion_lineal_cv con tus datos y el valor de k.
4. **Interpretar los resultados:** La función devolverá el error cuadrático medio promedio, el coeficiente de determinación promedio y el error absoluto medio promedio.

```
% Cargar datos (suponiendo que ya están cargados en X e y)
k = 5;

% Llamar a la función
[mse_mean, r2_mean, mae_mean] = regresion_lineal_cv(X, y, k);

% Mostrar resultados
fprintf('Error cuadrático medio promedio: %.4f\n', mse_mean);
fprintf('Coeficiente de determinación promedio: %.4f\n', r2_mean);
fprintf('Error absoluto medio promedio: %.4f\n', mae_mean);
```

Explicación paso a paso:

1. **Lectura de los datos:**

```
data = readtable(filename);
X = data{:, 1:end-1};  % Entradas
y = data{:, end};      % Variable dependiente (salida)
```

   - Leemos los datos desde un archivo Excel. Las primeras columnas son las características (**X**) y la última columna es la variable dependiente o valor a predecir (**y**).

2. **Definición de los parámetros de validación cruzada:**

```
K_outer = 5;  % trozos externos
K_inner = 5;  % trozos internos
cv_outer = cvpartition(y, 'KFold', K_outer);
```

- Usamos **cvpartition** para crear la validación cruzada externa y dividir los datos en 5 pliegues. La misma estructura se usa en la validación cruzada interna.

3. **Definición de la red neuronal**:

```
net = feedforwardnet(repmat(num_neurons, 1, num_layers), trainFcn);
net.trainParam.epochs = maxEpochs;
net.trainParam.showWindow = false;
```

- Utilizamos **feedforwardnet** para crear una red neuronal de propagación hacia adelante.
- Probamos diferentes configuraciones de la red con diferentes números de **neuronas** y **capas ocultas**. La función de entrenamiento que usamos es **adam**.

4. **Validación cruzada interna**:

```
for j = 1:K_inner
    X_train_inner = X_train(training(cv_inner, j), :);
    y_train_inner = y_train(training(cv_inner, j), :);
    X_val_inner = X_train(test(cv_inner, j), :);
    y_val_inner = y_train(test(cv_inner, j), :);

    % Entrenar la red neuronal en los datos internos
    net_inner = train(net, X_train_inner', y_train_inner');

    % Predecir en el conjunto de validación interna
    y_pred_inner = net_inner(X_val_inner')';

    % Calcular el MSE interno
    mse_inner_vals(j) = mean((y_val_inner - y_pred_inner).^2);
end
```

- Para cada partición de la validación cruzada interna, entrenamos el modelo con una arquitectura específica de la red neuronal y calculamos el **Error Cuadrático Medio (MSE)** en los datos de validación interna.

5. **Entrenamiento del mejor modelo**:

```
best_net = feedforwardnet(best_layers, trainFcn);
best_net.trainParam.epochs = maxEpochs;
best_net = train(best_net, X_train', y_train');
```

- Después de encontrar la mejor arquitectura de la red neuronal en la validación interna, entrenamos el modelo en el conjunto de entrenamiento externo.

6. **Evaluación en el conjunto de prueba externo**:

```
y_pred_test = best_net(X_test')';
mse_vals(i) = mean((y_test - y_pred_test).^2);  % MSE
mape_vals(i) = mape(y_test, y_pred_test);       % MAPE
r_vals(i) = corr(y_test, y_pred_test);          % r
```

- Predecimos los valores en los datos de prueba externos y calculamos las métricas de evaluación: **MSE**, **MAPE** y **coeficiente de correlación (r)**.

7. **Resultados finales**:

- Se calculan los valores medios