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

## 3.11. Clasificación con Redes Neuronales Artificiales

**Ejercicio 106_01. Clasificación con ANNs en Python**

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import numpy as np

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

# Suponer que la última columna es la etiqueta de clase (y)
# y el resto de las columnas son las entradas (X)
X = data.iloc[:, :-1].values  # Entradas
y = data.iloc[:, -1].values   # Etiquetas de clase

# Codificar las etiquetas de clase
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)  # Convertir a formato de clasificación

# Definir los parámetros de validación cruzada
K_outer = 5  # Número de pliegues para la validación cruzada externa
K_inner = 5  # Número de pliegues para la validación cruzada interna

# Validación cruzada externa
outer_cv = StratifiedKFold(n_splits=K_outer, shuffle=True, random_state=42)
outer_scores = []

# Función para construir la red neuronal
def build_model(input_dim, num_neurons, num_layers, output_dim):
    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(output_dim, activation='softmax'))  # Capa de salida
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

# Para cada pliegue de la validación cruzada externa
for train_idx, test_idx in outer_cv.split(X, y_encoded):
    # Conjunto de entrenamiento y prueba para la validación cruzada externa
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y_categorical[train_idx], y_categorical[test_idx]
```

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

```
    # Validación cruzada interna para ajustar la red neuronal
    inner_cv = StratifiedKFold(n_splits=K_inner)
    best_accuracy_inner = 0
    best_params = {}

    # Probar diferentes configuraciones de la red neuronal
    for num_neurons in [10, 20, 50]:  # Número de neuronas a probar
        for num_layers in [1, 2]:  # Número de capas ocultas a probar
            inner_scores = []

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

                # Construir y entrenar el modelo
                model = build_model(input_dim=X_train.shape[1],
num_neurons=num_neurons, num_layers=num_layers, output_dim=y_categorical.shape[1])
                model.fit(X_train_inner, y_train_inner, epochs=100, batch_size=32,
verbose=0)

                # Evaluar el modelo en el conjunto de validación interno
                loss, accuracy = model.evaluate(X_val_inner, y_val_inner,
verbose=0)
                inner_scores.append(accuracy)

            # Calcular la precisión promedio en la validación cruzada interna
            mean_inner_accuracy = np.mean(inner_scores)

            # Guardar el mejor modelo basado en la precisión
            if mean_inner_accuracy > best_accuracy_inner:
                best_accuracy_inner = mean_inner_accuracy
                best_params = {'num_neurons': num_neurons, 'num_layers':
num_layers}

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

    # Evaluar el modelo en el conjunto de prueba externo
    y_pred_test = best_model.predict(X_test)
    y_pred_test_labels = np.argmax(y_pred_test, axis=1)
    y_test_labels = np.argmax(y_test, axis=1)
    outer_accuracy = accuracy_score(y_test_labels, y_pred_test_labels)
    outer_scores.append(outer_accuracy)

    print(f"Precisión para este pliegue externo: {outer_accuracy * 100:.2f}%")

# Promedio de la precisión en los pliegues externos
```

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

```
mean_accuracy = np.mean(outer_scores)
print(f"Precisión promedio en la validación cruzada externa: {mean_accuracy *
100:.2f}%")

# Matriz de confusión del último pliegue
cm = confusion_matrix(y_test_labels, y_pred_test_labels)
print("Matriz de confusión:\n", cm)
```

**Explicación:**

1. **Lectura de los datos**:

```
data = pd.read_excel(filename)
X = data.iloc[:, :-1].values  # Entradas
y = data.iloc[:, -1].values   # Etiquetas de clase
```

   - Leemos los datos desde un archivo Excel, donde las primeras columnas son las características (**X**) y la última columna contiene las etiquetas de clase (**y**).

2. **Codificación de las etiquetas de clase**:

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)
```

   - Utilizamos **LabelEncoder** para transformar las etiquetas de clase en números enteros y **to_categorical** para convertir estas etiquetas a formato **one-hot encoding**, que es necesario para las redes neuronales en clasificación multiclase.

3. **Definición de la validación cruzada**:

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

   - Usamos **StratifiedKFold** para crear una validación cruzada estratificada externa, dividiendo los datos en 5 pliegues, asegurando que la proporción de clases sea similar en cada pliegue.

4. **Construcción de la red neuronal**:

```
def build_model(input_dim, num_neurons, num_layers, output_dim):
    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(output_dim, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model
```

   - Definimos una función **build_model** para construir una red neuronal con **input_dim** (dimensión de las entradas), **num_neurons** (número de neuronas en las capas ocultas), **num_layers** (número de capas ocultas) y **output_dim** (número de clases de salida).

5. **Validación cruzada interna**:

```
inner_cv = StratifiedKFold(n_splits=K_inner)
```

   - Para cada pliegue externo, realizamos una validación cruzada interna (5 pliegues) para ajustar el modelo probando diferentes configuraciones de la red neuronal.

6. **Entrenamiento y evaluación del modelo**:

- Para cada combinación de **num_neurons** y **num_layers**, entrenamos el modelo en los datos de entrenamiento interno y lo validamos en los datos de validación internos.
- Guardamos la mejor configuración del modelo basado en la precisión.

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

```
y_pred_test = best_model.predict(X_test)
y_pred_test_labels = np.argmax(y_pred_test, axis=1)
outer_accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred_test_labels)
```

- Una vez que encontramos el mejor modelo, lo entrenamos en el conjunto de entrenamiento externo y lo evaluamos en el conjunto de prueba externo.

8. **Precisión promedio de la validación cruzada externa**:

```
mean_accuracy = np.mean(outer_scores)
print(f"Precisión promedio en la validación cruzada externa: {mean_accuracy *
100:.2f}%")
```

**Librerías necesarias:**
Para ejecutar este código en Python, asegúrate de instalar las siguientes librerías:

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

**Resumen:**
Este código implementa la clasificación con **redes neuronales artificiales (RNA)** usando doble validación cruzada, ajustando los hiperparámetros de la red en la validación cruzada interna y evaluando su rendimiento en la validación.

**Ejercicio 106_02. Regresión con ANNs en MATLAB**

```
% Nombre del archivo Excel
filename = 'datos.xlsx'; % Reemplaza con tu archivo Excel

% Leer los datos del archivo Excel
data = readtable(filename);

% Suponer que la última columna es la etiqueta de clase (clase)
% y el resto de columnas son las características de entrada (entradas)
entradas = data{:, 1:end-1}; % Todas las columnas menos la última (entradas)
clase = data{:, end};        % Última columna (etiquetas de clase)

% Convertir las etiquetas de clase a formato categórico si no lo están
clase = categorical(clase);

% 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

% Crear partición de la validación cruzada externa
cv_outer = cvpartition(clase, 'KFold', K_outer);

% Inicializar variables para guardar los resultados
```

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

```
accuracy_vals = zeros(K_outer, 1);  % Para almacenar la precisión en cada pliegue
externo

% Opciones para el entrenamiento de la red neuronal
options = trainingOptions('adam', ...  % Algoritmo de optimización
    'MaxEpochs', 100, ...              % Número máximo de épocas
    'MiniBatchSize', 32, ...           % Tamaño del batch
    'Shuffle', 'every-epoch', ...      % Barajar los datos en cada época
    'Verbose', false, ...
    'Plots', 'training-progress');     % Mostrar progreso del entrenamiento

% Loop de validación cruzada externa
for i = 1:K_outer
    % Dividir los datos en conjunto de entrenamiento y prueba externo
    X_train = entradas(training(cv_outer, i), :);
    y_train = clase(training(cv_outer, i), :);
    X_test = entradas(test(cv_outer, i), :);
    y_test = clase(test(cv_outer, i), :);

    % Validación cruzada interna para ajustar la red neuronal (ajuste de
hiperparámetros)
    cv_inner = cvpartition(y_train, 'KFold', K_inner);
    best_accuracy_inner = 0;  % Inicializar la mejor precisión interna
    best_layers = [];         % Guardar la mejor arquitectura de la red

    % Probar diferentes configuraciones de la red neuronal
    for num_neurons = [10, 20, 50]  % Número de neuronas a probar
        for num_layers = [1, 2]  % Número de capas ocultas a probar
            accuracy_inner = zeros(K_inner, 1);  % Para almacenar la precisión
interna

            % Definir la arquitectura de la red neuronal
            layers = [
                featureInputLayer(size(X_train, 2))  % Capa de entrada
                fullyConnectedLayer(num_neurons)     % Primera capa oculta
            ];

            % Si se especifica más de una capa oculta
            if num_layers == 2
                layers = [
                    layers
                    fullyConnectedLayer(num_neurons)  % Segunda capa oculta
                ];
            end

            % Agregar la capa de salida
            layers = [
                layers
                fullyConnectedLayer(numel(categories(y_train)))  % Capa de salida
(número de clases)
                softmaxLayer
                classificationLayer
            ];

            % Validación cruzada interna
```

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

```
            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 = trainNetwork(X_train_inner, y_train_inner, layers, options);

                % Predecir en el conjunto de validación interna
                y_pred_inner = classify(net, X_val_inner);

                % Calcular la precisión interna
                accuracy_inner(j) = sum(y_pred_inner == y_val_inner) /
length(y_val_inner);
            end

            % Calcular la precisión promedio en la validación cruzada interna
            mean_accuracy_inner = mean(accuracy_inner);

            % Actualizar la mejor configuración de la red si la precisión mejora
            if mean_accuracy_inner > best_accuracy_inner
                best_accuracy_inner = mean_accuracy_inner;
                best_layers = layers;  % Guardar la mejor arquitectura
            end
        end
    end

    % Entrenar el mejor modelo encontrado en el conjunto de entrenamiento externo
    best_net = trainNetwork(X_train, y_train, best_layers, options);

    % Predecir en los datos de prueba externos
    y_pred_test = classify(best_net, X_test);

    % Calcular la precisión en los datos de prueba externos
    accuracy_vals(i) = sum(y_pred_test == y_test) / length(y_test);
end

% Mostrar el valor promedio de precisión de la validación cruzada externa
mean_accuracy = mean(accuracy_vals);
disp(['Precisión promedio en la validación cruzada externa: ',
num2str(mean_accuracy * 100), '%']);
```

**Explicación paso a paso:**

1. **Lectura de los datos**:

```
data = readtable(filename);
entradas = data{:, 1:end-1}; % Entradas
clase = data{:, end};        % Etiquetas de clase
clase = categorical(clase);  % Convertir las etiquetas a formato categórico
```

   - Leemos los datos desde un archivo Excel, donde las primeras columnas son las características (entradas) y la última columna es la etiqueta de clase.

- Las etiquetas de clase se convierten en **categóricas**, ya que las redes neuronales de MATLAB suelen trabajar mejor con etiquetas categóricas para clasificación.

2. **Definición de la validación cruzada**:

```
K_outer = 5;  % Validación cruzada externa
K_inner = 5;  % Validación cruzada interna
cv_outer = cvpartition(clase, 'KFold', K_outer);
```

- **K_outer**: Número de pliegues para la validación cruzada externa (5 en este caso).
- **K_inner**: Número de pliegues para la validación cruzada interna.
- **cvpartition**: Genera las particiones para la validación cruzada externa. En cada iteración, se divide el conjunto de datos en entrenamiento y prueba de forma estratificada.

3. **Opciones para el entrenamiento**:

```
options = trainingOptions('adam', 'MaxEpochs', 100, 'MiniBatchSize', 32, 'Shuffle',
'every-epoch', 'Verbose', false, 'Plots', 'training-progress');
```

- Usamos **trainingOptions** para definir los parámetros de entrenamiento de la red neuronal. En este caso, usamos el optimizador **Adam**, con un número máximo de épocas de 100 y un tamaño de batch de 32.

4. **Loop de validación cruzada externa**:

```
for i = 1:K_outer
    X_train = entradas(training(cv_outer, i), :);
    y_train = clase(training(cv_outer, i), :);
    X_test = entradas(test(cv_outer, i), :);
    y_test = clase(test(cv_outer, i), :);
```

- En cada iteración de la validación cruzada externa, se dividen los datos en conjuntos de **entrenamiento** y **prueba externos**.

5. **Validación cruzada interna para ajustar la red neuronal**:

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

- Dentro del bucle externo, se realiza una validación cruzada interna para ajustar los hiperparámetros de la red neuronal, como el **número de neuronas** y el **número de capas ocultas**.
- El **bucle interno** prueba diferentes configuraciones de la red, entrenando en los datos de entrenamiento internos y validando en los datos de validación internos.

6. **Entrenar el modelo con la mejor configuración**:

```
best_net = trainNetwork(X_train, y_train, best_layers, options);
y_pred_test = classify(best_net, X_test);
```

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

7. **Precisión promedio**:

```
mean_accuracy = mean(accuracy_vals);
disp(['Precisión promedio en la validación cruzada externa: ',
num2str(mean_accuracy * 100), '%']);
```

- Calculamos y mostramos la **precisión promedio** de todos los pliegues externos.