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

3.10. Clasificación con validación cruzada

Ejercicio 105_01. Clasificación con 2-CV en Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# 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 etiqueta de clase (y)
# y el resto de las columnas son las entradas (X)
X = data.iloc[:, :-1]  # Todas las columnas menos la última (entradas)
y = data.iloc[:, -1]   # Última columna (etiquetas de clase)

# Dividir los datos en conjunto de entrenamiento y prueba (80% - 20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Definir el valor de k (número de vecinos)
k = 5  # Puedes ajustar este valor según tu problema

# Crear el modelo k-NN
knn = KNeighborsClassifier(n_neighbors=k)

# Entrenar el modelo
knn.fit(X_train, y_train)

# Realizar predicciones con el conjunto de prueba
y_pred = knn.predict(X_test)

# Evaluar la precisión del modelo
precision = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo k-NN: {precision * 100:.2f}%")

# Mostrar la matriz de confusión
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()

# Mostrar la matriz de confusión
plt.title('Matriz de confusión para el k-NN')
plt.show()import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
```

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

```
import matplotlib.pyplot as plt

# 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 etiqueta de clase (y)
# y el resto de las columnas son las entradas (X)
X = data.iloc[:, :-1]  # Todas las columnas menos la última (entradas)
y = data.iloc[:, -1]   # Última columna (etiquetas de clase)

# 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
k_values = [1, 3, 5, 7, 9]  # Diferentes valores de k para probar

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

# Validación cruzada interna con búsqueda de parámetros (grid search)
inner_cv = StratifiedKFold(n_splits=K_inner)

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

    # Definir el modelo k-NN y la búsqueda de los mejores parámetros (valor óptimo
de k)
    knn = KNeighborsClassifier()
    param_grid = {'n_neighbors': k_values}

    # Usar GridSearchCV para encontrar el mejor valor de k con validación cruzada
interna
    grid_search = GridSearchCV(knn, param_grid, cv=inner_cv, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Mejor valor de k encontrado
    best_k = grid_search.best_params_['n_neighbors']
    print(f"Mejor valor de k para este pliegue: {best_k}")

    # Evaluar el modelo con el mejor k en el conjunto de prueba externo
    y_pred = grid_search.best_estimator_.predict(X_test)
    outer_accuracy = accuracy_score(y_test, y_pred)
    outer_scores.append(outer_accuracy)
    print(f"Precisión para este pliegue externo: {outer_accuracy * 100:.2f}%")

# Promedio de la precisión en los pliegues externos
mean_accuracy = sum(outer_scores) / K_outer
print(f"Precisión promedio en la validación cruzada externa: {mean_accuracy *
100:.2f}%")

# Mostrar la matriz de confusión para el último pliegue (ejemplo)
```

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Matriz de confusión para el k-NN (último pliegue)')
plt.show()
```

Explicación:

1. **Lectura del archivo Excel**:

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

   - Se cargan los datos del archivo Excel, donde las primeras columnas son las entradas (**X**) y la última columna es la etiqueta de clase (**y**).

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

```
K_outer = 5  # Pliegues externos
K_inner = 5  # Pliegues internos
k_values = [1, 3, 5, 7, 9]  # Valores de k para probar
```

   - Definimos los parámetros de la validación cruzada interna y externa. Usamos 5 pliegues tanto para la validación externa como para la interna.
   - **k_values**: Los valores de k que queremos probar para el algoritmo de **k-NN.**

3. **Validación cruzada externa**:

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

   - Usamos **StratifiedKFold** para crear una partición estratificada para la validación cruzada externa. En cada pliegue externo, el conjunto de datos se divide en entrenamiento y prueba de manera estratificada, asegurando que la proporción de clases sea similar en cada partición.

4. **validación cruzada externa**:

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

   - Para cada pliegue externo, dividimos los datos en conjunto de **entrenamiento externo** (X_train, y_train) y **prueba externa** (X_test, y_test).

5. **Validación cruzada interna y búsqueda de parámetros**:

```
knn = KNeighborsClassifier()
param_grid = {'n_neighbors': k_values}
grid_search = GridSearchCV(knn, param_grid, cv=inner_cv, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

   - Dentro del bucle externo, usamos **GridSearchCV** para hacer la búsqueda de los mejores parámetros (k) en el conjunto de entrenamiento externo usando validación cruzada interna.
   - **param_grid**: Definimos los valores de k que queremos probar (por ejemplo, k = 1, 3, 5, 7, 9).
   - **GridSearchCV** busca el mejor valor de k usando los pliegues internos.

6. **Evaluación del modelo con el mejor valor de k**:

```
y_pred = grid_search.best_estimator_.predict(X_test)
```

```
outer_accuracy = accuracy_score(y_test, y_pred)
outer_scores.append(outer_accuracy)
```

- Después de encontrar el mejor k en la validación interna, entrenamos el modelo en el conjunto de entrenamiento externo y lo probamos en el conjunto de prueba externo.
- Calculamos la precisión para el conjunto de prueba y la guardamos en **outer_scores**.

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

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

- Finalmente, calculamos y mostramos la precisión promedio de todos los pliegues externos.

8. **Matriz de confusión para el último pliegue**:

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

- Visualizamos la matriz de confusión del último pliegue externo, lo que nos permite ver el rendimiento del modelo en términos de verdaderos positivos, falsos negativos, etc.

Librerías necesarias:
Para ejecutar este código en Python, asegúrate de tener las siguientes librerías instaladas:

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

Este código realiza doble validación cruzada: una validación cruzada interna para encontrar el mejor valor de k, y una validación cruzada externa para evaluar el modelo con ese valor de k en datos de prueba no vistos. Además, se incluye una visualización de la matriz de confusión para el último pliegue.

Ejercicio 105_02. Clasificación con 2-CV 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)

% 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
k_values = [1, 3, 5, 7, 9];  % Diferentes valores de k para probar en la validación
interna

% Crear partición de la validación cruzada externa
```

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

```
cv_outer = cvpartition(clase, 'KFold', K_outer);

% Inicializar variables para guardar los resultados
best_k_values = zeros(K_outer, 1);  % Para almacenar el mejor valor de k para cada
pliegue
accuracy_vals = zeros(K_outer, 1);  % Para almacenar la precisión de cada pliegue
externo

% 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 seleccionar el mejor valor de k
    cv_inner = cvpartition(y_train, 'KFold', K_inner);
    best_accuracy_inner = 0;  % Inicializar la mejor precisión interna
    best_k = k_values(1);     % Inicializar el mejor valor de k

    for k = k_values
        accuracy_inner = zeros(K_inner, 1);  % Para almacenar la precisión interna

        % 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), :);

            % Crear el modelo k-NN con el valor actual de k
            modelo_knn = fitcknn(X_train_inner, y_train_inner, 'NumNeighbors', k);

            % Predecir en el conjunto de validación interna
            y_pred_inner = predict(modelo_knn, 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 el mejor valor de k si la precisión mejora
        if mean_accuracy_inner > best_accuracy_inner
            best_accuracy_inner = mean_accuracy_inner;
            best_k = k;
        end
    end

    % Guardar el mejor valor de k para este pliegue
    best_k_values(i) = best_k;
```

```
    % Entrenar el modelo con el mejor k en los datos de entrenamiento externo
    best_model_knn = fitcknn(X_train, y_train, 'NumNeighbors', best_k);

    % Predecir en los datos de prueba externos
    y_pred_test = predict(best_model_knn, 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), '%']);

% Visualización de los valores de k seleccionados
figure;
plot(1:K_outer, best_k_values, '-o');
xlabel('Pliegue externo');
ylabel('Mejor valor de k');
title('Mejor valor de k en cada pliegue de la validación cruzada externa');
grid on;
```

Explicación paso a paso:

1. Lectura del archivo Excel:

```
data = readtable(filename);
entradas = data{:, 1:end-1}; % Todas las columnas menos la última (entradas)
clase = data{:, end};        % Última columna (etiquetas de clase)
```

- **readtable**: Carga los datos desde un archivo Excel en una tabla. Asumimos que la última columna contiene las etiquetas de clase (**clase**), y todas las demás columnas contienen las características (**entradas**) que se usarán para hacer la predicción.
- **entradas**: Matriz que contiene los datos de entrada (variables predictoras).
- **clase**: Vector que contiene las etiquetas de clase, es decir, el valor que queremos predecir.

2. Definir los parámetros de validación cruzada:

```
K_outer = 5;  % Número de folds para la validación cruzada externa
K_inner = 5;  % Número de folds para la validación cruzada interna
k_values = [1, 3, 5, 7, 9];  % Diferentes valores de k para probar en la validación
interna
```

- **K_outer** y **K_inner**: Número de particiones o pliegues para la validación cruzada externa e interna, respectivamente. Esto significa que los datos se dividen en 5 subconjuntos (o pliegues) para cada validación.
- **k_values**: Es una lista de valores para k (el número de vecinos) que vamos a probar durante la validación cruzada interna. Estos son los diferentes valores que usaremos para el algoritmo k-NN y luego seleccionaremos el mejor.

3. Validación cruzada externa:

```
cv_outer = cvpartition(clase, 'KFold', K_outer);
```

- **cvpartition**: Genera las particiones para la validación cruzada externa. Se divide el conjunto de datos en K_outer subconjuntos de tamaño similar.

- En cada iteración de la validación cruzada externa, un subconjunto se usa como conjunto de prueba externo, y los demás se usan como conjunto de entrenamiento externo.

4. 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), :);
```

- Para cada pliegue externo i (hay K_outer pliegues en total), se divide el conjunto de datos en **entrenamiento externo** y **prueba externa**.
  - **training(cv_outer, i)**: Devuelve los índices de las filas del conjunto de entrenamiento.
  - **test(cv_outer, i)**: Devuelve los índices de las filas del conjunto de prueba.

En cada iteración, entrenamos un modelo k-NN y lo validamos en el conjunto de prueba externo. Sin embargo, primero debemos seleccionar el mejor valor de k para este modelo, lo cual se hace mediante una **validación cruzada interna**.

5. Validación cruzada interna para seleccionar el mejor valor de k:

```
cv_inner = cvpartition(y_train, 'KFold', K_inner);
```

- **cvpartition** también se utiliza para crear particiones en los datos de **entrenamiento externo** y hacer una validación cruzada interna. En esta validación cruzada interna, se prueban diferentes valores de k (número de vecinos) para ver cuál ofrece mejores resultados.

```
for k = k_values
    accuracy_inner = zeros(K_inner, 1);  % Para almacenar la precisión interna
    for j = 1:K_inner
        % Crear el modelo k-NN con el valor actual de k
        modelo_knn = fitcknn(X_train_inner, y_train_inner, 'NumNeighbors', k);
        % Predecir en el conjunto de validación interna
        y_pred_inner = predict(modelo_knn, X_val_inner);
        % Calcular la precisión interna
        accuracy_inner(j) = sum(y_pred_inner == y_val_inner) / length(y_val_inner);
    end

    % Actualizar el mejor valor de k si la precisión mejora
    if mean_accuracy_inner > best_accuracy_inner
        best_accuracy_inner = mean_accuracy_inner;
        best_k = k;
    end
end
```

- **bucle sobre k_values**: En este bucle interno, probamos cada valor de k para encontrar el que produce el mejor rendimiento en la validación cruzada interna.
  - Entrenamos un modelo k-NN con el valor actual de k en los datos de entrenamiento interno y lo probamos en el conjunto de validación interno.
  - Calculamos la precisión (porcentaje de predicciones correctas) en cada pliegue interno.

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

- Finalmente, seleccionamos el mejor valor de k que da la mayor precisión promedio en la validación cruzada interna.

7. Entrenar el modelo con el mejor k:

```
best_model_knn = fitcknn(X_train, y_train, 'NumNeighbors', best_k);
```

- Después de encontrar el mejor valor de k en la validación interna, entrenamos el modelo k-NN en el conjunto de entrenamiento externo usando ese valor óptimo de k.

8. Evaluar el modelo en el conjunto de prueba externo:

```
y_pred_test = predict(best_model_knn, X_test);
accuracy_vals(i) = sum(y_pred_test == y_test) / length(y_test);
```

- Utilizamos el modelo entrenado para hacer predicciones en el conjunto de prueba externo.
- Calculamos la precisión en este conjunto de prueba externo y almacenamos el resultado para cada pliegue externo.

9. Resultados finales:

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

- Finalmente, calculamos y mostramos la precisión promedio de todos los pliegues de la validación cruzada externa, lo que nos da una estimación confiable del rendimiento del modelo en datos no vistos.

10. Visualización del mejor k seleccionado en cada pliegue:

```
plot(1:K_outer, best_k_values, '-o');
xlabel('Pliegue externo');
ylabel('Mejor valor de k');
title('Mejor valor de k en cada pliegue de la validación cruzada externa');
```

- Creamos una gráfica que muestra el mejor valor de k seleccionado en cada pliegue externo, para observar si hay alguna variación en la elección del mejor k dependiendo de los subconjuntos de datos.

Resumen:

Este enfoque de **doble validación cruzada** garantiza que seleccionamos el mejor valor de k usando los datos de entrenamiento interno, y luego evaluamos el rendimiento de este modelo optimizado en un conjunto de prueba externo. Esto nos da una evaluación más precisa y robusta del modelo, minimizando el riesgo de sobreajuste y permitiendo encontrar el valor óptimo de k para la clasificación con k-NN.