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

3.9. Análisis de Clasificación

Ejercicio 104_01. Clasificación 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()
```

Explicación:

1. **Lectura del archivo Excel**:

```
data = pd.read_excel(filename)
```

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

Usamos **pandas** para leer el archivo Excel, donde cada fila es un ejemplo, las primeras columnas son las **entradas** y la última columna es la **etiqueta de clase.**

2. **Separación de entradas y etiquetas**:

```
X = data.iloc[:, :-1]  # Entradas
y = data.iloc[:, -1]   # Etiquetas de clase
```

- **X** contiene todas las columnas menos la última, que son las **características** o **entradas.**
- **y** contiene la última columna, que es la **etiqueta de clase** que queremos predecir.

3. **División en conjunto de entrenamiento y prueba**:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Usamos **train_test_split** de **scikit-learn** para dividir los datos. El 80% se usa para entrenamiento y el 20% para prueba.

4. **Definir y entrenar el modelo k-NN**:

```
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

- **KNeighborsClassifier** crea el modelo de k-NN con k = 5 (esto se puede ajustar).
- **fit** entrena el modelo con los datos de entrenamiento.

5. **Predicción**:

```
y_pred = knn.predict(X_test)
```

- **predict** realiza predicciones en el conjunto de prueba usando el modelo entrenado.

6. **Evaluación del modelo**:

- **Precisión**: Calculamos la precisión del modelo en los datos de prueba usando **accuracy_score.**

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

- **Matriz de confusión**: Visualizamos el rendimiento del clasificador con una matriz de confusión utilizando **confusion_matrix** y **ConfusionMatrixDisplay.**

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

Librerías necesarias:

Para ejecutar este código, asegúrate de tener instaladas las siguientes librerías:

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

Ajustes opcionales:

- **Valor de k**: Puedes ajustar el valor de k para probar diferentes configuraciones y encontrar el número óptimo de vecinos.
- **División de datos**: También puedes cambiar la proporción de la división entre el conjunto de entrenamiento y prueba (por ejemplo, 70% - 30%).

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

Ejercicio 104_02. Regresión con redes neuronales en MATLAB

```
% Nombre del archivo Excel
filename = 'datos.xlsx'; % Reemplaza con el nombre de 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)

% Dividir los datos en conjunto de entrenamiento y prueba (80% - 20%)
cv = cvpartition(clase, 'HoldOut', 0.2);
idxEntrenamiento = training(cv); % Índices de las filas para entrenamiento
idxPrueba = test(cv);            % Índices de las filas para prueba

X_train = entradas(idxEntrenamiento, :);
y_train = clase(idxEntrenamiento, :);
X_test = entradas(idxPrueba, :);
y_test = clase(idxPrueba, :);

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

% Crear el modelo k-NN
modelo_knn = fitcknn(X_train, y_train, 'NumNeighbors', k);

% Realizar predicciones con el conjunto de prueba
y_pred = predict(modelo_knn, X_test);

% Evaluar la precisión del modelo
precision = sum(y_pred == y_test) / length(y_test);
disp(['Precisión del modelo k-NN: ', num2str(precision * 100), '%']);

% Matriz de confusión
figure;
confusionchart(y_test, y_pred);
title('Matriz de confusión para el k-NN');
```

Explicación paso a paso:

1. **Lectura de los datos desde Excel**:

```
data = readtable(filename);
```

   - **readtable** carga los datos desde el archivo Excel en una tabla de MATLAB.

2. **Separación de entradas y etiquetas de clase**:

```
entradas = data{:, 1:end-1}; % Entradas
clase = data{:, end};        % Etiquetas de clase
```

   - Todas las columnas menos la última se consideran **entradas.**
   - La última columna se considera la **etiqueta de clase** que queremos predecir.

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

3. **División en conjunto de entrenamiento y prueba**:

```
cv = cvpartition(clase, 'HoldOut', 0.2);
```

   - Usamos el 80% de los datos para entrenamiento y el 20% para prueba, utilizando **cvpartition**.

4. **Definir y entrenar el modelo k-NN**:

```
modelo_knn = fitcknn(X_train, y_train, 'NumNeighbors', k);
```

   - **fitcknn** ajusta el modelo de clasificación k-NN con k = 5 (esto puede ajustarse según el problema).

5. **Predicción**:

```
y_pred = predict(modelo_knn, X_test);
```

   - **predict** realiza las predicciones en el conjunto de prueba.

6. **Evaluación del modelo**:

   - **Precisión**: Calculamos la precisión dividiendo el número de predicciones correctas entre el total de ejemplos de prueba.

```
precision = sum(y_pred == y_test) / length(y_test);
```

   - **Matriz de confusión**: Visualizamos el rendimiento del clasificador mediante una matriz de confusión.

```
confusionchart(y_test, y_pred);
```

Ajustes opcionales:

- **Valor de k**: se puede cambiar el valor de k (número de vecinos) para probar con diferentes configuraciones y ver cuál tiene mejor rendimiento.
- **División de datos**: La proporción entre el conjunto de entrenamiento y prueba (80% - 20%) también se puede ajustar según el tamaño de los datos.