SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL 2

MAESTROS:

DIEGO ALBERTO OLIVA NAVARRO

DIEGO CAMPOS PENA

ALUMNO:

EDUARDO BLANCO GONZALEZ

Tarea: Práctica 1, ejercicio 4.

Ejercicio 4: Iris es el género de una planta herbácea con flores que se utilizan en decoración. Dentro

de este género existen muy diversas especies entre las que se han estudiado la Iris setosa, la Iris versicolor y la Iris virginica (ver Figura 3).

Las tres especies se pueden diferenciar en base a las dimensiones de sus pétalos y sépalos. Se ha recopilado la información de 50 plantas de cada especie y se han almacenado en el archivo irisbin.csv. Dichas mediciones están en centímetros junto con un código binario que indica la especie a la que pertenece [-1, -1, 1] = setosa, [-1, 1, -1] = versicolor, [1, -1, -1] = virginica, la Figura 4 muestra la distribución de los datos contenidos en el archivo. Se debe crear un programa capaz de clasificar automáticamente los datos de 150 patrones usando un perceptrón multicapa. Es recomendable considerar 80% de los datos para entrenamiento y 20% para generalización.

Figura 3. Muestra de la especie Iris virginica.

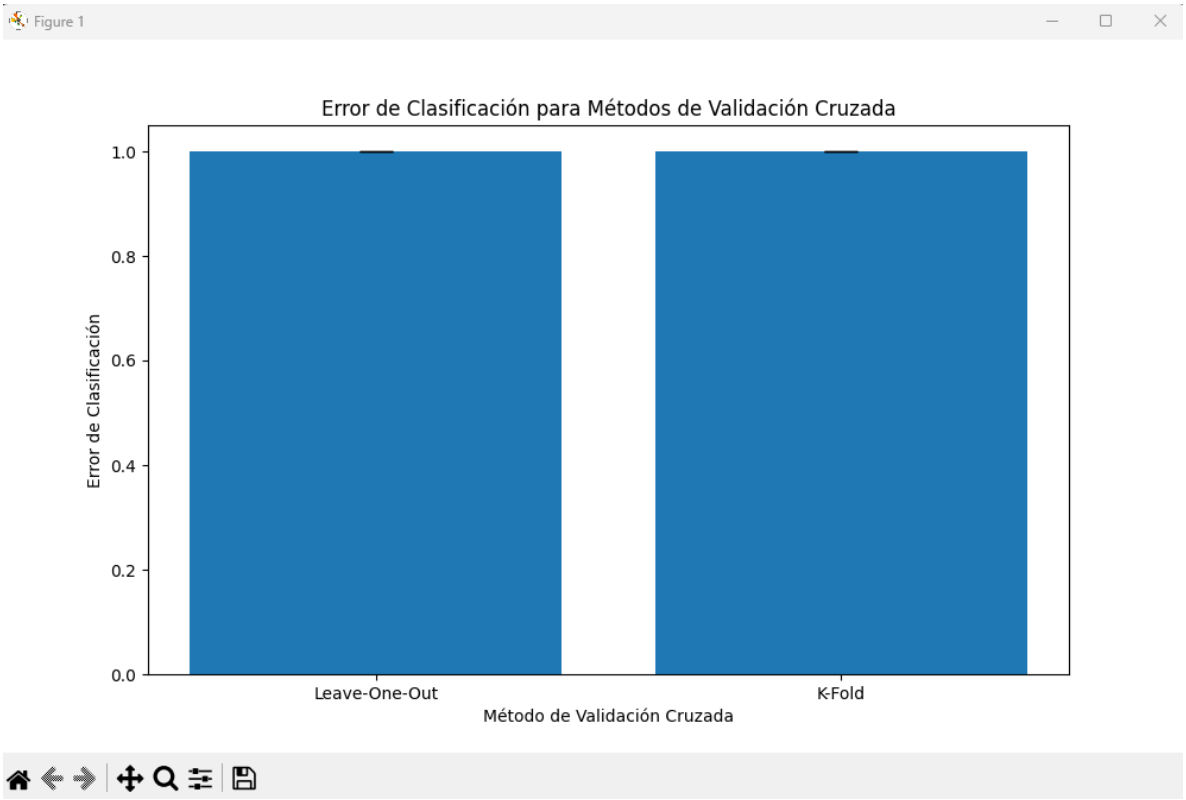
Con la estructura optima de la red, se deben validar los resultados usando lo métodos leave-k-out y leave-one-out con un perceptrón multicapa como clasificador. Se debe estimar el error esperado de clasificación, el promedio y la desviación estándar de ambos métodos.

CODIGO:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Paso 1: Cargar y preprocesar los datos
data = pd.read_csv("irisbin.csv")
X = data.iloc[:, :-3].values
y = data.iloc[:, -3:].values

# Paso 2: Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Paso 3: Definir y entrenar el perceptrón multicapa
model = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
model.fit(X_train, y_train)

# Paso 4: Validar los resultados con leave-k-out y leave-one-out
loo = LeaveOneOut()
kf = KFold(n_splits=5)
accuracies_loo = []
accuracies_kf = []

for train_index, test_index in loo.split(X):
    X_train_loo, X_test_loo = X[train_index], X[test_index]
    y_train_loo, y_test_loo = y[train_index], y[test_index]
    model_loo = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
    model_loo.fit(X_train_loo, y_train_loo)
    y_pred_loo = model_loo.predict(X_test_loo)
    accuracies_loo.append(accuracy_score(y_test_loo, y_pred_loo))

for train_index, test_index in kf.split(X):
    X_train_kf, X_test_kf = X[train_index], X[test_index]
    y_train_kf, y_test_kf = y[train_index], y[test_index]
    model_kf = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
    model_kf.fit(X_train_kf, y_train_kf)
    y_pred_kf = model_kf.predict(X_test_kf)
    accuracies_kf.append(accuracy_score(y_test_kf, y_pred_kf))

# Paso 5: Calcular el error esperado, promedio y desviación estándar
error_esperado_loo = 1 - np.mean(accuracies_loo)
error_esperado_kf = 1 - np.mean(accuracies_kf)
promedio_loo = np.mean(accuracies_loo)
promedio_kf = np.mean(accuracies_kf)
desviacion_estandar_loo = np.std(accuracies_loo)
desviacion_estandar_kf = np.std(accuracies_kf)
```

```python
# Paso 6: Graficar los resultados
plt.figure(figsize=(10, 6))
plt.bar(["Leave-One-Out", "K-Fold"], [error_esperado_loo, error_esperado_kf], yerr=[desviacion_estandar_loo, desviacion_estandar_kf], capsize=10)
plt.xlabel("Método de Validación Cruzada")
plt.ylabel("Error de Clasificación")
plt.title("Error de Clasificación para Métodos de Validación Cruzada")
plt.show()
```

FUNCION



Este código carga datos de iris, entrena un modelo de perceptrón multicapa y valida los resultados utilizando dos métodos de validación cruzada. Luego, calcula y grafica el error de clasificación para comparar los métodos de validación.