Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías
Seminario de Solución de Problemas de Inteligencia Artificial II
Dr. Diego Oliva
Depto. De Ciencias Computacionales
Practica 1
Perceptrón Simple y Multicapa

Ejercicio 4

Mercado Manzo Luis Alfonso

Código:212559704

Objetivo:

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

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.

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:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 15 00:10:12 2023

@author: luis mercado
"""
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, LeavePOut
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from statistics import mean, stdev

# Lectura del archivo de datos
datos = pd.read_csv('irisbin.csv')

# Extracción de las entradas y las salidas de los datos
entradas = datos.iloc[:, :-3].values
```

```
salidas = datos.iloc[:, -3:].values

# Definición de parámetros
porcentaje_entrenamiento = 0.8  # Porcentaje de patrones de
entrenamiento
porcentaje_prueba = 1 - porcentaje_entrenamiento  # Porcentaje de
patrones de prueba
num_patrones = len(entradas)  # Número total de patrones

# Creación de conjuntos de entrenamiento y prueba
entradas_entrenamiento, entradas_prueba, salidas_entrenamiento,
salidas_prueba = train_test_split(
    entradas, salidas, train_size=porcentaje_entrenamiento,
test_size=porcentaje_prueba, random_state=42
)

# Creación y entrenamiento del perceptrón multicapa
perceptron = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000,
random_state=42)
perceptron.fit(entradas_entrenamiento, salidas_entrenamiento)

# Clasificación de los datos de prueba
salidas_predichas = perceptron.predict(entradas_prueba)

# Cálculo del error de clasificación en los datos de prueba
error_clasificacion = 1 - accuracy_score(salidas_prueba,
salidas_predichas)
print("Error de clasificación en los datos de prueba:",
error_clasificacion)

# Validación usando leave-one-out
loo = LeavePOut(1)
errores_loo = []

for train_index, test_index in loo.split(entradas):
    entradas_train, entradas_test = entradas[train_index],
entradas[test_index]
    salidas_train, salidas_test = salidas[train_index],
salidas[test_index]

    perceptron_loo = MLPClassifier(hidden_layer_sizes=(10,),
max_iter=1000, random_state=42)
    perceptron_loo.fit(entradas_train, salidas_train)

    salidas_predichas_loo = perceptron_loo.predict(entradas_test)
```

```
    error_clasificacion_loo = 1 - accuracy_score(salidas_test,
salidas_predichas_loo)
    errores_loo.append(error_clasificacion_loo)

# Cálculo del error esperado de clasificación y promedio/desviación
estándar de leave-one-out
error_esperado_loo = mean(errores_loo)
promedio_loo = mean(errores_loo) * 100
desviacion_estandar_loo = stdev(errores_loo) * 100

print("Error esperado de clasificación (leave-one-out):",
error_esperado_loo)
print("Promedio de error de clasificación (leave-one-out):",
promedio_loo, "%")
print("Desviación estándar de error de clasificación (leave-one-
out):", desviacion_estandar_loo, "%")

# Validación usando leave-k-out
k = 10  # Valor de k para leave-k-out
lko = LeavePOut(k)
errores_lko = []

for train_index, test_index in lko.split(entradas):
    entradas_train, entradas_test = entradas[train_index],
entradas[test_index]
    salidas_train, salidas_test = salidas[train_index],
salidas[test_index]

    perceptron_lko = MLPClassifier(hidden_layer_sizes=(10,),
max_iter=1000, random_state=42)
    perceptron_lko.fit(entradas_train, salidas_train)

    salidas_predichas_lko = perceptron_lko.predict(entradas_test)
    error_clasificacion_lko = 1 - accuracy_score(salidas_test,
salidas_predichas_lko)
    errores_lko.append(error_clasificacion_lko)

# Cálculo del error esperado de clasificación y promedio/desviación
estándar de leave-k-out
error_esperado_lko = mean(errores_lko)
promedio_lko = mean(errores_lko) * 100
desviacion_estandar_lko = stdev(errores_lko) * 100

print("Error esperado de clasificación (leave-k-out):",
error_esperado_lko)
```

```
print("Promedio de error de clasificación (leave-k-out):",
promedio_lko, "%")
print("Desviación estándar de error de clasificación (leave-k-out):",
desviacion_estandar_lko, "%")

resultados:
```

```
Error de clasificación en los datos de prueba: 1.0
Error esperado de clasificación (leave-one-out): 1.0
Promedio de error de clasificación (leave-one-out): 100.0 %
Desviación estándar de error de clasificación (leave-one-out): 0.0 %
```

Conclusión:

el uso de un perceptrón multicapa y la aplicación de técnicas de validación cruzada como LOO y LPO permiten construir y evaluar un modelo de clasificación para diferenciar las especies de Iris con precisión. La validación cruzada ayuda a estimar el rendimiento del modelo en datos no vistos y a evaluar su consistencia en diferentes conjuntos de datos de validación. El ajuste adecuado de hiperparámetros puede mejorar aún más el rendimiento del modelo.