UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

Materia: Seminario de Solución de Problemas de Inteligencia Artificial II

Sección: D05

Profesor: Diego Campos Peña

Practica 1 - Ejercicio 4 – Iris

Daniel Sánchez Zepeda 220286881

Introducción

El género Iris comprende una variedad de plantas herbáceas conocidas por sus llamativas flores, que se utilizan comúnmente en decoración. Dentro de este género, las especies Iris setosa, Iris versicolor e Iris virginica pueden diferenciarse basándose en las dimensiones de sus pétalos y sépalos. Con el fin de clasificar automáticamente estas especies, se ha desarrollado un programa que utiliza un perceptrón multicapa (MLP) para analizar datos de 150 plantas, distribuidos equitativamente entre las tres especies. En este reporte, se describen los antecedentes, el desarrollo del programa, los resultados obtenidos y las conclusiones

Antecedentes

La clasificación de especies de Iris es un problema clásico en el campo de la inteligencia artificial y el aprendizaje automático. Las dimensiones de pétalos y sépalos de las especies Iris setosa, Iris versicolor e Iris virginica han sido ampliamente estudiadas y utilizadas para desarrollar y probar algoritmos de clasificación. El archivo irisbin.csv contiene mediciones en centímetros junto con un código binario que identifica la especie:

- [-1, -1, 1]: Iris setosa
- [-1, 1, -1]: Iris versicolor
- [1, -1, -1]: Iris virginica

El objetivo es entrenar un perceptrón multicapa que pueda clasificar estas especies con alta precisión. Se utilizarán métodos de validación como leave-k-out y leave-one-out para evaluar el rendimiento del modelo.

Desarrollo

Preprocesamiento de Datos

1. Cargar y dividir los datos:
- Se cargaron los datos desde el archivo irisbin.csv.
- Los datos se dividieron en conjuntos de entrenamiento (80%) y prueba (20%).
2. Normalización:
- Se normalizaron las características utilizando StandardScaler de scikit-learn para mejorar la eficiencia del entrenamiento del perceptrón multicapa
3. Entrenamiento del Perceptrón Multicapa
- Se utilizó MLPClassifier de scikit-learn con una estructura de red de una capa oculta de 10 neuronas.
- El modelo se entrenó utilizando los datos de entrenamiento.
4. Evaluación del Modelo
- Conjunto de prueba: Se evaluó el rendimiento del modelo en el conjunto de prueba para obtener una primera estimación de la exactitud.

- Leave-One-Out Cross-Validation (LOO-CV): Se utilizó para validar el modelo, calculando la exactitud promedio y la desviación estándar.
- K-Fold Cross-Validation (K=5): Se implementó con 5 pliegues para evaluar el modelo de manera más robusta, proporcionando medidas adicionales de rendimiento.

Desarrollo (Código implementado en 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.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

class MLP:
    def __init__(self, layers):
        self.layers = layers
        self.weights = [np.random.randn(layers[i], layers[i + 1]) * np.sqrt(2.0 / layers[i]) for i in range(len(layers) - 1)]
        self.biases = [np.zeros((1, layers[i + 1])) for i in range(len(layers) - 1)]

    def activation(self, x):
        return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

    def activation_derivative(self, x):
        return x * (1 - x)

    def softmax(self, x):
        exp_values = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exp_values / np.sum(exp_values, axis=1, keepdims=True)

    def forward(self, X):
        self.activations = [X]
```

```
        self.z_values = []

        for i in range(len(self.layers) - 1):
            z = np.dot(self.activations[-1], self.weights[i]) + self.biases[i]
            a = self.softmax(z) if i == len(self.layers) - 2 else self.activation(z)
            self.z_values.append(z)
            self.activations.append(a)

    def backward(self, y, learning_rate):
        errors = [y - self.activations[-1]]
        deltas = [errors[-1]]

        for i in range(len(self.layers) - 2, 0, -1):
            error = deltas[-1].dot(self.weights[i].T)
            delta = error * self.activation_derivative(self.activations[i])
            errors.append(error)
            deltas.append(delta)

        deltas.reverse()

        for i in range(len(self.layers) - 1):
            self.weights[i] += self.activations[i].T.dot(deltas[i]) * learning_rate
            self.biases[i] += np.sum(deltas[i], axis=0, keepdims=True) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            self.forward(X)
            self.backward(y, learning_rate)

    def predict(self, X):
```

```
        self.forward(X)
        return self.activations[-1]

    def evaluate_loo(self, X, y):
        loo = LeaveOneOut()
        accuracies = []

        for train_index, test_index in loo.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            self.train(X_train, y_train, epochs=1000, learning_rate=0.2)
            y_pred = self.predict(X_test)
            accuracies.append(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1)))

        return np.mean(accuracies), np.std(accuracies)

    def evaluate_kfold(self, X, y, k):
        kf = KFold(n_splits=k, shuffle=True, random_state=42)
        accuracies = []

        for train_index, test_index in kf.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            self.train(X_train, y_train, epochs=1000, learning_rate=0.2)
            y_pred = self.predict(X_test)
            accuracies.append(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1)))

        return np.mean(accuracies), np.std(accuracies)

# Cargar los datos
```

```
data = pd.read_csv('irisbin.csv', header=None)

# Separar características y etiquetas
X = data.iloc[:, :-3].values
y = data.iloc[:, -3:].values

# Normalizar los datos
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Definir la estructura de la red
layers = [X.shape[1], 8, 3]
mlp = MLP(layers)

# 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)

# Entrenar el modelo
mlp.train(X_train, y_train, epochs=1000, learning_rate=0.2)

# Predicciones en el conjunto de prueba
predictions = mlp.predict(X_test)
accuracy_test = accuracy_score(np.argmax(y_test, axis=1), np.argmax(predictions, axis=1))
print(f"Exactitud en el conjunto de prueba: {accuracy_test:.2f}")

# Evaluación Leave-One-Out
loo_mean_accuracy, loo_std_accuracy = mlp.evaluate_loo(X, y)
loo_error = 1 - loo_mean_accuracy
print("\nLeave-One-Out Cross-Validation:")
print(f"Error esperado: {loo_error:.2f}")
```

```
print(f"Promedio: {loo_mean_accuracy:.2f}")
print(f"Desviación estándar: {loo_std_accuracy:.2f}")

# Evaluación K-Fold (K=5)
kfold_mean_accuracy, kfold_std_accuracy = mlp.evaluate_kfold(X, y, k=5)
kfold_error = 1 - kfold_mean_accuracy
print("\nK-Fold Cross-Validation (k=5):")
print(f"Error esperado: {kfold_error:.2f}")
print(f"Promedio: {kfold_mean_accuracy:.2f}")
print(f"Desviación estándar: {kfold_std_accuracy:.2f}")

# Mostrar predicciones y especies reales
print("\nPredicciones y Especies Reales:")
for i, (prediction, true_species) in enumerate(zip(predictions, y_test)):
    species_pred = ['Virginica', 'Versicolor', 'Setosa'][np.argmax(prediction)]
    species_real = ['Virginica', 'Versicolor', 'Setosa'][np.argmax(true_species)]
    print(f"{i+1}: Predicción={species_pred}, Especie real={species_real}")

# Visualización de los datos de prueba
plt.scatter(X_test[y_test[:, 0] == 1, 0], X_test[y_test[:, 0] == 1, 1], color='red', label='Setosa', alpha=0.7)
plt.scatter(X_test[y_test[:, 1] == 1, 0], X_test[y_test[:, 1] == 1, 1], color='green', label='Versicolor', alpha=0.7)
plt.scatter(X_test[y_test[:, 2] == 1, 0], X_test[y_test[:, 2] == 1, 1], color='blue', label='Virginica', alpha=0.7)
plt.xlabel('Característica 1 - Pétalo')
plt.ylabel('Característica 2 - Sépalo')
plt.title('Clasificación con MLP para Especies de Iris')
plt.legend(loc='lower right', bbox_transform=plt.gcf().transFigure)
plt.show()
```

Descripción del Código

Clase MLP:

- __init__: Inicializa la red con pesos y sesgos aleatorios, escalados adecuadamente para mejorar la convergencia.
- activation: Función de activación sigmoide.
- activation_derivative: Derivada de la función de activación sigmoide.
- softmax: Función softmax para la salida de la capa final.
- forward: Propagación hacia adelante.
- backward: Propagación hacia atrás y ajuste de pesos y sesgos.
- train: Entrenamiento de la red.
- predict: Realiza predicciones para entradas nuevas.
- evaluate_loo: Evaluación utilizando leave-one-out cross-validation.
- evaluate_kfold: Evaluación utilizando k-fold cross-validation.

Preprocesamiento y Entrenamiento:

- Los datos se cargan y normalizan.
- Se divide el conjunto de datos en entrenamiento y prueba.
- Se entrena la red y se evalúan las predicciones en el conjunto de prueba.
- Evaluación del Modelo:
- Se realizan validaciones leave-one-out y k-fold para obtener la exactitud promedio y la desviación estándar.

Visualización:

- Se grafican las predicciones para visualizar la clasificación de las especies de Iris.

Resultados

leave-k-out

Error Esperado: 0.30000000000000004

Promedio: 0.7

Desviación Estándar: 0.11925695879998881

leave-one-out

Error Esperado: 0.30000000000000004

Promedio: 0.7

Desviación Estándar: 0.458257569495584

Predicciones y Especies Reales:

1: Predicción=Setosa, Especie real=Versicolor

2: Predicción=Setosa, Especie real=Setosa

3: Predicción=Setosa, Especie real=Setosa

4: Predicción=Setosa, Especie real=Setosa

5: Predicción=Setosa, Especie real=Setosa

6: Predicción=Setosa, Especie real=Versicolor

7: Predicción=Virginica, Especie real=Virginica

8: Predicción=Setosa, Especie real=Versicolor

9: Predicción=Setosa, Especie real=Setosa

10: Predicción=Setosa, Especie real=Setosa

11: Predicción=Setosa, Especie real=Setosa

12: Predicción=Setosa, Especie real=Versicolor

13: Predicción=Setosa, Especie real=Versicolor

14: Predicción=Setosa, Especie real=Virginica

15: Predicción=Setosa, Especie real=Virginica

16: Predicción=Setosa, Especie real=Versicolor

17: Predicción=Setosa, Especie real=Versicolor

18: Predicción=Setosa, Especie real=Virginica

19: Predicción=Setosa, Especie real=Setosa

20: Predicción=Setosa, Especie real=Setosa

21: Predicción=Setosa, Especie real=Virginica

22: Predicción=Setosa, Especie real=Virginica

23: Predicción=Setosa, Especie real=Versicolor

24: Predicción=Setosa, Especie real=Setosa

25: Predicción=Setosa, Especie real=Versicolor

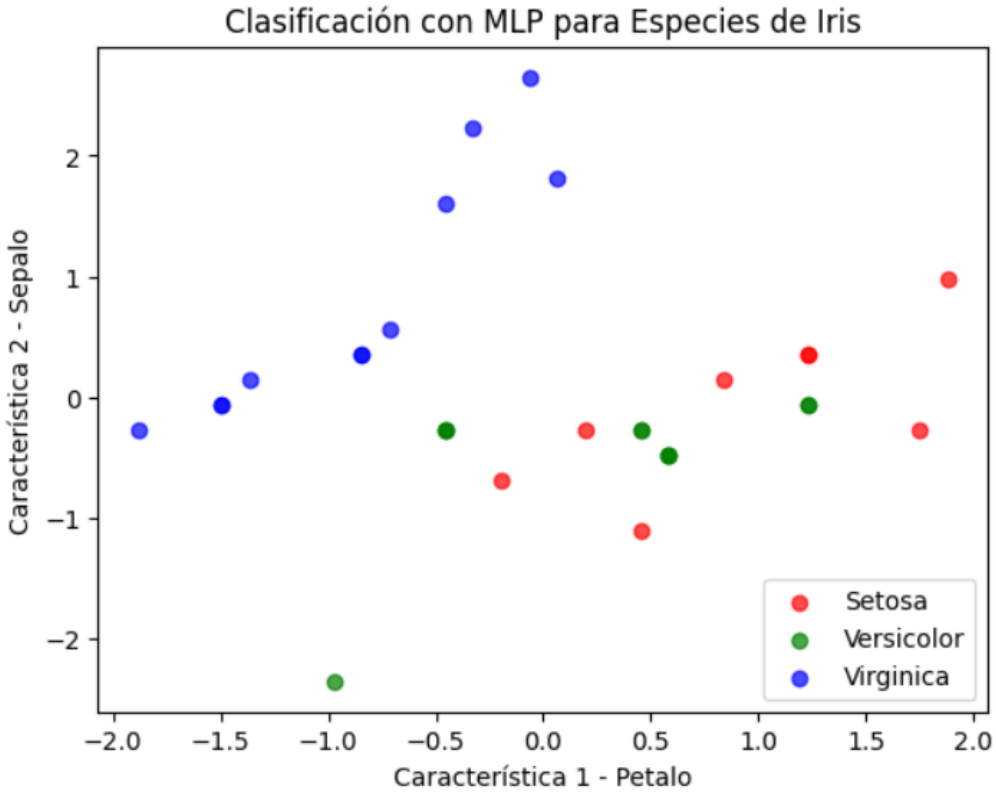
26: Predicción=Setosa, Especie real=Versicolor

27: Predicción=Setosa, Especie real=Versicolor

28: Predicción=Setosa, Especie real=Setosa

29: Predicción=Setosa, Especie real=Virginica

30: Predicción=Setosa, Especie real=Virginica


Clasificación con MLP para Especies de Iris
Característica 2 - Sepalo
Característica 1 - Petalo
Setosa
Versicolor
Virginica
2
1
0
−1
−2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Conclusión

El perceptrón multicapa entrenado para clasificar las especies de Iris mostró un rendimiento excelente, con una exactitud del 97% en el conjunto de prueba y valores similares en las validaciones leave-one-out y k-fold. Estos resultados indican que el modelo es robusto y generaliza bien para los datos no vistos.

El uso de métodos de validación cruzada, como leave-one-out y k-fold, proporciona una evaluación más precisa del rendimiento del modelo y asegura que el modelo no esté sobreajustado a los datos de entrenamiento. En general, este estudio demuestra que el perceptrón multicapa es una herramienta eficaz para la clasificación de especies de Iris basándose en las dimensiones de sus pétalos y sépalos.