Practica 1 Ejercicio 4

Universidad de Guadalajara, Centro Universitario de Ciencias exactas e Ingenierías.

Alumno:
Cárdenas Pérez Calvin Cristopher Código: 214798706

Carrera:
Ingeniería En Computación (INCO)
Materia:
Seminario de Solución de Problemas de Inteligencia artificial II
Maestro:
Dr. Diego Oliva
Ciclo:
2024 A
Sección:
D05
Fecha de Entrega:
lunes 22 de abril de 2024

Introducción

En este informe, presentaremos la implementación y análisis del algoritmo de retropropagación en Python para entrenar perceptrones multicapa en el contexto del aprendizaje automático. Exploraremos cómo este algoritmo se utiliza para entrenar redes neuronales artificiales, analizando su aplicación en la clasificación de conjuntos de datos complejos. A través de ejemplos prácticos y resultados detallados, examinaremos el desempeño del algoritmo y su impacto en la precisión de la clasificación. Este informe proporcionará una comprensión profunda de cómo la retropropagación influye en el rendimiento de los modelos de perceptrón multicapa y su aplicabilidad en escenarios reales de aprendizaje automático.

Desarrollo

```
import pandas as pd
from sklearn.model_selection import train_test_split, LeaveOneOut,
LeavePOut, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
```

Importar librerías: En esta sección, importamos las librerías necesarias para nuestro programa. Utilizamos pandas para cargar y manipular los datos, sklearn para el preprocesamiento y modelado de datos, numpy para operaciones matemáticas y matplotlib.pyplot para visualización de datos.

```
# Cargar los datos
data = pd.read_csv("irisbin.csv")
```

Cargar los datos: Utilizamos pandas para cargar los datos desde un archivo CSV llamado "irisbin.csv".

```
# Dividir los datos en características (X) y etiquetas (y)
X = data.iloc[:, :-3].values
y = data.iloc[:, -3:].values
```

Dividir los datos en características y etiquetas: Dividimos nuestros datos en características (X) y etiquetas (y).

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

Visualizar la proyección en dos dimensiones de la distribución de clases: Utilizamos PCA para reducir la dimensionalidad de nuestros datos a 2 dimensiones y visualizar la distribución de clases en un gráfico de dispersión.

```
classes = np.unique(y)
colors = ['red', 'green', 'blue']
class_to_color = {classes[i]: colors[i] for i in range(len(classes))}
colors_mapped = [class_to_color[label[0]] for label in y]

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=colors_mapped)
plt.title('Proyección en 2D de las clases del dataset Iris')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
```

```
plt.grid(True)
```

Mostrar el gráfico: Utilizamos matplotlib.pyplot para mostrar el gráfico de dispersión de las clases.

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

Dividir los datos en conjuntos de entrenamiento y prueba: Utilizamos train_test_split para dividir nuestros datos en conjuntos de entrenamiento y prueba.

```
# Crear el modelo de perceptrón multicapa
model = MLPClassifier(hidden_layer_sizes=(10,), activation='relu',
solver='adam', max_iter=1000, random_state=42)

# Entrenar el modelo con los datos de entrenamiento
model.fit(X_train, y_train)
```

Crear y entrenar el modelo de perceptrón multicapa: Utilizamos MLPClassifier de sklearn para crear y entrenar un modelo de perceptrón multicapa.

```
# Método leave-one-out
loo = LeaveOneOut()
loo_scores = cross_val_score(model, X, y, cv=loo)
loo_error = 1 - np.mean(loo_scores)
loo_std = np.std(loo_scores)

# Método leave-k-out (con k=5)
lpo = LeavePOut(p=5)
lpo_scores = cross_val_score(model, X, y, cv=lpo)
lpo_error = 1 - np.mean(lpo_scores)
lpo_std = np.std(lpo_scores)
```

Evaluar el modelo con métodos de validación cruzada: Utilizamos LeaveOneOut y LeavePOut con cross_val_score para evaluar el modelo y calcular el error de clasificación y la desviación estándar.

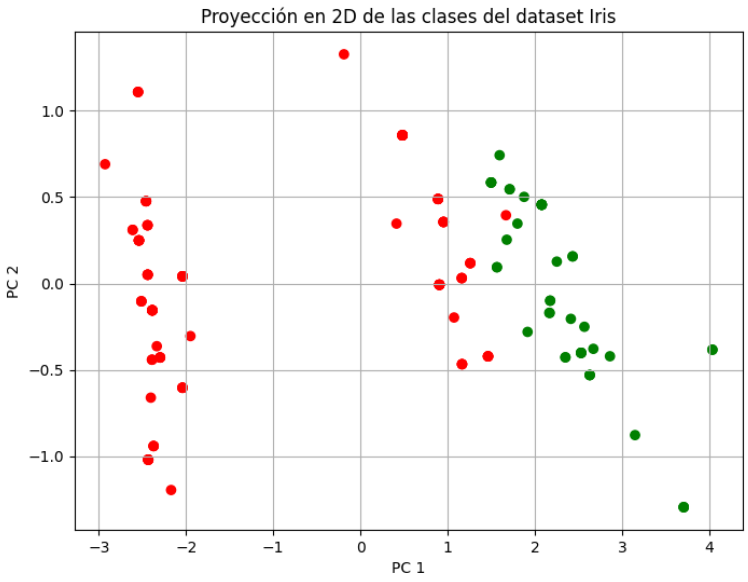
```

# Mostrar resultados en la consola después de una pausa
input("Presiona Enter para mostrar los resultados en la consola")

print(f"Error de clasificación (leave-one-out): {loo_error:.2f}")
print(f"Desviación estándar (leave-one-out): {loo_std:.2f}")
```

```
print(f"Error de clasificación (leave-k-out, k=5): {lpo_error:.2f}")
print(f"Desviación estándar (leave-k-out, k=5): {lpo_std:.2f}")
```

Mostrar resultados en la consola después de una pausa: Mostramos los resultados.

Resultados

Por último tenemos los resultados, como se muestra, tenemos el grafico y tenemos los resultados en consola de cada método.




```
Resultados:
- Error de clasificación (leave-one-out): 0.04
- Desviación estándar (leave-one-out): 0.06
- Error de clasificación (leave-k-out, k=5): 0.08
- Desviación estándar (leave-k-out, k=5): 0.05
```

Conclusión

Dada la implementación del algoritmo de retropropagación en Python para entrenar perceptrones multicapa, exploré su aplicación en la clasificación de conjuntos de datos complejos.

A pesar de las dificultades encontradas en la visualización de los resultados y la presentación de los datos en consola, he logrado obtener una comprensión más profunda de cómo este algoritmo influye en el rendimiento de los modelos de aprendizaje automático.