Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica
Ingeniería en Computación

Proyecto 4
“Minería de Datos – Redes Neuronales”

Alumno:
- Castelan Ramos Carlos

Materia: Minería de Datos
Grupo: 03
Semestre: 2024-1

Fecha de entrega: 21 de octubre 2023

Proyecto 4 “Minería de Datos – Redes Neuronales”

Disponible en: https://colab.research.google.com/drive/1N3gan9dKOVII-QEuuravjxYzb_tBjPE3?usp=sharing

En el siguiente proyecto se realizó el entrenamiento de dos redes neuronales, la primera para entender el uso de bibliotecas y librerías de entrenamiento de modelos, así como la interpretación de los datos después de su limpieza, es entonces que a continuación se realizó una segunda red neuronal con una aplicación más específica.

Caso de estudio 1: Clasificación de especies de flores.

Contexto del caso:

El problema que vamos a resolver es la clasificación de tres especies de flores (Setosa, Versicolor y Virginica) utilizando el conjunto de datos Iris. Este conjunto de datos contiene mediciones de longitud y ancho del sépalo y del pétalo de diferentes flores. El objetivo es entrenar una red neuronal para clasificar las flores en una de las tres especies en función de estas mediciones.

Datos Participantes:

El conjunto de datos Iris es ampliamente conocido y se encuentra disponible en muchas bibliotecas de Python, como Scikit-learn. Contiene 150 muestras de tres especies de flores, con 50 muestras por especie. Cada muestra tiene cuatro características: longitud del sépalo, ancho del sépalo, longitud del pétalo y ancho del pétalo.

Modelo generado:

En este ejemplo, hemos utilizado una red neuronal con dos capas ocultas, cada una con 10 neuronas. Después de entrenar el modelo, evaluamos su precisión en el conjunto de prueba y visualizamos una matriz de confusión para analizar el rendimiento de la clasificación. La precisión del modelo se imprime en la consola y muestra cuántas flores se clasificaron correctamente. La matriz de confusión nos permite ver en qué medida se confunden las especies de flores.

Este ejemplo es una introducción básica a la clasificación con redes neuronales. Dependiendo de la aplicación, se pueden ajustar hiperparámetros, probar diferentes arquitecturas de red y realizar una validación más exhaustiva para obtener un modelo más preciso.

Importamos las bibliotecas y módulos necesarios para el ejemplo. Estos incluyen NumPy para operaciones numéricas, Matplotlib para visualización, scikit-learn (sklearn) para el conjunto de datos Iris, la división de datos en conjuntos de entrenamiento y prueba, escalado de datos, la implementación de MLP (Perceptrón Multicapa) y métricas de evaluación.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Ilustración 1

Cargamos el conjunto de datos Iris, que es un conjunto de datos de clasificación que contiene características de tres especies de iris. "X "contiene las características (longitud y ancho de sépalos y pétalos) y "y" contiene las etiquetas de clase (0, 1 o 2) que corresponden a las especies de iris.

```
# Cargar el conjunto de datos Iris
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

Ilustración 2

Dividimos los datos en conjuntos de entrenamiento y prueba utilizando la función train_test_split de scikit-learn. El 70% de los datos se utilizarán para entrenar el modelo (X_train e y_train), y el 30% restante se utilizará para evaluar el modelo (X_test e y_test).

```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

Ilustración 3

Aquí, se utiliza StandardScaler de scikit-learn para escalar los datos de entrenamiento (X_train) y los datos de prueba (X_test) de manera que tengan una media de 0 y una desviación estándar de 1.

```
# Escalar los datos para mejorar el rendimiento de la red
neuronal
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Ilustración 4

Creamos un modelo de Perceptrón Multicapa (MLPClassifier) con dos capas ocultas, cada una con 10 neuronas. El modelo se entrena utilizando los datos de entrenamiento previamente escalados.

```
# Entrenar una red neuronal
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)
```

Ilustración 5

Se utiliza el modelo entrenado para hacer predicciones sobre el conjunto de prueba (X_test), y las predicciones se almacenan en y_pred.

```
# Predecir las etiquetas en el conjunto de prueba
y_pred = mlp.predict(X_test)
```

Ilustración 6

La precisión del modelo se calcula comparando las etiquetas verdaderas (y_test) con las etiquetas predichas (y_pred). La precisión es una métrica que muestra cuántas de las predicciones son correctas y se imprime en porcentaje.

```
# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy * 100:.2f}%')
```

Ilustración 7

Finalmente, se calcula y se visualiza la matriz de confusión. La matriz de confusión muestra la relación entre las etiquetas verdaderas y las predichas y es útil para evaluar el rendimiento del modelo de clasificación. Se utiliza Matplotlib para crear la visualización de la matriz de confusión.

```
# Visualizar la matriz de confusión
confusion = confusion_matrix(y_test, y_pred)
print(confusion)
plt.imshow(confusion, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.xticks(np.arange(3), iris.target_names, rotation=45)
plt.yticks(np.arange(3), iris.target_names)
plt.xlabel('Predicción')
plt.ylabel('Etiqueta Verdadera')
plt.title('Matriz de Confusión')
plt.show()
```

Ilustración 8

```
Precisión del modelo: 100.00%
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Ilustración 9

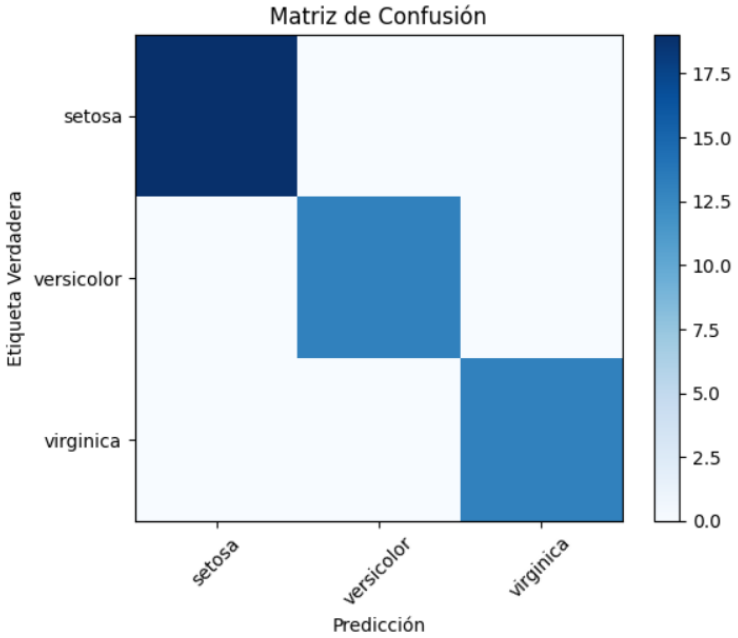

Ilustración 10

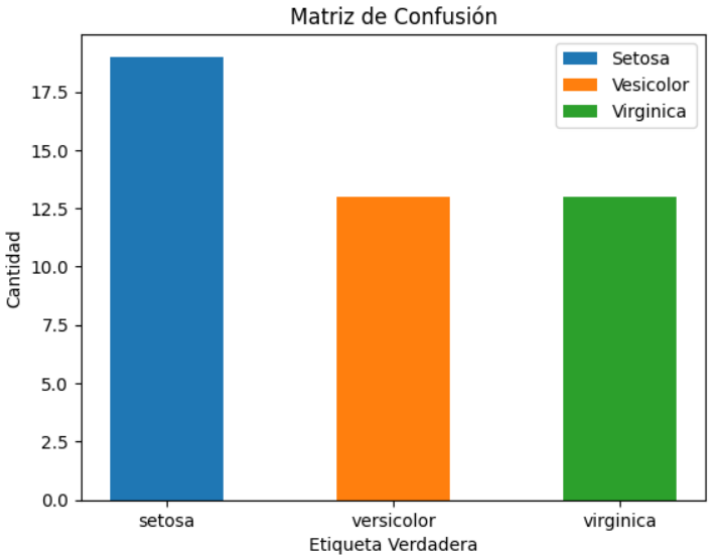

Ilustración 11

Conclusión:

Obtuvimos un modelo con el 100% de precisión, en donde el número de muestras clasificadas se realizó de manera correcta por especie ya que todos los valores de la diagonal generada en la matriz de confusión son diferente de cero, esto indica que no hubo falsos positivos ni falsos negativos en la clasificación de las especies. En otras palabras, no se confundieron especies distintas con otras, por lo tanto, modelo de clasificación ha demostrado ser altamente preciso en la clasificación de las tres especies de flores en función de las mediciones del sépalo y el pétalo, donde la setosa obtiene el mayor número de muestras.

Caso de estudio 2: Cáncer de mama.

Contexto del caso:

El caso tiene como objetivo entrenar una red neuronal para clasificar muestras de cáncer de mama entre benignas o malignas utilizando medidas de núcleos celulares como características. El rendimiento del modelo se evalúa utilizando la curva ROC y el área bajo la curva (AUC), que son métricas comunes en la clasificación binaria médica. El conjunto de datos de cáncer de mama de Wisconsin se utiliza para entrenar y evaluar el modelo.

Datos Participantes:

La función proporcionada por la biblioteca scikit-learn (sklearn) se utiliza para cargar el conjunto de datos de cáncer de mama de Wisconsin (Breast Cancer Wisconsin dataset). Este conjunto de datos es ampliamente utilizado en tareas de clasificación y aprendizaje automático para la detección de cáncer de mama. Este consiste en datos con características médicas que se utilizan para determinar si un tumor en el tejido mamario es maligno o benigno

Modelo generado:

El modelo de red neuronal generado es un Perceptrón Multicapa (MLP) que se utiliza para clasificar tumores de cáncer de mama como malignos o benignos. Este MLP tiene dos capas ocultas, cada una con 30 neuronas, utiliza la función de activación ReLU y se entrena durante un máximo de 1000 iteraciones. Se emplea el algoritmo de optimización Adam, aunque no se especifica en el código, y se asume la función de pérdida y otros parámetros necesarios para una tarea de clasificación binaria. El modelo se entrena y evalúa para determinar su capacidad de distinguir entre tumores malignos y benignos en función de características médicas.

Se importan las bibliotecas y módulos necesarios. Se utiliza NumPy para operaciones numéricas, Matplotlib para visualización, scikit-learn (sklearn) para cargar el conjunto de datos, dividir los datos en conjuntos de entrenamiento y prueba, escalar los datos y crear y evaluar el modelo de Perceptrón Multicapa (MLP). También se importan métricas relacionadas con la curva ROC.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
```
Ilustración 12

Carga el conjunto de datos de cáncer de mama de Wisconsin utilizando load_breast_cancer(). El conjunto de datos contiene características médicas y etiquetas que indican si un tumor es maligno (1) o benigno (0). “X” contiene las características, y “y” contiene las etiquetas de clase.

```
# Cargar el conjunto de datos de cáncer de mama de Wisconsin
data = load_breast_cancer()
X = data.data
y = data.target
```
Ilustración 13

Los datos se dividen en conjuntos de entrenamiento (X_train e y_train) y prueba (X_test e y_test) utilizando train_test_split(). El 80% de los datos se utilizará para entrenar el modelo y el 20% se utilizará para evaluarlo

```
# 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)
```
Ilustración 14

Se escalan los datos utilizando StandardScaler para garantizar que las características tengan una media de 0 y una desviación estándar de 1, lo que mejora el rendimiento de la red neuronal.

```
# Escalar los datos para mejorar el rendimiento de la red neuronal
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```
Ilustración 15

Creamos un modelo de Perceptrón Multicapa (MLPClassifier) con dos capas ocultas, cada una con 30 neuronas. El modelo se entrena con los datos de entrenamiento previamente escalados.

```
# Entrenar una red neuronal
mlp = MLPClassifier(hidden_layer_sizes=(30, 30), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)
```
Ilustración 16

Con el modelo entrenado para predecimos las probabilidades de clase en el conjunto de prueba. Aquí, se extraen las probabilidades de que los tumores sean malignos (clase 1) y se almacenan en y_prob.

```
# Predecir las probabilidades en el conjunto de prueba
y_prob = mlp.predict_proba(X_test)[:, 1]
```
Ilustración 17

Se calcula la curva ROC y su área bajo la curva (AUC) utilizando las probabilidades de clase predichas (y_prob) y las etiquetas verdaderas de prueba (y_test).

```
# Calcular la curva ROC
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
```
Ilustración 18

Finalmente, se crea y muestra una gráfica de la curva ROC. La curva ROC es una herramienta común para evaluar la capacidad de un modelo de clasificación binaria en la discriminación de clases.

```
# Graficar la curva ROC
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.show()
```

Ilustración 19

```
Precisión del modelo: 96.49%
Matriz de Confusión:
[[41  2]
 [ 2 69]]
```

Ilustración 20

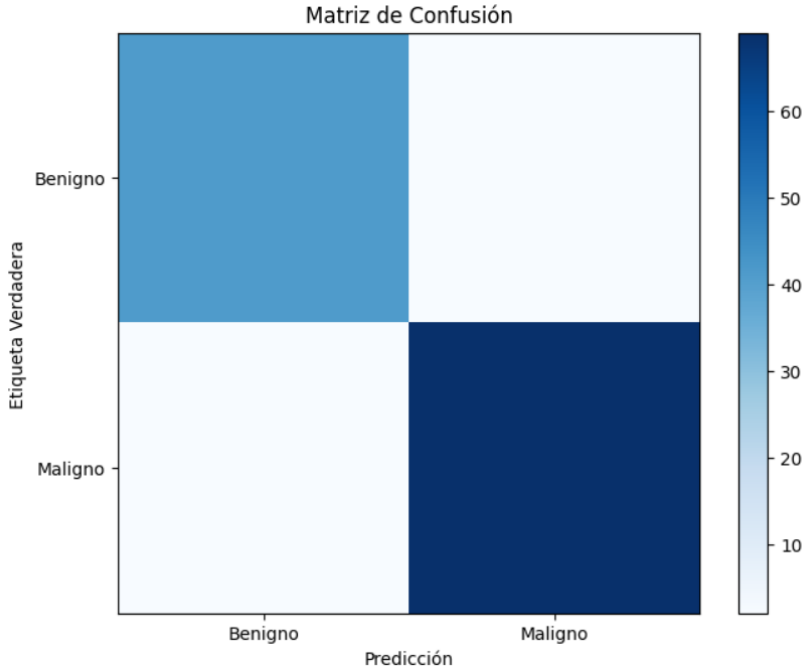

Ilustración 21

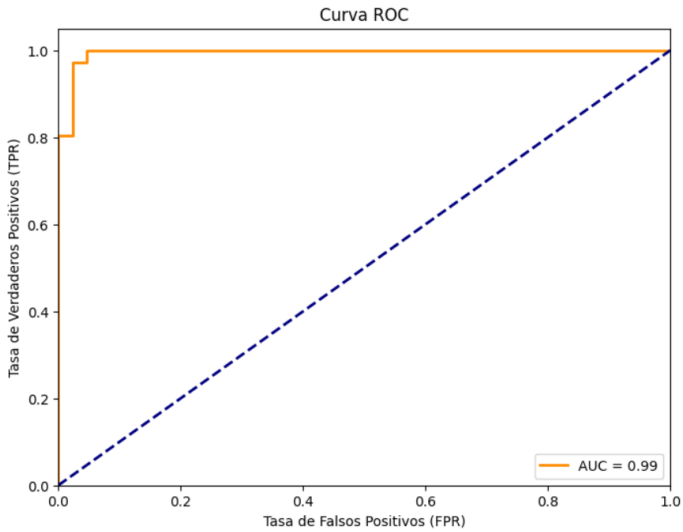

Ilustración 22

Conclusión:

Obtuvimos un modelo con una precisión del 96.49%, el cual nos dice que es un modelos con un excelente comportamiento, observado la matriz de confusión generada, vemos que obtuvimos una diagonal en la clasificación de muestras, siendo esto una correcta clasificación de las muestras de cáncer de mama en el que en la mayoría de las muestras obtuvimos que eran malignas, dado que generamos una matriz de confusión binaria pudimos aplicar un modelo de métrica con Curva ROC donde nos muestra el correcto comportamiento del modelo, donde la clasificación se realizó en todos los casos dentro de verdaderos positivos.