Bootcamp Data Science - Ejercicio semana 11 - Gerardo Rodríguez

Código sesión 11 comentado

a) **Código 1**

```
# Importar librerias numpy para calculos, make blobs para datos aleatorios, train_test_split para entrenar el modelo, y pyplot para graficar
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generar un conjunto de datos aleatorios usando make blobs
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=42)

import pandas as pd #importar libreria pandas para análisis de datos
data = pd.read_csv('./3. Perceptron.csv')  # Importar un archivo CSV de un archivo en la misma carpeta del código
X = data.iloc[:, :2]  # Asignar las primeras dos columnas a X (matriz de datos)
y = data.iloc[:, 2]   # Asignar la tercera columna a y (vector de respuesta/target)

# 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)

# Definir la clase Perceptron
class Perceptron:
    def __init__(self, learning_rate=0.1): #funcion init con las variables iniciales
        self.learning_rate = learning_rate
        self.weights = np.random.rand(3) #se definen 3 números aleatorios
```

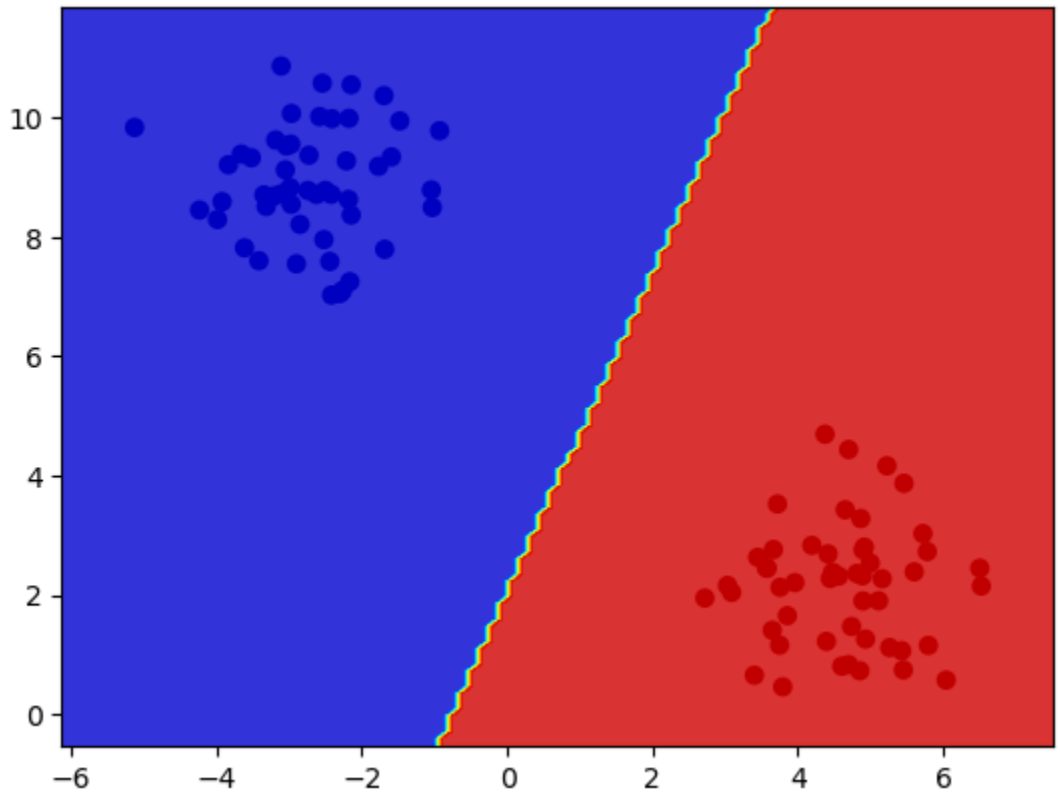
```
    def predict(self, inputs): #función predict con np.dot evalua la multiplicacion de los pesos aleatorios por las variables de entrada (inputs), es parte del proceso del perceptron
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        if summation > 0:
            activation = 1
        else:
            activation = 0
        return activation
    def train(self, training_inputs, labels): #función train entrena el modelo con la data de aprendizaje
        for __ in range(100): # Número de épocas
            for inputs, label in zip(training_inputs, labels):
                inputs = np.array(inputs, dtype=np.float64)
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction) * inputs
                self.weights[0] += self.learning_rate * (label - prediction)

# Instanciar y entrenar el perceptrón
perceptron = Perceptron()
perceptron.train(np.array(X_train, dtype=np.float64), y_train) #instanciar permite vincular los valores de entrenamiento a la función de la clase

# Armonizar formatos, ajusta el tipo de dato para simplificar los decimales
X_test = np.array(X_test, dtype=np.float64)
perceptron.weights = perceptron.weights.astype(np.float32)

# Verificar la exactitud del modelo
accuracy = sum([1 if perceptron.predict(x) == y else 0 for x, y in zip(X_test, y_test)]) / len(y_test)
print("Exactitud:", accuracy)
```

```
# Visualizar la frontera de decisión, esto muestra el funcionamiento del modelo con un rango de 100
xx, yy = np.meshgrid(np.linspace(X.X.min() - 1, X.X.max() + 1, 100),  #genera una cuadricula de puntos en varias direcciones
                     np.linspace(X.Y.min() - 1, X.Y.max() + 1, 100))
Z = np.array([perceptron.predict(x) for x in np.c_[xx.ravel(), yy.ravel()]]) #Array Z asignado a cada punto de la malla incluyendo las coordenadas de cada Z
Z = Z.reshape(xx.shape) #se ajusta Z para visualizarse de manera comparativa con X y Y
plt.scatter(X.X, X.Y, c=y, cmap='jet')
plt.contourf(xx, yy, Z, alpha=0.8, cmap='jet')
plt.show() #se muestra gráfica donde el perceptrón separa las regiones de las diferentes clasificaciones identificadas
```



b) Código 2

```
# Importar librerías
import tensorflow as tf #Libreria de aprendizaje automático
from sklearn import datasets  #Funciones para cargar datos de ejemplo
from sklearn.model_selection import train_test_split #divide datos de entrenamiento y prueba
from sklearn.preprocessing import StandardScaler #estandariza datos a una misma varianza
from tensorflow.keras.models import Sequential  #crea modelos de redes neuronales secuenciales
from tensorflow.keras.layers import Dense #crea capas entre capas de la red

# Cargar el conjunto de datos Iris
iris = datasets.load_iris()
X = iris.data[:100]  # Tomar solo las primeras 100 muestras para clasificación binaria
y = iris.target[:100]

# 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)

# Escalar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) #se transforman los datos de los datos de entrenamiento
X_test = scaler.transform(X_test) #Se estandarizan los datos de prueba

# Crear el modelo de perceptrón multicapa
model = Sequential([ #Se genera una clase para analizar la data de la red neuronal
    Dense(8, activation='relu', input_shape=(X_train.shape[1],)), #se define el nodo inicial para una red de 8 capas
    Dense(1, activation='sigmoid'),   #se define el nodo final o de salida del modelo
])
```

```
# Compilar el modelo
model.compile(optimizer='adam', #Se define el algoritmo de optimización Adam
        loss='binary_crossentropy', #se define la función pérdida para calcular el error de las predicciones
        metrics=['accuracy']) #Define la métrica de precisión a utilizar

# Entrenar el modelo
model.fit(X_train, y_train, epochs=50, batch_size=1, verbose=1)

# Evaluar el modelo
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Exactitud: {accuracy * 100:.2f}%') #devuelve el nivel de precisión del modelo
```