18 de febrero del 2025

Centro Universitario de Ciencias Exactas e Ingeniería

Inteligencia Artificial ll

Practica 1

JULIO ESTEBAN VALDES LOPEZ

Juan Pablo Hernández Orozco

219294285

Objetivo:

El objetivo de este proyecto es implementar una neurona perceptrón para clasificación. La aplicación permite que el usuario ingrese puntos (x_1, x_2) mediante clics en un canvas 2D, y a partir de la introducción de los parámetros del perceptrón (pesos w_1, w_2 y bias), se clasifiquen dichos puntos y se grafique la línea de decisión (hiperplano) definida por las fórmulas:

- Pendiente (m): $m = -w_1/w_2$
- Intersección (c): $c = -bias/w_2$

Esta simulación interactiva resulta útil para visualizar y comprender el comportamiento de un perceptrón en tareas de clasificación.

introducción:

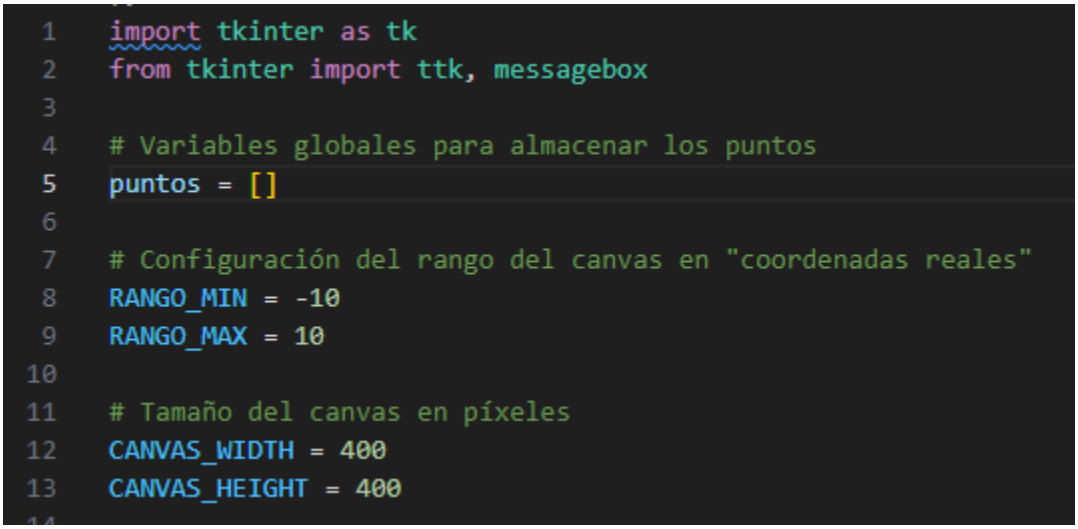
El perceptrón es uno de los algoritmos fundamentales en el campo del aprendizaje automático, diseñado para resolver problemas de clasificación linealmente separables. Su funcionamiento se basa en calcular una suma ponderada de las entradas y aplicar una función de activación (en este caso, una función escalón) para determinar a qué clase pertenece cada punto.

En este proyecto, se utiliza la biblioteca Tkinter de Python para crear una interfaz gráfica interactiva. Los puntos se capturan mediante eventos del mouse sobre un canvas, y se convierten de coordenadas de pantalla a coordenadas reales para permitir un mapeo preciso en el plano definido. Con la introducción de los parámetros del perceptrón, el programa clasifica cada punto, asignándole un color (azul para una clase y rojo para la otra), y dibuja la línea de decisión que separa las dos clases.

La estructura del código es modular y está compuesta por funciones que:

- Realizan conversiones entre coordenadas reales y de canvas.
- Capturan la interacción del usuario y actualizan la lista de puntos.
- Clasifican los puntos mediante la evaluación de la función lineal.
- Dibujan tanto los puntos clasificados como la línea de decisión y los ejes de referencia.

Screenshots:



```python
import tkinter as tk
from tkinter import ttk, messagebox

# Variables globales para almacenar los puntos
puntos = []

# Configuración del rango del canvas en "coordenadas reales"
RANGO_MIN = -10
RANGO_MAX = 10

# Tamaño del canvas en píxeles
CANVAS_WIDTH = 400
CANVAS_HEIGHT = 400
```

- Importaciones:
  - Se importa la biblioteca tkinter con el alias tk para facilitar la creación de la interfaz gráfica, y se traen módulos adicionales (ttk y messagebox) que permiten crear widgets con estilos modernos y mostrar mensajes de alerta, respectivamente.
- Variables globales:
  - La lista puntos se utiliza para almacenar cada punto ingresado por el usuario en el formato (x, y). Esta lista será fundamental para la posterior clasificación y graficación de los puntos.
- Configuración del rango:
  - RANGO_MIN y RANGO_MAX definen el intervalo de las coordenadas reales que se representarán en el canvas. Esto permite mapear de forma precisa los valores reales a la representación en píxeles.
- Tamaño del canvas:
  - CANVAS_WIDTH y CANVAS_HEIGHT determinan las dimensiones en píxeles del canvas, estableciendo el área de dibujo para los puntos y la línea de decisión.

```python
def coord_a_canvas(x, y):
    cx = (x - RANGO_MIN) * CANVAS_WIDTH / (RANGO_MAX - RANGO_MIN)
    cy = CANVAS_HEIGHT - ((y - RANGO_MIN) * CANVAS_HEIGHT / (RANGO_MAX - RANGO_MIN))
    return cx, cy

def canvas_a_coord(cx, cy):
    x = cx * (RANGO_MAX - RANGO_MIN) / CANVAS_WIDTH + RANGO_MIN
    y = RANGO_MAX - (cy * (RANGO_MAX - RANGO_MIN) / CANVAS_HEIGHT)
    return x, y
```

- coord_a_canvas: Mapea un punto definido en el rango real (por ejemplo, de -10 a 10) a las dimensiones del canvas en píxeles. La coordenada Y se invierte para ajustarse al sistema de coordenadas del canvas (donde el origen está en la esquina superior izquierda).
- canvas_a_coord: Realiza la conversión inversa para poder interpretar correctamente la posición del mouse y registrar el punto en el sistema de coordenadas reales.

```python
def on_canvas_click(event):
    """Cuando el usuario hace clic en el canvas, agrega un punto en negro."""
    x, y = canvas_a_coord(event.x, event.y)
    puntos.append((x, y))
    actualizar_lista_puntos()
    dibujar_puntos("black")  # Dibuja los puntos en negro al inicio
```

- Al hacer clic, se convierten las coordenadas del evento (event.x y event.y) a coordenadas reales.
- Se almacena el punto en la lista global puntos y se actualiza la interfaz (listbox) que muestra los puntos ingresados.
- Inicialmente, los puntos se dibujan en negro; su color se actualizará una vez se aplique la clasificación.

```python
def clasificar_punto(x, y, w1, w2, bias):
    """Clasifica el punto según la ecuación del perceptrón."""
    v = (w1 * x) + (w2 * y) + bias
    return "blue" if v >= 0 else "red"
```

- Se calcula el valor v mediante la suma ponderada de las entradas y el bias.
- Se utiliza una comparación simple: si v es mayor o igual a cero, el punto se clasifica en una clase (azul); de lo contrario, se clasifica en la otra (rojo).

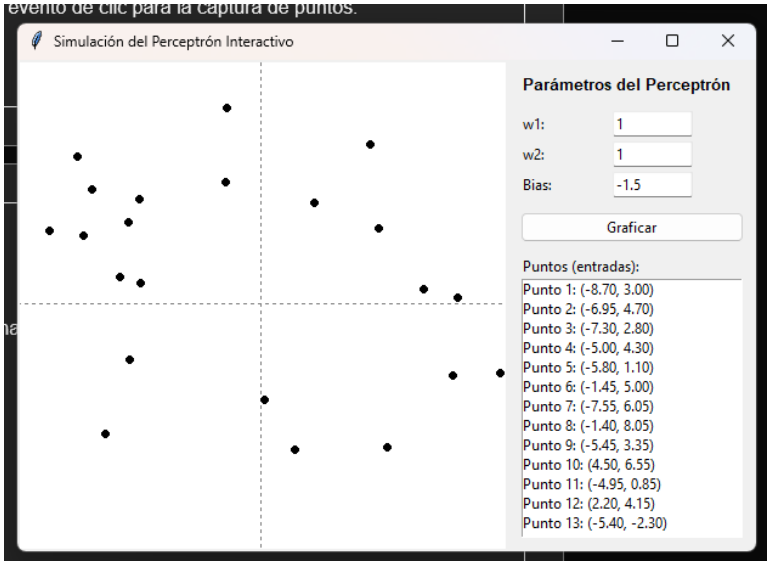
```python
def dibujar_linea_decision(w1, w2, bias):
    """Dibuja la línea de decisión y colorea los puntos según su clasificación."""
    canvas_hiperplano.delete("all")  # Limpiar todo antes de redibujar
    dibujar_ejes()

    # Calcular dos puntos extremos de la línea
    if w2 == 0:
        x_line = -bias / w1 if w1 != 0 else 0
        p1 = coord_a_canvas(x_line, RANGO_MIN)
        p2 = coord_a_canvas(x_line, RANGO_MAX)
    else:
        x1, x2 = RANGO_MIN, RANGO_MAX
        y1 = -(w1 / w2) * x1 - (bias / w2)
        y2 = -(w1 / w2) * x2 - (bias / w2)
        p1, p2 = coord_a_canvas(x1, y1), coord_a_canvas(x2, y2)

    # Dibujar la línea
    canvas_hiperplano.create_line(p1[0], p1[1], p2[0], p2[1], fill="red", width=2, tags="linea")

    # Cambiar los colores de los puntos según su clasificación
    dibujar_puntos()
```

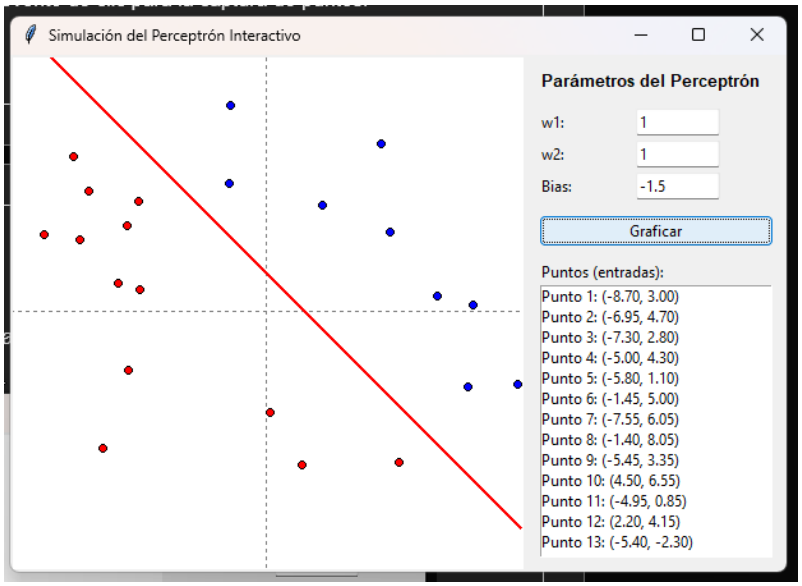
- Se elimina el contenido previo del canvas para evitar sobreposición.
- Se dibujan los ejes de referencia utilizando la función dibujar_ejes.
- Se calcula la línea de decisión considerando casos particulares (por ejemplo, cuando $w_2 = 0$ para evitar división por cero).
- Finalmente, se redibuja la línea y se actualizan los colores de los puntos según la clasificación.
- Si $w_2 \neq 0$, se toman dos valores extremos para x (RANGO_MIN y RANGO_MAX) y se calcula la correspondiente coordenada y usando la ecuación de la recta reordenada:

$$y = -\frac{w_1}{w_2}x - \frac{bias}{w_2}$$

```python
# --- Configuración de la ventana ---
root = tk.Tk()
root.title("Simulación del Perceptrón Interactivo")

frame_canvas = tk.Frame(root)
frame_controles = tk.Frame(root, padx=10, pady=10)
frame_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
frame_controles.pack(side=tk.RIGHT, fill=tk.Y)

canvas_hiperplano = tk.Canvas(frame_canvas, width=CANVAS_WIDTH, height=CANVAS_HEIGHT, bg="white")
canvas_hiperplano.pack()
dibujar_ejes()

canvas_hiperplano.bind("<Button-1>", on_canvas_click)
```

- Se crea la ventana principal (root) y se configuran dos frames: uno para el canvas y otro para los controles (parámetros y listado de puntos).
- Se inicializa el canvas y se asocia el evento de clic para la captura de puntos.
- Se organiza la interfaz para lograr una separación clara entre la zona gráfica y los controles.





Resultados:

En la primera captura de pantalla, todos los puntos se muestran en color **negro** porque aún no se ha realizado la clasificación (simplemente se colocaron en el canvas con clics del mouse). En la

segunda captura, después de ingresar los valores de los parámetros del perceptrón (w1, w2 y bias) y pulsar el botón "Graficar", se producen dos acciones principales:

1. Clasificación de los puntos:
    - El programa evalúa la siguiente función lineal:
      v = w1*x* + *w2*y + bias
    - Si v es mayor o igual a 0, el punto se colorea en **azul**; de lo contrario, se colorea en **rojo**.
    - De esta manera, los puntos que se encuentran en el “lado positivo” de la frontera de decisión se ven en azul, mientras que los que están en el “lado negativo” aparecen en rojo.
2. Dibujo de la línea de decisión (hiperplano):
    - La línea en el plano 2D se calcula a partir de:
      m = -w1 / w2
      c = -bias / w2
    - Donde m es la pendiente y c es la intersección con el eje Y.
    - Esta línea roja (mostrada en la segunda imagen) divide el plano en dos regiones: una donde v >= 0 (puntos azules) y otra donde v < 0 (puntos rojos).

Código:

```
import tkinter as tk

from tkinter import ttk, messagebox


# Variables globales para almacenar los puntos

puntos = []  # Lista de puntos (x, y)


# Configuración del rango del canvas en "coordenadas reales"

RANGO_MIN = -10
```

```
RANGO_MAX = 10


# Tamaño del canvas en píxeles
CANVAS_WIDTH = 400
CANVAS_HEIGHT = 400


def coord_a_canvas(x, y):
    """Convierte coordenadas reales a coordenadas del canvas."""
    cx = (x - RANGO_MIN) * CANVAS_WIDTH / (RANGO_MAX - RANGO_MIN)
    cy = CANVAS_HEIGHT - ((y - RANGO_MIN) * CANVAS_HEIGHT / (RANGO_MAX - RANGO_MIN))
    return cx, cy


def canvas_a_coord(cx, cy):
    """Convierte coordenadas del canvas a coordenadas reales."""
    x = cx * (RANGO_MAX - RANGO_MIN) / CANVAS_WIDTH + RANGO_MIN
    y = RANGO_MAX - (cy * (RANGO_MAX - RANGO_MIN) / CANVAS_HEIGHT)
    return x, y


def on_canvas_click(event):
    """Cuando el usuario hace clic en el canvas, agrega un punto en negro."""
    x, y = canvas_a_coord(event.x, event.y)
    puntos.append((x, y))
    actualizar_lista_puntos()
    dibujar_puntos("black")  # Dibuja los puntos en negro al inicio


def actualizar_lista_puntos():
    """Actualiza el Listbox con los puntos actuales."""
    listbox_puntos.delete(0, tk.END)
    for i, (x, y) in enumerate(puntos):
        listbox_puntos.insert(tk.END, f"Punto {i+1}: ({x:.2f}, {y:.2f})")


def clasificar_punto(x, y, w1, w2, bias):
```

```
    """Clasifica el punto según la ecuación del perceptrón."""
    v = (w1 * x) + (w2 * y) + bias
    return "blue" if v >= 0 else "red"

def dibujar_puntos(color=None):
    """Dibuja los puntos en el canvas. Si `color` es None, usa la clasificación."""
    canvas_hiperplano.delete("puntos")  # Eliminamos puntos anteriores

    try:
        w1 = float(entry_w1.get())
        w2 = float(entry_w2.get())
        bias = float(entry_bias.get())
    except ValueError:
        w1, w2, bias = 1, 1, -1.5  # Valores por defecto

    for (x, y) in puntos:
        cx, cy = coord_a_canvas(x, y)
        punto_color = color if color else clasificar_punto(x, y, w1, w2, bias)
        canvas_hiperplano.create_oval(cx-3, cy-3, cx+3, cy+3, fill=punto_color, tags="puntos")

def dibujar_linea_decision(w1, w2, bias):
    """Dibuja la línea de decisión y colorea los puntos según su clasificación."""
    canvas_hiperplano.delete("all")  # Limpiar todo antes de redibujar
    dibujar_ejes()

    # Calcular dos puntos extremos de la línea
    if w2 == 0:
        x_line = -bias / w1 if w1 != 0 else 0
        p1 = coord_a_canvas(x_line, RANGO_MIN)
        p2 = coord_a_canvas(x_line, RANGO_MAX)
    else:
        x1, x2 = RANGO_MIN, RANGO_MAX
```

```
        y1 = -(w1 / w2) * x1 - (bias / w2)
        y2 = -(w1 / w2) * x2 - (bias / w2)
        p1, p2 = coord_a_canvas(x1, y1), coord_a_canvas(x2, y2)

    # Dibujar la línea
    canvas_hiperplano.create_line(p1[0], p1[1], p2[0], p2[1], fill="red", width=2, tags="linea")

    # Cambiar los colores de los puntos según su clasificación
    dibujar_puntos()

def dibujar_ejes():
    """Dibuja los ejes X e Y en el canvas."""
    cx1, cy1 = coord_a_canvas(RANGO_MIN, 0)
    cx2, cy2 = coord_a_canvas(RANGO_MAX, 0)
    canvas_hiperplano.create_line(cx1, cy1, cx2, cy2, fill="gray", dash=(4, 2))

    cx1, cy1 = coord_a_canvas(0, RANGO_MIN)
    cx2, cy2 = coord_a_canvas(0, RANGO_MAX)
    canvas_hiperplano.create_line(cx1, cy1, cx2, cy2, fill="gray", dash=(4, 2))

def accion_graficar():
    """Obtiene los valores de los pesos y el bias, y grafica la línea de decisión solo si hay puntos."""

    # Verificar si hay puntos antes de graficar
    if not puntos:
        messagebox.showwarning("Advertencia", "No hay puntos para graficar. Agrega al menos un punto en el canvas.")
        return  # Detiene la ejecución si no hay puntos

    try:
        w1 = float(entry_w1.get())
        w2 = float(entry_w2.get())
        bias = float(entry_bias.get())
```

```
    except ValueError:
        messagebox.showerror("Error", "Ingresa valores numéricos válidos para w1, w2 y bias.")
        return

    dibujar_linea_decision(w1, w2, bias)

# --- Configuración de la ventana ---
root = tk.Tk()
root.title("Simulación del Perceptrón Interactivo")

frame_canvas = tk.Frame(root)
frame_controles = tk.Frame(root, padx=10, pady=10)
frame_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
frame_controles.pack(side=tk.RIGHT, fill=tk.Y)

canvas_hiperplano = tk.Canvas(frame_canvas, width=CANVAS_WIDTH, height=CANVAS_HEIGHT, bg="white")
canvas_hiperplano.pack()
dibujar_ejes()

canvas_hiperplano.bind("<Button-1>", on_canvas_click)

# Etiqueta de parámetros
ttk.Label(frame_controles, text="Parámetros del Perceptrón", font=("Arial", 10, "bold")).pack(pady=(0, 10), anchor="w")

# Crear un Frame para cada fila de "Label + Entry"
def crear_fila(parent, text, default_value):
    frame = ttk.Frame(parent)
    frame.pack(fill="x", pady=2)  # Alineación horizontal y separación

    label = ttk.Label(frame, text=text, width=5)  # Ancho fijo para alinear bien
    label.pack(side="left")
```

```
    entry = ttk.Entry(frame, width=10)
    entry.pack(side="left", expand=True)  # Expande para que no se vea muy comprimido
    entry.insert(0, default_value)

    return entry

# Crear inputs en una misma fila con su respectiva etiqueta
entry_w1 = crear_fila(frame_controles, "w1:", "1")
entry_w2 = crear_fila(frame_controles, "w2:", "1")
entry_bias = crear_fila(frame_controles, "Bias:", "-1.5")

# Botón Graficar (centrado)
btn_graficar = ttk.Button(frame_controles, text="Graficar", command=accion_graficar)
btn_graficar.pack(pady=(10, 10), fill="x")

# Etiqueta y Listbox para mostrar los puntos
ttk.Label(frame_controles, text="Puntos (entradas):").pack(anchor="w")
listbox_puntos = tk.Listbox(frame_controles, width=30, height=10)
listbox_puntos.pack(fill="both", expand=True)

root.mainloop()
```

Conclusión:

La aplicación implementa de forma exitosa una simulación interactiva de un perceptrón para clasificación. Utilizando Tkinter, se permite que el usuario introduzca puntos en un plano 2D y se visualice la frontera de decisión generada a partir de los parámetros ingresados (w_1, w_2 y bias). El código, organizado en módulos que gestionan la conversión de coordenadas, la captura de eventos y el dibujo gráfico, demuestra de forma clara y didáctica cómo un modelo de clasificación tan básico puede ser implementado y visualizado en tiempo real.

Esta herramienta no solo facilita la comprensión del funcionamiento interno de un perceptrón, sino que también sienta las bases para futuras extensiones o mejoras, como la incorporación de más características o la exploración de modelos más complejos. En resumen, el proyecto cumple con el objetivo de ofrecer una experiencia interactiva y educativa en el ámbito del aprendizaje automático.