Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER

HUMANA.

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

TEMA: Practica1

NOMBRE DEL ESTUDIANTE:

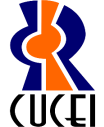
Padilla Perez Jorge Daray

NOMBRE DE LA MATERIA: IA2

SECCION: D02

CICLO ESCOLAR: 2024-B

NOMBRE DEL PROFESOR: Julio Esteban



A black background with blue lines

Description automatically generated

Contenido

[**Introducción.** 3](#_Toc161404956)

[Capturas de pantalla. 4](#_Toc161404957)

[Código fuente. 7](#_Toc161404958)

[Main(): **Error! Bookmark not defined.**](#_Toc161404959)

[Conclusiones: 11](#_Toc161404960)

# **Introducción.**

En este documento que se muestra como se realizó el programa numero 1 de esta materia, el cual se basa en hacer que un perceptrón se muestre gráficamente separando unos puntos que el usuario puede dibujar en un plano cartesiano,, este ultimo en la interfaz deberá de permitir al usuario ajustar los parámetros del perceptrón (pesos, bias), en mi caso este se hace de manera automática sin necesidad de botón.

# Capturas de pantalla.

A screenshot of a computer

Description automatically generated

Esta es la interfaz de usuario.

A screenshot of a computer screen

Description automatically generated

Aquí se puede ver ya con puntos dibujados y con el perceptrón actualizado.

A screenshot of a computer screen

Description automatically generated

Aquí mas de lo mismo con otros pesos y bias.

# Código fuente.

import tkinter as tk

from tkinter import messagebox

class Perceptron:

    def \_\_init\_\_(self, weights, bias):

        self.weights = weights

        self.bias = bias

    def predict(self, x):

        z = sum(w \* xi for w, xi in zip(self.weights, x)) + self.bias

        return 1 if z >= 0 else 0

def on\_canvas\_click(event):

    x, y = event.x, event.y

    input\_vector = [(x - 200) / 20, (200 - y) / 20]  # Normalizar las coordenadas a [-10, 10]

    prediction = perceptron.predict(input\_vector)

    color = "blue" if prediction == 1 else "red"

    # Buscar el óvalo existente en las coordenadas (x, y)

    items = canvas.find\_enclosed(x - 5, y - 5, x + 5, y + 5)

    if items:

        canvas.itemconfig(items[0], fill=color)  # Actualizar el color del óvalo existente

    else:

        canvas.create\_oval(x - 5, y - 5, x + 5, y + 5, fill=color)  # Crear un nuevo óvalo

def clear\_canvas():

    canvas.delete("all")  # Elimina todos los objetos en el canvas

    draw\_cartesian\_plane()  # Dibujar el plano cartesiano nuevamente

    update\_perceptron()    # Actualizar la visualización después de limpiar

def update\_perceptron(\*args):

    try:

        weight1 = weight1\_scale.get()

        weight2 = weight2\_scale.get()

        bias = bias\_scale.get()

        perceptron.weights = [weight1, weight2]

        perceptron.bias = bias

        # Eliminar la línea de decisión existente

        canvas.delete("line")

        # Dibujar la nueva línea de decisión

        x1 = -10

        y1 = (-weight1 / weight2) \* x1 - (bias / weight2)

        x2 = 10

        y2 = (-weight1 / weight2) \* x2 - (bias / weight2)

        # Convertir los valores de y a coordenadas en el canvas

        canvas\_x1 = x1 \* 20 + 200

        canvas\_y1 = 200 - y1 \* 20

        canvas\_x2 = x2 \* 20 + 200

        canvas\_y2 = 200 - y2 \* 20

        # Dibujar la nueva línea

        canvas.create\_line(canvas\_x1, canvas\_y1, canvas\_x2, canvas\_y2, fill="green", width=2, tags="line")

        # Actualizar los colores de los puntos existentes

        for item in canvas.find\_all():

            if canvas.type(item) == "oval":

                x, y, \_, \_ = canvas.coords(item)

                input\_vector = [(x - 200) / 20, (200 - y) / 20]

                prediction = perceptron.predict(input\_vector)

                color = "blue" if prediction == 1 else "red"

                canvas.itemconfig(item, fill=color)

        # Actualizar las etiquetas de los valores actuales

        weight1\_value\_label.config(text=f"Peso 1: {weight1\_scale.get()}")

        weight2\_value\_label.config(text=f"Peso 2: {weight2\_scale.get()}")

        bias\_value\_label.config(text=f"Sesgo: {bias\_scale.get()}")

    except ValueError:

        messagebox.showerror("Error", "Se produjo un error al actualizar el perceptrón.")

def draw\_cartesian\_plane():

    # Dibujar ejes x e y

    canvas.create\_line(0, 200, 400, 200, fill="white", width=2)

    canvas.create\_line(200, 0, 200, 400, fill="white", width=2)

    # Etiquetas de coordenadas

    for i in range(-10, 11, 1):

        canvas.create\_text(200 + i \* 20, 210, text=str(i), fill="white")

        canvas.create\_text(210, 200 - i \* 20, text=str(-i), fill="white")

# Crear la ventana principal

root = tk.Tk()

root.title("Perceptrón")

# Crear un marco para los controles

control\_frame = tk.Frame(root, padx=10, pady=10, bg="lightgray")

control\_frame.pack(side=tk.RIGHT, fill=tk.Y)

# Crear un canvas para dibujar

canvas = tk.Canvas(root, width=400, height=400, bg="black")

canvas.pack(side=tk.LEFT)

# Dibujar el plano cartesiano

draw\_cartesian\_plane()

# Crear un perceptrón con pesos y bias iniciales

initial\_weights = [1.0, 1.0]

initial\_bias = 0

perceptron = Perceptron(initial\_weights, initial\_bias)

# Escalas para los valores de los pesos y el sesgo

weight1\_label = tk.Label(control\_frame, text="Peso 1:", bg="lightgray")

weight1\_label.pack()

weight1\_scale = tk.Scale(control\_frame, from\_=-10, to=10, orient=tk.HORIZONTAL, resolution=0.1, command=update\_perceptron)

weight1\_scale.set(initial\_weights[0])

weight1\_scale.pack()

weight1\_value\_label = tk.Label(control\_frame, text=f"Peso 1: {initial\_weights[0]}", bg="lightgray")

weight1\_value\_label.pack()

weight2\_label = tk.Label(control\_frame, text="Peso 2:", bg="lightgray")

weight2\_label.pack()

weight2\_scale = tk.Scale(control\_frame, from\_=-10, to=10, orient=tk.HORIZONTAL, resolution=0.1, command=update\_perceptron)

weight2\_scale.set(initial\_weights[1])

weight2\_scale.pack()

weight2\_value\_label = tk.Label(control\_frame, text=f"Peso 2: {initial\_weights[1]}", bg="lightgray")

weight2\_value\_label.pack()

bias\_label = tk.Label(control\_frame, text="Sesgo:", bg="lightgray")

bias\_label.pack()

bias\_scale = tk.Scale(control\_frame, from\_=-10, to=10, orient=tk.HORIZONTAL, resolution=0.1, command=update\_perceptron)

bias\_scale.set(initial\_bias)

bias\_scale.pack()

bias\_value\_label = tk.Label(control\_frame, text=f"Sesgo: {initial\_bias}", bg="lightgray")

bias\_value\_label.pack()

# Actualizar el perceptrón con los valores iniciales

update\_perceptron()

# Botón para limpiar la pantalla

clear\_button = tk.Button(control\_frame, text="Limpiar Pantalla", command=clear\_canvas)

clear\_button.pack(pady=5)

# Botón para restablecer los valores

def reset\_values():

    weight1\_scale.set(initial\_weights[0])

    weight2\_scale.set(initial\_weights[1])

    bias\_scale.set(initial\_bias)

    update\_perceptron()

reset\_button = tk.Button(control\_frame, text="Restablecer Valores", command=reset\_values)

reset\_button.pack(pady=5)

# Asociar el evento de clic al canvas

canvas.bind("<Button-1>", on\_canvas\_click)

root.mainloop()

# Conclusiones:

**Padilla Perez Jorge Daray:**

Para concluir con esta actividad interesante ya que es nuestra primera neurona por así decirlo, esta se comporta de manera sencilla por así decirlo, al ser nuestro primer programa es fácil relativamente por que la interfaz fue lo mas tedioso, siendo este lo que mas dure en hacer porque el perceptrón ya lo tenia hecho de otra clase.