CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTATIVA PROFESIONALIZANTE II: MACHINE LEARNING Y DEEP LEARNING

10° "A"

ACTIVIDAD 1.01: ALGORITMO DEL PERCEPTRÓN PARA EL CONJUNTO DE DATOS: FLOR DE IRIS

Profesor: Dr. Francisco Javier Luna Rosas

Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags.**, 19** de febrero de 2023

Actividad 1.01: Algoritmo del Perceptrón para el conjunto de datos: Flor de Iris

La Regla de Aprendizaje del Perceptrón

La idea general que hay detrás de la neurona de MCP (McCulloch Pitts) y del modelo del perceptrón umbralizado de Rosenblatt es utilizar un enfoque reduccionista para imitar cómo trabaja una simple neurona en el cerebro: si se excita o si no. Así, la regla del perceptrón inicial de Rosenblatt es bastante sencilla y se puede resumir en los siguientes pasos:

1) Iniciar los pesos a números aleatorios
2) Para cada muestra de entrenamiento $x(i)$:
   a) Calcular el valor de salida "y"
   b) Actualizar los pesos

Se deberá elaborar un documento (`*.pdf`) y documento autoreproducible (`*.html`) donde implemente el modelo del perceptrón en el conjunto de datos Iris considerando solo dos clases de flor (Setosa y Versicolor) y solo tendrá que considerar las características de longitud del sépalo y la longitud del pétalo.

El alumno deberá subir a la plataforma el archivo (`*.pdf` y `*.html`) que deberá contener:

- Portada
- Evidencias de la actividad
- Conclusiones
- Referencias (formato APA)
- Letra Arial, tamaño 12 e interlineado 1.5

La realización personal de esta actividad con importar las librerías necesarias para la realización de esta actividad:

```
import random
import pandas as pd
import matplotlib.pyplot as plt
```

Lo siguiente será importar al código el dataset con el que se trabajará siendo éste el de Flor de Iris:

```
df = pd.read_csv('iris.csv', sep = ';')
df
```

	s.largo	s.ancho	p.largo	p.ancho	tipo
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

La idea detrás de lo que queremos hacer es, con base en las cuatro variables que poseemos, determinar de qué tipo de flor se trata. Esto se hará dándole un mayor o menor peso a cada variable usando la siguiente suma ponderada:

$$(w_1 \cdot x_1) + (w_2 \cdot x_2) + (w_3 \cdot x_3) + (w_4 \cdot x_4) > \theta$$

Donde:

- x_1 son valores pertenecientes a la columna `s.largo`.
- x_2 son valores pertenecientes a la columna `s.ancho`.
- x_3 son valores pertenecientes a la columna `p.largo`.
- x_4 son valores pertenecientes a la columna `p.ancho`.
- w_1 es el peso por el que se multiplicará x_1.
- w_2 es el peso por el que se multiplicará x_2.
- w_3 es el peso por el que se multiplicará x_3.
- w_4 es el peso por el que se multiplicará x_4.

- θ es el sesgo o bias que determinará la separación lineal del problema.

Así que primeramente inicializaremos los pesos aleatoriamente:

```
w1 = random.uniform(-10, 10)
w2 = random.uniform(-10, 10)
w3 = random.uniform(-10, 10)
w4 = random.uniform(-10, 10)
theta = random.uniform(-10, 10)
print("w1 = " + str(w1))
print("w2 = " + str(w2))
print("w3 = " + str(w3))
print("w4 = " + str(w4))
print("theta = " + str(theta))
```

```
w1 = -8.00169654598273
w2 = 8.087111825763976
w3 = -4.538451944881263
w4 = 8.329577390102799
theta = 3.3999190341375574
```

Anterior a esto, para trabajar con mayor comodidad convertiremos los resultados deseados en un valor numérico. Además hay que eliminar las plantas de clasificación `virginica`. Personalmente se eligió cambiar en la columna `tipo` los valores `setosa` por `0` y `versicolor` por `1`.

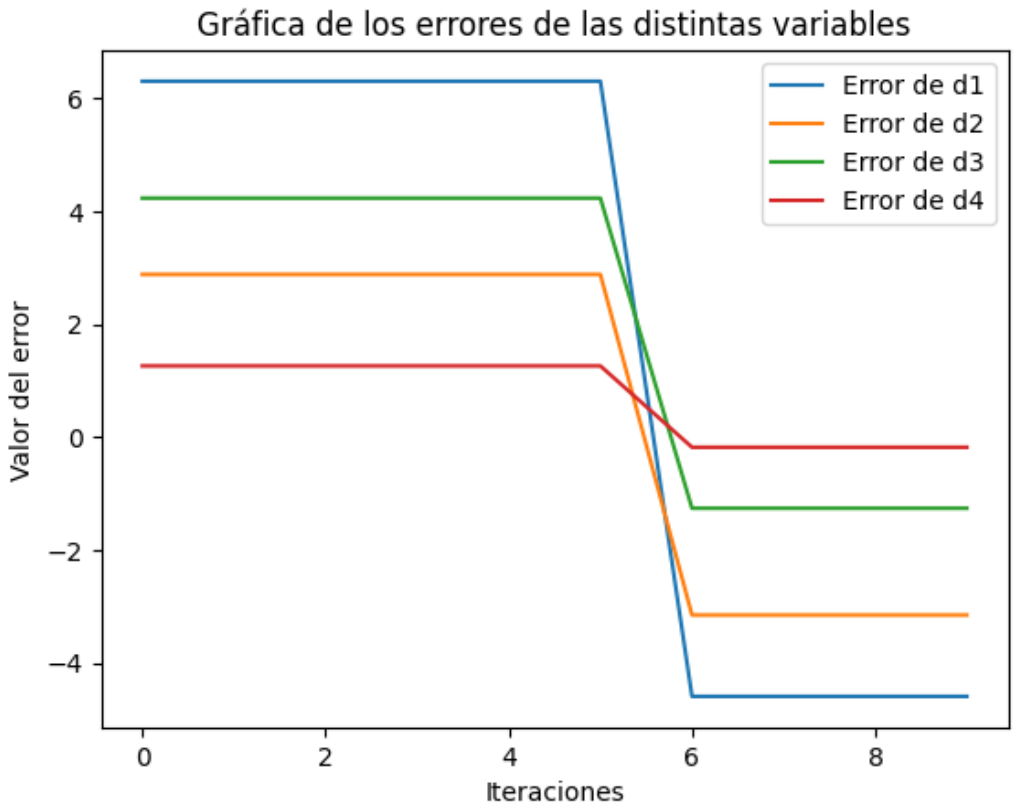
```
df = df.drop(df[df['tipo'] == 'virginica'].index)
df["tipo"] = df["tipo"].replace({"setosa": 0, "versicolor": 1})
```

	s.largo	s.ancho	p.largo	p.ancho	tipo
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1
98	5.1	2.5	3.0	1.1	1
99	5.7	2.8	4.1	1.3	1

Con estos valores comienza el procedimiento que consistirá en entrenar a la red neuronal. Según sus aciertos o errores estos valores irán modificándose según las definiciones del algoritmo. En el caso personal se decidió establecer una variable `i` sobre cuántas iteraciones se desearían realizar la cual marcaría el final del entrenamiento del algoritmo. Así también se determinó como `learnRate` la tasa de aprendizaje que el algoritmo tomará para aprender en cada iteración.

```
learnRate = 0.9
for i in range(10):
    errores = 0
    for j in range(len(df)):
        # Suma ponderada
        f = (w1 * df.loc[j, 's.largo']) + (w2 * df.loc[j, 's.ancho']) + (w3 * df.loc[j, 'p.largo']) + (w4 * df.loc[j, 'p.ancho'])

        # Comparamos la superación del umbral
        if(f > theta):
            f = 1
        else:
            f = 0

        # Se contabilizan los errores
        if(f != int(df.loc[j, 'tipo'])):
            errores = errores + 1

            # Cálculo de deltas
            d1 = learnRate * (int(df.loc[j, 'tipo']) - f) * df.loc[j, 's.largo']
            d2 = learnRate * (int(df.loc[j, 'tipo']) - f) * df.loc[j, 's.ancho']
            d3 = learnRate * (int(df.loc[j, 'tipo']) - f) * df.loc[j, 'p.largo']
            d4 = learnRate * (int(df.loc[j, 'tipo']) - f) * df.loc[j, 'p.ancho']

            # Actualización de pesos
            w1 = w1 + d1
            w2 = w2 + d2
            w3 = w3 + d3
            w4 = w4 + d4
```

Esto, posteriormente se graficó para observar cómo evolucionaban los pesos de cada variable, los deltas por cada variable y la cantidad de errores por iteración que, respectivamente se muestran a continuación:


Gráfica de los errores de las distintas variables
Error de d1
Error de d2
Error de d3
Error de d4
Valor del error
Iteraciones
6
4
2
0
−2
−4
0
2
4
6
8


Gráfica de los errores de clasificación por iteración
Cantidad
Iteraciones
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8

Conclusión: Es interesante e importante poder implementar las bases de estos temas para entenderlos en un futuro, pues, posteriormente no basta con sólo importar librerías que realicen el trabajo pesado, ya que, implementar manualmente estos algoritmos nos enseña a qué hay detrás del algoritmo, cómo funciona y poder comprender realmente qué está ocurriendo como la base de una red neuronal, en este caso, los fundamentos matemáticos del perceptrón. Es muy útil la implementación de estos algoritmos en estas tareas para las futuras tareas de la materia y aplicaciones de Machine Learning en la vida personal.

Referencias:

- Anónimo (s.f.) "*Red neuronal artificial*". Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Red_neuronal_artificial.
- Data Scientest (2021) "*Perceptrón. ¿Qué es y para qué sirve?*". Obtenido de Data Scientest: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve.
- Luna, F. (2023) "*El Modelo de McCulloch – Pitts*". Apuntes de ICI 10°.