Centro Universitario de Ciencias Exactas e Ingenierías

Licenciatura en Ingeniería en Computación

Materia: Seminario de Solución de Problemas de Inteligencia Artificial II. Clave: I7041.

Profesor: Valdés López Julio Esteban

Estudiante: Silva Moya José Alejandro. Código: 213546894.

Práctica II: Perceptrón entrenado

Problema a resolver: En la presente práctica programaremos un perceptrón capaz de aprender, a diferencia de lo presentado en el trabajo previo. En este caso inicializaremos los pesos w y el bias b de manera aleatoria entre -1 y 1, así como también recibiremos los patrones de aprendizaje desde un archivo csv y lo cargaremos así al programa. Finalmente imprimiremos en tiempo real las modificaciones de los pesos sinápticos mediante la línea de la neurona.

Desarrollo

```
16 class neurona:
17     def __init__(self, dim, eta):   #Dimension y coeficiente de aprendizaje.
18         self.n = dim
19         self.eta = eta
20         self.w = -1 + 2 * np.random.rand(dim, 1)  #x = min + (max - min)*rand()
21         self.b = -1 + 2 * np.random.rand()
```

Comenzaremos inicializando la neurona, recibiendo como parámetro la dimensión (cantidad de patrones de aprendizaje) y el coeficiente de aprendizaje, que es elegido por el programador más adelante.

El vector de pesos sinápticos estará conformado por n elementos delimitados por la cantidad de dimensiones. El Bias es un único valor, también aleatorio entre -1 y 1.

```
def predict(self, x):
    y = np.dot(self.w.transpose(), x) + self.b
    if y >= 0:
        return 1
    else:
        return -1
```

A continuación, generamos nuestra función de predicción, en la que tomaremos los valores de nuestro vector de pesos sinápticos y generaremos la propagación de acuerdo a la ecuación del perceptrón. Posteriormente pasaremos los resultados por nuestra función de activación, que en este caso es la función signo (o step function) y obtendremos los resultados binarios deseados.

```
def train(self, X, y, epochs):  #x = matriz de entrenamiento, y = vector con
    n, m = X.shape
    #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
    for i in range(epochs):
        for j in range(m):
            y_pred = self.predict(X[:, j])
            #what that line did is sliced the array,
            #taking all rows (:) but keeping the column (j)
            if y_pred != y[j]:  #Si nuestro estimado es diferente a nuestro
                self.w += self.eta*(y[j] - y_pred) * X[:, j].reshape(-1, 1)
                self.b += self.eta*(y[j] - y_pred)
            graph(self.w)
```

Finalmente escribiremos nuestra función de entrenamiento. Comenzaremos por obtener la forma de nuestra matriz de patrones de aprendizaje X, y entrenaremos de acuerdo a la cantidad de épocas seleccionadas y patrones ingresados. Para cada patrón de aprendizaje generaremos una predicción de acuerdo a nuestros pesos sinápticos, y en caso de que la predicción difiera de nuestras salidas esperadas, habrá que entrenar al perceptrón; es decir: modificar los pesos y el bias.

Las fórmulas de aprendizaje son las siguientes:

$$w = w + \eta\,(y^i - \hat{y}^i)\,x^i$$

$$b = b + \eta\,(y^i - \hat{y}^i)$$

Donde:

- η es el coeficiente de aprendizaje.
- Y es nuestra salida deseada.
- $\hat{Y}$ es nuestra salida predicha.
- i es la iteración en la que nos encontramos, dentro del rango de los patrones de aprendizaje, que siempre es cíclico.

Después de esto graficamos cada modificación de los pesos sinápticos. La función será mostrada el final de este entregable.

Ahora comenzaremos con el proceso de generación de la neurona.

```
45 file = open("DataSet.csv")
46 rows = len(file.readlines())
47 file.close()
48 #To obtain the number of rows from the CSV file
49
50 f = open("DataSet.csv",'r')
51 reader = csv.reader(f,delimiter=',')
52 columns = len(next(reader))
53 f.close()
54 #To obtain the number of columns from the CSV file
55
56 net = neurona(columns-1, 0.1)
57 #Perceptron initialization.
```

Lo primero será obtener la cantidad de filas y columnas del archivo CSV para poder inicializar de manera correcta la neurona. Una vez teniendo dichos datos, los pasaremos las columnas-1 como parámetro, y el coeficiente de aprendizaje. La razón de columns-1 se debe a que este parámetro se utiliza para inicializar aleatoriamente el vector de pesos sinápticos, y suponemos que el archivo CSV contiene todas las columnas de entradas X y la última que sea la columna de salidas esperadas, así que esa se ignora, para poder corresponder con las dimensiones correctas.

```
60 patterns = []
61
62 for i in range(columns-1):
63     x = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=i))
64     patterns.append(x)
65 X = np.array(patterns)
66
67 y = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=columns-1))
68 #Obtaining training patterns in X and output values in y.
```

Ahora cargaremos los vectores de patrones de aprendizaje y salidas esperadas. Tomaremos todas las columnas-1 para los patrones, y la última para las salidas.

```
net.train(X, y, 20)

for i in range(rows):
    print(net.predict(X[:, i]))

plt.clf()
plt.scatter(X[0], X[1], color="black")
plt.axhline(color="blue")
plt.axvline(color="blue")
x_values = [-3,3]
y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
            -(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
plt.plot(x_values, y_values, color="red")
```

Procederemos a entrenar a la neurona, con todos nuestros datos y 20 generaciones (que son modificables en cualquier momento).

Finalmente generaremos la predicción de datos con nuestra neurona entrenada, para confirmar que haya aprendido correctamente.

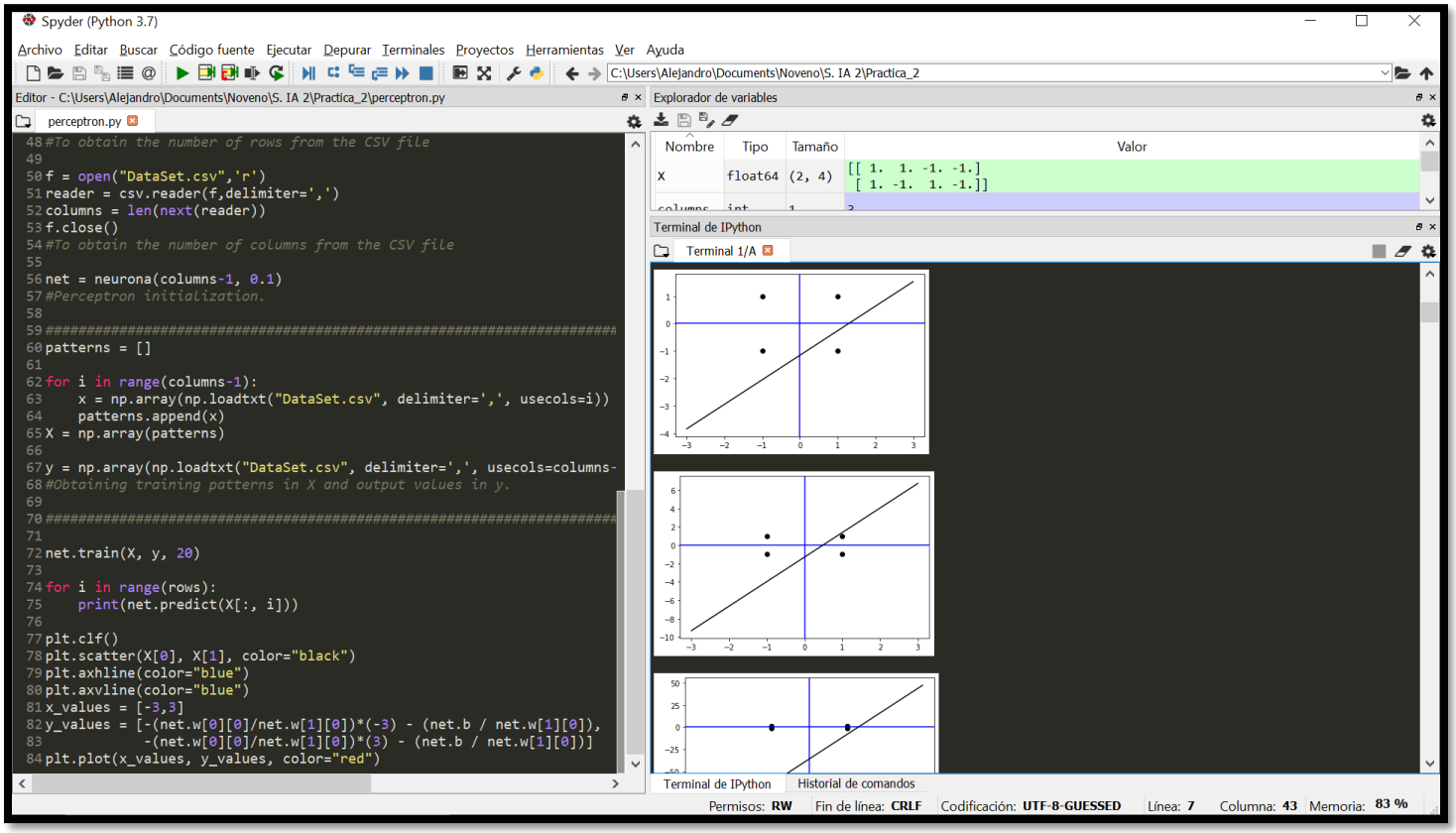
El resto del código es para imprimir exclusivamente la última línea de la neurona. Este proceso se repite en durante el entrenamiento, que lo veremos a continuación.

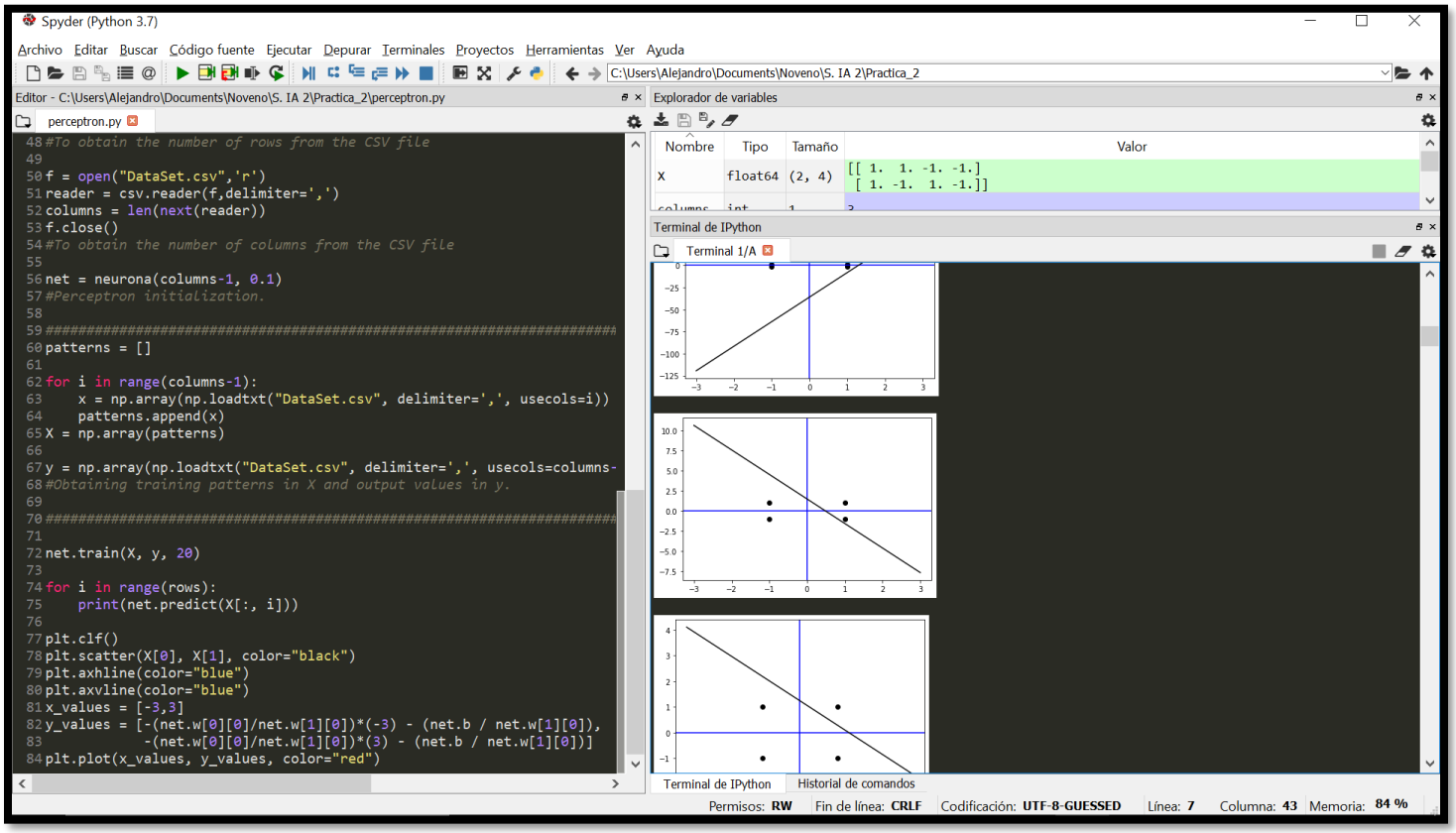
```
def graph(w_values):
    plt.clf()
    plt.scatter(X[0], X[1], color="black")
    plt.axhline(color="blue")
    plt.axvline(color="blue")
    x_values = [-3,3]
    y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
                -(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
    plt.plot(x_values, y_values, color="black")
    plt.pause(0.05)
```

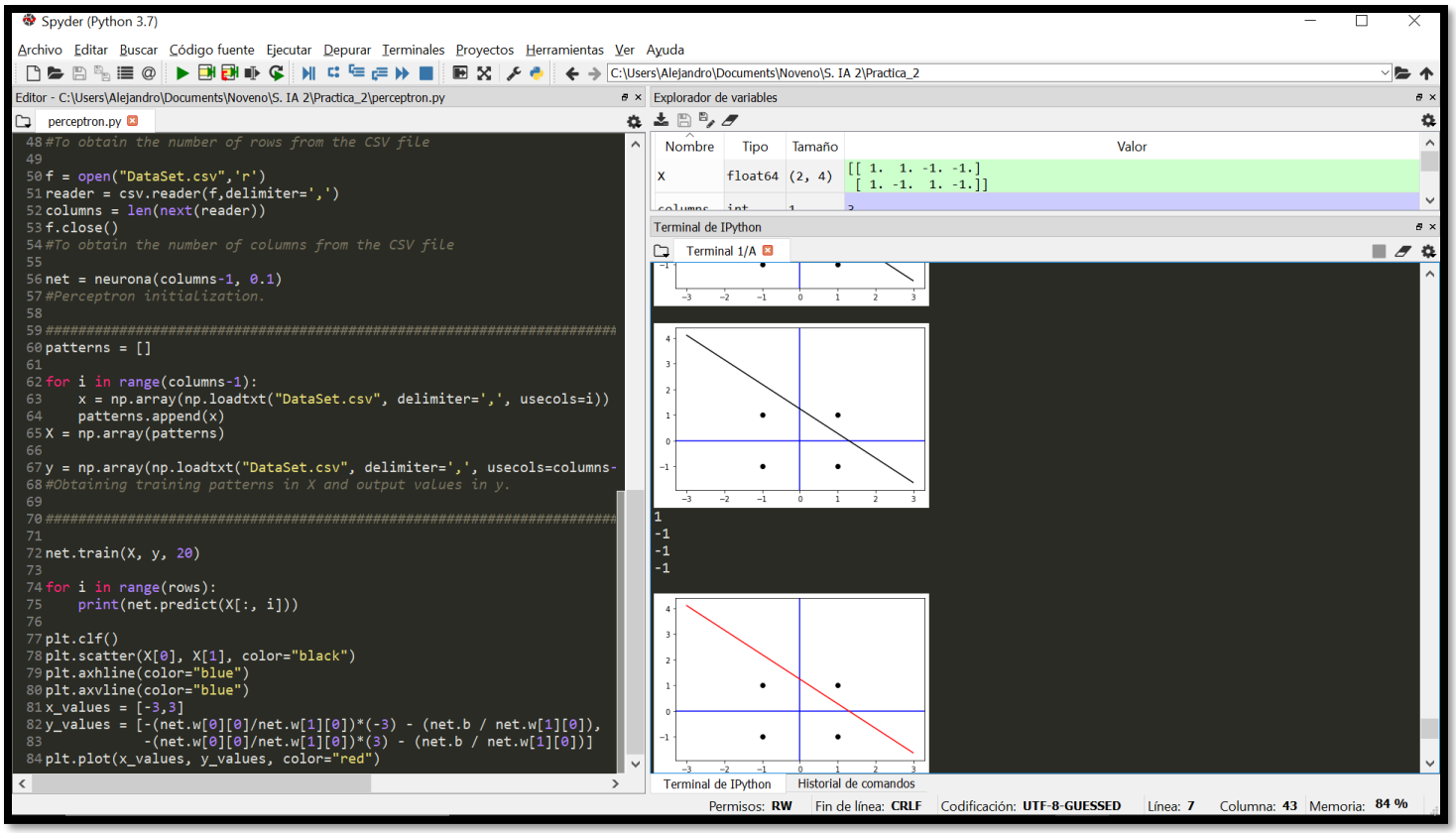
Para graficar la línea de la neurona en tiempo real limpiaremos el plano, dibujaremos los puntos a clasificar y los ejes, calcularemos la línea, la imprimiremos, y esperaremos una corta cantidad de tiempo antes de mostrar la siguiente.

Limpiar el plano al inicio y no al final nos evita perder el registro del último resultado, que de hecho es el más importante.

Resultados


Spyder (Python 3.7)
Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda
C:\Users\Alejandro\Documents\Noveno\S. IA 2\Practica_2
Editor - C:\Users\Alejandro\Documents\Noveno\S. IA 2\Practica_2\perceptron.py
perceptron.py
#To obtain the number of rows from the CSV file
f = open("DataSet.csv",'r')
reader = csv.reader(f,delimiter=',')
columns = len(next(reader))
f.close()
#To obtain the number of columns from the CSV file
net = neurona(columns-1, 0.1)
#Perceptron initialization.
patterns = []
for i in range(columns-1):
x = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=i))
patterns.append(x)
X = np.array(patterns)
y = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=columns-
#Obtaining training patterns in X and output values in y.
net.train(X, y, 20)
for i in range(rows):
print(net.predict(X[:, i]))
plt.clf()
plt.scatter(X[0], X[1], color="black")
plt.axhline(color="blue")
plt.axvline(color="blue")
x_values = [-3,3]
y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
-(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
plt.plot(x_values, y_values, color="red")
Explorador de variables
Nombre Tipo Tamaño Valor
X float64 (2, 4) [[1. 1. -1. -1.]
[1. -1. 1. -1.]]
Terminal de IPython
Terminal 1/A
Terminal de IPython Historial de comandos
Permisos: RW Fin de línea: CRLF Codificación: UTF-8-GUESSED Línea: 7 Columna: 43 Memoria: 83 %


Spyder (Python 3.7)
Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda
C:\Users\Alejandro\Documents\Noveno\S. IA 2\Practica_2
Editor - C:\Users\Alejandro\Documents\Noveno\S. IA 2\Practica_2\perceptron.py
perceptron.py
#To obtain the number of rows from the CSV file
f = open("DataSet.csv",'r')
reader = csv.reader(f,delimiter=',')
columns = len(next(reader))
f.close()
#To obtain the number of columns from the CSV file
net = neurona(columns-1, 0.1)
#Perceptron initialization.
patterns = []
for i in range(columns-1):
x = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=i))
patterns.append(x)
X = np.array(patterns)
y = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=columns-
#Obtaining training patterns in X and output values in y.
net.train(X, y, 20)
for i in range(rows):
print(net.predict(X[:, i]))
plt.clf()
plt.scatter(X[0], X[1], color="black")
plt.axhline(color="blue")
plt.axvline(color="blue")
x_values = [-3,3]
y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
-(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
plt.plot(x_values, y_values, color="red")
Explorador de variables
Nombre Tipo Tamaño Valor
X float64 (2, 4) [[1. 1. -1. -1.]
[1. -1. 1. -1.]]
Terminal de IPython
Terminal 1/A
Terminal de IPython Historial de comandos
Permisos: RW Fin de línea: CRLF Codificación: UTF-8-GUESSED Línea: 7 Columna: 43 Memoria: 84 %



Lo que podemos observar en las imágenes anteriores son los resultados del entrenamiento en vivo: cómo se mueve y adapta la línea de la neurona a los resultados esperados, para finalmente imprimir la línea definitiva en rojo, y mostrar la última predicción.

La prueba en tiempo real no se muestra en múltiples impresiones como se ve aquí, sino en una ventana externa que se modifica de manera dinámica; sin embargo, resultaba bastante difícil tomar impresiones de pantalla de esa manera, ya que el tiempo de cambio es bastante rápido, además de que no es posible demostrar que es la misma ejecución, a diferencia de mostrar varias gráficas el mismo tiempo como en este caso. Para hacer que la gráfica se muestre como debería es necesario ingresar el comando %matplotlib qt en la consola de comandos de Spyder antes de ejecutar el programa.

Conclusión

Como podemos observar, el aprendizaje del perceptrón muestra resultados bastante satisfactorios por lo menos a la hora de trabajar con pocos datos; aun así, por su arquitectura es bastante posible que trabaje de manera muy exitosa con un número mayor de datos y patrones de aprendizaje, aunque eso quedará pendiente a demostrarse.

Ahora que hemos realizado el aprendizaje de esta neurona podemos presenciar la confirmación del teorema del perceptrón, que nos indica que si tenemos dos conjuntos de datos que sean linealmente separables, el perceptrón converge en un número finito de iteraciones.

Así pues, podemos confirmar ahora que tenemos un perceptrón funcional y adaptable a diferentes situaciones, obteniendo datos externos en lugar de internos a mano. Los resultados fueron más que satisfactorios.

Código

```
import numpy as np
import matplotlib.pyplot as plt
import csv

def graph(w_values):
    plt.clf()
    plt.scatter(X[0], X[1], color="black")
    plt.axhline(color="blue")
    plt.axvline(color="blue")
    x_values = [-3,3]
    y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
                -(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
    plt.plot(x_values, y_values, color="black")
    plt.pause(0.05)

class neurona:
    def __init__(self, dim, eta):   #Dimension y coeficiente de aprendizaje.
        self.n = dim
        self.eta = eta
        self.w = -1 + 2 * np.random.rand(dim, 1)  #x = min + (max - min)*rand()
        self.b = -1 + 2 * np.random.rand()

    def predict(self, x):
        y = np.dot(self.w.transpose(), x) + self.b
        if y >= 0:
            return 1
        else:
            return -1

    def train(self, X, y, epochs):  #x = matriz de entrenamiento, y = vector con resultado esperados, epochs =
épocas.
        n, m = X.shape
        #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
        for i in range(epochs):
            for j in range(m):
                y_pred = self.predict(X[:, j])
                #what that line did is sliced the array,
                #taking all rows (:) but keeping the column (j)
                if y_pred != y[j]:  #Si nuestro estimado es diferente a nuestro esperado, entrenamos.
                    self.w += self.eta*(y[j] - y_pred) * X[:, j].reshape(-1, 1)
                    self.b += self.eta*(y[j] - y_pred)
                graph(self.w)

##########################################################################################
########################

file = open("DataSet.csv")
rows = len(file.readlines())
file.close()
#To obtain the number of rows from the CSV file
```

```
f = open("DataSet.csv",'r')
reader = csv.reader(f,delimiter=',')
columns = len(next(reader))
f.close()
#To obtain the number of columns from the CSV file

net = neurona(columns-1, 0.1)
#Perceptron initialization.

################################################################################
patterns = []

for i in range(columns-1):
    x = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=i))
    patterns.append(x)
X = np.array(patterns)

y = np.array(np.loadtxt("DataSet.csv", delimiter=',', usecols=columns-1))
#Obtaining training patterns in X and output values in y.

################################################################################

net.train(X, y, 20)

for i in range(rows):
    print(net.predict(X[:, i]))

plt.clf()
plt.scatter(X[0], X[1], color="black")
plt.axhline(color="blue")
plt.axvline(color="blue")
x_values = [-3,3]
y_values = [-(net.w[0][0]/net.w[1][0])*(-3) - (net.b / net.w[1][0]),
            -(net.w[0][0]/net.w[1][0])*(3) - (net.b / net.w[1][0])]
plt.plot(x_values, y_values, color="red")
```

Link al repositorio

- https://github.com/TheGenesisX/S_IA_2/tree/master/Practica_2