



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ZUMPANGO

INGENIERÍA EN COMPUTACIÓN



REDES NEURONALES

IMPLEMENTACIÓN DE UN PERCEPTRÓN SIMPLE

Alumno: Martín Márquez Cervantes

Profesor: Dr. Asdrúbal López Chau

Fecha de entrega: 15 de Octubre 2019

Introducción

Para encontrar una solución analítica en un perceptrón sencillo de dos o tres variables, es necesario graficar y encontrar una solución mediante la ecuación de una recta: $y = mx + b$. Esto se hace cada vez más complicado debido a la imposibilidad de visualizar cuatro o más variables en un plano por el momento y el trabajo de calcularlo a mano. Rosenblatt en su tesis de doctorado nos obsequio el algoritmo para poder generar los pesos sinápticos automáticamente con una idea bastante ingeniosa que se implementara en el siguiente trabajo.

El algoritmo es simple pero poderoso, esté trata de buscar una solución, una frontera de decisión que nos permita clasificar las salidas de un perceptrón, consta de dos bucles finitos y un condicional para evaluar los casos.

Algorithm 1: Entrenamiento Perceptrón

Input: ω inicial, X, Y

Output: Neurona entrenada

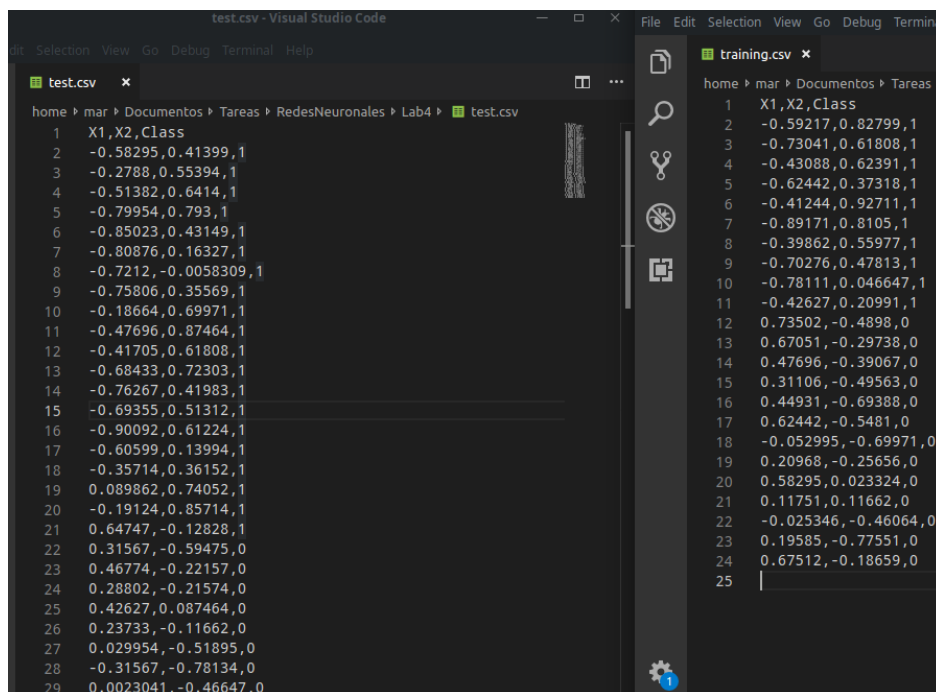
```
1 foreach  $x_i \in X$  do
2   if  $\omega^T x_i \geq 0$  then
3      $y = 1$ 
4   else
5      $y = 0$ 
6    $\omega = \omega + (y_i - y)x_i$ 
```

Figura 1: Algoritmo perceptrón de Rosenblatt

Este algoritmo nos arrojará un vector, con sus respectivos pesos sinápticos ya calibrados por así decirlo, que nos ayudara con el entrenamiento de nuestro perceptrón.

Desarrollo

La implementación empieza con los datos de prueba y de entrenamiento en este caso:

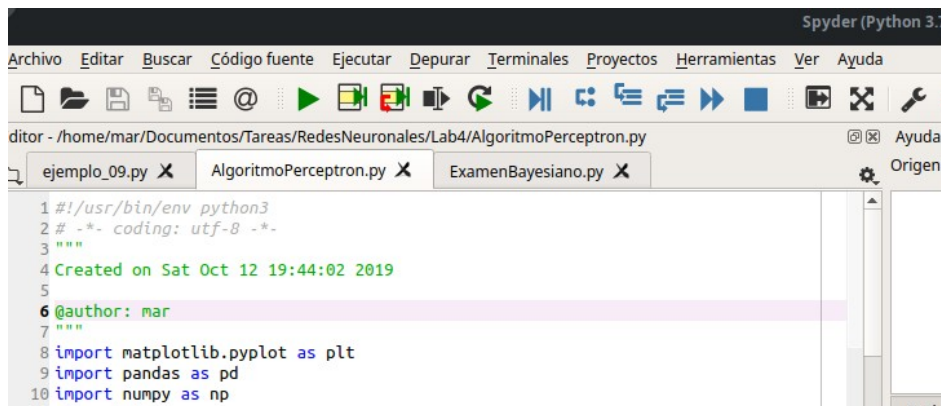


```
test.csv
1 X1,X2,Class
2 -0.58295,0.41399,1
3 -0.2788,0.55394,1
4 -0.51382,0.6414,1
5 -0.79954,0.793,1
6 -0.85023,0.43149,1
7 -0.80876,0.16327,1
8 -0.7212,-0.0058309,1
9 -0.75806,0.35569,1
10 -0.18664,0.69971,1
11 -0.47696,0.87464,1
12 -0.41705,0.61808,1
13 -0.68433,0.72303,1
14 -0.76267,0.41983,1
15 -0.69355,0.51312,1
16 -0.90092,0.61224,1
17 -0.60599,0.13994,1
18 -0.35714,0.36152,1
19 0.089862,0.74052,1
20 -0.19124,0.85714,1
21 0.64747,-0.12828,1
22 0.31567,-0.59475,0
23 0.46774,-0.22157,0
24 0.28802,-0.21574,0
25 0.42627,0.087464,0
26 0.23733,-0.11662,0
27 0.029954,-0.51895,0
28 -0.31567,-0.78134,0
29 0.0023041,-0.46647,0

training.csv
1 X1,X2,Class
2 -0.59217,0.82799,1
3 -0.73041,0.61808,1
4 -0.43088,0.62391,1
5 -0.62442,0.37318,1
6 -0.41244,0.92711,1
7 -0.89171,0.8105,1
8 -0.39862,0.55977,1
9 -0.70276,0.47813,1
10 -0.78111,0.046647,1
11 -0.42627,0.20991,1
12 0.73502,-0.4898,0
13 0.67051,-0.29738,0
14 0.47696,-0.39067,0
15 0.31106,-0.49563,0
16 0.44931,-0.69388,0
17 0.62442,-0.5481,0
18 -0.052995,-0.69971,0
19 0.20968,-0.25656,0
20 0.58295,0.023324,0
21 0.11751,0.11662,0
22 -0.025346,-0.46064,0
23 0.19585,-0.77551,0
24 0.67512,-0.18659,0
25
```

Figura 2 : Data-sets

Nuestros archivos test.csv y training.csv que contienen la data necesaria para entrenar y probar el algoritmo, nuestro entorno de desarrollo spyder:



```

1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sat Oct 12 19:44:02 2019
5
6@author: mar
7"""
8import matplotlib.pyplot as plt
9import pandas as pd
10import numpy as np
```

Figura 3 : Bibliotecas utilizadas

Las bibliotecas que utilizaremos durante la implementación constarán de la biblioteca pandas, numpy y matplotlib.

La programación en este caso es orientado a objetos es por eso que necesitamos de una clase en este caso la clase sera MyNeuron que contendrá los métodos para la implementación del algoritmo, el flujo empieza de esta manera:

```
109 #Instancia de la clase MyNeuron
110 clf = MyNeuron()
111 #carga de la data training
112 datos = pd.read_csv("training.csv")
113 #X matriz pandas Y vector de pandas
114 X=datos.iloc[:,range(0,2)]
115 Y=datos.iloc[:,range(2,3)]
116 #método
117 clf.training(X,Y)
```

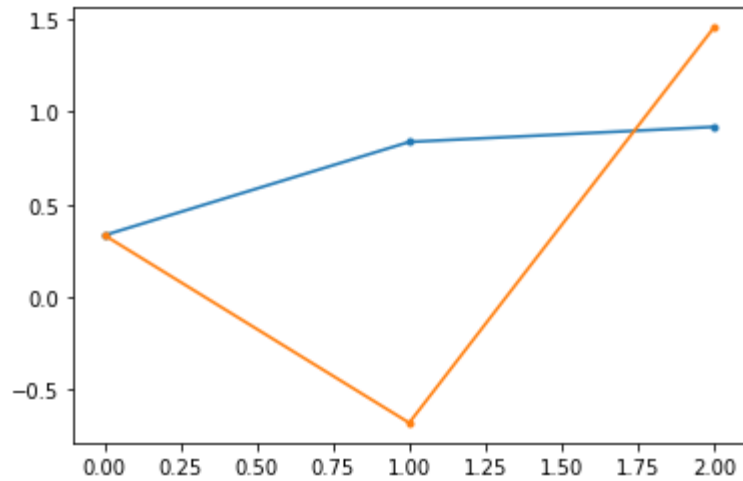
Figura 4: Implementación

Instanciando la clase, cargando los datos de entrenamiento y extraer las columnas X que representa los datos e Y que representa la clase o clasificación de esos datos, y pasandolos como parametros a el método training de la clase. El método training:

```
def training(self,X,Y):
    w = np.random.rand(3)
    xOnes = np.ones(int((X.size)/2)).reshape((23,1))
    datos.insert(0,"Ones",xOnes)
    X1 = np.array(datos.iloc[:,range(0,3)])
    Y = np.array(Y)
    wInicial = w
    XN = int(X.size/3)
    for i in range(1,21):
        for j in range(XN):
            if np.dot(w,X1[j]) >= 0:
                y=1
            else:
                y=0
            w=w+(Y[j]-y)*X1[j]
    self.WG=w
    #impresión de resultados
    print("w Inicial: "+str(wInicial))
    print("w Final: "+str(w))
    #graficación de vectores
    plt.plot(wInicial,'.-')
    plt.plot(w,'.-')
    print('Línea azul W Inicial')
    print('Línea naranja W Final aplicando algoritmo')
```

Figura 5 : Método training.

El método se encuentra completamente comentado en el código y a grandes rasgos inicia un vector pseudoaleatorio que se modificara con el algoritmo, la matriz X se complementa con números unos para que sea una matriz Nx3 y poder realizar la matriz transpuesta con w de 1x3: $W^T X$ e implementa el algoritmo previamente visto, actualizando w y repitiendo veinte veces.



```
AlgoritmoPerceptron.py', wdir='/home/mar/Docume
w Inicial: [0.33426056 0.83797614 0.91923591]
w Final: [ 0.33426056 -0.67815386 1.45568291]
#Fin del Algoritmo
```

Figura 6 : Grafica de los vectores w (Pesos sinápticos), Azul w inicial, Naranja w final.

Una vez obtenido w podemos seguir pasando a la predicción del test-set.

```
#carga de la data test
datosTest = pd.read_csv("test.csv")
#Crear una matriz del mismo tamaño que las filas con nuevo metodo shape[0]
XTestOnes = np.ones(datosTest.shape[0]).reshape((datosTest.shape[0],1))

#insertarlas en la variable datos en el indice cero
datosTest.insert(0,"Ones",XTestOnes)

#asignar los nuevos datos a X(X1,X2,X3)
xTest = np.array(datosTest.iloc[:,range(0,3)])

#for de 0 hasta tamaño de fila de los datos shape[0], mandamos indice a método
#llenamos el array Predicción
Predicciones = []
for i in range(datosTest.shape[0]):
    Predicciones.append(clf.predic(i,xTest))
Predicciones = np.array(Predicciones)
#impresión
for i in range(datosTest.shape[0]):
    print("Indice " +str(i) + " prediccion " +str(Predicciones[i]))
#COMpara y calcula la tabla de confusión
ClaseTest = np.array(datosTest.iloc[:,3])
#Comparación es el método de las columnas Clasificación predicción predicción real
```

Figura 7: Flujo del algoritmo de predicción.

Básicamente se repite lo mismo, se crea una lista que almacenara las predicciones llamada “Predicciones”, en el primer ciclo llama a el método predic que retornara valores según la clasificación:

```
#predice si esta aprobado o no
def predic(self,i,xTest):
    #calcular y la salida de la prediccion
    y=np.dot(self.WG,xTest[i]) #producto interno entre w y x
    #clasificación
    if y>=0:
        return 1 #Aprobado
    else:
        return 0 #No aprobado
```

Figura 8: Método predic

Inicialmente Rosenblatt lo adecuo a lo que hoy se conoce como función de activación “Heaviside” que toma los valores de cero y uno de acuerdo a la salida del perceptrón (y) que es el resultado del producto interno entre w y x como ya se vio.

Usaremos esas predicciones para comparar y poder medir la precisión de nuestro perceptrón.

```
139 #Compara y calcula la tabla de confusión
140 ClaseTest = np.array(datosTest.iloc[:,3])
141 #Impresión en el método de los calculos Clasificación precisión, precisión,recall
142 clf.comparar(ClaseTest,Predicciones)
```

Figura 9: Flujo de algoritmo comparar.

Con este ultimo bloque llamamos al método comparar y le pasamos las etiquetas de test.csv y las Predicciones que obtuvimos.

```
def comparar(self,ClaseTest,Predicciones):
    TP = 0
    FP = 0
    FN = 0
    TN = 0
    #contar true positive, true negative, false negative, false positive
    for i in range(ClaseTest.shape[0]):
        if ClaseTest[i] ==1 and Predicciones[i]==1:
            TP +=1
        if ClaseTest[i] ==0 and Predicciones[i]==0:
            TN +=1
        if ClaseTest[i] ==1 and Predicciones[i]==0:
            FN +=1
        if ClaseTest[i] ==0 and Predicciones[i]==1:
            FP +=1
    print("\n\nTP = "+str(TP)+"\nTN = "+str(TN)+"\nFP = "+str(FP)+"\nFN = "+str(FN))
    #calculo de Precisión de clasificación
    ClassificationAccuary = ( (TP+TN)/(TP+TN+FN+FP) ) *100
    print("\nPresición de clasificación: "+str(ClassificationAccuary)+" %")

    #calculo de Presición
    Presicion = TP/(TP+FP)*100
    print("Presición : "+str(Presicion)+" %")

    #calculo de Presición
    Recall = TP/(TP+FN)*100
    print("Recall : "+str(Recall)+" %")

    #calculo de F-Score
    FScore = 2* ( (Presicion*Recall)/(Presicion+Recall) )
    print("F-Score : "+str(FScore))
```

Figura 10: Método comparar

Comparando la entrada y salida podemos extraer los verdaderos positivos, verdaderos negativos, falsos positivos, falsos negativos y calcular la exactitud de la clasificación, precisión, recall y f-score. Esta es la culminación de la implementación del algoritmo de Rosenblatt.

Pruebas Desarrolladas

Los resultados arrojados son los siguientes:

```
TP = 19
TN = 13
FP = 2
FN = 1

Presición de clasificación: 91.42857142857143 %
Presición : 90.47619047619048 %
Recall : 95.0 %
F-Score : 92.6829268292683
```

Figura 11: Resultados de la implementación.

En donde TP representa los datos positivos que coincidieron las clases de test con la predicción y los TN con los negativos, mientras que FP representa un error en la predicción afirmando que es positivo cuando en realidad es negativo y FN afirmando que es negativo cuando en realidad es positivo.

El dato de precisión de clasificación nos indica en porcentaje que tan bueno es el modelo.

El de precisión nos responde la pregunta ¿Qué tan buenos somos para clasificar positivos? en forma de porcentaje.

El dato de recall nos responde en porcentaje cuantas clases positivas fue identificada correctamente.

El F-score nos dice que tan equilibrado esta el sistema.

Conclusiones

Debido a las pruebas y resultados considero que realice de manera correcta la implementación del algoritmo de perceptrón simple, esta tarea me ayudo mucho a reforzar conocimientos y entender muy bien el tema.

Referencias

Sokolova, Marina & Japkowicz, Nathalie & Szpakowicz, Stan. (2006). Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. AI 2006: Advances in Artificial Intelligence, Lecture Notes in Computer Science. Vol. 4304. 1015-1021. 10.1007/11941439_114.

von der Malsburg, Christoph. (1986). Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Brain Theory. 245-248. 10.1007/978-3-642-70911-1_20.

CIBERGRAFIA:

1.-https://www.researchgate.net/publication/225215404_Beyond_Accuracy_F-Score_and_ROC_A_Family_of_Discriminant_Measures_for_Performance_Evaluation

2.-https://www.researchgate.net/publication/285851582_Frank_Rosenblatt_Principles_of_Neurodynamics_Perceptrons_and_the_Theory_of_Brain_Mechanisms