Clase2 IMA539

Alejandro Ferreira Vergara

March 20, 2023

1 PERCEPTRON

1.1 ¿Qué aprenderemos hoy?

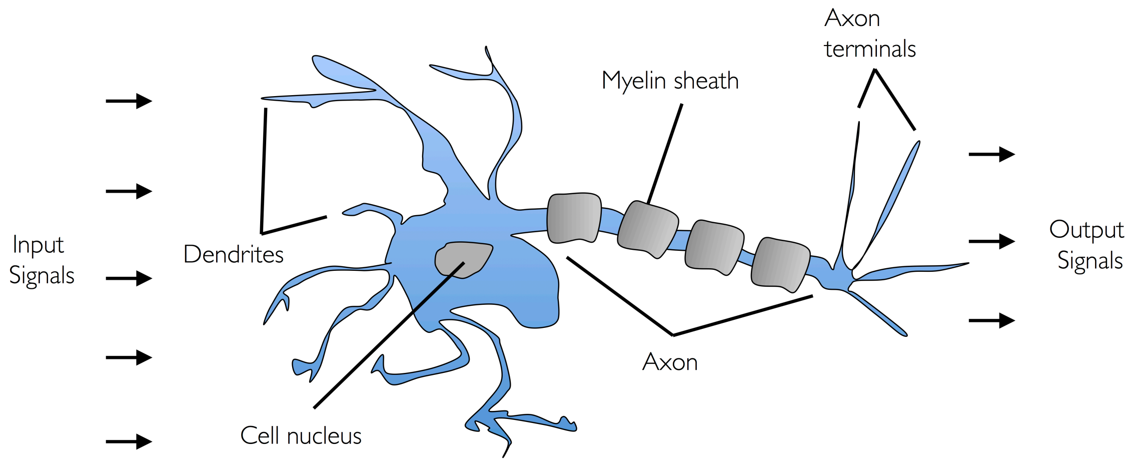
- **Concepto de Neurona Artificial**
- **Matemáticas detrás del algoritmo del Perceptron**
- **Implementaremos un Perceptron y lo entrenaremos en el conjunto de datos** ***iris***

1.2 Neuronas Artificiales (W. McCullock & W. Pitts, 1943)

[1]:
```python
from IPython.display import Image

Image(filename=r'clase2/2_1.png', width=600)
```

[1]:



1.3 Perceptron de Rosenblatt (1957)

- **Problema de Clasificación Binaria**

  clase positiva: 1

  clase negativa: -1

- **Vector del dato x y vector de pesos w:**

  $$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

- **Si $z = w_1x_1 ... + w_mx_m$, se define la Función de decisión $\phi(z)$.**

  $$\phi(z) = \begin{cases} 1 & \text{si } z \geq \theta \\ -1 & \text{en otro caso} \end{cases}$$

- Para simplificar, podemos llevar el umbral θ al lado izquierdo de la relación y definir $w_0 = -\theta$ y $x_0 = 1$. Luego, z se calcula como:

  $$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = w^Tx$$

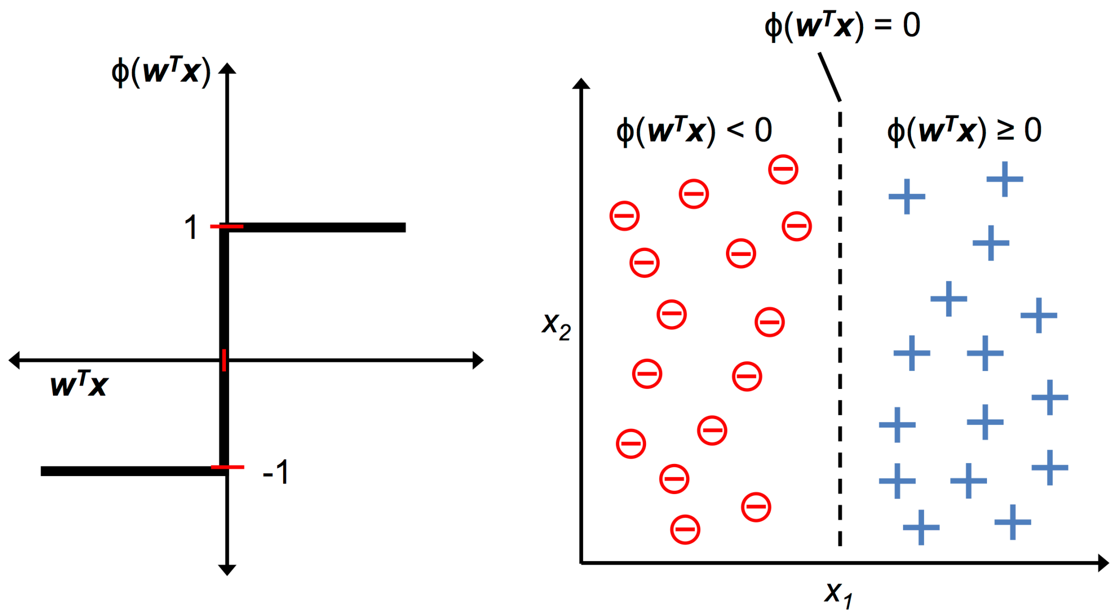
  Por lo tanto:

  $$\phi(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ -1 & \text{en otro caso} \end{cases}$$

- **A $w_0 = -\theta$, se le denomina unidad de sesgo**.

[2]:
```
Image(filename=r'clase2/2_2.png', width=600)
```

[2]:



1.4 Aprendizaje del Perceptron

La regla inicial del Perceptron de Rosenblatt es bastante simple y se puede resumir en los siguientes pasos:

- **Inicializar los pesos cercanos a cero o a pequeños números aleatorios.**

- **Para cada muestra de entrenamiento $x^{(i)}$:**

    1) **Calcular el valor de salida $\hat{y}$.**

    2) **Actualizar los pesos:**

  $w_j := w_j + \Delta w_j$

  El valor de Δw_j, que se utiliza para actualizar el peso w_j, se calcula mediante la **regla de aprendizaje del perceptron**:

  $\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$

  Donde η es la **tasa de aprendizaje** (normalmente una constante entre 0.0 y 1.0), $y^{(i)}$ es la **etiqueta de clase verdadera** de la i-ésima muestra de entrenamiento, e $\hat{y}^{(i)}$ es la **etiqueta de clase predicha**.

VEAMOS UN EJEMPLO:

- **Para un conjunto de datos bidimensional, podríamos escribir la actualización como:**

  $\Delta w_0 = \eta \left(y^{(i)} - output^{(i)} \right) 1$

  $\Delta w_1 = \eta \left(y^{(i)} - output^{(i)} \right) x_1^{(i)}$

  $\Delta w_2 = \eta \left(y^{(i)} - output^{(i)} \right) x_2^{(i)}$

- **En los casos que el Perceptron predice correctamente** (no hay actualización):

  $\Delta w_j = \eta \left(-1 - (-1) \right) x_j^{(i)} = 0$

  $\Delta w_j = \eta \left(1 - 1) \right) x_j^{(i)} = 0$

- **En el caso de una predicción errónea** (esto es lo que se busca):

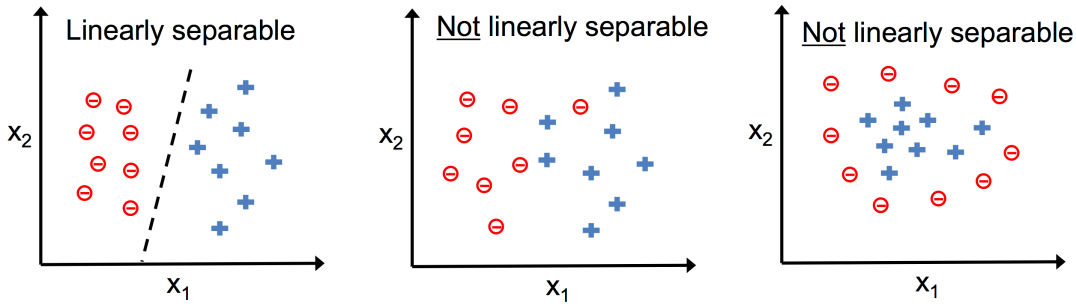
  $\Delta w_j = \eta \left(1 - -1 \right) x_j^{(i)} = \eta(2) x_j^{(i)}$

  $\Delta w_j = \eta \left(-1 - 1 \right) x_j^{(i)} = \eta(-2) x_j^{(i)}$

Es importante señalar que la convergencia del Perceptron sólo está garantizada si **las dos clases son linealmente separables y la tasa de aprendizaje es suficientemente pequeña**. Si las dos clases no se pueden separar mediante una frontera de decisión lineal, podemos establecer un número máximo de pasadas por el conjunto de datos de entrenamiento (épocas) y/o fijar un umbral para el número de errores de clasificación tolerados; de lo contrario, el perceptrón nunca dejaría de actualizar los pesos:

[3]:
```
Image(filename=r'clase2/2_3.png', width=750)
```

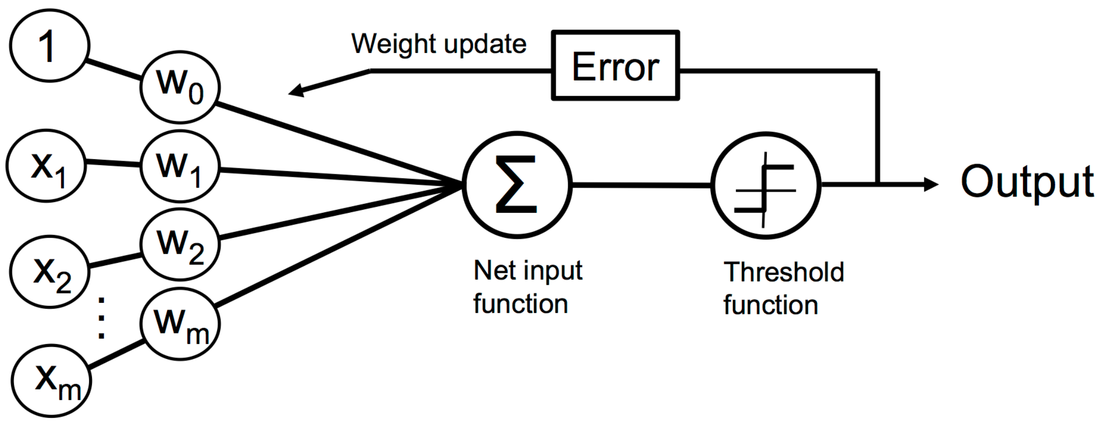
[3]:



1.5 Esquema resumen del modelo

[4]:
```
Image(filename=r'clase2/2_4.png', width=600)
```

[4]:



2 Implementación Perceptron con Python

[ ]:
```
import numpy as np

class Perceptron(object):
    """Perceptron classifier.

    Parametros
    ------------
    eta : float
      Learning rate (entre 0.0 y 1.0)
    n_iter : int
      Cantidad de épocas de entrenamiento.
    random_state : int
      Semilla del generador de números aleatorios para
```

```python
      la inicialización de pesos aleatorios.

    Atributos
    -----------
    w_ : 1d-array
      Vector de peso después del entrenamiento.
    errors_ : list
      Número de clasificaciones erróneas (actualizaciones) en cada época.

    """
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        """Entrenamiento.

        Parametros
        ----------
        X : {array-like}, shape = [n_samples, n_features]
          Vector de entrenamiento, donde n_samples es el número de muestras y
          n_features es el número de características.
        y : array-like, shape = [n_samples]
          Valor de salida.

        Returns
        -------
        self : object

        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        """Calcular entrada neta, z"""
```

```
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        """Etiqueta de clase después del paso unitario"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

[ ]:
```
import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪iris/iris.data',header=None,encoding='utf-8')

df.columns = ['sepal_len','sepal_wid','petal_len','sepal_wid','class']
df.head(5)
```

[ ]:
```
df.describe(include='all')
```

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

X = df.iloc[0:100, [0, 2]].values

plt.scatter(X[:50, 0], X[:50, 1],color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],color='blue', marker='x',␣
 ↪label='versicolor')

plt.xlabel('largo sétalo [cm]')
plt.ylabel('largo pétalo [cm]')
plt.legend(loc='upper left')

#plt.savefig('02_06.png', dpi=300)
plt.show()
```

[ ]:
```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)

for i in range(len(ppn.w_)):
    print('w[{}] = {}'.format(i,ppn.w_[i]))
```

[ ]:
```
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Épocas')
plt.ylabel('Número de actualizaciones')

#plt.savefig('02_07.png', dpi=300)
```

```
plt.show()
```

Tarea: Revisar el siguiente código para aprender su funcionamiento.

[ ]:
```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        if cl == -1:
            label = 'setosa'
        else:
            label = 'versicolor'
        plt.scatter(x=X[y == cl, 0],y=X[y == cl, 1],alpha=0.8,c=colors[idx],
                    marker=markers[idx], label=label,edgecolor='black')
```

[ ]:
```
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('largo sétalo [cm]')
plt.ylabel('largo pétalo [cm]')
plt.legend(loc='upper left')

#plt.savefig('02_08.png', dpi=300)
plt.show()
```

2.1 Perceptron con Scikit Learn

[ ]:
```
from sklearn.linear_model import Perceptron

ppn_skl = Perceptron(max_iter=10, alpha=0.1, random_state=1, n_jobs=-1)
ppn_skl.fit(X, y)

plot_decision_regions(X, y, classifier=ppn_skl)
plt.xlabel('largo sétalo [cm]')
plt.ylabel('largo pétalo [cm]')
plt.legend(loc='upper left')
```

```
plt.show()
```

[ ]:

```
```