PERCEPTRON

¿Qué aprenderemos hoy?

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

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

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

Image(filename=r'Imagenes_Clase_02/2_1.png', width=600)
```

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 $z = w_1 x_1 + \ldots + w_m x_m$, 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:

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = w^T x$$

  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.

In [ ]:
```python
Image(filename=r'Imagenes_Clase_02/2_2.png', width=600)
```

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 $\hat{y}^{(i)}$.
  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 ($\hat{y}^{(i)} = output^{(i)}$):

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

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

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

- 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:

In [ ]:
```python
Image(filename=r'Imagenes_Clase_02/2_3.png', width=750)
```

Esquema resumen del modelo

In [ ]:
```python
Image(filename=r'Imagenes_Clase_02/2_4.png', width=600)
```

Implementación Perceptron con Python

In [ ]:
```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
      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)
```

In [ ]:
```python
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_length','sepal_width','petal_length','petal_width','class']
df.head(5)
```

In [ ]:
```python
Image(filename=r'Imagenes_Clase_02/Flores.png', width=750)
```

In [ ]:
```python
df.tail(5)
```

In [ ]:
```python
df.describe()
```

In [ ]:
```python
df.iloc[45:55, :]
```

In [ ]:
```python
df.iloc[95:105, :]
```

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

# describe con include='all' es la que fuerza que se muestren todas variables
# incluyendo las categóricas

# Todas las variables deben tener el type correcto para que sean identificadas como
# tal por describe (especialmente al leer de csv)
```

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

y = df.iloc[:100, 4].values
print(y)
```

In [ ]:
```python
name_clases=list(np.unique(y))
y_numeric = np.where(y == name_clases[0], -1, 1)
print(y_numeric)
```

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

variable_names=list(df.columns[[0,2]])

#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.scatter(X[:50, 0], X[:50, 1],color='red', marker='o', label=name_clases[0])
plt.scatter(X[50:100, 0], X[50:100, 1],color='blue', marker='x', label=name_clases[1])


plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
plt.legend(loc='upper right')

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

In [ ]:
```python
ppn = Perceptron(eta=0.01, n_iter=50)
ppn.fit(X, y_numeric)

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

In [ ]:
```python
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.

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

def plot_decision_regions(X, y, classifier, clases_names=['clase 0','clase 1'], resolution=0.02):
    markers = ('s', 'o', '^', 'v','x')
    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'
            label = clases_names[0]
        else:
            #label = 'versicolor'
            label = clases_names[1]
        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')
```

In [ ]:
```python
plot_decision_regions(X, y_numeric, classifier=ppn,clases_names=name_clases)
plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')

plt.legend(loc='upper right')

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

Perceptron con Scikit Learn

In [ ]:
```python
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_numeric)

plot_decision_regions(X, y_numeric, classifier=ppn_skl, clases_names=name_clases)
plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
#plt.xlabel('largo sétalo [cm]')
#plt.ylabel('largo pétalo [cm]')
plt.legend(loc='upper right')

plt.show()
```

Ejercicio

1. Encontrar dos variables y dos tipos de flores tal que las clases no sean linealmente separables

2. Plotear ambas variables en el plano
3. Correr el algoritmo Perceptron con 10 iteraciones
4. Plotear el resultado con *plot_decision_regions()*