Perceptron Simple

March 18, 2019

1 Importamos las librerias necesarias

In [6]:
```
#!/usr/bin/env python3
# coding: utf-8

# # Importar librerías
from pylab import rand, plot, show, norm
```

2 Definimos la clase Perceptron

In [7]:
```
# # Clase Perceptron
class Perceptron:
    """
    Perceptron Simple

    Parametros:
        w_: array-1d
            Pesos actualizados después del ajuste.

        tasaApren_: float
            Tasa de aprendizaje.
    """
    def __init__(self, w_= rand(2)*2-1, tasaApren_ = 0.1):
        """
        Metodo constructor del preceptron,
        inicialza los valores por defecto.
        """
        self.w = w_                         # Vector w, representa los pesos.
        self.tasaApren = tasaApren_         # Tasa de aprendizaje.

    def respuesta(self, x):
        """
        Salida del perceptron, aplica el producto punto entre w (pesos) y x (data).

        Parametros:
            x: list, forma [coordenada x, coordenada y]
```

```
            Data que se esta analizando.

        Retorna:
            int: Si el producto punto es mayor o igual a cero devuelve '1' de lo contr
        """
        y = (x[0] * self.w[0]) + (x[1] * self.w[1])     # Producto punto entre w y x.

        if y >= 0:
            return 1
        else:
            return -1

    def actualizarPesos(self, x, error):
        """
        Metodo encargado de actualizar el valor de los pesos en el vector w:
            w(t+1) = w(t) + (tasaApren * error * x)

            Donde:
                w(t+1): Es el peso para la siguiente iteracion de aprendizaje.
                w(t): Es el peso para la iteracion actual de aprendizaje.
                tasaApren: Tasa de aprendizaje.
                error: (respuesta deseada) - (respuesta del perceptron).
                x: Coordenada actual.

        Parametros:
            x: list, forma [coordenada x, coordenada y]
                Data que se esta analizando.
        """
        self.w[0] += self.tasaApren * error * x[0]
        self.w[1] += self.tasaApren * error * x[1]

    def entrenamiento(self, data):
        """
        Metodo encargado de entrenar el perceptron simple, el vector en los datos, cad
        el tercer elemento (x[2]) debe ser etiquetado (salida deseada)

        Parametros:
            data: list, forma [[x1, y1, resp1], [x2, y2, resp2], ... , [xn, yn, respn]
                Vector con los datos, cada uno debe tener la forma, coordenada x, coor
        """
        aprendio = False                        # Determina si el perceptron aprendio
        iteracion = 0                           # Nunero de iteracion que le tomo al p

        while not aprendio:                     # Mientras no aprenda.
            globalError = 0.0                   # Mantiene el error general que se va

            for x in data:                      # Recorremos los datos.
                r = self.respuesta(x)           # Obtenemos la respuesta del perceptro
```

```
                if x[2] != r:                               # Si la respuesta no es la des
                    error = x[2] - r                        # El error en la iteracion se
                    self.actualizarPesos(x, error)          # Se actualiza los pesos con e
                    globalError += abs(error)               # Se actualiza el error genera

            iteracion += 1                                  # Se contabiliza la iteracion

            if globalError == 0.0 or iteracion >= 100:      # Criterio de salida: si e
                print("Iteraciones {}".format(iteracion))   # Se imprime las iteracion
                aprendio = True                             # Salida del perceptron.
```

3 Definimos el metodo para generar los datos de prueba.

```
In [8]: def datosGenerados(n):
            """
            Metodo encargado de generar un conjunto de datos de prueba, linealmente separables
            con la siguiente forma:
                [[x1, y1, resp1], [x2, y2, resp2], ... , [xn, yn, respn]]

                Donde:
                    xn: Representa la coordenada X.
                    yn: Representa la coordenada Y.
                    respn: Representa la etiqueta de la muestra.

            Parametros:
                n: int
                    Numero de datos que se desea generar.

            Retorna:
                list: Lista con los datos con la siguiente forma:
                    [[x1, y1, resp1], [x2, y2, resp2], ... , [xn, yn, respn]]
            """
            xb = (rand(n)*2-1)/2-0.5
            yb = (rand(n)*2-1)/2+0.5
            xr = (rand(n)*2-1)/2+0.5
            yr = (rand(n)*2-1)/2-0.5
            datos = []

            for i in range(len(xb)):
                datos.append([xb[i],yb[i],1])
                datos.append([xr[i],yr[i],-1])

            return datos
```

4 Finalmente definimos el main para probar el Perceptron

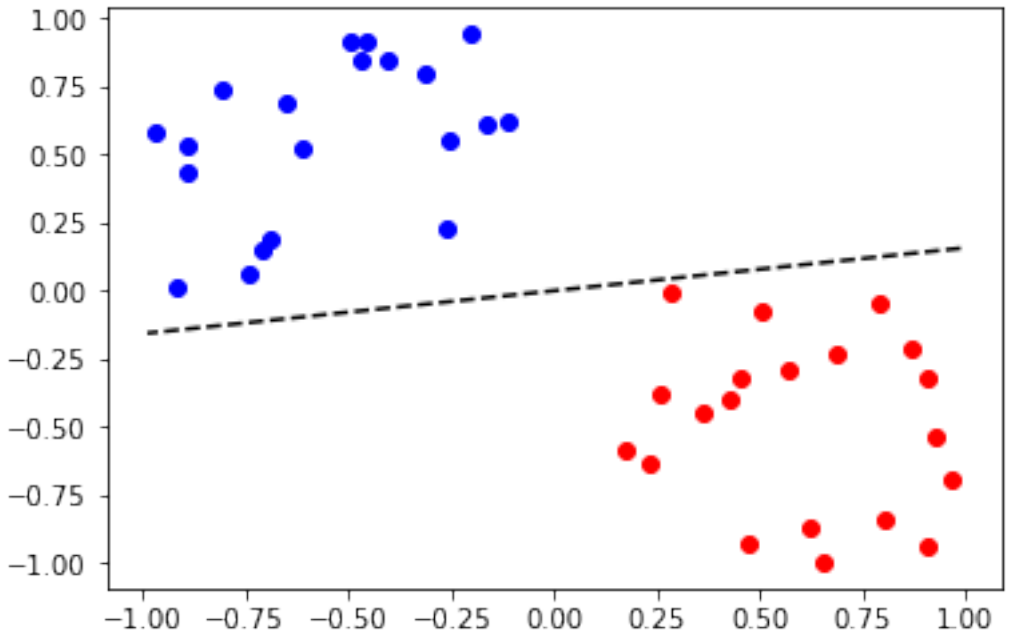
```
In [9]: if __name__ == "__main__":
            datosEntrenamiento = datosGenerados(30)              # Se genera los datos de prueb
            perceptron = Perceptron()                            # Se instancia del perceptron.
            perceptron.entrenamiento(datosEntrenamiento)         # Se entrena el perceptron con
            datosPrueba = datosGenerados(20)                     # Se genera los datos con los

            # Se prueba el perceptron con los datos de prueba.
            for x in datosPrueba:                                # Se recorre los datos de prue
                r = perceptron.respuesta(x)                      # Obtenemos la respuesta del p

                if r != x[2]:                                    # Verificamos si la respuesta
                    print ('error')                              # Si no es correcta, imprimimo
                if r == 1:
                    plot(x[0],x[1],'ob')                         # Si la respuesta es 1, lo pin
                else:
                    plot(x[0],x[1],'or')                         # De lo contrario, lo pintamos

            # Se gráfica una línea de separación, la cual es ortogonal a w.
            n = norm(perceptron.w)
            ww = perceptron.w/n
            ww1 = [ww[1],-ww[0]]
            ww2 = [-ww[1],ww[0]]
            plot([ww1[0], ww2[0]],[ww1[1], ww2[1]],'--k')
            show()
```

```
Iteraciones 2
```



Los puntos azules representan los datos de la primera clase, y los puntos rojos a la segunda, estos son los datos de pruebas que generamos para probar el perceptron, y la linea punteada es la linea de separacion que aprendio el perceptron durante el entrenamiento.