

Prácticas de Aprendizaje Automático

Trabajo 2: Complejidad de H y Modelos Lineales

Pablo Mesejo

Universidad de Granada

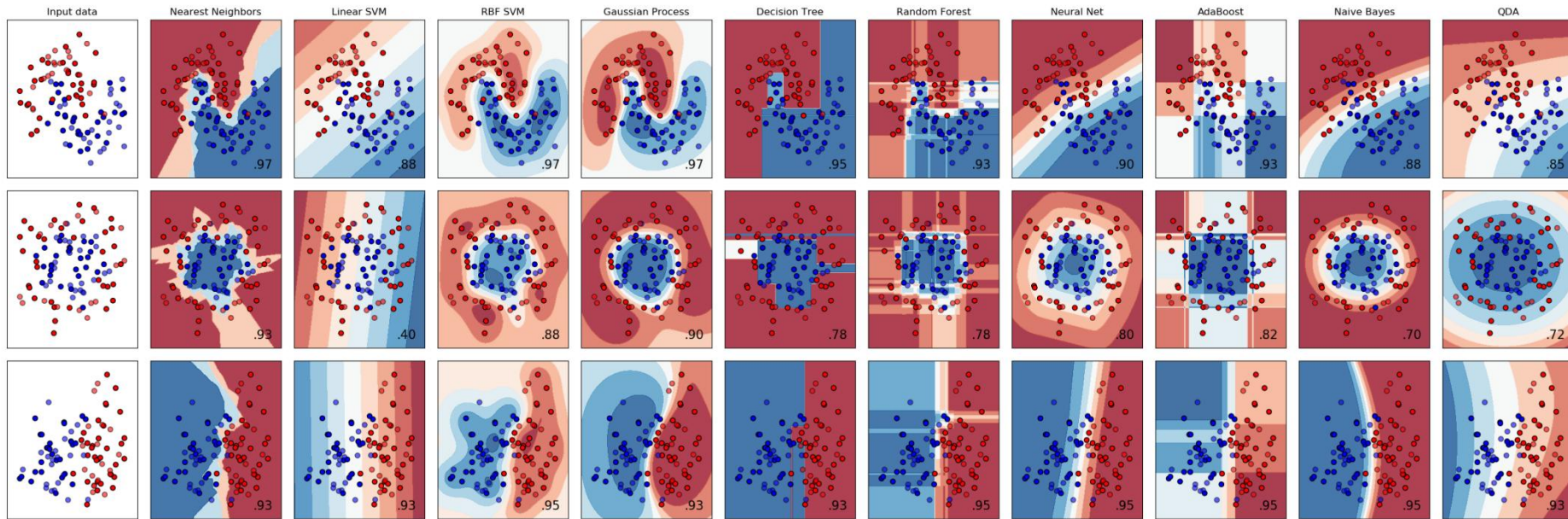
Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Interesante referencia para visualizar fronteras de decisión

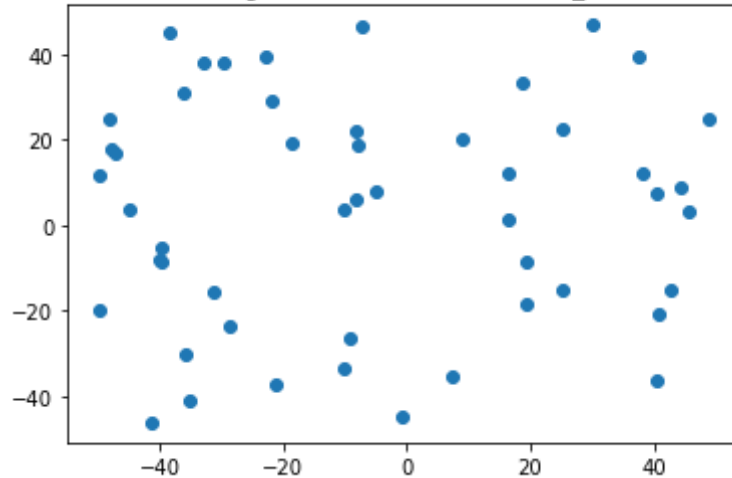


https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

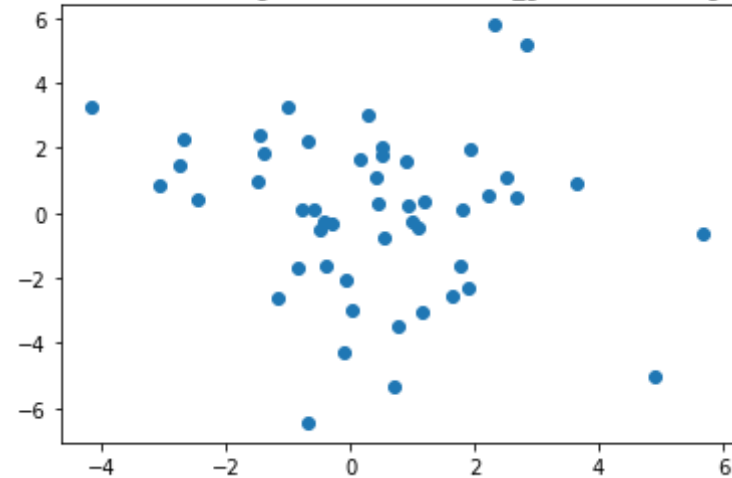
1. Ejercicio sobre la complejidad de H y el ruido

- Dibujar gráficas con nubes de puntos simuladas bajo ciertas condiciones

$N = 50$, $\text{dim} = 2$, $\text{rango} = [-50, +50]$ con `simula_unif(N, dim, rango)`



$N = 50$, $\text{dim} = 2$, $\text{sigma} [5,7]$ con `simula_gaus(N, dim, sigma)`



1. Ejercicio sobre la complejidad de H y el ruido

- Generar otra muestra de puntos 2D a la que vais a añadir una etiqueta usando el signo de la función $f(\mathbf{x}, \mathbf{y}) = \mathbf{y} - \mathbf{a}\mathbf{x} - \mathbf{b}$

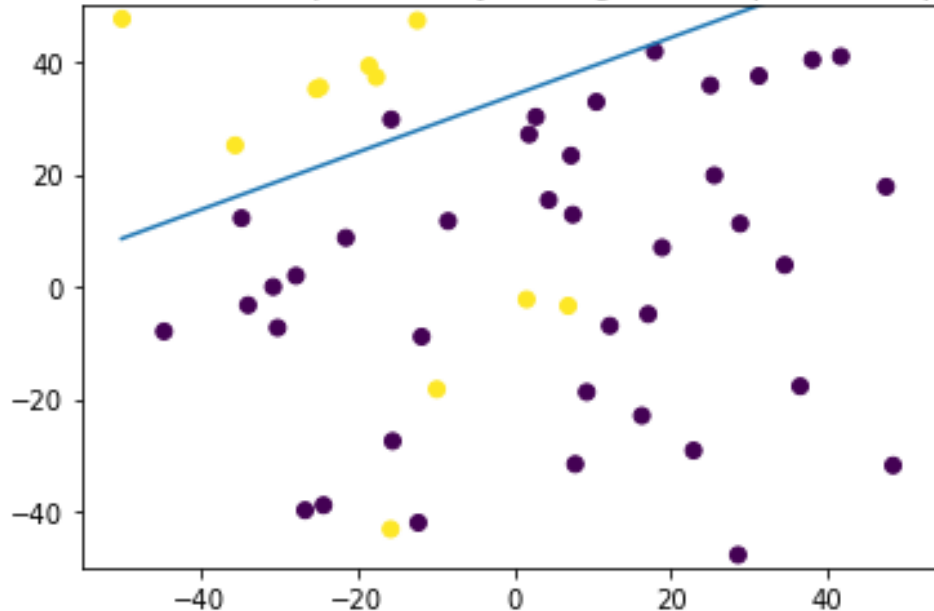


Misclassification
rate: 0.0%

1. Ejercicio sobre la complejidad de H y el ruido

- Modificar de forma aleatoria un 10% de las etiquetas positivas y otro 10% de las negativas

Datos (con un 10% de ruido por clase) y recta generada para el etiquetado inicial.



Misclassification
rate: 10.0%

1. Ejercicio sobre la complejidad de H y el ruido

- Emplear otras funciones para definir la frontera de clasificación de los puntos de la muestra en lugar de una recta

$$f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$$

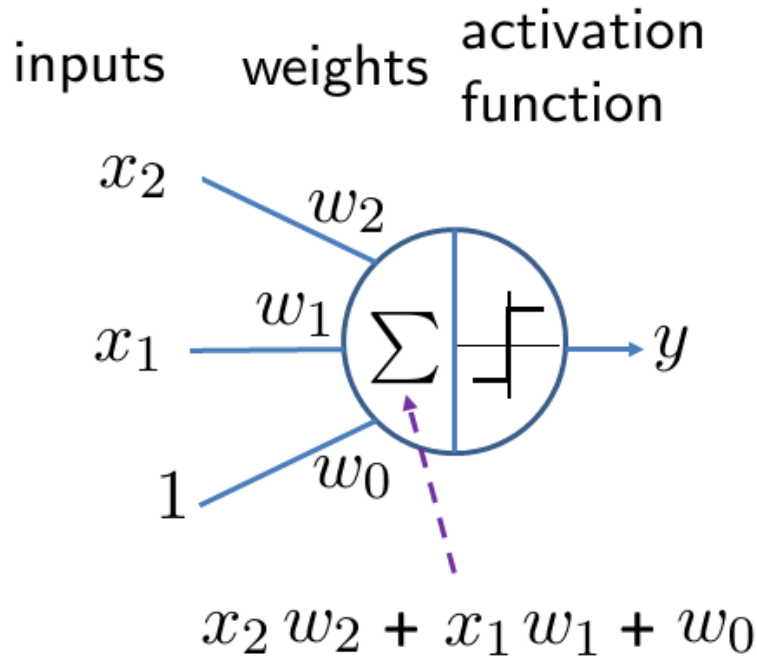
$$f(x, y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$$

$$f(x, y) = 0,5(x - 10)^2 - (y + 20)^2 - 400$$

$$f(x, y) = y - 20x^2 - 5x + 3$$

NOTA: se aconseja utilizar la función `plot_datos_cuad` proporcionada en `template_trabajo2.py`.

2. Modelos Lineales. Perceptron



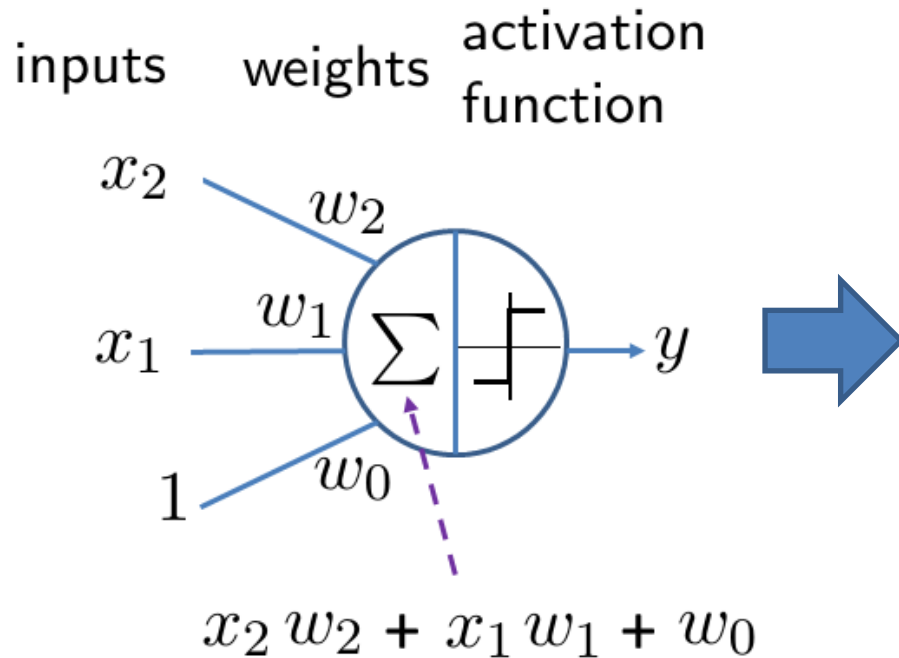
Referencias de apoyo sobre el perceptrón:

“Pattern Recognition and Machine Learning” (Christopher M. Bishop, 2006, págs. 192-196)

“Learning from Data” (Yaser S. Abu-Mostafa et al., 2012, págs. 5-8)

Perceptron Learning example: http://130.243.105.49/~lilien/ml/seminars/2007_02_01b-Janecek-Perceptron.pdf

2. Modelos Lineales. Perceptron



$$h(\mathbf{x}) = \text{sign} \left(\left(\sum_{i=1}^d w_i x_i \right) + w_0 \right)$$

Introduce an artificial coordinate $x_0 = 1$:

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=0}^d w_i x_i \right)$$

In vector form, the perceptron implements

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$$

2. Perceptron Learning Algorithm (PLA)

- Given the data set $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$
- Step.1: Fix $\mathbf{w}_{\text{ini}} = 0$
- Step.2: Iterate on the \mathcal{D} -samples improving the solution:
- repeat
 - For each $\mathbf{x}_i \in \mathcal{D}$ do
 - if: $\text{sign}(\mathbf{w}^T \mathbf{x}_i) \neq y_i$ then
 - update \mathbf{w} : $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + y_i \mathbf{x}_i$
 - else continue
 - End for
- Until No changes in a full pass on \mathcal{D}

2. Perceptron. Apuntes finales.

- Si **datos linealmente separables** → Convergencia garantizada
De lo contrario → No convergerá
- Es un algoritmo que **suele requerir muchas iteraciones para converger** (en problemas linealmente separables)
En nuestro caso, del orden de miles o decenas de miles...
- **Muy sensible a la inicialización**

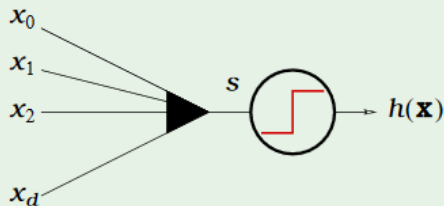
2. Modelos Lineales. Logistic Regression

A third linear model

$$s = \sum_{i=0}^d w_i x_i$$

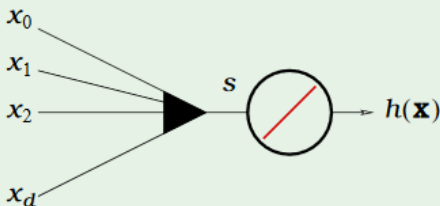
linear classification

$$h(\mathbf{x}) = \text{sign}(s)$$



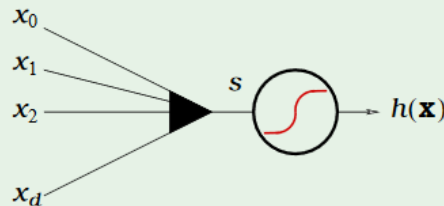
linear regression

$$h(\mathbf{x}) = s$$



logistic regression

$$h(\mathbf{x}) = \theta(s)$$



2. Modelos Lineales. Logistic Regression

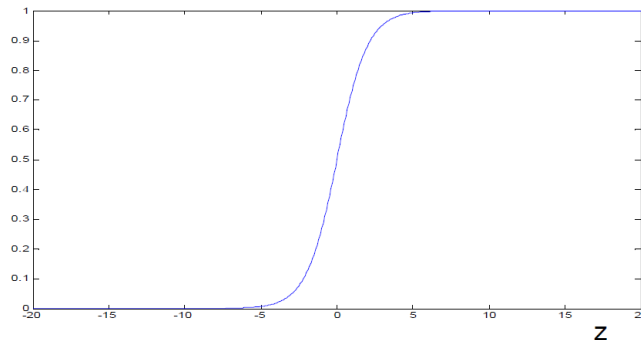
The LR classifier is defined as

$$\sigma(f(\mathbf{x}_i)) \begin{cases} \geq 0.5 & y_i = +1 \\ < 0.5 & y_i = -1 \end{cases}$$

where $\sigma(f(\mathbf{x})) = \frac{1}{1+e^{-f(\mathbf{x})}}$

The [logistic function](#) or [sigmoid function](#)

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



2. Modelos Lineales. Logistic Regression

Logistic regression algorithm

$$p(Y = 1|x) + p(Y = -1|x) = 1$$

RECOMENDACIÓN: N=1

- 1: Initialize the weights at $t = 0$ to $\mathbf{w}(0)$
- 2: **for** $t = 0, 1, 2, \dots$ **do**
- 3: Compute the gradient

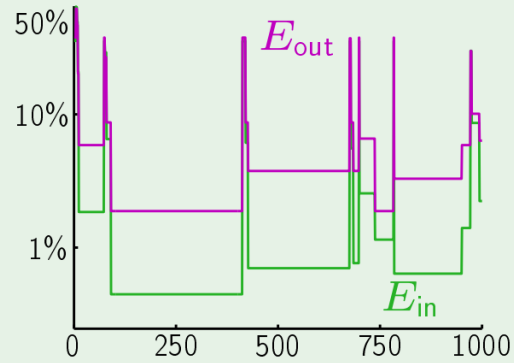
$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^N \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T(t) \mathbf{x}_n}}$$

- 4: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}$
- 5: Iterate to the next step until it is time to stop
- 6: Return the final weights \mathbf{w}

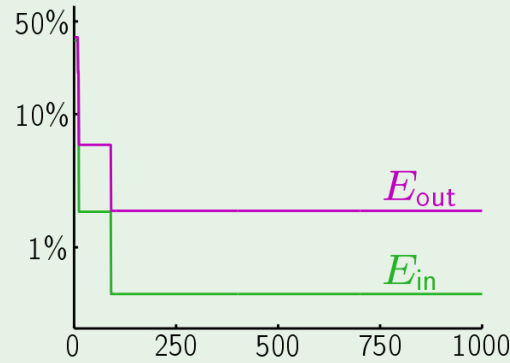
Parar el algoritmo cuando
 $\|\mathbf{w}^{(t-1)} - \mathbf{w}^{(t)}\| < 0,01$,
donde $\mathbf{w}^{(t)}$ denota el vector
de pesos al final de la época t .

BONUS

PLA:

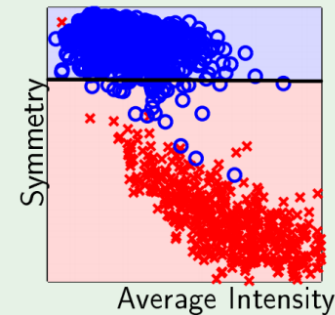


Pocket:

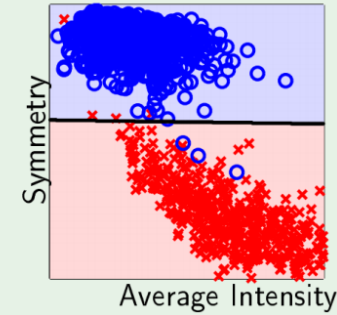


Classification boundary - PLA versus Pocket

PLA:



Pocket:



BONUS

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{8}{N} \log \frac{4((2N)^{d_{vc}} + 1)}{\delta}}$$

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N} \log \frac{2|\mathcal{H}|}{\delta}}$$

Template

- Podéis partir, si queréis, del template que os hemos preparado

– template_trabajo2.py

```
# -*- coding: utf-8 -*-
"""
TRABAJO 2
Nombre Estudiante:
"""
import numpy as np
import matplotlib.pyplot as plt

# Fijamos la semilla
np.random.seed(1)

def simula_unif(N, dim, rango):
    return np.random.uniform(rango[0],rango[1],(N,dim))

def simula_gaus(N, dim, sigma):
    media = 0
    out = np.zeros((N,dim),np.float64)
    for i in range(N):
        # Para cada columna dim se emplea un sigma determinado. Es decir, para
        # la primera columna (eje X) se usará una N(0,sqrt(sigma[0]))
        # y para la segunda (eje Y) N(0,sqrt(sigma[1]))
        out[i,:] = np.random.normal(loc=media, scale=np.sqrt(sigma), size=dim)

    return out

def simula_recta(intervalo):
    points = np.random.uniform(intervalo[0], intervalo[1], size=(2, 2))
    x1 = points[0,0]
    x2 = points[1,0]
    y1 = points[0,1]
    y2 = points[1,1]
    # y = a*x + b
    a = (y2-y1)/(x2-x1) # Calculo de la pendiente.
    b = y1 - a*x1        # Calculo del termino independiente.

    return a, b

# EJERCICIO 1.1: Dibujar una gráfica con la nube de puntos de salida correspondiente

x = simula_unif(50, 2, [-50,50])
#CODIGO DEL ESTUDIANTE

x = simula_gaus(50, 2, np.array([5,7]))
#CODIGO DEL ESTUDIANTE

input("\n--- Pulsar tecla para continuar ---\n")
```