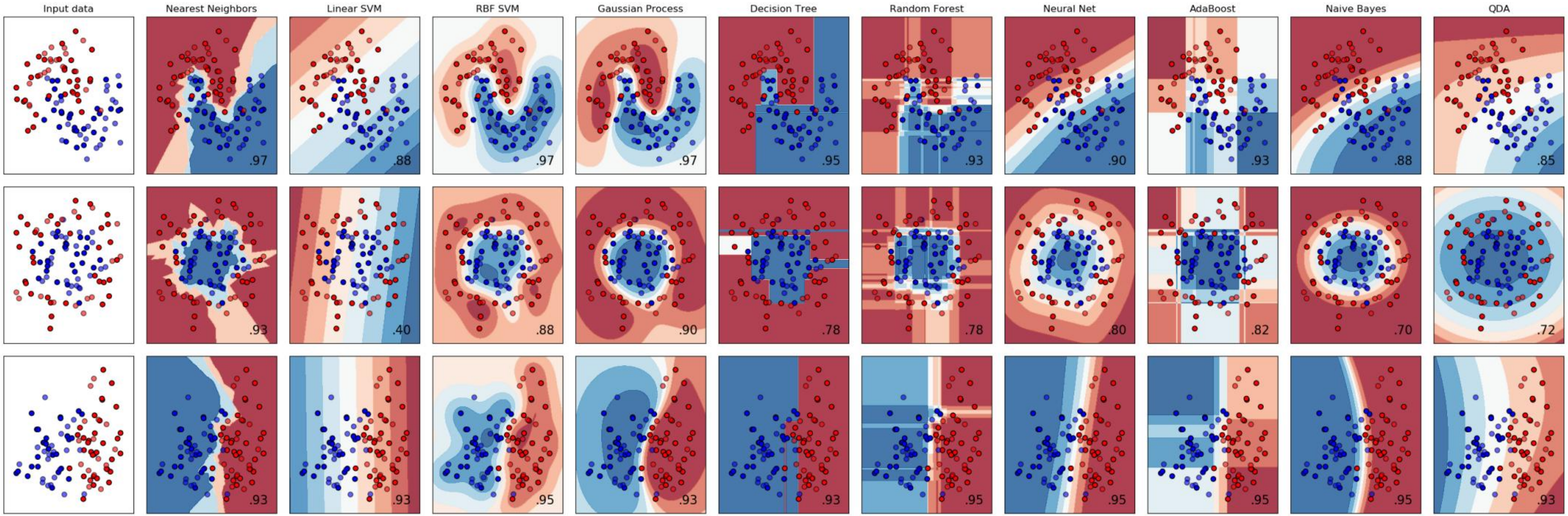
Prácticas de Aprendizaje Automático
Grupo 3

Trabajo 2: Complejidad de H y Modelos Lineales

Pablo Mesejo
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

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

Perceptron Learning Algorithm (PLA):

- Given the data set $(\mathbf{x}_n, y_n), n = 1,2,\cdots,N$
- Step.1: Fix $\mathbf{w}_{\text{ini}}=0$
- Step.2: Iterate on the $\mathcal{D}$-samples improving the solution:
- repeat

  For each $x_i \in \mathcal{D}$ **do**

  if: $sign(\mathbf{w}^T x_i) \neq y_i$ **then**

  update **w**: $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \mathrm{y}_i\mathbf{x}_i$

  else continue

  End for
- Until No changes in a full pass on $\mathcal{D}$

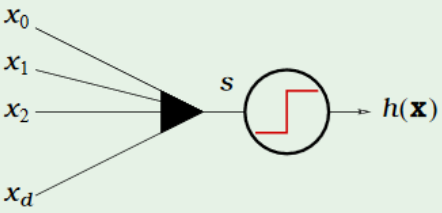
NOTA: Es un algoritmo que suele requerir muchas iteraciones para converger (en problemas linealmente separables)

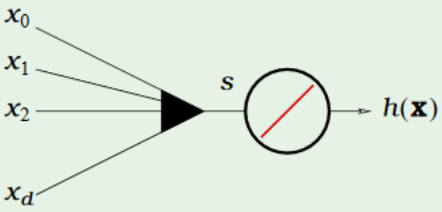
Logistic Regression

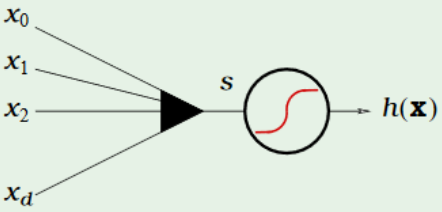
A third linear model

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

linear classification	linear regression	logistic regression
$h(\mathbf{x}) = \text{sign}(s)$	$h(\mathbf{x}) = s$	$h(\mathbf{x}) = \theta(s)$







Logistic Regression

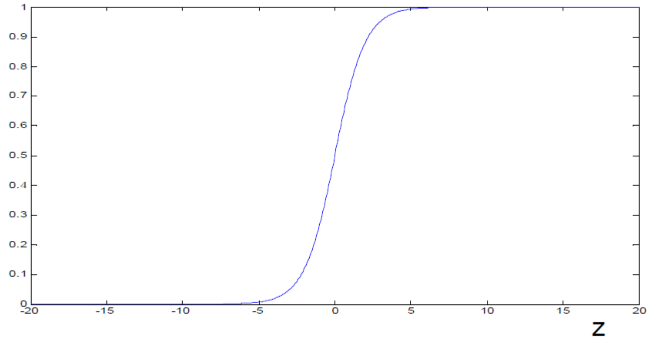
The LR classifier is defined as

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

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

The logistic function or sigmoid function

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



Logistic Regression

Logistic regression algorithm

$\boxed{p(Y=1|x)+p(Y=-1|x)=1}$

RECOMENDACIÓN: N=1

Initialize the weights at $t=0$ to $\mathbf{w}(0)$

for $t=0,1,2,\ldots$ **do**

Compute the gradient

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

Update the weights: $\mathbf{w}(t+1)=\mathbf{w}(t)-\eta\nabla E_{\text{in}}$

Iterate to the next step until it is time to stop

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

$$\mathrm{E}_{out}(h) \leq \mathrm{E}_{in}(h) + \sqrt{\frac{1}{2N}\log\frac{2|\mathcal{H}|}{\delta}} \;\; with\;\; probability\; al\; least\; 1-\delta\; on\; \mathcal{S}$$

- The higher N the narrow the interval (The sample size is important !!)
- The smaller δ the larger the interval (The higher guarantee the lesser accuracy)

Template

- Podéis partir, <u>si queréis</u>, del template que os he preparado
  - template_trabajo2.py

```python
# -*- 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")
```