Prácticas de Aprendizaje Automático Grupo 1

Trabajo 2: Complejidad de H y Modelos Lineales

Francisco Javier Baldán Lozano
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

Recordatorio normas (1). Informe.

.zip = Código (.py) + Informe (.pdf)

- Presentar un informe escrito con las valoraciones y **decisiones** adoptadas en cada apartado.
  - No es solo hacer algo → hay que argumentar el por qué
- Incluir en el informe los **gráficos** generados.
- Incluir una **valoración/discusión de los resultados obtenidos.**

- **El informe debe presentarse en PDF**
- Si no hay informe → se considera que el trabajo no se ha presentado.

Recordatorio normas (2). Código.

- **Un script de Python por ejercicio**.
  - Los distintos ejercicios van en diferentes ficheros .py.

- **Todos los resultados numéricos o gráficas serán mostrados por pantalla**, parando la ejecución después de cada apartado.
  - No escribir nada en el disco.

- El path que se use en la lectura de cualquier fichero auxiliar de datos debe ser siempre "**datos/nombre_fichero**".
  - Crear directorio llamado "datos“ dentro del directorio donde se desarrolla y se ejecuta la práctica.

Recordatorio normas (3). Código.

- El código **debe ejecutarse de principio a fin sin errores**.
- No es válido usar opciones en las entradas.
  - **Fijar al comienzo los parámetros por defecto** que considere óptimos.
- **El código debe estar obligatoriamente comentado** explicando lo que realizan los distintos apartados.
  - **Id comentando el código** que hagáis: sirve para que entendáis mejor lo que habéis hecho, y facilita mi trabajo a la hora de corregir los ejercicios.
- **Entregar solo el código fuente, nunca los datos.**

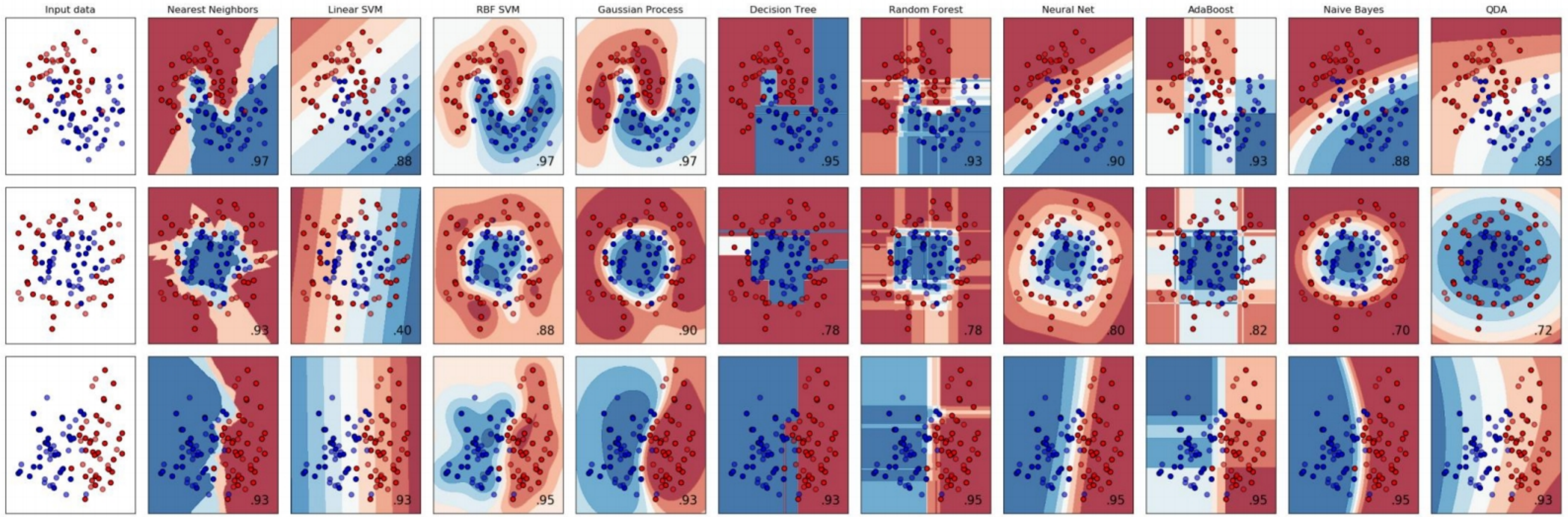
Recordatorio normas (y 4)

.zip = Código (.py) + Informe (.pdf)

Subir el zip en la correspondiente entrega en PRADO.

Fecha de entrega: 26 de Abril

Interesante referencia para visualizar fronteras de decisión.



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

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 $\boldsymbol{x}_i \in \mathcal{D}$ **do**

  if: $sign(\boldsymbol{w}^T \boldsymbol{x}_i) \neq y_i$ **then**

  update **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 $\mathcal{D}$

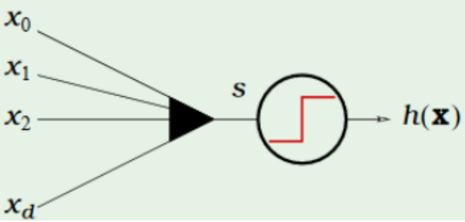
Logistic Regression

A third linear model

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

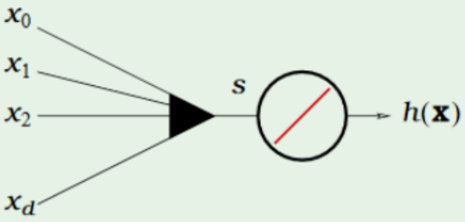
linear classification

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



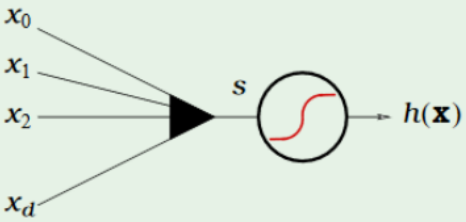
linear regression

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



logistic regression

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



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 $\mathbf{w}$

BONUS

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N}\log\frac{2}{\delta}} \quad \textit{with probability at least } 1-\delta \textit{ on } \mathcal{D}$$

- 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, si queréis, del template

que os he preparado:

– template_trabajo2.zip

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

Bibliografía

Transparencias realizadas y actualizadas a partir del trabajo de Pablo Mesejo.