

# Aprendizaje Automático. Complejidad de H y Modelos Lineales

*Ruido y complejidad, Algoritmo de aprendizaje del Perceptrón,  
Regresión Logística*

**Ricardo Ruiz Fernández de Alba**

Escuela Técnica Ingeniería Informática y Matemáticas  
DECSAI  
Universidad de Granada

17 de abril de 2022

# Índice general

<b>Índice general</b>	<b>ii</b>
<b>1 Sobre la complejidad de H y el ruido</b>	<b>1</b>
1.1 Dibujar gráficas de nubes de puntos simuladas . . . . .	1
1.1.1 Uniformemente distribuidos . . . . .	1
1.1.2 Siguiendo distribución gaussiana de media 0 varianza dada . . . . .	2
1.2 Ejercicio 2 . . . . .	2
1.2.1 Dibujo de puntos con etiqueta y recta usada . . . . .	2
1.2.2 Añadir ruido aleatorio . . . . .	3
1.2.3 Otras fronteras de clasificación . . . . .	4
<b>2 Modelos Lineales</b>	<b>6</b>
2.1 Algoritmo de aprendizaje del Perceptrón (PLA) . . . . .	6
2.1.1 Ejecutar el algoritmo PLA con los datos empleados en el apartado 2a del ejercicio 1. . . . .	6
2.1.2 Repetir usando los datos del apartado 2b del ejercicio 1. . . . .	8
2.2 Regresión Logística (RL). . . . .	9
<b>Bibliografía</b>	<b>11</b>

# Sobre la complejidad de H y el ruido

En este capítulo, trataremos la dificultad que introduce la aparición de ruido en las etiquetas a la hora de elegir la clase de funciones más adecuadas.

## 1.1 | Dibujar gráficas de nubes de puntos simuladas

### 1.1.1 | Uniformemente distribuidos

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

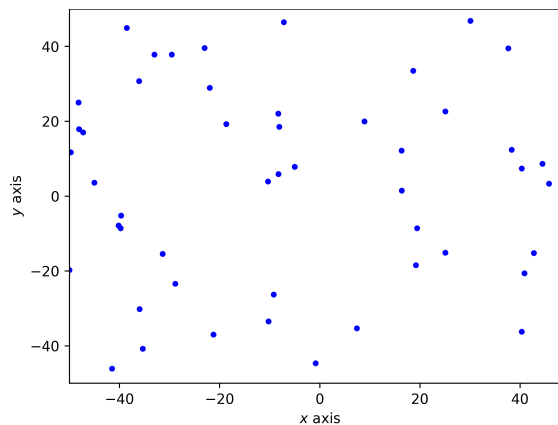


Figura 1.1: Gráfica de nube de puntos uniformemente distribuidos

### 1.1.2 | Siguiendo distribución gaussiana de media 0 varianza dada

Considere  $N = 50$ ,  $\dim = 2$  y  $\sigma = [5, 7]$  con `simula_gauss(N, dim, sigma)`

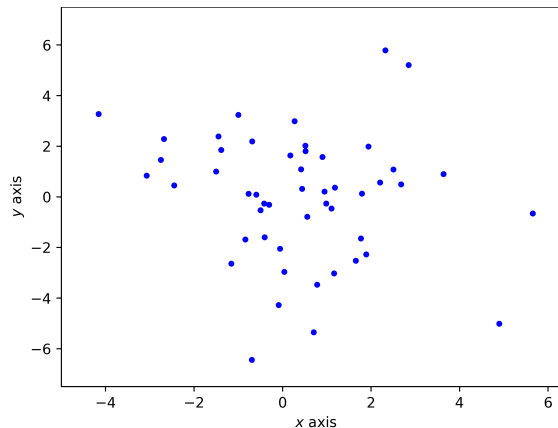


Figura 1.2: Gráfica de nube de puntos en distribución gaussiana.

Al seguir una distribución normal de media cero y varianza  $\sigma$ , los puntos se acumulan en  $[-\sqrt{5}, \sqrt{5}] \times [-\sqrt{7}, \sqrt{7}] \approx [-2.2, 2.2] \times [-2.6, 2.6]$

## 1.2 | Ejercicio 2

Vamos a valorar la influencia del ruido en la selección de la complejidad de la clase de funciones. Con ayuda de la función `simula_unif(100, 2, [-50, 50])` generamos una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función  $f(x, y) = y - ax - b$ , es decir el signo de cada punto con respecto a la recta simulada con `simula_recta()`.

### 1.2.1 | Dibujo de puntos con etiqueta y recta usada

Dibujamos un gráfico 2D con los puntos clasificados por etiquetas junto con la recta usada para etiquetar.

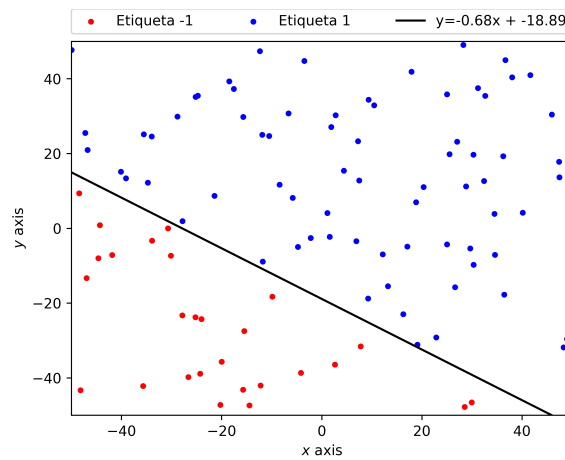


Figura 1.3: Etiquetado de puntos uniformemente distribuidos según recta.

Es claro que si usamos una recta para etiquetar los puntos en dos clases, estos datos están bien clasificados por esta recta.

### 1.2.2 | Añadir ruido aleatorio

Modifique de forma aleatoria un 10 % de las etiquetas positivas y otro 10 % de las negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior. Ahora habrá puntos mal clasificados respecto de la recta.

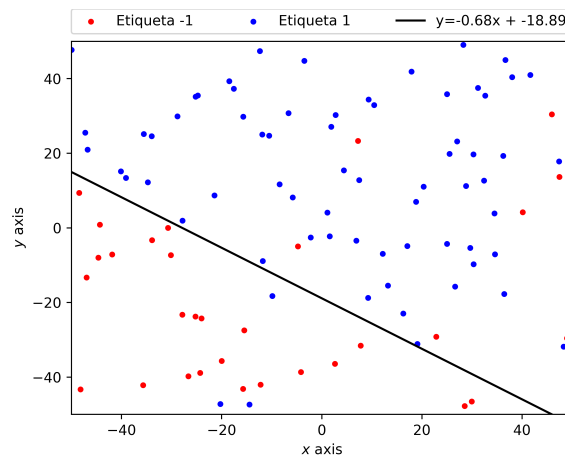


Figura 1.4: Nube de puntos anterior con 10 % de ruido en cada etiqueta.

En efecto, 3 puntos con etiqueta  $-1$  (rojos) ahora tienen etiqueta 1 (son azules). Esto es el 10 % del total (27) redondeado.

### 1.2.3 | Otras fronteras de clasificación

Supongamos ahora que las siguientes funciones ( $f_1, f_2, f_3, f_4$ ) definen la frontera de clasificación de los puntos de la muestra en lugar de una recta.

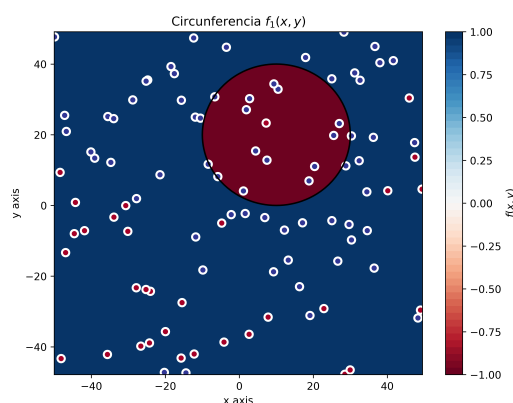
**Visualizar el etiquetado generado en el apartado 2b junto con la gráfica de cada una de las funciones. Comparar las regiones positivas y negativas de estas nuevas funciones con las obtenidas en el caso de la recta. Argumente si estas funciones más complejas son mejores clasificadores que la función lineal. Observe las gráficas y diga qué consecuencias extrae sobre la influencia de la modificación de etiquetas en el proceso de aprendizaje. Explique el razonamiento.**

Usando la función `plot_datos_cuad` proporcionada en la plantilla de código, podemos visualizar y comparar las regiones positivas y negativas (azul y rojo respectivamente) que define la frontera dada por  $f_i(x, y) = 0$  con  $i = 1, 2, 3, 4$ .

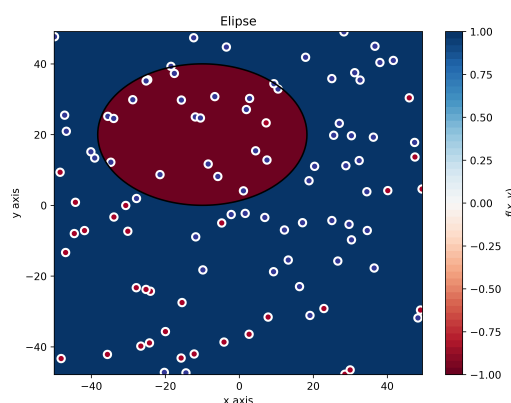
#### 1.2.3.1 | Circunferencia y Elipse

$$\blacksquare f_1(x, y) = (x - 10)^2 + (y - 20)^2 - 400$$

$$\blacksquare f_2(x, y) = \frac{(x+10)^2}{2} + (y - 20)^2 - 400$$



(a) Muestra clasificada por circunferencia  $f_1$

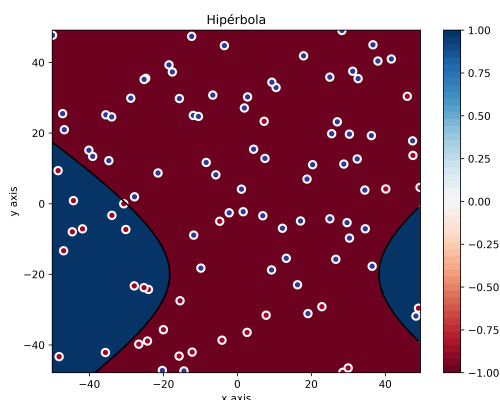
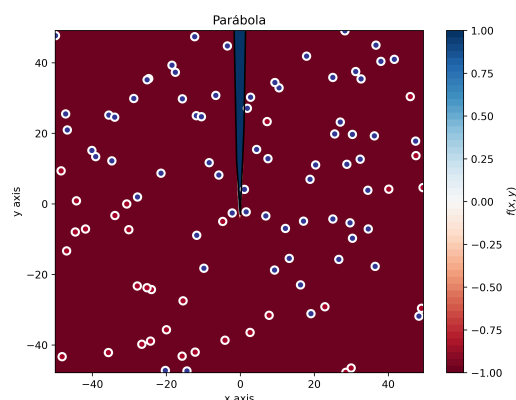


(b) Muestra clasificada por elipse  $f_2$

#### 1.2.3.2 | Hipérbola y Parábola

$$\blacksquare f_3(x, y) = \frac{(x-10)^2}{2} - (y+20)^2 - 400$$

$$\blacksquare f_4(x, y) = y - 20x^2 - 5x + 3$$

(a) Muestra clasificada por hipérbola  $f_3$ (b) Muestra clasificada por parábola  $f_4$ 

Si calculamos el error de clasificación en cada uno de los casos obtenemos:

$f_i$	Error de clasificación (%)
Recta $f$	10 %
Circunferencia $f_1$	44 %
Elipse $f_2$	52 %
Hipérbola $f_3$	81 %
Parábola $f_4$	68 %

Podemos confirmar lo que se podía intuir por las gráficas obtenidas. A pesar de ser funciones más complejas que la lineal, no son mejores clasificadores.

En cuanto a la influencia del ruido, no es posible obtener un mejor clasificador que la recta  $f$ . Esto se debe a que el ruido ha sido generado aleatoriamente y que es la propia recta  $f$  la que se ha usado para etiquetar. Así, el 10 % es cota inferior del de error de clasificación para distinto  $\mathcal{H}$ .

De hecho, para funciones demasiado complejas es probable que tengamos un problema de **sobreajuste** (overfitting) impidiendo una buena generalización (alto valor de  $E_{out}$ ).

## Modelos Lineales

### 2.1 | Algoritmo de aprendizaje del Perceptrón (PLA)

Implementar la función `ajusta_PLA(datos, label, max_iter, vini)` que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada `datos` es una matriz donde cada ítem con su etiqueta está representado por una fila de la matriz, `label` el vector de etiquetas (cada etiqueta es un valor  $+1$  o  $-1$ ), `max_iter` es el número máximo de iteraciones permitidas y `vini` el valor inicial del vector. La función devuelve los coeficientes del hiperplano.

#### 2.1.1 | Ejecutar el algoritmo PLA con los datos empleados en el apartado 2a del ejercicio 1.

Inicializar el algoritmo con:

- el vector cero y,
- con vectores de números aleatorios en  $[0, 1]$  (10 veces).

Anotar el número medio de iteraciones necesarias en ambos para converger.

Se deben mostrar en una tabla cada uno de los pesos iniciales empleados, los finales (obtenidos tras el proceso de entrenamiento), y el porcentaje de error de clasificación.

**Valorar el resultado relacionando el punto de inicio con el número de iteraciones.**



El algoritmo de aprendizaje del perceptrón (PLA) implementado en `ajusta_PLA` consiste en actualizar los pesos por cada punto mal clasificado de la muestra de acuerdo a la siguiente regla:

$$w(t+1) = w(t) + x_i \cdot y_i$$

Esto se hace hasta que el vector de pesos no cambie en toda una época (recorrido de la muestra) o se alcance el número máximo de iteraciones `max_iter`. La implementación realizada devuelve el histórico de pesos, el número de iteraciones dadas y el error de clasificación.

Para visualizar el proceso de convergencia de PLA, se ha implementado una clase `Animacion`. En la ejecución del código se muestra esta animación para el vector inicial nulo en los datos sin ruido, donde tras 75 iteraciones se encuentra un clasificador con 0% de error.

Vector inicial	Iteraciones	Coeficientes w	Error de clasificación
[0.000, 0.000, 0.000]	75	[661.00, 23.20, 32.39]	0 %
[0.574, 0.349, 0.057]	257	[1115.57, 43.48, 62.12]	0 %
[0.229, 0.664, 0.497]	43	[464.23, 15.39, 23.75]	0 %
[0.519, 0.175, 0.571]	231	[1078.52, 39.47, 53.76]	0 %
[0.997, 0.817, 0.594]	71	[664.00, 23.15, 31.90]	0 %
[0.976, 0.902, 0.596]	76	[661.98, 24.90, 36.20]	0 %
[0.032, 0.094, 0.065]	59	[558.03, 19.36, 29.71]	0 %
[0.452, 0.375, 0.975]	274	[1145.45, 40.28, 60.81]	0 %
[0.168, 0.973, 0.767]	235	[1089.17, 39.45, 53.53]	0 %
[0.824, 0.633, 0.669]	257	[1148.82, 39.90, 60.95]	0 %
[0.477, 0.013, 0.353]	74	[673.48, 22.59, 31.35]	0 %
Promedio	157.7		0 %

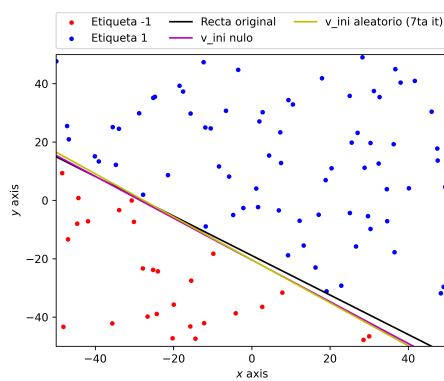
Cuadro 2.1: Resultados de ejecución de PLA **muestra sin ruido** para vector inicial nulo y generados aleatoriamente (10 iteraciones)

Podemos observar en los resultados, que al ser los datos **linealmente separables**, PLA converge en todos los casos a una recta con error de clasificación cero en una media de  $\approx 158$  iteraciones (vectores iniciales aleatorios).

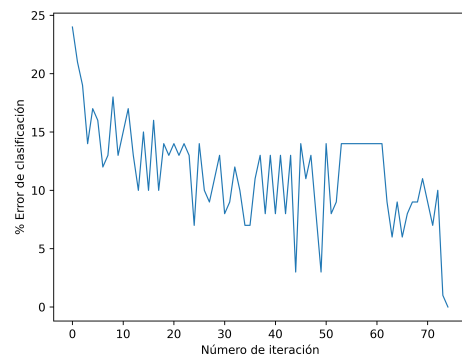
Si hallamos la desviación típica del número de iteraciones, obtenemos 94.17, un valor considerablemente alto que indica la alta sensibilidad de la convergencia (nº de iteraciones en llegar al 0% de error) con respecto al vector inicial. Podemos afirmar por tanto, que escoger el vector nulo como vector inicial resulta ser una buena heurística.

En las siguiente figura, vemos los clasificadores obtenidos para el vector nulo, el vector generado aleatoriamente en la iteración 7 y el clasificador original. Le sigue una gráfica para visualizar la disminución del error de clasificación a lo largo de las iteraciones.

Figura 2.1: Muestra linealmente separable (sin ruido)



(a) Clasificadores para distintos  $v_{ini}$

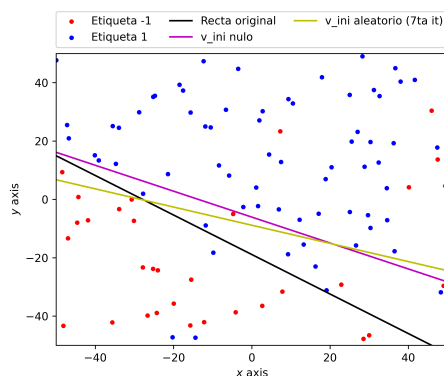


(b) % Error clasif. a lo largo de las épocas

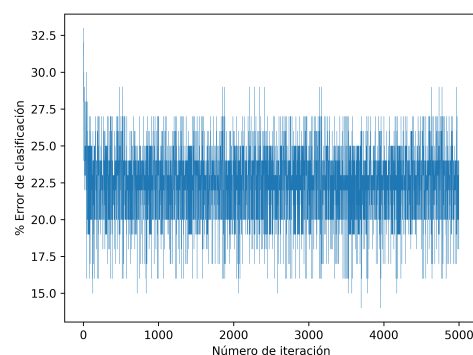
### 2.1.2 | Repetir usando los datos del apartado 2b del ejercicio 1.

¿Observa algún comportamiento diferente? En caso afirmativo diga cuál y las razones para que ello ocurra.

Figura 2.2: Muestra con ruido (no linealmente separable)



(a) Clasificadores para distintos  $v_{ini}$



(b) % Error clasif. a lo largo de las épocas

En este caso, partimos de la muestra de datos con 10 % de ruido. Así, PLA no tiene garantizada una convergencia pues los datos **no son linealmente separables** y la cota inferior del error de clasificación para las rectas obtenidas con PLA son justo ese 10 %.

Por tanto, la ejecución se para al alcanzar el máximo de iteraciones (épocas) que se ha fijado en 5000.

Vector inicial	Iteraciones	Coeficientes w	Error de clasificación
[0.000, 0.000, 0.000]	5000	[339.00, 24.81, 55.86]	24 %
[0.492, 0.730, 0.469]	5000	[348.49, 17.77, 42.43]	24 %
[0.457, 0.138, 0.011]	5000	[349.46, 16.66, 44.13]	24 %
[0.758, 0.320, 0.984]	5000	[339.76, 21.61, 56.29]	22 %
[0.220, 0.339, 0.524]	5000	[336.22, 17.72, 43.85]	25 %
[0.755, 0.464, 0.125]	5000	[336.75, 17.32, 53.11]	27 %
[0.313, 0.505, 0.674]	5000	[339.31, 15.10, 26.87]	20 %
[0.770, 0.130, 0.023]	5000	[333.77, 27.72, 49.69]	20 %
[0.519, 0.810, 0.013]	5000	[343.52, 12.09, 38.81]	22 %
[0.672, 0.687, 0.449]	5000	[331.67, 12.87, 21.73]	19 %
[0.915, 0.644, 0.005]	5000	[341.91, 21.10, 56.13]	22 %
Promedio	5000		22.5 %

Cuadro 2.2: Resultados de ejecución de PLA **muestra con ruido** para vector inicial nulo y generados aleatoriamente (10 iteraciones)

Al contrario que en el caso anterior (muestra sin ruido), ejecutar el algoritmo con vector nulo como vector inicial produce un error de clasificación mayor que la media obtenida para los 10 vectores iniciales generados aleatoriamente (24 % vs 22.5 %). Además, en este caso, por lo comentado anteriormente, el vector inicial no influye en el número de iteraciones.

En cuanto a la variación del error por número de iteraciones, podemos observar grandes oscilaciones, estando concentrados la mayoría en el intervalo  $22.5 \pm 2.5$  %. El mínimo se alcanza entorno a la iteración 4000 donde se alcanza el 14 %, bastante cerca de la cota inferior del 10 % antes mencionada.

## 2.2 | Regresión Logística (RL).

En este ejercicio emplearemos nuestra propia función objetivo  $f$  y un conjunto de datos  $D$  para ver cómo funciona regresión logística. Consideraremos  $d = 2$  para que los datos sean fácilmente visualizables, y emplearemos  $X = [0, 2] \times [0, 2]$  con probabilidad uni-

forme de elegir cada  $x \in X$ . Elegir una línea en el plano que pase por  $X$  como la frontera que separa la región en donde  $y$  toma valores  $+1$  y  $-1$ .

Para ello, seleccionar dos puntos aleatorios de  $X$  y calcular la línea que pasa por ambos.

Impleméntese RL con Gradiente Descendente Estocástico (SGD) del siguiente modo:

- Inicializar el vector de pesos con valores 0.
- Parar el algoritmo cuando  $\|w(t+1) - w(t)\| < 0.01$ , donde  $w(t)$  denota el vector de pesos al final de la época  $t$ . Recuérdese que una época es un pase completo a través de los  $N$  ejemplos de nuestro conjunto de datos.
- Aplicar una permutación aleatoria de  $\{1, 2, \dots, N\}$  a los índices de los datos, antes de usarlos en cada época del algoritmo.

A continuación, empleando la implementación anterior, realícese el siguiente experimento:

- Seleccione  $N = 100$  puntos aleatorios  $\{x_n\}$  de  $X$  y evalúe las respuestas  $\{y_n\}$  de todos ellos respecto de la frontera elegida.
- Ejecute RL para encontrar la función solución  $g$ , y evalúe el error  $E_{out}$  usando para ello una nueva muestra de datos ( $> 999$ ). Se debe escoger experimentalmente tanto el learning rate (tasa de aprendizaje  $\eta$ ) como el tamaño de batch.
- Repita el experimento 100 veces, y calcule los valores promedio de  $E_{out}$ , de porcentaje de error de clasificación, y de épocas necesarias para converger.

---

## Bibliografía