Práctica 2 - Complejidad de $\mathcal{H}$ y modelos lineales

Luis Miguel Guirado Bautista - Universidad de Granada

1 de Mayo de 2022

Índice

1 Complejidad de $\mathcal{H}$ y el ruido

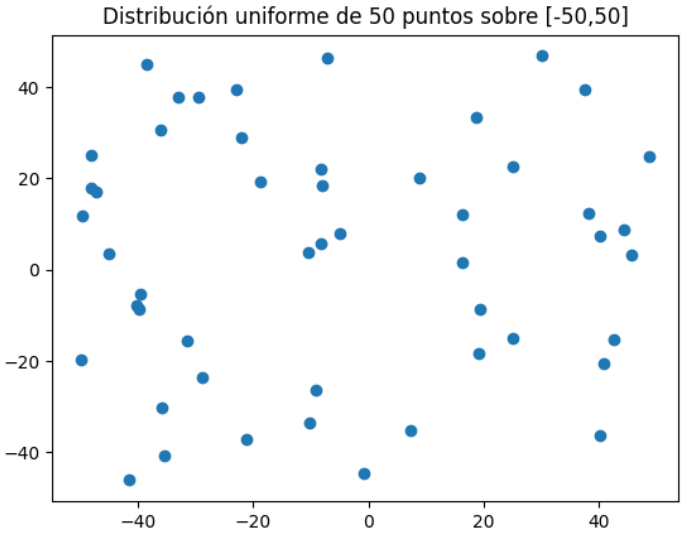
Nota Para la obtención de resultados se ha ajustado la semilla de `NumPy` a 1

Funciones usadas

- `simula_unif(N, dim, rango)`

  Genera `N` vectores aleatorios uniformes de dimensión `dim`. dentro del `rango`

Figura 1: Datos generados por `simula_unif(N=50, dim=2, rango=[-50,50])`

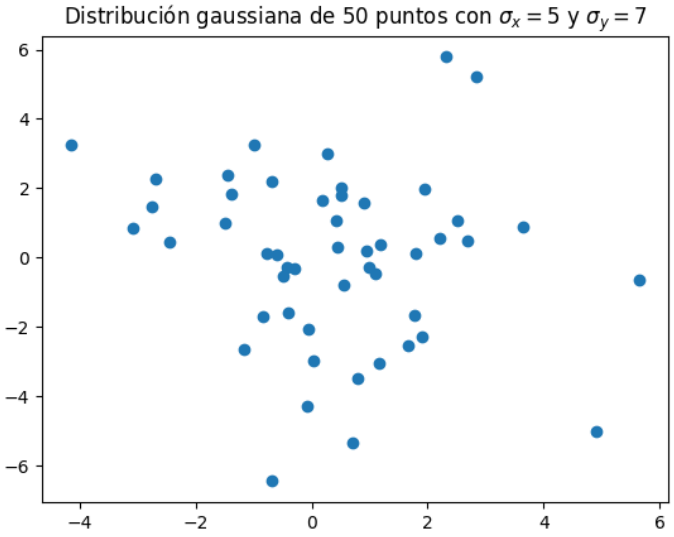


- `simula_gauss(N, dim, sigma)`

  Genera `N` vectores aleatorios de dimensión `dim` dentro de una distribución Gaussiana.

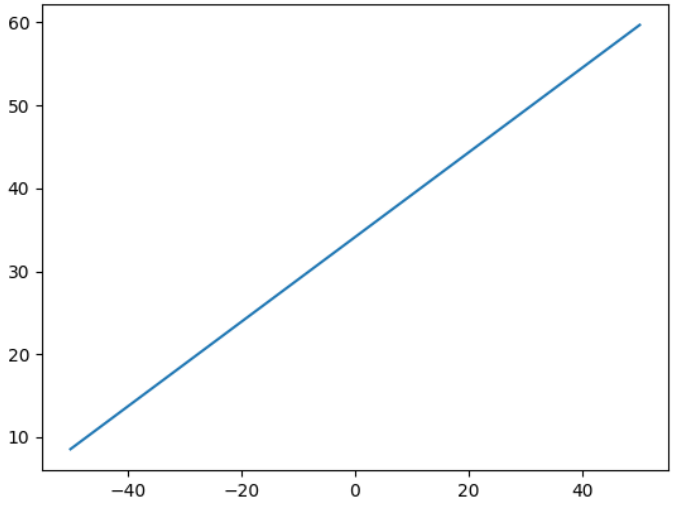
  `sigma` es un vector de dimensión `dim` que posee el valor de la varianza en la distribución Gaussiana para cada una de las dimensiones.

Figura 2: Datos generados por `simula_gauss(N=50, dim=2, sigma=[5,7])`



- `simula_recta(intervalo)`
  Simula una recta del tipo $y = ax + b$ aleatoria con valores dentro de `intervalo`.
  Devuelve los coeficientes a y b.

Figura 3: Recta generada por `simula_recta(intervalo=[-50,50])`



1.1 Influencia del ruido en la selección de la complejidad de $\mathcal{H}$

Vamos a generar una muestra aleatoria uniforme con `simula_unif(100, 2, [-50,50])`, y vamos a clasificar los puntos según la siguiente función:

$$f(x, y) = sign(y - ax - b)$$

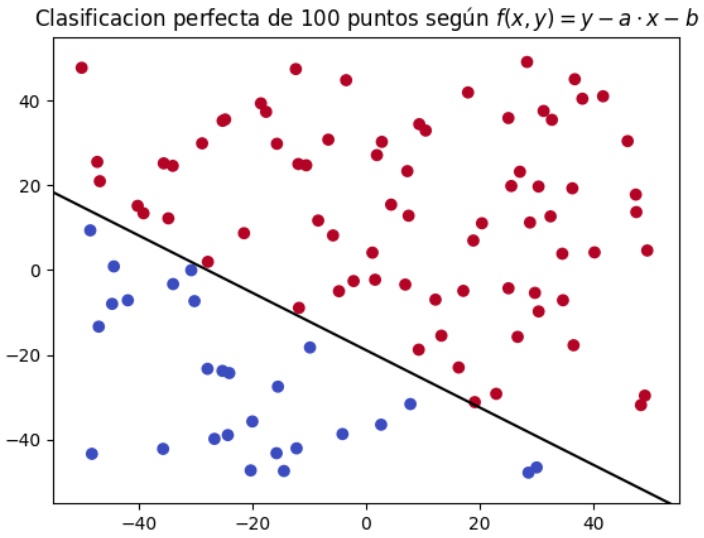
Donde la tupla (x, y) representa las coordenadas del punto de la función y la tupla (a, b) representa los coeficientes de la recta $r = ax + b$, generados por la función `simula_recta`.

Los coeficientes generados son $a = -0.677$ y $b = -18.89$, de modo que f pasa a ser:

$$f(x, y) = sign(y + 0.667x + 18.89)$$

Y la recta que clasifica los puntos es:

$$r = -0.677x - 18.89$$

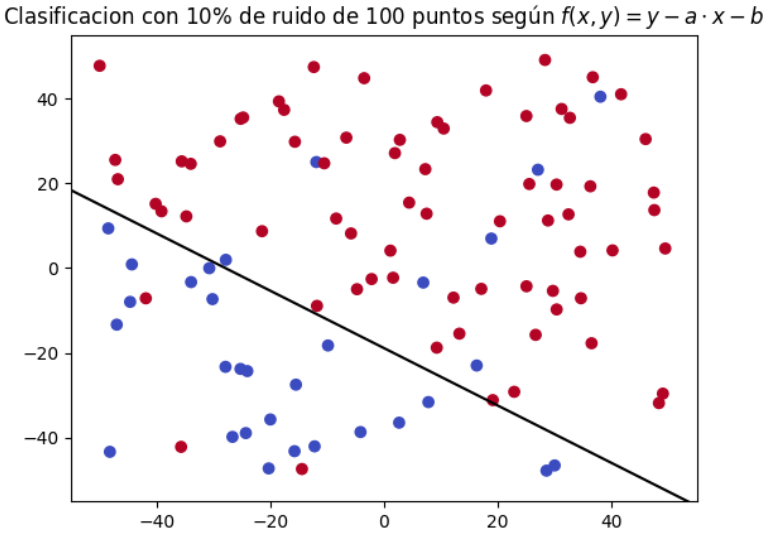


Donde los puntos rojos tienen etiqueta positiva (+1) y los puntos azules tienen etiqueta negativa (-1)

f	Proporción(%)
$+1$	73
-1	27

Como la clasificación es perfecta, $E_{in} = 0$.

Ahora vamos a cambiar la clasificación del 10% de los puntos etiquetados positivamente y otro 10% de los puntos clasificados negativamente.



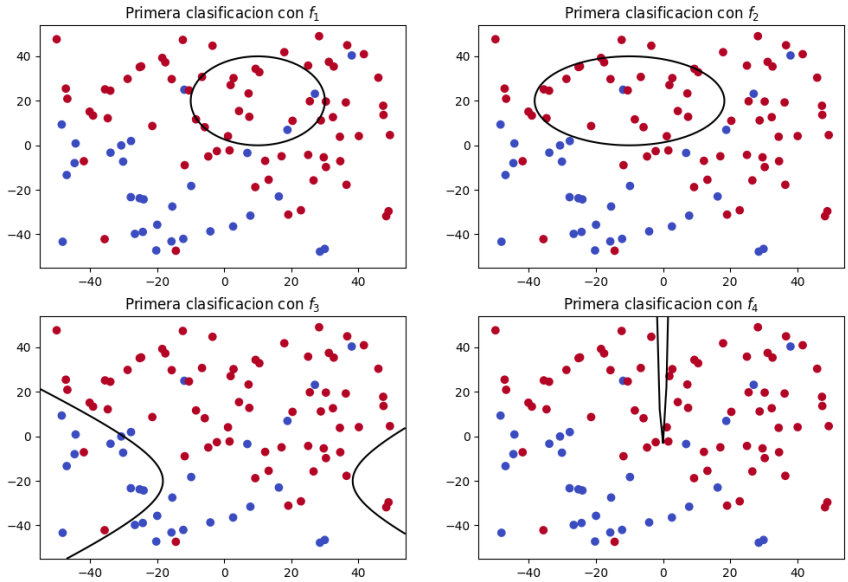
f	Proporción(%)	Puntos de ruido
$+1$	69	7
-1	31	3

Como 10 puntos de 100 están mal clasificados, entonces

$$E_{in} = \frac{10}{100} = 0.1$$

Ahora si decidimos cambiar la frontera de clasificación por estas funciones no lineales

$$f_1(x,y) = (x-10)^2 + (y-20)^2 - 400 \qquad f_2(x,y) = 0.5(x+10)^2 + (y-20)^2 - 400$$
$$f_3(x,y) = 0.5(x+10)^2 - (y+20)^2 - 400 \qquad f_4(x,y) = y - 20x^2 - 5x + 3$$



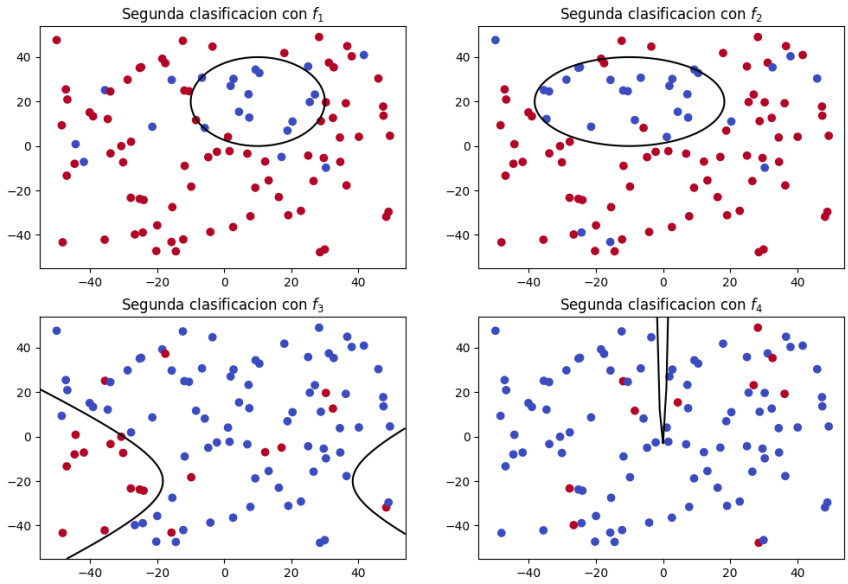
Obtenemos los siguientes errores de clasificación:

f	E_{in}
f_1	0.41
f_2	0.51
f_3	0.76
f_4	0.69

Podemos observar que f_3 es el que peor clasifica los puntos con respecto a f, mientras el que mejor lo hace es f_1.

Aclaramos que las etiquetas son las mismas que en la página anterior, por tanto, la tabla de la página anterior también es aplicable.

Si ahora generamos etiquetas con un 10% de ruido para cada una de las fronteras:



f	Etiqueta	Proporción(%)	Puntos de ruido
f_1	+1	78	9
	-1	22	1
f_2	+1	72	8
	-1	28	2
f_3	+1	21	2
	-1	79	8
f_4	+1	10	0
	-1	90	10

Como hemos aplicado un 10% de ruido a clasificaciones perfectas: $E_{in} = 0.1$

Conclusiones

Podemos observar en la página 7 que los errores obtendidos con los clasificadores no lineales $\{f_1, f_2, f_3, f_4\}$ es mayor que el error del clasificador lineal con ruido en la página 6 (0.1) Por tanto, podemos decir que **no existe correlación directa entre el nivel de complejidad del modelo y la "figura" que forman las etiquetas en el plano**, es decir, fronteras de clasificación más complejas no tienen por que clasificar los resultados mejor, como hemos podido observar en los gráficos anteriores.

También podemos observar que el error de clasificación en f_4 en la página 7 se corresponde con la proporción de etiquetas $+1$. Es decir, todos los puntos con esta etiqueta están mal clasificados porque se encuentran en la región de etiquetas -1. Esto se debe a que la región $+1$ es demasiado pequeña y por tanto, ningún punto se encuentra dentro de ella.

Si generamos ruido respecto a f después de clasificar los puntos según f, es imposible que $E_{in} < 0.1$ o análogamente, **siempre se cumple que** $E_{in} \geq 0.1$ al realizar clasificación con un 10% de ruido. Aunque podría darse el caso de que el ruido se haya generado lo más cerca posible de la frontera de tal forma que se puedan seguir separando todos los puntos sin errores, aunque este caso es muy improbable.

2 Modelos lineales

2.1 Perceptron Learning Algorithm (PLA)

Este algoritmo consiste en encontrar un conjunto de pesos **w** tales que la recta $h \approx f$:

$$h(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 = \mathbf{w^T x}$$

Siendo **x** un conjunto de características de un punto, en este caso $[x_0, x_1, x_2]$
$x_0 = 1$ en todos los puntos, y la tupla (x_1, x_2) son las coordenadas del punto $(x, y) \in \mathcal{D}$.

El algoritmo iterará sobre todos los puntos de $\mathcal{D}$ y recibirá un vector de pesos iniciales $\mathbf{w}_0$ que se irá actualizando siempre que se encuentre un punto mal clasificado $\mathbf{x}_i$ por h con respecto a f, es decir, cuando $h(\mathbf{x}) \neq y_i$.

Cada vez que esta condición ocurra, **w** se verá modificado de la siguiente manera:

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i \cdot y_i$$

El algoritmo se detendrá y devolverá **w** cuando h haya clasificado bien todos los puntos de $\mathcal{D}$ o se haya alcanzado un máximo de iteraciones (usaremos $M = 1000$ iteraciones)

Una iteración es un paso completo por $\mathcal{D}$ (Si fuera sobre cada punto solamente tendríamos que multiplicar esta cantidad por $|\mathcal{D}|$)

Pseudocódigo

function PLA($\mathcal{D}$, $\mathcal{Y}$, M, $\mathbf{w}_0$)
  $\mathbf{w} \leftarrow \mathbf{w}_0$
  $iters \leftarrow 0$
  $cambio \leftarrow$ `True` ▷ Para entrar en el bucle
  while $iters \leq M$ **and** $cambio =$ `True` **do**
    $cambio \leftarrow$ `False`
    $iters \leftarrow iters + 1$
    for $i \in \{1, 2, ..., |\mathcal{D}|\}$ **do**
      if $signo(\mathbf{w^T x}_i) \neq y_i$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i \cdot y_i$
        $cambio \leftarrow$ `True`
      end if
    end for
  end while
  return $(w, iters)$
end function

El error de clasificación dentro de la muestra viene dada por la siguiente expresión:

$$E_{in}(h) = \sum_{i=1}^{N} 1_{signo(h(\mathbf{x})) \neq y_i}$$

Donde $1_{signo(h(\mathbf{x})) \neq y_i} = 1$ si se cumple $signo(h(\mathbf{x})) \neq y_i$

En esta tabla se recogen varias ejecuciones de PLA (se recuerda que $M = 1000$) usando el etiquetado de la página 5 (sin ruido).

$\mathbf{w}_0$	$\mathbf{w}$	Iteraciones	E_{in}
$[0, 0, 0]$	$[661, 23.204, 32.391]$	75	0
$[0.006, 0.613, 0.031]$	$[1124.006, 38.938, 59.415]$	260	0
$[0.063, 0.258, 0.592]$	$[1163.063, 40.012, 61.521]$	286	0
$[0.088, 0.861, 0.072]$	$[558.088, 20.15, 30.224]$	62	0
$[0.329, 0.810, 0.933]$	$[815.328, 29.196, 43.789]$	118	0
$[0.247, 0.554, 0.56]$	$[464.247, 15.28, 23.81]$	43	0
$[0.477, 0.45, 0.936]$	$[654.477, 21.406, 29.8]$	70	0
$[0.39, 0.047, 0.256]$	$[1111.39, 43.41, 61.93]$	255	0
$[0.432, 0.294, 0.541]$	$[653.432, 23.09, 32.012]$	75	0
$[0.761, 0.907, 0.772]$	$[654.761, 23.524, 31.708]$	70	0
$[0.565, 0.171, 0.426]$	$[1122.565, 38.641, 59.61]$	267	0

Siendo 150.6 el valor medio de iteraciones necesario para que PLA pueda converger.

Podemos ver que en todas las ejecuciones no hay error alguno porque termina antes de alcanzar el máximo de iteraciones. ¿Qué pasaría si usamos el etiquetado con ruido de la página 6?

$\mathbf{w}_0$	$\mathbf{w}$	Iteraciones	E_{in}
$[0, 0, 0]$	$[384, 24.28, 43.544]$	1000	0.06
$[0.75, 0.033, 0.482]$	$[388.75, 2.623, 33.674]$	1000	0.16
$[0.522, 0.537, 0.163]$	$[384.522, 23.137, 45.453]$	1000	0.07
$[0.638, 0.236, 0.157]$	$[383.638, 30, 44.728]$	1000	0.1
$[0.389, 0.5, 0.166]$	$[393.389, 17.474, 55.412]$	1000	0.1
$[0.557, 0.81, 0.136]$	$[386.557, 22.697, 49.78]$	1000	0.08
$[0.775, 0.982, 0.942]$	$[386.775, 23.325, 31.18]$	1000	0.07
$[0.339, 0.693, 0.662]$	$[385.34, 23.81, 34.587]$	1000	0.1
$[0.828, 0.846, 0.555]$	$[389.828, 11.89, 31.602]$	1000	0.08
$[0.673, 0.498, 0.008]$	$[391.673, 22.04, 39.723]$	1000	0.06
$[0.712, 0.508, 0.19]$	$[401.712, 23.676, 47.911]$	1000	0.07

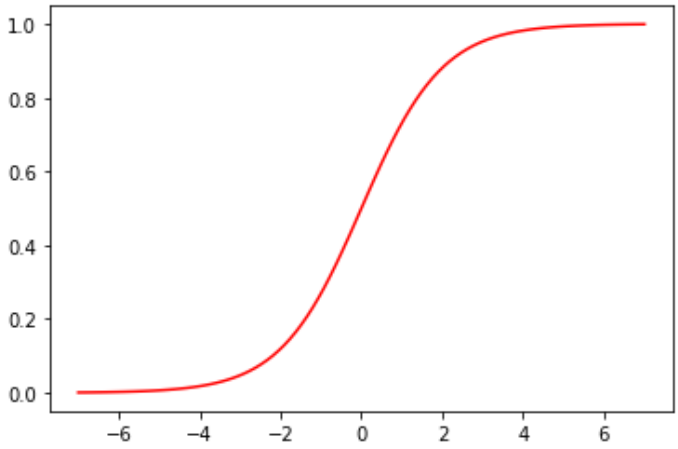
Podemos observar que el algoritmo no llegará a converger hasta el máximo de iteraciones debido al ruido que le hemos añadido al etiquetado y que E_{in} se ha vuelto existente debido a eso. No obstante, podemos observar que en la mayoría de los casos $E_{in} \leq 0.1$, ¡y eso es bueno por que las hipótesis dadas por PLA suelen clasificar incluso mejor que la frontera!

Y en comparación con los valores de **w** en la tabla anterior, estos son experimentalmente más estables, ya que w_0 sin ruido oscila más que con ruido e w_1 y w_2 suelen estar más acotados.

A pesar de los buenos resultados con respecto a las iteraciones y el error, los valores de los pesos no me inspiran confianza al ser tan elevados.

2.2 Regresión Logística con Gradiente Descendente Estocástico (SGDLR)

2.2.1 La función sigmoide (σ)



Este algoritmo utiliza tanto una manera de clasificar como un método de actualización de pesos distintos. Ahora para clasificar cada uno de los puntos, en vez de usar la función h anterior usaremos la función *sigmoide* σ (correspondiente al gráfico de arriba).

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

Y según esta función, el etiquetado para cada punto será:

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

De manera que ahora la clasificación de cada punto ahora depende de una probabilidad (el dominio de σ es $[0,1]$ al igual que $\mathcal{P}$). La función h ahora es una versión binaria de σ.

¿Y el error? Utilizaremos el *cross-enthropy error*

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

Nota $N = |\mathcal{D}|$

2.2.2 Descripción del algoritmo y pseudocódigo

Pseudocódigo

function $\mathrm{SGDLR}(\mathcal{D}, \mathcal{Y}, \eta, T, N)$ ▷ N es el tamaño del *batch*
  $\mathbf{w} \leftarrow 0_d$ ▷ Vector de d ceros. d es la dimensión de $\mathcal{D}$.
  $\mathbf{w}_{old} \leftarrow \infty$ ▷ Pesos de la época anterior.
  $t \leftarrow 0$
  while $t \leq T$ **and** $||\mathbf{w}_{old} - \mathbf{w}|| \geq 0.01$ **do**
    $t \leftarrow t + 1$
    $M \leftarrow elegir([1, |\mathcal{D}|], N)$ ▷ N números en $[1, |\mathcal{D}|]$ sin repeticiones
    $\nabla E_{in} = -\frac{1}{|M|} \sum_n^M \frac{y_n \mathbf{x}_n}{1+e^{y_n \mathbf{w}^T \mathbf{x}_n}}$
    $\mathbf{w}_{old} \leftarrow \mathbf{w}$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_{in}$
  end while
  return (w, t)
end function

Ahora el algoritmo irá iterando por épocas. Una época t es una hipótesis h identificada por sus pesos, es decir, estamos hablando de la clasificación de los puntos adoptada por h con los pesos de esa época. Comenzaremos a partir del vector de pesos cero, en vez de un vector pasado como parámetro, y entraremos en un bucle en el que **w siempre** recibe una actualización.

Entonces, ¿cuál es la condición de parada? $||\mathbf{w}_{old} - \mathbf{w}|| < 0.01$

Esto significa que si la actualización de los pesos no ha sido lo suficientemente significante, entonces podemos parar. Los pesos se actualizan mediante el gradiente del error dentro de la muestra (∇E_{in}) multiplicado por una tasa de aprendizaje η, un método muy similar al de la práctica anterior. Estamos **minimizando** E_{in}, entonces es fácil caer en que ∇E_{in} es la derivada de E_{in} *cross-enthropy*.

¿Por qué escogemos puntos aleatorios en cada época? Con esto podemos asegurar que los pesos que estamos intentando obtener sean más tolerables al resto de puntos y que no solo se centren en una elección para evitar errores en ciertos casos particulares.

A continuación vamos a realizar ejecuciones de nuestro algoritmo, pero no tenemos una η y N predeterminadas, por tanto tendremos que realizar varias ejecuciones experimentales para decidirlo.

2.2.3 Escogiendo los parámetros

Vamos a ejecutar el algoritmo en un nuevo subconjunto de datos $\mathcal{D}$ donde $\mathcal{X} = [0, 2] \times [0, 2]$

Para todas las ejecuciones ajustaremos $T = 1000$.

Vamos a probar primero con $\eta = 0.1$ (igual que la práctica anterior) y $N = 80$ (igual que el % de training). Se irán registrando los datos en la tabla si se nota un cambio en el porcentaje de aciertos o en el número de épocas t.

η	N	**w**	E_{in}	Acierto(%)	t
0.1	80	$[-0.447, -0.477, 2.574]$	0.525	82	79
0.25	80	$[-1.287, -0.8, 1.989]$	0.425	86	118
0.5	80	$[-1.973, -0.787, 2.574]$	0.395	87	118
0.7	80	$[-2.284, -0.742, 2.78]$	0.387	88	112
1.4	80	$[-2.714, -0.632, 3.067]$	0.38	89	87
1.6	80	$[-3.404, -0.58, 3.5]$	0.376	91	223

Paro el experimento porque se ha alcanzado una tasa de acierto superior al 90% A continuación mostraré una gráfica por ejecución de la tabla, ordenadas como en la tabla, para que se vea el ajuste de η. Además, con N no creo que haya problema ya que sigue la filosofía de la separación de los datos de entrenamiento y test.

En conclusión, usaremos $\eta = 1.6$ y $N = 80$.

¿Cómo imprimimos la frontera de clasificación generada? Hay que realizar un razonamiento matemático.

$$\frac{1}{1 + e^{-f(x)}} = 0.5$$

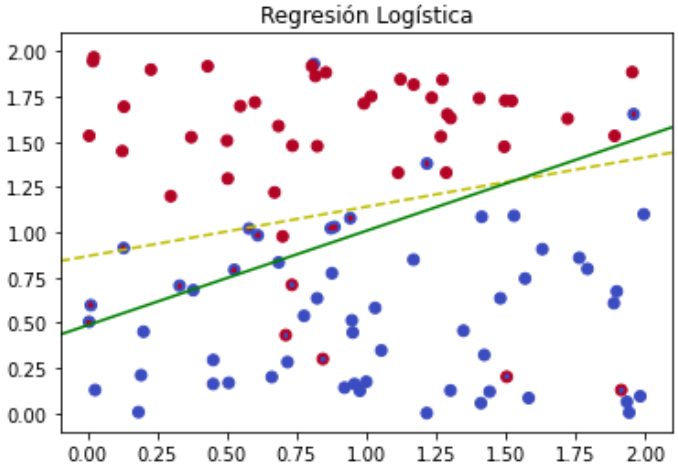
$$\frac{1}{2} + \frac{e^{-f(x)}}{2} = 1$$

$$e^{-f(x)} = 1$$

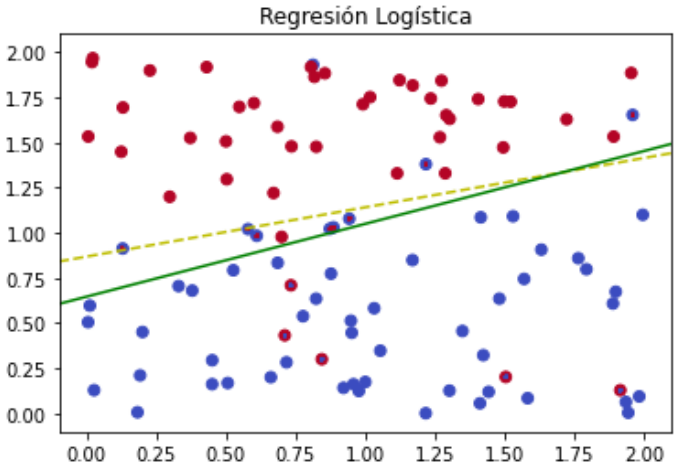
$$f(x) = w_2 x_2 + w_1 x_1 + w_0 = 0$$

$$x_2 = \frac{-w_0 - w_1 x_1}{w_2}$$

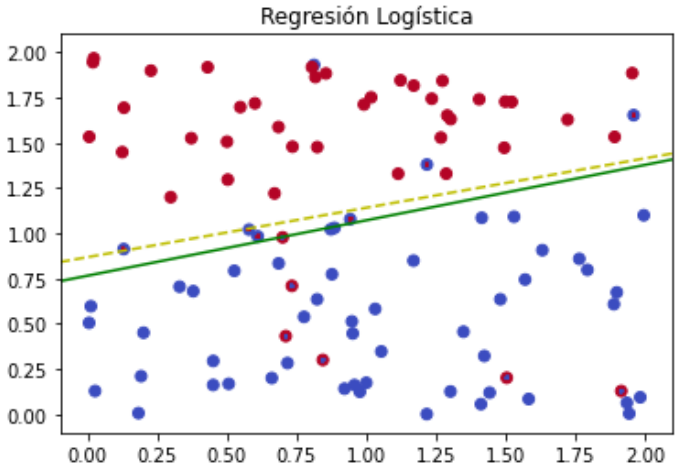
Para saber el porqué, véase la tercera línea de texto de la página 10, donde explica el significado de los componentes de **x**.



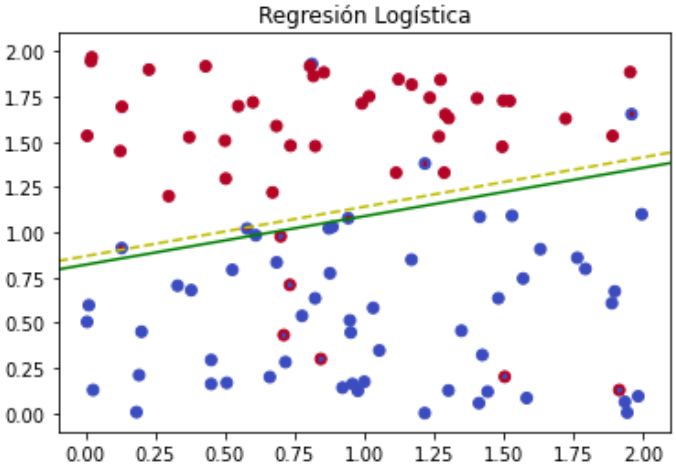
$\eta = 0.1$



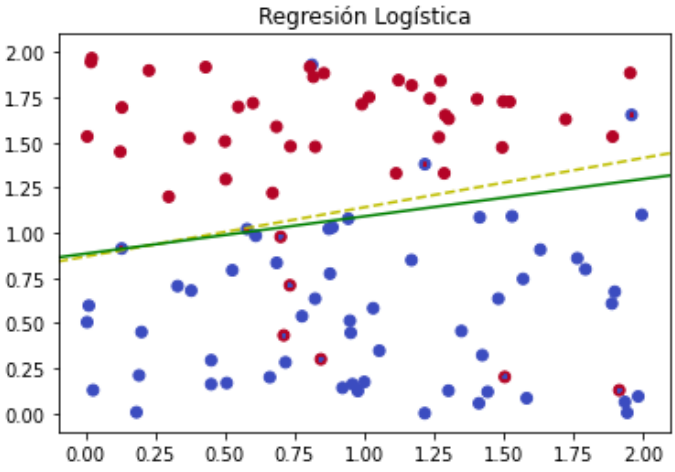
$\eta = 0.25$



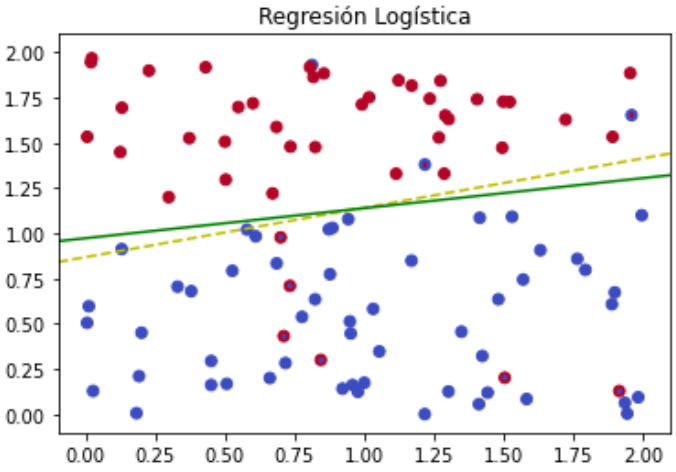
$\eta = 0.5$



$\eta = 0.7$



$\eta = 1.4$



$\eta = 1.6$

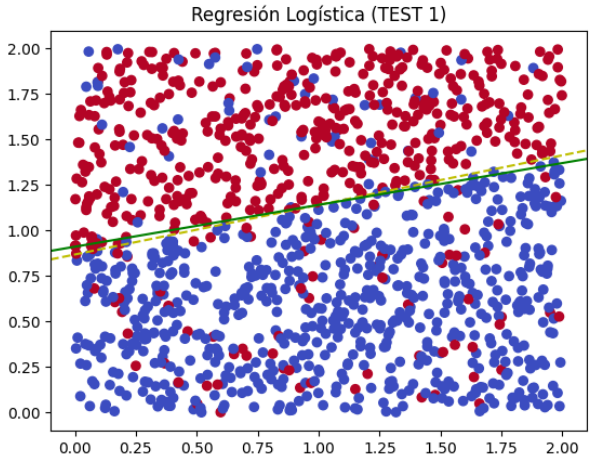
2.2.4 Experimentación con muestras grandes para el test

Emplearemos la última ejecución de nuestro algoritmo durante el experimento anterior para saber si clasifica bien en otros conjuntos de datos más grandes.

Emplearemos $|\mathcal{D}_{test}| = 1200$ y repetiremos este experimento 100 veces.

Si hacemos la media del error, el porcentaje de acierto y de epocas empleadas:

E_{out}	Acierto(%)	t
0.408	0.881	353.58



Los resultados son buenos ya que $E_{in} \approx E_{out}$ y la tasa de acierto es ≈ 0.9.