Aprendizaje Automático (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 1

UNIVERSIDAD
DE GRANADA

Antonio Jesús Heredia Castillo

2 de mayo de 2019

Índice

Índice de figuras

Índice de tablas

1. Complejidad de H y ruido

1.1. Dibujar una gráfica con la nube de puntos de salida correspondiente

1.1.1. Considere $N = 50$, $dim = 2$, $rango = [-50+50]$ con $simula_unif(N, dim, rango)$

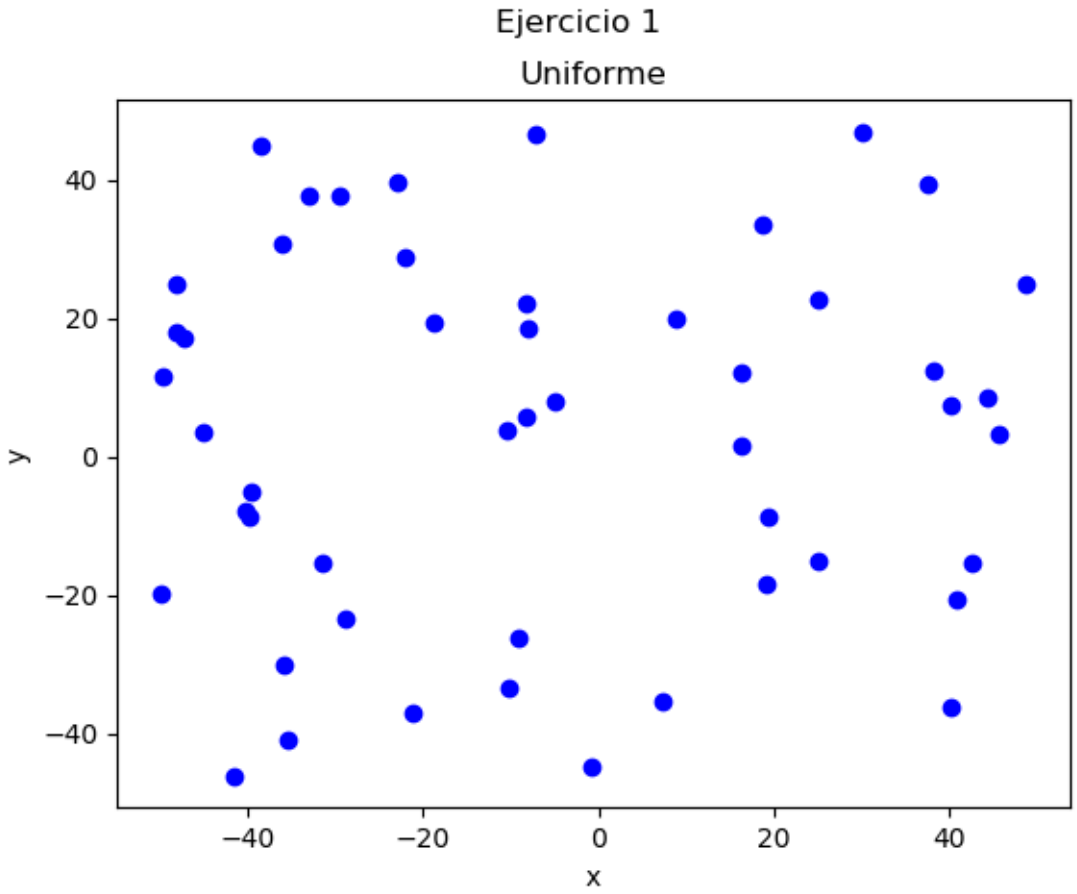

Figura 1: Distribución uniforme de puntos en el intervalo $[-50, +50]$

1.1.2. **Considere** $N = 50$**,** $dim = 2$**,** $rango = [-50{+}50]$ **con** $simula_unif(N, dim, rango)$

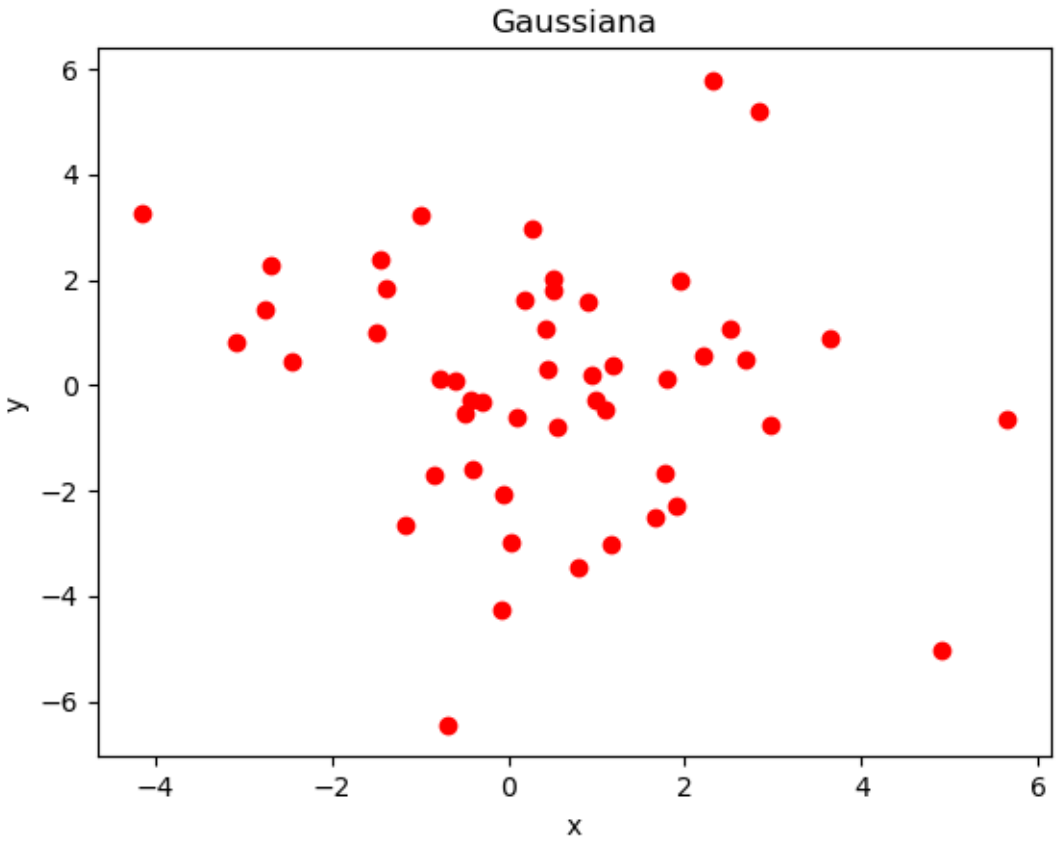

Figura 2: Distribución gaussiana de media 0 con sigma $[5, 7]$

1.2. Con ayuda de la función $simula_unif()$ generar una muestra de puntos 2D a los que vamos añadir una etiqueta usando la función $f(x, y) = y - ax - b$, es decir el signo de la distancia de cada punto a la recta simulada con $simula_recta()$

1.2.1. Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello.(Observe que todos lo puntos están bien clasificados respecto de la recta.)

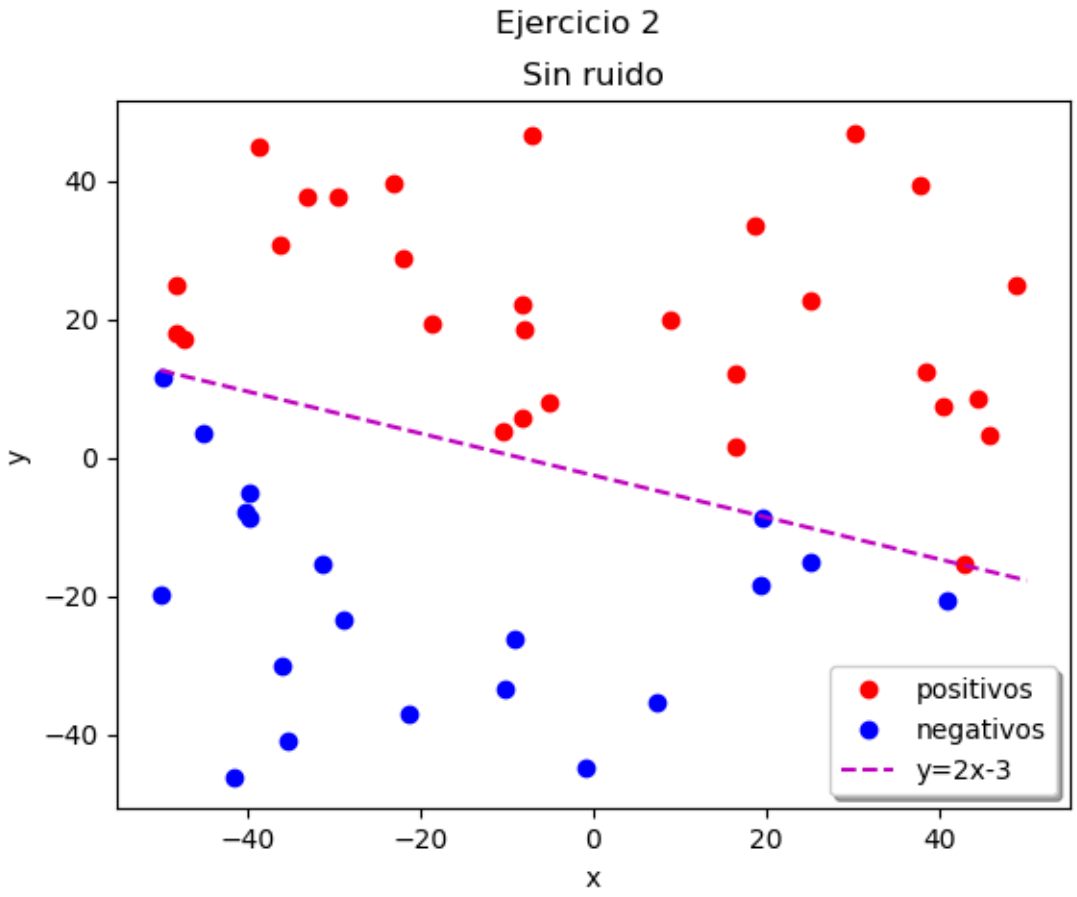

Figura 3: Puntos "sin ruido" clasificados con la función $f(x, y) = y - ax - b$

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

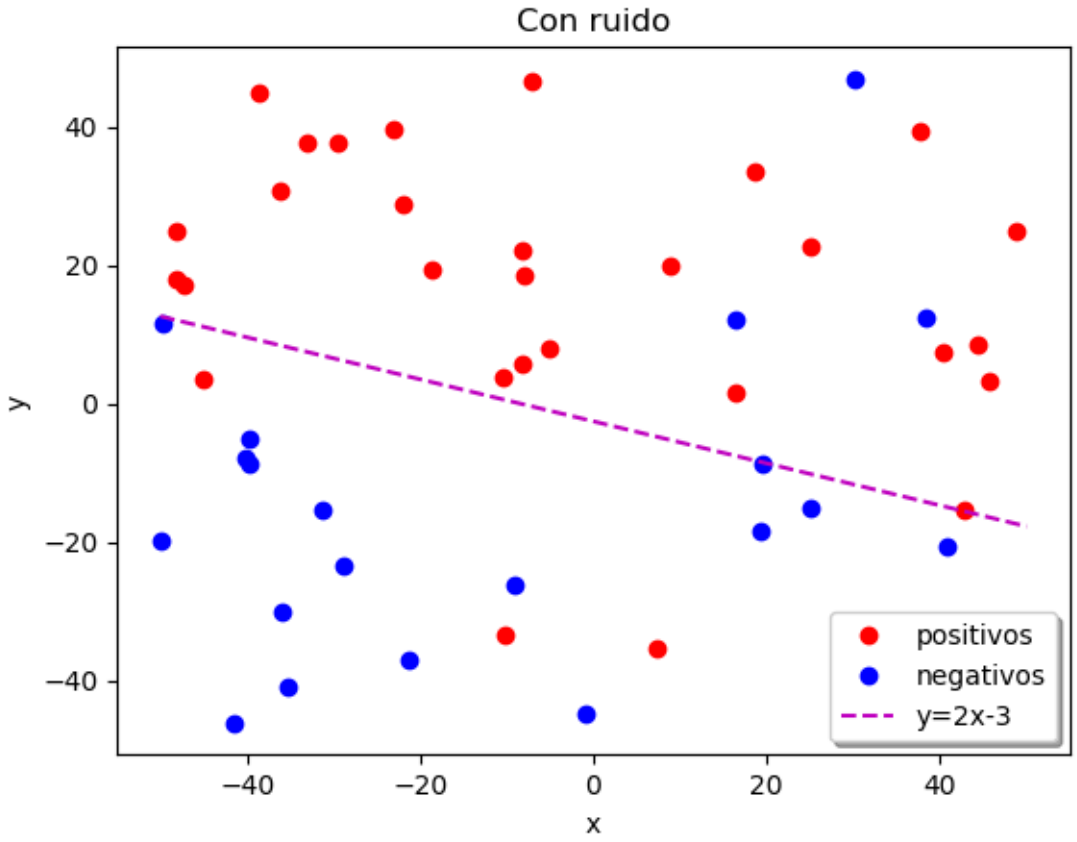

Figura 4: Puntos "con ruido" clasificados con la función $f(x, y) = y - ax - b$

1.3. Supongamos ahora que las siguientes funciones definen la frontera de clasificación de los puntos de la muestra en lugar de una recta

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

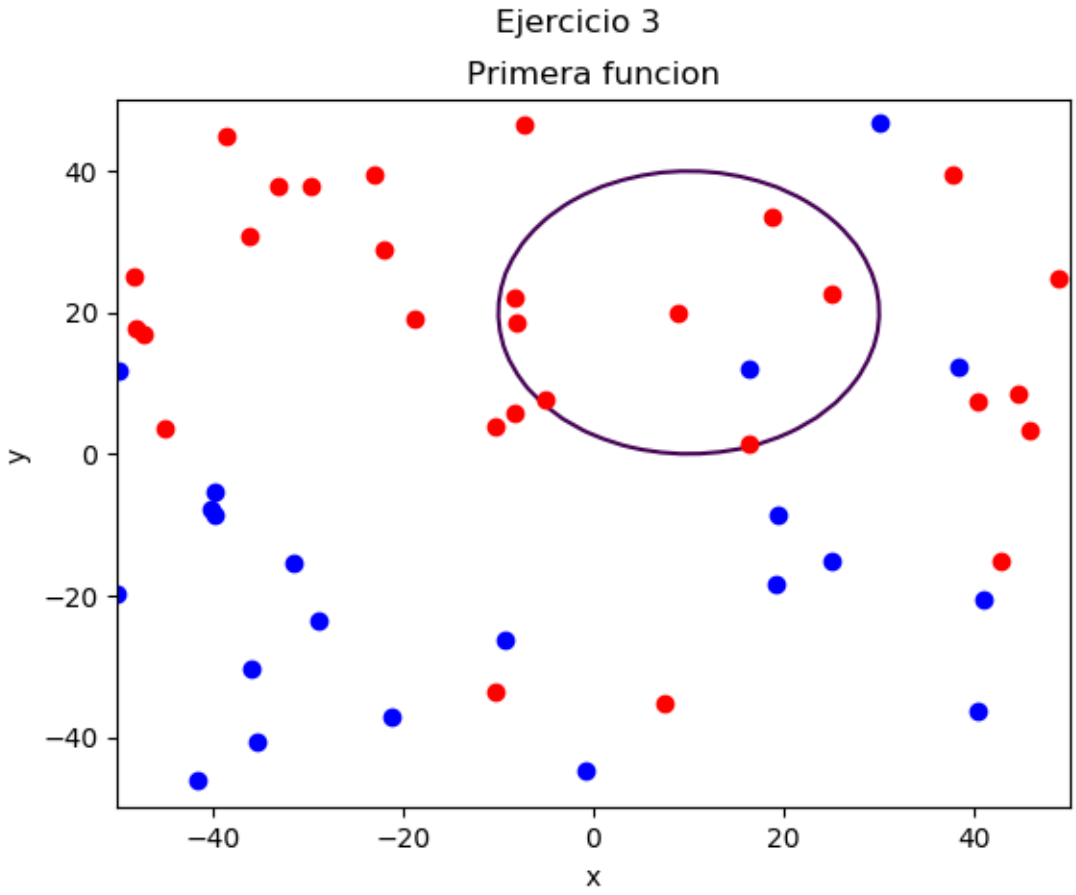

Figura 5: Primera función de frontera de clasificación

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

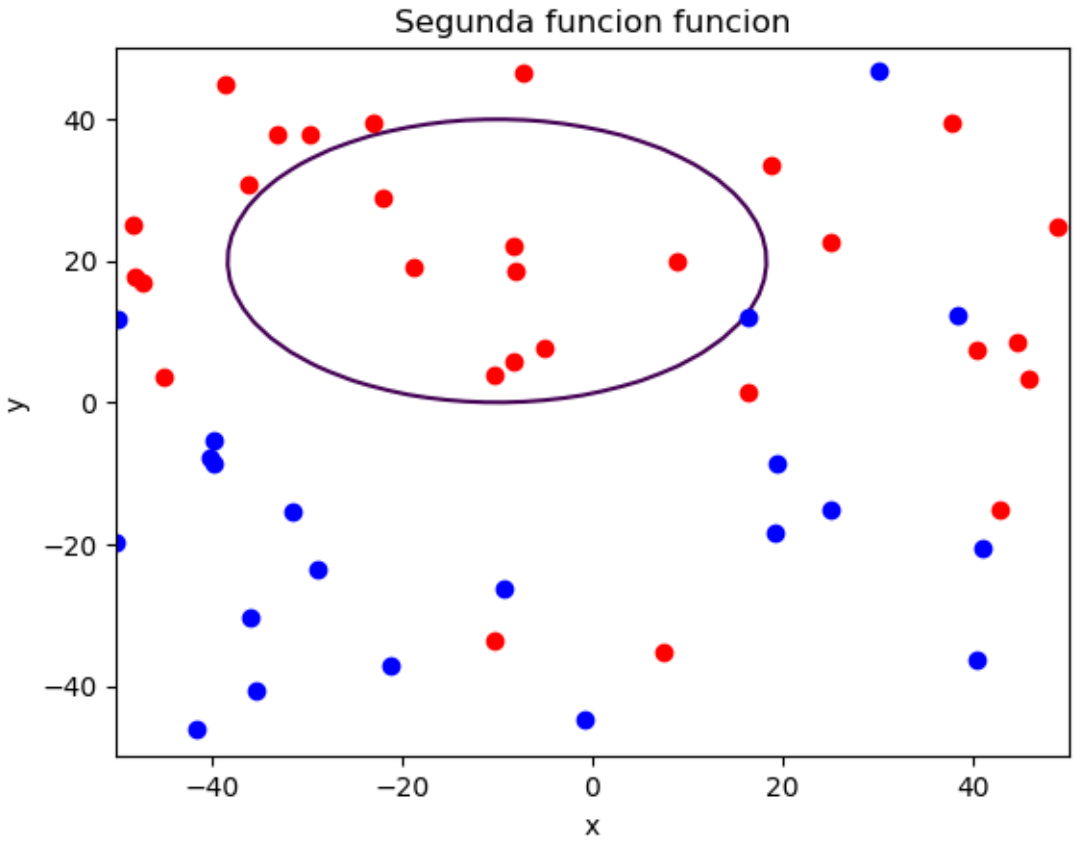

Figura 6: Segunda función de frontera de clasificación

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

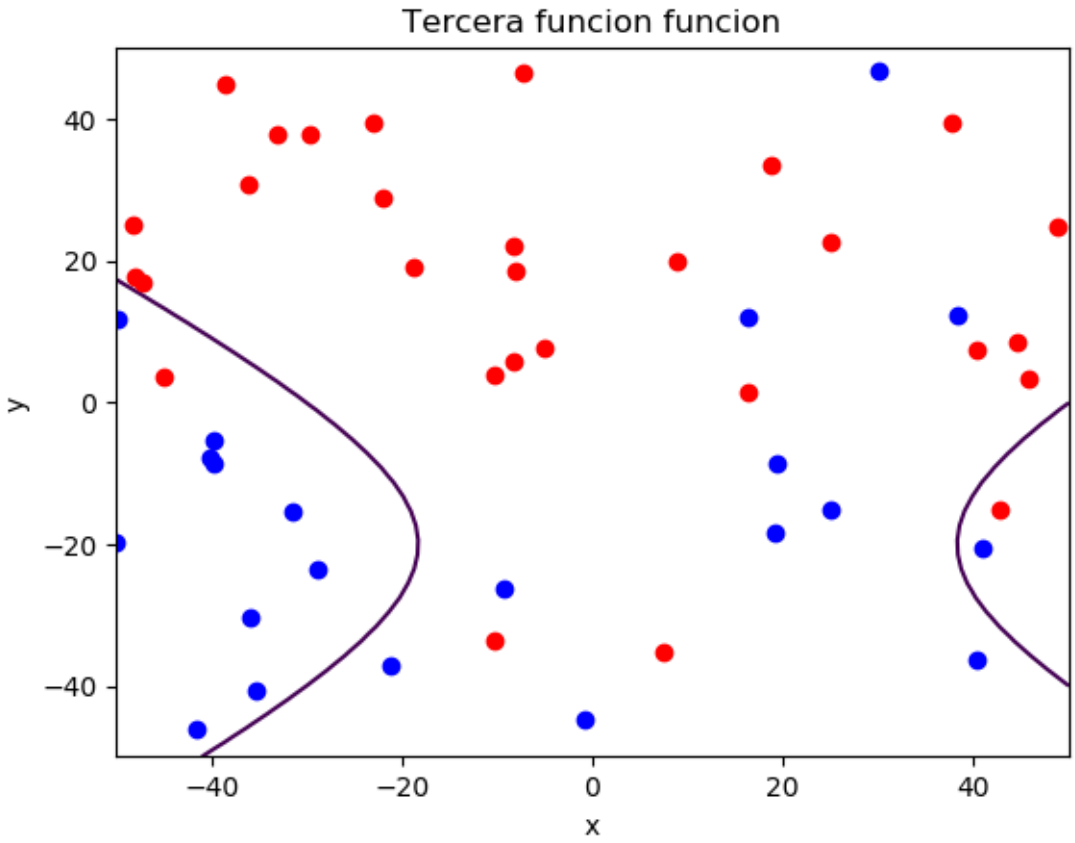

Figura 7: Tercera función de frontera de clasificación

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

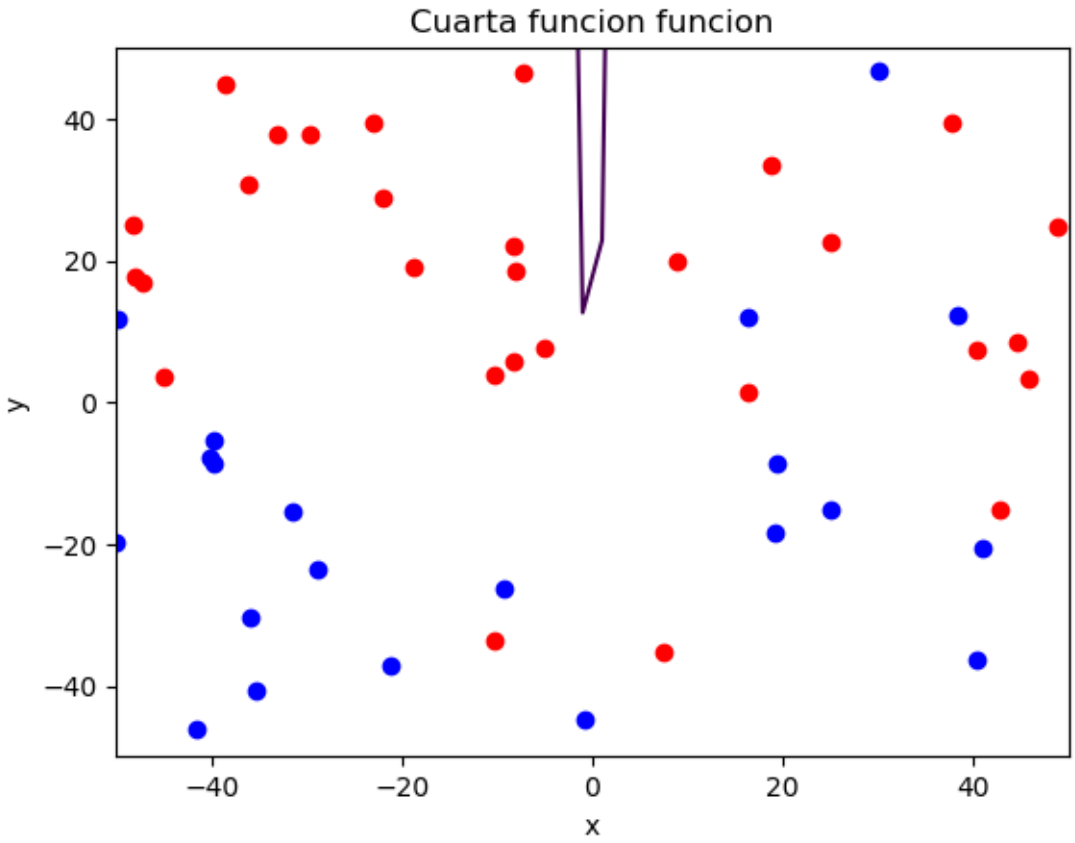

Figura 8: Cuarta función de frontera de clasificación

1.3.5. Análisis de las clasificaciones obtenidos

Empezamos a analizar la primera y segunda función ya que son muy parecidas entre ellas. Podemos ver que al contrario que la recta que divide el plano en dos partes infinitas, las elipses separan para el plano en dos partes, pero una de ellas (la interior) estaría acotada. Ademas en la segunda, todo el interior estaría bien clasificado, pero en cambio "fuera" hay muchos mas fallos.
En la tercera volvemos a tener dos partes de tamaño infinito. Aunque sigue sin clasificar bien todos los puntos. En la cuarta, no clasifica bien ninguno, ya que el interior de la función no tiene ningún punto dentro. Esta tipo de funciones pueden ser buenas para otro tipos de datos(como los del ejercicio

2 de la practica anterior), pero no para los nuestros.

2. Modelos lineales

2.1. Perceptron

Debemos 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 iteracciones permitidas y *vini* el valor inicial del vector. La función devuelve los coeficientes del hiperplano. El codigo de la implementación es el siguiente:

```
def ajusta_PLA(datos, label, max_iter, vini):
_w = vini
_X =  np.insert(datos,0,1,axis=1)
_iteraciones = 0
_cant_cambios = 1
while _iteraciones < max_iter and _cant_cambios != 0:
        _cant_cambios = 0
        _iteraciones+= 1
        for _ix, xf in enumerate(_X):
                if np.sign(_w.T @ xf) != label[_ix]:
                _w = _w+label[_ix]*xf
                _cant_cambios += 1
return _w,_iteraciones
```

2.1.1. Ejecutar PLA con los datos simulados en los apartado 2a de la sección 1

El algoritmo se ejecutara con diferentes inicializaciones.

1. El vector a cero
2. Con vectores de números aleatorios en $[0, 1]$ (10 veces).Anotando el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado relacionando el punto de inicio con el numero de iteracciones.

Cuando tenemos el vector inicializado a ceros, obtenemos **15 iteracciones** y cuando lo inicializo a números aleatorios obtengo una media de **9.2 iteracciones**. Por tanto tenemos que cuando el vector esta inicializado aleatoriamente necesita menos pasos, esto se puede deber a que de esta forma podemos encontrar pesos que están mas cercanos a la solución final.

2.1.2. Ejecutar ahora PLA pero con los datos simulados en los apartado 2b de la sección 1

En este caso, las dos veces llega al numero máximo de iteracciones, es decir **1000 iteracciones**. Esto se debe a que el perceptron no es capaz de convencer cuando hay ruido en las muestras de datos. Ya que siempre va a cambiar algún elemento y nunca a va ser capaz de recorrer todos los datos sin cambiar nada.

2.2. Regresión logística

En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos $\mathcal{D}$ para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores 0/1 y por tanto que la etiqueta y es una función determinista de $\mathbf{x}$.
Consideremos $d = 2$ para que los datos sean visualizables, y sea $\mathcal{X} = [0, 2] \times [0, 2]$ con probabilidad uniforme de elegir cada $\mathbf{x} \in \mathcal{X}$. Elegir una linea en el plano que pase por $\mathcal{X}$ como frontera entre $f(x) = 1$ (donde y toma valores $+1$) y v$f(x) = 0$ (donde y toma valores -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos. Seleccionar $N = 100$ puntos aleatorios $[x_n]$ de $\mathcal{X}$ y evaluar las respuestas $\{y_n\}$ de todos ellos respecto de la frontera elegida.

2.3. Implementar RL con SGD bajo las siguientes condiciones

1. Inicializar el vector de pesos a 0
2. Parar el algoritmo cuando $\left\|\mathbf{w}^{(t-1)} - \mathbf{w}^{(t)}\right\| < 0,01$, donde $w^{(t)}$ denota el vector de pesos al final de la época t. Una época es un pase completo a través de los **N** datos.
3. Aplicar una permutanción aleatoria $1, 2, ..., N$, en el orden de los datos antes de usarlos en cada epoca del algoritmo.

4. Usar una tasa de aprendizaje de $\eta = 0,01$

Era un código bastante largo que he evitado poner aquí. No obstante esta todo en los ficheros Python que se adjuntan con la practica.

Podemos ver como RL ajusta bien para la muestra que tenemos de 100 puntos.

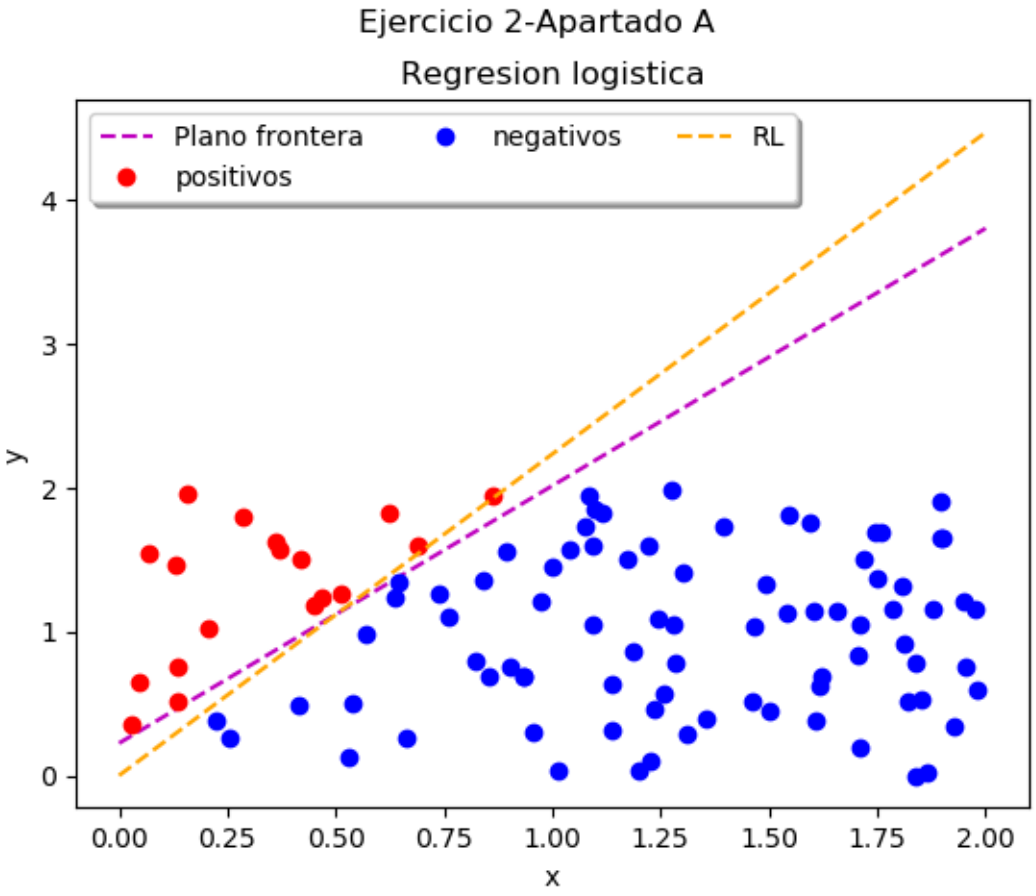

Figura 9: 100 puntos uniformes, RL y el plano frontera original

2.4. Usar una muestra de datos etiquetada para encontrar nuestra solución g y estimar E_{out} usando para ello un número suficientemente grande de nuevas muestras (> 999)

Usando los mismo pesos que hemos obtenido en el apartado anterior, vamos obtener que el error va aumentar, ya que habrá puntos que se encuentre entre el plano frontera y el hiperplano que nos devuelven los pesos.

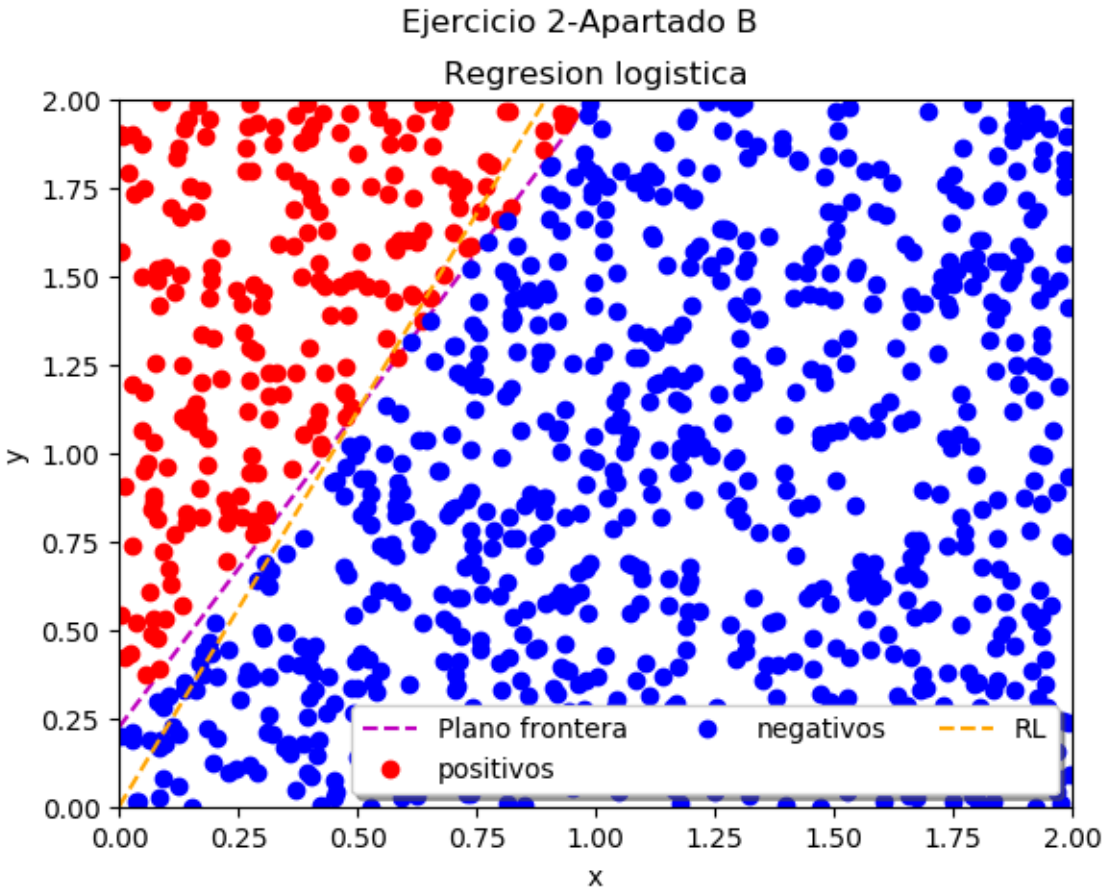

Figura 10: 1000 puntos uniformes, RL y el plano frontera original

El e_{out} = 0,11304005135194581, que aun siendo bajo, ya es algo mas significativo.