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

Práctica 1

UNIVERSIDAD
DE GRANADA

Antonio Jesús Heredia Castillo

18 de abril de 2019

Índice

Índice de figuras

Índice de tablas

1. Gradiente descendente

1.1. Implementar el algoritmo de gradiente descendente

He querido implementar el algoritmo del gradiente descendente de la forma mas general que he podido. Para ello, he creado una función que lo calcula. Esta función recibe los siguientes parámetros:

- lr (float): Tasa de aprendizaje
- punto (array[x,y]): Punto desde el cual se quiere aplicar el algoritmo
- función (float): Función a la que se le quiere aplicar el gradiente descendente
- iteraciones (int): Numero máximo de iteraciones que aplicara el algoritmo

Esta subrutina se encarga de hacer las derivadas parciales de la función y de calcular de forma iterativa el punto mínimo de la función. En esta primera versión del algoritmo, la única condición de parada va a ser el numero de iteraciones que le pasa el usuario. Podemos ver el código del algoritmo aquí.

```
_iter = 0 #Contador de iteraciones
_dx = sp.diff(funcion, x) #Derivadas parciales
_dy = sp.diff(funcion, y)
#Lambdifyco las derivadas parciales para calcular mas rapido el valor
_lam_dx = sp.lambdify((x,y),_dx)
_lam_dy = sp.lambdify((x,y),_dy)
while True:
        #creo una copia del punto, para que a la hora de evuluar el x,
        #no lo pierda para y
        _punto_copia = np.copy(_punto)
        #si hay mas iteraicones de las que yo quiero paro
        if _iter > _iteraciones:
                break
        #calculo el gradiente
        _punto[0] = _punto[0]-_lr*_lam_dx(_punto_copia[0],_punto_copia[1])
        _punto[1] = _punto[1]-_lr*_lam_dy(_punto_copia[0],_punto_copia[1])
        _iter =_iter+1
return _punto
```

1.2. Considerar la Función $E(u,v) = (u^2e^v - 2v^2e^{-u})^2$. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto $(u,v) = (1,1)$ y usando una tasa de aprendizaje $\eta = 0,01$.

1.2.1. Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u,v)$

Lo primero que tenemos que saber es que para calcular el gradiente, necesitamos las derivadas parciales de $E(u,v)$.
La derivada respecto de u seria:

$$\frac{\partial E(u,v)}{\partial u} = 2\left(u^2e^v - 2v^2e^{-u}\right)\left(2e^v u + 2v^2e^{-u}\right) \tag{1}$$

La derivada respecto de v seria:

$$\frac{\partial E(u,v)}{\partial v} = 2\left(u^2e^v - 2v^2e^{-u}\right)\left(u^2e^v - 4e^{-u}v\right) \tag{2}$$

El vector gradiente seria las dos derivadas parciales en el punto $(u,v) = (1,1)$.

$$\nabla E(1,1) = (E_u(1,1), E_v(1,1)) = (12{,}236771, 2{,}4717385) \tag{3}$$

Ahora con las derivadas, ya podemos obtener el punto a donde tiende el gradiente con ese η. Los nuevos valores de u y v serán:

$$u' = u - 2\eta\left(u^2e^v - 2v^2e^{-u}\right)\left(2e^v u + 2v^2e^{-u}\right) \tag{4}$$

$$v' = v - 2\eta\left(u^2e^v - 2v^2e^{-u}\right)\left(u^2e^v - 4e^{-u}v\right) \tag{5}$$

Por lo tanto para el punto $(u,v) = (1,1)$ el valor del gradiente seria:

$$u' = 1 - 2 * 0{,}01\left(1^2e^1 - 21^2e^{-1}\right)\left(2e^1 + 21^2e^{-1}\right) = 0{,}75526458 \tag{6}$$

$$v' = 1 - 2 * 0{,}01\left(1^2e^1 - 21^2e^{-1}\right)\left(1^2e^1 - 4e^{-1}1\right) = 0{,}95056523 \tag{7}$$

1.2.2. ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u,v)$ inferior a 10^{-14} . (Usar flotantes de 64 bits)

Para realizar este ejercicio, solo he metido una condición de parada en el algoritmo. La condición de parada en este caso es que el valor de evaluar $E(u,v)$ en un determinada iteración sea $E(u,v) < 10^{-14}$

```
if _iter > _iteraciones or
   _lam_f(_punto_copia[0],_punto_copia[1]) < np.float64(10**-14):
        print ("La funcion ha acabado a las "
        + str(_iter) + " iteraciones")
        break
```

En este caso el numero de iteraciones necesarias ha sido **33**, como se puede ver en la Figura1.

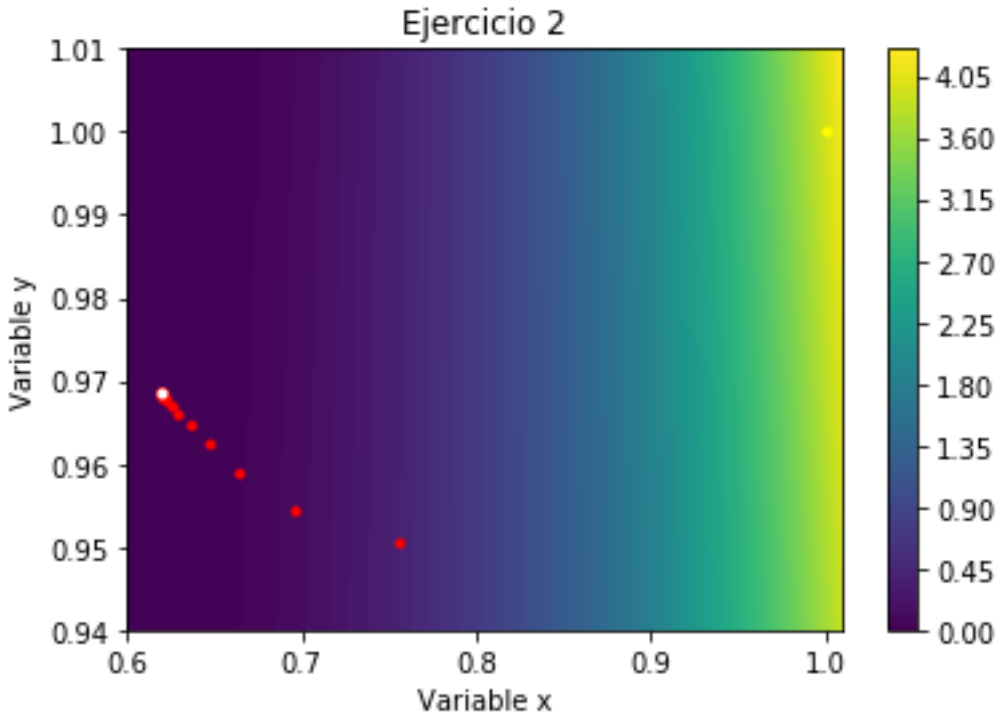

Figura 1: Representación de como avanza el gradiente en la función $E(u,v)$. Punto en amarillo inicio y punto final en blanco.

1.2.3. ¿En qué coordenadas (u, v) se alcanzó por primera vez un valor igual o menor a 10^{-14} en el apartado anterior.

Como el algoritmo del gradiente me devuelve el punto en el que acaba, lo único que tengo que hacer es mostrarlo por pantalla. En este caso el punto en el que se alcanza 10^{-14} es $(0{,}61920768, 0{,}96844827)$.

1.3. Considerar ahora la función $f(x, y) = x^2 + 2y^2 + 2\sin(2\pi x)\sin(2\pi y)$

1.3.1. Usar gradiente descendente para minimizar esta función. Usar como punto inicial $(x_0 = 0{,}1, y_0 = 0{,}1)$, (tasa de aprendizaje $\eta = 0,01$ y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando $\eta = 0{,}11$, comentar las diferencias y su dependencia de η.

Usando un $\eta = 0{,}01$, podemos gráficamente Figura 2 como el gradiente desciende de forma adecuada. En este caso acaba en una altura de $-1{,}8200785415471563$

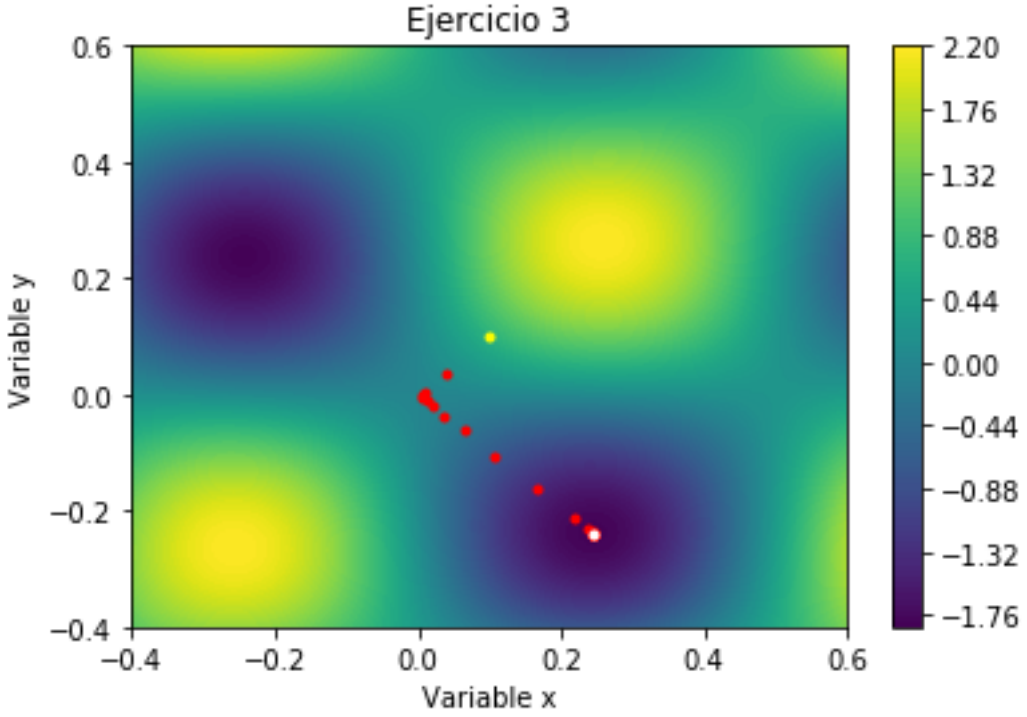

Figura 2: Gradiente de la función $F(u, v)$ con $(x_0 = 0{,}1, y_0 = 0{,}1)$ y $\eta = 0{,}01$. Punto en amarillo inicio y punto final en blanco.

En cambio cuando usamos $\eta = 0{,}1$, podemos ver (Figura 3) como no funciona bien. En este caso acaba en una altura de 2,5007003656467446, que es incluso mas alto que desde donde empezamos (0,7209830056250534).

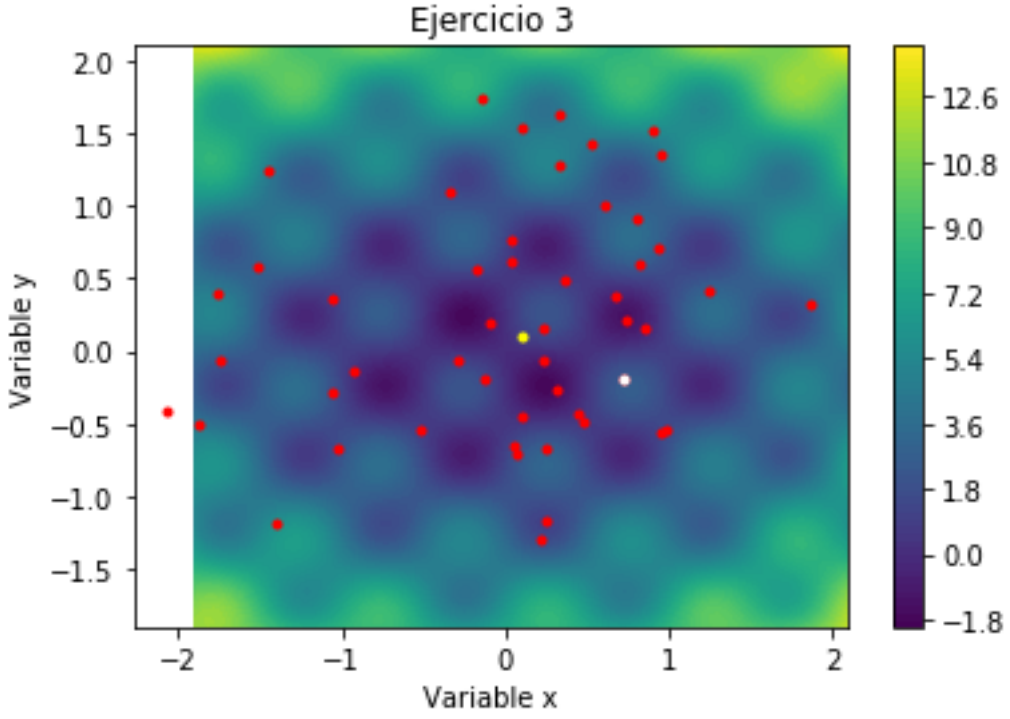

Figura 3: Gradiente de la función $F(u, v)$ con $(x_0 = 0{,}1, y_0 = 0{,}1)$ y $\eta = 0{,}1$. Punto en amarillo inicio y punto final en blanco.

Las diferencias entre usar un η muy grande o uno adecuado es clara viendo las imágenes, con η adecuado se encuentra "rápidamente" el mínimo. Con el η muy grande, el algoritmo da saltos muy grandes y por lo tanto no llega a encontrar el mínimo debido a que de una iteración a otra lo excede y empieza a rebotar sin encontrar nada.

1.3.2. Obtener el valor mínimo y los valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija: $(0, 1, 0, 1), (1, 1), (-0, 5, -0, 5), (-1, -1)$. Generar una tabla con los valores obtenidos

Los datos obtenidos podemos verlos en la Tabla 1. En este caso, no he dejado el calculo de todos los puntos en el código para no sobrecargar la salida por pantalla con datos y que el tiempo de ejecución no fuera muy alto. El campo "valor" indica la altura que alcanza la función en el punto (x, y).

Tabla 1: La altura minima conseguida en los diferentes puntos con GD

Punto / Valor final	$(0{,}1, 0{,}1)$	$(1, 1)$	$(-0{,}5, -0{,}5)$	$(-1, -1)$
x	0.24380497	1.2180703	-0.73137746	-1.2180703
y	-0.23792582	0.71281195	-0.23785536	-0.71281195
Valor mínimo	-1.82007854	0.59326937	-1.33248106	0.59326937

1.3.3. ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

Teniendo en cuenta los datos obtenidos en los apartados anteriores y como afecta tanto la tasa de aprendizaje como el punto donde se inicia el algoritmo. Llego a la conclusión que la verdadera dificultad radica en la elección de estos dos parámetros. Por ejemplo al elegir un "mal" punto de inicio puede no encontrar el punto mínimo absoluto de la función, si no un punto mínimo relativo a la posición desde donde ha iniciado.

Elegir una tasa de aprendizaje que no sea adecuada también nos va a dar bastantes problemas. Esto se debe a que si es muy grande, el algoritmo nunca va a llegar a converger debido a que va a ir rebotando de un lado al otro del mínimo, pero sin llegar a acercarse nunca. Si lo elegimos muy pequeño, el algoritmo daría pasos muy pequeños y tardaría demasiado en llegar al mínimo.

Por lo tanto desde mi punto de vista una vez que tienes todos los algoritmos implementados, la dificultad radica en elegir buenos valores para conseguir que nuestro algoritmo funcione de forma adecuada.

2. Regresión lineal

2.1. Estimar un modelo de regresión lineal a partir de los datos proporcionados usando tanto el algoritmo de la pseudoinversa como SGD

En mi SGD, para cada iteración, lo que realizado primero es realizar una copia desordenada de todo el array de los datos junto con sus etiquetas. Cuando están desordenados empiezo a sacar minibatch's del tamaño que se le indique, y los saca hasta que no queden datos en el array. En la siguiente empiezo el proceso de nuevo.

```
while _error >_epsilon and _iter < _iteraciones:
        #realizo la mezcla de los datos
        _batch, _y_batch = shuffle(_X_train, _y_train)

        #cojo minibatchs hasta que se acaben
        while _batch.shape[0] != 0:
                #cojo un minibatch y sus etiquetas correspondientess
                _minibatch = _batch[:_minibatch_size,:]
                _y_mini = _y_batch[:_minibatch_size]

                #elemino ese minibatch para no coger en la siguiente iteracion
                _batch = np.delete(_batch,np.s_[:_minibatch_size:], axis = 0)
                _y_batch = np.delete(_y_batch,np.s_[:_minibatch_size:])

                #calculo el gradiente
                _w = _w - _lr * _gradiente(_w, _minibatch, _y_mini)
                #aumento el numero de iteraciones
                _iter =_iter+1
```

2.1.1. Pintar las soluciones obtenidas junto con los datos usados en el ajuste.

La configuración que he usado para realizar el SGD ha sido la siguiente:

- $\eta = 0{,}01$.
- Tamaño del minibatch: 64.
- N$^{\text{o}}$ de iteraciones: 1000.

Los pesos que he conseguido con esta configuración de SGD y la pseudoinversa los podemos ver en la Tabla 2

Tabla 2: Los pesos conseguido con la pseudoinversa y con SGD

Pesos / Método	w_0	w_1	w_2
Pseudoinversa	-1.11588	-1.2486	-0.497532
SGD	-1.24421	-0.180105	-0.458887

Podemos ver los hiperplanos que conseguimos con estos pesos en la Figura 4

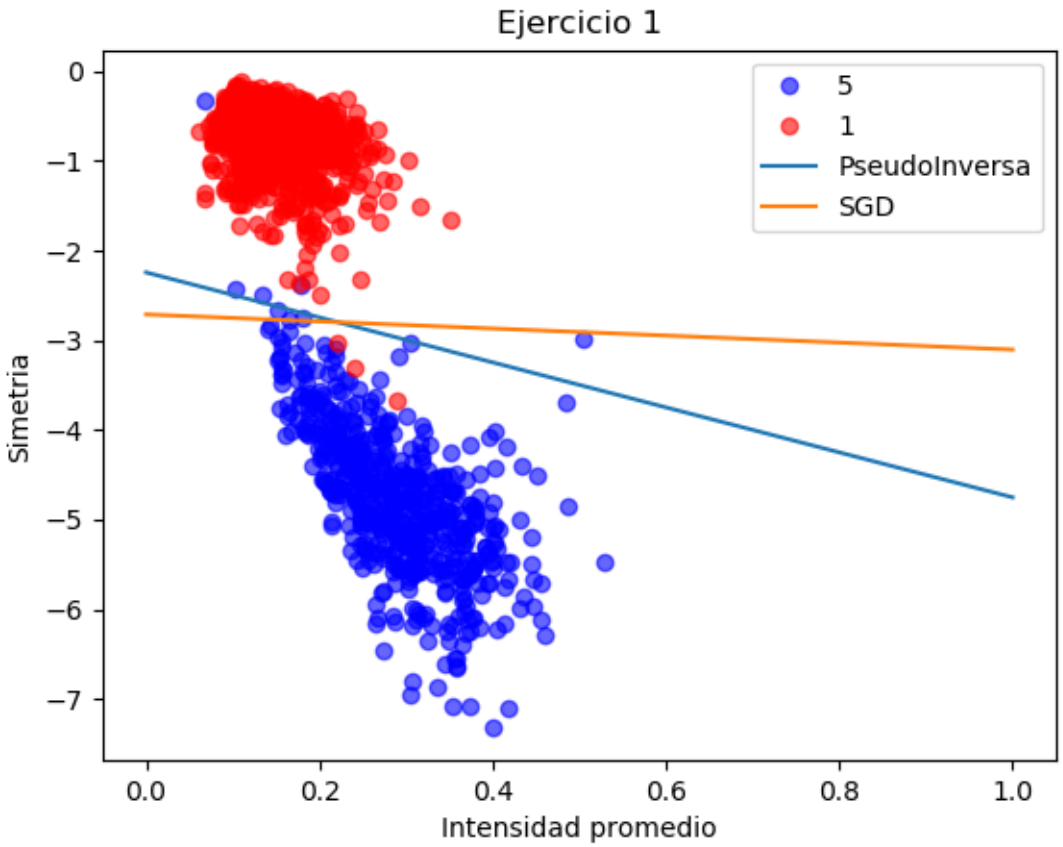

Figura 4: Representación de los hiperplanos que conseguimos con la pseudo-inversa y con SGD junto a los datos de entrenamiento.

2.1.2. Valorad la bondad del resultado usando E_{in} y E_{out}

Los errores que obtenemos con ambos modelos son los que podemos ver en la Tabla 3

Tabla 3: Errores obtenidos con la pseudoinversa y con SGD

Error / Método	E_{in}	E_{out}
Pseudoinversa	0.07918658628900402	0.13095383720052586
SGD	0.08164527370294046	0.1351534264237533

Viendo los datos del error, podemos observar que los datos de error no son tan diferentes en los datos de entrenamiento. Pero en cambio en los datos

de test, si que son mas grande para ambos, siendo aun mayores en el SGD que en la pseudoinversa.

2.2. En este apartado exploramos como se transforman los errores E_{in} y E_{out} cuando aumentamos la complejidad del modelo lineal usado.

Ahora hacemos uso de la función $simula_unif(N, 2, size)$ que nos devuelve N coordenadas 2D de puntos uniformemente maestreados dentro del cuadrado definido por $[-size, size][-size, size]$

2.2.1. Generar una muestra de entrenamiento de N = 1000 puntos en el cuadrado $X = [-1, 1][-1, 1]$. Pintar el mapa de puntos 2D.

Genero los puntos y los muestro en la Figura 5

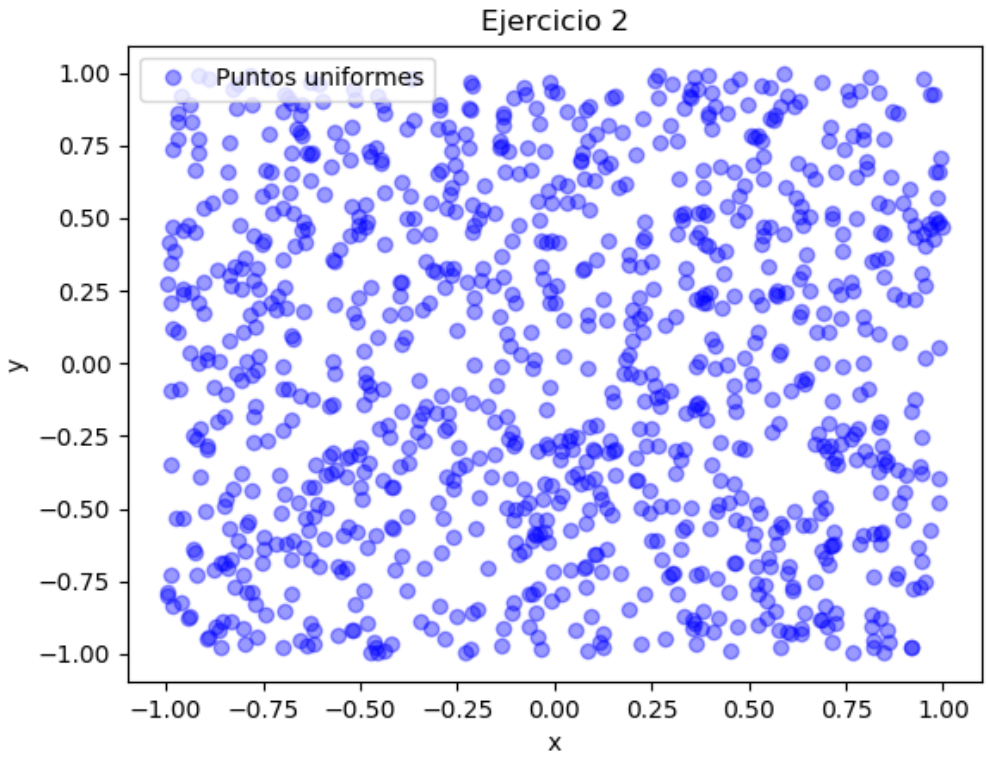

Figura 5: Representación de los puntos.

2.2.2. Consideremos la función $f(x_1, x_2) = sign((x_1 - 0,2)^2 + x_2^2 - 0,6)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10 % de las mismas. Pintar el mapa de etiquetas obtenido.

Una vez etiquetados los puntos y añadido el ruido obtenemos la Figura 6

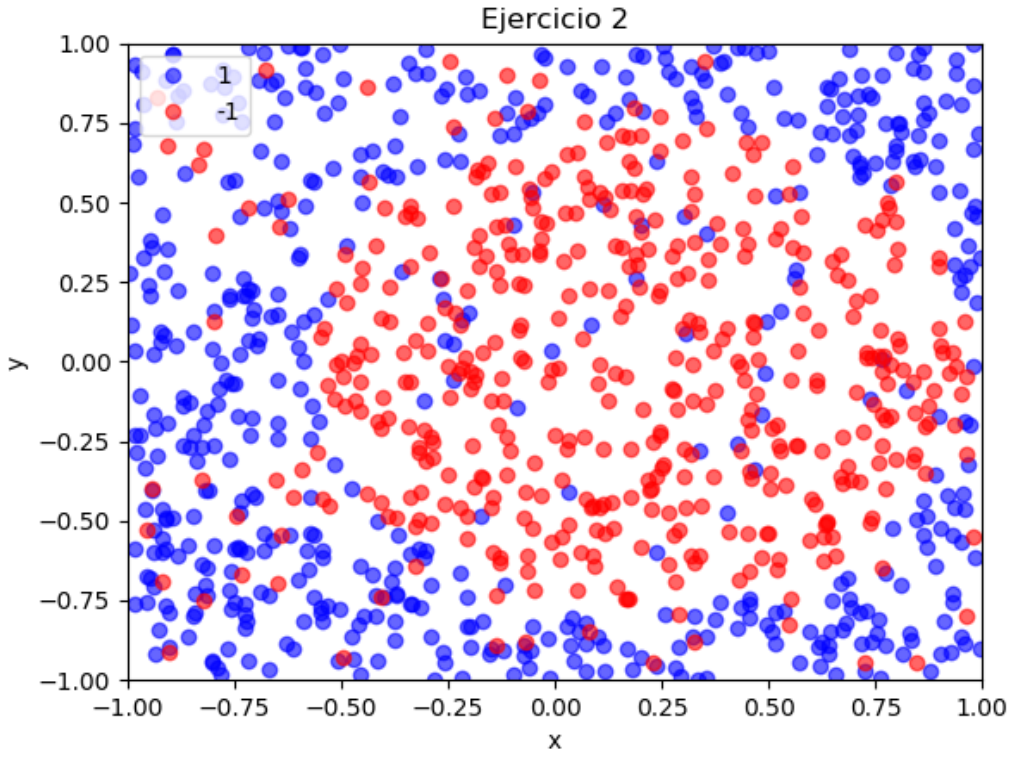

Figura 6: Representación de los puntos etiquetados.

2.2.3. Usando como vector de características (1, x 1 , x 2) ajustar un modelo de regresión lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste E in usando Gradiente Descendente Estocástico (SGD).

La configuración que he usado para realizar el SGD ha sido la siguiente:

- $\eta = 0{,}01$.
- Tamaño del minibatch: 64.

- Nº de iteraciones: 50.

He elegido esta configuración con tan pocas iteraciones ya que en el siguiente apartado deberemos ejecutarla 1000 veces. Entonces para que se obtengan datos adecuados y en un tiempo razonable, he elegido esta configuración.
El E_{in} que he obtenido es de 0,9459233234186475 y el vector de pesos es el siguiente: $w = [-0{,}00095213, -0{,}16145304, 0{,}02704213]$. Que realizando pruebas con mas iteraciones no difiere mucho de lo obtenido.El modelo generado quedaría como lo que se ve en la Figura 7

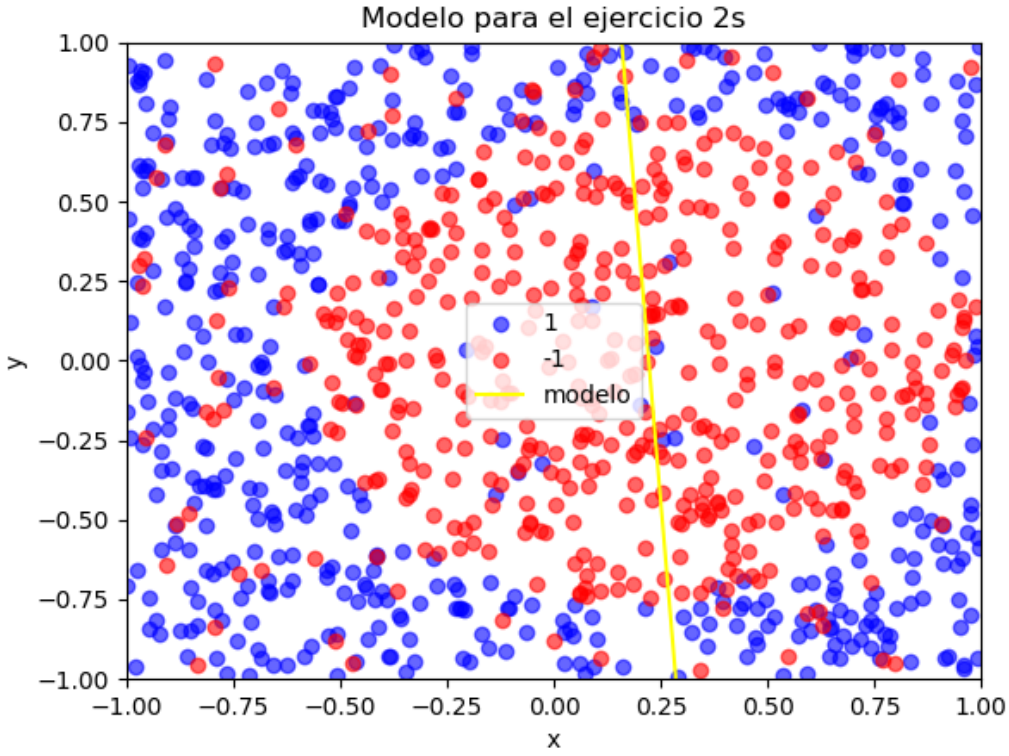

Figura 7: Representación de los puntos etiquetados junto al modelo obtenido por SGD.

2.2.4. Ejecutar todo el experimento definido 1000 veces (generamos 1000 muestras diferentes) y calcular tanto E_{in} como E_{out} (generaremos 1000 datos para cada test)

He realizado el experimento con los mismos parámetros que en el ejercicio anterior. Y he obtenido que:

El valor medio para el E_{in} en las 1000 iteraciones es de 0,963238531019716.
El valor medio para el E_{out} en las 1000 iteraciones es de 0,9551645546822114.

2.2.5. Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{out} y E_{in}

Podemos ver que el E_{out} es menor, esto se debe a que los test que he utilizado no tenían ningún "ruido" y por lo tanto, en media, se ajusta mejor. De una iteración a hacer la media con 1000 el E_{in} no tiene un cambio tan acuciado.
Podemos ver que esta vez el ajuste no es tan bueno como en casos anteriores. Esto se debe a que la función que podría ajustar nuestros datos, no es lineal, como la que realizamos ahora, si no que debería usar alguna función que tuviera curvatura.