Aprendizaje Automático
Trabajo 1:
Programación

Rafael Vázquez Conejo

Marzo 2020

Índice

1. Ejercicio sobre la Búsqueda Iterativa de óptimos

1.1. Implementar el algoritmo de gradiente descendiente

El algoritmo de gradiente descendiente es un algoritmo de optimización usado para minimizar funciones de forma iterativa. Partiendo de un punto inicial nuestro objetivo es alcanzar un mínimo local.

La función “gd” recibe como parámetros:

- La posición de inicio (w), en este caso se trata de un punto con dos coordenadas.
- La tasa de aprendizaje, learning rate (lr), representa el tamaño del paso que habrá entre las distintas iteraciones.
- Gradiente de la función que buscamos minimizar (grad_fun), en este primer caso nuestra función será E(u, v), y en el siguiente será F(x, y), ambas funciones definidas tanto en el enunciado como en mi código.
- El siguiente parámetro (fun) nos permite diferenciar cuando el algoritmo es llamado para la función E(u, v), y cuando para F(x, y). Cuando éste tome el valor de 0 será porque estamos actuando sobre E(u, v), mientras que si su valor es de 1, nuestra función será F(x, y).
- El error mínimo (epsilon) que esperamos alcanzar antes de que se cumpla el máximo de iteraciones establecido. Este valor será una condición de parada cuando actuemos con la función E(u, v).
- Número máximo de iteraciones, por defecto será de 10000 iteraciones.

Respecto a los valores que devuelve esta función:

- En el caso de actuar sobre la función E(u, v) devolverá el número de iteraciones ejecutadas.
- En el caso de actuar sobre la función F(x, y) devolverá un array con todas la iteraciones y otro con el valor de la función en cada iteración realizada.
- En ambos casos devolverá un punto en el cual el gradiente se habrá minimizado hasta tal punto en el cual, o bien ha superado el número máximo de iteraciones, o bien ha obtenido un valor inferior al epsilon.

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

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

El gradiente de E(u, v) es fácilmente calculable si seguimos la regla de la cadena al calcular sus derivadas parciales. La expresión del gradiente sería:

$$\nabla E(u,v) = \begin{bmatrix} \frac{\partial E(u,v)}{\partial u} \\ \frac{\partial E(u,v)}{\partial v} \end{bmatrix}$$

Dónde:

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

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

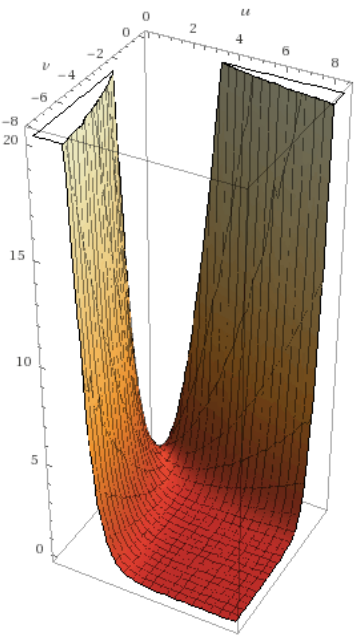
b) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u, v) inferior a 10 −14 ? (Usar flotantes de 64 bits)

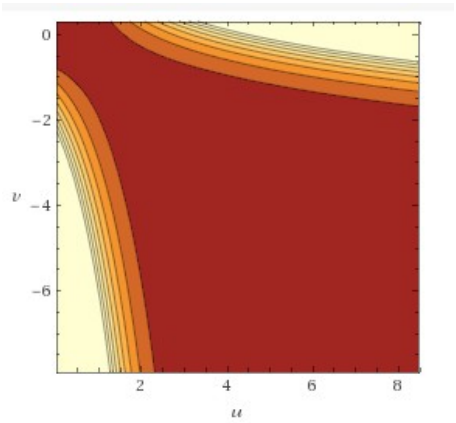
El algoritmo necesita un total de 10 iteraciones

c) ¿En qué coordenadas (u, v) se alcanzó por primera vez un valor igual o menor a 10 −14 en el apartado anterior?

Se alcanzó por primera vez en las coordenadas (0.04473629039778204 , 0.02395871409914174).

El valor alcanzado es de E(u, v) = $1.208683393457121 \cdot 10^{-15}$





Gracias a la gráfica que he obtenido en wolframalpha.com, podemos observar que se desciende muy rápido en las primeras iteraciones, por ello hemos necesitado muy pocas iteraciones para encontrar un mínimo.

1.3. Considerar ahora la función f (x, y) = (x − 2)² + 2(y + 2)² + 2 sin(2πx) sin(2πy)

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial (x 0 = 1, y 0 = −1), (tasa de aprendizaje η = 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 η = 0,1, comentar las diferencias y su dependencia de η.

Volvemos a calcular el gradiente de la función que optimizaremos:

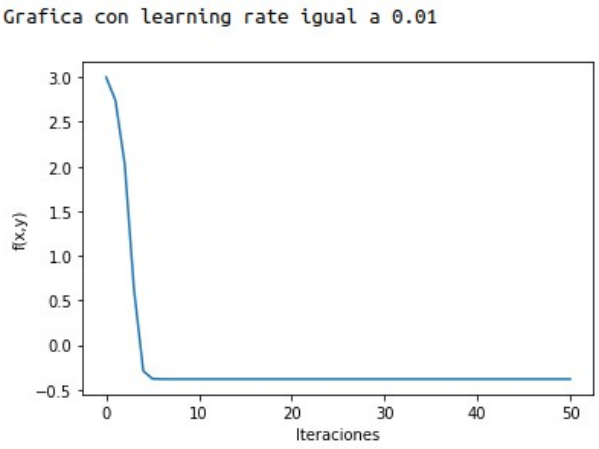
$$\nabla f(x,y)=\begin{bmatrix}\frac{\partial f(x,y)}{\partial x}\\ \frac{\partial f(x,y)}{\partial y}\end{bmatrix}$$

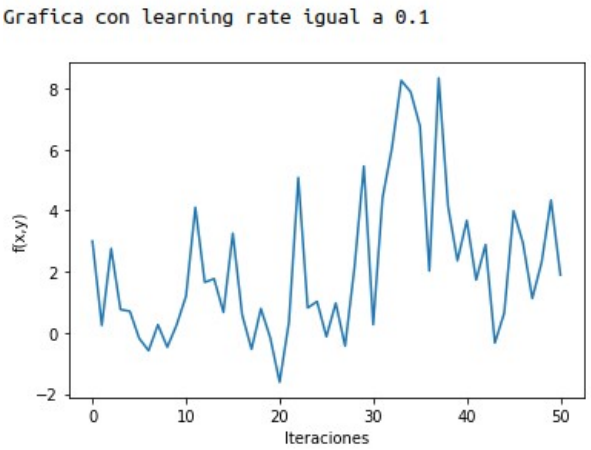
Dónde:

$$\frac{\partial f(x,y)}{\partial x} = 2\,(\,2\,\pi\cos(\,2\,\pi\,x\,)\sin(\,2\,\pi\,y\,)+x-2\,)$$

$$\frac{\partial f(x,y)}{\partial y} = 4\,(\,\pi\sin(\,2\,\pi\,x\,)\cos(\,2\,\pi\,y\,)+y-2\,)$$

Observemos los gráficos generados en cada caso:





Gradiente descendente sobre F(x,y), learning rate = 0.01 e iteraciones = 50

Valor mínimo = -0.3812494974381

Gradiente descendente sobre F(x,y), learning rate = 0.1 e iteraciones = 50

Valor mínimo = 0.060882024317680195

La tasa de aprendizaje determina “la amplitud de los saltos” que va realizando el gradiente a través de la función conforme avanza en su búsqueda de un mínimo.
Comenzando con el caso en el que la tasa es muy pequeña (0.01), los saltos realizado son muy cortos. Observamos que al principio desciende bruscamente al mínimo y a partir de ahí el valor se mantiene invariable.

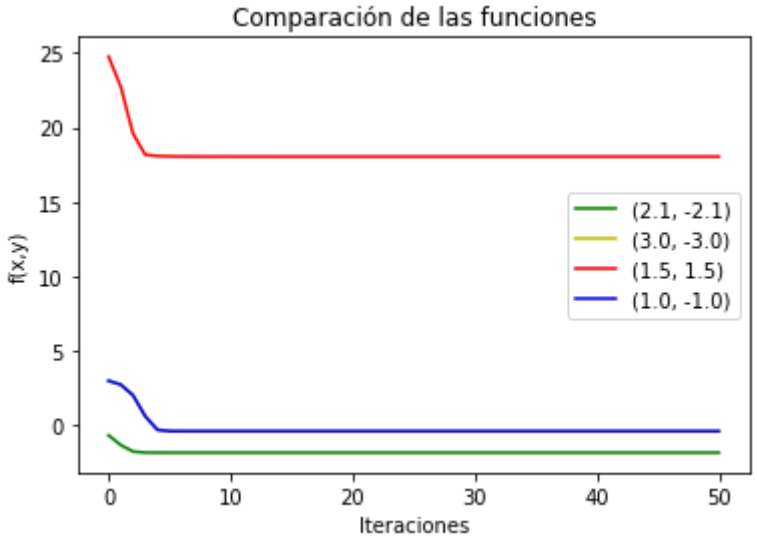
En el segundo caso los pasos que realiza son muy amplios. Son tan altos que comienza a saltarse óptimos locales, esto es lo que provoca tantas fluctuaciones en el gráfico. Además alcanzamos un mínimo alto (0.060882024317680195) comparado con el anterior. Observando los resultados podemos concluir que la tasa de aprendizaje tiene un valor relativamente grande (0.1).

b) Obtener el valor mínimo y los valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija en: (2,1, −2,1), (3, −3),(1,5, 1,5),(1, −1). Generar una tabla con los valores obtenidos.

Por defecto he establecido la tasa de aprendizaje con un valor de 0.01, puesto que en el apartado anterior obtuve un resultado más adecuado.

Tabla de valores:

Punto de inicio	Coordenada x	Coordenada y	Valor de la función
[2.1, -2.1]	2.2438	-2.23793	-1.82008
[3.0, -3.0]	2.73094	-2.71328	-0.381249
[1.5, 1.5]	1.77798	1.03201	18.0421
[1.0, -1.0]	1.26906	-1.28672	-0.381249



(3.0, -3.0) y (1.0, -1.0)

Podemos observar que cuando partimos del punto (3.0, -3.0) ó del punto (1.0, -1.0) ambos se estabilizan en el mismo mínimo local. El mejor mínimo local lo encontramos cuando comenzamos por el punto (2.1, -2.1).

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

Aunque es importarte fijar un correcto criterio de parada para buscar el equilibrio entre tiempos de cómputo y calidad de la solución, concluiría que saber elegir un punto de inicio y una tasa de aprendizaje (learning rate) adecuados es lo más importante además de difícil. Estos dos parámetros cuya configuración desconocemos a priori son los que, según he comprobado, influyen con mayor fuerza en el proceso de optimización, pudiendo obtener resultados totalmente distintos al ser estos modificados.

- El punto de inicio: hemos comprobado en el apartado anterior que dependiendo del punto en el que comencemos se pueden obtener mejores o peores resultados. Puede que estemos cerca de un óptimo (local o global) y lo encontremos en pocos pasos, o puede que sea la situación contraria, estemos lejos y no lo encontremos.
- La tasa de aprendizaje: tiene una gran importancia, ya que como hemos mencionado en el apartado “a” del ejercicio 3, determina la amplitud de los “saltos” que realizaremos en la función para encontrar el óptimo. Si establecemos una tasa de aprendizaje baja, puede que ralentice el proceso; si por el contrario es una alta puede hacer que nos saltemos la solución.

Estos motivos me llevan a la conclusión de que el algoritmo de gradiente descendente no es tan bueno como puede parecer al comienzo. Su funcionamiento para funciones convexas es de los mejores, pero para las demás funciones no lo es tanto.

2. Ejercicio sobre Regresión Lineal

2.1 Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudo- inversa como Gradiente descendente estocástico (SGD). Las etiquetas serán {−1, 1}, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando E in y E out (para E out cal- cular las predicciones usando los datos del fichero de test).

Comenzamos con la implementación del algoritmo del gradiente descendente estocástico, con el cual supondremos que vamos a minimizar el error de ajuste, cuyo gradiente tendremos que calcular.

Cada iteración emplea un subconjunto de datos pertenecientes a una partición aleatoria del conjunto de datos de partida, variable llamada minibatch, y que utilizaremos para evaluar el gradiente del error, lo que nos aportará una mayor variedad al evaluar al gradiente en cada iteración.

Los algoritmos se encuentran comentados e implementados en el .py.

En ambos casos he establecido una tasa de aprendizaje de 0.01, un tamaño del minibatch de 32 y un número máximo de iteraciones de 6000. Por defecto he establecido un epsilon de 10^{-20}.

- Para el Gradiente Descendente Estocástico se define con la expresión matricial:

$$Err(w) = \frac{1}{N} \| X w - y \|^2$$

Y para minimizarla habrá que usar el gradiente:

$$\nabla Err(w) = \frac{2}{N} X^T (X w - y)$$

Valores obtenidos con el Gradiente Descendente Estocástico:

$\mathbf{E_{in}} = 0.08307527$
$\mathbf{E_{out}} = 0.13903238$

- Para el caso de la Pseudo-inversa:

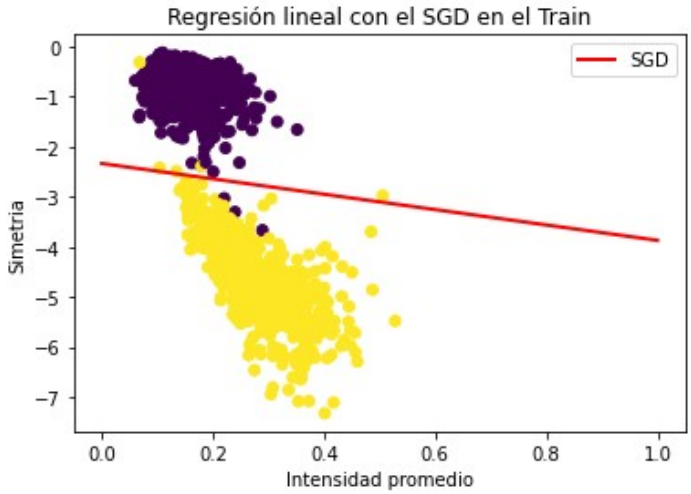
Este método es una alternativa de cálculo totalmente analítico en el que sólo se tiene en cuenta el propio conjunto de datos. Se debe calcular:

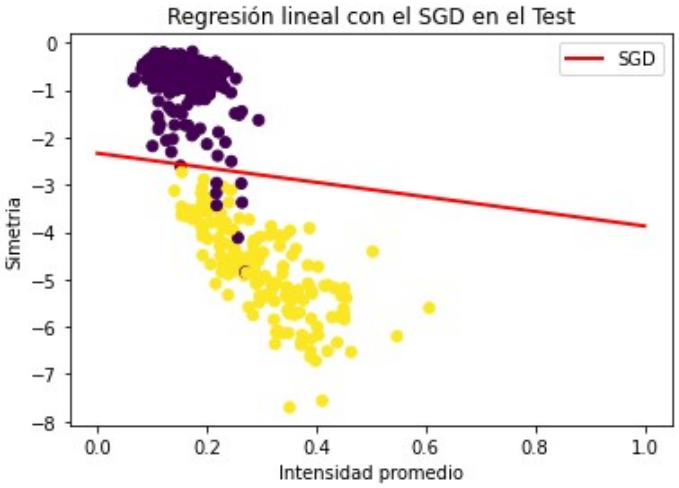
$w = X^T y;$ Dónde: $X^t = (X^T X)^{-1} X^T$

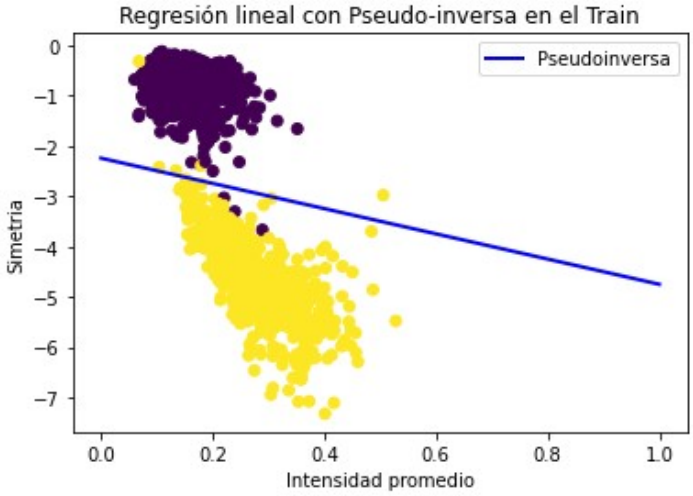
En nuestro caso no hay problemas, porque la dimensionalidad de nuestros datos es pequeña, pero si fuera mayor el cálculo de X^t es bastante costoso.

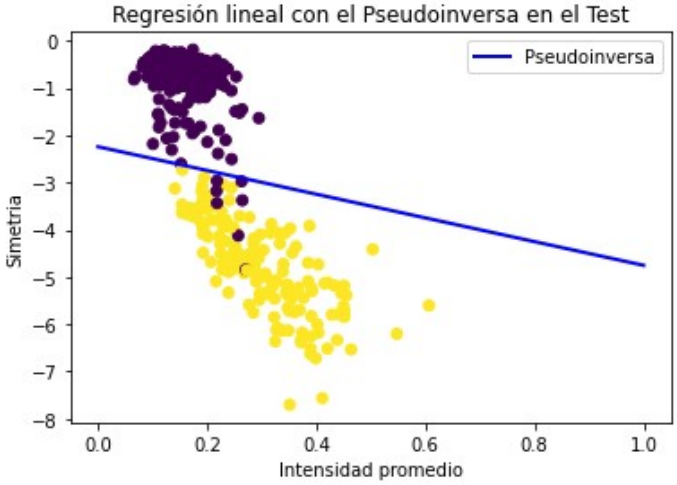
Valores obtenidos con la Pseudo-inversa:

$\mathbf{E_{in}} = 0.07918659$
$\mathbf{E_{out}} = 0.13095384$







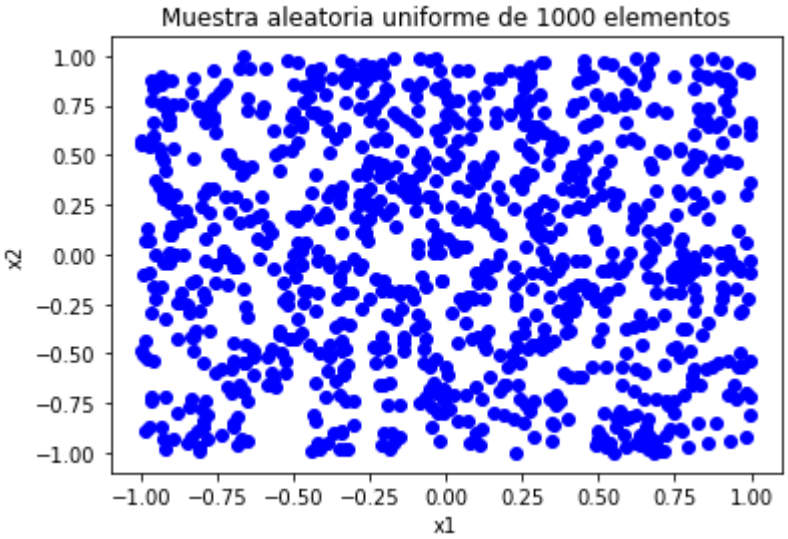


Respecto a la bondad de los resultados, observamos que los ajustes están bastante correctos. En ambos casos hemos obtenido una valores muy similares, con un valor de E_{in} de 0.08 aproximadamente y de 0.13 para el E_{out}. Las gráficas insertadas, tanto del Training como del Test, nos permiten observar que en ambos algoritmos la línea que establecen de separación entre ambos conjuntos es bastante razonable.

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 muestreados dentro del cuadrado definido por [−size, size] × [−size, size]

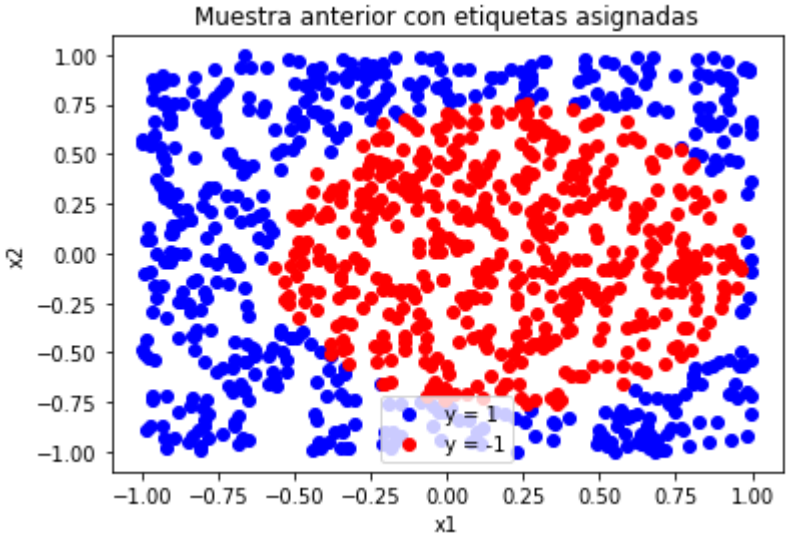
a) Generar una muestra de entrenamiento de N = 1000 puntos en el cuadrado X = [−1, 1] × [−1, 1]. Pintar el mapa de puntos 2D. (ver función de ayuda)

Muestra generada:

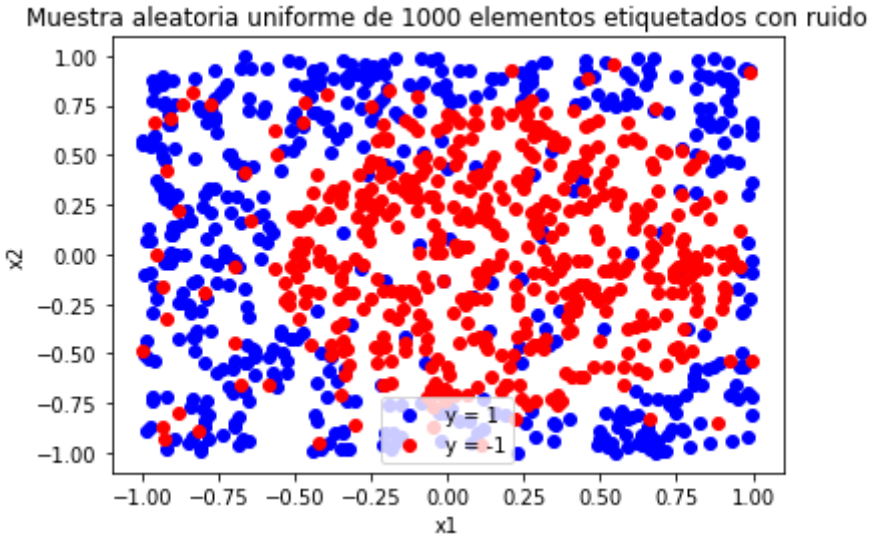


b) Consideremos la función f (x1 , x2) = sign((x₁ − 0,2) ² + x₂² − 0,6) que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10 % de las mismas. Pintar el mapa de etiquetas obtenido.

Para realizarlo he definido el método “asignar_etiquetas”, en el cual utilizalo función f(x1, x2) definida en el enunciado para asignarle una etiqueta a cada valor. Si la función para ese punto es mayor o igual a 0, le asigno el color azul, y en caso contrario, el rojo. Con lo cual el gráfico resultante sería:



A continuación introduciremos un 10% de ruido, tal y como se nos pide en el enunciado. Para ello he cambiado la etiqueta de un 10% de la muestra de forma aleatoria, cambiando el signo de las etiquetas asignadas anteriormente. Con lo cual he obtenido:



c) 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).

He establecido una tasa de aprendizaje de 0.01, un tamaño de minibatch de 32 y un máximo de 1000. Además la semilla para establecer los valores aleatorios la he inicializado a 49137372 (mi dni).
Para añadir la nueva columna con 1's e implementado un método add_col_1(), que recibe como parámetro nuestro antiguo vector de características.

Muestra aleatoria de 1000 elementos	
Ajuste "w" obtenido	[[0.23643854] [-0.60111915] [0.12138666]]
E_{in}	1.0172768

d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes). Calcular el valor medio de los errores E in de las 1000 muestras. Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E out en dicha iteración. Calcular el valor medio de E out en todas las iteraciones.

1000 muestras aleatorias de 1000 elementos	
E_{in} promedio	1.0193697394413286
E_{out} promedio	1.0254731216542567

Repito el procedimiento realizado con el segundo vector de características usando la misma muestra que en el primer vector de características, obteniendo:

El ajuste obtenido en este caso en la muestra aleatoria de 1000 elementos:

Muestra aleatoria de 1000 elementos	
Ajuste “w” obtenido	[[-0.61157372] [-0.48530898] [0.02236191] [-0.40698598] [0.57550975] [1.07494854]]
E_{in}	0.64876217

Y al repetir todo el experimento 1000 veces, generando 1000 muestras diferentes:

1000 muestras aleatorias de 1000 elementos	
E_{in} promedio	0.6665473392724605
E_{out} promedio	0.6753081576711677

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

Observamos que en el primer experimento, con el primer vector de características, hemos obtenido resultados decentes dadas las circunstancias.

No hemos obtenido un valor de error lo suficientemente pequeño, pero un error prácticamente de 1 tanto en E_{in} como en E_{out}., parece razonable. Razonable pues partimos de haber realizado una regresión lineal sobre un conjunto de datos que no seguían un comportamiento lineal, y además tenemos que tener en cuenta la fuerte presencia de ruido que hemos añadido al cambiar las etiquetas de un 10% de la muestra.

Si observamos la gráfica del mapa de etiquetas en los apartados anteriores es claramente visible un patrón circular, por ello, no vamos a poder hacer una separación entre las dos clases, etiqueta 1 y -1, con una línea recta.

Respecto a ejecutarlo con el segundo vector de características he obtenido menores errores tanto dentro como fuera, siendo estos prácticamente de 0.6 (realizando varias casos con diferentes semillas he obtenido valores aproximados a 0.6), la principal causa de esto se debe a que este segundo vector es de características no lineales y como hemos mencionado anteriormente este conjunto de datos no sigue un comportamiento lineal.