Aprendizaje Automático Trabajo 1: Programación

Alfonso García Martínez

Marzo de 2019

1. Ejercicio sobre la búsqueda iterativa de óptimos

1.- Implementar el algoritmo de gradiente descendente.
La función *gradient_descent_array* recibe como parámetros:

- El gradiente $\nabla f(x_1, x_2, ..., x_n)$ de una función $f(x_1, x_2, ..., x_n)$
- La posición de inicio dada por $(a_1, a_2, ..., a_n)$
- La tasa de aprendizaje (*learning rate*) η (0,01 por defecto)
- El número máximo de iteraciones (100000 por defecto)
- La precisión ϵ a alcanzar (10^{-20} por defecto)

La función devolverá:

- Un punto $(b_1, b_2, ..., b_n)$, en el cual el gradiente se habrá minimizado hasta tal punto en el cual, o bien se ha superado el número máximo de iteraciones, o bien la norma de la diferencia entre el punto actual y el punto explorado en la iteración anterior es menor o igual a la precisión introducida como parámetro: $\|b_{act} - b_{prev}\| \leq \epsilon$.
- El número de iteraciones ejecutadas.
- Un array con las coordenadas de todos los puntos explorados.
- Un array con los gradientes de todos los puntos explorados.

La razón por la que se ha optado por esta forma de comprobar la precisión es que el gradiente descendiente puede aplicarse a funciones cuyos mínimos locales y global no tienen por qué valer 0, cosa que sí ocurre cuando la función a minimizar es el error de un ajuste. Tendremos esto en cuenta más adelante.

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

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

El gradiente de $E(u, v)$ es fácil de calcular siguiendo la regla de la cadena al calcular las derivadas parciales. Dicho gradiente se describe mediante la expresión:

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

Donde

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

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

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)

Haciendo varias ejecuciones del algoritmo con distintos valores para el máximo de iteraciones, podemos saber cuándo un punto encontrado obtiene tal valor. Empezando, por ejemplo, con un máximo de 20 iteraciones, y aumentándolo en uno en cada ejecución del algoritmo, finalmente vemos que hacen falta como mínimo **34 iteraciones** para lograr minimizar E(u,v) a tal valor.

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

Basta con fijarnos en las coordenadas del punto que devuelve la función cuando de fijamos un máximo de 34 iteraciones, tal y como acabamos de ver en el apartado b). Concretamente, las coordenadas son $(0{,}619207678450638, 0{,}968448269010049)$.

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

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial ($x0 = 0{,}1, y0 = 0{,}1$), 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 $\eta = 0,1$, comentar las diferencias y su dependencia de η.

Volvemos a calcular analíticamente el gradiente de la función a optimizar:

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

Donde

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

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

Para $\eta = 0,01$, el mínimo alcanzado es -1,820079.

Para $\eta = 0,1$, el mínimo alcanzado es 2,048279.

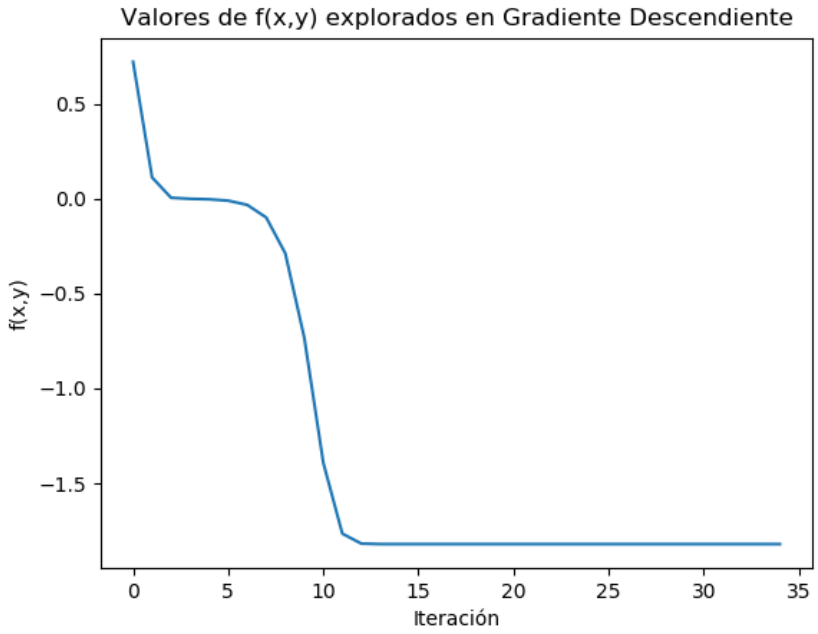


Figura 1: Evolución del mínimo de f(x,y) en cada iteración para learning rate 0,01

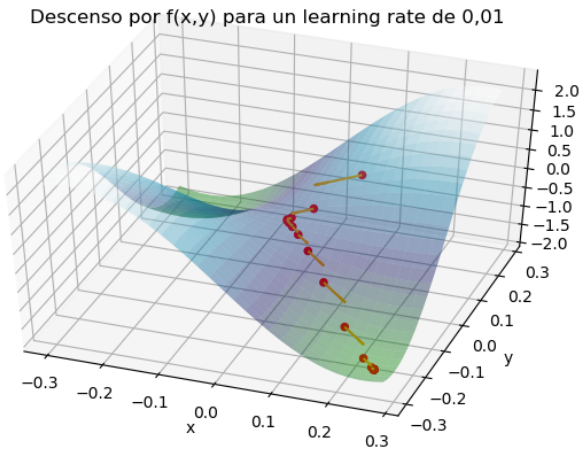


Figura 2: Descenso del gradiente en f(x,y) para learning rate 0,01

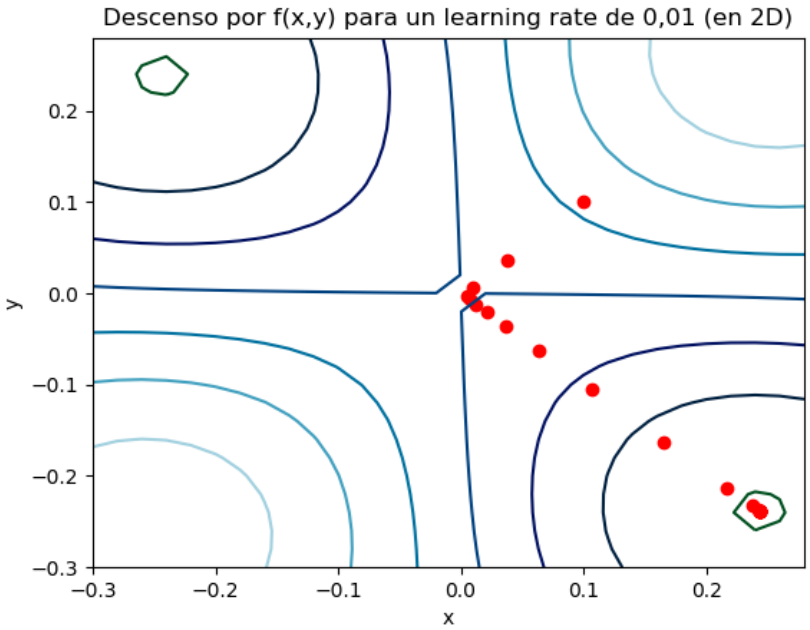


Figura 3: Descenso del gradiente con contornos de f(x,y) para learning rate 0,01

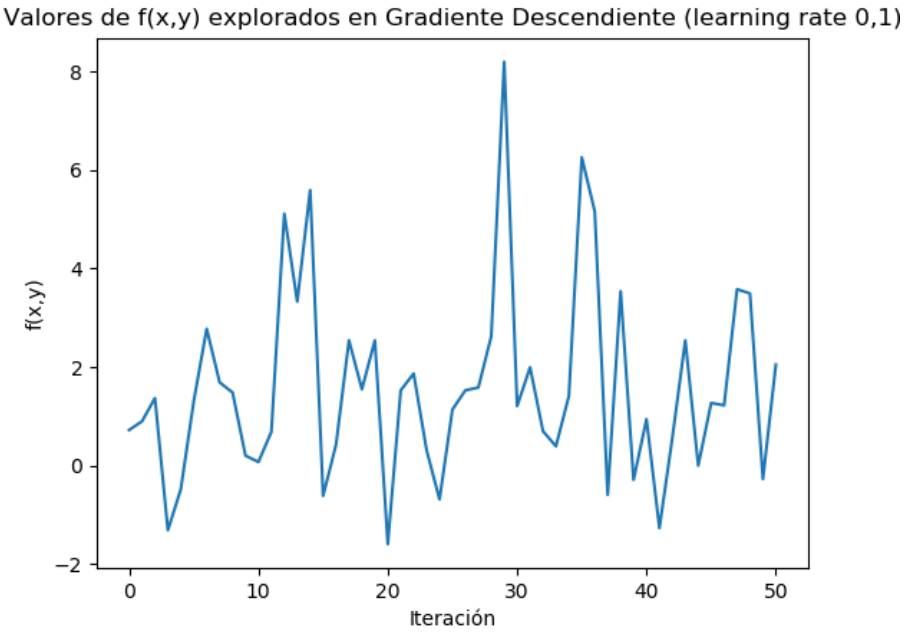


Figura 4: Evolución del mínimo de f(x,y) en cada iteración para learning rate 0,1

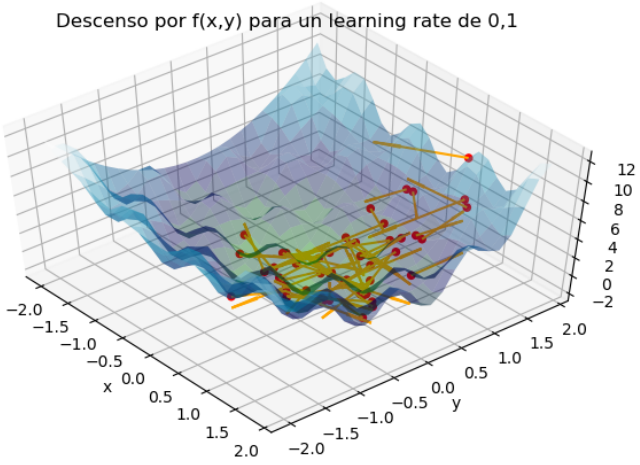


Figura 5: Descenso del gradiente en f(x,y) para learning rate 0,1

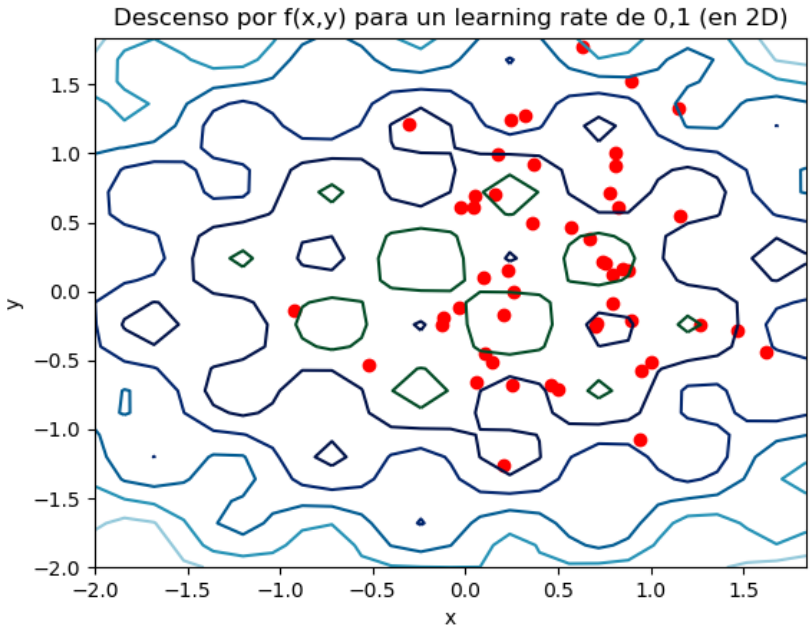


Figura 6: Descenso del gradiente con contornos de f(x,y) para learning rate 0,1

Para un $\eta = 0{,}01$, el algoritmo ha convergido rápidamente al mínimo, siguiendo la trayectoria esperada. Podríamos decir que el η en este caso está perfectamente calibrado.

Sin embargo, al subir a $\eta = 0{,}1$, los cambios de un punto a otro han sido demasiado bruscos, dando como resultado una búsqueda mucho más errática, sin seguir una trayectoria claramente definida y con un mínimo alcanzado demasiado alto (2,048279). En vista de los resultados, es lógico pensar que para esta función un η de 0,1 es demasiado alto.

b) 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.

Para un $\eta = 0,01$:

Mínimos obtenidos		
Punto de inicio	Mínimo	Coordenadas (x,y)
(0.1,0.1)	-1.820079	(0.24,-0.24)
(1.0,1.0)	0.593269	(1.22,0.71)
(-0.5,-0.5)	-1.332481	(-0.73,-0.24)
(-1.0,-1.0)	0.593269	(-1.22,-0.71)

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

Aunque es importante fijar un buen criterio de parada para buscar el equilibrio entre tiempo de cómputo y calidad de la solución, diría que saber elegir un **punto de inicio** y un **learning rate** (η) adecuados es lo más difícil además de lo más importante. Estos dos parámetros, que no sabemos cómo configurar a priori, son los que más influyen en el proceso de minimización, dando resultados totalmente diferentes al ser modificados tal y como hemos podido ver en los experimentos anteriores:

- Cuanto más alejado esté el punto de inicio del mínimo que buscamos, más tiempo puede llevar encontrarlo. Además, si la función posee diversos mínimos locales, el punto de inicio determinará en cuál de ellos acabaremos.
- El learning rate, como ya sabemos, fija en qué medida el gradiente modifica el punto actual en cada iteración. Un learning rate más alto provocará cambios mucho más bruscos, lo cual puede ser idóneo si las curvas de la función no son muy pronunciadas o si el mínimo se encuentra a gran distancia del punto de partida.

2. Ejercicio sobre regresión lineal

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 pseudoinversa 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} calcular las predicciones usando los datos del fichero de test).**

La implementación del Gradiente Descendiente Estocástico es bastante similar a la del Gradiente Descendiente convencional, pero con algunas evidentes diferencias:

- Supondremos que vamos a minimizar el error de un ajuste, cuyo gradiente habrá que calcular.
- En cada iteración se emplea un minibatch (subconjunto de datos perteneciente a un particionado aleatorio del conjunto de datos) para evaluar el gradiente del error,lo que disminuirá el tiempo de cómputo y dará más variedad al evaluar el gradiente en cada iteración. Al llegar al último minibatch, volverá al primero para seguir optimizando. Por defecto, tendrán un tamaño de 32 items.
- En este caso, la precisión sí se determina en cada iteración directamente con el valor de la función a minimizar (el error), ya que nuestro objetivo ahora es que valga 0: $Err(w) < \epsilon$

Gradiente Descendiente Estocástico

El error se define mediante la siguiente expresión:

$$Err(w) = \frac{1}{N}\sum_{n=1}^{N}(h(x_i) - y_i)^2 = \frac{1}{N}\sum_{n=1}^{N}(w^T x_i - y_i)^2$$

O en notación matricial:

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

Y cómo no, para minimizarla habrá que usar su gradiente:

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

Los resultados, para un $\eta = 0,01$, un tamaño de minibatch de 32 elementos, un máximo de 8000 iteraciones y un punto inicial generado aleatoriamente:

Resultados SGD	
Ajuste w obtenido	(1.08981587, 1.01760577, 0.49142284)
E_{in}	0.082758
E_{out}	0.138091

Método de la pseudoinversa

Esta alternativa es un cálculo totalmente analítico en el que no hay que tener en cuenta ningún parámetro más que el propio conjunto de datos.
Basta con calcular:

$$w = X^{\dagger}y$$

Donde

$$X^{\dagger} = (X^T X)^{-1} X^T$$

El inconveniente de este enfoque está en lo costoso que resulta calcular $X^{\dagger}$, pero al ser nuestro conjunto de datos de una dimensionalidad tan baja no tendremos problema en absoluto.

Resultados Pseudoinversa	
Ajuste w obtenido	(1.11588016, 1.24859546, 0.49753165)
E_{in}	0.079187
E_{out}	0.130954

Con suficientes iteraciones, el SGD obtiene resultados muy cercanos al método de la pseudoinversa.

A la hora hacer gráficas, hay que tener en cuenta que trabajamos en un espacio de 3 dimensiones (intensidad promedio, simetría y etiqueta). Por tanto, el ajuste w define un **plano** $h(x) = w_0 + w_1 x_1 + w_2 x_2$.

Para pintar los resultados en 2D, asignaremos colores a los datos según las etiquetas, y en lugar de dibujar el plano dado por w, pintaremos su recta intersección con el plano $z = 0$ (o sea, la recta en la que $h(w) = 0$), justo el punto intermedio entre -1 y 1, para poder hacernos una idea de cómo es el modelo obtenido:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

De donde podemos despejar x_2:

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

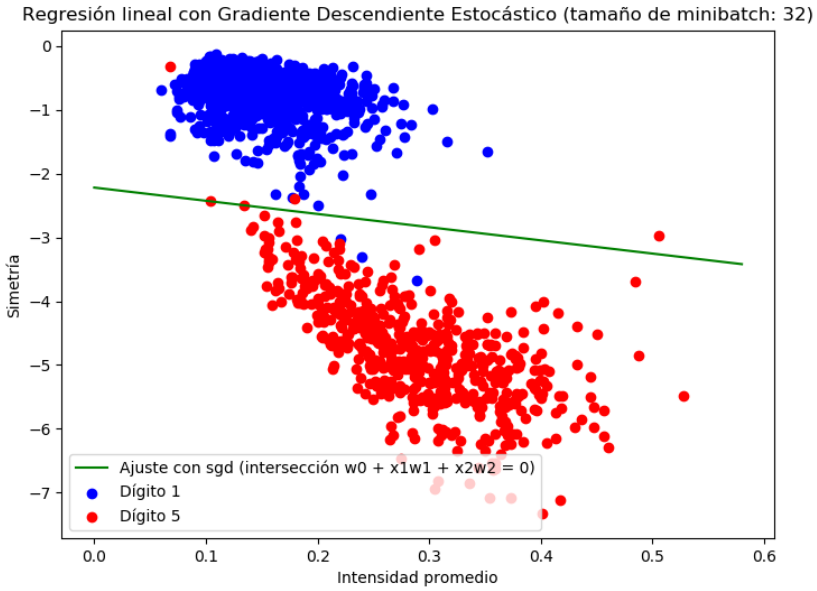


Figura 7: Regresión lineal con SGD de dígitos 1 y 5 en el conjunto de entrenamiento

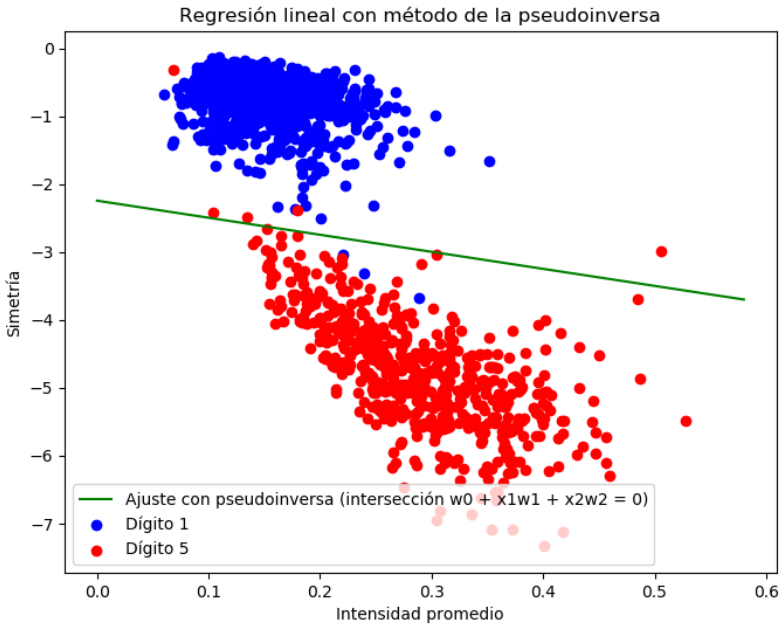
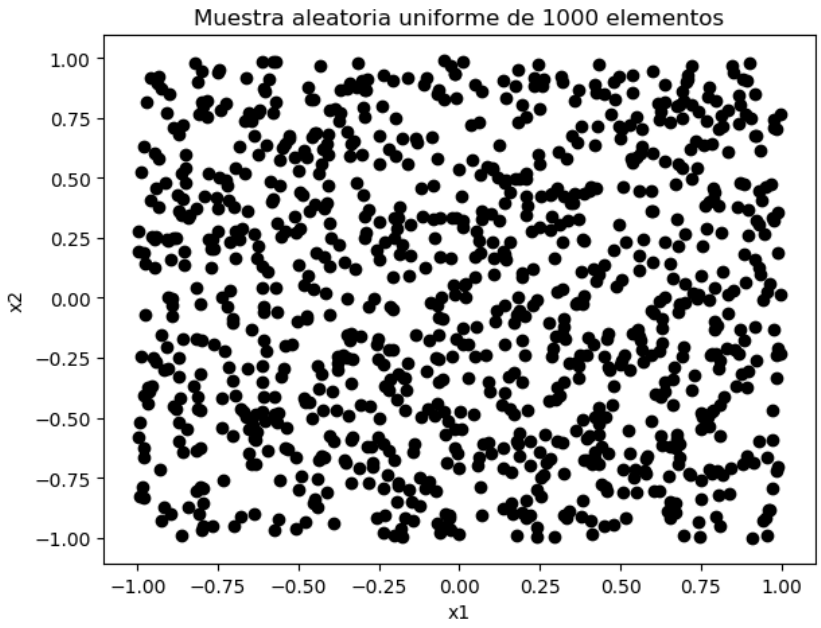


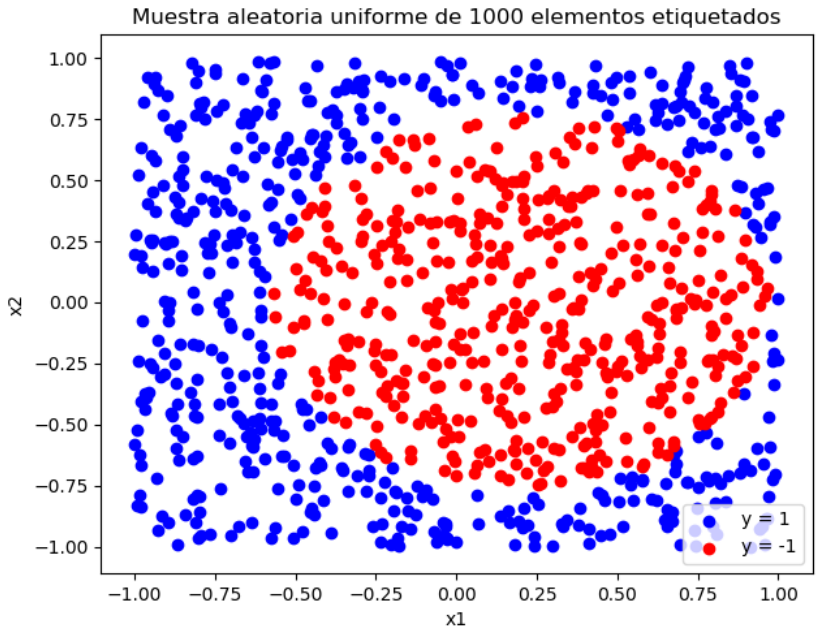
Figura 8: Regresión lineal con pseudoinversa de dígitos 1 y 5 en el conjunto de entrenamiento

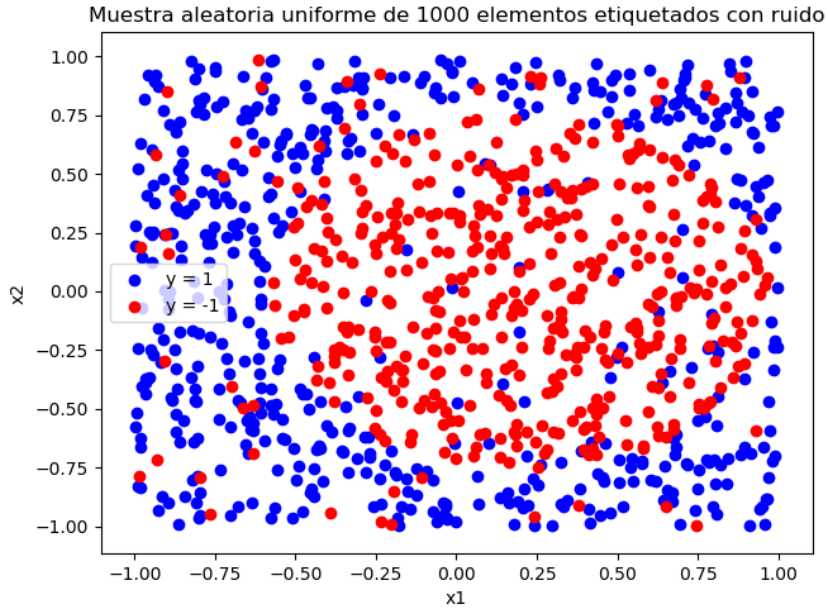
2. En este apartado exploramos como se transforman los errores E_{in} y E_{out} cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función $simula\_unif(N, 2, size)$ que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por $[-size, size] \times [-size, size]$

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


Muestra aleatoria uniforme de 1000 elementos
x2
x1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

b) 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.


Muestra aleatoria uniforme de 1000 elementos etiquetados
x2
x1
y = 1
y = -1


Muestra aleatoria uniforme de 1000 elementos etiquetados con ruido
x2
x1
y = 1
y = -1

c) Usando como vector de características $(1, x_1, x_2)$ ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste E_in usando Gradiente Descendente Estocástico (SGD).

Con

Experimento: muestra aleatoria de 1000 elementos	
Ajuste w obtenido	(-0.16466379, -0.86643772, 0.36376915)
E_{in}	1.104470

d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y

- 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.

Experimento: 1000 muestra aleatoria de 1000 elementos	
E_{in} promedio	0.971259
E_{out} promedio	0.960155

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}.

Los ajustes no han dado buenos resultados, con errores tanto dentro como fuera de la muestra cercanos al 1.

Es de esperar que al intentar una regresión **lineal** sobre un conjunto de datos que no sigue un comportamiento lineal (fijémonos en la función de etiquetado $f(x_1, x_2)$, con ambas variables elevadas al cuadrado, y tengamos también en cuenta la fuerte presencia de ruido) no obtengamos un ajuste fiable.

3. Bonus

1. Implementar el algoritmo de minimización de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio 3. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

- Generar un gráfico de cómo desciende el valor de la función con las iteraciones.
- Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento obtenida con gradiente descendente.

Ya conocemos el gradiente de $f(x, y)$. Ahora, calcularemos su matriz Hessiana:

$$Hf(x,y) = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x x^2} & \frac{\partial^2 f(x,y)}{\partial x y^2} \\ \\ \frac{\partial^2 f(x,y)}{\partial y x^2} & \frac{\partial^2 f(x,y)}{\partial y y^2} \end{bmatrix}$$

Donde:

$$\frac{\partial^2 f(x,y)}{\partial x x^2} = 2 - 8\pi^2 sin(2\pi y) sin(2\pi x)$$

$$\frac{\partial^2 f(x,y)}{\partial x y^2} = \frac{\partial^2 f(x,y)}{\partial y x^2} = 8\pi^2 cos(2\pi y) cos(2\pi x)$$

$$\frac{\partial^2 f(x,y)}{\partial x x^2} = 4 - 8\pi^2 sin(2\pi x) sin(2\pi y)$$

Según las trasparencias de teoría, la regla para actualizar el punto actual en cada iteración es la siguiente:

$$\Delta w = -H^{-1} f(x,y) \nabla f(x,y)$$

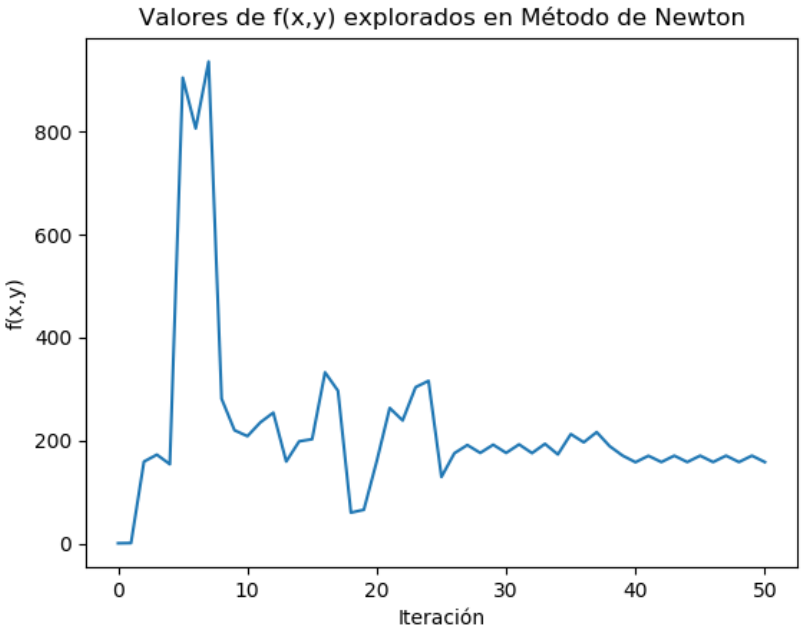
Al igual que en la pseudoinversa, el inconveniente de este método radica en tener que invertir una matriz, la Hessiana en este caso.

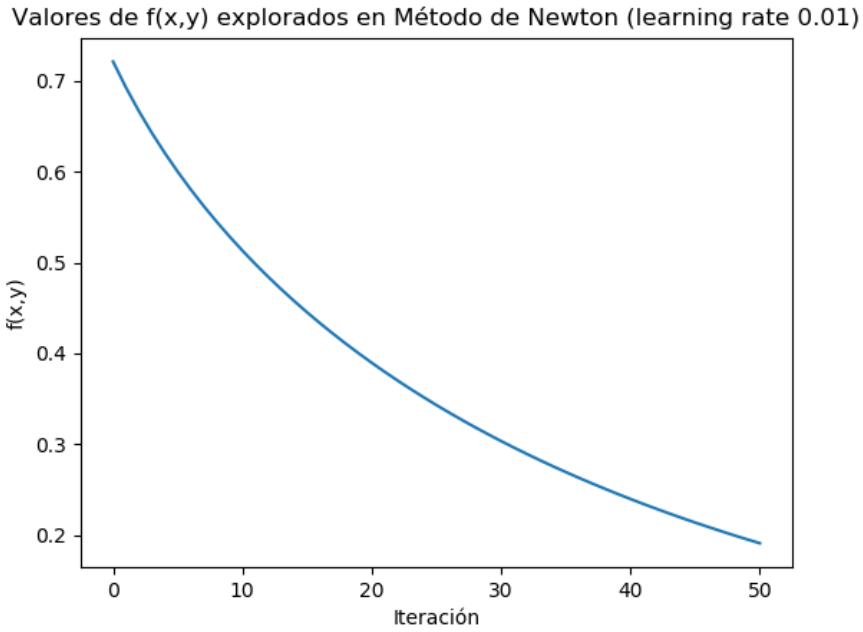
Al usar esta expresión tal cual para encontrar el óptimo, el resultado es desastroso, con un *mínimo* igual a 158,352001. Sin embargo, si incluimos de nuevo un coeficiente η:

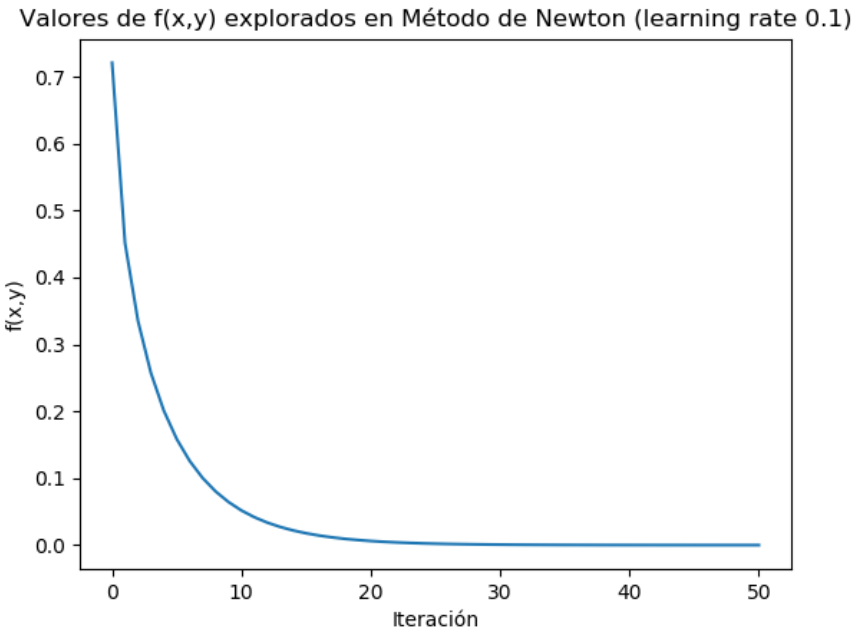
$$\Delta w = -\eta H^{-1} f(x,y) \nabla f(x,y)$$

mejoramos abismalmente los resultados.

Método de Newton $\eta = 1,0$	
Punto alzanzado obtenido	(-1.21899132, 8.80383709)
Mínimo	158.352001
Método de Newton $\eta = 0,01$	
Punto alzanzado obtenido	(0.04920161, 0.04890013)
Mínimo	0.191232
Método de Newton $\eta = 0,1$	
Punto alzanzado obtenido	(0.00036805 0.00036412)
Mínimo	0.000011


Valores de f(x,y) explorados en Método de Newton
f(x,y)
Iteración
0
200
400
600
800
0
10
20
30
40
50


Valores de f(x,y) explorados en Método de Newton (learning rate 0.01)
f(x,y)
Iteración
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Valores de f(x,y) explorados en Método de Newton (learning rate 0.1)
f(x,y)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50
Iteración

Mínimos obtenidos con método de Newton		
Punto de inicio	Mínimo	Coordenadas (x,y)
(0.1,0.1)	0.191232	(0.05,0.05)
(1.0,1.0)	2.937804	(0.98,0.99)
(-0.5,-0.5)	0.734512	(-0.49,-0.5)
(-1.0,-1.0)	2.937804	(-0.98,-0.99)

Añadir el learning rate ayuda a regular el ritmo en que se desciente por la función. Aunque la Hessiana ya cumple este papel regulando los cambios en función de la curvatura (la Hessiana es una especie de *derivada segunda*), el learning rate mejora esta regulación. Por ejemplo, en este caso, meter un $\eta = 0,1$ en el método de Newton incluso mejora los resultados del Gradiente Descendiente con esa misma tasa, ya que el primero ya incluye por defecto un mecanismo para controlar las transiciones.