1. EJERCICIO SOBRE LA BÚSQUEDA ITERATIVA DE ÓPTIMOS

1.1. Implementar el algoritmo de gradiente descendente.

Se ha implementado el algoritmo con un bucle *while* que itera mientras el valor de la función no sea menor que el mínimo buscado y el número de iteraciones actual no superen el número de iteraciones máximo.

Cada iteración se estima el nuevo *w* dado por la ecuación general $w_j := w_j - \eta \frac{\partial Ein(w)}{\partial w_j}$ y se calcula el nuevo error.

Para más información, consultar el código donde se explica línea por línea la implementación.

1.2. Considerar la función $E(u,v) = (u^3 e^{(v-2)} - 2v^2 e^{-u})^2$. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto (u, v) = (1, 1) y usando una tasa de aprendizaje η = 0.1 .

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

El gradiente de E queda definido como el vector con la derivada parcial de E respecto a u y la derivada parcial de E respecto a v.

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

$\frac{\partial E(u,v)}{\partial u} = 2(u^3 e^{(v-2)} - 2v^2 e^{-u})(3u^2 e^{(v-2)} + 2v^2 e^{-u})$

$\frac{\partial E(u,v)}{\partial v} = 2(u^3 e^{(v-2)} - 2v^2 e^{-u})(u^3 e^{(v-2)} - 4v e^{-u})$

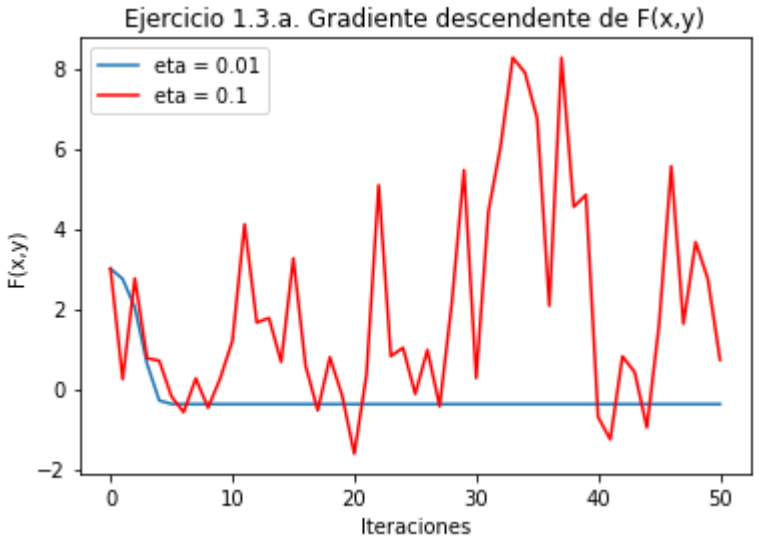
b) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u, v) inferior a 10^{-14}? 10 iteraciones.

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

(1.1572888496465497, 0.9108383657484799)

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

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial ($x_0 = -1$, $y_0 = 1$), 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 usando $\eta = 0.1$, comentar las diferencias y su dependencia de η.



Resultados:

$\eta = 0.01$ -> Coordenadas: (-1.269064351751895 , 1.2867208738332965)

Ein: -0.3812494974381

$\eta = 0.1$ -> Coordenadas: (-2.939376479816701 , 1.607795422435394)

Ein: 0.7241149424996057

Comentario

Para $\eta = 0.01$ podemos observar que, a las pocas iteraciones, F(x,y) converge a un valor que prácticamente no varía durante las 50 iteraciones, mientras que para $\eta = 0.1$ tenemos un comportamiento errático y que en muchas ocasiones empeora el mínimo encontrado.

Ninguno de los dos casos son ideales, y esto se debe a varios factores. En el caso de $\eta = 0.01$, posiblemente se deba a una de las principales deficiencias del algoritmo

descendente no estocástico, y es que tiende a mínimos locales sin posibilidad de abandonarlos.

Por otro lado, con η = 0.1 tenemos otro problema. La tasa de aprendizaje es demasiado alta para la pendiente, lo que provoca un comportamiento impreciso que suele desviarse a mínimos notablemente peores. Tendría fácil solución con la implementación de una tasa de aprendizaje dinámica, la cual variase según la pendiente en la que se encontrase el algoritmo.

b) Obtener el valor mínimo y valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija en: (-0.5, -0.5), (1, 1), (2.1, -2.1), (-3, 3), (-2, 2). Generar una tabla con los valores obtenidos. Comentar la dependencia del punto inicial.

Ejercicio 1.3.b. Para eta = 0.01 y un máximo de 50 iteraciones

	x	y	F(x,y)
(-0.5,-0.5)	-0.7934994705090673	-0.12596575869895063	9.125146662901855
(1,1)	0.6774387808772109	1.290469126542778	6.4375695988659185
(2.1,-2.1)	0.14880582855887767	-0.09606770499224294	12.490971442685037
(-3,3)	-2.7309356482481055	2.7132791261667037	-0.38124949743809955
(-2,2)	-2.0	2.0	-4.799231304517944e-31

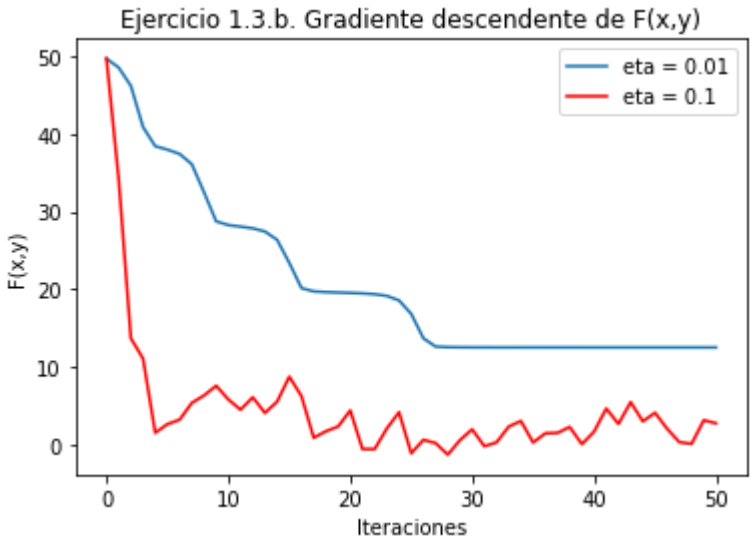
Esta tabla evidencia lo mucho que depende el algoritmo del punto inicial. Mientras que para (-2, 2) inmediatamente alcanzamos un F(x,y) menor que 10^{-14}, para el resto de puntos se consumen las 50 iteraciones sin alcanzar mínimos decentes. En algunos casos, estos mínimos son de hecho extremadamente malos, como para (2.1, -2.1) donde obtenemos 12.

Ejercicio 1.3.b. Para eta = 0.1 y un máximo de 50 iteraciones

	x	y	F(x,y)
(-0.5,-0.5)	-0.414320171517534	1.4015138347414056	2.635939075253365
(1,1)	-4.227819739304707	2.6928375389338943	7.777458402540317
(2.1,-2.1)	-1.8529328639793632	1.1320463260277653	2.7057723725975418
(-3,3)	-2.3225631551842083	2.521107236036219	0.8846027295504482
(-2,2)	-2.0	2.0	-4.799231304517944e-31

Si probamos a cambiar la tasa de aprendizaje a 0.1, podemos observar como los tres primeros puntos se favorecen de ello mientras que (-3, 3) empeora.

Considero por tanto que esta dependencia tan explícita de los puntos iniciales se debe, nuevamente, a tener una tasa de aprendizaje estática. Aunque el algoritmo de gradiente descendente siempre vaya a depender en cierta medida del punto inicial, esta dependencia no sería tan rotunda con una tasa de aprendizaje dinámica.



Si hacemos una gráfica como la que hicimos para el apartado anterior pero tomando como punto inicial (2.1, -2.1) —que era el punto que peor mínimo nos había dado en la primera tabla—, se hace evidente cómo el algoritmo se hubiera beneficiado de una tasa de aprendizaje más alta como $\eta = 0.1$ en las primeras iteraciones, y otra más pequeña como $\eta = 0.01$ para el resto.

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

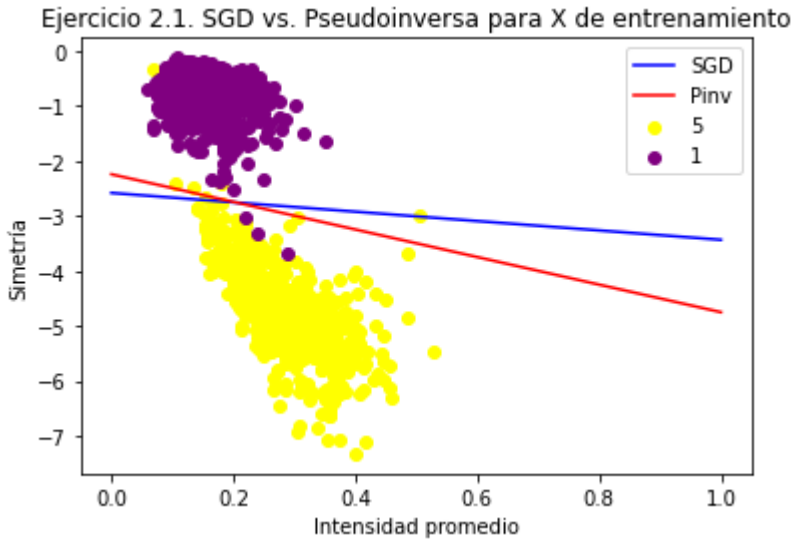
El algoritmo de gradiente descendente es muy dependiente de ambos la tasa de aprendizaje y el punto inicial escogido, siendo incluso determinantes para poder alcanzar una solución aceptable. Aún mejorándolo con una tasa de aprendizaje dinámica, el algoritmo podría converger a un mínimo local deficiente sin posibilidad de salir de éste (lo que se conoce como punto de silla). Es por ello que necesitamos técnicas más sofisticadas, como el gradiente descendente estocástico.

2. EJERCICIO SOBRE REGRESIÓN LINEAL

2.1. Estimar un modelo de regresión lineal a partir de los datos proporcionados por los vectores de características (Intensidad promedio, Simetría) usando tanto el algoritmo de la pseudo-inversa como el gradiente descendente estocástico (SGD). Las etiquetas serán {–1,1}, una por 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}.

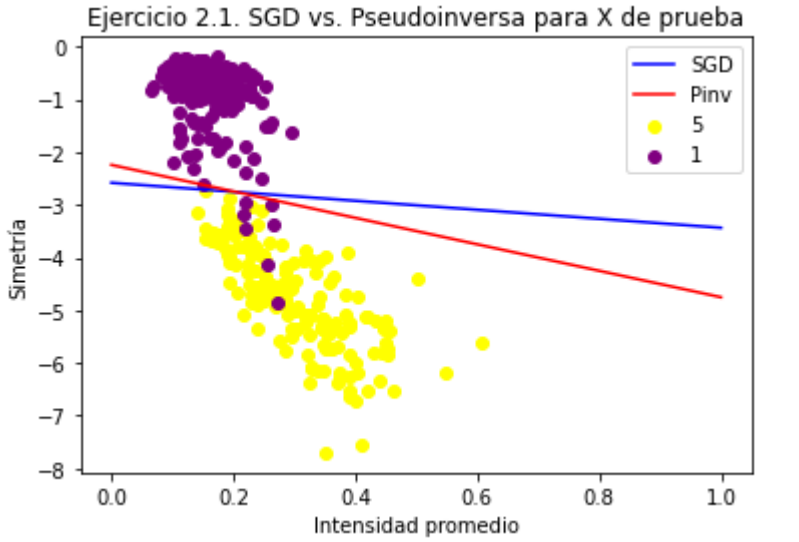
El gradiente descendente estocástico decidí implementarlo con un primer bucle *while* donde, mientras el error no fuese menor que el mínimo buscado y no se superasen el máximo de iteraciones, barajase las muestras, las dividiese en *n* minibatches dado un tamaño de minibatch y las recorriera con un bucle *for* anidado calculando un *w* nuevo para cada minibatch. La intención de esto era que se tuviesen en cuenta todos los minibatches en cada iteración a la hora de calcular el *w*.
Anidar los bucles al revés (siendo *while* el bucle anidado al *for* que recorre el vector de minibatches) no era una opción porque tenía varias implicaciones que habrían empeorado el funcionamiento del algoritmo. Principalmente: que cada minibatch solo se habría tenido en cuenta una vez por ejecución, y que se estaría calculando un *w* durante *j* iteraciones con información incompleta (cada minibatch).
La pseudo-inversa fue implementada simplemente utilizando funciones de la biblioteca numpy. Para más información, consultar el código.

Estos fueron los resultados:



SGD: Ein = 0.08091535844121585

Pseudo-inversa: Ein = 0.07918658628900396



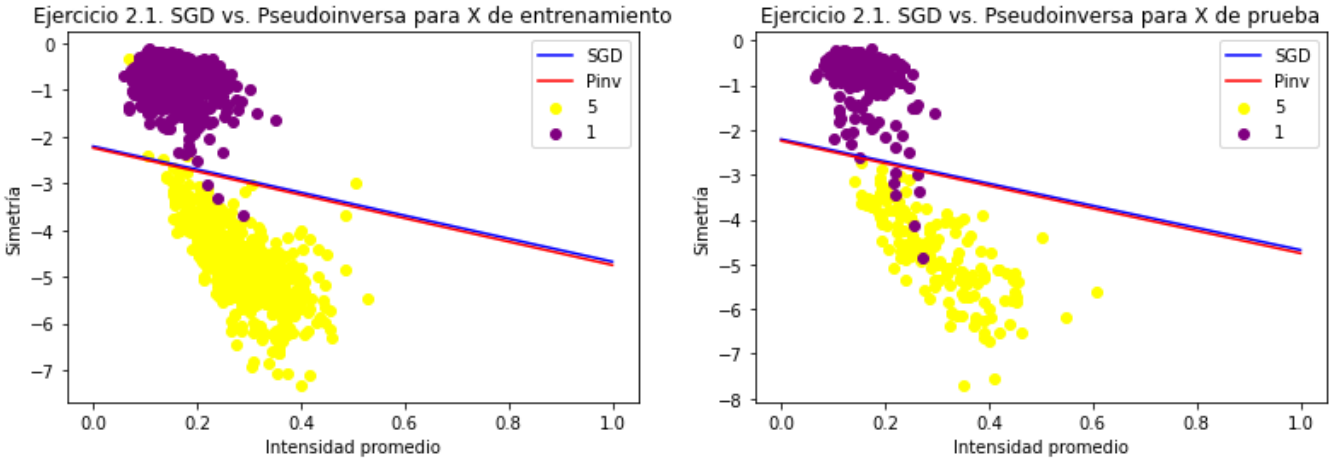
SGD: Eout = 0.13538561549194933

Pseudo-inversa: Eout = 0.13095383720052578

Numéricamente los errores obtenidos son bastante parecidos (y relativamente buenos) para ambas las muestras de entrenamiento y las de prueba. Sin embargo, en la gráfica queda más claro que la recta obtenida por la pseudo-inversa es mejor.

Para el SGD he escogido una tasa de aprendizaje del 0.01, un error mínimo de 10^{-14}, 100 iteraciones de máximo y un tamaño de minibatch de 24. Cambiar la tasa de

aprendizaje no suponía mejorar su rendimiento (para η=0.001 obteníamos un Ein muy similar al anterior y para η=0.1 subía a 0.5), y el error mínimo era irrelevante porque no estaba ni cerca de alcanzarlo en ningún caso. Sin embargo, subir las iteraciones y disminuir el tamaño del minibatch sí que aproximaba el error de SGD al de la pseudo-inversa, como podemos observar en las siguientes gráficas.

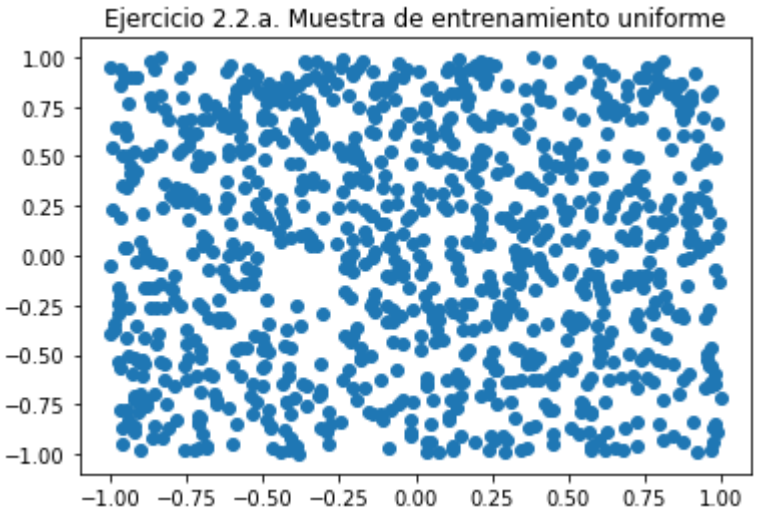


(máximo 1000 iteraciones, 10 de tamaño de minibatch)

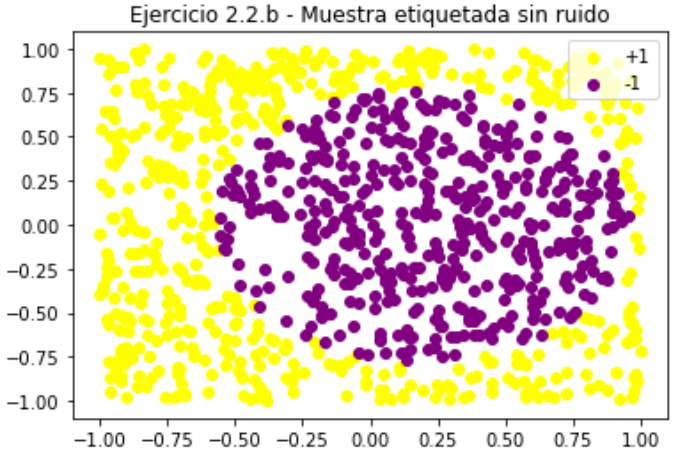
Sin embargo, supone un coste computacional considerablemente mayor, al pasar de obtener los datos inmediatamente a tener que esperar varios segundos. Además, el error sigue siendo (ligeramente) peor que el de la pseudo-inversa. ¿Es entonces la pseudo-inversa simplemente mejor?

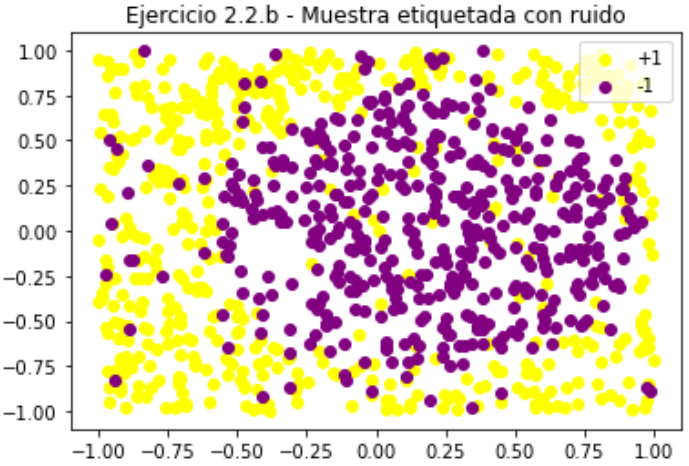
No. La pseudo-inversa ofrece una estimación de *w* de manera analítica, sin iteraciones, con solución única (determinista) y sin necesidad de ajustar parámetros. Pero tiene un gran problema, y es que tiene una complejidad de n^3, siendo *n* el número de características. En este experimento teníamos dos características (intensidad promedio y simetría) y por tanto esa complejidad no era relevante (2^3 = 8), pero si por ejemplo fuésemos a estimar un *w* para imágenes pequeñas de 100x100 píxeles, estaríamos hablando de $(100*100)^3$ = 1000000000000 de complejidad. Es por esto que el rendimiento de la pseudoinversa en este experimento en concreto no eclipsa al SGD como potencial algoritmo para estimar un modelo de regresión lineal.

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



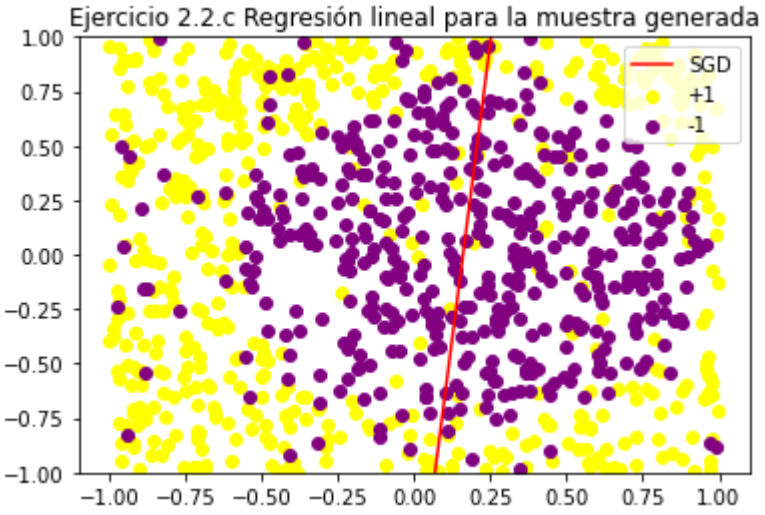
b) Consideremos la función $f(x_1, x_2) = \text{sign}((x_1 - 0.2)^2 + x_2^2 - 0.6)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10 % de las mismas. Pintar el mapa de etiquetas obtenido.





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).

Para este apartado se han escogido los mismos parámetros en SGD que en el ejercicio 2.1. (η= 0.01, ε= 10^{-14}, máximo 100 iteraciones, 24 de tamaño de minibatch) ya que cambiarlos no suponía una diferencia significativa en el Ein final obtenido.



Ein: 0.9228746729114184

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.**

  Ein medio: 0.9231914934318881

- **Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E_{out} en todas las iteraciones.**

  Eout medio: 0.9272833069696086

Este experimento se hizo en cada caso con los mismos parámetros del apartado anterior pero con un máximo de 10 iteraciones de SGD, ya que incrementarlo

ralentizaba considerablemente la ejecución sin mejorar de manera significativa los errores obtenidos.

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 errores (0.923 y 0.927 respectivamente) no son buenos, y es que no pueden serlo. En la gráfica del apartado anterior ya se hizo evidente que un modelo lineal no podía, en ningún caso, separar correctamente la muestra. Y esto se debe a la naturaleza de los datos, donde el grupo etiquetado como *negativo* se halla en mayor medida concentrado en un círculo mientras que el grupo de *positivos* se encuentra rodeando al primer grupo. Es sencillamente imposible que con una recta podamos separar esos datos y predecir con fiabilidad a qué grupo pertenece cada punto.

▪ Repetir el experimento anterior pero usando características no lineales. Ahora usaremos el siguiente vector de características: $\phi_2(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ .Ajustar el nuevo modelo de regresión lineal y calcular el nuevo vector de pesos $\hat{w}$. Calcular los errores promedio de E_{in}y E_{out}.

(η= 0.01, ε= 10^{-14}, máximo 10 iteraciones SGD, 24 de tamaño de minibatch.)
Ein medio: 0.6466000046730609
Eout medio: 0.6664514159525001

(η= 0.01, ε= 10^{-14}, máximo 25 iteraciones SGD, 24 de tamaño de minibatch.)
Ein medio: 0.5527627789496238
Eout medio: 0.5800480070492928

(η= 0.01, ε= 10^{-14}, máximo 100 iteraciones SGD, 24 de tamaño de minibatch.)
Ein medio: 0.5335838214545993
Eout medio: 0.5659265622900116

(η= 0.01, ε= 10^{-14}, máximo 25 iteraciones SGD, 10 de tamaño de minibatch.)
Ein medio: 0.5345595030960857
Eout medio: 0.5665478195825935

▪ A la vista de los resultados promedios de E_{in} y E_{out} obtenidos en los dos experimentos. ¿Qué modelo considera que es el más adecuado? Justifique la decisión.

Como es obvio, el modelo no lineal es un predictor más potente para este problema. Los errores de entrada y salida son casi la mitad de los que obteníamos para el vector de características lineal, y además podemos reducirlo hasta 0.5 aumentando el número de iteraciones máximo o disminuyendo el tamaño por minibatch.

Por ello, y aunque el error aún esté bastante lejos de alcanzar el mínimo de 10^{-14}, considero que el modelo no lineal es el más adecuado entre los dos para este problema.

BONUS. MÉTODO DE NEWTON

BONUS.1. Implementar el algoritmo de minimización de Newton y aplicarlo a la función f(x, y) dada en el ejercicio 1.3. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

El método de Newton decidí implementarlo como el algoritmo de gradiente descendente pero estimando *w* dada la ecuación $\Delta w = - H^{-1} \nabla f(w_0)$, donde H es la matriz hessiana y ∇f es el gradiente de la función.

La matriz hessiana (en este caso que tenemos dos variables) viene dada por:

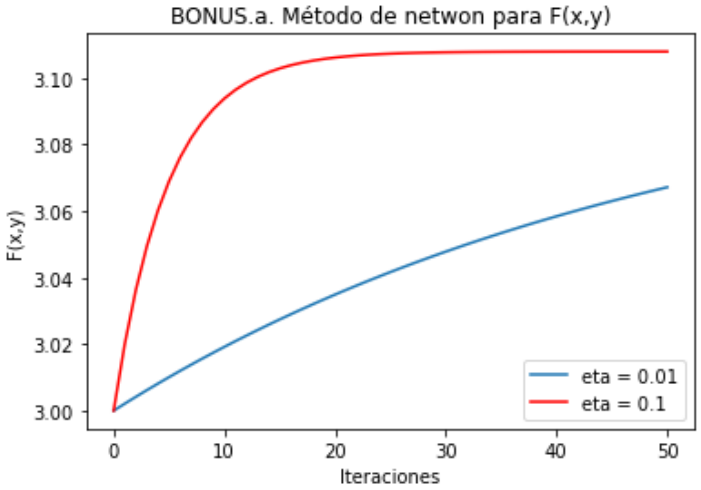
$$H_x = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

Es decir, la matriz que incluye la derivada segunda de f respecto de x_1, la derivada de f respecto de x_1 y x_2, la derivada segunda de f respecto de x_2 y la derivada de f respecto de x_2 y x_1.

Además, también decidir aplicar una tasa de aprendizaje estática para mejorar los resultados, quedando la ecuación de estimación así: $w_j := w_j - \eta H^{-1} \nabla f(w)$.

Para más información, consultar el código.

▪ Generar un gráfico de cómo desciende el valor de la función con las iteraciones.



Para η = 0.01

Número de iteraciones: 50

Coordenadas obtenidas: (-0.9793498427941949 , 0.9893767407575305)

Ein: 3.0671859515488302

Para η = 0.1

Número de iteraciones: 50

Coordenadas obtenidas: (-0.9463274028190676 , 0.9717082373563009)

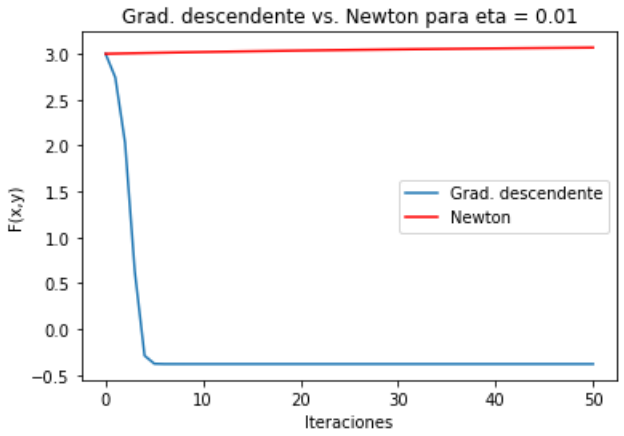
Ein: 3.1079767229661335

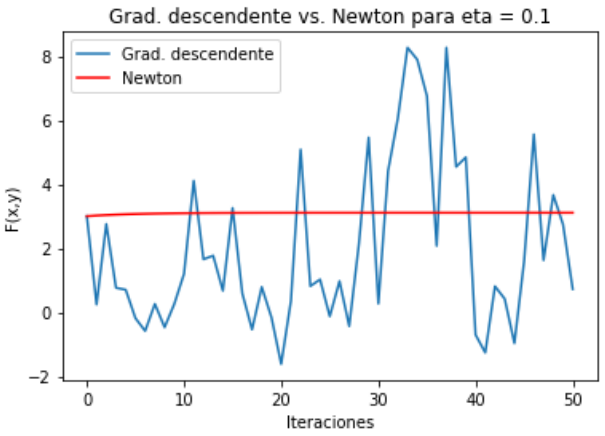
▪ Obtener el valor mínimo y valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija en: (-0.5, -0.5), (1, 1), (2.1, -2.1), (-3, 3), (-2, 2). Generar una tabla con los valores obtenidos.

Ejercicio BONUS.b. Para eta = 0.01 y un máximo de 50 iteraciones

	x	y	F(x,y)
(-0.5,-0.5)	-0.4479020699080101	-0.5172536371220581	15.012564277395944
(1,1)	1.0220605138235894	0.9689440101380584	11.205423036736665
(2.1,-2.1)	2.1175989871977006	-2.0777457395556844	49.57852918928229
(-3,3)	-3.0206501572058055	3.010623259242468	3.067185951548828
(-2,2)	-2.0	2.0	-4.799231304517944e-31

▪ **Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.**





Como queda de manifiesto tras ver las gráficas y los resultados obtenidos, el método de Newton no solo nos da resultados pobres en comparación con el gradiente descendente sino que también empeora conforme avanzan las iteraciones.

Esto se debe a que el método de Newton es un método abierto, es decir, que no está garantizada su convergencia al mínimo global. Para alcanzar esa convergencia se debe seleccionar un punto inicial lo suficientemente cercano a la raíz buscada, empezando las iteraciones por un valor considerablemente cercano a 0 denominado punto de arranque o valor supuesto. Esto hace al método de Newton un algoritmo extremadamente dependiente del punto inicial.

Además, el método de Newton solo garantiza alcanzar el mínimo global si (entre otras condiciones) las derivadas segundas f ''(x) y f ''(y) son siempre mayores que 0, y en este caso no se cumple.

Es por estos motivos que el método de Newton posiblemente no sea el mejor candidato para minimizar el f(x,y) dado en el ejercicio 1.3.