Doble Grado en Ingeniería Informática y Matemáticas

APRENDIZAJE AUTOMÁTICO

Práctica 1: Programación

UNIVERSIDAD DE GRANADA

Alberto Estepa Fernández - *albertoestep@correo.ugr.es*

Marzo de 2020

Índice

1. Ejercicio sobre la búsqueda iterativa de óptimos: Gradiente descendente

Hemos usado la plantilla proporcionada para la realización del ejercicio, modificando alguna parte si fuese necesario.

1.1. Apartado 1 (1 punto)

Implementar el algoritmo de gradiente descendente.

Para este apartado hemos creado la función *gd* al que le proporcionamos los parametros siguientes:

- w: Es el punto inicial de dos coordenadas.
- lr: Es el parámetro de "learning rate" o tasa de aprendizaje.
- $grad\_fun$: Es la función gradiente de la función fun.
- fun: Es la función de la cual queremos aplicar el gradiente descendente.
- *epsilon*: Error.
- max_iters: Es el número máximo de iteraciones a aplicar.

```
def gd(w, lr, grad_fun, fun, epsilon, max_iters = 1000):
        it = 0
        while it < max_iters:
                it += 1
                w = w - lr * grad_fun(w)
                if fun(w) < epsilon:
                        break
        return w, it
```

1.2. Apartado 2 (2 puntos)

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 $\eta = 0,1$.

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

   La expresión del gradiente de la función $E(u,v)$ es:

$$\nabla E(u,v) = (2e^{-2u}(ue^{u+v} - 2v)(e^{u+v} + 2v), 2e^{-2u}(ue^{u+v} - 2)(ve^{u+v} - 2v))$$

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)**

   Tarda 10 iteraciones el algortimo en obtener por primera vez un valor inferior a dicho valor.

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

   Se alcanzó dicho valor en $(0.044736290397782055, 0.023958714099141746)$.

1.3. Apartado 3 (2 puntos)

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

1. **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 pero usando $\eta = 0,1$, comentar las diferencias y su dependencia de η.**

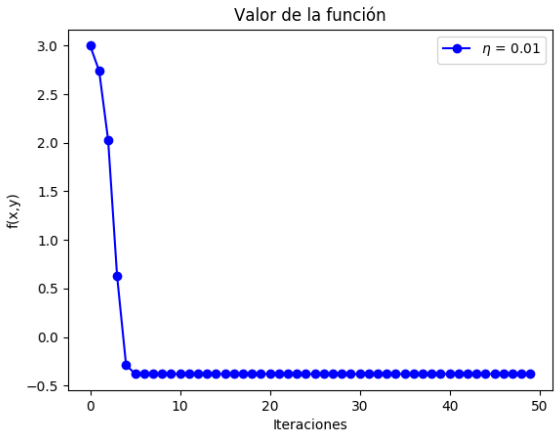

Figura 1: $\eta = 0.01$

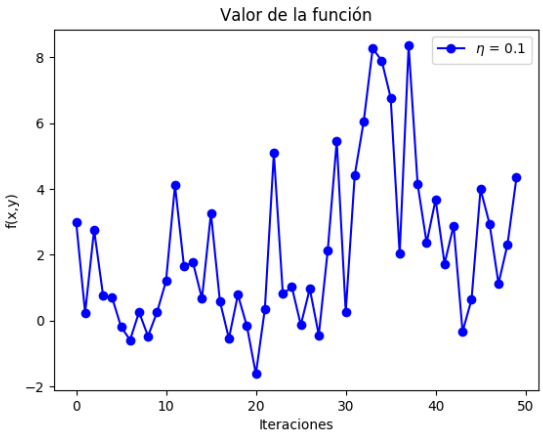

Figura 2: $\eta = 0.1$

Vemos que al aumentar el valor de η, no se llega a alcanzar el mínimo, pues la función oscila entre valores pequeños y grandes. Así éste es un claro ejemplo de la fuerte dependencia que tiene el método del gradiente descendente del parámetro η: como hemos visto para un mismo problema puede converger al mínimo o no hacerlo dependiendo del valor que tome dicho parámetro.

2. **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**

Puntos Iniciales	Coordenadas del mínimo	Valor de la función
(2.1, -2.1)	(2.243804968769898, -2.2379258212801045)	-1.8200785415471563
(3, -3)	(2.7309356482482143, -2.713279126166773)	-0.3812494974381009
(1.5, 1.5)	(1.7791194985761598, 1.0309456416573834)	18.04207234931203
(1, -1)	(1.269064351713527, -1.2867208738084945)	-0.3812494974381009

Para obtener los resultados de esta tabla hemos usado un valor de learning rate o $\eta = 0.01$. Además hemos fijado el máximo de iteraciones en 1000.

Hay que ser consciente que los valores mínimos dependen del valor de learning rate. Para otro valor de η puede ser que no obtengamos el mismo mínimo local o incluso que no converja si el valor de η es suficientemente grande.

Por ejemplo, veamos que ocurre con un valor de $\eta = 0.1$:

Puntos Iniciales	Coordenadas del mínimo	Valor de la función
(2.1, -2.1)	(1.7677264443737795, -1.741826691264284)	-1.7977293626051805
(3, -3)	(2.2233378559755197, -2.240674017658855)	-1.8028892303454023
(1.5, 1.5)	(1.7891456578350042, -1.7537362941795278)	-1.7735226462427427
(1, -1)	(2.263513398762896, -2.2665981874276566)	-1.7703793038589763

Cabe aclarar que éstos valores son los mínimos de las 1000 iteraciónes que hemos realizado, no el último valor del algoritmo (que idealmente sería el mínimo si convergiese).

En el código implementado hemos comprobado que con un valor de $\eta = 0.01$ los valores mínimos y los devueltos por el algoritmo (última iteración) coinciden salvo un error ínfimo en algunos casos. Sin embargo si $\eta = 0.1$ éstos valores no coindicen, por lo que podemos afirmar que no converge el algoritmo para éste valor de η en ésta función y dichos valores iniciales.

1.4. Apartado 4 (2 puntos)

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

Como hemos visto, al aplicar el algoritmo sobre una misma función, encontrar un mínimo local depende de muchos factores, como una buena elección del punto inicial, una buena elección del valor de learning rate (aunque si disponemos de una buena capacidad de computo, nos valdria con coger un valor bastante pequeño para evitar que oscile, pero ésto haría que la solución fuese el mínimo local más cercano y no el mínimo global).

Destacamos que para encontrar el mínimo global la función debe ser derivable. Además si ésta función posee más de un mínimo local, éste algoritmo del gradiente descendente puede estancarse en algún mínimo local que no sea global. Ésto no ocurriría si la función fuese convexa, ya que solo tendríamos un mínimo local que sería global.

Así vemos que es dificultoso encontrar el mínimo global de la función mediante el algoritmo delgradiente descendente y depende de varios factores que dependen tanto de la función en sí como de los parámetros que usamos para buscarlo (learning rate y punto inicial).

2. Ejercicio sobre Regresión Lineal

Este ejercicio ajusta modelos de regresión a vectores de características extraidos de imágenes de dígitos manuscritos. En particular se extraen dos caracterísctcas concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

2.1. Apartado 1 (2.5 puntos)

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). (Usar $Regress\_Lin(datos, label)$ como llamada para la función (opcional)).

SGD

Para éste ejercicio hemos mantenido los parámetros del anterior ejercicio: $\eta = 0.01$ y 1000 iteraciones. Además el tamaño de minibatch lo hemos fijado en 32. Incluimos el gráfico de dispersión junto a la recta de ajuste de los datos a continuación:

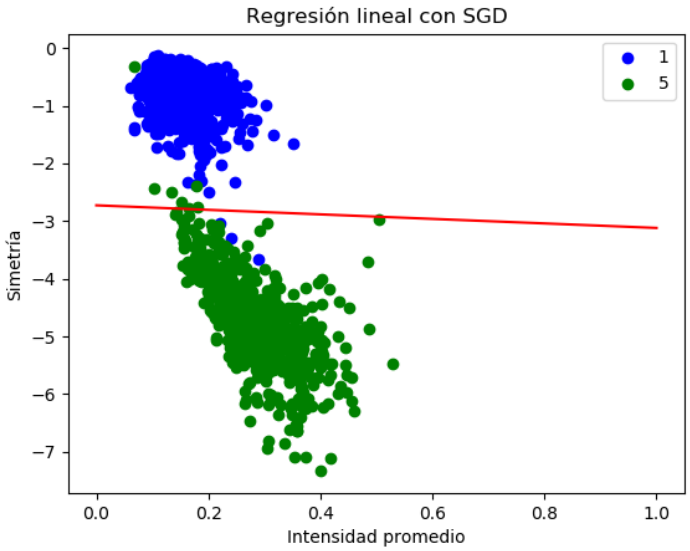

Figura 3: $SGD : \eta = 0.01, minibatch = 32$ *e* $iteraciones = 1000$

Para realizar el cálculo de la recta de ajuste hemos usado la ecuación:

$$0 = w_0 + w_1 x_1 + w_2 x_2 \tag{1}$$

donde tenemos que despejar x_2 sabiendo que $x_1 = 0$ para la primera coordenada y $x_1 = 1$ para la segunda por ser $[-1, 1]$ el intervalo por donde se mueve la primera coordenada de la recta.

Otro tema que queríamos aclarar es que llamamos iteración a cada actualización de w. En nuestro caso barajamos la lista de índices y obtenemos un minibatch de tamaño 32 con los primeros índices y a partir de él actualizamos w y completaríamos una iteración. La siguiente iteración partiríamos de los índices siguientes y obtendríamos el minibatch correspondiente para actualizar w. Así seguiríamos hasta que los siguientes índices sean superiores al límite (habremos completado una época del proceso), en éste caso volveríamos a barajar la lista de índices y obtendríamos los 32 primeros. Este proceso se repite hasta terminar las iteraciones precisas. Así aclaramos que como no se especifica, hemos considerado que iteraciones no es lo mismo que épocas.

Podemos observar que obtenemos unos resultados bastante buenos, aunque hay datos que no están situados en el lugar que les corresponde con respecto a la línea de separación. Pero es obvio, viendo el gráfico de dispersión, que es imposible que alguna recta separe completamente los datos, por lo que los datos no son linealmente separables.

Hemos calculado también los valores de bondad del resultado obtenido usando E_{in} y E_{out}:

$$E_{in} : 0.08165353324023997 \quad E_{out} : 0.13495529497627995$$

Estos datos nos confirman que el ajuste de los datos es bastante notable, ya que el error de la muestra es bastante pequeño y el error de los datos de test, aunque es un poco mayor, podemos considerarlo bueno, sabiendo que es imposible separar los datos de forma completa mediante una recta.

Pseudoinversa

Manteniendo los mismos parámetros, incluimos el gráfico de dispersión junto a la recta de ajuste de los datos que hemos obtenido mediante el algoritmo de la pseudoinversa:

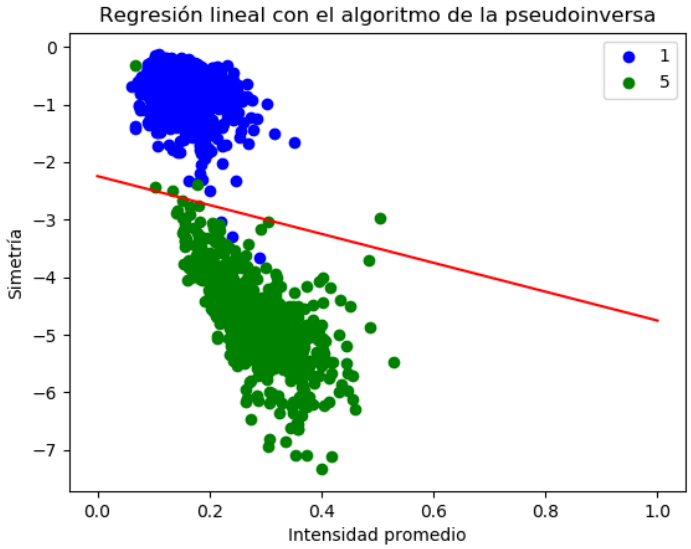

Figura 4: $SGD : \eta = 0.01, minibatch = 32$ *e* $iteraciones = 1000$

Observamos que al igual que pasaba con el método de SGD obtenemos unos resultados bastante buenos aunque, obviamente, tenemos el mismo problema: hay datos que no están situados en el lugar que les corresponde con respecto a la línea de separación. La recta con respecto al método de SGD ha variado ligeramente.

Los valores de bondad del resultado obtenido usando E_{in} y E_{out} son los siguientes:

$$E_{in} : 0.0.07918658628900395 \quad E_{out} : 0.0.1309538372005258$$

Vemos que han disminuido con respecto al algoritmo SGD pero de forma ínfima. Ésto indica que ambos métodos nos han proporcionado una buena separación de los datos y para éste ejemplo con los parámetros impuestos, el algoritmo de la pseudoinversa es ligeramente mejor.

2.2. Apartado 2 (3 puntos)

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 $[-size, size]x[-size, size]$

Experimento 1

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

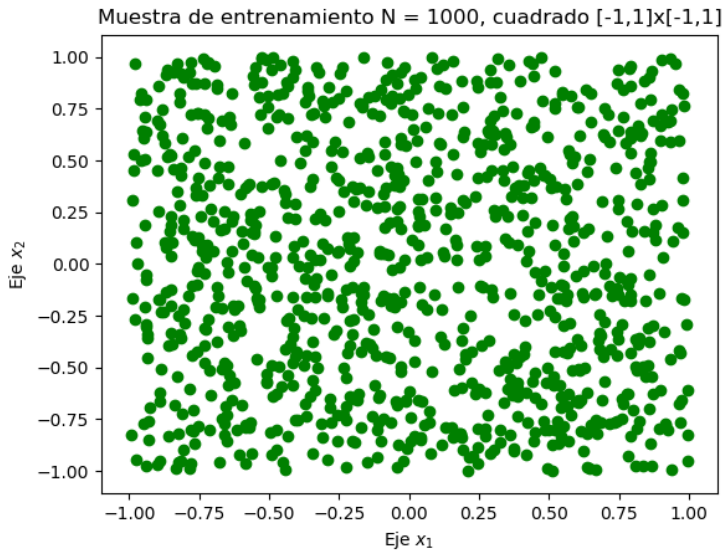

Figura 5: $Muestra\ aleatoria\ uniforme : N = 1000$

Aclaración: No hemos incluido leyenda en éste gráfico porque no hay nada que poner en ella.

Para la realización de éste ejercicio hemos usado la función $simula\_unif$. Dicho método usa la función $uniform$ de $numpy.random$ para generar los 1000 datos.

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.

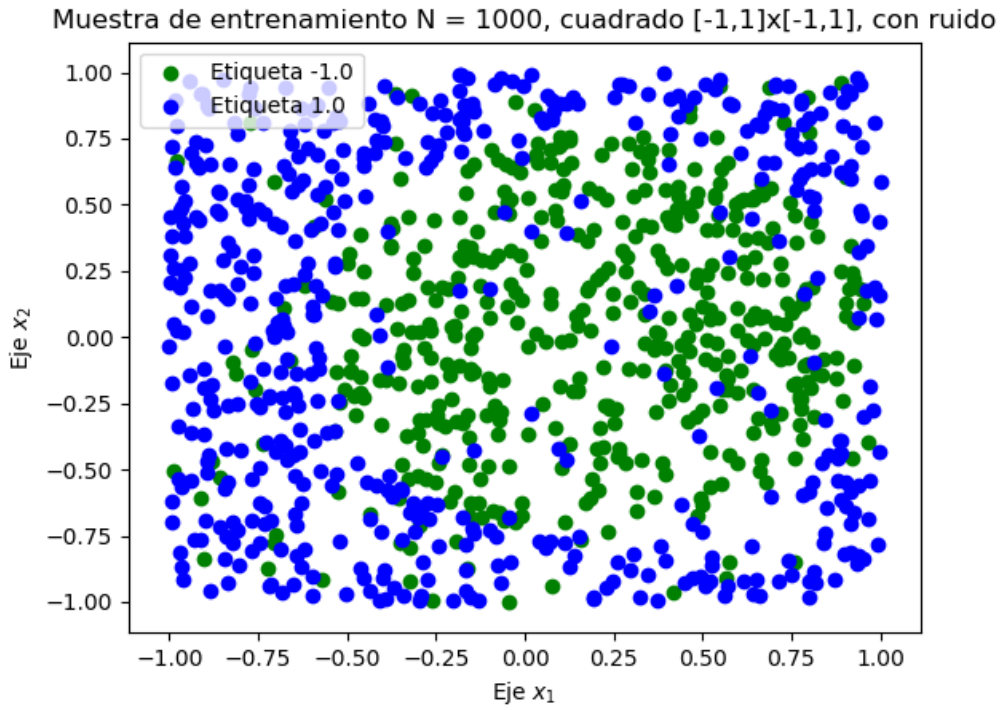

Figura 6: *Muestra aleatoria uniformeconruido* : $N = 1000$

Para la obtención de dichos resultados hemos implementado dos funciones:

1. *asigna_etiquetas*: que devuelve la evaluación de la función f en cada punto.
2. *introduce_ruido*: que intercambia el signo de un 10 % de las etiquetas pasadas.

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).

Para éste ejercicio hemos usado la función *sgd* que habíamos creado en el apartado anterior. Los parámetros son los mismos que hemos usado durante toda la práctica $\eta = 0.01$ y el número de iteraciones = 1000.

Para realizar el cálculo de la recta de ajuste hemos usado la ecuación:

$$0 = w_0 + w_1 x_1 + w_2 x_2 \tag{2}$$

donde tenemos que despejar x_2 sabiendo que $x_1 = -1$ para la primera coordenada y $x_1 = 1$ para la segunda por ser $[-1, 1]$ el intervalo por donde se mueve la primera coordenada de la recta.

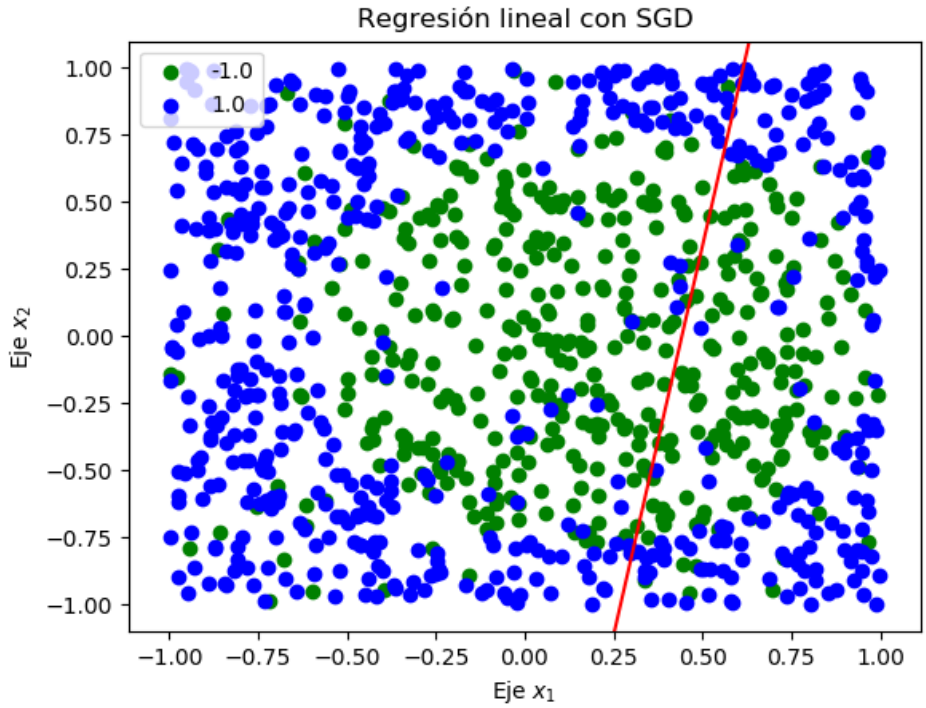

Figura 7: Regresión lineal con SGD

Vemos que el ajuste de la recta es bastante malo y ésto mismo nos podemos decir con los resultados del error E_{in} que hemos obtenido:

$$E_{in} : 0.9313543282025132$$

Como vemos el error es bastante alto por lo que concluimos que no se produce un ajuste buenos de los datos.

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. Además 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.

Los resultados obtenidos son los siguientes:

$$E_{in}media : 0.9276219012630318 \quad E_{out}media : 0.932623875409464$$

Las valoraciones están incluidas en el apartado siguiente.

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 resultados obtenidos son bastante malos. Los valores de errores son bastante altos. Como se puede apreciar en la representación de un experimento (7), los datos no permiten una separación lineal. A continuación estudiaremos otro tipo de ajustes no lineales.

Experimento 2

Repetir el mismo 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} .

Para una ejecución hemos obtenido el siguiente gráfico:

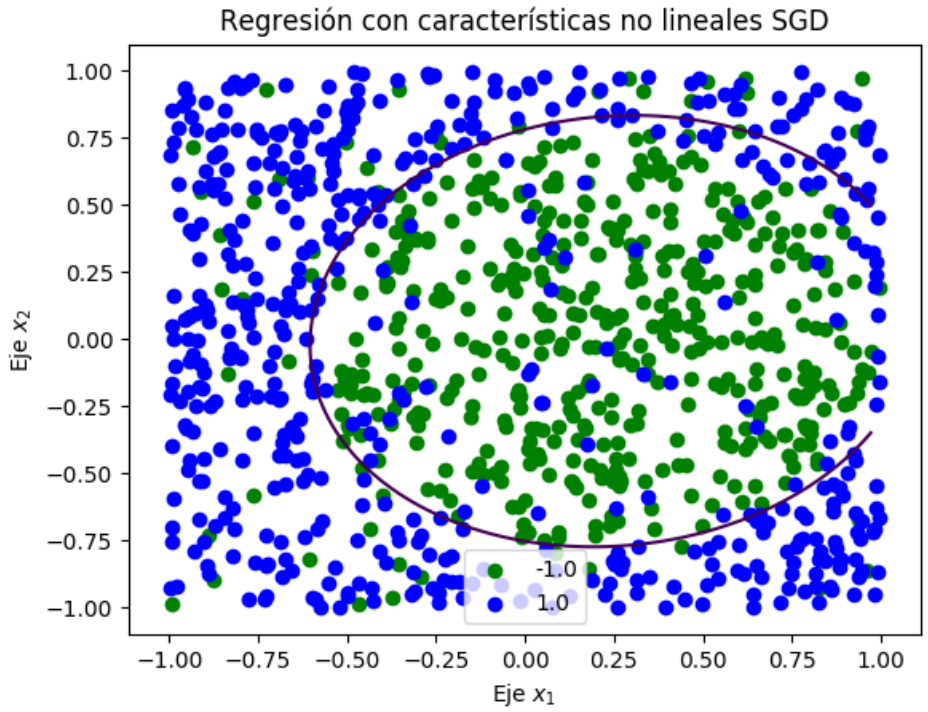

Figura 8: Regresión con vector de características no lineal con SGD

Los resultados del error obtenidos son los siguientes:

$$E_{in}media : 0.5997862133022258 \quad E_{out}media : 0.605630388520231$$

La valoración de los resultados se encuentra en el apartado siguiente.

A la vista de los resultados de los errores promedios E in y E out obtenidos en los dos experimentos ¿Que modelo considera que es el más adecuado? Justifique la decisión.

Como vemos, los resultados son bastante mejores que los obtenidos usando el vector de características $(1, x_1, x_2)$. Sin embargo tampoco llegan a ser lo suficientemente buenos.

Esto puede ser debido al ruido. Hemos hecho la prueba de ejecutar el mismo experimento pero ahora sin incorporar ruido a los datos y el resultado de la bondad según el error ha sido el siguiente:

$$E_{in}media : 0.3772253582474472 \quad E_{out}media : 0.38156952880621486$$

Vemos que aunque llega a bajar de forma considerable el error, sigue siendo bastante destacado. De esta forma podemos comprobar que aunque el éste vector de características es más apropiado que el anterior, sigue siendo mejorable.

3. Bonus (2 puntos)

Método de Newton: 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.

Hemos desarollado los mismo experimentos que en el apartado 3 del ejercicio 1.

El primer experimento consistía en probar varios η y ver el resultado:

1. Punto $(1, -1)$ con $\eta = 0.01$: Adjuntamos gráfica de valor según las iteraciones del método, coordenadas devueltas y evaluación de las coordenadas en la función:

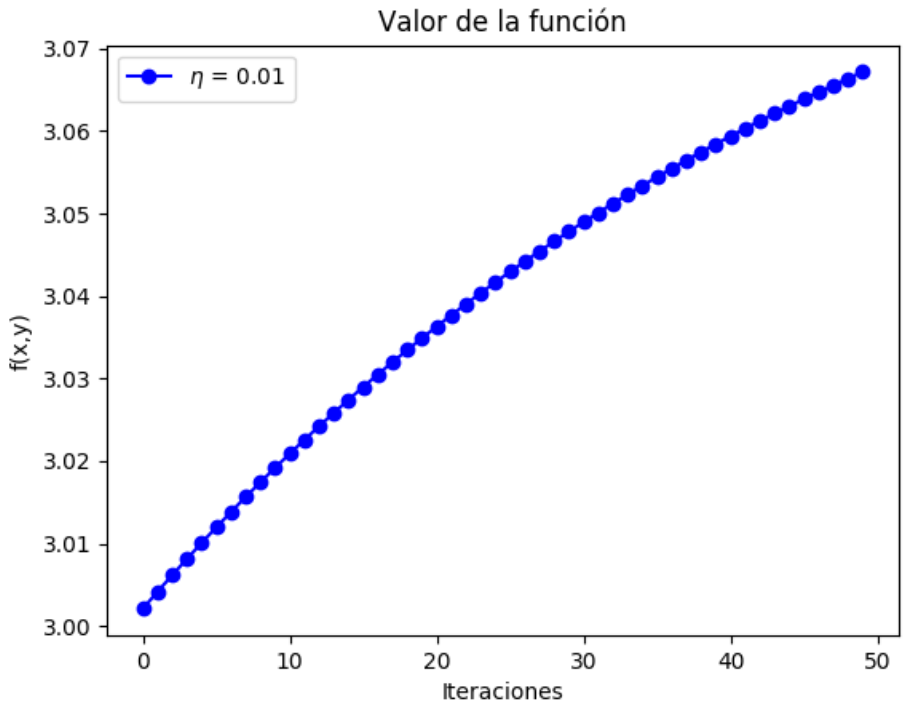

Figura 9: Método Newton con (1,-1) y $\eta = 0.01$

$$(x, y) = (0.9793498427941949, -0.9893767407575305)$$
$$f(x, y) = 3.0671859515488302$$

Como vemos el método para justo éste problema realiza una labor totalmente contraria a lo que queremos. Intentaremos darle una explicación a los resultados a continuación.

2. Punto $(1, -1)$ con $\eta = 0.1$: Adjuntamos gráfica de valor según las iteraciones del método, coordenadas devueltas y evaluación de las coordenadas en la función:

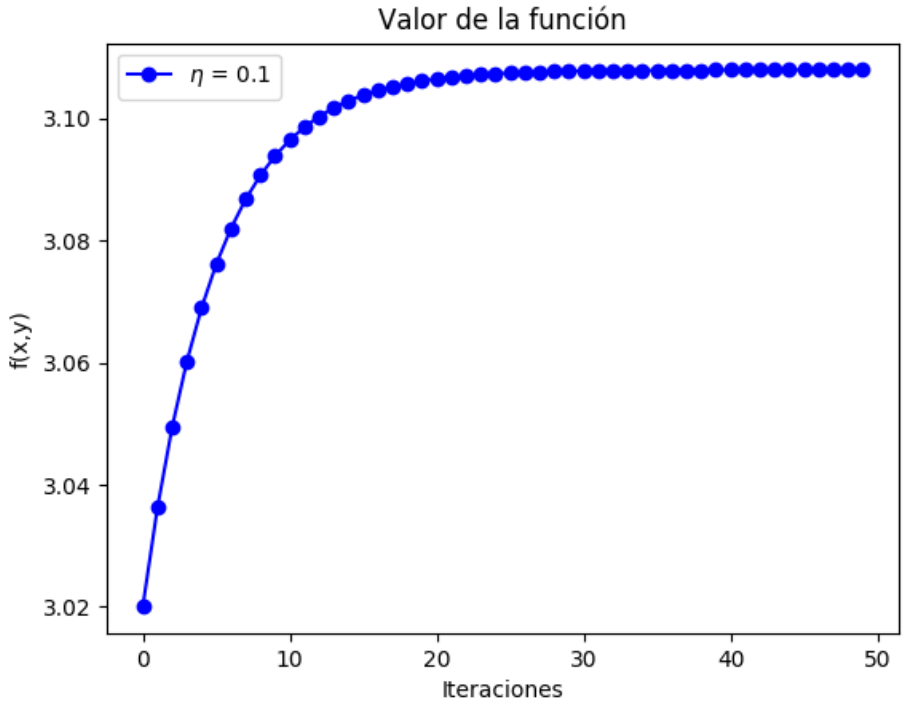

Figura 10: Metodo Newton con (1,-1) y $\eta = 0.1$

$$(x, y) = (0.9463274028190676, -0.9717082373563009)$$
$$f(x, y) = 3.1079767229661335$$

Vemos que el resultado sigue siendo parecido, aunque ahora lo vemos con mayor claridad, el resultado converge hacia un máximo y no un mínimo.

El problema del método de Newton es que solo buscamos hacer el gradiente igual a cero, pero ésto ocurre tanto en los mínimos como en los máximos o puntos de silla. Así depende del punto inicial escogido podemos acercarnos a cualquiera de estas tres situaciones.

Para el segundo experimento cabe destacar que hemos realizado varias pruebas y la que mejor resultados proporcionaba era con pocas iteraciones (hemos usado 50) y un $\eta = 1$, o sea, sin uso. Así, hemos obtenido la siguiente tabla:

Puntos Iniciales	**Coordenadas del mínimo**	**Valor de la función**
(2.1, -2.1)	(1.7561950306352119, -1.7620741785138223)	-1.8200785415471565
(3, -3)	(3.05397555493864, -3.028461459660191)	3.1079800610352026
(1.5, 1.5)	(1.7048309830043302, 0.9731715834776464)	18.08874203270784
(1, -1)	(0.9460244450613605, -0.9715385403398092)	3.1079800610352017

Aunque habíamos comprobado que para el método del gradiente descendente lo resultados dependían de los parámetros escogidos, intentamos hacer una comparativa entre dichos datos y los obtenidos con el método de Newton recogidos en la tabla anterior: Vemos que si partimos de los puntos iniciales $(2.1, -2.1)$ o $(1.5, 1.5)$ los resultados obtenidos son muy similares. Podemos decir que el método de Newton cumple la función que queremos para dichos valores iniciales. Sin embargo para los puntos $(3, -3)$ y $(1, -1)$ el resultado es paupérrimo y curiosamente el mismo entre ellos.

La explicación es la anterior dada: parece que en los puntos $(3, -3)$ y $(1, -1)$ el método de Newton no se acerca al mínimo sino todo lo contrario. Al intentar hacer el gradiente cero, nos acercamos a un máximo o a un punto de silla.

A continuación vamos a establecer una comparativa con el método del gradiente descendente para el punto $(1, -1)$ que vemos que tiene un mayor interés:

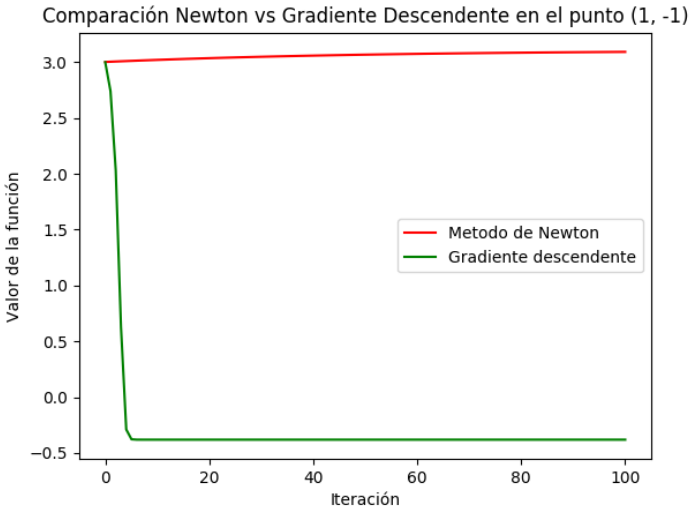

Figura 11: Método Newton vs Gradiente Descendente partiendo (1,-1)

Vemos como el método del gradiente descendente es mejor (100 iteraciones y $\eta = 0.01$) que el método de Newton para éste caso.

Sin embargo, también habíamos visto puntos en los que ambos algorítmos obtenían resultados similares como por ejemplo para esos mismos parámetros en el punto $(2.1, -2.1)$. Lo estudiamos:

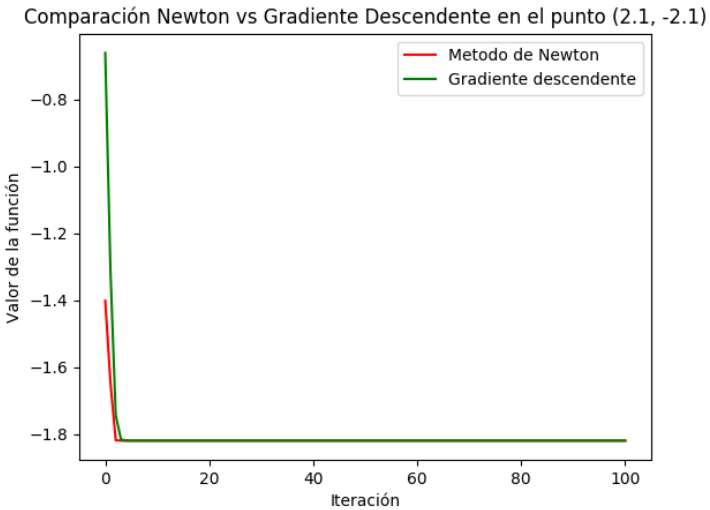

Figura 12: Método Newton vs Gradiente Descendente partiendo (2.1,-2.1)

Así deducimos que, aunque para algunos puntos puede ser útil el método de Newton en la labor que queremos realizar, en general dicho método es peor que el algoritmo del Gradiente Descendente porque el método de Newton no garantiza converger a un mínimo local mientras que el algoritmo del Gradiente Descente si lo garantiza escogiendo correctamente los parámetros.