UNIVERSIDAD DE GRANADA

AA

Aprendizaje Automático

Práctica 1

Autores:
Mario Carmona Segovia

Profesor: Pablo Mesejo Santiago

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice

Índice de figuras

Índice de tablas

1. Ejercicio 1: Búsqueda iterativa de óptimos

1.1. Ejercicio 1.1

El algoritmo de gradiente descendente consiste en una búsqueda iterativa del óptimo, partiendo desde un punto concreto.

La ecuación general del algoritmo de gradiente descendente es la siguiente:

$$w_j := w_j - \eta \frac{\partial E_{in}(w)}{\partial w_j}$$

En nuestro caso buscamos el mínimo error. En este ejercicio nos dan las funciones que nos proporcionan el valor del error en cierto punto, por lo que para minimizar el error, hay que obtener el gradiente de esa función.

En cada iteración del algoritmo, es el gradiente

$$\frac{\partial E_{in}(w)}{\partial w_j}$$

el que guía la minimización del error. El valor del gradiente es positivo, lo que es útil para maximizar, pero en nuestro caso necesitamos lo contrario; por ello ponemos el signo menos delante del valor del gradiente para que nos de como valor la pendiente descendente que hay en el punto actual de la iteración.

Cada iteración busca acercarnos más al mínimo, para ello modifica el punto en que nos encontramos en base a la tasa de aprendizaje y al gradiente. Y utiliza este punto resultante para la siguiente iteración.

El algoritmo se para cuando se han realizado un máximo de iteraciones, para prevenir bucles infinitos; o cuando se ha llegado a un mínimo de error que se considera adecuado.

1.2. Ejercicio 1.2

1.2.1. Apartado A

La función es:

$$E(u,v) = (u^3 e^{v-2} - 2v^2 e^{-u})^2$$

La derivada parcial de E respecto de u es:

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

La derivada parcial de E respecto de v es:

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

Estas derivadas de la función E se usarán para crear el gradiente.

1.2.2. Apartado B

El criterio de parada del algoritmo en este ejercicio es no sobrepasar el límite máximo de iteraciones y no sobrepasar el límite mínimo de error.

En este ejercicio el punto inicial es el ($u_o = 1$, $v_o = 1$) y la tasa de aprendizaje (η) es igual a 0.1 .

El número de iteraciones necesarias para llegar al límite mínimo de error indicado (10^{-14}) en el ejercicio es de 10 iteraciones.

En la figura 1 se puede comprobar como se produce un descenso monótono del valor del error con el paso de las iteraciones.

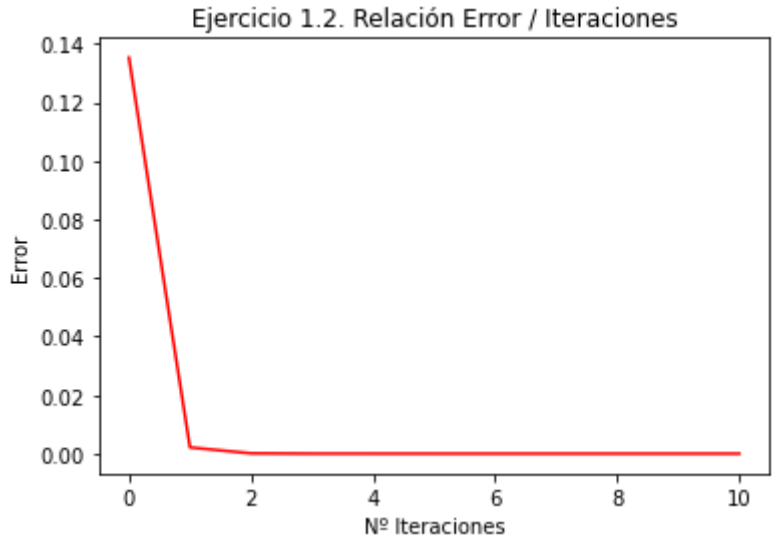

Figura 1: Ejercicio 1.2: Relación error/iteraciones

La minimización a partir de la primera iteración es más lenta, debido a que el gradiente de la función es menor, lo que provoca una menor varianza del error de un punto a otro.

1.2.3. Apartado C

Las coordenadas (u,v) donde se obtiene el límite mínimo de error (10^{-14}) es [1.1572888496465497, 0.9108383657484799].

En la figura 2 se puede observar la representación gráfica de la función E(u,v) en el rango de valores de [-30, 30] en ambos ejes. Además se ha colocado una estrella en las coordenadas obtenidas.

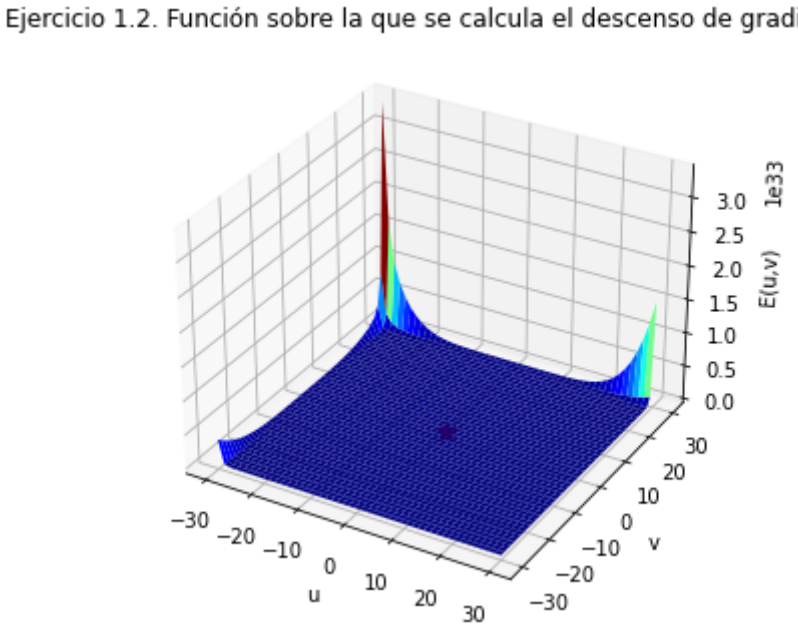

Figura 2: Ejercicio 1.2: Función E(u,v)

Al observar la figura 2 se puede entender porque en la figura 1, a partir de la iteración 1 el descenso del error era muy pequeño. Esto es debido a la pequeña inclinación ó gradiente en esa parte de la función.

1.3. Ejercicio 1.3

La función es:

$$f(x,y) = (x+2)^2 + 2(y-2)^2 + 2\sin(2\pi x)\sin(2\pi y)$$

La derivada parcial de f respecto de x es:

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

La derivada parcial de f respecto de y es:

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

1.3.1. Apartado A

El criterio de parada del algoritmo en este ejercicio es no sobrepasar el límite máximo de iteraciones, que en este caso es igual a 50.

En este apartado el punto inicial es el (x_o = -1, y_o = 1) y vamos a coger dos valores para la tasa de aprendizaje, que son 0.01 y 0.1 .

En la figura 3 se puede comprobar el comportamiento del algoritmo con el paso de las iteraciones con ambas tasas de aprendizaje.

Figura 3: Ejercicio 1.3: Relación error/iteraciones η=0.01

Como se puede observar la diferencia es muy notable. Mientras que la tasa de aprendizaje más pequeña sigue un descenso monótono, la tasa de aprendizaje más grande empieza a oscilar. En la figura 4 tenemos la función usada en el ejercicio, para tener una idea de como es la función en su mínimo.

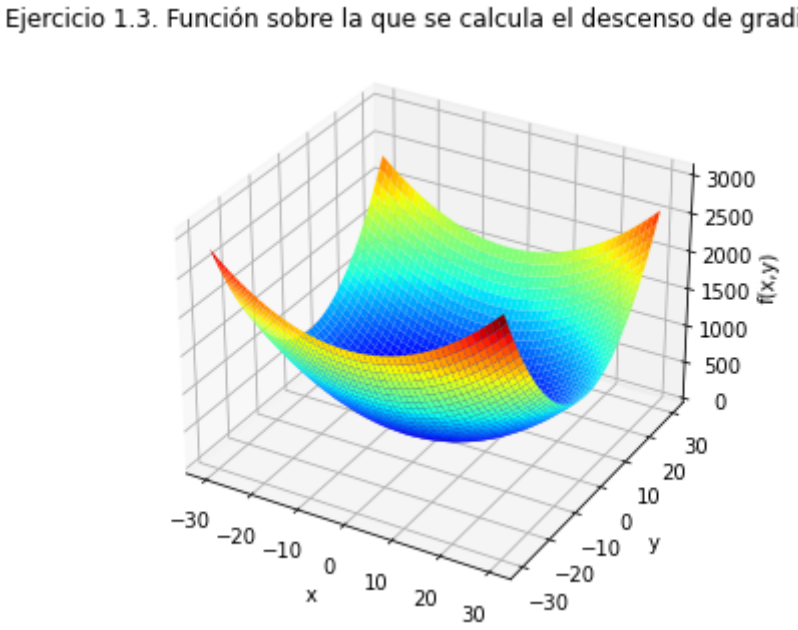

Figura 4: Ejercicio 1.3: Función f(x,y)

Como la única variable del algoritmo que ha cambiado entre ambos ejemplos es la tasa de aprendizaje, esta diferencia es debida a ella. El valor de la tasa de aprendizaje afecta al tamaño del desplazamiento que se hace siguiendo el gradiente descendente.

Teniendo en cuenta que con un mayor η se realizan mayores desplazamiento, en la segunda opción con el η=0.1 se deberían realizar menos iteraciones, pero no sucede así. Esto es debido a que al estar el punto inicial muy cerca del mínimo, si se realizan desplazamientos muy grandes el algoritmo empieza iteración tras iteración a oscilar alrededor del mínimo, ya que la función en el mínimo tiene forma de semicírculo por lo que irá de un lado a otro siguiendo el gradiente. Esta oscilación hace que tarden las mismas iteraciones, y además provoca que la solución obtenida con esta opción sea de menos calidad. El problema de un η grande es que la calidad de la solución no tiene porque ser mayor cuantas más iteraciones han pasado, a diferencia del η más pequeño en el que si sucede esto.

En el caso de el η=0.01, se realizan pequeños desplazamientos en cada iteración, por lo que al estar el punto inicial cerca del mínimo en pocas iteraciones se llegará a un mínimo de calidad, sin realizar muchas oscilaciones.

1.3.2. Apartado B

El criterio de parada del algoritmo en este ejercicio es no sobrepasar el límite máximo de iteraciones, que en este caso es igual a 50.

En este apartado la tasa de aprendizaje es igual a 0.01, y como punto inicial vamos a coger una serie de puntos para comparar los resultados obtenidos.

Punto inicial	**Coor. obtenidas**	**Error obtenido**
(-0.5, -0.5)	(-0.793, -0.126)	9.125
(1, 1)	(0.677, 1.29)	6.437
(2.1, -2.1)	(0.149, -0.096)	12.491
(-3, 3)	(-2.731, 2.713)	-0.381
(-2, 2)	(-2, 2)	-4.799

Tabla 1: Ejer. 1.3: Comparativa del error con diferentes puntos iniciales

En esta tabla se reflejan los resultados del algoritmo con cada punto inicial. En la tabla sólo se ha mostrado una precisión de tres decimales para facilitar la comprensión de la tabla. Para ver los resultados exactos se puede ejecutar el script de python.

El algoritmo de gradiente descendente es un algoritmo de búsqueda local, ya que en cada iteración se busca el mínimo valor dentro del entorno del punto actual, esta característica hace que el algoritmo de gradiente descendente sea muy dependiente del punto inicial, y sobretodo si se pone un límite de iteraciones y una tasa de aprendizaje pequeña, que hace que los desplazamientos entre iteraciones sean más pequeños; la importancia del punto inicial aumenta, ya que la ejecución que parta desde un punto más alejado del mínimo necesitará más iteraciones para llegar a él, pero al tener unas iteraciones limitadas no llegará a alcanzar un buen error. Además esta característica de búsqueda local puede hacer que el algoritmo se quede en mínimos locales según el punto inicial que elijamos.

De esta tabla podemos deducir que puntos son los que están más cerca o más lejos del mínimo global, ya que como se pudo ver en la figura 4 4, la función con la que trabajamos sólo tiene un mínimo, por lo que todos los gradientes irán hacia él. Los puntos más cerca del mínimo global son (-3, 3) y (-2, 2), que tiene un error inferior a -0.4. Y los puntos más alejados del mínimo local son (-0.5, -0.5), (1, 1) y (2.1, -2.1), que tienen un error superior a 6.4 .

1.4. Ejercicio 1.4

La dificultad de encontrar el mínimo global con el algoritmo del gradiente descendente sobre una función arbitraria reside en la elección del valor de estas variables:

- Tasa de aprendizaje: si el valor de la tasa de aprendizaje es grande encontraremos un mínimo en menos iteraciones, pero puede suceder que el algoritmo empiece a divergir alrededor del mínimo; y si el valor es pequeño se tardarán muchas iteraciones en llegar al mínimo, por lo que será difícil establecer un límite de iteraciones máximo, pero nos aseguramos que no diverge alrededor del mínimo. Pero como la función es arbitraria no sabemos que valor de η es mejor en cada momento del algoritmo.

- Punto inicial: como el algoritmo de gradiente descendente se basa en una búsqueda local, al lanzar el algoritmo a pesar de llegar a un mínimo no se asegura que sea el mínimo global. Por lo que otro factor es la elección correcta de un punto inicial, pero nos volvemos a encontrar con el problema de que el algoritmo se está ejecutando sobre una función arbitraria.

- Cálculo de las derivadas: la función debe ser identificable para poder calcular sus derivadas y obtener el gradiente descendente.

2. Ejercicio 2: Regresión lineal

En este ejercicio vamos a usar el algoritmo del gradiente descendente para ajustar modelos de regresión, para ser usados en clasificación.

En concreto vamos a usar una variante del gradiente descendente, que es el gradiente descendente estocástico (SGD). Lo que cambia respecto al gradiente descendente, es que a la hora de modificar los pesos no se tienen en cuenta todos los datos, sino que se tiene en cuenta porciones de los datos de entrada, estas porciones son llamadas minibatch; por lo que ahora otra variable del algoritmo será el tamaño de este minibatch.

Como función para obtener el error en cierto punto se va a utilizar la función del error cuadrático:

$$E_{in}(w) = 1/M * \sum_{n=1}^{M} (w^T * x_n - y_n)$$

Por lo que el gradiente del SGD se obtendrá de la derivada del error cuadrático respecto de los pesos:

$$\frac{\partial E_{in}(w)}{\partial w_j} = 2/M * \sum_{n=1}^{M} x_{nj} * (w^T * x_n - y_n)$$

Además del SGD vamos a usar el algoritmo de la pseudoinversa.

$$X^{'} = (X^T * X)^{-1} * X^T$$

2.1. Ejercicio 2.1

Las etiquetas usadas para los datos de salida son -1 y 1. Deben ser el mismo número pero en sentido opuesto para que al mover los pesos, los errores con una etiqueta sean proporcionales a los errores con la otra etiqueta.

Antes de lanzar los algoritmos he realizado un análisis de la importancia del número de iteraciones y del tamaño del minibatch. Y los resultados obtenidos son los siguientes ($\eta = 0.01$):

Nº Iteraciones \Tam. Batch	**50**	**100**	**150**
200	Ein: 1.55 Eout: 1.61	Ein: 1.56 Eout: 1.63	Ein: 1.57 Eout: 1.63
500	Ein: 1.74 Eout: 1.81	Ein: 1.73 Eout: 1.81	Ein: 1.73 Eout: 1.80
1000	Ein: 1.74 Eout: 1.81	Ein: 1.75 Eout: 1.83	Ein: 1.75 Eout: 1.82

Tabla 2: Ejer. 2.1: Importancia del nº de iteraciones y del tam. de minibatch

Los valores de la tabla no son exactamente los valores obtenidos, ya que han sido redondeados al segundo decimal. Como podemos ver, con los valores que hemos realizado el análisis, hemos

obtenido casi los mismos errores tanto dentro como fuera de la muestra. Por lo que en principio para tamaños tan pequeños no hay casi diferencia entre las distintas combinaciones. En mi caso para el algoritmo SGD elegiré un nº de iteraciones igual a 500, un tamaño de minibatch igual a 50 y un η igual a 0.01 .

Para comparar los resultados de ambos algoritmos con los datos proporcionados para el ejercicio, he obtenido un modelo con ambos algoritmos y he calculado su error, tanto dentro como fuera de la muestra. Los resultados son los siguientes:

Error \Algoritmo	**Pseudoinversa**	**SGD**
Ein	0.079	1.721
Eout	0.131	1.791

Tabla 3: Ejer. 2.1: Comparativa de errores obtenidos con SGD y pseudoinversa

La ventaja de la pseudoinversa sobre el SGD es clara, mientras que para la pseudoinversa sólo ha hecho falta unas pocas operaciones, para el SGD han hecho falta muchas iteraciones, además ha obtenido un modelo peor. Pero esto no quiere decir que la pseudoinversa sea mejor algoritmo que el SGD, ya que el algoritmo de la pseudoinversa sólo se puede realizar en ejercicios en los que sea posible calcular la inversa del producto escalar del vector de características por su traspuesta, en aquellos casos en que sea imposible o muy costoso este algoritmo deja de ser una posibilidad.

Pero en el caso de que sea posible usar la pseudoinversa, es una bunea opción para obtener una solución rápida, ya que en la prueba que he realizado, el SGD tarda más de 19 veces el tiempo que tarda en ejecutarse la pseudoinversa. En concreto se han obtenido los siguientes tiempos:

- SGD: 0.0252
- Pseudoinversa: 0.0013

Estos tiempos pueden variar entre ejecución ya que en este tiempo no se tiene en cuenta sólo el tiempo gastado por el algoritmo, pero nos sirve para hacernos una idea de la diferencia de tiempo que puede existir entre ambos algoritmos.

El modelo obtenido en el SGD junto con los datos de train y test se ha representado en las siguientes figuras:

La recta del modelo se ha dibujado con dos puntos que pertenecen a la recta y que se encuentran en cada uno de los límites de x en la gráfica. Para obtener estos puntos se ha seguido la siguiente dedución:

- Como los puntos que usamos pertenecen a la recta y = 0.

$$y = w^T * x = w_0 + w_1 * x_1 + w_2 * x_2$$
$$0 = w_0 + w_1 * x_1 + w_2 * x_2$$

- Como queremos obtener x_2, sólo tenemos que despejarla de la fórmula, y ya se pueden obtener los valores de x_2 a partir de x_1 y los pesos.

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

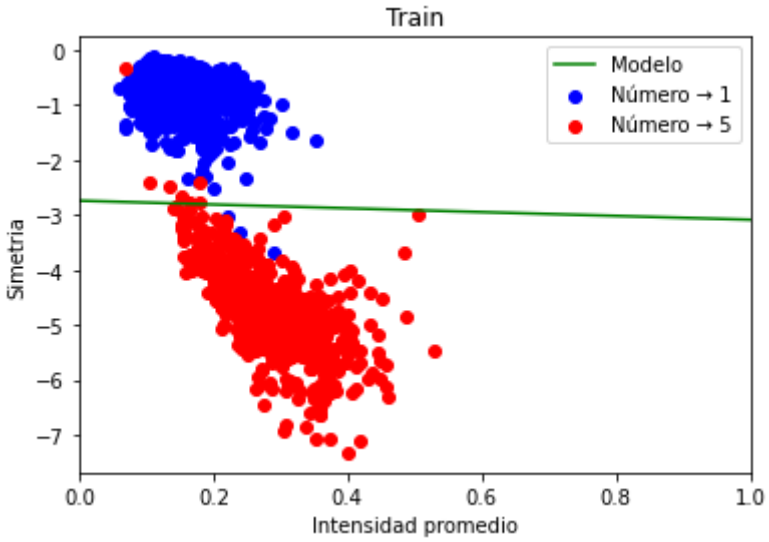

Figura 5: Ejercicio 2.1: Datos de train junto con el modelo obtenido

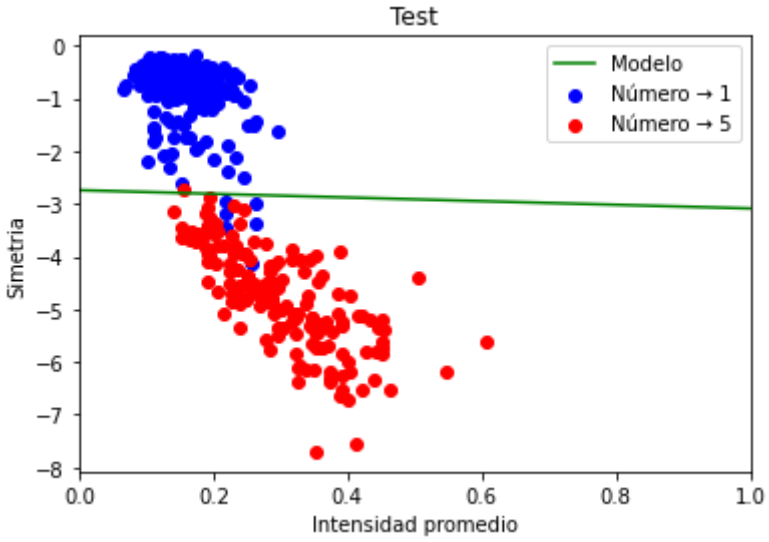

Figura 6: Ejercicio 2.1: Datos de test junto con el modelo obtenido

Como se puede observar el modelo no acierta en algunas de las etiquetas en ambas figuras, y esto provoca el error obtenido tanto en train como en test.

2.2. Ejercicio 2.2

2.2.1. Apartado D (Lineal)

Para este apartado, tanto en el caso del modelo lineal como del no lineal, se ha considerado que el realizar una actualización de los valores de los pesos (w_j) cuenta como una iteración.

En las ejecuciones se han dado los siguientes valores a las variables del algoritmo:

- Número de datos en cada muestra = 1000
- Porcentaje de ruido = 10 %
- Tasa de aprendizaje = 0.01
- Tamaño del minibatch = 50
- Número máximo de iteraciones = 500

Los errores medios obtenidos son los siguientes:

- Ein: 0.9196994225322596
- Eout: 0.940395077612014

2.2.2. Apartado E (Lineal)

El ajuste con el modelo lineal no es muy bueno, ya que se ha obtenido un error cercano a 1 y lo que buscamos es un error los más cercano a cero.

En esta figura se muestra la representación de los datos junto con el modelo lineal:

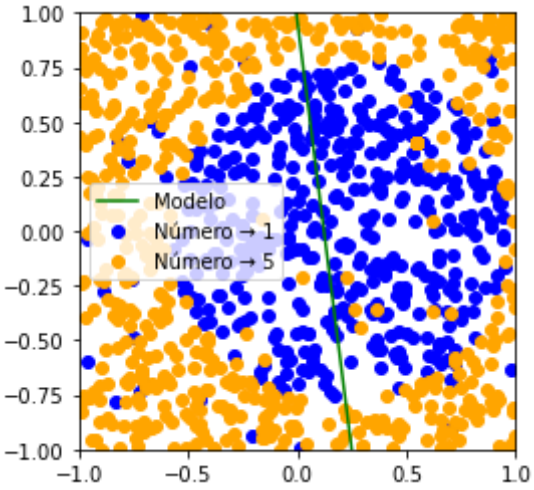

Figura 7: Ejercicio 2.2: Datos junto con el modelo lineal obtenido

Como se puede observar los datos se encuentra en un espacio cuadrático, debido a que las salida etiquetadas han sido creadas con una función cuadrática. Un modelo lineal es imposible que pueda clasificar los datos en este espacio obteniendo un error igual a cero. Por lo que parece mejor opción un modelo no lineal que pueda adaptarse más a la distribución de los datos en el espacio.

2.2.3. Apartado D (No Lineal)

En el modelo no lineal el vector de características es el siguiente:

$$\{1, x_1, x_2, x_1 * x_2, x_1^2, x_2^2\}$$

En las ejecuciones se han dado los siguientes valores a las variables del algoritmo:

- Número de datos en cada muestra = 1000
- Porcentaje de ruido = 10 %
- Tasa de aprendizaje = 0.01
- Tamaño del minibatch = 50
- Número máximo de iteraciones = 500

Los errores medios obtenidos son los siguientes:

- Ein: 0.5716841791407002
- Eout: 0.576199946593712

2.2.4. Apartado E (No Lineal)

El resultado es el esperado, con el modelo no lineal se ha obtenido casi la mitad de error que con el modelo lineal, esto es debido a lo explicado en el apartado E del modelo lineal.

Además de la diferencia de error entre ambos modelos, también es notable la diferencia de tiempos, esto es debido al mayor tamaño del vector de características del modelo no lineal, lo que conlleva un mayor número de operaciones con los vectores. Pero vale la pena gastar ese tiempo, ya que puede dar soluciones de una mayor calidad, que con el modelo lineal a partir de los datos de la muestra de este ejercicio.

3. Ejercicio Bonus: Método de Newton

3.1. Ejercicio 1

El Método de Newton hace uso de las segundas derivadas. La principal diferencia del Método de Newton con respecto al gradiente descendente, es que el Método de Newton en vez de hacer uso de la tasa de aprendizaje, que es un valor difícil de establecer ya que no se sabe como va a actuar con los datos a priori; hace uso de la matriz hessiana para guiar la optimización sustituyendo la tasa de aprendizaje.

Este algoritmo es muy útil cuando estamos muy cerca de un óptimo local. Este algoritmo consiste en una aproximación al óptimo, ajustando una superficie al óptimo, usando una función de segundo orden o paraboloide; y lo que tiene que buscar el algoritmo es el óptimo del paraboloide, que corresponde con el óptimo local.

Este algoritmo garantiza que en un número de pasos finito llega al óptimo local. Por esta característica es usado cuando estamos cerca del óptimo.

El coste de este algoritmo es calcular la inversa de H, que es la matriz hessiana. Si H es muy grande este método no es tan buena opción, es mejor cuanto menor es el tamaño de H.

La matriz hessiana tiene tantas filas y columnas como variables tiene la función, por lo tanto a mayor número de variables, mayor tamaño va a tener H; y es simétrica tomando como eje la diagonal principal.

El contenido de la matriz hessiana en este ejercicio tendrá la siguiente forma:

$$\begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

La fórmula que actualiza los pesos quedaría de la siguiente forma:

$$w_j = w_j - H^{-1} * Gradiente$$

El criterio de parada es llegar a un punto en que la derivada valga cero, es decir, se ha llegado a un óptimo local. En esta práctica utilizaremos como criterio de parada un número máximo de iteraciones para poder crear una gráfica que compare este algoritmo con SGD.

3.1.1. Apartado A

En este apartado se ha vuelto a optimizar el error como en el apartado A del ejercicio 1.3.

Se han tomado los mismos valores para las variables de las que dependen los algoritmos:

- Tasa de aprendizaje: 0.01

- Punto de inicio: [-1, 1]
- Número de iteraciones: 50

En esta figura se puede ver la comparación del descenso del error durante la ejecución en ambos algoritmos:

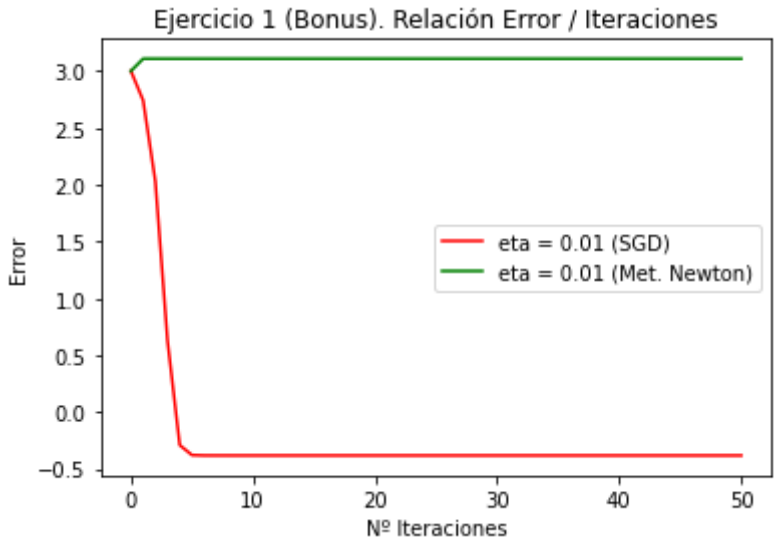

Figura 8: Ejercicio 3.1: Relación error/iteraciones

Como se puede apreciar el Método de Newton no ha minimizado el error, incluso lo ha aumentado un poco, esto es debido a la matriz hessiana, ya que para que el Método de Newton busque un mínimo local, la matriz hessiana debe ser definida positiva, es decir, que todos los elementos de la matriz deben tener un valor mayor que cero. Que la matriz hessiana sea definida positiva indica que la función es estrictamente convexa. Por lo que la matriz hessiana de esta función no debe ser definida positiva, por lo que la línea de la gráfica no será estrictamente descendente.

3.1.2. Apartado B

En este apartado se ha obtenido la siguiente tabla:

Punto inicial	**Coor. obtenidas**	**Error obtenido**
(-0.5, -0.5)	(224.353, -186.131)	122020.964
(1, 1)	(1.067, 0.91)	11.345
(2.1, -2.1)	(3.828, -3.66)	96.546
(-3, 3)	(-3.054, 3.028)	3.108
(-2, 2)	(-2, 2)	-4.799

Tabla 4: Ejer. 3.1: Comparativa del error con diferentes puntos iniciales

En todos los puntos se ha empeorado, menos en el punto (-2, 2), donde se ha mantenido cerca el error de ambos algoritmos. Este empeoramiento es debido a lo explicado en el apartado A, es decir, la matriz hessiana no es definida positiva.

Referencias

[1] Anónimo. (5 de abril de 2021). Matriz hessiana. Wikipedia. Recuperado el 5 de abril del 2021 de https://es.wikipedia.org/wiki/Matriz_hessiana

[2] Anónimo. Matriz Hessiana (o Hessiano). matricesydeterminantes. Recuperado el 5 de abril del 2021 de https://www.matricesydeterminantes.com/matrices/matriz-hessiana-hessiano-2x2-3x3/