Aprendizaje Automático

Búsqueda Iterativa de Óptimos y Regresión Lineal

Práctica 1

Juan José Herrera Aranda - juanjoha@correo.ugr.es

3 de abril de 2022

Índice

1. Introducción **3**

2. Ejercicio 1 **3**
2.1. Gradiente Descendente 3
2.2. Empleo del gradiente descendente: I 4
2.3. Empleo del gradiente descendente: II 5
2.4. Conclusión 9

3. Ejercicio2 **10**
3.1. Ajuste lineal al problema de los dígitos simplificado 10
3.2. Regresión Lineal II 12

4. Bonus **15**

1. Introducción

Se presenta a continuación la memoria de la Práctica 1 de la asignatura Aprendizaje Automático elaborada por Juan José Herrera Aranda. Esta práctica consta de tres ejercicios. El primero se centra en la aplicación del gradiente descendente como método de optimización local para funciones diferenciables. Se estudiará la dependencia que juegan los parámetros en el algoritmo y se concluirá con una reflexión sobre la complejidad que presentan los problemas de optimización.

El segundo ejercicio se basa en aplicar regresión lineal a un problema dado desde dos puntos de vista. Uno es de forma directa aplicando el método de la pseudoinversa que nos da la solución directamente y el otro es a través del gradiente descendente. Tras lo cual, se cambiará a otro problema más complejo, (no lineal) donde el modelo lineal no sea bueno, pero tras añadir características no lineales se verán mejoras sustanciales. Se analizarán también los errores fuera y dentro de la muestra. Finalmente, el último ejercicio será aplicar y analizar el conocido método de Newton.

Las prácticas se han realizado en un ordenador con una CPU *Quad Core Intel Core i7-1065G7*, una memoria RAM de *16GB* y con el sistema operativo *Ubuntu 20.04*.

Nota: El ejercicio 1.3 me ha dado resultados distintos al ejecutarlo en mi ordenador local y en google colab (solo cuando $\mu = 0{,}1$. Los resultados expuestos en este guión son los que ha proporcionado mi ordenador local. Algo parecido ocurre con la visualización de las superficies en tres dimensiones, en google colab se ven algo distorsionadas pero tanto en la memoria como en mi ordenador no.

2. Ejercicio 1

2.1. Gradiente Descendente

Se va a usar el algoritmo del gradiente descendente para la búsqueda de óptimo. La implementación del mismo queda reflejada en el código pero básicamente se puede resumir obviando muchos detalles de la siguiente manera:

Algorithm 1: Gradiente descendente

```
Data: w_o: tupla, ∇f: función, lr: float,...
begin
    w ← w_o;
    while Criterio de parada do
        w ← (w − lr · ∇f);
    end
end
```

Este algoritmo parte de unos puntos iniciales, y en cada iteración se desplaza en la dirección negativa del gradiente en el punto actual y lo hace una cantidad proporcional al parámetro *learning rate*. El algoritmo acaba según un criterio de parada, que o bien puede ser un número determinado de iteraciones o cuando la función alcance un determinado umbral. Comentar que los criterios anteriores tienen un problema, por una parte, si el algoritmo tarda en converger, puede ser que las iteraciones que especifiquemos sean insuficientes y nos devuelva un resultado malo, también puede ocurrir que el algoritmo converja muy rápido y estemos desperdiciando tiempo de cómputo. Por otra parte, el criterio del umbral es bastante peligroso ya que si no

conocemos la función per se, puede desencadenar en un bucle infinito en el caso de que especificamos un umbral menor que el óptimo local al que llegamos.

El gradiente descendente es un algoritmo de búsqueda local, por lo que el óptimo encontrado normalmente no va a ser global, salvo que la función sea convexa o que la elección de los puntos iniciales estén lo suficientemente cerca del óptimo global. Puede ser que el óptimo que nos de sea bastante malo, por lo que generalmente se suele emplear la técnica de la *búsqueda local multiarranque*, que consiste en ejecutar varias veces el algoritmo pero con distintos puntos iniciales o bien, emplear alguna técnica metaheurística para intentar escapar de óptimos locales como por ejemplo hace el algoritmo del enfriamiento simulado.

2.2. Empleo del gradiente descendente: I

Vamos a poner en práctica este algoritmo, para ello vamos a considerar el campo escalar $E : \mathbb{R}^2 \to \mathbb{R}$ de clase infinito definido como

$$(u, v) \to (u \cdot v \cdot e^{(-u^2-v^2)})^2 \tag{1}$$

y cuyo gradiente es $\nabla E : \mathbb{R}^2 \to \mathbb{R}^2$ el campo vectorial dado por las primeras derivadas parciales de la función E

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

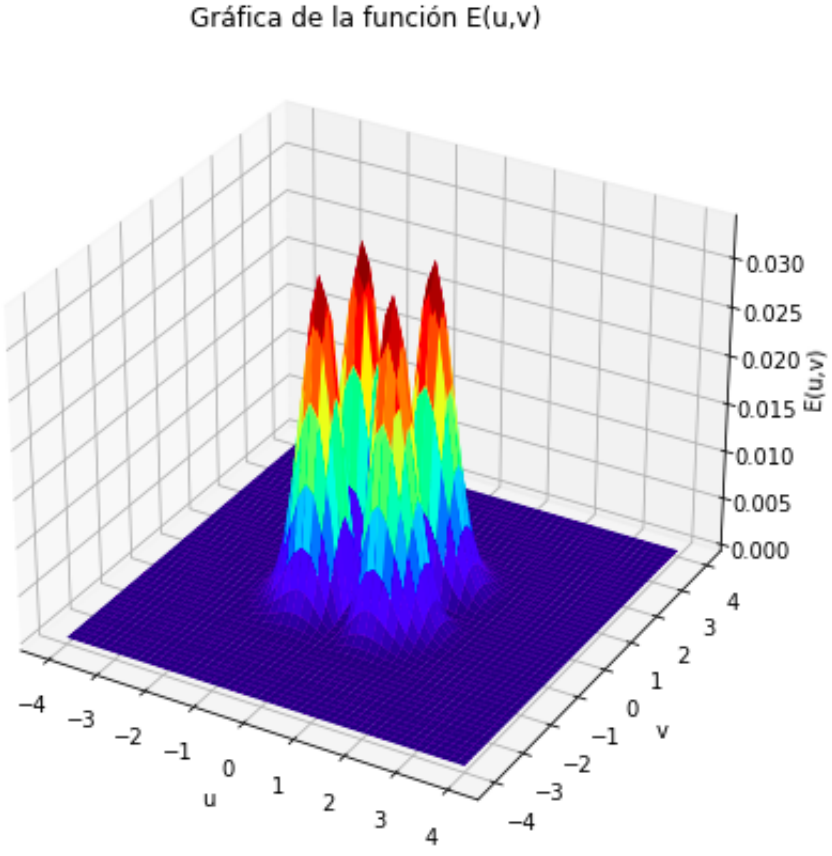


Figura 1: La forma de la gráfica de la función E es un plano en Z=0 y cerca del centro de coordenadas hay como cuatro paraboloides achatados cuya base empieza en el plano Z=0 y van ascendiendo positivamente en el eje Z hasta alcanzar un valor aproximado de 0.030.

Se parte del punto inicial $w_o = (0{,}5, -0{,}5)$ y se aplica el gradiente descendente con una tasa de aprendizaje de $\mu = 0{,}1$. Observando la gráfica de la función (Fig. 1) se intuye que el punto inicial va a estar en uno de los cuatro paraboloides y que el mínimo a alcanzar va a ser global. Por lo que en este caso tiene sentido usar como criterio de parada el umbral, se fija un error de $\epsilon = 10e - 8$ y vamos iterando hasta que la función tome un valor menor o igual que ϵ

Tras ejecutar el algoritmo se observa que el número de iteraciones necesarias para alcanzar dicho umbral fueron 25117. El extremo lo alcanzó en el punto $x = [0{,}010000840{,}01000084]$ y el valor de la imagen en dicho punto es $E(x) = 9{,}99936950202903e - 09$.

Los resultados sugieren pensar que el algoritmo convergió hasta un entorno del óptimo de manera bastante rápida, sin embargo, se quedó oscilando durante bastantes iteraciones hasta que finalmente pudo llegar a un punto inferior al umbral.

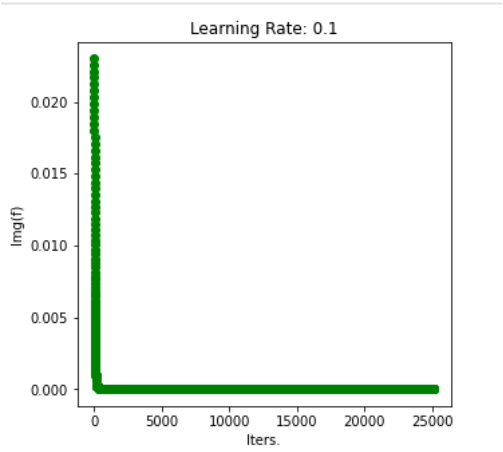


Figura 2: Gráfica en la que el eje X representa el número de iteraciones y en el eje Y el valor de la imagen de la función. Se puede apreciar que prácticamente el algoritmo converge a una zona cerca del óptimo con muy pocas iteraciones, pero tarda muchísimas más en obtener un valor inferior al error.

2.3. Empleo del gradiente descendente: II

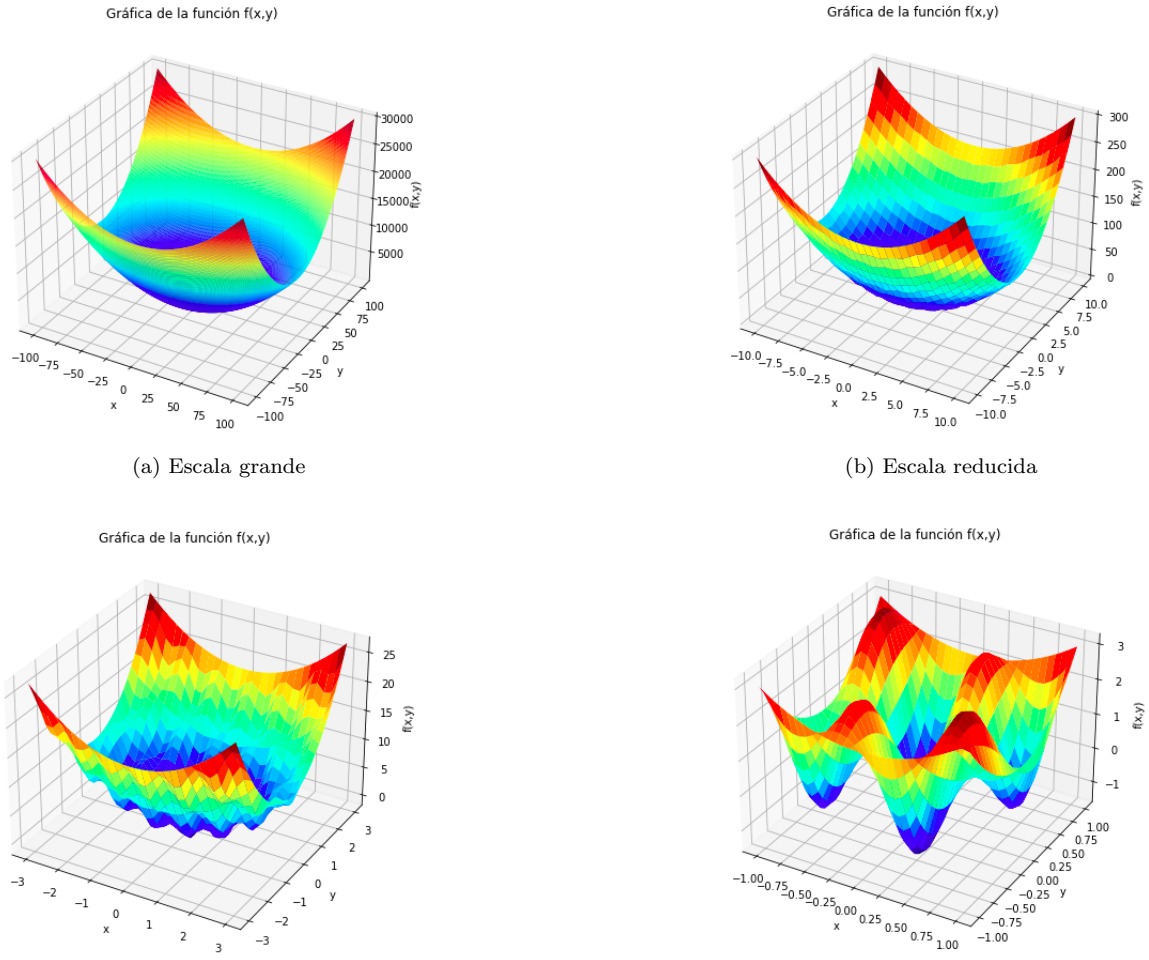
Consideramos ahora el cambio escalar $f : \mathbb{R}^2 \to \mathbb{R}$ de clase infinito definida como

$$(x, y) \to (x^2 + 2y^2 + 2\sin(2\pi x)\sin(\pi y)) \tag{2}$$

y cuyo gradiente es $\nabla f : \mathbb{R}^2 \to \mathbb{R}^2$ el campo vectorial dado por las primeras derivadas parciales de la función f

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

Esta función merece especial atención por tener un gráfica bastante engañosa (Fig. 3) ya que dependiendo de la escala a la que se muestre la gráfica se aprecia un comportamiento distinto. De manera global, la función tiene forma de paraboloide, pero a nivel local se observa un comportamiento oscilante y periódico. Así que en este caso tiene más sentido usar como criterio de parada el criterio del número máximo de iteraciones que el del umbral, ya que hay muchos extremos locales ocultos que no se ven a simple vista y sería un craso error afirmar que la función es convexa y por ende hay un único extremo.



(a) Escala grande

(b) Escala reducida

(c) Escala pequeña

(d) Escala muy pequeña

Figura 3: Gráficas de la función $f(x, y)$ mostradas a distintas escalas.

Como primera aproximación se va a usar el punto inicial $w_o = (-1, 1)$, un número máximo de iteraciones de 50, y se va a estudiar el comportamiento del algoritmo empleando como tasas de aprendizaje $\mu = 0{,}1$ y $\mu = 0{,}01$. Tras la ejecución, se observa gracias a la imagen 4 la evolución del algoritmo del gradiente descendente a lo largo de las iteraciones y en la tabla 1 se presentan los resultados. En ella se observa resultados distintos para cada tasa de aprendizaje.

Como ya se sabe, la tasa de aprendizaje viene a indicar cómo de rápido avanza el punto en la dirección negativa del gradiente ya que es proporcional. Así pues, si la tasa es pequeña, la diferencia entre dos iteraciones es también pequeña y el algoritmo converge lentamente a un mínimo. En cambio, si la tasa de aprendizaje es grande, la diferencia va a ser por lo general, también grande, el algoritmo avanza más rápido, pero puede no converger. La tasa de aprendizaje es un hiperparámetro que se debe de tunear en los modelos de Machine Learning.

Todo esto queda reflejado en la figura (Fig 4); en la imagen (a) se observa una rápida convergencia a un óptimo, y pese a que el algoritmo sigue iterando, se deja de descender porque queda atrapado en un óptimo local. En cambio, en la imagen (b) se observa cómo se va dando saltos sin llegar a converger y cómo los valores de la función van oscilando. Se aprecia también que se va escapando de los óptimo locales ya que el valor de μ es lo bastante grande como para que el algoritmo se salte los extremos. El óptimo alcanzado con $\mu = 0{,}01$ es un valor que se aproxima por la izquierda al cero, mientras que usando $\mu = 0{,}1$ y aunque el algoritmo pare en un valor que probablemente no sea ni óptimo local, se observa que hay momentos en los que se

alcanzan valores más pequeños que el óptimo usando $\mu = 0{,}1$.

Lo anterior es bastante interesante, ya que se podría modificar el gradiente descendente de tal manera que poco antes de alcanzar un óptimo local, guarde las coordenadas de un punto próximo al extremo para luego volver atrás tras quedar atrapados en un extremo local. y aplicando una tasa de aprendizaje elevada, poder escapar del extremo. Se van almacenando todos los extremos por los que pasan en un vector y al finalizar el algoritmo por cualquier criterio de parada se devolvería el extremo obtenido más pequeño.

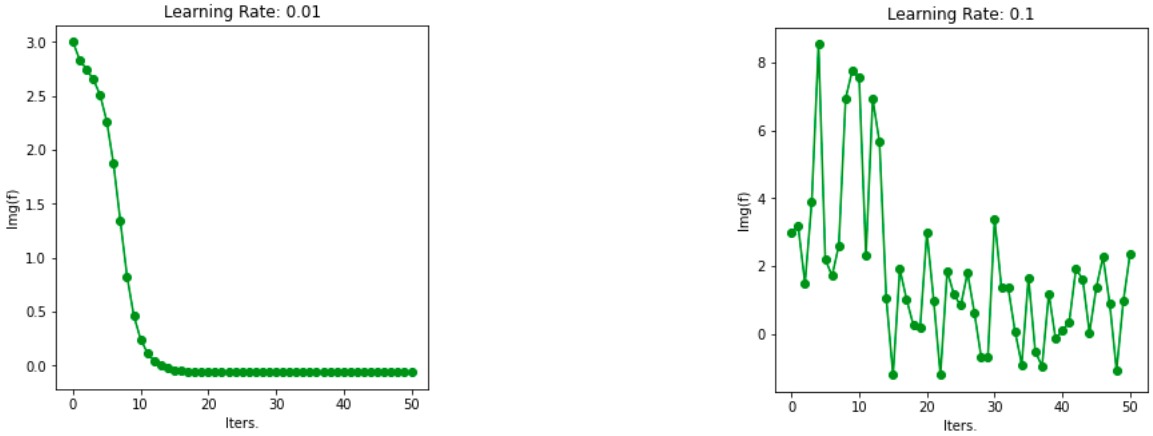


Figura 4: Evolución del gradiente descendente a lo largo de las iteraciones

Tabla 1: Resultados del ejercicio 1.3 (a)

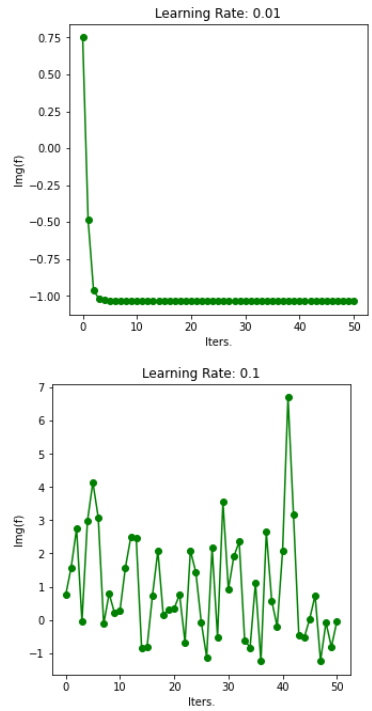
Learning Rate	**Extremos**	**Imagen**
0,01	1.21775 , 0.41341	-0.062308
0,1	1.46915 , 0.15015	2.378558

Ya se ha visto la dependencia del algoritmo con respecto a la tasa de aprendizaje y ahora se va a ver con respecto a los puntos iniciales. Ejecutamos el algoritmo de la misma forma que antes pero usando como puntos iniciales: $\{(-0{,}5, -0{,}5), (1, 1), (2{,}1, -2, 1), (-3, 3), (-2, 2)\}$. Los resultados se presentan en la tabla 2 y en la figura 5 se observa la evolución del algoritmo.

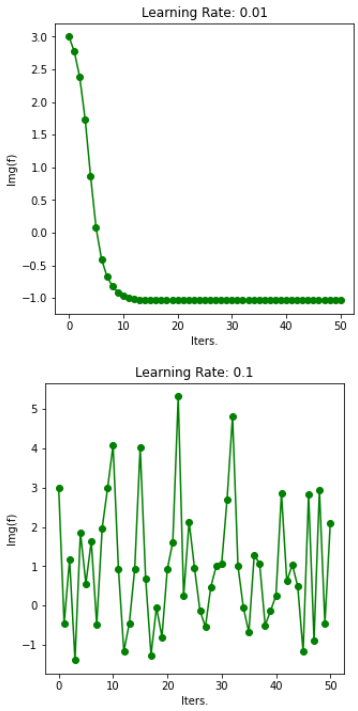
El algoritmo converge usando $\mu = 0{,}01$ para todos los puntos iniciales exceptuando el punto $w_o = (-3, 3)$, este caso es algo más dudoso y quizás hagan falta algunas iteraciones más para poder afirmarlo con rotundidad. Para $\mu = 0{,}1$ se observa que para el resto de puntos no hay convergencia pues la función va oscilando.

En cuanto a los resultados, en los puntos $(-0{,}5, -0{,}5) y (1, 1)$ se obtiene un mejor óptimo usando $\mu = 0{,}01$, en el resto de puntos, los mejores óptimos se logran usando $\mu = 0{,}1$ ya que se escapan del óptimo en el que se encuentran desplazándose así a unas coordenadas situadas por debajo del óptimo anterior. Además, no solo el algoritmo depende de la tasa de aprendizaje, sino también depende mucho de las condiciones iniciales si la función tiene muchos óptimos, pues como se ha visto, partiendo de condiciones iniciales distintas se obtienen resultados distintos incluso cuando el algoritmo converge, por lo que se puede observar la no unicidad de solución. Además, la convergencia tampoco va igual de rápido para todos los puntos iniciales, ya que por ejemplo, el algoritmo necesita menos iteraciones para converger usando como punto inicial el

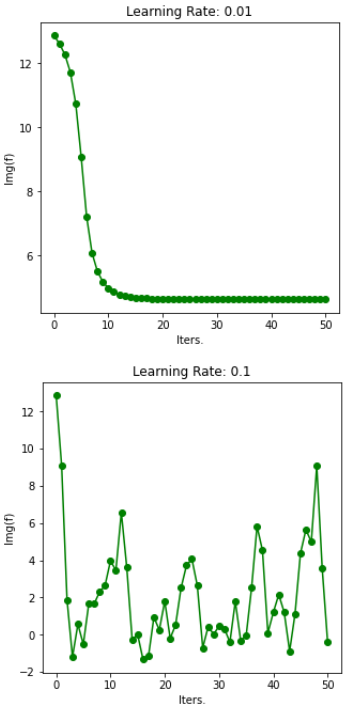
$(-0{,}5, -0{,}5)$ que por ejemplo usando $(1, 1)$.



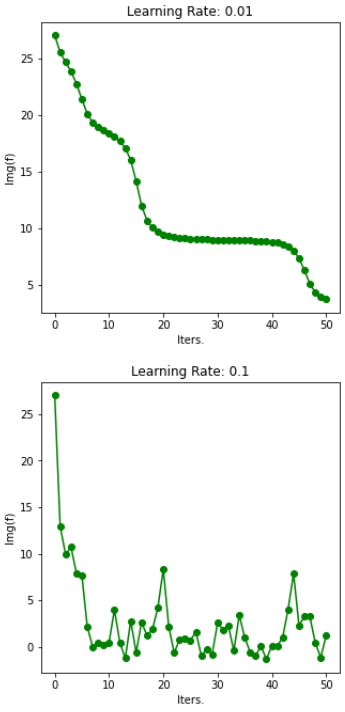
(a) $w_o = [-0{,}5, 0{,}5]$



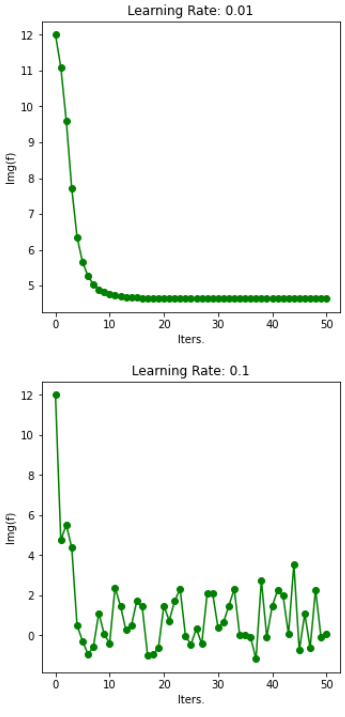
(b) $w_o = [1, 1]$



(c) $w_o = [2{,}1, -2{,}1]$



(d) $w_o = [-3, 3]$



(e) $w_o = [-2, 2]$

Figura 5: Evolución del gradiente descendente

Tabla 2: Resultados Ejercicio 1.3 (b)

Puntos Iniciales	**Learning Rate**	**Extremos**	**Imagen**
-0.5,-0.5	0.01	-0.7307,-0.4143	-1,0365
-0.5,-0.5	0.1	-0.5905,-0.5686	-0,0572
1.0,1.0	0.01	0.7307,0.4143	-1,0365
1.0,1.0	0.1	0.3294,0.4034	2,1099
2.1,-2.1	0.01	1.6651,-1.1727	4,6338
2.1,-2.1	0.1	0.8395,0.2587	-0,3899
-3.0,3.0	0.01	-2.1888,0.5868	3,6941
-3.0,3.0	0.1	-0.1554,-0.2297	1,2247
-2.0,2.0	0.01	-1.6643,1.1712	4,6337
-2.0,2.0	0.1	0.5378,0.2884	0,0846

2.4. Conclusión

Ya se ha visto el impacto de la tasa de aprendizaje y de las condiciones iniciales en el algoritmo del gradiente descendente, la dificultad de encontrar un mínimo global de una función en estos casos sería una acertada elección de los parámetros. Ya que de la tasa de aprendizaje y de los puntos iniciales depende la convergencia del algoritmo y el óptimo encontrado. Si la función es convexa no hay ningún problema, el mínimo encontrado va a ser global, en cambio, el problema pasa a ser algo más complejo cuando la función ya no lo es, por ejemplo como la del apartado anterior, donde el gradiente descendente queda atrapado en extremos locales ya que solo tiene en cuenta la información de un entorno del punto.

Las funciones con las que se han trabajado hasta ahora han sido continuas y diferenciables, propiedades deseadas por los matemáticos puesto que casi todos los estudios se centran en ese tipo de funciones. Pero...¿qué ocurre cuando la función no es diferenciable? o incluso peor, ¿y si la función no es ni siquiera continua?

En un problema de la vida real, se tienen que minimizar funciones que son totalmente desconocidas, que no son continuas y mucho menos diferenciables. Por ejemplo, en el problema de la mochila (que no deja de ser un problema de juguete) es un problema de optimización combinatoria que consiste en seleccionar un subconjunto M de m elementos ($|M| = m$) de un conjunto inicial N de n elementos (con $n > m$) de forma que se maximice la diversidad entre los elementos escogidos. Este problema es un problema de maximización, que no deja de ser un problema de minimización si cambiamos los signos. Ahora la función es totalmente desconocida, por lo que no se va a poder derivar, ni siquiera graficar y por si fuera poco, toma valores discretos haciendo que no sea ni continua. En este caso tendríamos que aplicar otro tipo de técnicas.

En estos tipos de problemas, en los que los métodos matemáticos clásicos de optimización ya no son factibles se suelen emplear lo que se conoce como metaheurísticas, las cuales juegan con una cierta componente estocástica y con técnicas para escapar de extremos locales. No hay una metaheurística que dado un problema y unos datos de entrada devuelva un óptimo global, por lo que ya se puede ver la complejidad de estos tipos de problemas de optimización. El objetivo de las metaheurísticas generalmente no es encontrar el mínimo global, si no más bien encontrar el mínimo local menos malo.

3. Ejercicio2

3.1. Ajuste lineal al problema de los dígitos simplificado

Se parte de un conjunto de datos con imágenes dos dígitos manuscritos (2 y 5) y dos características, una mide la intensidad media de gris y otra la simetría del dígito respecto de su eje vertical. Se tienen dos conjuntos de datos con sus respectivas etiquetas $\{-1, 1\}$ correspondientes a los dígitos 1 y 5 respectivamente. Uno de los conjuntos será usado para el entrenamiento y otro para el test. Dichos conjuntos tienen una característica extra para poder realizar correctamente el producto escalar con el vector de pesos a la hora de crear el modelo de regresión lineal. De este modo, las características son $[1, x1, x2]$ siendo $x1$ y $x2$ las características comentadas anteriormente. Para estimar el modelo se van a usar dos algoritmos, el primero es el de la Pseudoinversa y el seguendo el el gradiente descendente estocástico (SGD)

Se aplica primero el algoritmo de la pseudo-inversa. Este algoritmo parte de la función de error

$$E_{in}(w) = \frac{1}{N}||Xw - y||^2 \tag{3}$$

siendo N el número de datos de entrenamiento, X el conjunto de características de los datos de entrenamiento, y las etiquetas y w el vector con los pesos que caracterizan a la función lineal la cual se va a buscar. Tras calcular el gradiente, igualarlo a 0 y despejar se llega a

$$w = \hat{X}y \tag{4}$$

donde $\hat{X} = (X^T X)^{-1} X^T$ es la llamada pseudo-inversa de X. Obviamente, este método se puede aplicar únicamente cuando la matriz X es regular y cuando el número de ejemplos que tengamos no sea lo suficientemente elevado, ya que computacionalmente se volvería muy costoso. En caso de aplicarse, se obtendrían los pesos óptimos, esto es, se conseguiría la función lineal que mejor se ajusta a los datos de entrenamiento. En la implementación se realizó una descomposición en valores singulares de X, obteniendo X como producto de tres matrices, una diagonal y otras dos ortogonales, y a partir de ahí se calculó la pseudoinversa.

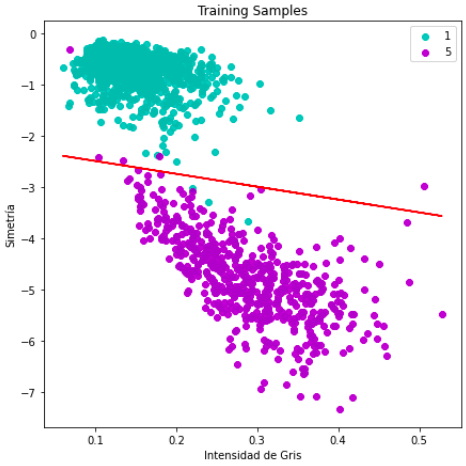
Tras aplicar el algoritmo al conjunto de entrenamiento se obtienen unos pesos

$$w = [-1{,}11588016 - 1{,}24859546 - 0{,}49753165]$$

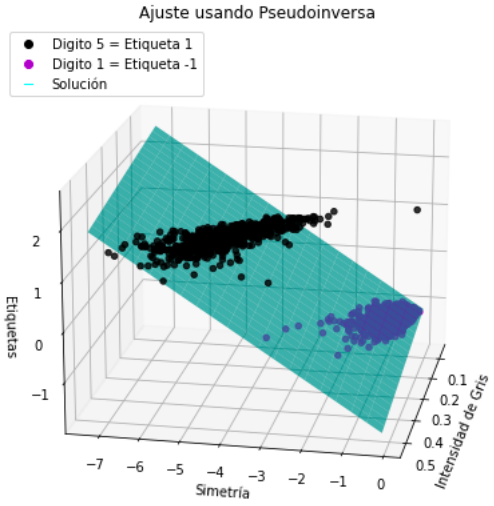
originando un hiperplano que visualmente se aprecia en la figura 6 y es el óptimo, es decir, no hay ningun hiperplano mejor que este en lo que respecta a los datos de entrenamiento, pero no tiene por qué ser así también para los datos de test. Por otra parte en la tabla 3 se aprecian los resultados del experimento. Se tiene que el error dentro de la muestra es bastante pequeño, aproximadamente *cero*. El error fuera de la muestra, aunque sigue siendo pequeño es ligeramente superior al otro.

Tabla 3: Resultados del algoritmo de la Pseudoinversa

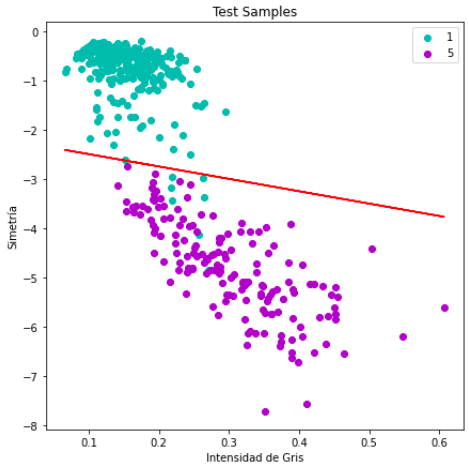
E_{in}:	0.07918658628
E_{out}:	0.13095383720



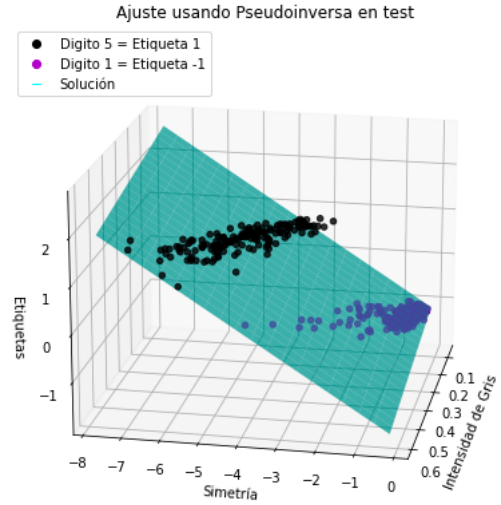
(a) Proyección Z=0



(b) 3D



(c) Proyección Z=0



(d) 2D

Figura 6: Separación obtenida tras aplicar el método de la pseudoinversa en el conjunto de datos tanto para los de entrenamiento (Arriba) como los de test (Abajo).

A continuación se implementa el algoritmo del SGD, que básicamente consiste en aplicar en cada iteración el algoritmo del GD a un subconjunto aleatorio reducido de la población (minibatch, minilote). En un modelo de regresión lineal, la función de error es el error cuadrático medio (ECM) que viene dado por la ecuación 3 y cuyo gradiente es

$$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y) \tag{5}$$

Vamos a estudiar el comportamiento del algoritmo en función de los tamaños del lote, para ello fijamos la tasa de aprendizaje a 0,01, no usamos una tasa mayor para no propiciar inestabilidad al algoritmo ni tampoco una más pequeña para que no tarde demasiado en converger. Tras lo cual, probamos los siguientes valores para el tamaño del minilote, $\{2, 8, 16, 64, 128, 256, tam(X)\}$. Se han ejecutado durante 200 épocas, un número razonable para asegurar la convergencia, al menos en un entorno del óptimo lo suficientemente pequeño. Hemos usado esos tamaños de minilote porque se suelen usar pontencias de dos y el último tamaño del minilote por ser el total de ejemplos a entrenar, por lo que solo habrá un único minilote.

Tabla 4: Resultados usando SGD con distintos tamaños del batch y 200 épocas

Batch size	**Pesos**	**Ein**	**Eout**
2	-1.1194,-1.2574,-0.4674	0,0879	0,1327
8	-1.1495,-1.2661,-0.4750	0,0887	0,1338
16	-1.1609,-1.2484,-0.5174	0,0808	0,1357
64	-1.1113,-1.2572,-0.5012	0,0793	0,1321
128	-1.0973,-1.2443,-0.5104	0,0821	0,1383
256	-1.1200,-1.2461,-0.4912	0,0796	0,1297
1561	-1.0223,-0.0768,-0.3943	0,1077	0,1555

Los resultados quedan reflejado sen la tabla 4 en la cual podemos observar que son bastante buenos ya que tenemos un E_{in} y un E_{out} bastantes pequeños. Hay discrepancia usando distintos tamaños del batch, el tamaño de batch 64 es el que mejor E_{in} presenta, acercándose al error dentro de la muestra que proporciona el método de la pseudoinversa. En cambio, para un tamaño de batch de 256 tenemos el mejor E_{out}, incluso mejor que en el método de la pseudoinversa. Para un tamaño de batch de 1561 tenemos los peores resultados. En general, los pesos son muy similares entre sí, lo que provoca también hiperplanos y resultados bastantes parecidos en casi todos los puntos.

Comentar que los resultados obtenidos tanto en el método de la pseudoinversa como en el gradiente descendente estocástico son muy similares. Se aprecia que el Error dentro de la muestra usando el método de la pseudoinversa es menor estricto que todos los errores dentro de la muestra proporcionados por el algoritmo del gradiente. Esto es lógico, ya que el ajuste de la pseudoinvesa es el óptimo. En cambio, el error fuera de la muestra es menor en el caso del SGD (usando un tamaño de minilote 256) , que en el método de la pseudoinversa. Esto no es extraño que ocurra, ya que el algoritmo de la pseudoinversa da el óptimo para una muestra dada, no para la población en general.

3.2. Regresión Lineal II

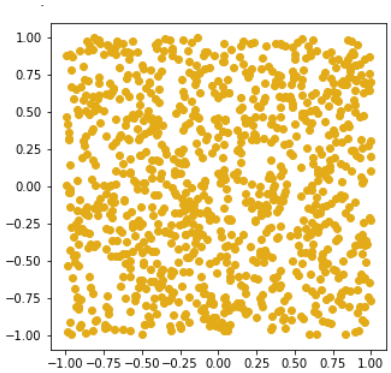
En este apartado se va a estudiar cómo los errores E_{in} y E_{out} se transforman cuando aumenta la complejidad del modelo lineal empleado. Vamos a generar una muestra de entrenamiento de mil puntos en el cuadrado $\mathcal{X} = [-1, 1] \times [-1, 1]$ (7a) y las etiquetamos de acuerdo a la función $f(x_1, x_2) = sign((x_1 - 0{,}2)^2 + x_2^2 - 0{,}6)$ añadiendo además un 10 % de ruido cambiando de manera aleatoria el signo de algunas (7b). Tras lo cual aplicamos un modelo de regresión lineal, para ello añadimos una coordenada con un 1 a cada par (x_1, x_2) y se aplica el SGD con $lr = 0{,}01$, $epocas = 200$ y $tam\_minilote = 64$.

Aunque la elección de los hiperparámetros depende del problema, se optado por estos y como se verá un poco más adelante, proporcionan los resultados esperados. Una tasa de aprendizaje mas alta puede ocasionarnos inestabilidad y una más baja, lentitud en la convergencia. Las épocas usadas han sido esas ya que el problema dado tiene pocos datos y se ha considerado que son suficientes como para que se dé la convergencia. El tamaño del minilote debe de ser bastante pequeño en comparación con el número total de datos y en vista del apartado anterior, hemos optado por 64 ya que no hay mucha diferencia entre usar unos u otros. Se ejecuta el algoritmo obteniendo así el hiperplano de regresión de la figura (7c) con unos pesos de

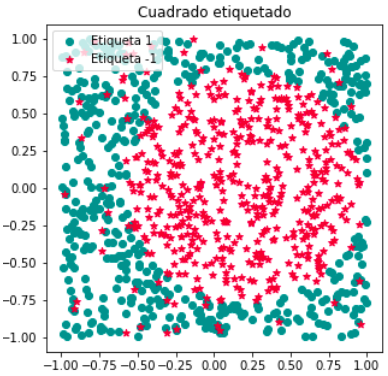
$$w = [-0{,}05895169 - 0{,}59872622 - 0{,}00308709]$$

y un error de

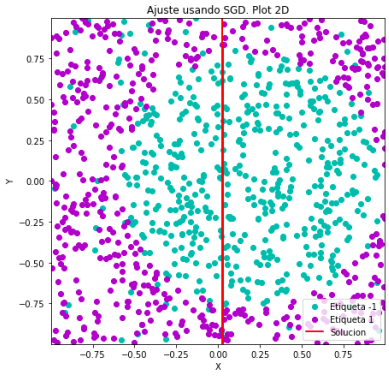
$$E_{in} = 0{,}8825263$$



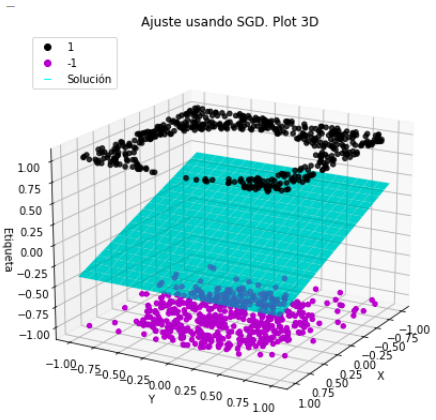
(a) Muestra



(b) Muestra etiquetada y con ruido



(c) Ajuste tras aplicar regresión lineal usando SGD. Proyección plano Z=0



(d) Ajuste tras aplicar regresión lineal usando SGD.

Realizando el mismo experimento 1000 veces usando muestras aleatorias nuevas y generando también para cada prueba un conjunto de test de tamaño 1000, tenemos que la media de los errores dentro y fuera de la muestra son,

$$MediaE_{in} = 0{,}907844 \quad MediaE_{out} = 0{,}913466$$

los cuales son bastantes altos, siendo esto un indicador de que un modelo lineal no es lo más apropiado para este conjunto de datos. De hecho, los datos se han etiquetado con una función que no es lineal, si no cuadrática, por lo que de ninguna manera vamos a poder aproximarla usando funciones lineales. Esto mismo se aprecia de manera visual en la figura 7c, en la que apreciamos dos regiones, una es la región interna de la circuferencia y otra la región externa. El hiperplano de regresión, al ser lineal, no puede ajustarse a la cirfurencia originando así unos errores tan altos.

Como alternativa a lo anterior tenemos dos opciones. La primera es probar un modelo no lineal y la segunda es hacer un modelo lineal, pero usando como características las resultantes de aplicar operaciones no lineales a las características de entrada iniciales. De este modo, dado un vector $x = (1, x_1, x_2)$ definimos $\phi(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ y usamos estas nuevas características. Volvemos a aplicar el SGD con exactamente los mismos parámetros que los experimentos anteriores y a una muestra aleatoria de tamaño 1000 sobre el cuadrado $[-1, 1] \times [-1, 1]$

etiquetadas con la función usada anteriormente y añadiendo un 10 % de ruido obteniendo así los pesos

$$w = [-1{,}05625958 \ ; -0{,}4877254 \ ; -0{,}01507462 \ ; -0{,}03012445 \ ; 1{,}32984383 \ ; 1{,}71781246]$$

con un Error dentro de la muestra de

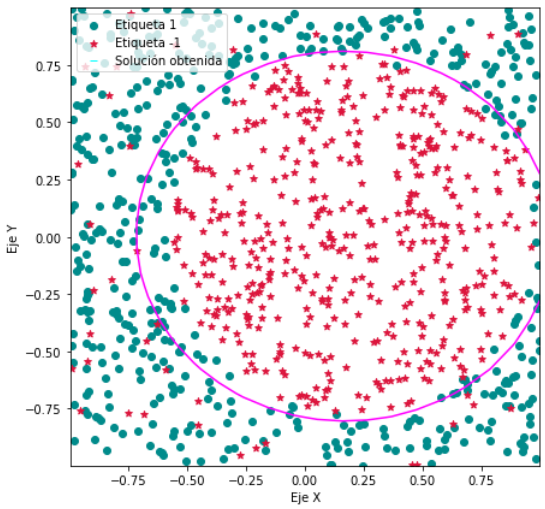
$$E_{in} = 0{,}49662$$

de manera visual ahora podemos apreciar las soluciones gracias a las figuras 8a y 8b. La segunda figura (Fig. 8b) nos muestra en un espacio de tres dimensiones el conjunto de datos donde el valor de la etiqueta está en el Eje Z. En esta figura se ve que el hiperplano de regresión es un paraboloide y no un plano como en el caso anterior, todo gracias al uso de características no lineales. En la primera figura (Fig. 8a) tenemos los puntos proyectados en el plano $Z = 0$ y la curva de nivel también en $Z = 0$ de la superficie anterior. Se aprecia que la solución se ajusta mucho mejor a la circunferencia que contiene las etiquetas -1 que en el caso anterior.

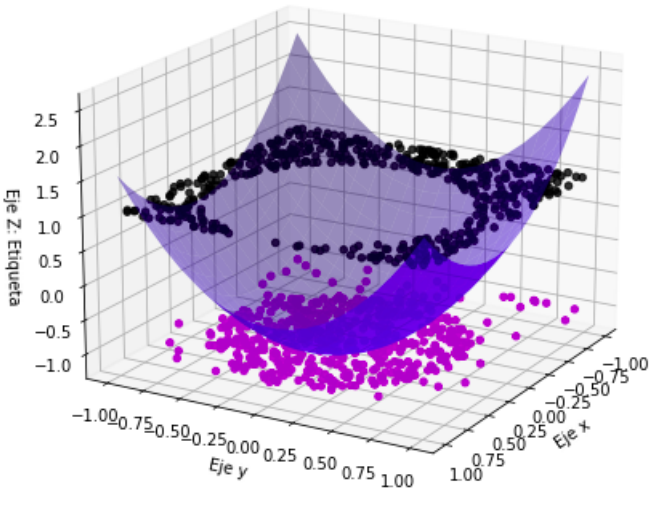
Como antes, repetimos el experimento anterior unas 1000 veces y en cada repetición generamos otro conjunto con las mismas características que el primero con el fin de usarlo como datos de test. Dicho lo cual, obtenemos que las medias de los errores dentro y fuera de la muestra son

$$Media\_E_{in} = 0{,}4665 \quad Media\_E_{out} = 0{,}47044$$

A modo de conclusión, hemos realizado regresión lineal a un conjunto de datos etiquetados con una función cuadrática y añadiendo un poco de ruido. El primer modelo de regresión se ha llevado a cabo con las características originales de los datos, dando lugar a unos resultados bastante malos, pues la solución no se ha ajustado para nada a los datos. El segundo modelo de regresión lineal se ha llevado a cabo usando características no lineales obtenidas a partir de las iniciales, como resultado tenemos una mejor aproximación y una reducción significativa del error con respecto al modelo anterior. Por tanto, afirmamos con rotunda seguridad que el segundo modelo es mejor que el primero para el problema planteado.



(a) Curva de nivel en Z=0



(b) Hiperplano de regresión visto en 3D

4. Bonus

Vamos a implementar el algoritmo de minimización de Newton y se lo vamos a aplicar a la función del ejercicio 3. Este método actualiza los pesos de acuerdo a la igualdad 6, donde w_k son los pesos en la iteración k, f es la función a minimizar y $Hess$ es la matriz Hessiana de f.

$$w_{k+1} = w_k - Hess^{-1}\nabla f(w_k) \tag{6}$$

donde W_k son las coordenadas obtenidas en la k-ésima iteración del algoritmo, f es la función sobre la que vamos a trabajar y H es la matriz Hessiana de la misma.

En particular, la función a miminizar es

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

cuyo gradiente viene dado por la ecuación 2.3 y cuya matriz Hessiana es

$$Hess(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(x,y) & \frac{\partial^2 f}{\partial x \partial y}(x,y) \\ \frac{\partial^2 f}{\partial x \partial y}(x,y) & \frac{\partial^2 f}{\partial y^2}(x,y) \end{pmatrix} = \begin{pmatrix} 2 - 8\pi^2\sin(2\pi x)\sin(\pi y) & 4\pi^2\cos(2\pi x)\cos(\pi y) \\ 4\pi^2\cos(2\pi x)\cos(\pi y) & 4 - 2\pi^2\sin(2\pi x)\sin(\pi y) \end{pmatrix}$$

ocurre que $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$ por ser la función f de segundo orden.

A continuación ejecutamos para los puntos iniciales $(-0{,}5, -0{,}5), (1,1), (2{,}1, -2{,}1), (-3,3), (-2,2)$ este algoritmo junto con el gradiente descendente con unos learning rates de 0,01 y de 0,1 y unas 50 iteraciones. La siguiente tabla (5) muestra los puntos iniciales, los learnings rate usados, los extremos y el valor de la función que alcanzan usando el método de Newton. Los resultados también se muestran gráficamente en la figura 9

Tabla 5: Resultados de aplicar el algoritmo de Newtion

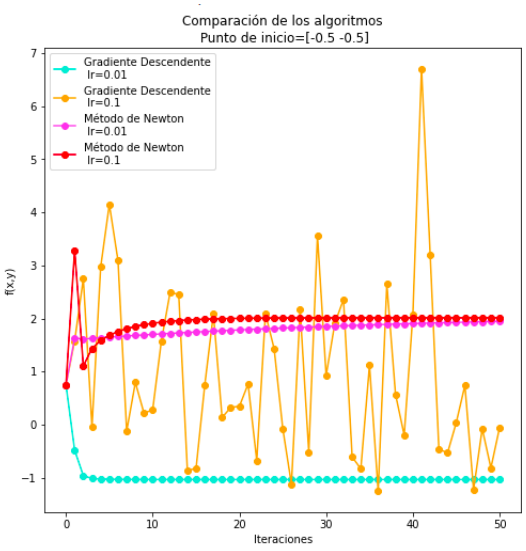
P. iniciales	**lr**	**Extremos**	**Imagen**
[-0.5,-0.5]	0.01	[-0.4899 -0.9144]	1,94608
[-0.5,-0.5]	0.1	[-0.1105,-1.0063]	2,01217
[1,-1]	0.01	[1.0471 ,1.02551]	3,15310
[1,1]	0.1	[1.1259 ,1.0820]	3,24677
[2.1,-2.1]	0.01	[2.2130 ,-2.2170]	13,50172
[2.1,-2.1]	0.1	[1.7252,0.4345]	1,41994
[-3,3]	0.01	[-2.8523, 2.8983]	25,43898
[-3,3]	0.1	[-1.898 , 0.8778]	5,59137
[-2,2]	0.01	[-2.0826, 2.0466]	12,56980
[-2,2]	0.1	[-1.6974 , 1.2276]	4,65492

Notamos que el método de Newton no da buenos resultados y en función del punto de partida encuentra o no un mínimo. De hecho, este método no está pensado para buscar mínimos, sino puntos críticos que pueden ser además de mínimos, máximos y puntos de silla. De hecho, para los puntos de partida cercanos a un máximo o a un punto de silla, este método tenderá a buscar dichos puntos críticos en vez de un mínimo. Esto se puede ver por ejemplo para el caso del punto $(1, 1)$ ya que el extremo que encuentra, el cual no es un mínimo (se puede apreciar visualmente en la Fig. 9a), está muy cercano a dicho punto. Además, si las derivadas parciales fueran muy pequeñas, al calcular la inversa de la matriz Hessiana se tendría la posibilidad de que ésta explotase y fuera hacia un punto bastante alejado, lo cual no ocurre en estos casos.

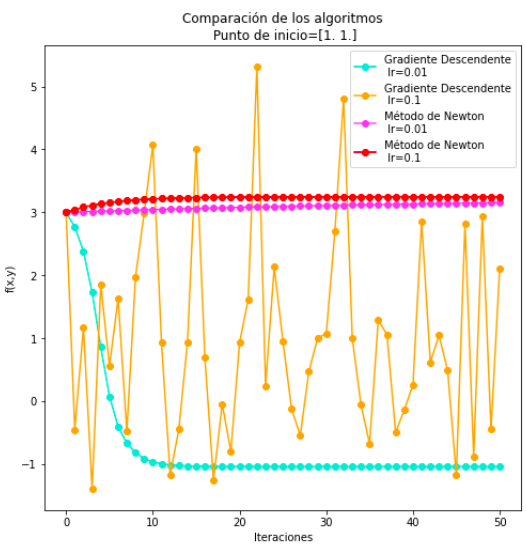
En las gráficas de más abajo, vemos que si usamos una tasa de aprendizaje de 0,01 en este método el resultado final es bastante pésimo pues se queda estancado en un entorno del punto inicial. En cambio, para una tasa de aprendizaje más alta, vemos cambios bruscos para los puntos $(2{,}1, -2{,}1)$, $(-3, 3)$ y $(-2, -2)$, gracias a que escapa de los entornos de los óptimos en los que está llegando así a un mínimo. Para el resto de puntos, el método de Newton con esta tasa de aprendizaje llega al mismo óptimo local que usando la otra tasa y en ningún caso es un mínimo.

Este método no está pensado como sustituto del gradiente descendente, si no para aplicarse después, en caso de que éste no sea capaz de llegar al extremo, como por ejemplo pasa en el ejercicio 1.2. Este método se aplica cuando estemos en un entorno cercano al extremo y desafortunadamente, eso no lo vamos a saber con certeza como en la mayoría de los casos.

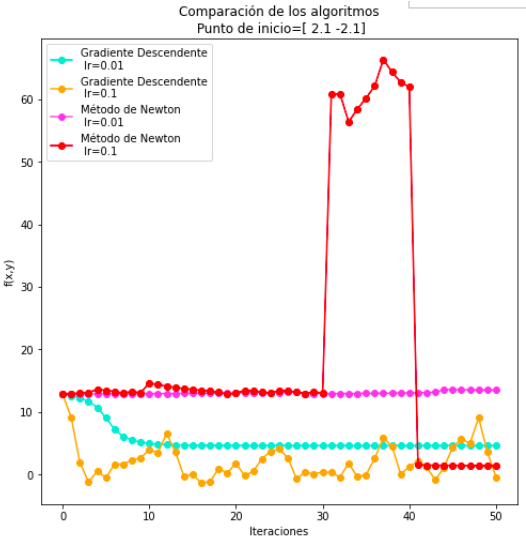
El método presenta la desventaja típica del cómputo de la inversa de una matriz, que para un espacio de dimensión elevada, puede llegar a ser computacionalmente costosa. Por otra parte, tenemos que este método usa más información adicional de la función, como la curvatura, a través de la Hessiana y con esto conseguimos tomar un camino más directo al óptimo que usando el gradiente descendente, y como consecuencia, se necesitan muchas menos iteraciones para converger.



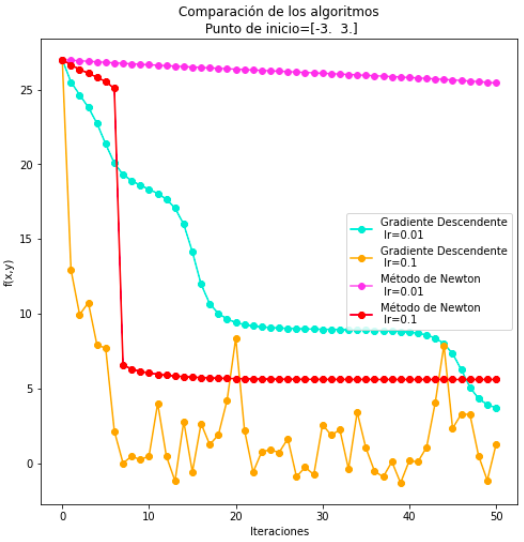
(a) $w_o = [-0{,}5, 0{,}5]$



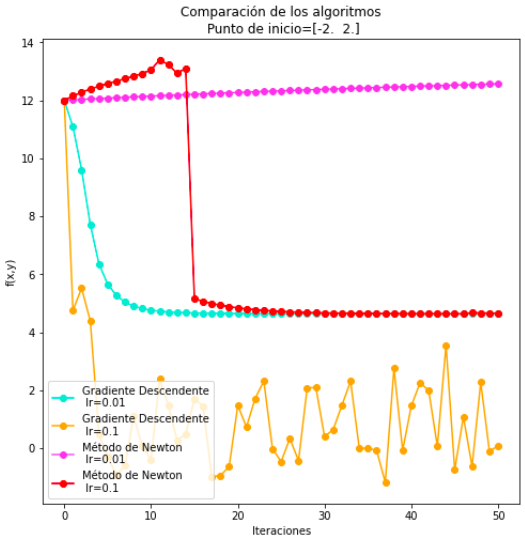
(b) $w_o = [1, 1]$



(c) $w_o = [2{,}1, -2{,}1]$



(d) $w_o = [-3, 3]$



(e) $w_o = [-2, 2]$

Figura 9: Evolución del método de Newton y el gradiente descendente