PRÁCTICA 1:

Búsqueda Iterativa de Óptimos y Regresión Lineal

Aprendizaje Automático

Ángel Cabeza Martín

1. Ejercicio sobre la búsqueda iterativa de óptimos

1.1. Implementar el algoritmo de gradiente descendente.

Para implementar este algoritmo, he usado su ecuación general:

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

En esta ecuación nos encontramos los siguientes elementos:

1. **wj:** Es el punto en el cual nos encontramos en la iteración actual del algoritmo. Al principio toma un valor asignado por el programador, más adelante veremos la importancia de elegir de manera correcta el punto de inicio.
2. **η:** Denominado tasa de aprendizaje (learning rate). También designada por el programador. Nos indica la "velocidad" con la que el algoritmo se va ajustando por iteración a lo que quiera optimizar.
3. $\frac{\partial Ein(w)}{\partial wj}$**:** es el gradiente de la función,es decir, las derivadas parciales de w respecto a sus dos parámetros (u,v en este ejercicio).

Un apunte muy importante en esta fórmula, es que el signo del producto de la tasa de aprendizaje y las derivadas parciales es negativo. Esto indica que estamos descendiendo por la función, es decir, queremos minimizar. Si el signo fuese positivo estaríamos ascendiendo y por tanto maximizando.

Una vez comprendida la ecuación, solo queda pensar en cómo pasarla a código. Lo primero que hay que decidir son los parámetros que le tendremos que pasar al algoritmo.Hay dos parámetros que son muy claros: la tasa de aprendizaje y el punto inicial. Estas dos variables las tiene que "saber" el algoritmo antes de iniciarse y son elegidas por el programador teniendo un cierto criterio (aunque sea por la experiencia del ensayo y error y no matemático) por lo tanto es lógico pensar que pueden variar según lo que optimizas y por tanto deben de ser dos parámetros del algoritmo. Los otros dos parámetros que he elegido tienen que ver con el criterio de parada del algoritmo, es decir, ¿cuándo debería parar el algoritmo?. He decidido que el algoritmo debe parar cuando alcance un número determinado de iteraciones para que no se tire de manera indefinida ejecutándose o cuándo se llegue a un error aceptable. Por ejemplo: si estamos minimizando una función y alcanzamos el mínimo absoluto de la función (pongamos que el mínimo es 0.5) en la iteración 50000 pero a partir de la iteración 1000 el algoritmo toma valores muy cercanos a 0.5, es posible que nos salga rentable ahorrarnos 4000 iteraciones ya que el error es tan pequeño que no tiene efectos prácticos visibles.

Teniendo en cuenta estos criterios comentados anteriormente, el pseudocódigo del algoritmo es el siguiente:

```
def gradient_descent (initial_point, learning rate, error2get, maxIter)
      w = initial_point
      it = 0

      while ( valor_funcion > error2get and iteracion < maxIter)
            w = w - learning_rate * derivada_parcial_de_w
            iterations += 1
      end

      return w, it

end
```

Donde valor_funcion será el valor que toma la función que estamos optimizando en el punto actual de la iteración y derivada_parcial_de_w contendrá el valor de las derivadas parciales respecto a u y v (x e y). Devolvemos el punto en el que hemos encontrado el mínimo de la función y el número de iteraciones porque esta información nos resulta útil para saber si tanto el learning rate como el initial point que hemos escogido son buenos (el mínimo se alcanza en un número razonable de iteraciones) y para saber cúal es el mínimo de la función (a partir del punto mínimo de la función podemos obtener el valor del mínimo).

La idea del algoritmo es empezar en un punto inicial dado y mientras el valor de la función (el error) sea mayor que el que queremos encontrar o aún no hayamos llegado al número máximo de iteraciones. le restamos al punto que tenemos en este momento el valor del gradiente por el learning rate escogido, es decir, bajamos por la función en función del learning rate.

1.2 Buscar un mínimo de una función

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 la función E(u,v) es la derivada respecto de u de E y la derivada de v de E. Por lo tanto para mostrar la expresión del gradiente necesitaremos calcular estas dos derivadas.

La derivada de E(u,v) respecto a u es la siguiente:

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

La derivada de E(u,v) respecto a v es la siguiente:

$$\frac{d}{d(v)} = 2(u^3 e^{(v-2)} - 2v^2 e^{-u}) * (u^3 e^{(v-2)} - 4ve^{-u})$$

Al ser E una potencia de orden dos, he usado la regla de derivación de la potencia:

$$f(x) = u^k \Rightarrow f'(x) = k * u^{k-1} * u'$$

Como se puede observar en ambas derivadas encontramos un 2 (antes era la potencia) multiplicando a E(u,v) que sería $k * u^{k-1}$ y esto multiplicando a lo que sería u'.

Esta u' es lo que más cambia respecto a una derivada y otra pero en las dos derivadas se utilizan las mismas reglas de derivación; la de la potencia, la de la función exponencial y la del producto de dos funciones.

Derivada de la función exponencial base e:

$$f(x) = e^u \Rightarrow f'(x) = u' * e^u$$

Derivada del producto de funciones:

$$f(x) = u(x) * v(x) \Rightarrow f'(x) = u'(x) * v(x) + u(x) * v'(x)$$

Lo único que es resaltable es que al derivar e^{-u} ,respecto a u, el signo cambia a ser negativo y por eso cambia el signo en la derivada.

b)¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u,v) inferior a 10^{-14}. (Usar flotantes de 64 bits)

Para realizar este experimento lo que he hecho es igualar el error a buscar a 10^{-14} y la variable que indica el máximo de iteraciones igualarla a un número muy grande para que la condición de parada sea casi exclusivamente cuando alcanzamos el error indicado.

Mi algoritmo ha encontrado un valor de E(u,v) inferior a 10^{-14} en la iteración 10, ha necesitado pocas iteraciones para llegar a este valor por lo que podemos pensar que el learning rate que hemos escogido (0.1) es idóneo para el objetivo que nos hemos marcado, pero de esto hablaremos en los siguientes apartados en los que profundizaremos más en este tema.

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

Este valor se alcanzó con las siguientes coordenadas:

(1.1572888496465497 , 0.9108383657484797)

donde el primer término corresponde al valor de x (u en nuestro caso) y el segundo al valor de y (v en nuestro caso).

Hemos alcanzado el objetivo con una coordenada muy parecida al punto inicial, así que podemos asegurar que hemos hilado muy fino a la hora de escoger el punto de inicio del algoritmo. Si hubiéramos escogido este punto de inicio un poco más alejado (por ejemplo en el punto (1.5,1.5)) tardaríamos algunas iteraciones más. Para este caso se tardarían 75 iteraciones pero encontraríamos un mínimo algo mejor. Sin embargo como en este experimento estamos buscando un valor menor a 10^{-14}nos convendría más dejar el punto inicial donde está y ahorrarnos muchas iteraciones, ya que está muy cerca de nuestro objetivo.

1.3 Importancia de la tasa de aprendizaje y del punto inicial

Considerar ahora la función f(x,y) = (x+ 2)2+ 2(y−2)2+ 2sin(2πx)sin(2πy)

Ahora vamos a cambiar de función, así que si queremos hacer experimentos con esta función lo primero que tenemos que hacer es calcular la expresión del gradiente de la función, es decir, la derivada de la función respecto x e y (en el código (x = u) e (y = v) para ahorrarnos nuevas funciones):

.

Derivada respecto de x:

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

Derivada respecto de y:

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

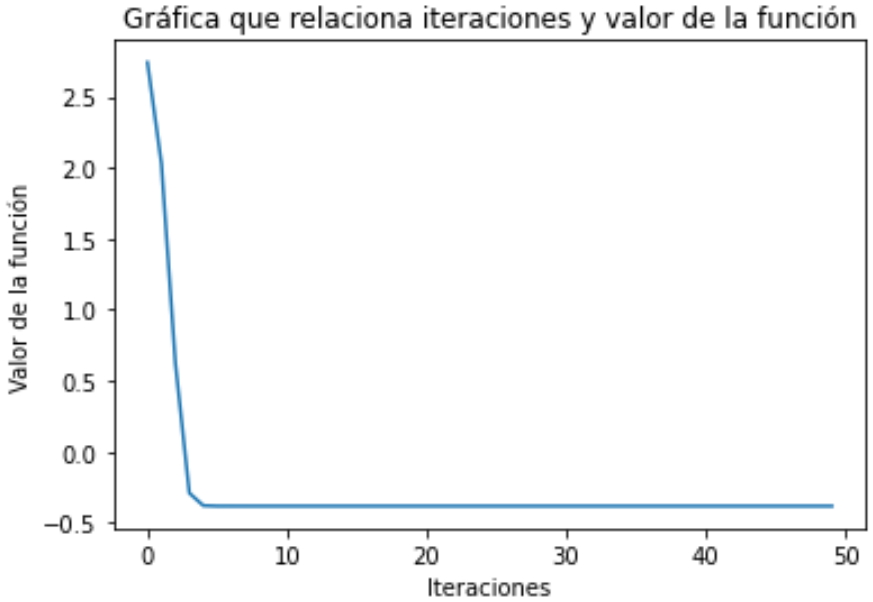
Para calcular estas derivadas hemos usado una regla de derivación nueva. La de la función seno, que se define así.

$$f(x) = sin(x) \Rightarrow f'(x) = cos(x)$$

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial (x0=−1,y0= 1), (tasa de aprendizaje η= 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 η= 0,1, comentar las diferencias y su dependencia de η.

Tras realizar estos dos experimentos he obtenido estas dos gráficas.

$\eta = 0,01$



$\eta = 0,1$

En la primera gráfica podemos ver que el algoritmo ha ido descendiendo por la gráfica a una velocidad considerable y ha encontrado un mínimo en la iteración 4. Sin embargo, a partir de ahí y hasta el final no ha sido capaz de salir de ese mínimo local. Esto se debe a que el learning rate escogido es pequeño y como vimos en teoría esto puede hacer que el algoritmo se quede estancado en un mínimo relativo y que le lleve muchas iteraciones salir de ahí o que directamente no salga.

En la segunda gráfica vemos como el valor de la función sube y baja con el paso de las iteraciones. Esto se debe a que el learning rate es alto y en vez de explorar la zona en la que está, el algoritmo se ha dedicado a explorar la función dejando atrás los mínimos que ha encontrado. Además gracias a esta gráfica podemos ver que el mínimo que encontramos con un learning rate de 0.01 es un mínimo local y no global lo que confirma nuestras sospechas.

b) Obtener el valor mínimo y los 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.

Punto inicial	Mínimo	Coordenadas
(−0,5,−0,5)	9.125146662901855	[-0.79349947 -0.12596576]
(1,1)	6.4375695988659185	[0.67743878,1.29046913]
(2,1,−2,1)	12.490971442685037	[0.14880583, -0.0960677]
(−3,3)	-0.38124949743809955	[-2.73093565 ,2.71327913]
(−2,2)	-4.799231304517944e-31	[-2. 2]

En esta tabla podemos ver como la elección de un punto inicial influye en gran medida en los resultados obtenidos. Ya que dependiendo de este punto de inicio encontraremos un mínimo local más o menos bueno o un mínimo global, menos cuando la función sea cóncava que siempre encontraremos un mínimo global.

1.4 Conclusión final

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

La dificultad de encontrar un mínimo global en una función arbitraria es muy alta, y depende principalmente de los siguientes factores:

1. Tasa de aprendizaje (learning rate): como hemos visto si el learning rate es pequeño, se quedará estancado en un mínimo local y si es muy grande explorará la función en vez de buscar un mínimo en una zona.
2. Punto de inicio: este parámetro nos indicará la zona que vamos a explorar, si no lo elegimos bien y empezamos en una zona muy alejada del mínimo global nos costará mucho encontrar el mínimo y probablemente solo encontremos un mínimo local.
3. De la función que estemos optimizando: además los dos factores anteriores también dependen de lo que estemos optimizando, según la función que optimicemos, escoger un learning rate grande o pequeño puede penalizar más o menos a la hora de buscar un mínimo.

Por lo tanto encontrar un mínimo global en una función arbitraria no es sencillo y se requiere un gran conocimiento del problema para poder configurar bien los parámetros del algoritmo.

2. Ejercicio sobre regresión lineal

Ejercicio 1

Estimar un modelo de regresión lineal a partir de los datos proporcionados por los vectores de características (Intensidad promedio, Simetria) 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 Ein y Eout(para Eout calcular las predicciones usando los datos del fichero de test).

SGD

La filosofía del SGD es la misma que el GD, con la única diferencia de que el SGD solo considera una parte del espacio de búsqueda para ir calculando el vector de pesos. A estas pequeñas partes del espacio de búsqueda las llamamos minibatchs. Estos minibatchs puede ser de tamaño variable, aunque suelen ser de un tamaño entre 32-256. Yo he elegido un tamaño de 64. En nuestro ejemplo, el vector de características es una matrix con 3 columnas, por lo que nuestros minibatchs serán una matriz de 64x3.

Para implementar el SGD he seguido la misma fórmula que para el GD con una función conocida.

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

Sin embargo, ahora al ser la función desconocida he usado como función error la función de mínimos cuadrados. Por lo tanto ahora:

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

En esta fórmula estamos calculando la derivada de la función de mínimos cuadrados. Los elementos que están implicados aquí son:

- M: es el tamaño del minibatch del que ahora hablaremos más en profundidad.
- x_{nj}: es un valor concreto del minibatch
- h(x): es el producto de la traspuesta del vector de pesos (w^T) por el vector de características.
- y: es el vector de etiquetas asociado al vector de características.
-

Una vez identificados todos los elementos que participan vamos a hablar de la implementación del algoritmo. La implementación que he realizado del algoritmo es la siguiente:

```
# Gradiente Descendente Estocastico
def sgd(x,y,learning_rate,num_batch,maxIter):

    # el tamaño de w será dependiendo del numero de columnas de x
    # shape[1] == columnas
    # shape[0] == filas
    w = np.zeros(x.shape[1])

    iterations = 1


    # en este caso solo tenemos de condicion las iteraciones
    while (iterations < maxIter ) :


        # Mezclamos x e y. Esta función solo cambia el orden de la matriz
        # el contenido no lo mezcla, es decir, si la columna 4 contiene los
        # valores 5 y 3, ahora puede que sea la columna 15 pero seguirá conteniendo
        # los mismos valores. Además lo hace en relación y para que aunque esté
        # todo mezclado a cada punto le siga correspondiendo su etiqueta
        utils.shuffle(x,y,random_state=1)


        # En este bucle vamos a crear tantos minibatchs como le hemos indicado
        # y vamos a aplicar la ecuación general para cada minibatch
        for i in range(0,num_batch):

            # Cogemos de x e y las filas que van desde i * tam_batch hasta i*tam_batch+tam_batch
            # p.ej si tam_batch = 64 cogeremos las filas 0-64, luego 64,128 y así
            minibatch_x = x[i*tam_batch:i*tam_batch+tam_batch]
            minibatch_y = y[i*tam_batch:i*tam_batch+tam_batch]

            w = w - learning_rate*dErr(minibatch_x,minibatch_y,w)


        iterations = iterations + 1

    return w
```

Lo primero que hacemos en este algoritmo es inicializar w (vector de pesos que queremos calcular) a tantos 0 como columnas tenga nuestro vector de características. Una vez hecho esto, comenzamos a iterar hasta que se cumpla la condición de parada de llegar a un máximo de iteraciones (en este algoritmo no se especificaba ningún criterio de parada y en clase decidimos que nuestro criterio de parada iban a ser las iteraciones, pero podría haber otros como llegar a un error concreto). Dentro de este bucle mezclamos x e y para que en cada iteración empiece con índices distintos y iteramos en el número de minibatches que hayamos indicado (tam_muestra/tam_minibatch).

Una vez dentro de este segundo bucle formamos el minibatch cogiendo índices de 64 en 64 de la muestra (recordad que esta muestra está mezclada aleatoriamente así que es como si los estuviéramos cogiendo aleatoriamente) y aplicamos la fórmula vista anteriormente.

Para calcular la derivada del error he hecho el siguiente algoritmo, que realiza la fórmula vista anteriormente con operaciones matriciales.

```
def dErr(x,y,w):
    h_x = x.dot(w.T)
    dErr = h_x - y.T
    dErr = x.T.dot(dErr)
    dErr = (2 / x.shape[0]) * dErr
    return dErr.T
```

Lo primero que hacemos es calcular h(x) como la multiplicación de la traspuesta de w por x (siendo x el minibatch no la muestra entera). Esta operación nos dará como resultado una matriz 3x1, seguidamente a h(x) le restamos el valor de las etiquetas y esto lo multiplicamos por x de nuevo. Tras realizar estas operaciones tendremos una matriz 3x1 en la que en cada elemento tendremos: $\sum_{n=1}^{M} x_{nj} * (h(x) - y)$. Por lo tanto ahora tenemos que multiplicar cada elemento por $\frac{2}{M}$para obtener su derivada. Finalmente, devolvemos la traspuesta de esta matriz para tener un vector de 3 elementos.

PseudoInversa

El algoritmo de la pseudoinversa se basa en que si nosotros tenemos unos datos de entrada X y unos datos de salida Y, asociados a X, debe existir una función que aplicada a X, transforme X a Y.

Para implementar el algoritmo de la pseudoinversa, he seguido esta fórmula:

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y}.$$

Siendo

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

Y este es el algoritmo que he realizado.

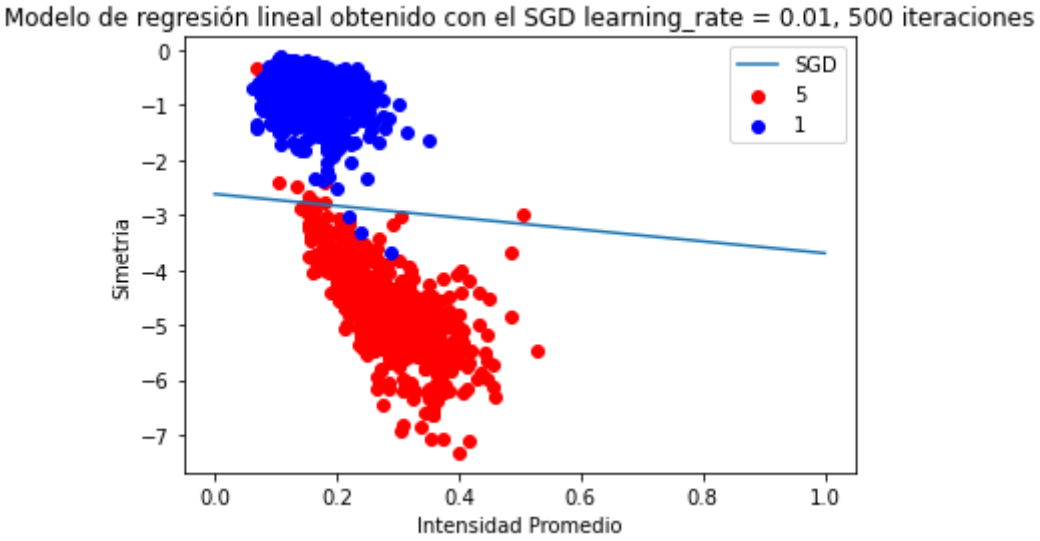
```python
# Pseudoinversa
def pseudoinverse(x,y):

    x_traspuesta = x.T
    y_traspuesta = y.T

    x_pseudoinversa = x_traspuesta.dot(x)

    x_pseudoinversa = np.linalg.inv(x_pseudoinversa)

    x_pseudoinversa = x_pseudoinversa.dot(x_traspuesta)

    w = x_pseudoinversa.dot(y_traspuesta)

    return w
```

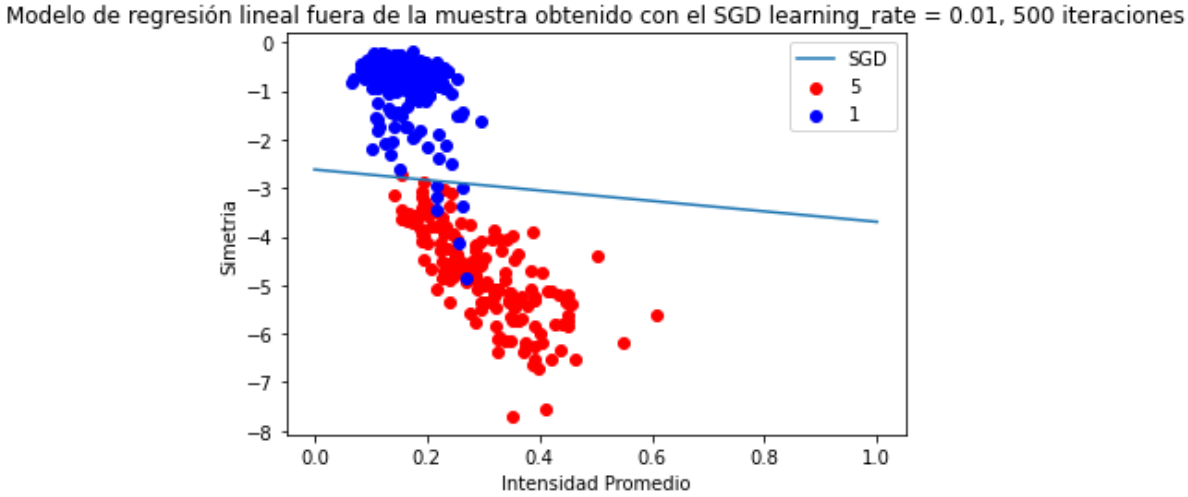
Lo primero que hago es calcular las traspuestas tanto de x como de y (necesitamos la traspuesta de y porque tal y como vemos en la diapositiva 13 del temario de teoría en el tema 1, y es una matriz Nx1 y nosotros en el código la tenemos almacenada como un vector de N elementos, con lo cual tenemos que calcular su traspuesta para convertir y en una matriz Nx1). Después de hacer esto me centro en calcular la pseudoinversa de X siguiendo la fórmula anterior. Calculo x traspuesta por x a y esto le hago la inversa, después esta inversa la multiplico por x traspuesta y ya obtengo la pseudoinversa de X.

Finalmente devuelvo el resultado de multiplicar la pseudoinversa de X por Y.

Una vez explicadas las implementaciones de los dos algoritmos voy a enseñar los resultados que he obtenido con ellos.

Con el SGD he obtenido lo siguiente:

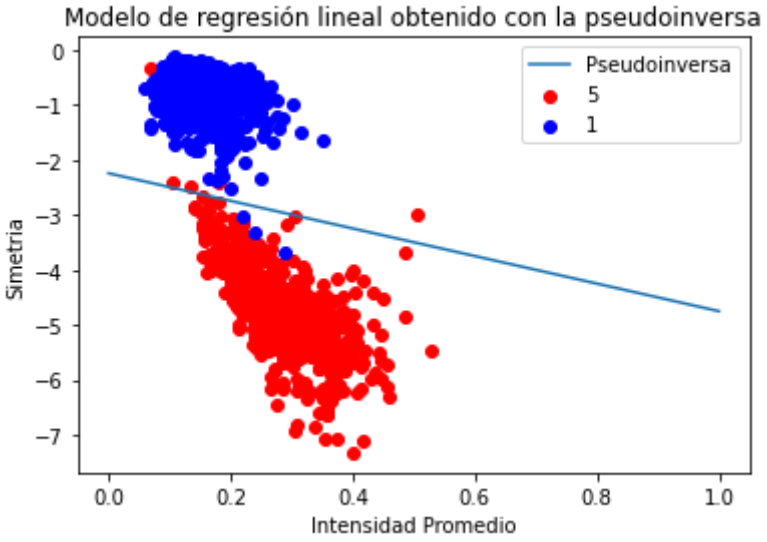


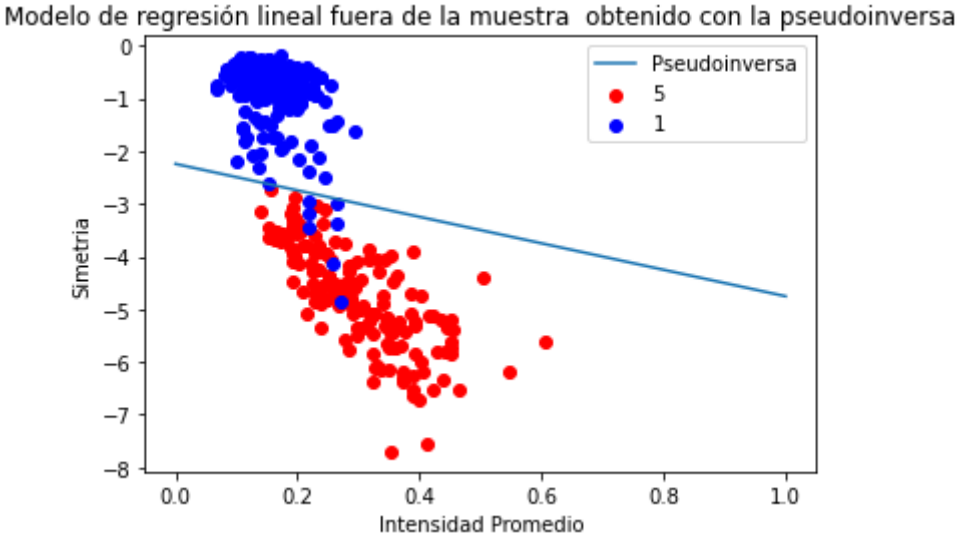


Bondad del resultado para el gradiente descendente estocástico:

Ein: 0.08127579333511419
Eout: 0.13129177323230315

Y con la pseudoinversa he obtenido los siguientes resultados:





Bondad del resultado para el algoritmo de la pseudoinversa:

Ein: 0.07918658628900395
Eout: 0.1309538372005258

Para pintar las rectas he despejado la siguiente ecuación.

$$y = w_0 + w_1 * x_1 + w_2 * x_2$$

queremos saber x_2 para despejar vamos a tomar los puntos (x1 = (0,0) y x1 = (1,0), con lo cual obtenemos las siguientes expresiones:

$$x_2 = -w_0/w_2$$
$$x_2 = (-w_0 - w_1)/w_2$$

y con esto podemos pintar la recta.

Como podemos observar a simple vista y fijándonos en los resultados de las bondades, el algoritmo de la pseudoinversa es capaz de conseguir un modelo un poco mejor, disminuyendo tanto el valor dentro de la muestra como fuera de la muestra un poco. Pero tanto el SGD como la pseudoinversa son métodos muy buenos.

Ejercicio 2

En este apartado exploramos como se transforman los errores Ein y Eout 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] × [−size,size]

EXPERIMENTO:

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

b) Consideremos la función f(x1,x2) = sign((x1−0,2)2+x22−0,6) 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.

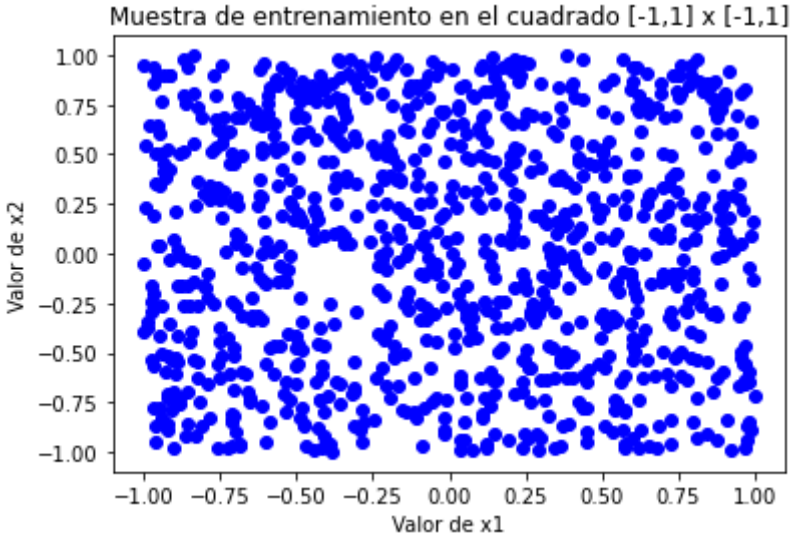
c) Usando como vector de características (1,x1,x2) ajustar un modelo de regresión lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste Ein usando Gradiente Descendente Estocástico (SGD).

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

- **Calcular el valor medio de los errores Ein de las 1000 muestras.**
- **Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de Eout en dicha iteración. Calcular el valor medio de Eout en todas las iteraciones.**

e) Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de Ein y Eout.

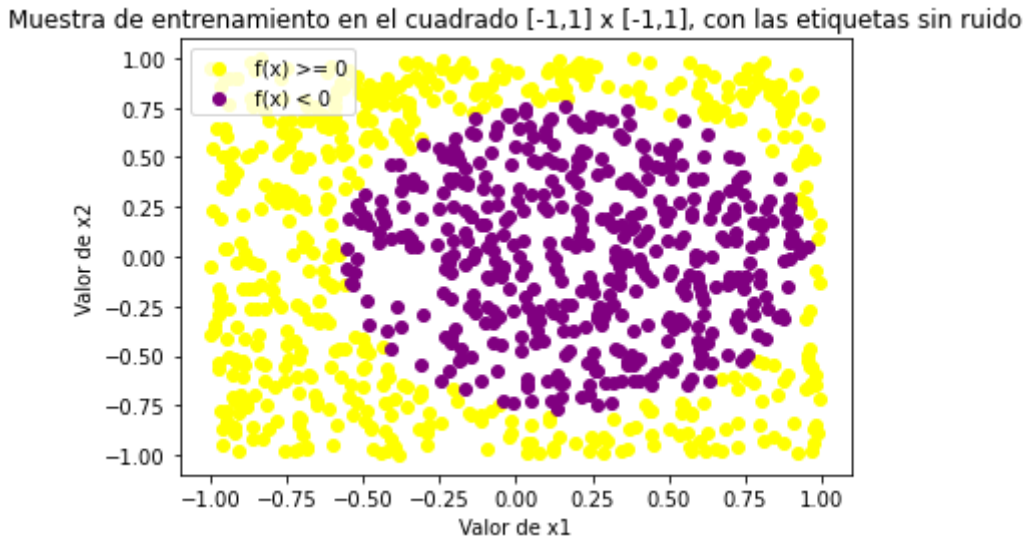
Para el apartado a lo único que he hecho ha sido hacer una llamada a la función que se indica en el apartado (dada en el template) y he obtenido los siguientes puntos:

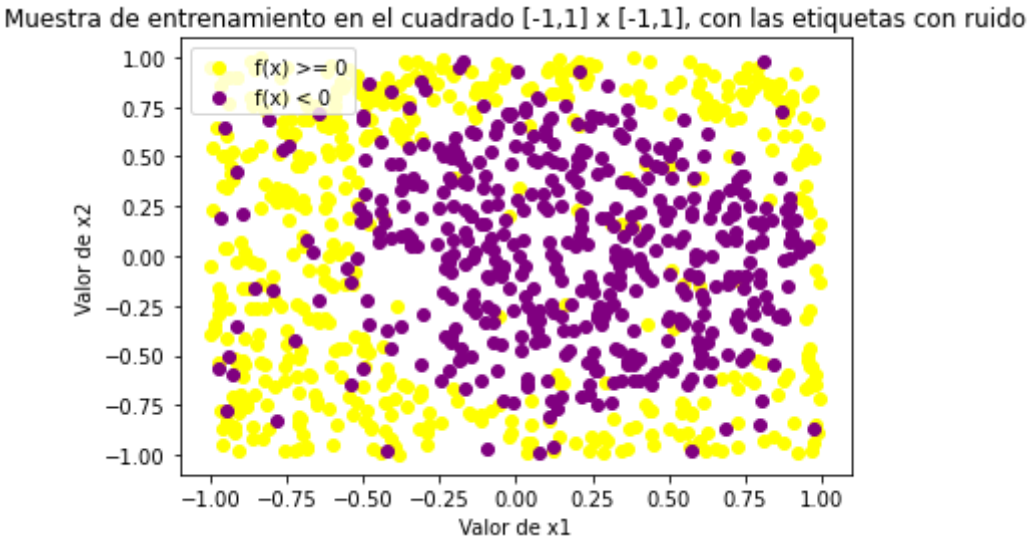


En el apartado b, para asignar las etiquetas a los puntos, he hecho un bucle que recorre el vector de puntos y por cada punto he llamado a la función indicada en el apartado que asigna las etiquetas 1 si el signo es positivo y -1 si el signo es negativo. Para introducir el ruido he implementado el siguiente método.

```
def ruido(etiquetas,porcentaje):
    num_etiquetas = len(etiquetas)

    etiquetas_a_cambiar = num_etiquetas * porcentaje

    etiquetas_a_cambiar = int(round(etiquetas_a_cambiar))

    for i in range (etiquetas_a_cambiar):
        indice = np.random.randint(0,1000)
        etiquetas[indice] = -etiquetas[indice]

    return etiquetas
```

En él calculo el número de etiquetas que tengo que cambiar y genero tantos números aleatorios entre [0,1000) como número de etiquetas tenga que cambiar, el número aleatorio generado actuará como índice y a la etiqueta con este índice le cambio el signo. Tras aplicar esta función he obtenido los siguientes puntos.



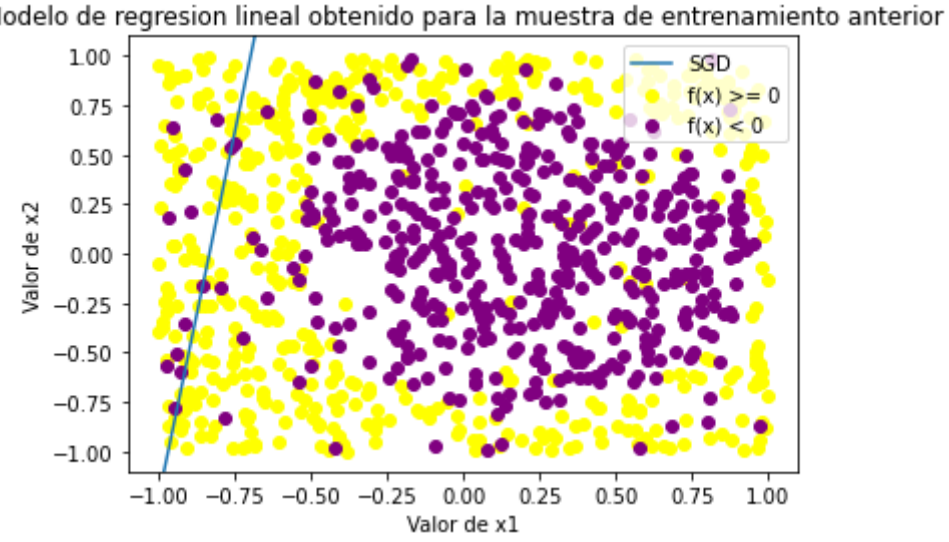


Por último, voy a llamar al algoritmo del sgd para que calcule un modelo de regresión. Para ello he de construir el vector de características, de manera que la primera columna siempre tenga un 1, para que w tenga un término independiente y el resto de columnas contengan el valor de la coordenada x e y del punto.

Para hacer esto he usado una función de numpy llamada _c que lo que hace es concatenar vectores por su posición, es decir, concatena los valores de la posición 1,2,....N de dos vectores. Tras hacer esto obtenemos un vector de características así (voy a enseñar solo las 10 primeras filas)

```
[[ 1.         -0.16595599  0.44064899]
 [ 1.         -0.99977125 -0.39533485]
 [ 1.         -0.70648822 -0.81532281]
 [ 1.         -0.62747958 -0.30887855]
 [ 1.         -0.20646505  0.07763347]
 [ 1.         -0.16161097  0.370439  ]
 [ 1.         -0.5910955   0.75623487]
 [ 1.         -0.94522481  0.34093502]
 [ 1.         -0.1653904   0.11737966]
 [ 1.         -0.71922612 -0.60379702]]
```

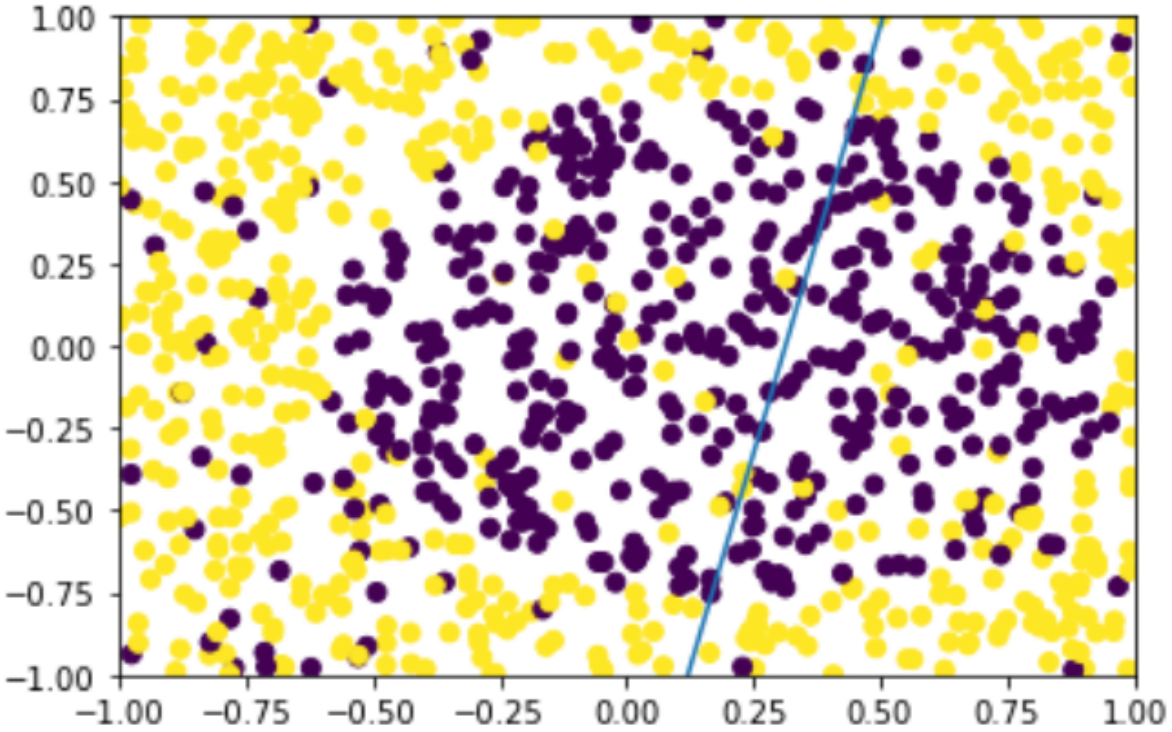
He insistido mucho en la construcción de este vector de características porque el modelo de regresión obtenido es el siguiente:



Ein: 0.9171981741506597

El modelo obtenido no es el esperado porque no hace una diferenciación entre las dos clases que hay en el problema, sin embargo el error que obtenemos si es el esperable, un error alto porque diferenciar las dos clases que tenemos en la gráfica con una recta es imposible. Se esperaba del algoritmo que se metiese entre los datos con signo negativo e intentase realizar la mejor división posible.

El resultado que se esperaría obtener sería este **(la siguiente fotografía está sacada de las diapositivas de prácticas NO es un modelo realizado por mi).**



También he comprobado que la recta esté construida de manera correcta y la construyo igual que en los otros apartados. Además el algoritmo del SGD debe estar bien implementado porque en los otros apartados sí he obtenido las soluciones esperables, con sus respectivos errores. Por lo tanto el error debe encontrarse en alguna tontería que no he conseguido encontrar.

Para el apartado d he obtenido los siguientes valores medios de Ein y de Eout:

Valor medio Ein: 0.9240769459635705

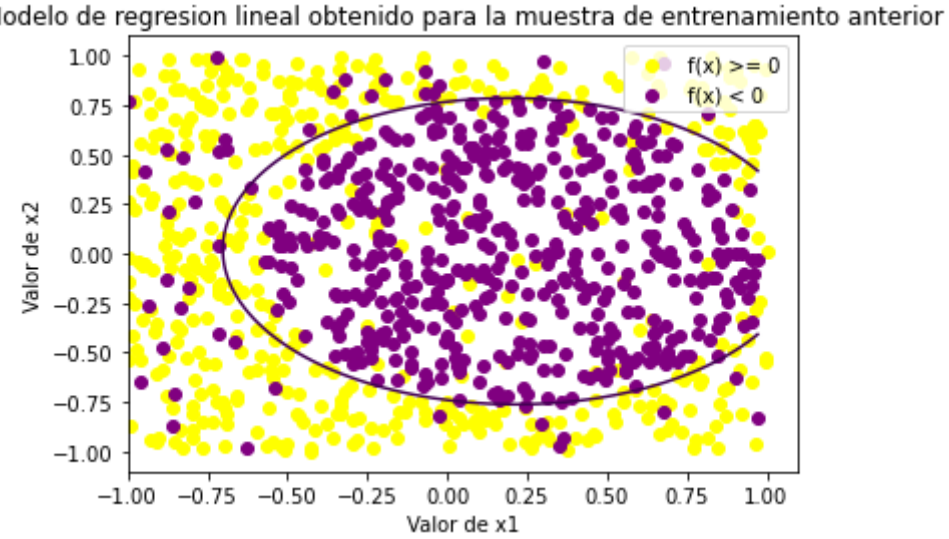
Valor medio de Eout: 0.9299705834624137

como vemos son valores muy altos ya que como hemos dicho una línea recta no es capaz de separar adecuadamente los datos.

Repetir el mismo experimento anterior pero usando características no lineales. Ahora usaremos el siguiente vector de características: Φ2(x) = (1,x1,x2,x1x2,$x1^2$,$x2^2$). Ajustar el nuevo modelo de regresión lineal y calcular el nuevo vector de pesos w. Calcular los errores promedio de Ein y Eout.A la vista de los resultados de los errores promedios Ein y Eout obtenidos en los dos experimentos ¿Que modelo considera que es el más adecuado? Justifique la decisión.

En este experimento, el vector de características que vamos a utilizar no es lineal, con lo cual el ajuste de regresión no va a ser una linea recta, será una elipse con lo que es esperable que divida mejor los puntos y el error sea menor.

Para no hacer más larga la memoria no voy a explicar el proceso de como he ido creando las muestras y el vector de características ya que es un proceso análogo al de el experimento anterior. Con el vector de características no lineales he obtenido el siguiente resultado:



Ein: 0.5672150617548325

Como podemos ver, este modelo es un mejor modelo que una línea recta para este conjunto de datos porque es capaz de hacer una separación buena de los datos.

Para 1000 iteraciones he obtenido los siguientes valores medios de Ein y Eout

Valor medio Ein: 0.5607422639166924

Valor medio de Eout: 0.5671107139178724

Tras realizar estos dos experimentos, el modelo más adecuado para este tipo de conjunto de datos (en los que una función separa en un subconjunto a los datos) es mejor utilizar un modelo de regresión no lineal, pero para conjuntos de datos en los que sí hay una clara separación es mejor utilizar un modelo de regresión lineal ya que este da unos errores tanto fuera como dentro de la muestra menores.

Bibliografía

https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a
https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/
https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Diapositivas de teoría y prácticas.

https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse