Aprendizaje Automático - Práctica 1

David Gil Bautista

Grupo 1

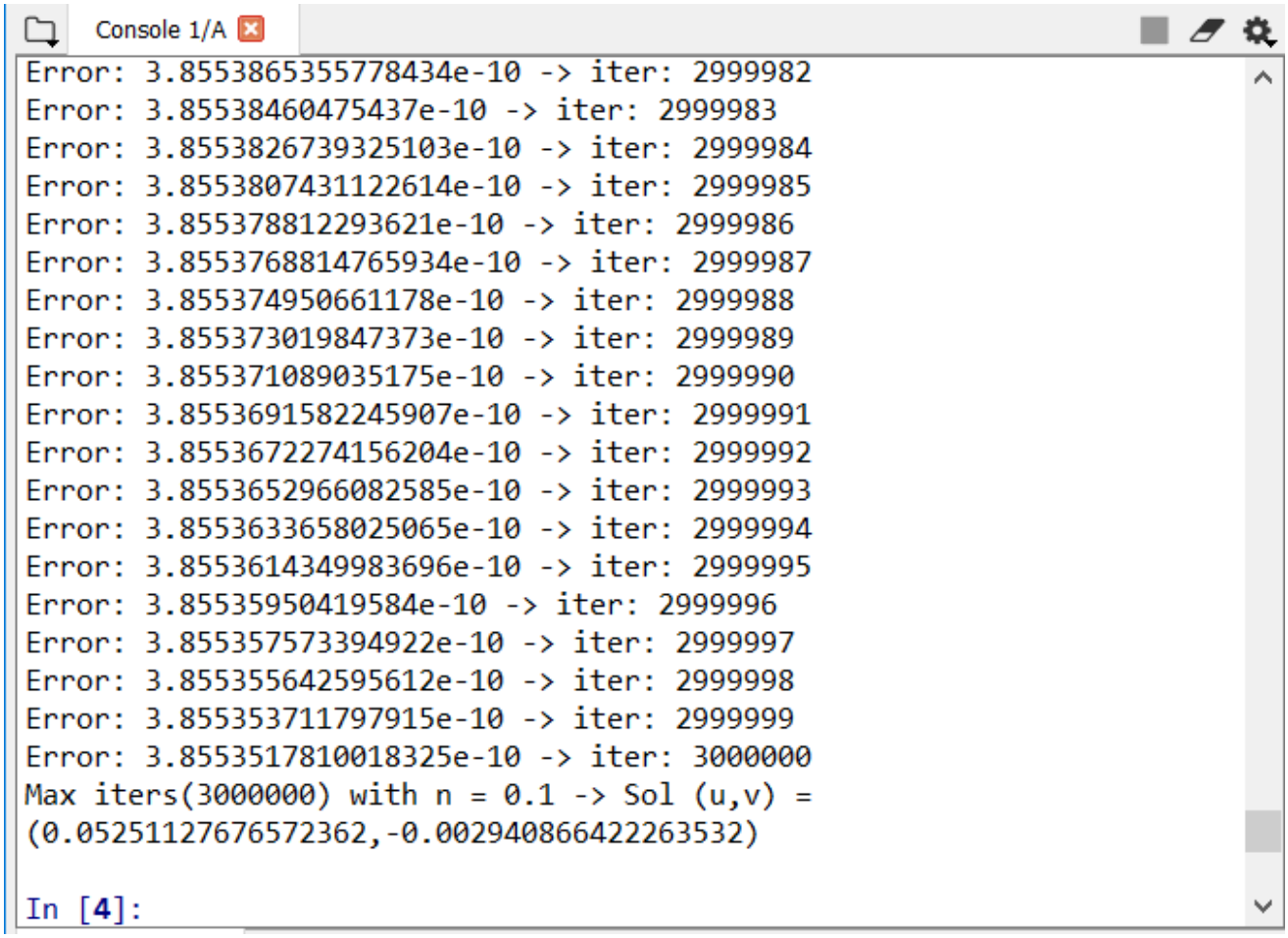
Ejercicio 1

Apartado 1)

Para la resolución de este primer ejercicio se nos pedía implementar el algoritmo del gradiente descendente y encontrar el mínimo de dos funciones utilizando dicho algoritmo.

Para la primera función se nos pide el nº de iteraciones cuando el error/coste es de un tamaño inferior a 10^{-14} partiendo de una $(u, v) = (1, 1)$.

Con un learning rate de 0.1 obtenemos este resultado:



Con 3 000 000 de iteraciones obtenemos un error de 3,86* 10^-10, por lo que podemos deducir que nuestro learning rate no es el adecuado para el problema puesto que el error varía muy poco en cata iteración.

Probando con un learning rate menor obtenemos el siguiente resultado:

```
Initial w = ( 1.0 , 1.0 )
Error: 8.795203949600008e-15 -> iter: 37
Sol (u,v) = ( 1.1195438968186378 , 0.6539880585437983 ) with 37 iterations
and n = 0.05.
```

En tan solo **36 iteraciones** encontramos un mínimo que cumple con nuestras restricciones, esto es así ya que a más bajo sea el learning rate más modificamos la función del gradiente y nos permite desplazarnos más rápido sobre la función para encontrar un mínimo.

Con la primera función un learning rate de 0.1 es totalmente ineficiente puesto que las variaciones sobre el gradiente son mínimas y necesita de muchas iteraciones para encontrar un mínimo.

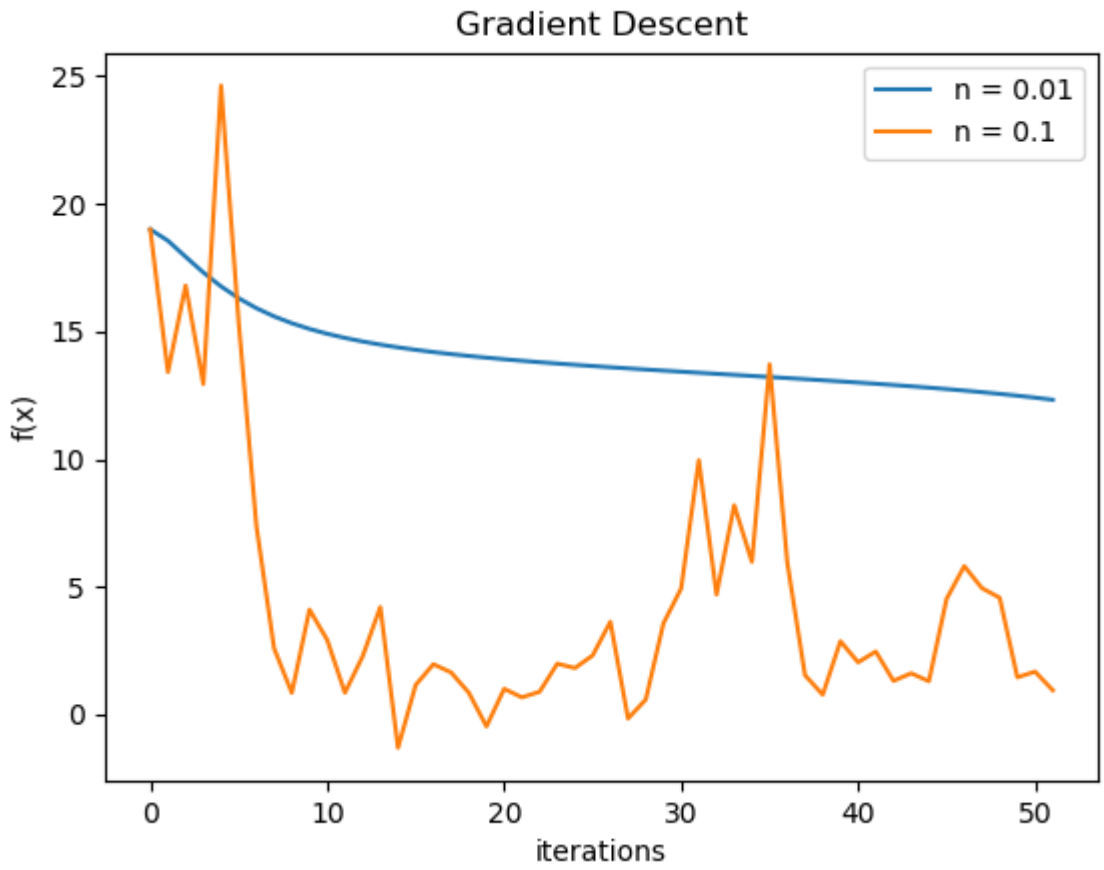
Con un **learning rate de 0.04** obtenemos las coordenadas de un mínimo con

$(u, v) = (1.158790924144973, 0.6953493114399102)$

Apartado 2)

Para esta segunda función se utiliza un learning rate de 0.01 y un máximo de 50 iteraciones partiendo de una $(u, v) = (1, 1)$.

En el siguiente gráfico podemos ver las salidas de error que obtenemos respecto de las iteraciones con dos learning rate distintos, 0.1 y 0.01.



En este caso, al contrario que en el anterior, un learning rate mayor encuentra antes el mínimo. En algunas funciones usar un learning rate menos puede ocasionar que el gradiente se pase el mínimo en cada iteración y aunque ocasionalmente el gradiente encuentre el mínimo, con un learning rate mayor no lo sobrepasaría tantas veces y lo encontraría con menos iteraciones.

```
Initial w = ( 1.0 , 1.0 )
Error: 12.323775536209313 -> iter: 50
Max iters(50) with n = 0.01 -> Sol (u,v) = ( 1.0251088405409725 ,
0.3590289001740001 )

Initial w = ( 1.0 , 1.0 )
Error: 0.9455563038754615 -> iter: 50
Max iters(50) with n = 0.1 -> Sol (u,v) = ( 1.8913938034405142 ,
-1.4562434676722351 )
```

En mi caso, con un learning rate de 0.01 ha llegado al máximo de iteraciones con un error de $12,3$ mientras que con un learning rate de 0.1 el error es 0.94 en la última iteración, aunque como podemos ver en el gráfico en la iteración 19 tenemos el error más cercano a 0.

Para la segunda parte se nos pide ejecutar el gradiente descendente partiendo de los puntos iniciales $(2.1, -2.1)$, $(3, -3)$, $(1.5, 1.5)$ y $(1, -1)$. Con los cuales obtenemos los siguientes resultados:

```
Initial w = ( 2.1 , -2.1 )
Error: 4.063492168759897e-05 -> iter: 25833
Sol (u,v) = ( 1.0880513601333828 , -2.1584654052644185 ) with 25833
iterations and n = 0.1.

Initial w = ( 3.0 , -3.0 )
Error: 8.417387137846233e-06 -> iter: 30669
Sol (u,v) = ( 2.2392778992297173 , -2.0045711132776938 ) with 30669
iterations and n = 0.1.

Initial w = ( 1.5 , 1.5 )
Error: 6.665016638967246e-05 -> iter: 9060
Sol (u,v) = ( 2.0152934619566443 , -2.0001387435200737 ) with 9060
iterations and n = 0.1.

Initial w = ( 1.0 , -1.0 )
Error: 5.446967283728732e-05 -> iter: 12845
Sol (u,v) = ( 2.102752133125463 , -2.4451085408970696 ) with 12845
iterations and n = 0.1.

 ------------------------------------------------------

Initial
w1 ( 2.1 , -2.1 )
w2 ( 3.0 , -3.0 )
w3 ( 1.5 , 1.5 )
w4 ( 1.0 , -1.0 )

After gradient
w1 ( 1.0880513601333828 , -2.1584654052644185 )
w2 ( 2.2392778992297173 , -2.0045711132776938 )
w3 ( 2.0152934619566443 , -2.0001387435200737 )
w4 ( 2.102752133125463 , -2.4451085408970696 )
```

Para esta parte he partido de n = 0.1 y 100000 iteraciones.

En principio he usado un error de parada de 10^{-4} (cuando un error es menor que ese error de parada, para y devuelve el resultado). Podemos apreciar que la muestra que ha empezado en un punto simétrico (1.5, 1.5) ha llegado al mínimo en muchas menos iteraciones que las otras muestras, sin embargo, el mínimo error de todas las muestras lo ha proporcionado la que se ha inicializado en (3, -3).

```
Initial w = ( 2.1 , -2.1 )
Error: 3.706345713880288e-06 -> iter: 68740
Sol (u,v) = ( 2.339472991115529 , -2.0108697954496053 ) with 68740
iterations and n = 0.1.

Initial w = ( 3.0 , -3.0 )
Error: 8.417387137846233e-06 -> iter: 30669
Sol (u,v) = ( 2.2392778992297173 , -2.0045711132776938 ) with 30669
iterations and n = 0.1.

Initial w = ( 1.5 , 1.5 )
Error: 2.6290645522145812 -> iter: 299999
Max iters(299999) with n = 0.1 -> Sol (u,v) = ( 1.088210642578085 ,
-3.1552649764059364 )

Initial w = ( 1.0 , -1.0 )
Error: 3.145940318360907e-06 -> iter: 180414
Sol (u,v) = ( 1.4694653011324532 , -2.1846203229799253 ) with 180414
iterations and n = 0.1.
```

Experimentando y cambiando el error de parada a 10^{-5} y el número de iteraciones a 300000 se puede observar que todos los modelos, menos el que comenzaba en $(1.5, 1.5)$, llegan a un error menor a 10^{-5}. Habiendo hecho esta prueba podemos determinar que dependiendo del punto inicial podemos llegar más o menos rápido a un mínimo, en nuestro caso una muestra inicial ha llegado rápidamente a un mínimo local pero no ha sido capaz de encontrar el mínimo global.

En conclusión, a la hora de intentar resolver cualquier función mediante el algoritmo de gradiente descendente es recomendable estudiar la función para escoger un buen punto inicial y un learning rate apropiado que no te haga iterar demasiado para encontrar un mínimo aceptable.

Ejercicio 2

A partir de un conjunto de datos de números escritos a mano se nos pide estimar la etiqueta a unos datos de un conjunto tipo test para diferenciar los 1 de los 5.

Apartado 1)

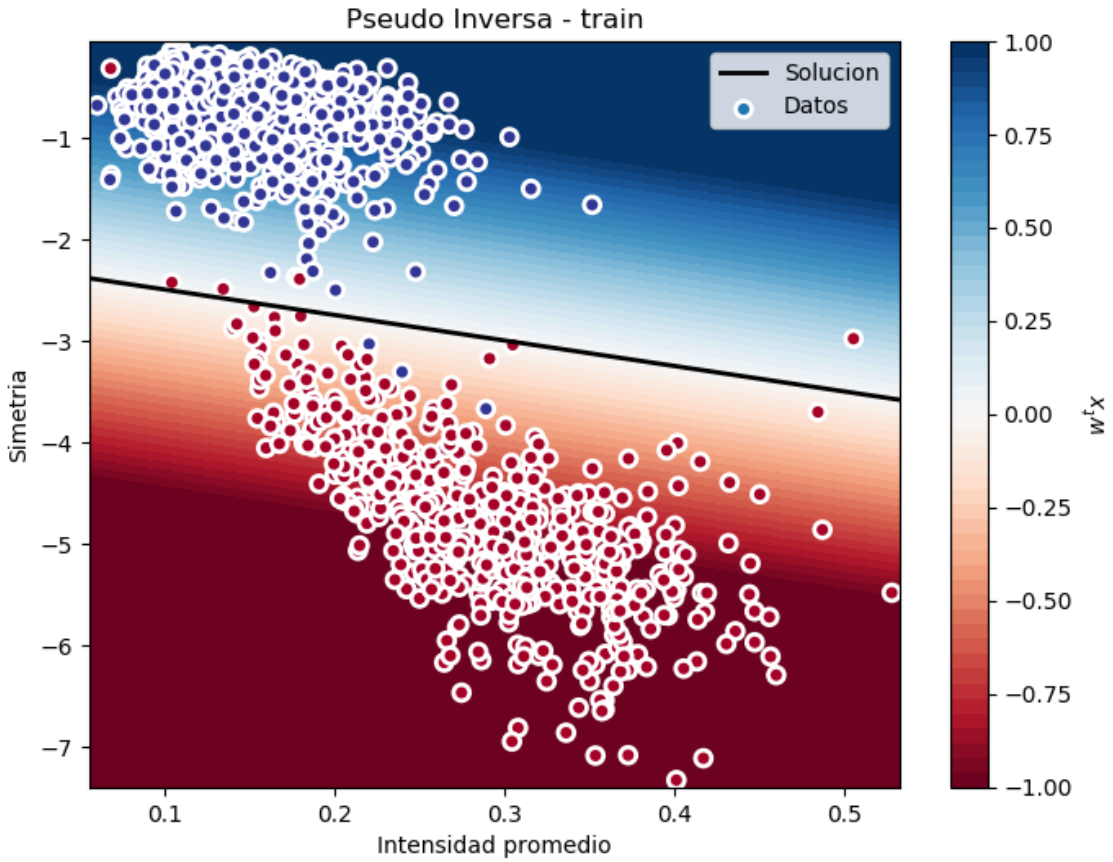
Mediante el uso de la matriz pseudo inversa obtenemos los siguientes resultados:

```
In [36]: runfile('C:/Users/David/Documents/MEGA/Dropbox/UGR/CSI/AA/
Practicas/practica1/ej2/apartado1b.py', wdir='C:/Users/David/
Documents/MEGA/Dropbox/UGR/CSI/AA/Practicas/practica1/ej2')
Classes =  [-1  1]
w = [8.29554904 0.68563697]
Error train = 0.22057033726848418
Error test = 0.2820685745604107

In [37]: runfile('C:/Users/David/Documents/MEGA/Dropbox/UGR/CSI/AA/
Practicas/practica1/ej2/apartado1b.py', wdir='C:/Users/David/
Documents/MEGA/Dropbox/UGR/CSI/AA/Practicas/practica1/ej2')
Classes =  [-1  1]
w = [1.24859546 0.49753165 1.11588016]
Error train = 0.07918658628900395
Error test = 0.13095383720052572
```

En la primera solución estamos calculando nuestra w con solo dos tuplas y obtenemos unos valores de $[8.29554904, 0.68563697]$ y podemos ver que para el **train** obtenemos un error de 0.22 y para el **test** 0.28.

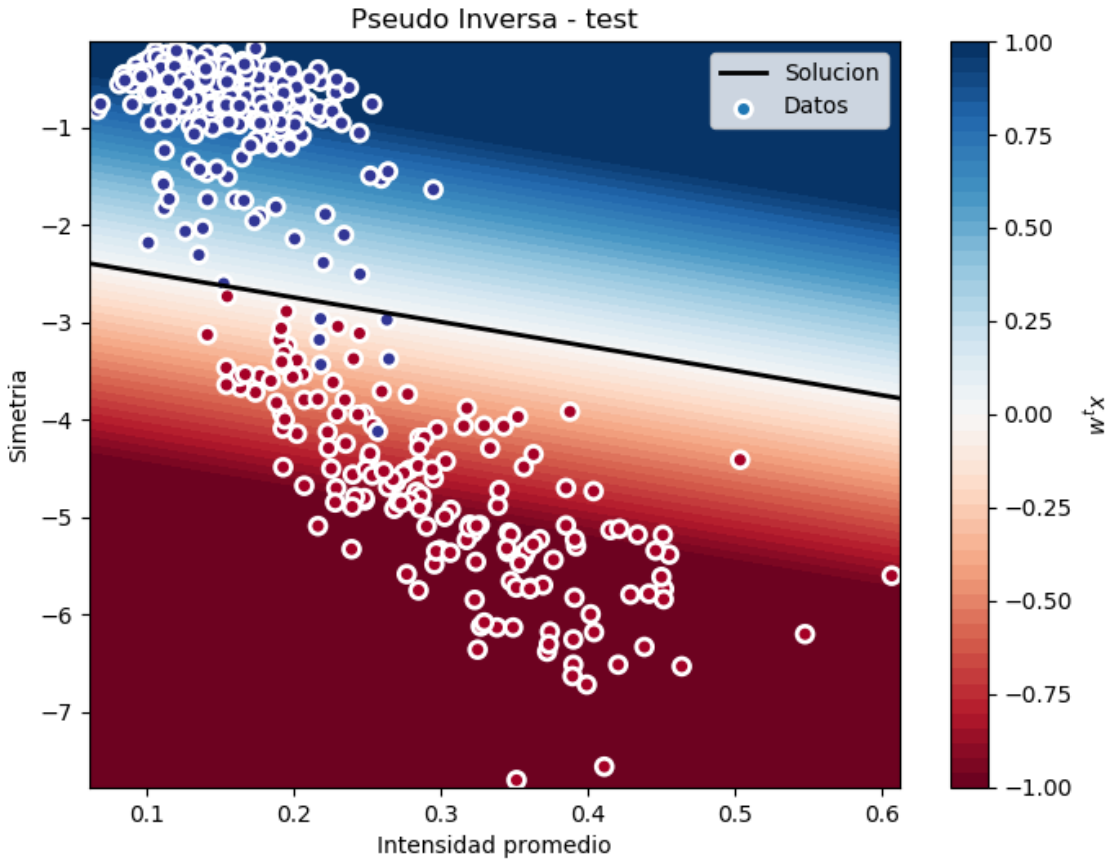
Usando tres tuplas aumentamos la precisión (reducimos el error) ya que tenemos un término independiente que modifica nuestra w y permite ajustar mejor la función. Ahora solo tenemos un error de 0.079 para el **train** y 0.13 para el **test**, que es menor que el error del **train** con dos tuplas.

Mediante la matriz pseudo inversa obtenemos la siguiente recta de regresión:



La cual se ajusta bastante bien y tan solo es incapaz de dividir unos puntos que podremos considerar mal medidos o con ruido.

Si aplicamos ahora dicha recta de regresión a nuestro conjunto de testeo podemos observar lo siguiente:



En este caso la recta se ajusta bien aunque hay varios datos que no ha conseguido dividir, ya que no hay forma de hacerlo.

Los errores de clasificación son los siguientes:

```
w = [1.24859546 0.49753165 1.11588016]
Error train = 0.07918658628900395

Error de clasificación en el train
De un conjunto de datos de 1561 elementos ha fallado en predecir 8
ErrorClasificacion = 0.005124919923126201

Error test = 0.13095383720052572

Error de clasificación en el test
De un conjunto de datos de 424 elementos ha fallado en predecir 7
ErrorClasificacion = 0.01650943396226415
```

En el conjunto del train podemos comprobar que hay 8 puntos que no están bien predichos mientras que en el conjunto del test hay 7. También hay que decir que a pesar de que el error en el train sea casi la mitad que el del test no significa que nuestra solución no se adapte bien al problema, sino que, al ser un conjunto con menos datos cada uno tiene un mayor peso y los errores penalizan más.

Uso del gradiente descendente estocástico:

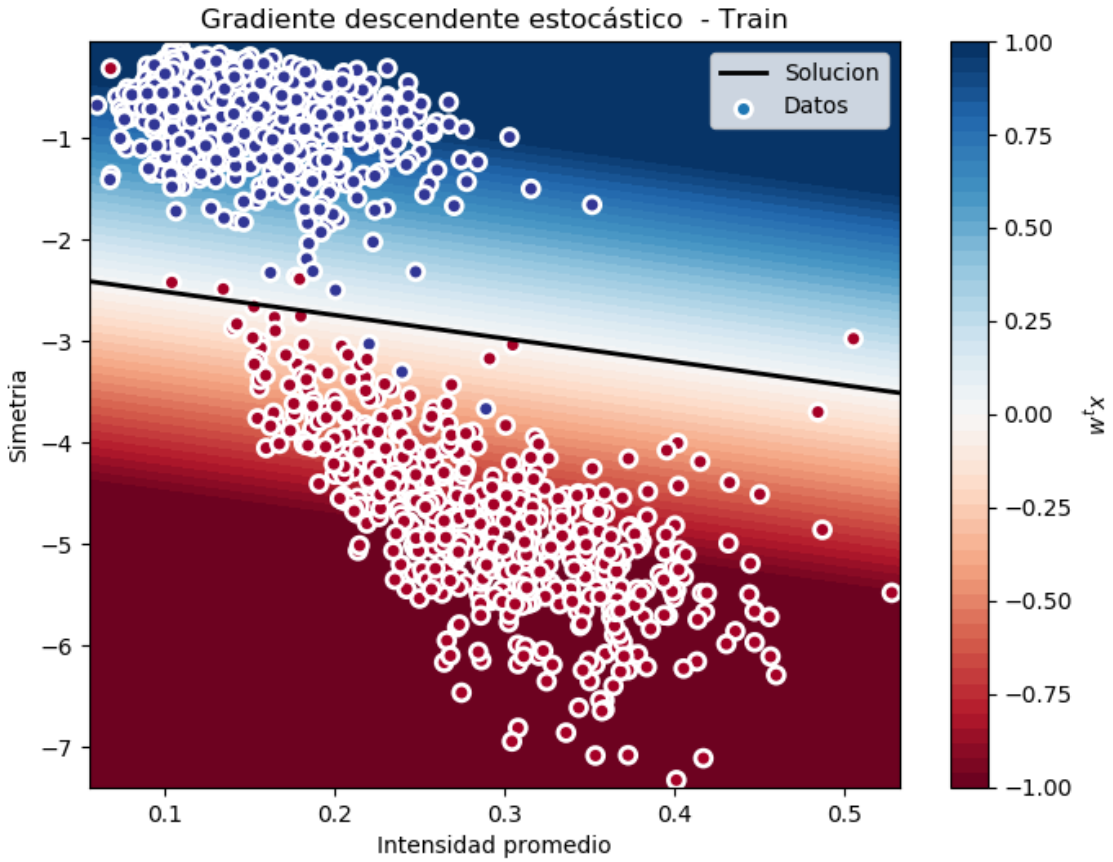
Para la implementación de este algoritmo he optado por tomar un tamaño de minibatch que dependa del tamaño del conjunto de datos de entrenamiento. El tamaño del minibatch es $log_2(data.shape[0]) * 5$.

```
def sgd(model, X_train, y_train, iters, minibatch_size):

    tam = len(X_train)
    Error = []
    err = []

    for i in range(iters):
        print('Iteration {}'.format(i))

        aux = list(zip(X_train,y_train))

        random.shuffle(aux)

        X_train, y_train = zip(*aux) #Random datos

        for j in range(0, tam, minibatch_size): #For minibatch
            X = X_train[j:j+minibatch_size]
            y = y_train[j:j+minibatch_size]

            model, err = minibatch_step(model, X, y)
            Error += err.copy()

    return model, Error
```

Nuestro algoritmo recibe el modelo a estimar inicializado, el conjunto de entrenamiento y sus etiquetas, un número de iteraciones y el tamaño de nuestro minibatch.

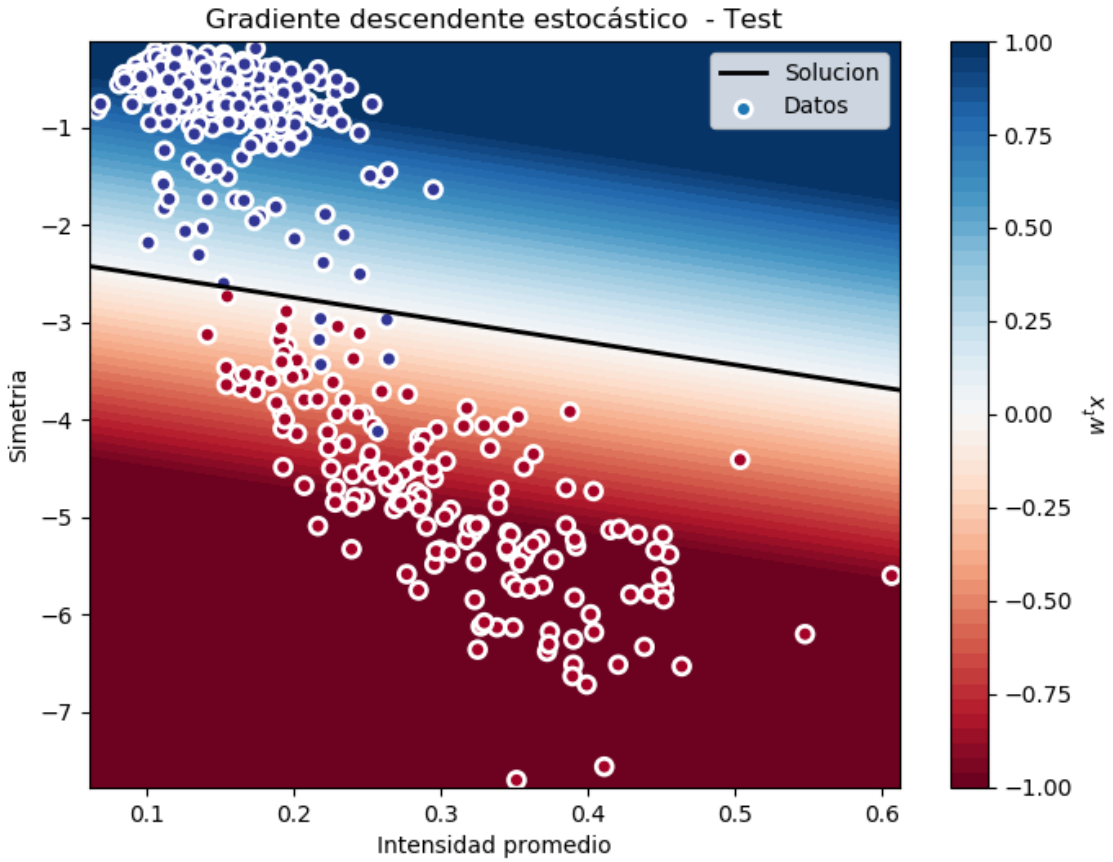
Para cada iteración desordenaremos nuestros conjuntos de datos y etiquetas y después en ese conjunto desordenado escogeremos un subconjunto con el tamaño del minibatch y le calcularemos su gradiente mediante la minimización de la función de error.

Con este algoritmo se obtienen los siguientes resultados para el train:



En este gráfico podemos observar, como en el anterior (pseudo inversa), que la recta de regresión se ajuste bastante bien.

Para el conjunto del test obtenemos lo siguiente:



En este caso nuestra recta de regresión también consigue ajustarse bien al conjunto de datos a pesar de poder ver algunos puntos que no ha podido dividir aunque podemos suponer que no estaban bien medidos.

Hemos obtenido nuestro modelo de regresión con 3000 iteraciones y un tamaño de minibatch de 65.

```
Iteration 2999
stochastic gradient descent -> Minibatch size : 65
w = [1.1386176  0.4941735  1.12890119]
Error train = 0.07959706920052236

Error de clasificación en el train
De un conjunto de datos de 1561 elementos ha fallado en predecir 8

Error test = 0.13128263489527714

Error de clasificación en el test
De un conjunto de datos de 424 elementos ha fallado en predecir 7
```

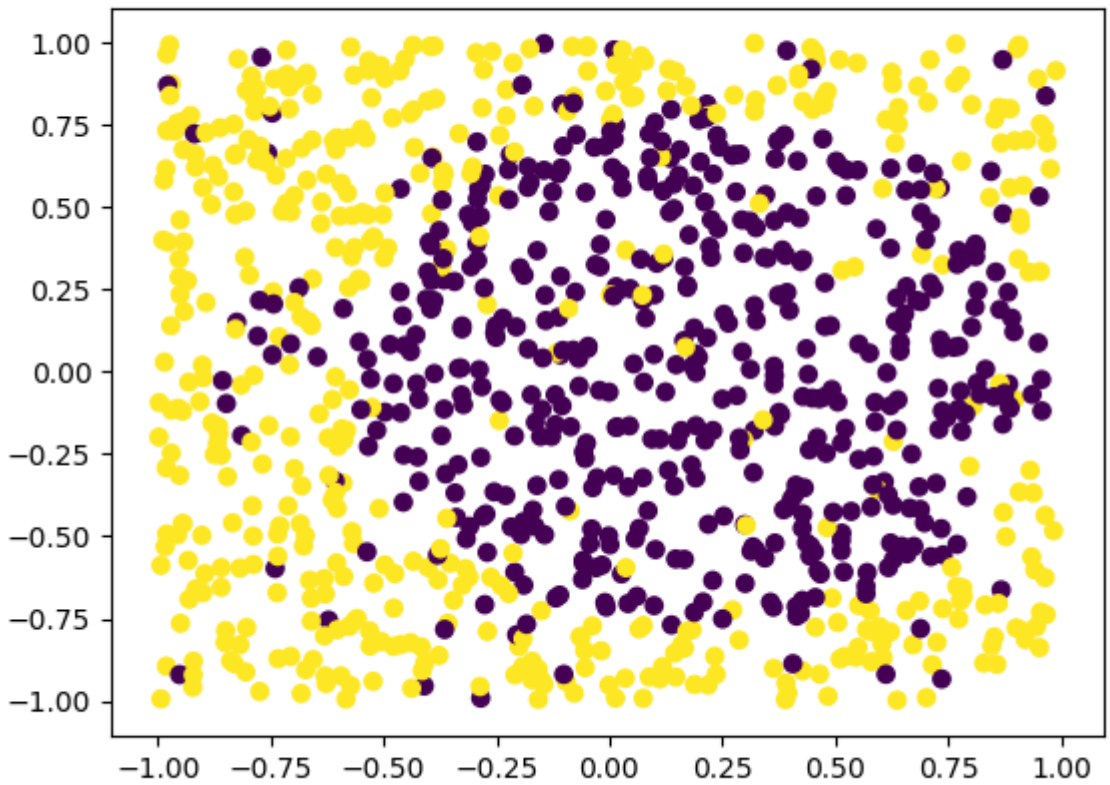
Se puede observar que los valores del modelo se parecen a los obtenidos a los valores de la pseudo inversa aunque obtenemos un error que es un poco mayor. Dado que con este método a mayor número de iteraciones obtienes una mejor solución mientras que con la pseudo inversa obtenemos el valor mínimo.

Tras haber ejecutado los dos algoritmos me he percatado de que para este conjunto tan pequeño de datos el método de la pesudo inversa te ofrece el mejor valor en un tiempo razonable mientras que con el gradiente ha tardado un poco más al tratarse de un método iterativo.

En conclusión, para obtener una solución óptima de un problema pequeño lo mejor es optar por operaciones matriciales, pero si se trata de un conjunto mayor se podría optar por un método iterativo en el cual el implementador tuviera la libertad de escoger el tiempo de computación que quiera para escoger un resultado más o menos exacto.

Apartado 2)

En este apartado se nos pide generar una muestra de entrenamiento de 1000 puntos para comprobar como se comportan los errores de entrada y salida cuando aumentamos la complejidad del modelo lineal usado.



Una vez generada la muestra y usando como vector de características $(1, x_1, x_2)$ obtenemos la siguiente salida para 3000 iteraciones usando el gradiente descendence estocástico.

```
Iteration 2999
stochastic gradient descent -> Minibatch size : 49
w = [ 0.23635522 -0.41799496 -0.00082111]
Learning rate : 0.1
Error train = 0.7424150644445068

Error de clasificación en el train
De un conjunto de datos de 799 elementos ha fallado en predecir 377
ErrorClasificacion = 0.4718397997496871

Error test = 0.9568865511243859

Error de clasificación en el test
De un conjunto de datos de 201 elementos ha fallado en predecir 93
ErrorClasificacion = 0.4626865671641791
```

Como podemos observar se obtiene un error de clasificación cercano al 50%. Viendo la gráfica con los datos ya podemos deducir que nuestro modelo lineal no es el adecuado para resolver dicho problema puesto que al dividir nuestro conjunto mediante una linea recta el fallo rondará el 50% y es lo que hemos demostrado usando el gradiente descendente estocástico.

Pd: Para la práctica no he puesto 3000 iteraciones porque tarda un buen rato