Natalia Fernández Martínez, grupo A2

TRABAJO 1. Búsqueda iterativa de óptimos y regresión lineal

Ejercicio 1. Ejercicio sobre la búsqueda iterativa de óptimos

Apartado 1

Implementar el algoritmo de gradiente descendente

Pseudo-código del algoritmo:

```
while(numIteraciones < maxIteraciones && error > errorMinimo){
      aux = puntoActual
      puntoActual = puntoActual - tasaAprendizaje * gradiente(aux)
      error = abs(puntoActual - aux)
      numIteraciones++
}
```

Calcular el gradiente de un punto consiste en calcular la derivada parcial para cada componente. Es decir, en un punto P = (x,y), el gradiente de P es P' = (derivada parcial respecto a x, derivada parcial respecto a y)

Las derivadas parciales las he calculado con dos funciones distintas, dEu y dEv, y dentro de estas he usado la función `diff` de la librería `sympy` para calcular las derivadas.

La razón por la que en las funciones E(), dEu(u,v) y dEv(u,v) se usan x e y en lugar de u y v, es que al usar la librería `sympy` hay que usar "símbolos" en lugar de variables para hacer los cálculos. Entonces cuando es necesario usar funciones con las incógnitas sin resolver, como E(), se devuelven en función de x e y; sin embargo, en las derivadas parciales sí necesitamos conocer los valores numéricos, así que se sustituyen las incógnitas por u y v.

Apartado 2

Considerar la función E(u,v) = $(u^2e^v - 2v^2e^{-u})^2$ **. 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= 0,01**

(a) Calcular analíticamente y mostrar la expresión del gradiente de la función E(u,v)

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

$$\frac{\partial E}{\partial x} = (4ue^v + 4v^2e^{-u}) \cdot (u^2e^v - 2v^2e^{-u})$$

$$\frac{\partial E}{\partial y} = (u^2e^v - 2v^2e^{-u}) \cdot (2u^2e^v - 8ve^{-u})$$

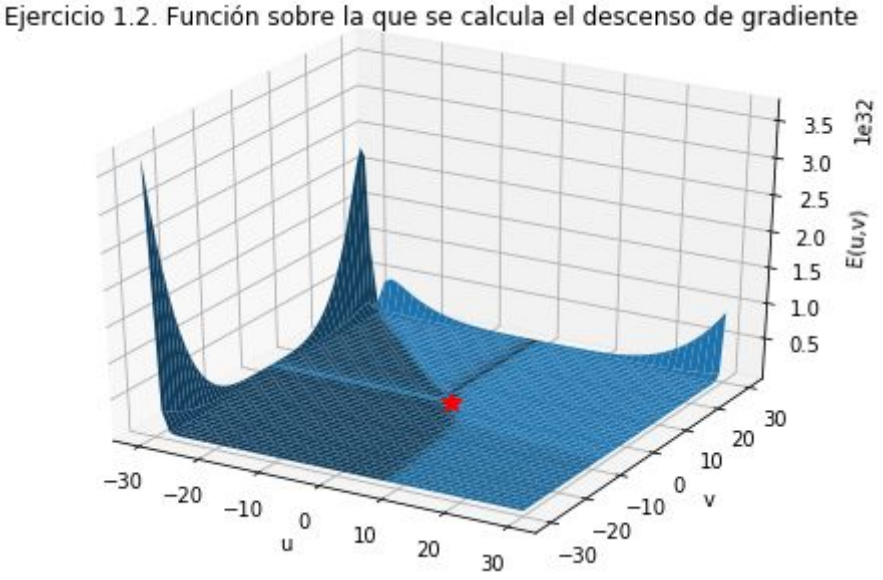
$$GradienteE(u,v) = [\frac{\vartheta E}{\vartheta x}, \frac{\vartheta E}{\vartheta y}]$$

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

El algoritmo hace 59 iteraciones hasta que para porque la diferencia entre pasos es menor que 10^{-14}

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

El algoritmo para cuando llega a las coordenadas (0.619207661402470 , 0.968448273303026)

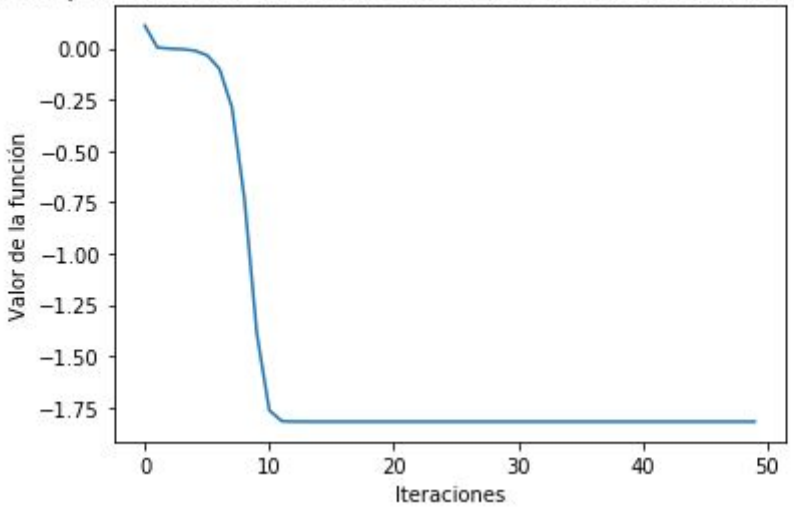


Apartado 3

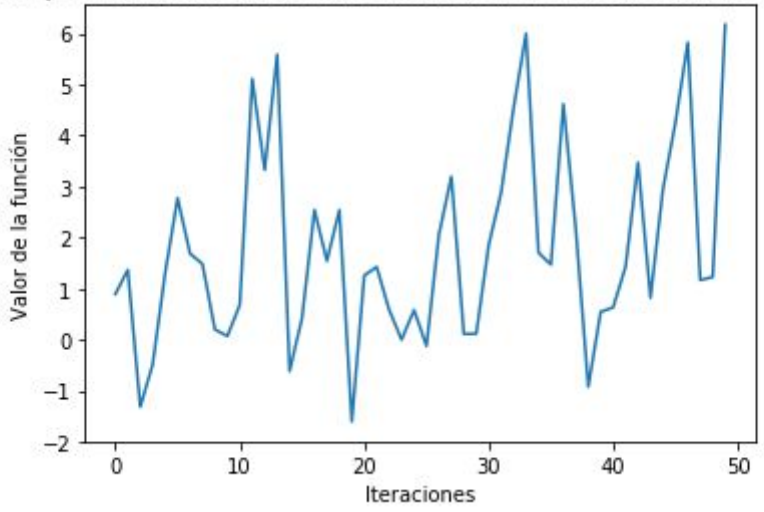
Considerar ahora la función f(x,y) = $x^2 + 2y^2 + 2 \cdot sin(2\pi x) \cdot sin(2\pi y)$**.**

(a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial (x0= 0,1,y0= 0,1), (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.

Ejercicio 1.3. Representación del descenso del valor de la función con tasa de aprendizaje 0,01



Ejercicio 1.3. Representación del descenso del valor de la función con tasa de aprendizaje 0,1



Como se puede ver en los dos gráficos, para una tasa de aprendizaje de 0,01, el algoritmo se queda atascado en un mínimo local en el que la función toma un valor aproximado de -1,75. Sin embargo, para una tasa de 0,1 los “pasos” que se toman son demasiado grandes, por lo que la pendiente de la función cambia más rápido y hay “overshooting”. Es decir, se pasa del valor mínimo, y al final el algoritmo devuelve un valor aproximado de 6 para la función, bastante lejano del mínimo obtenido con la tasa de 0,01.

(b) Obtener el valor mínimo y los valores de las variables (x,y) en donde se alcanzan cuando el punto de inicio se fija: (0,1,0,1), (1,1), (−0,5,−0,5), (−1,−1). Generar una tabla con los valores obtenidos.

	x	y	f(x,y)
[0.1, 0.1]	0.243804969364788	-0.237925821486178	-1.82007854154716
[1.0, 1.0]	1.21807030131108	0.712811950601778	0.593269374325836
[-0.5, -0.5]	-0.731377460413804	-0.237855362901572	-1.33248106233098
[-1.0, -1.0]	1.21807030131108	0.712811950601778	0.593269374325836

Apartado 4

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

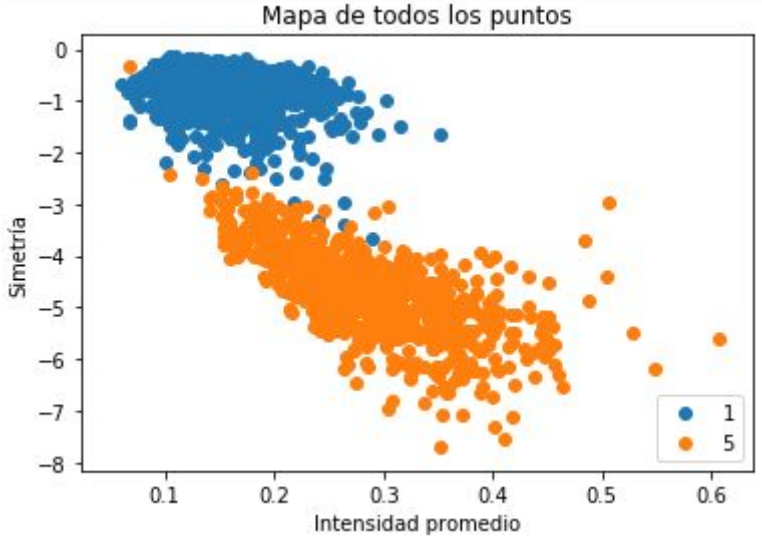
Usando el gradiente descendente, lo crucial para encontrar el mínimo es:

1. Saber asignar un buen valor a la tasa de aprendizaje, de manera que no sea ni muy pequeño, porque entonces necesitará hacer más iteraciones y requerirá más poder computacional; ni muy grande porque entonces puede que se pase del mínimo.
2. Encontrar un buen punto de comienzo para el algoritmo, ya que esto va a determinar cuánto tardará en encontrar el mínimo dependiendo de lo cerca o lo lejos que esté de este. Para ayudar con esto, se pueden usar otros algoritmos más simples primero para encontrar una aproximación de este punto que sea un buen punto de partida.
3. El valor de precisión también es importante. Este es el valor con el que se compara la diferencia entre dos pasos del algoritmo para determinar si se ha llegado a un mínimo. Si es muy grande, puede que nos quedemos con un valor pudiendo tener uno mejor más adelante; si es muy pequeño, el algoritmo hará más iteraciones de las necesarias y puede que se quede en un bucle infinito.

Ejercicio 2. Ejercicio sobre regresión lineal

Apartado 1

Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetría) usando tanto el algoritmo de la pseudo-inversa como Gradiente descendente estocástico (SGD). Las etiquetas serán {−1, 1}, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando Ein y Eout (para Eout calcular las predicciones usando los datos del fichero de test).



1. **Algoritmo de la pseudoinversa**

Para usar el algoritmo de la pseudoinversa, se usan dos vectores:

- La matriz X con todos los inputs, que tendrá dimensiones N x (d+1) (con N siendo el número de inputs y d el número de características que se usan para evaluar los inputs)
- El vector Y de tamaño N con los valores que corresponden a cada input de acuerdo a la función objetivo f: X → Y

Con estos elementos, podemos obtener el vector de pesos w = $X^{\dagger}Y$. La X y la Y que le pasamos al algoritmo son los valores de entrenamiento, mientras que después obtendremos los valores para x_test usando la w calculada con estos valores.

$X^{\dagger}$ es la pseudo-inversa de X, que se calcula así:

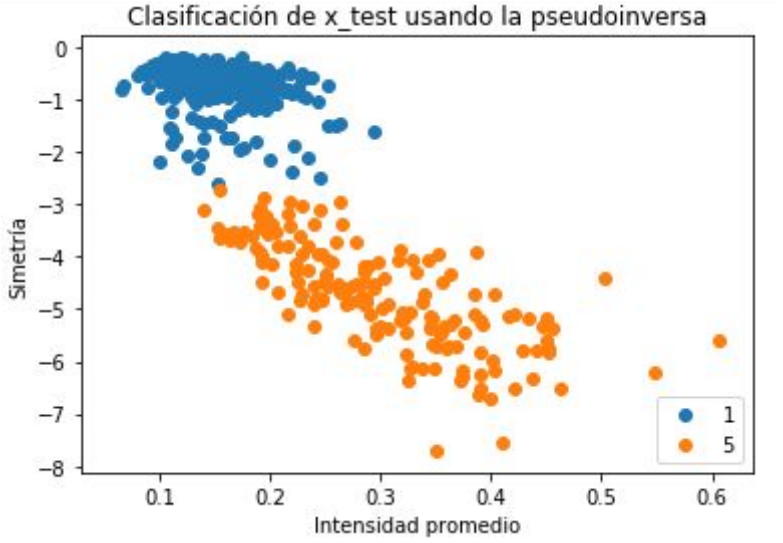
$$X\dagger = (X^T X)^{-1} X^T$$

Con esto obtenemos w, y usando la función clasificar(x,w) clasificamos a x_test.

Resultados:

```
Bondad del resultado para la pseudoinversa:

Ein:  0.07918658628900396
Eout:  0.1309538372005258
```



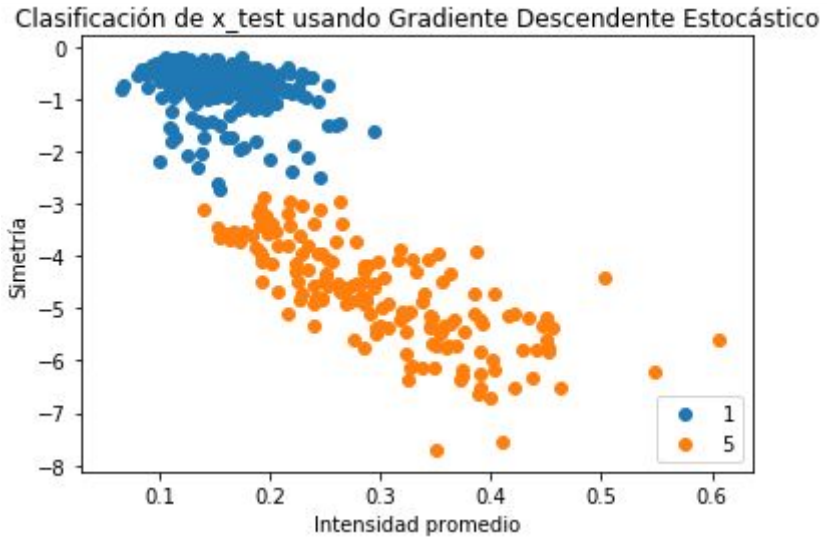
2. **Gradiente descendente estocástico (SGD)**

El SGD usa los mismos argumentos que el algoritmo de la pseudoinversa, X e Y. Lo que hace el algoritmo es crear minibatches de 32 elementos de X e Y y calcular un vector de pesos que los separe, y repite esto hasta que el Ein generado por el minibatch sea menor que ε o hasta que haga un número de iteraciones. Se va guardando el vector de pesos que mejor resultado ha dado, y en caso de que no se encuentre una buena solución se devuelve este.

Resultados:

```
Bondad del resultado para grad. descendente estocastico:

Ein:  0.09085933431864283
Eout:  0.13754643210682677
```



Cálculo del error en la muestra:

$$Ein(w) = \frac{1}{N} \cdot \sum_{n=1}^{N} (w^T x_n - y_n)^2$$

En la práctica he implementado el cálculo de Ein con la forma matricial
Forma matricial de Ein:

$$Ein(w) = \frac{1}{N} \cdot ||Xw - y|| = \frac{1}{N} \cdot (Xw - y)^T \cdot (Xw - y)$$

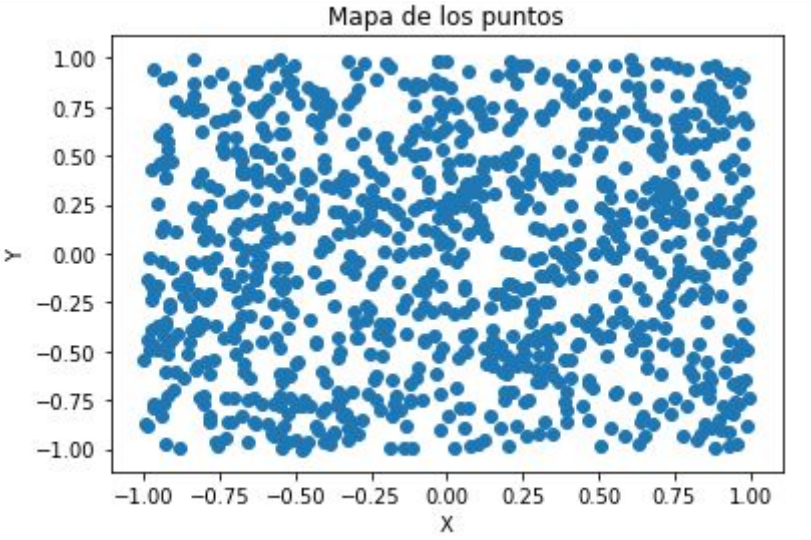
Función de clasificación:

La función `clasificar(x,w)` devuelve un vector `y` con las etiquetas correspondientes a los inputs pasados por el argumento `x`. Calcula el valor de la expresión `w[0]+w[1]*x[i][1]+w[2]*x[i][2]` para cada input dentro de `x`; si el valor es mayor que 0, lo considera como un 1 (5), y si es menor que 0 lo considera como un -1 (1).

Apartado 2

Apartado a

Generar una muestra de entrenamiento de N= 1000 puntos en el cuadrado X = [−1,1]×[−1,1]. Pintar el mapa de puntos 2D.
Distribución de los puntos:

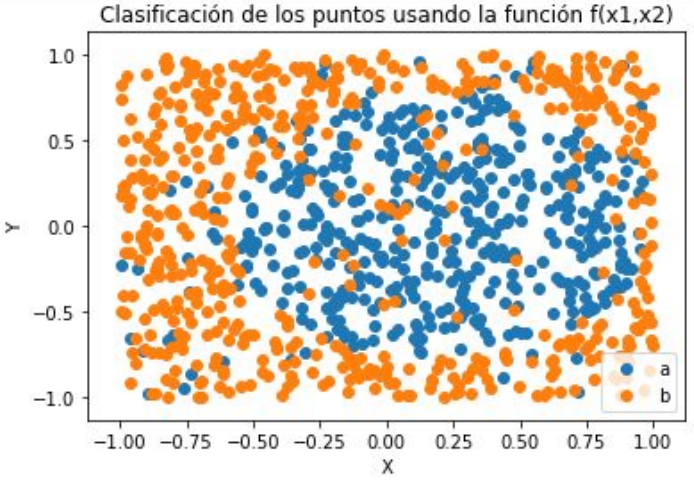


El ejercicio pide puntos 2D, pero en la llamada a la función le doy como argumento 3 dimensiones para poder iterar sobre todos los valores después y cambiar el primer elemento de todos los inputs a 1.0. Esto nos servirá en los apartados posteriores para poder hacer los cálculos.

Apartado 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.

Mapa de etiquetas:



Apartado 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).

Viendo la función f o la gráfica con la clasificación de los puntos, queda claro que no se va a poder encontrar una buena estimación para los datos, ya que no son linealmente

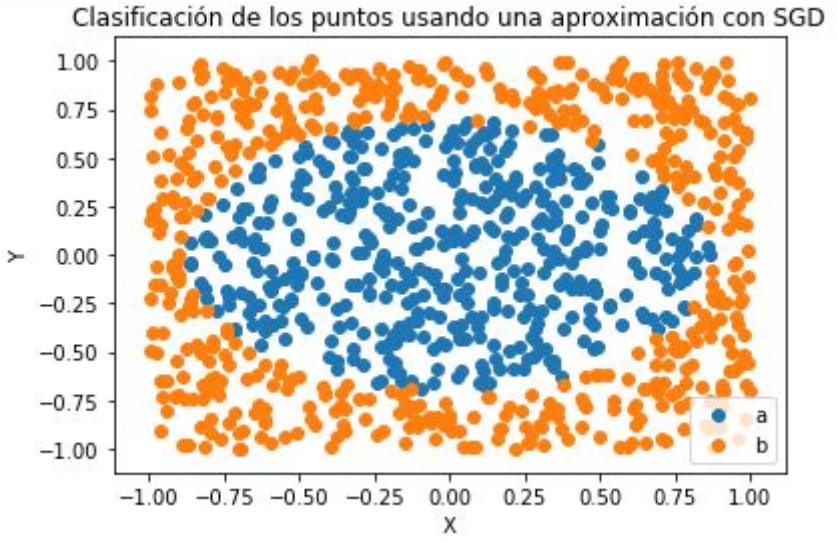
separables. Calculando el Ein generado para los datos usando SGD, da un valor cercano a 1.

Sin embargo, si le hacemos una transformación no-lineal a los inputs, elevando x1 y x2 al cuadrado, la distribución de los datos cambia y es un poco más fácil separarlos.

```
Bondad del resultado para grad. descendente estocastico:

Ein:  0.6331085610122945
```

El Ein es mucho menor, pero sigue sin ser una buena aproximación.



Apartado 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.**

Para este apartado he metido todos los elementos de los apartados anteriores (menos las líneas de generación de gráficos) en una función llamada funcion_d(). Dentro de esta además se generan otros 1000 puntos, se transforman de la misma manera que los puntos de entrenamiento, y se calculan Ein y Eout con los valores de entrenamiento y los de prueba respectivamente. Lo que devuelve la función es una tupla que contiene a Ein y Eout. La función se llama 1000 veces dentro de un for loop, que va acumulando los valores de error.
Cuando acaba el loop, se calculan los Ein y Eout medios dividiendo los valores acumulados por 1000 y se sacan por pantalla.

```
Ein medio para 1000 iteraciones:  0.6483473159675627
Eout medio para 1000 iteraciones:  0.5147430546275921
```

El loop tarda unos 2 minutos en ejecutarse entero.

Apartado 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

El ajuste, aunque da mejores resultados que sin transformación lineal, no es muy bueno a la vista de los valores de Ein y Eout. Esto ya es fácil de predecir antes de hacer los cálculos, ya que no solo hay un 10% de ruido introducido en la muestra de entrenamiento, sino que además los valores no son linealmente separables.