APRENDIZAJE AUTOMÁTICO

Práctica 1 – Búsqueda iterativa de óptimos y regresión lineal

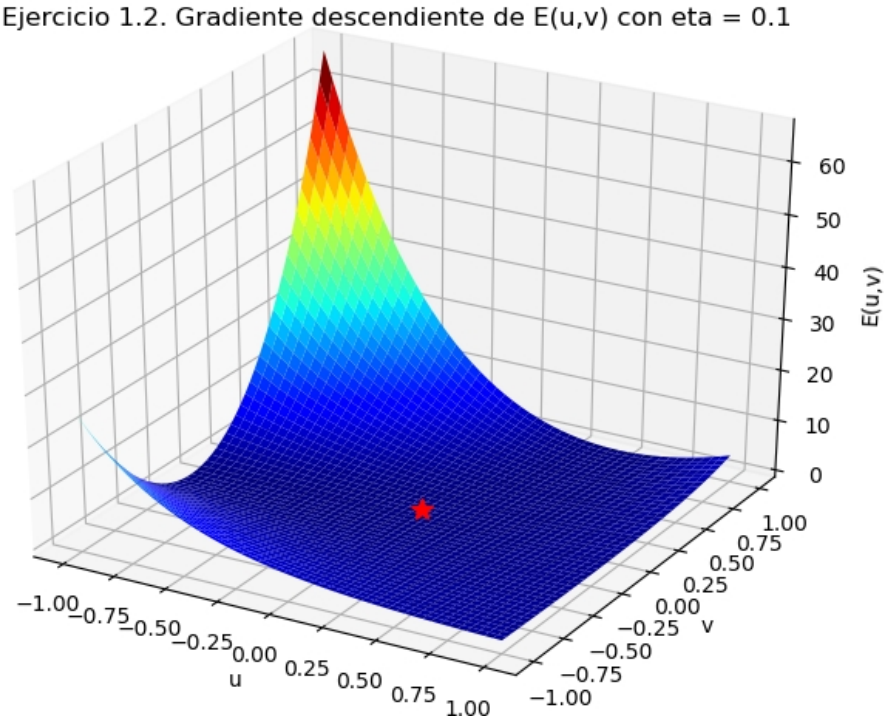
Juan Pablo García Sánchez
Grupo 1
3º GII - UGR

Ejercicio 1 – Búsqueda iterativa de óptimos

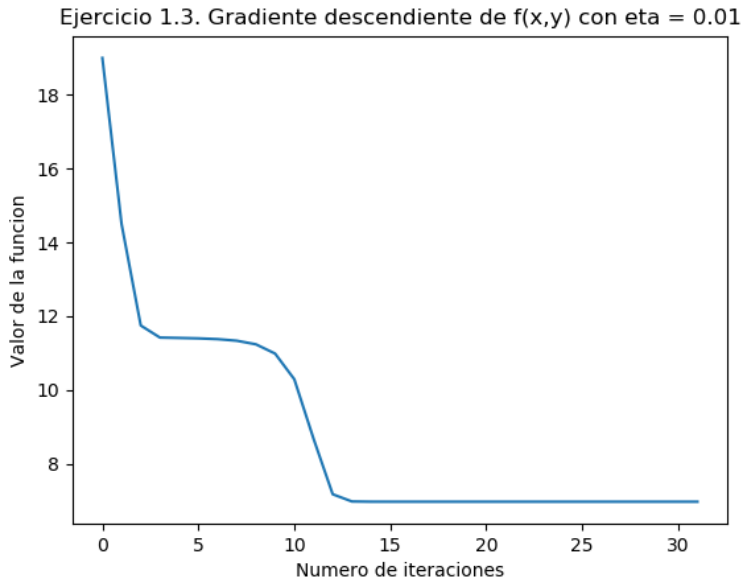
1. Para la búsqueda de óptimos en este ejercicio se usa el gradiente descendente. Consiste en calcular la derivada de un punto en la función que queremos minimizar (o maximizar) y moverse en dirección al mínimo utilizando esa pendiente calculada. Utiliza una tasa de aprendizaje para saber cuanto "desciende" en la función. El algoritmo para cuando se han hecho muchas iteraciones o cuando el error es lo suficientemente pequeño.

En la implementación se utiliza *sympy* para poder operar con la función usando símbolos en vez de números.

2. El algoritmo encuentra un mínimo en 17 iteraciones en el punto (0.0447, 0.0240)



3. Para el caso de η = 0.01, se encuentra un mínimo en 32 iteraciones en el punto (1.2693, -0.2064)



Cuando η = 0.1, no he acabado la ejecución porque tarda mucho. El problema es que se produce *overshooting* con esta tasa de entrenamiento. La coordenada *y* del punto pasa de positivo a negativo en cada iteración y con números muy elevados. La tasa es tan grande que sube en la función en vez de descender al mínimo.

```
Punto:  [2.94281963673316 -1.51714126820204e+201]
Iteracion:  9
Punto:  [1.80637237324040 2.79362464056322e+603]
Iteracion:  10
Punto:  [1.40924980111487 -1.74419141653328e+1810]
Iteracion:  11
Punto:  [2.34584621871034 4.24494846123190e+5430]
Iteracion:  12
Punto:  [1.62426864437714 -6.11937759753267e+16291]
Iteracion:  13
Punto:  [1.20776247356641 1.83320799983382e+48875]
Iteracion:  14
Punto:  [1.14014171039006 -4.92861867518262e+146625]
Iteracion:  15
Punto:  [1.93121278457406 9.57779730663075e+439876]
Iteracion:  16
Punto:  [3.02052840016211 -7.02889268502811e+1319630]
Iteracion:  17
Punto:  [2.28623368646033 2.77811823480470e+3958892]
Iteracion:  18
Punto:  [2.38864539821153 -1.71530819385650e+11876677]
Iteracion:  19
Punto:  [2.32530101102806 4.03753661291669e+35630031]
```

Luego se prueba con varios puntos iniciales distintos:

```
(2.1, -2.1)
( 2.24359194160226 ,  -0.206659409303214 )      f(x,y) =  6.47991044705444  Iteraciones:  27
(3, -3)
( 3.21771256334112 ,  -0.205931526866006 )      f(x,y) =  7.94131499384886  Iteraciones:  25
(1.5, 1.5)
( 1.28892155663845 ,  0.577407864845654 )       f(x,y) =  10.4881210288455  Iteraciones:  50
(1, -1)
( 1.26927281927820 ,  -0.206421439928688 )      f(x,y) =  6.96707815795529  Iteraciones:  32
```

Todos encuentran un valor de la función parecido menos el punto (1.5, 1.5), que supera el límite de iteraciones. Es posible que con más encuentre un mínimo también.

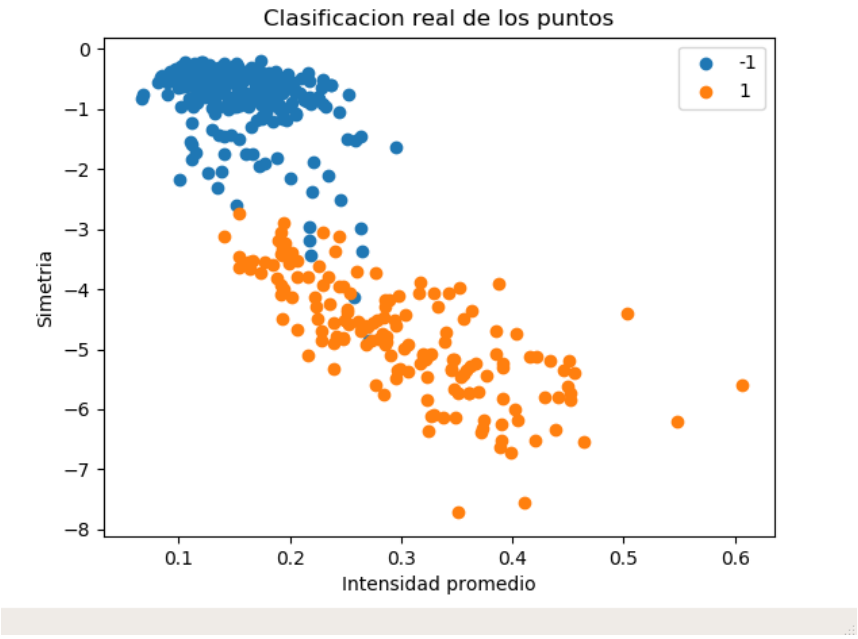
4. Encontrar un mínimo local o global depende totalmente del punto inicial donde se empieza a recorrer la función. Después, habría que encontrar una tasa de aprendizaje ajustado a la función, no muy grande para no producir *overshooting* ni demasiado pequeño para no tardar demasiadas iteraciones. Pero de nada sirve si el punto inicial que se toma lleva a un mínimo local. En definitiva, el punto inicial que se tome determinará si se llega a un óptimo local o global, pero la tasa de aprendizaje debe ser buena para encontrar ese mínimo en un tiempo razonable.

Ejercicio 2 – Regresión Lineal

1. En este ejercicio se utilizan dos algoritmos: pseudoinversa y gradiente descendente escotástico (SGD). El algoritmo de pseudoinversa encuentra los pesos a través del producto de la matriz de inputs inversa y el vector de los valores de la función con esos inputs. Es básicamente como si se “resolvieran” un conjunto de ecuaciones.

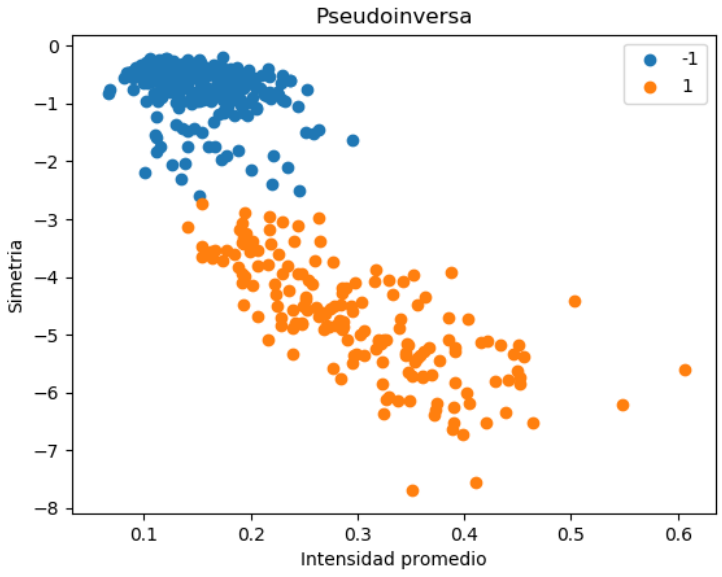
El algoritmo SGD trabaja igual que el gradiente descendente, pero con mini-batches en vez de con todo el conjunto. Una vez que se calcule un error menor al que buscamos o se supere el límite de iteraciones, se devuelve el mejor peso encontrado.

Partiendo de los datos:

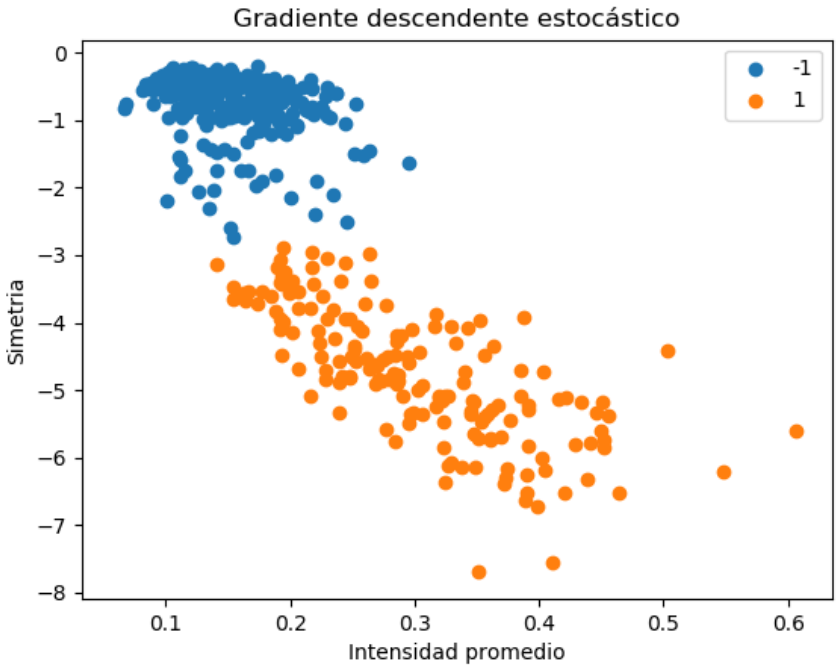


Los resultados de los algoritmos son:

```
Bondad del resultado para pseudoinversa:

Ein:  0.07918658628900395
Eout:  0.1309538372005259
```



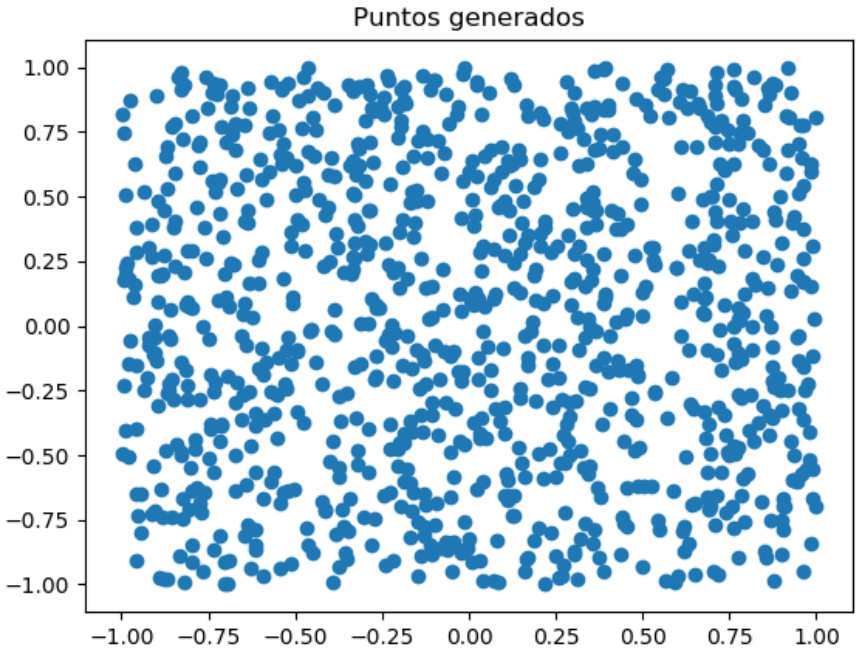
```
Bondad del resultado para grad. descendente estocastico:

Ein:  0.09085933431864283
Eout:  0.13754643210682677
```

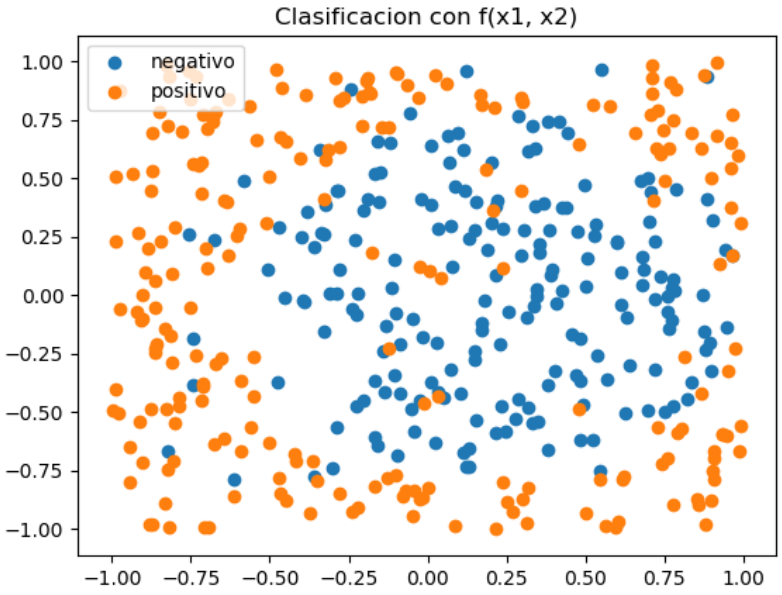


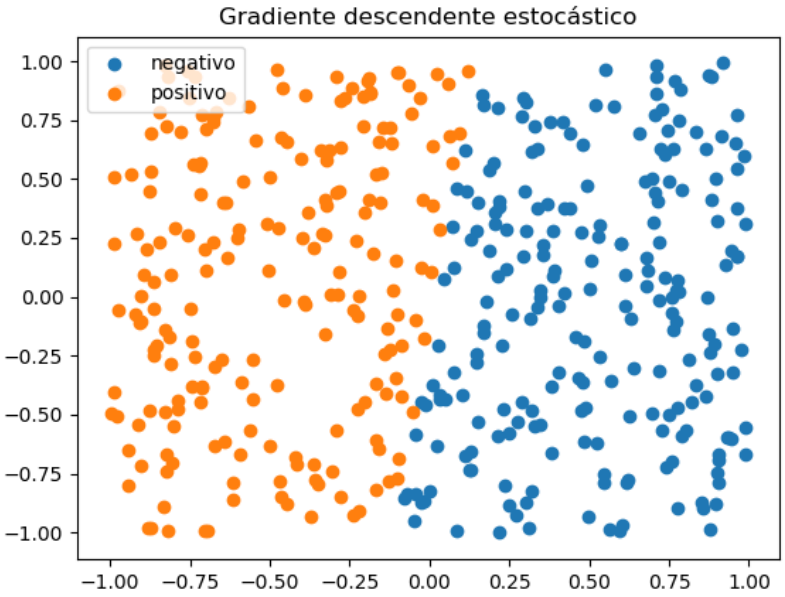
Los resultados son muy similares, se diferencian en la clasificación de solo 1 caso. Los porcentajes de error son muy parecidos, aunque el SGD tiene un poco más de error.

2. Como dice el ejercicio, se crean 1000 puntos al azar y se clasifican según la función dada. Esto al visualizarse se ve como una elipse donde los valores de dentro tienen una etiqueta y los de fuera otra, pero al meter ruido no se nota tanto. Esto se repite 1000 veces para ver el error medio que obtiene la función. Para calcular E_{out} generamos otras 1000 muestras al azar.

Ejecutamos una vez de prueba para visualizar los datos:







Como se puede ver, la clasificación es muy mala ya que como he dicho los datos forman una especie de elipse y el algoritmo SGD es una regresión lineal. Es imposible clasificar ese tipo de datos con un modelo lineal. Esto se ve en el error encontrado, que es enorme tanto para E_{in} y E_{out}.

```
Ein medio:  1.028868912724799
Eout medio:  1.0292848100914038
```