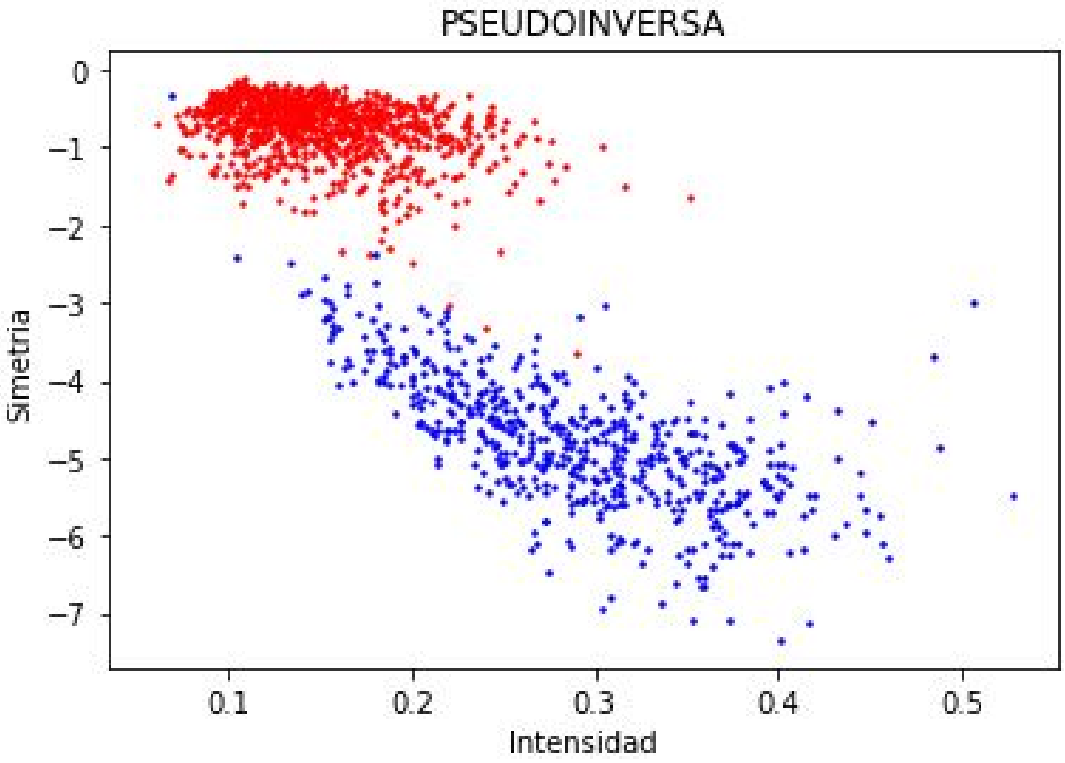
PRÁCTICA 1: Programación



Sergio Aguilera Ramírez

ÍNDICE

1. EJERCICIO 1

1.1 Implementación gradiente descendente

Este algoritmo consiste en la búsqueda de las coordenadas de una función donde su valor en la función sea igual o menor que el error (1e-14). Para ello, fijo el valor inicial de w (punto inicial), para la implementación uso un bucle donde la condición de parada es superar un número máximo de iteraciones o que el valor de la función en un punto determinado es menor que el error establecido, para calcular w en cada iteración utilizo la ecuación w = w - eta * gradE(w[0],w[1]) donde w es el punto actual, “eta” es el learning rate y gradE(w[0],w[1]) calcula los valores de las derivadas parciales de la función. Una vez encontrado las coordenadas (w), calculo el valor mínimo de la función con F(w[0] , w[1]).

```
def gradient_descent():
    iterations = 0
    salir = False
    w = initial_point

    while iterations < maxIter and salir == False:
        w = w - eta*gradE(w[0],w[1])
        salir = E(w[0],w[1]) < error2get
        iterations = iterations + 1
    return w, iterations
```

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

En este ejercicio calculo el valor mínimo y las coordenadas donde obtenemos el mismo tomando como punto inicial (u , v) = (1 ,1) y un learning rate de 0.01.

1.2.1 Derivadas parciales y gradE

- Las derivadas parciales de la función E(u,v) son:
    - Derivada de E con respecto a u:

$$d/du(E(u,v) = (u^2e^v - 2v^2e^{-u})^2) = 2(u^2e^v - 2v^2e^{-u})(2ue^v + 2e^{-u}v^2)$$

- Derivada de E con respecto a V:

$$d/dv(E(u,v) = (u^2e^v - 2v^2e^{-u})^2) = 2(u^2e^v - 2v^2e^{-u})(u^2e^v - 4e^{-u}v)$$

- El gradiente de E se muestra a continuación, en las funciones dEu(u,v) y dEv(u,v) corresponderían a las correspondientes derivadas parciales.

```
#Gradiente de E
def gradE(u,v):
    return np.array([dEu(u,v), dEv(u,v)])
```

1.2.2 Número de iteraciones para alcanzar un valor < 10^{-14}

Este algoritmo encuentra un valor inferior al error2get (10^{-14}) en la iteración 33 (como se muestra en la ejecución del código), este valor se denomina el valor mínimo de la función.

1.2.3 Coordenadas del valor mínimo

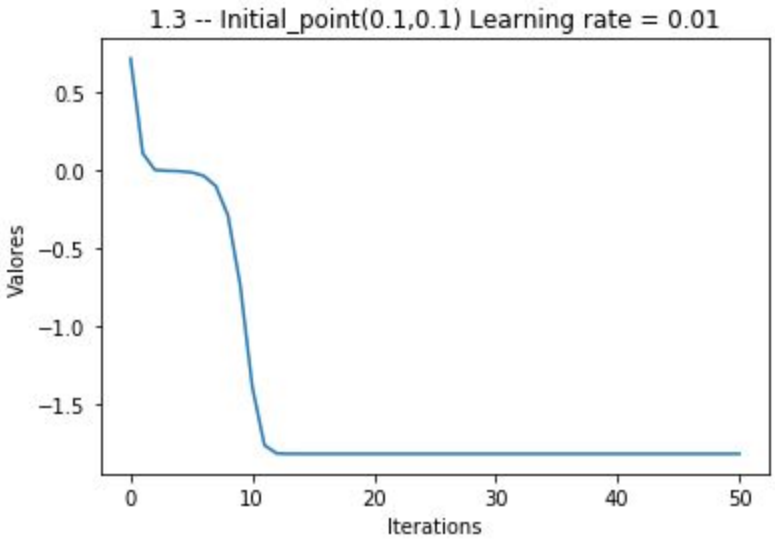
El valor mínimo anterior se obtuvo en las coordenadas (0.619207678450638 , 0.9684482690100487**).**

1.3 Función f(x,y) = x^2 + 2y^2 + 2 sin(2x) sin(2y)

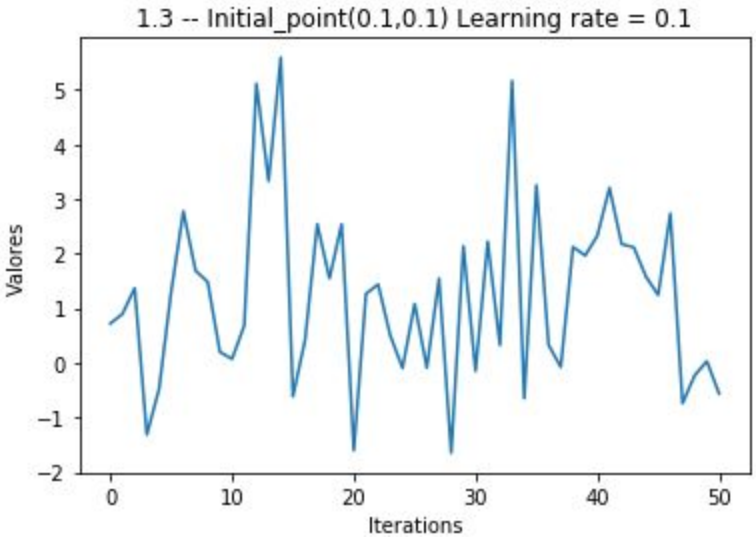
1.3.1 Gráficas

Se compara dos gráficas que muestran el descenso del valor de la función para el punto inicial (0.1 , 0.1) con diferente valor del learning rate $\eta_1 = 0.01$ y $\eta_2 = 0.1$.

1. Gráfica punto inicial (0.1 , 0.1) y $\eta_1 = 0.01$



2. Gráfica punto inicial (0.1 , 0.1) y $\eta_1 = 0.1$



Como podemos observar cuando el learning rate es mayor nos permite salir de mínimos locales, si observamos la segunda gráfica podemos ver como hay varios mínimos locales (picos inferiores) ya que como hemos dicho puede salir de ellos, esto permite encontrar otros valores mínimos, esto supone que no nos garantice devolvernos el valor mínimo más pequeño de la función.

1.3.2 Tabla

Punto inicial	Coordenadas donde alcanza el mínimo (x , y)	Valor mínimo
(0.1 , 0.1)	(0.24380496936478835, -0.23792582148617766)	(-1.8200785415471563)
(1 , 1)	(1.2180703013110816 , 0.7128119506017776)	(0.5932693743258357)
(-0.5 , -0.5)	(-0.731377460413803 , -0.23785536290157222)	(-1.3324810623309777)
(-1 , -1)	(-1.2180703013110816, -0.7128119506017776)	(0.5932693743258357)

1.3 Conclusión

De todos los resultados y experimentos estudiados podemos concluir que la búsqueda del mínimo global depende de varios factores, entre ellos podemos destacar la propia función ya que funciones más uniformes será más fácil encontrar el mínimo global más ràpidamente, también como he comentado antes un factor muy importante es la elección del learning rate (tasa de aprendizaje η) ya que si elegimos un learning rate muy alto esto provocará que demos saltos muy grandes pudiendo pasar de largo el mínimo global por lo contrario si establecemos un learning rate muy bajo necesitaremos más iteraciones para conseguir el mínimo global, al igual que la elección de las condiciones de parada ya que es difícil ajustar todos los parámetros.

2. EJERCICIO 2 (Regresión lineal)

2.1 PseudoInversa y SGD

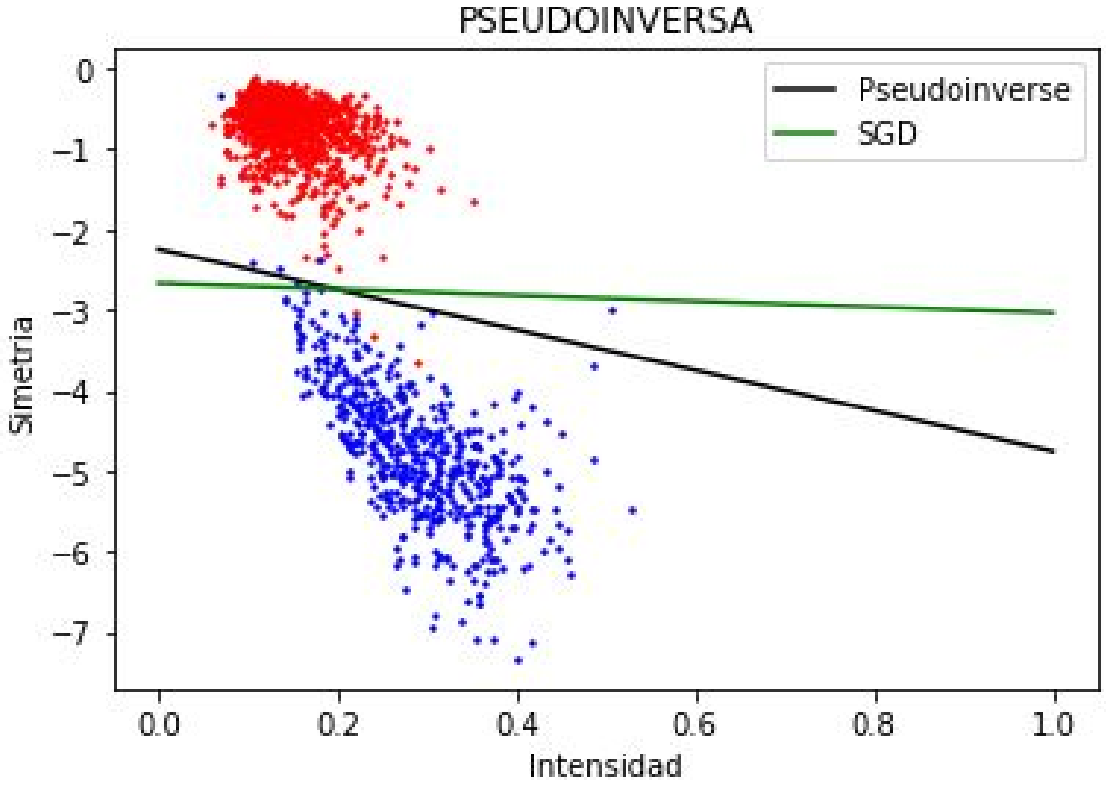
En este ejercicio he implementado dos algoritmos, el algoritmo de pseudoinversa y el gradiente descendente estocástico. Primero separo los datos x en sus respectivas clases para poder representarlos en el plot. El algoritmo de la pseudoInversa consiste en calcular la inversa de los datos de "x", para ello he hecho uso de la función de numpy np.linalg.pinv() y hacer el producto matricial con las etiquetas ("y"), esto nos proporciona un vector de dos componentes con el cual representamos el hiperplano y proporcionamos la bondad del resultado (Ein y Eout), en segundo lugar he implementado el algoritmo del gradiente descendente estocástico el cual consiste en separar los datos en minibatches y realizar el estudio sobre estos subconjunto, establecemos el punto inicial a 0, realizamos un bucle hasta que llegamos al máximo de iteraciones o el error es mayor que el umbral, fijamos un grupo de minibatch aleatorio y lo estudiamos con la fórmula w[t] = w[t] - learning_rate * $\sum_{n \in minibacth} x_{nt}$ * (transpuesta(w)*x_n - etiq[p]). Los resultados obtenidos de los dos algoritmos son:

Bondad del resultado para Pseudoinversa

Ein:	0.07918658628900384
Eout:	0.13095383720052572

Bondad del resultado para SGD

Ein:	0.0824297591617955
Eout:	0.13904497327237528

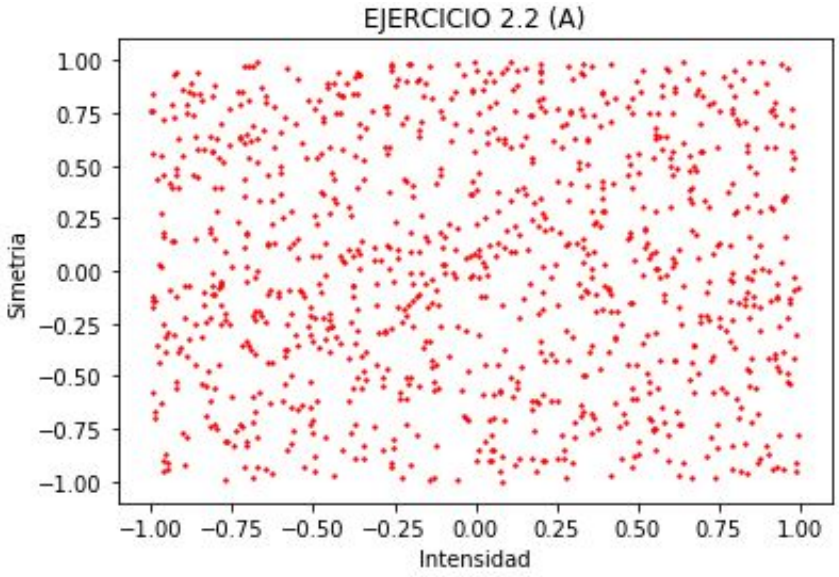


Como podemos observar los resultados obtenidos por el algoritmo de pseudoinversa son más óptimos que los obtenidos por el SGD, ya que esta separa mejor las dos clases por lo tanto la bondad del error es menor.

2.2 Experimentos

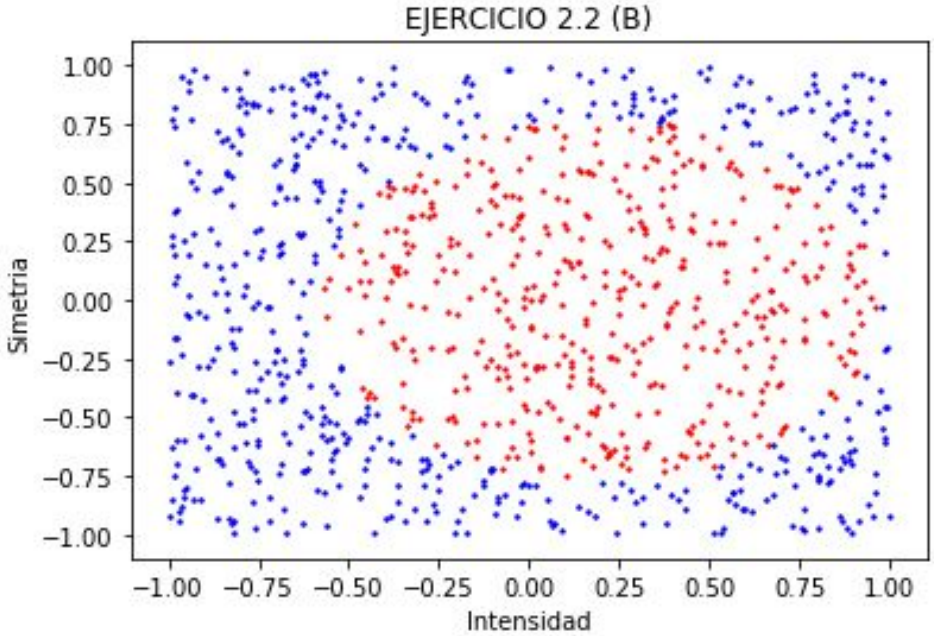
2.2.1 Generar 1000 puntos(simula_unif)

En este experimento he creado 1000 puntos diferentes utilizando la función sima_unif la cual genera valores aleatorios comprendidos en el rango [-size, size], el cual para nosotros el [-1.1], con ello he obtenido la siguiente gráfica:



2.2.2 Establecer etiquetas y generar ruido

En este experimento he asignado etiquetas a lo datos obtenidos en el experimento A, como indica el ejercicio he utilizado la función $\text{sign}((x_1-0.2)^2+x_2^2-0.6)$, cuyo funcionamiento es devolver +1 si el resultado es positivo o -1 si el resultado es negativo. Una vez establecidas las etiquetas genero ruido en dichas etiquetas, es decir, cambio el signo de forma aleatoria a un 10% de las etiquetas, cojo un array de índices aleatorio entre 0 y etiquetas.size de tamaño 100 ya que el 10% de 1000 es 100, y camibo el signo de las mismas. La gráfica queda de la siguiente forma:



2.2.3 Ajuste del conjunto de datos a-b

En este experimento ajusto un modelo de regresión lineal al conjunto de datos de los experimento a y b, para ello establezco un vector de características añadiendo (1, x_1 , x_2) y con dicho vector calculo el ajuste con el algoritmo del gradiente descendente estocástico (SGD), obteniendo un Ein de 0.9139207386227731, este valor es alto ya que no consigue encontrar de manera óptima un hiperplano que separe las dos clases.

2.2.4 Ejecución a-c 1000 veces

Este experimento consiste en realizar 1000 veces los pasos de a-c y calcular los datos Ein y Eout medios obtenidos, para ello en cada iteración establezco 1000 puntos y sus respectivas etiquetas y realizo el ajuste de Ein (train), también en cada iteración genero otros 1000 puntos para calcular el Eout (test), realizo la sumatoria de los diferentes Ein y Eout y calculo la media dividiendolos entre 1000 (número de iteraciones), este experimento tarda bastante, he intentado optimizar el código todo lo posible aun así tarda continua tardando bastante.

Ein medio:	0.895383767136494
Eout medio:	0.899666382858304

2.2.4 Valoración

Como podemos observar los valores medios obtenidos son altos, por lo que concluimos que el modelo lineal no se ajusta bien a la función de los datos, es decir, no encuentra un división buena entre las diferentes clases. Esto puede suceder por que los datos de las respectivas clases esten muy dispersos.

3. EJERCICIO 2.1 (Método de Newton)

El método de Newton consiste en la búsqueda de las raíces de una función mediante el uso de las derivadas de dicha función, las raíces son aquellos valores cercanos a 0, este método no garantiza la convergencia global. Para el cálculo de las raíces de los distintos puntos de inicio (0.1,0.1) (1.0 , 1.0) , (-0.5 , -0.5) , (-1.0 , -1.0), utilizo un bucle while con la condición de para de llegar a un máximo de iteraciones o que el valor absoluto de restar el valor de la función de la iteración anterior con la actual es menor que el umbral establecido, dentro del bucle creo la matriz heussiana utilizando la segunda derivada de u, la segunda derivada parcial de F respecto a u, la segunda derivada parcial de F sobre v y la segunda derivada de v, con dicha matriz (heussiana) calculamos w mediante la fórmula w = w - learning rate * np.dot(np.linalg.pinv(heussiana),gradF(w[0],w[1]), donde np.dot es el producto matricial y np.linalg.pinv es la inversa de la matriz heussiana, con los valores devueltos de dicho algoritmo obtengo los siguientes resultados:

Punto Inicial	Coordenadas de la raíces (x , y)	Valor de F
(0.1,0.1)	(0.010271854433189941 , 0.01017955728441247)	(0.008557361496128136)
(1.0 , 1.0)	(0.9562839873620104 , 0.9780230533613848)	(2.9022064979076094)
(-0.5 , -0.5)	(-0.478485983020264 , -0.4894412149727393)	(0.7259225374458934)
(-1.0 , -1.0)	(-0.9562839873620104 , -0.9780230533613848)	(2.9022064979076094)