Memoria Práctica 1

Aprendizaje Automático

Christian Vigil Zamora
3º - Grupo 2
Marzo,2019

Índice

1. EJERCICIO SOBRE LA BÚSQUEDA ITERATIVA DE ÓPTIMOS

Gradiente Descendente

1. Implementar el algoritmo de gradiente descendente.

El algoritmo de gradiente descendente tiene como objetivo aproximarse al óptimo, es decir al punto dónde se alcanza el mínimo. Su funcionamiento es muy sencillo, partimos de un punto inicial con el que para ir avanzando hacia el óptimo, obtiene la pendiente de la función en el punto en el que nos encontremos y puesto que estamos en un espacio multidimensional, usa las derivadas parciales. El hecho de obtener la pendiente mediante las derivadas parciales de la función es lo que se conoce como Gradiente. El Gradiente nos indica pendientes mayores, por lo que el algoritmo usa el valor del Gradiente restando, es decir vamos actualizando las coordenadas de búsqueda del óptimo restándole el valor del Gradiente obtenido al punto anterior. Esa actualización está condicionada por el valor del learning rate, que básicamente nos dice cuánto avanzamos en la búsqueda. Una vez que tiene las coordenadas del punto donde se alcanza el mínimo, el valor de ese mínimo se obtiene evaluando ese punto en la función que estemos considerando, la función coste. Pese a ser su objetivo encontrar el punto mínimo, el algoritmo tiene dos posibles salidas para evitar un exceso de cálculo, por ello se estima un número de iteraciones máximo y una comprobación de que si el error calculado para el punto que estemos evaluando es menor que un umbral dado, se sale.

Código ilustrativo de lo explicado:

```
# Algoritmo de Gradiente Descendente
def gradient_descent(f,df,eta,maxIter,error2get,initial_point,consider_error):
    w = initial_point # Inicializo w con el valor del punto inicial
    # Para controlar las iteraciones
    iterations = 0
    # Valor de la función en el punto inicial
    value = f(w[0],w[1])
    # Lista con los valores obtenido hasta alcanzar el mínimo
    values = []
    while iterations < maxIter: # Condición de salida
        iterations += 1
        # Actualizo las coordenadas restandole el valor del gradiente
        w = w - eta * df(w[0],w[1])
        # Obtengo el valor en esas coordenadas
        value = f(w[0],w[1])
        values.append(value)
        if value < error2get and consider_error: # Condicion de salida
            break;
    # Devuelve las coordenadas del punto minimo, su valor y la lista con
    # los valores que se han ido obteniendo en las coordenadas
    return w, iterations, values
```

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)

Para mostrar la expresión del gradiente previamente hay que calcular la derivada parcial de E con respecto a u y v.

- Derivada parcial de E con respecto a u:

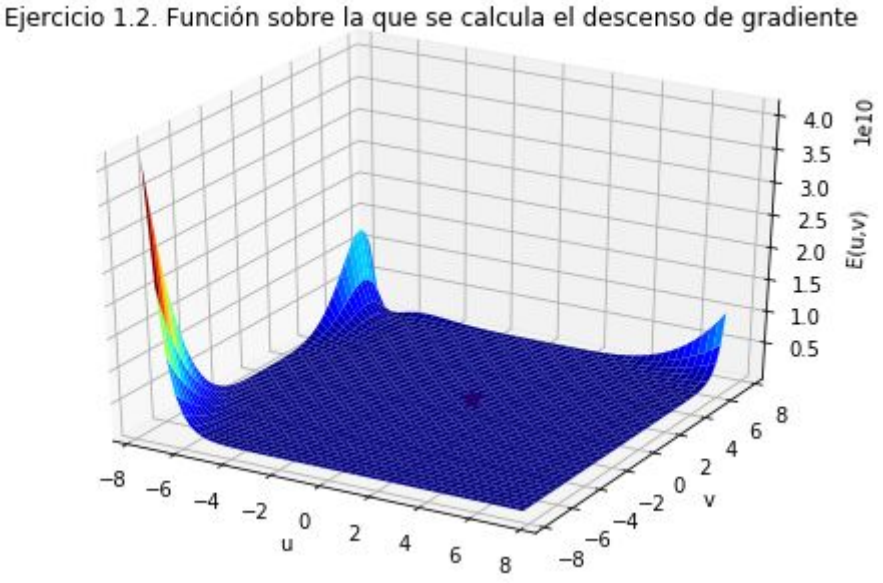
$$\frac{\partial E}{\partial u} = 2 * (u^2e^v - 2v^2e^{-u}) * (2ue^v + 2v^2e^{-u})$$

- Derivada parcial de E con respecto a v:

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

Finalmente, la expresión del gradiente de la función E(u,v) sería una tupla con el valor de evaluar un punto en la derivada parcial con respecto a U y con respecto a V:

$$\text{GradE(u,v)} = [\frac{\partial E}{\partial u}, \frac{\partial E}{\partial v}]$$



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)

El algoritmo tarda 33 iteraciones en obtener por primera vez un valor inferior a 10^-14.

```
* APARTADO B:

Numero de iteraciones:  33
```

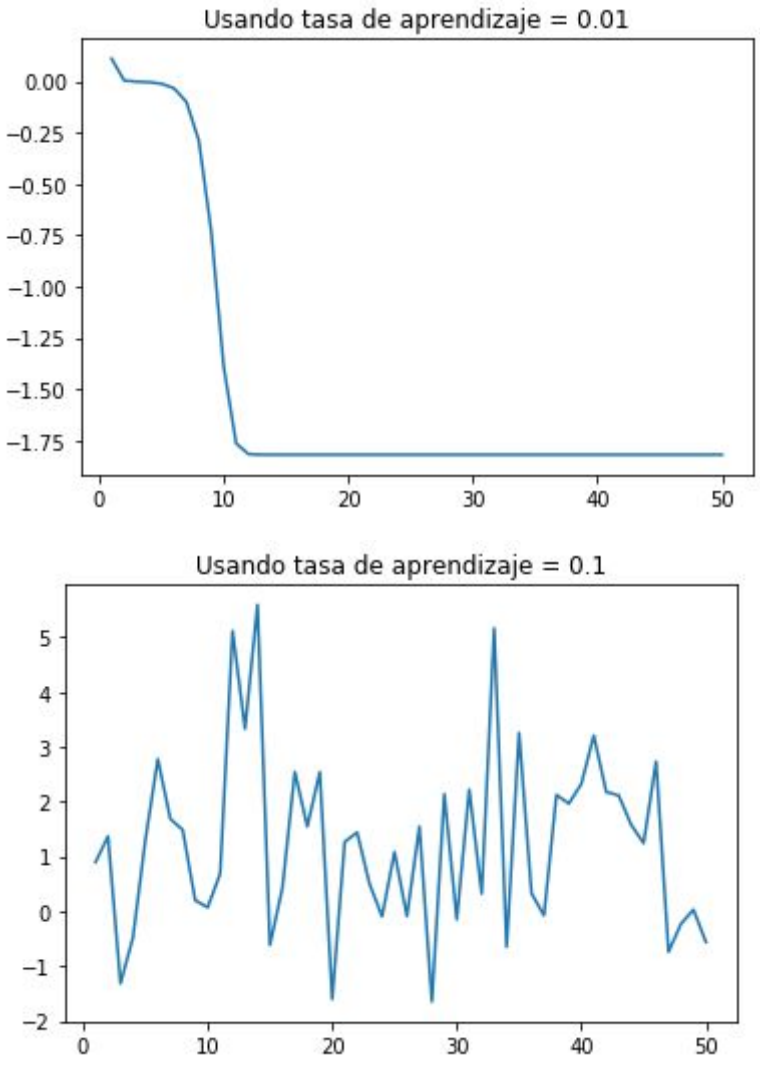
c) ¿En qué coordenadas (u, v) se alcanzó por primera vez un valor igual o menor a 10^-14 en el apartado anterior.

Por primera vez, se alcanzó un valor igual o menor a 10^-14 en las coordenadas u = 0.619207678450638 , v = 0.9684482690100487.

```
* APARTADO C:

Coordenadas obtenidas: ( 0.619207678450638 ,  0.9684482690100487 )

Valor minimo obtenido:  5.99730090741157e-15
```

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

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 .



La principal diferencia que se observa entre ambas gráficas es que en la primera se evidencia el descenso de la pendiente hacia el mínimo mientras que en la segunda no. Eso es debido principalmente al valor del learning rate. En la primera gráfica, se usa un learning rate de 0.01, lo que supone que avanzamos muy poco en cada aproximación a las coordenadas del mínimo. Tener un learning rate bajo como es 0.01 partiendo de unas coordenadas pequeñas como ocurre en éste ejercicio tiene el riesgo de que el algoritmo quede atrapado en un mínimo local con mayor facilidad, lo cual se evidencia en la gráfica. En cambio, en la segunda gráfica, usando un learning rate de 0.1 se avanza demasiado en cada iteración, es por eso que la pendiente de la función no para de variar, y por lo tanto, no se alcanzan muchos de los mínimos locales.

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.

Punto Inicial	Coordenada X	Coordenada Y	Mínimo
[0.1, 0.1]	0.24380497	-0.23792582	-1.82007854
[1.0, 1.0]	1.2180703	0.71281195	0.59326937
[-0.5, -0.5]	-0.73137746	-0.23785536	-1.33248106
[-1.0, -1.0]	-1.2180703	-0.71281195	0.59326937

```
* APARTADO B:

--- Tabla con los valores obtenidos: ---

P.Inicial  CoordenadaX    CoordenadaY     Minimo
[0.1,0.1]   [0.24380497]  [-0.23792582]  [-1.82007854]
[1.0,1.0]   [1.2180703]   [0.71281195]   [0.59326937]
[-0.5,-0.5] [-0.73137746]  [-0.23785536]  [-1.33248106]
[-1.0,-1.0] [-1.2180703]  [-0.71281195]  [0.59326937]
```

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

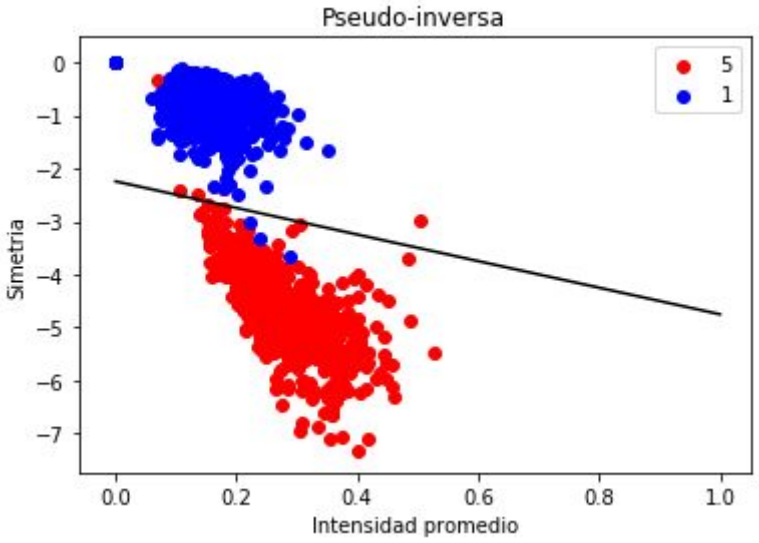
Mi conclusión es que encontrar el mínimo global de una función arbitraria es un objetivo bastante complejo del que dependen varios factores. En primer lugar, el ajuste del learning rate me parece la tarea más dificultosa puesto que hay que estimar cuánto se avanza en cada iteración, teniendo en cuenta los riesgos que tiene aproximarse a valores tanto altos como bajos. Por un lado existe el riesgo de que con un learning rate bajo te quedes atrapado en un mínimo local, y con un valor alto, hay una probabilidad alta de que te saltes valores mínimos.
Por último, considero que elegir el número de iteraciones máximo y el valor de epsilon, son dos tareas que van cogidas de la mano, ya que las iteraciones deben ser suficientes para que el algoritmo pueda llegar a converger y el valor de epsilon no debe ser muy alto para no quedarnos con un mínimo local.

2. EJERCICIO SOBRE REGRESIÓN LINEAL

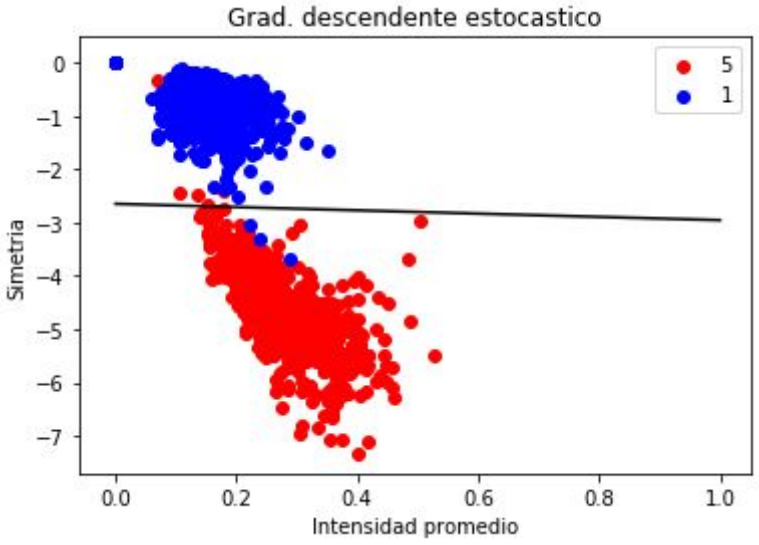
Este ejercicio ajusta modelos de regresión a vectores de características extraídos de imágenes de dígitos manuscritos. En particular se extraen dos características concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

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 pseudoinversa 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). (usar Regress_Lin(datos; label) como llamada para la función (opcional)).

Para el algoritmo de la pseudoinversa, he recurrido a una función muy útil de la librería Numpy, 'linalg.pinv()', la cual calcula la pseudo-inversa de la matriz 'x' directamente, usando la pseudo -inversa de Moore-Penrose. Para obtener la solución, aplico mínimos cuadrados a esa pseudo-inversa y al array que contiene las etiquetas de los datos, por lo que el algoritmo consiste en la suma de productos de los componentes de la pseudo-inversa de la matriz de datos por el array de etiquetas.



El algoritmo de Gradiente Descendente Estocástico tiene el mismo funcionamiento que el Gradiente Descendente explicado anteriormente con la particularidad de que no se considera la muestra total, sino divisiones de esa muestra, los llamados minibatch, sobre los que se va calculando el gradiente. En este algoritmo, al no tener una función de coste ni derivadas parciales, hay que desglosar con más detalle las operaciones. He considerado 64 como tamaño óptimo de los minibatch. Su funcionamiento sería el siguiente: el algoritmo va a iterar continuamente mientras no se satisfaga ninguna condición de salida. En cada iteración, selecciono 64 elementos de la muestra total de forma aleatoria, es decir, un minibatch y calculo el gradiente para esos elementos mediante la multiplicación de ese minibatch por el resultado de restarle a la clase H del minibatch sus etiquetas correspondientes. Una vez que tiene el gradiente para ese minibatch, actualiza el punto que estamos evaluando (w) y así sucesivamente hasta aproximarse al óptimo. He considerado adecuado para la resolución de éste apartado un learning rate de 0.001 y como condiciones de salida he estimado 100 iteraciones máximas y un valor de epsilon de 0.085.



Para representar correctamente la recta de regresión, parto de la ecuación y = w0 + w1x1 + w2x2. Igualo y = 0 y puesto que la muestra se encuentra en el intervalo [0,1], obtengo el valor de x2 considerando que x1 = 1, por lo que queda $x2 = -\frac{w0}{w2} - \frac{w1}{w2}$.

A continuación muestro los resultados Ein e Eout obtenidos con los diferentes algoritmos:

```
Bondad del resultado para pseudoinversa:

Ein:  0.079186586289003 84
Eout:  0.13095383720052572
```

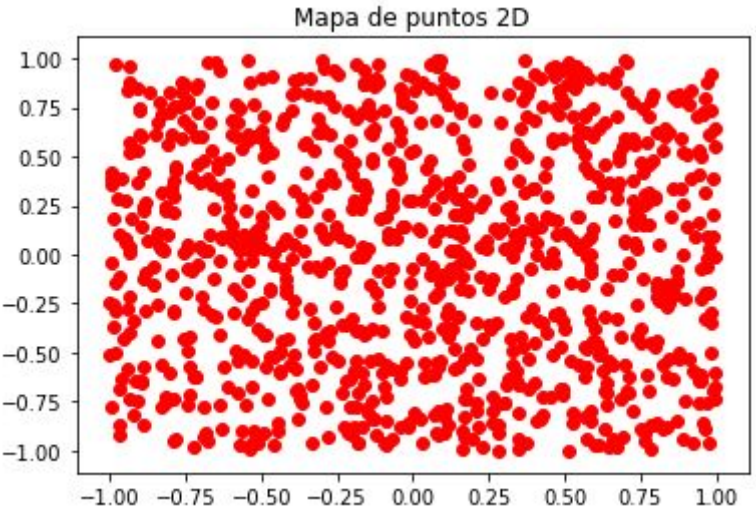
```
Bondad del resultado para grad. descendente estocastico:

Ein:  0.08497550135426318
Eout:  0.13713398760947976
```

En ambos, algoritmos considero que la bondad de sus resultados es bastante aceptable. El error de ajuste Ein de ambos es ínfimo, lo que supone que dentro de la muestra ambos algoritmos se comportan muy bien, ajustando el modelo de regresión a los datos de forma casi perfecta. Aún así, para comprobar cómo se comportan los algoritmos frente a datos con los que no se han entrenado, se obtiene el error de ajuste Eout que es el error obtenido por ambos algoritmos con datos fuera de la muestra. Pese a haber obtenido un valor de Eout más elevado que con los datos de la muestra, sigue siendo un valor de error más que aceptable, dando lugar a la conclusión de que ambos algoritmos ajustan modelos de regresión casi a la perfección tanto a datos que conocen como a los que no.

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].

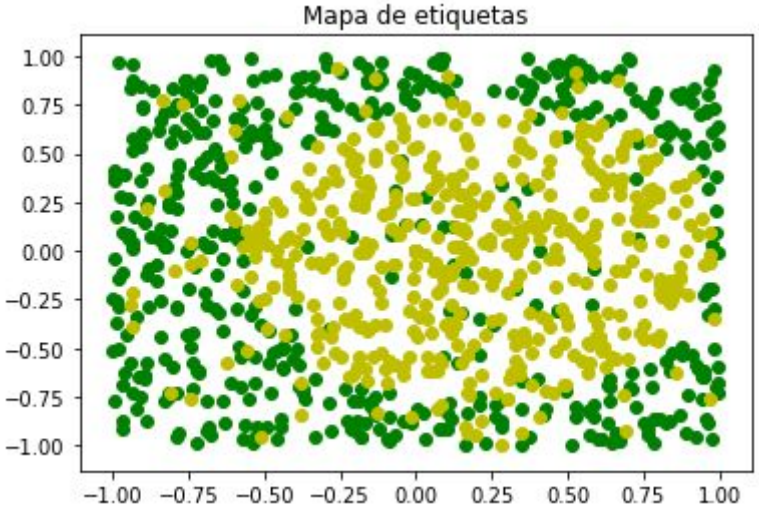
- **EXPERIMENTO :**

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



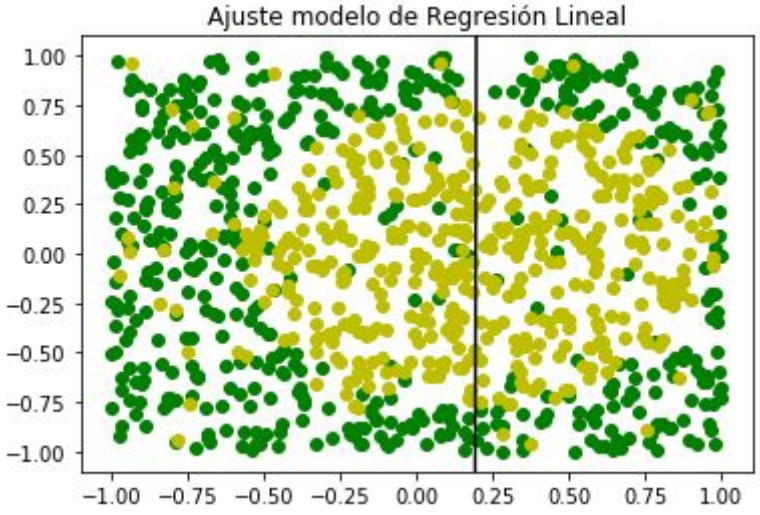
b) Consideremos la función f(x1, x2) = sign((x1 - 0,2)^2 + x2^2 - 0,6) que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10% de las mismas. Pintar el mapa de etiquetas obtenido.

En primer lugar, asigno una etiqueta a cada punto de la muestra mediante la función del enunciado, obteniendo etiquetas con valores 1 ó -1. Para introducir ruido sobre ellas, lo que hago es seleccionar aleatoriamente 100 y alternar su valor. Por último, separo la muestra en dos, según su etiqueta y represento el mapa de etiquetas.



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).

En éste apartado simplemente añado una columna de 1's a la muestra de datos inicial y le ajusto un modelo de regresión lineal usando como learning rate = 0.001, maxIter = 100 y valor de epsilon = 0.095 para el Gradiente Descendente Estocástico.




```
* APARTADO C:

Bondad del resultado para grad. descendente estocastico:

Ein:  0.9280543201843053
```

El error de ajuste no es especialmente bajo, pero considero que obtener un error por debajo de 1 en éste caso es aceptable, dada la distribución de los datos de la muestra.

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.

En éste apartado, como bien indica el enunciado hay que ejecutar 1000 veces lo ya realizado en los apartado a), b) y c). Al algoritmo de Gradiente Descendente Estocástico le paso un valor de learning rate de 0.01, un número máximo de iteraciones = 10 y un valor de epsilon = 0.95. Por cada una de las 1000 iteraciones del experimento, voy acumulando en una variable el error Ein obtenido en cada llamada al algoritmo. Para el valor de Eout, genero 1000 puntos nuevos e igualmente, acumulo en una variable el error Eout obtenido en las iteraciones. Finalmente, presentos los resultados dividiendo los valores Ein e Eout obtenidos entre el tamaño del experimento, en éste caso 1000. Resultado:

```
* APARTADO D:


-- Tiempo de cálculo aproximado: 50-60s --

Valor medio Ein de las 1000 muestras:  0.9342924997238247
Valor medio Ein de las 1000 muestras:  0.9395011586206534
```

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.

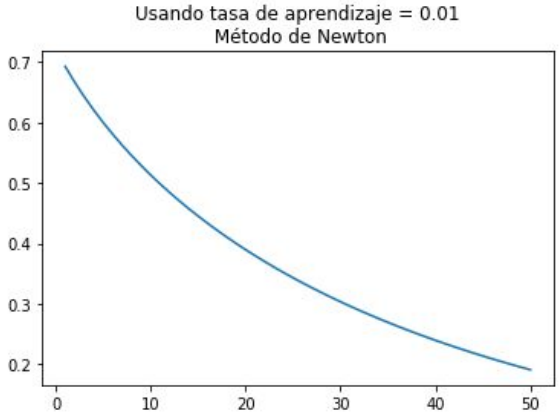
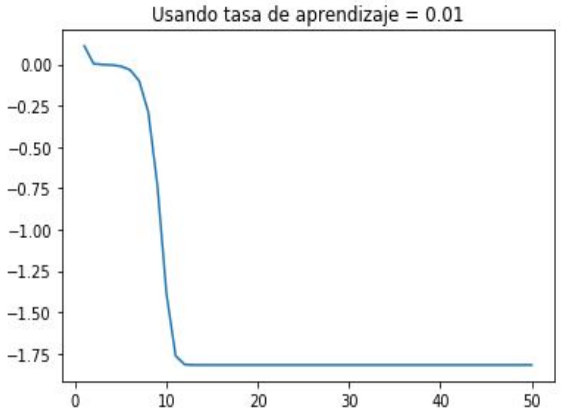
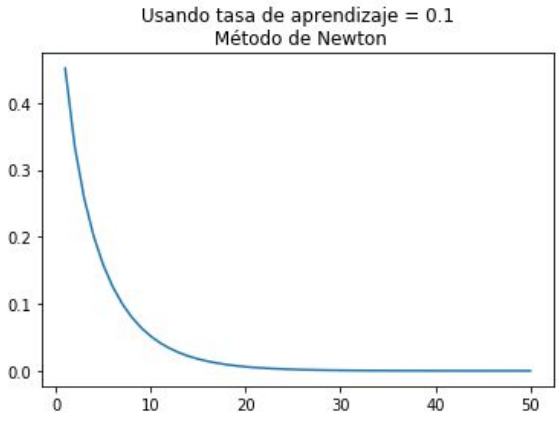
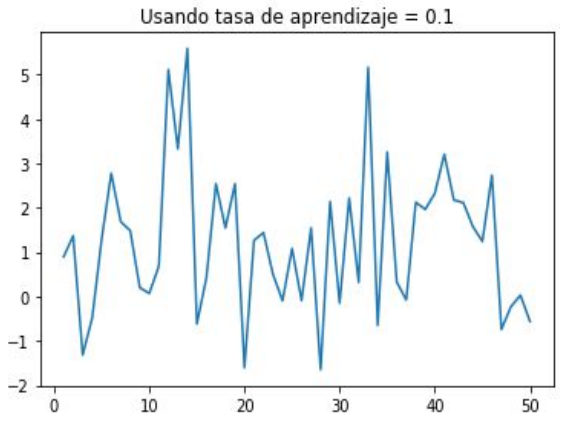
El ajuste con este modelo lineal considero que es bastante malo, pero tiene su motivo. Antes de dar explicaciones, debo decir que el ajuste da unos resultados muy similares tanto con muestras que conoce como con muestras que no conoce, por lo que en ese aspecto si cumple, pero eso no quiere decir que el ajuste sea bueno. El principal motivo es que el ajuste con un modelo lineal no es el más adecuado para este tipo de muestra, puesto que está generada entre los puntos [-1,1],[-1,1], lo que da lugar a una representación circular de los datos como hemos visto en el apartado a), b) y c). El hecho de que los datos se representen de forma circular hace que trazar la recta de regresión sea una tarea muy complicada, pues los datos están distribuidos de forma no lineal. A eso hay que sumarle el matiz de aleatorizar el 10 % de las etiquetas, consiguiendo un plus de dificultad para que el modelo lineal consiga un buen ajuste.

2.1 BONUS

1. Método de Newton Implementar el algoritmo de minimización de Newton y aplicarlo a la función f(x, y) dada en el ejercicio 3. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

El funcionamiento de éste método consiste al igual que el Gradiente Descendente en iterar en busca de converger hacia un mínimo. Por lo tanto, se parte de un punto inicial, el cual se evalúa y se va actualizando ese punto en busca de la convergencia mediante la inversa de la matriz Hessiana, que marca la dirección de avance producto vectorial con el gradiente de la función en el punto que se esté evaluando.

- Generar un gráfico de como desciende el valor de la función con las iteraciones:

Método de Newton	**Gradiente Descendente**
	
	

Punto Inicial	Coordenada X	Coordenada Y	Mínimo
[0.1, 0.1]	0.00036805	0.00036412	1.09819292e-05
[1.0, 1.0]	0.94940858	0.9747153	2.90040761
[-0.5, -0.5]	-0.47524365	-0.48786929	0.72548275
[-1.0, -1.0]	-0.94940858	-0.9747153	2.90040761

```
--- Tabla con los valores obtenidos: ---

P.Inicial  CoordenadaX    CoordenadaY     Minimo

[0.1,0.1]   [0.00036805]  [0.00036412]  [1.09819292e-05]

[1.0,1.0]   [0.94940858]  [0.9747153]  [2.90040761]

[-0.5,-0.5] [-0.47524365]  [-0.48786929]  [0.72548275]

[-1.0,-1.0] [-0.94940858]  [-0.9747153]  [2.90040761]
```

- Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.

Para obtener los valores de la tabla, he utilizado en el Método de Newton un learning rate = 0.1 ya que como bien se observa en las gráficas anteriores, es con ese valor con el que obtiene una mejor convergencia hacia al mínimo. **Conclusiones:**
Ambos algoritmos tienen como objetivo converger hacia al mínimo, pero su rendimiento no es similiar del todo. En el Método de Newton el factor del learning rate condiciona aún más que en el Gradiente Descendente la búsqueda, de hecho como se observa en la primera gráfica, el Gradiente Descendiente obtiene un mínimo mucho más acertado que el Método de Newton. Por último, decir que su comportamiento es totalmente opuesto según el punto inicial, es decir, cuando el Método de Newton parte de un punto inicial próximo a la raíz, obtiene un mejor resultado que el Gradiente mientras que si el punto inicial está más alejado, el Gradiente obtiene una convergencia considerablemente mejor.