Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación
Grado en Ingeniería Informática
Departamento de Ciencias de la Computación e Inteligencia Artificial
Aprendizaje Automático
Curso 2019/2020
Grupo A2
Javier Rodríguez Rodríguez

Trabajo 1: Programación

1. Ejercicio sobre la búsqueda iterativa de óptimos:

1. Para este apartado lo primero que se pide es la implementación del algoritmo de gradiente descendente. El algoritmo lo que hace es actualizar valores de la función a partir de un punto arbitrario, disminuyendo según el valor de su gradiente por la tasa de aprendizaje. Por tanto, en la implementación es necesario establecer como parámetros el punto inicial w, la tasa de aprendizaje y el gradiente de la función, aprovechando la capacidad de Python de pasar funciones como parámetro. Además, se incorporan dos parámetros, epsilon y max_iters, que limitan el tiempo de ejecución del algoritmo según el orden de magnitud de las mejoras y la cantidad de iteraciones.

En la implementación, en vez de utilizar un ciclo interno para recorrer cada dimensión del punto, vectorizamos las operaciones, de forma que se actualicen los valores de cada dimensión de forma más eficiente.

2. Se considera la función $E(u,v) = (ue^v - 2ve^{-u})^2$, con los valores $(u,v) = (1,1)$ y $\eta = 0{,}1$

a) Lo primero que hay que hacer es determinar el gradiente de la función.
Por definición del gradiente,

$$\nabla E(u,v) = \left(\partial E(u,v)/\partial u, \partial E(u,v)/\partial v \right)$$

por lo que es necesario calcular las derivadas parciales respecto a u y v.
Respecto a u resulta

$$\partial E(u,v)/\partial u = \partial (ue^v - 2ve^{-u})^2/\partial u = 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u})$$

mientras que **respecto a v resulta**

$$\partial E(u,v)/\partial v = \partial (ue^v - 2ve^{-u})^2/\partial v = 2(ue^v - 2ve^{-u})(ue^v - 2e^{-u})$$

Por ende, **el gradiente resultante es de la forma**

$$\nabla E(u,v) = \left(2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}), 2(ue^v - 2ve^{-u})(ue^v - 2e^{-u}) \right)$$

b) La siguiente tarea consiste en determinar el mínimo de iteraciones del algoritmo necesarias para alcanzar un valor de E(u,v) menor a 10^{-14}. Como el algoritmo de Gradiente Descendente no calcula directamente el valor de E(u,v) en ningún momento, el mecanismo empleado consiste en ejecuciones sucesivas del algoritmo, partiendo de una iteración como máximo, evaluar E para el punto resultante, y compararlo con la cota de interés, 10^{-14}. Si está por

encima, se incrementa en uno el máximo de iteraciones y se vuelve a ejecutar el algoritmo. Apenas el resultado esté por debajo de la cota, indica el número de iteraciones. Este procedimiento dio como resultado que **toma 9 iteraciones del algoritmo alcanzar el valor de**

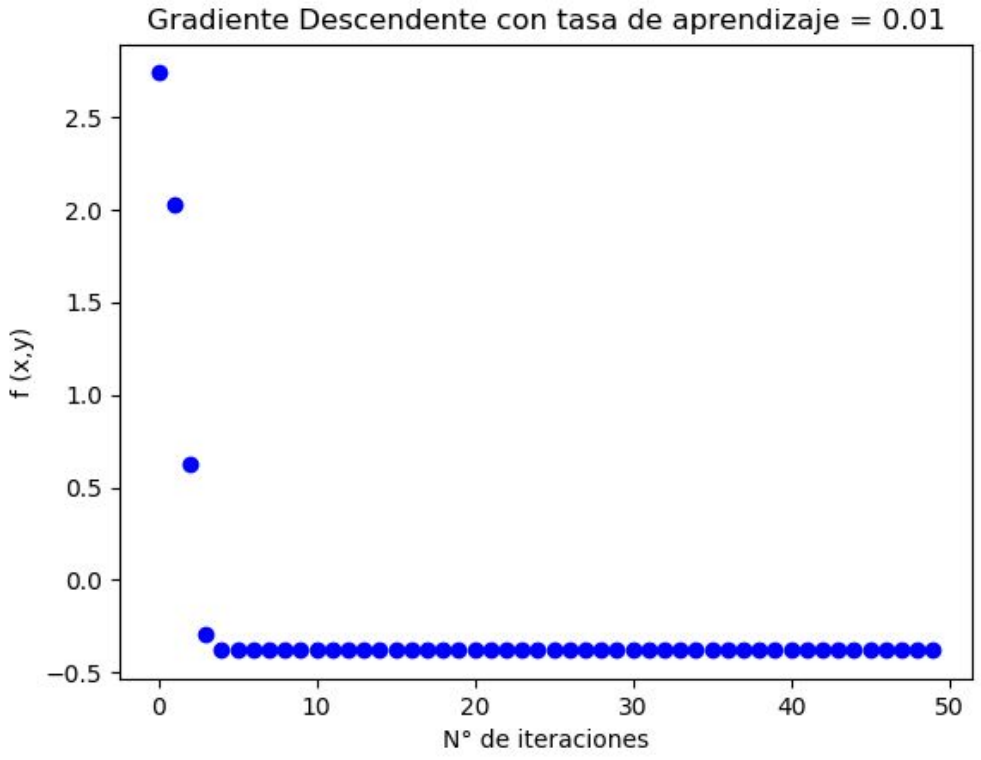
$$\mathbf{E(u,v) = 1{,}2086833944220747 * 10^{-15} < 10^{-14}}.$$

c) Para conocer las coordenadas (u,v) en las que se alcanza el valor anterior, basta con conservar el valor w retornado por el algoritmo de Gradiente Descendente, pues corresponde a dichos puntos y es con el que se calcula el E(u,v) presentado anteriormente. Dichos valores son

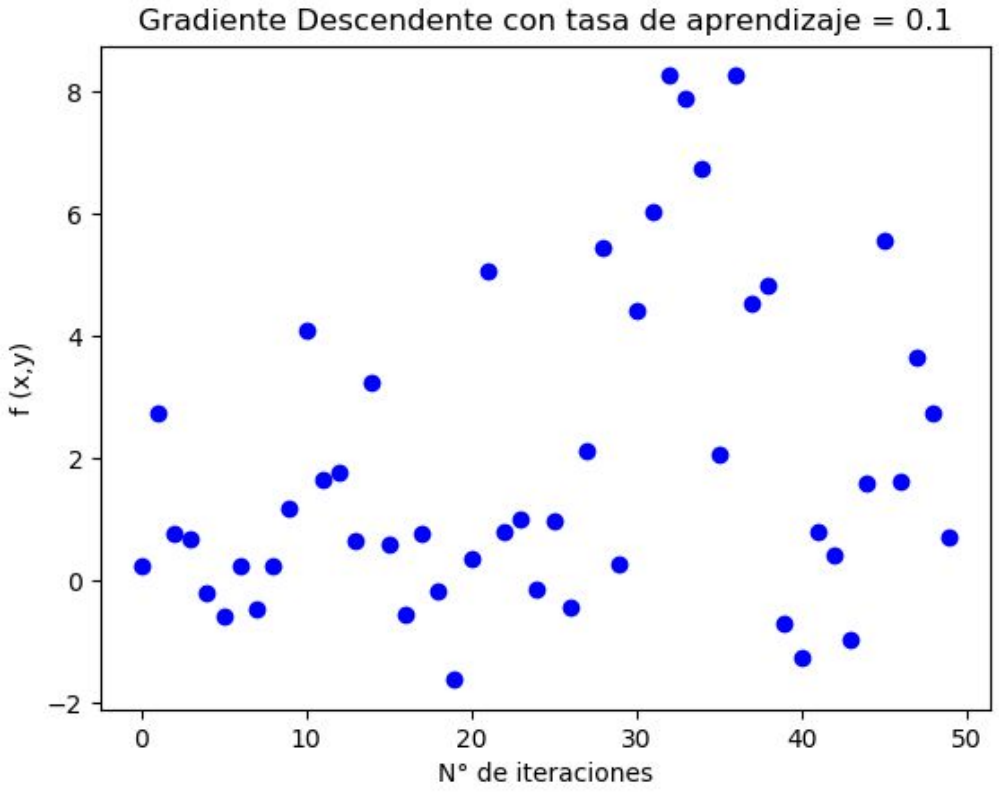
$$\mathbf{(u,v) = (0{,}04473629039778207;\ 0{,}023958714099141746)}$$

3. Se considera ahora la función $\mathbf{f(x,y) = (x-2)^2 + 2(y+2)^2 + 2\sin(2\pi x)\sin(2\pi y)}$

a) Teniendo como punto inicial $(x_0,y_0) = (1,-1)$ y $\eta = 0{,}01$ y un máximo de 50 iteraciones, se minimiza la función a través del algoritmo de Gradiente Descendente. El interés es observar cómo decrece el valor de f(x,y) a lo largo de la ejecución del algoritmo, por lo que es necesario evaluar la función en cada iteración del algoritmo. Para evitar modificar la implementación realizada, se opta por un mecanismo similar al empleado al contabilizar la cantidad mínima de iteraciones para superar una cota, pero en este caso siempre se realizan ejecuciones de una iteración, pero actualizando en cada ejecución el punto de partida por el resultado de la ejecución anterior, replicando así el comportamiento del algoritmo mientras se almacenan los valores de f(x,y). Una vez finalizadas las 50 iteraciones, a través de matplotlib se grafican los resultados, obteniendo:



Al repetir el experimento, pero con η = 0,1, la gráfica resultante es la siguiente:



Las diferencias resultan evidentes: para η = 0,1, el algoritmo es incapaz de alcanzar y converger en un punto mínimo, por lo que salta constantemente entre valores. Con η = 0.01, las disminuciones resultan mucho más controladas y converge rápidamente en el valor mínimo para ese punto de partida, sirviendo de ejemplo sobre el problema de optar por η muy altos deseando mejor eficiencia.

b) A diferencia del apartado anterior, que evidencia la importancia de la elección de η, ahora se plantea un experimento que evidencia cómo afecta la elección del punto inicial. Para ello, se llevan a cabo ejecuciones del algoritmo tomando como punto de partida cada uno de los siguientes puntos: (2,1, −2,1), (3, −3), (1,5, 1,5), (1, −1), con η = 0,01 y máximo de 500 iteraciones. La tabla resultante es la siguiente:

(x_0, y_0)	x	y	f(x,y)
(2,1, -2,1)	2.2438049693647883	-2.237925821486178	-1.8200785415471563
(3, -3)	2.7309356482481055	-2.7132791261667037	-0.38124949743809955
(1,5, 1,5)	1.779119517755436	1.0309456237906103	18.042072349312033
(1, -1)	1.269064351751895	-1.2867208738332965	-0.3812494974381

Analizando esta tabla resulta evidente el peso que conlleva la elección del punto inicial en el rendimiento del algoritmo, resultando puntos que inicialmente pueden parecer cercanos resultados tremendamente distintos. Resalta en particular el punto (3, -3), pues el valor de f(x,y) que considera mínimo tras la ejecución es de 18, notablemente por encima de los valores negativos cercanos a 0 que alcanza en el resto de ejecuciones. La razón de este fenómeno está sin duda asociada a posibles mínimos locales encontrados.

4. Tras la realización de los experimentos, queda en evidencia que la elección de los parámetros iniciales son clave en el correcto rendimiento del algoritmo, pero la razón de este fenómeno es debido a los mínimos locales. Ante una función arbitraria, es imposible determinar si un mínimo alcanzado es local o global. Una posible forma de solventar la dificultad es incrementar la tasa de aprendizaje, para así "salir" de mínimos locales, pero quedó probado que también puede llevar a no alcanzar nunca un mínimo, y por ende no convergiendo. El algoritmo de Gradiente Descendente no tiene ninguna "configuración óptima" de parámetros con la cual tener certeza de la globalidad del mínimo hallado. Para hallar el mínimo global habría de probar para todas las posibles combinaciones de punto de partida y tasa de aprendizaje, de forma que se obtenga aquella que converja en el mínimo global, pero dada la infinitud de posibles entradas, resulta computacionalmente inviable.

2. Ejercicio sobre regresión lineal:

1. Lo primero que se pide es estimar un modelo de regresión lineal para la identificación de dígitos manuscritos según intensidad promedio y simetría, distinguiendo en particular solamente los 1 y los 5. Es prerrequisito de los experimentos extraer correctamente la información de los ficheros e implementar los algoritmos a considerar: el algoritmo de la pseudoinversa y el Gradiente Descendente Estocástico.

Para la lectura de los datos basta con, utilizando PathLib, fijar la ruta y nombre de fichero: "datos/<nombre_fichero>.npy", y mediante NumPy volcar dichos datos en vectores.

Una vez obtenidos los datos, utilizamos la indización con vectores de NumPy para limitarnos a los datos de interés, los etiquetados como 1 y como 5, y

generamos su respectivo vector de etiquetas, pero esta vez utilizando las descritas: -1 si es un 1, 1 si es un 5. Además, a la matriz x se le añade al inicio una columna de 1, la cual representa el término independiente.

Para el algoritmo de la pseudoinversa tan solo es necesario emplear la función de pseudoinversa contenida en el paquete de álgebra lineal de NumPy, junto con la operación de multiplicación de matrices.

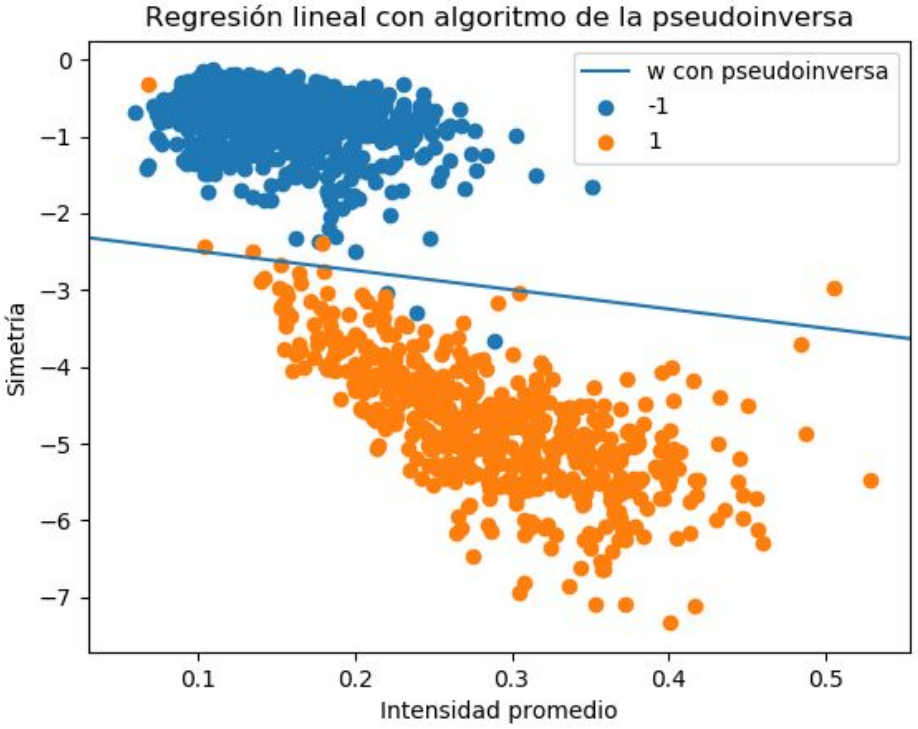
El Gradiente Descendente Estocástico, por otro lado, requiere reimplementar Gradiente Descendente pero involucrando minibatches, pequeños subconjunto aleatorio de datos bajo el cual se realiza la iteración. La verdadera dificultad de esta implementación respecto a la anterior es el desconocimiento de la función, y por ende, de su gradiente, el cual se aproxima mediante la expresión $2(x_n^T (x_n w - y_n))/M$. Esta forma vectorizada hace uso de la capacidad de NumPy para realizar el cálculo para cada elemento del minibatch de forma simultánea. Del resto, se ajusta a la implementación anterior.

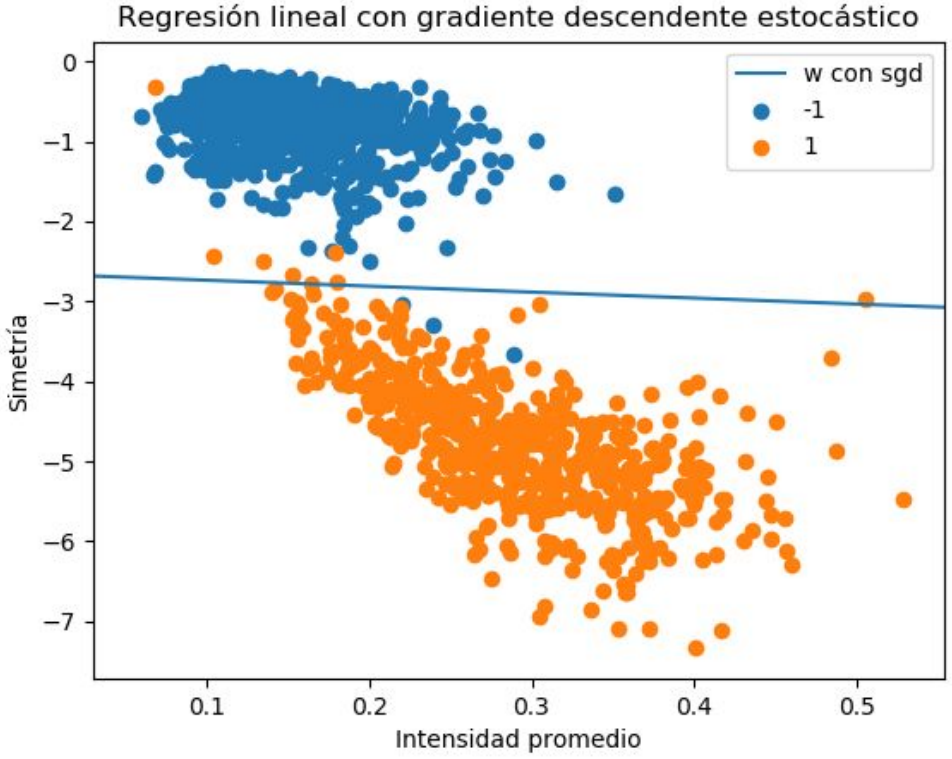
Para la ejecución de los algoritmos con estos datos, se obtienen los siguientes resultados:

Método	E_{in}	E_{out}
Algoritmo de la pseudoinversa	0.07918658628900398	0.13095383720052578
Gradiente Descendente Estocástico	0.0811104753636879	0.1330636510676983

En el Gradiente Descendente Estocástico se utiliza η = 0,01, máximo de 5000 iteraciones y minibatches de 64 elementos.

Si graficamos la nube de datos y la frontera de decisión que representa cada método, resultan las siguientes gráficas. La representación de la función resulta de sustituir en la ecuación de la recta y = mX + b los valores de m y b por sus cálculos a partir del w obtenido.

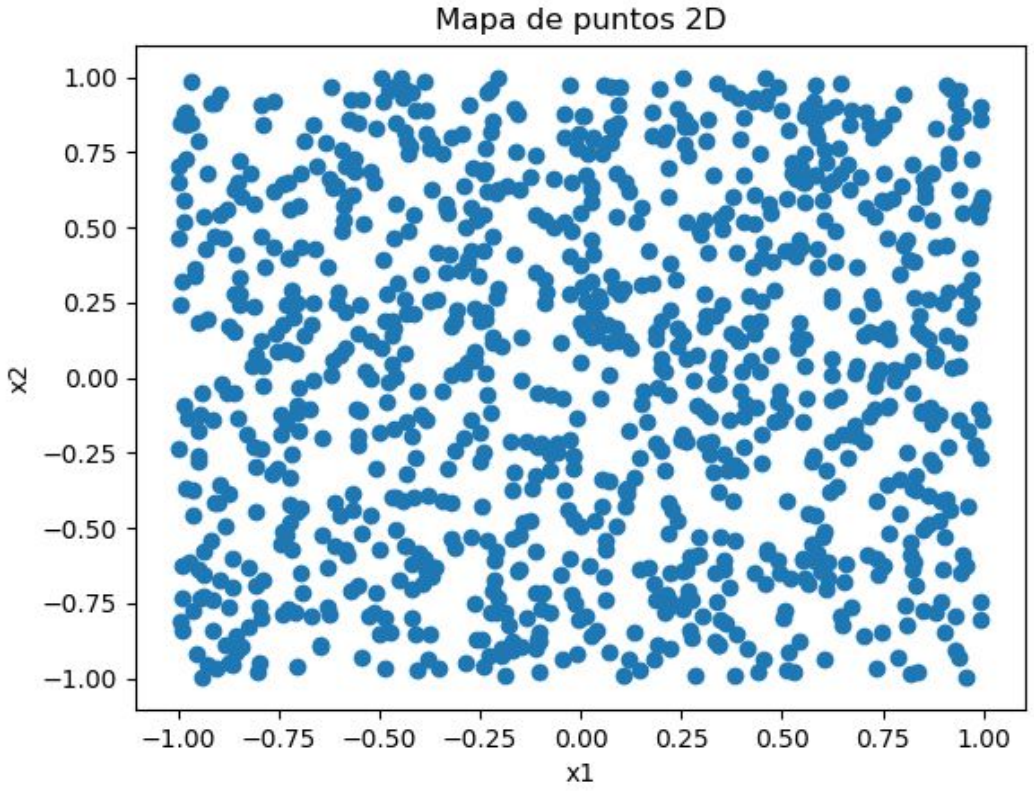




Comparando datos y gráficas encontramos que el algoritmo de la pseudoinversa es ligeramente mejor, pero las diferencias son despreciables para el problema en cuestión. Ambos resultan adecuados, con tasas de error cercanas a 0,1 para tanto dentro como fuera del conjunto de entrenamiento.

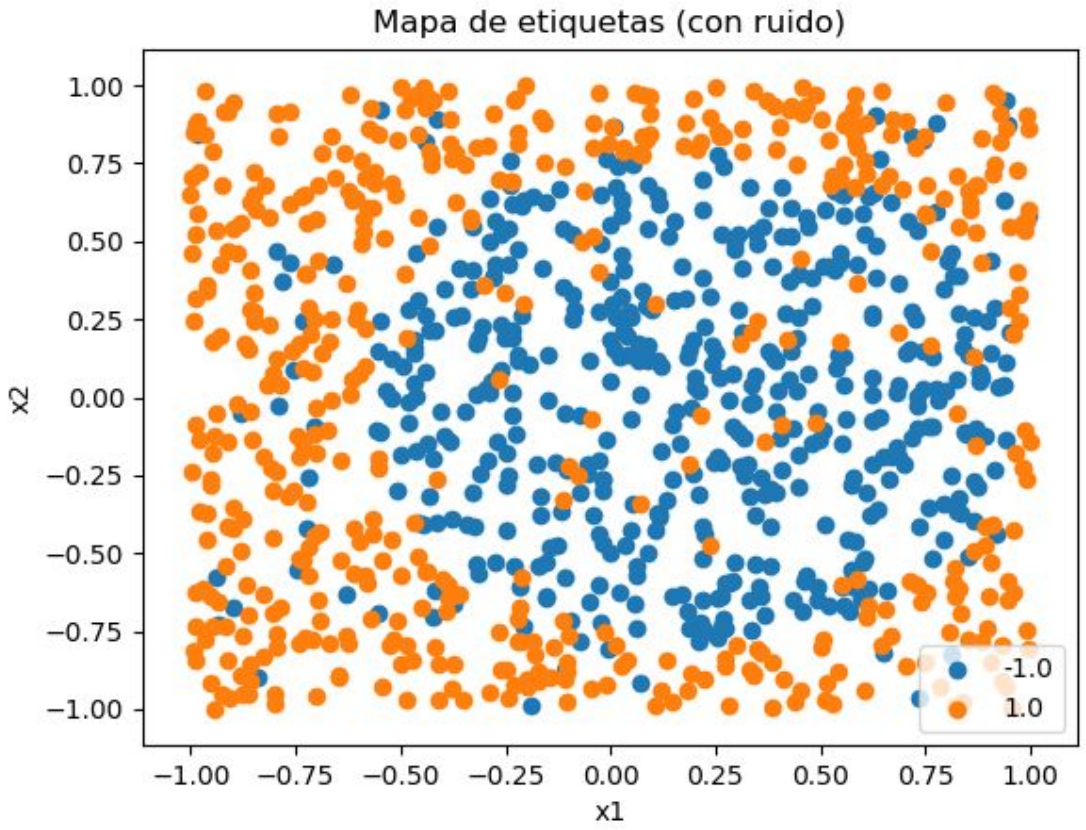
2. Se propone un experimento diferente, utilizando puntos 2D uniformemente muestreados en un espacio cuadrado. Para establecer dichos datos, el generador de números aleatorios de NumPy provee una función que genera valores uniformemente distribuidos en un intervalo cerrado directamente en forma de pares.

a) Se genera la muestra de 1000 pares aleatorios en el cuadrado [-1,1] x [-1,1] a través de la función previamente descrita. Al graficarlo, usando matplotlib, resulta:

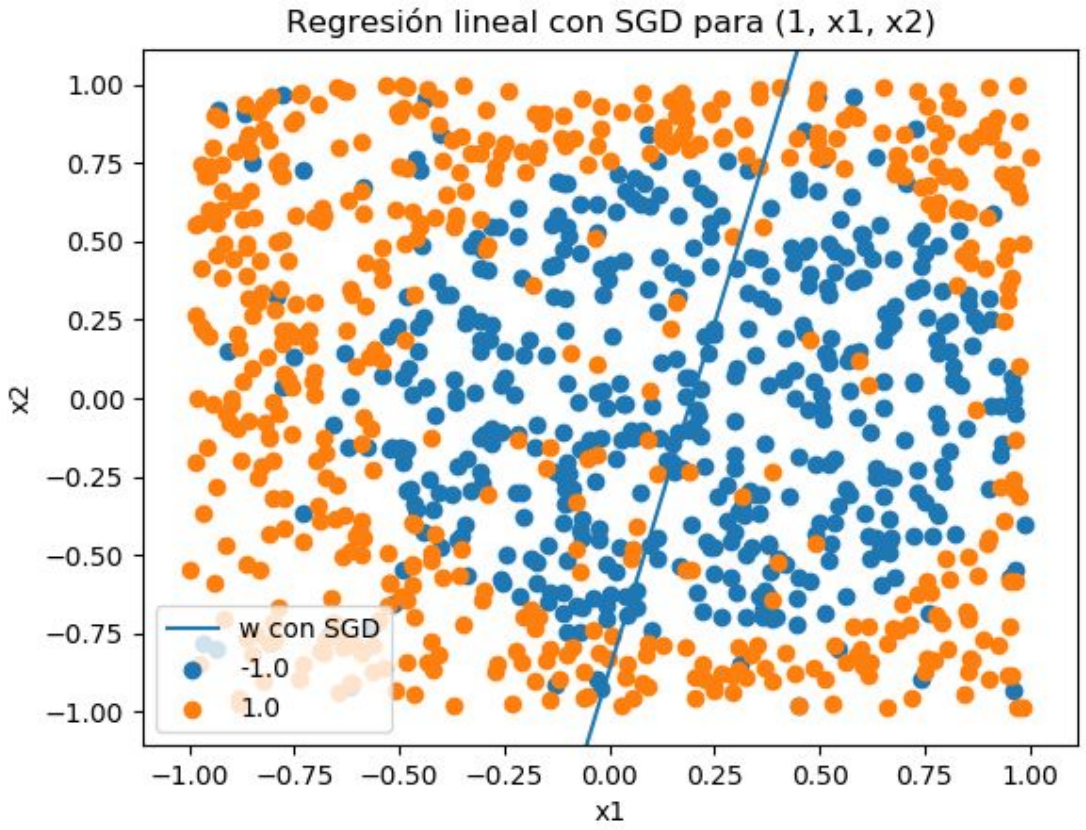


b) Los datos se etiquetan según el resultado que den para la función $f(x1,x2) = sign((x_1-0,2)^2+x_2^2-0,6)$. Además, se proporciona ruido, representado por un cambio en la etiqueta del 10% de los datos. Para ello, se toma una muestra aleatoria sin repeticiones de números entre 0 y 999, y se multiplica por -1 las etiquetas en dichos índices. El mapa etiquetado resulta

así:



c) Usando como vector de características (1, x_1, x_2), se utiliza el ya implementado Gradiente Descendente Estocástico para ver qué tan adecuado resulta para este nuevo problema. Una ejecución de éste, con η =0,01, máximo de 500 y minibatches de 64 elementos iteraciones genera pesos w = (0,01317568, -0,5004598, 0,02674834), y una tasa de error E_{in} = 0,923832347572259. Al graficar la frontera de decisión generada por dicha ejecución resulta



Resulta evidente que la representación no es adecuada para este tipo de problemas, pues el error está por encima de 0,9. En la gráfica resulta aún más evidente, pues en su naturaleza lineal corta el espacio sin poder "envolver" los valores realmente distintos.

d) Ahora se pide repetir el experimento anterior 1000 veces, y así determinar un error E_{in} promedio de todas ellas. Además, en cada iteración se pide generar un nuevo conjunto de datos, idéntico al de entrenamiento, que sirva de conjunto test para medir E_{out}.

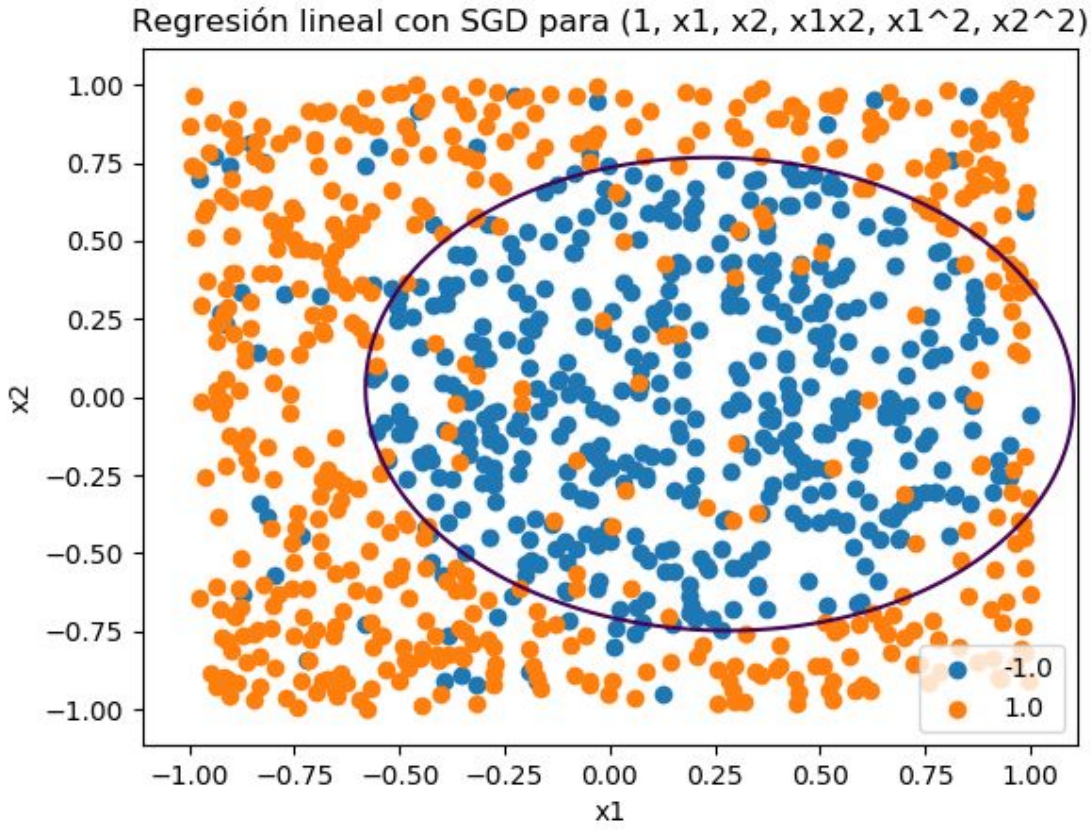
Los valores promedio obtenidos son:

$$E_{in} = 0{,}9270316802716156$$
$$E_{out} = 0{,}9326041935861016$$

e) Evidentemente, no representan ninguna mejora respecto al experimento inicial, probando que la deficiencia se halla en la forma lineal de w y no en una mala elección de parámetros. En la gráfica resulta mucho más claro: los la mayoría de valores etiquetados como -1 forman una forma redondeada rodeada de valores etiquetados con 1, por lo que delimitar una frontera que los separe utilizando una línea recta resulta en una pérdida de información y capacidad de predicción altísima.

- Al repetir el experimento, pero esta vez utilizando el vector de características $\Phi_2(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$, se obtiene que el primer

experimento da lugar a un vector de pesos w = (-0,40296114, -0,4432235, 0,05387439, 0,02663044, 0,66446573, 0,79695353), y una tasa de error E_{in} = 0,6412107144942271. La frontera de decisión queda graficada así



Al realizar 1000 experimentos iguales, y repitiendo la generación de un conjunto test en cada una, resulta

E_{in} = 0,6600471512944202

E_{out} = 0,6658665621094294

- Comparando los resultados obtenidos para los dos vectores de características, queda en evidencia que el segundo es muy superior. El error promedio es cercano a 0,66, que si bien está lejos de ser óptimo, es notablemente mejor que el 0,93 que da si usamos el vector de características de menos dimensiones. En el análisis de las gráficas es donde se representa más notoriamente esto: la frontera de decisión para el vector de características más extenso forma una elipse que envuelve a gran parte de los datos etiquetados como -1 y los separa de aquellos etiquetados como 1 que los rodean, mientras que con el vector de características más pequeño, la única interpretación es mediante una recta que no tiene más opción que cortar este espacio, dando lugar a una frontera de decisión mucho menos precisa. Este experimento prueba el valor de considerar múltiples formas que pudiesen tomar los datos y tenerlo en cuenta al ejecutar algoritmos de aprendizaje. Esta conclusión, si bien resulta ilustrativa, representa la existencia de aún más elementos a tener en cuenta a la hora de realizar

algoritmos como Gradiente Descendente Estocástico que pueden afectar notablemente el rendimiento de los resultados. El apartado 1 deja claro que tanto η como el punto inicial son determinantes, y este experimento va más allá demostrando el impacto del vector de características.