UNIVERSIDAD
DE GRANADA

Aprendizaje Automático. Búsqueda Iterativa de Óptimos y Regresión Lineal

Gradiente Descendente (Estocástico), Pseudoinversa de Moore-Penrose, Características No Lineales

Ricardo Ruiz Fernández de Alba

Escuela Técnica Ingeniería Informática y Matemáticas

DECSAI

Universidad de Granada

3 de abril de 2022

Índice general

1

Ejercicio sobre la búsqueda iterativa de óptimos

1.1 | Descenso de gradiente

El algoritmo de descenso de gradiente es una técnica general para minimizar funciones diferenciables. Es un algoritmo iterativo basado en el cálculo del gradiente de la función, y la actualización de cierto vector de pesos.

Formalmente, sea $w(0) \in \mathbb{R}^d$ un punto inicial, y $E : \mathbb{R}^d \longrightarrow \mathbb{R}$ una función diferenciable. Sea $\eta \in \mathbb{R}^+$. La actualización del peso en la t-ésima iteración viene definida por

$$w(t+1) = w(t) - \eta \nabla E_{in}(w(t))$$

donde al parámetro η se le conoce como tasa de aprendizaje. El algoritmo se basa en seguir la dirección opuesta al vector gradiente pues se maximiza el decrecimiento de la función en cada iteración. La convergencia no está garantizada en general y dependerá de la tasa de aprendizaje escogida así como de las características de la función.

Formalmente, esto se deduce de la expansión de Taylor de primer orden de la función: Yaser S. Abu-Mostafa (2012)

$$\begin{aligned} \Delta E_{in} &= E_{in}(w(0) + \eta\hat{v}) - E_{in}(w(0)) \\ &= \eta \nabla E_{in}(w(0))^T \hat{v} + O(\eta^2) \\ &=\geq -\eta \|\nabla E_{in}(w(0))\| \end{aligned} \tag{1.1}$$

con $\hat{v}$ vector unitario. La igualdad se da si y sólo si

$$\hat{v} = -\frac{\nabla E_{in}(w(0))}{\|\nabla E_{in}(w(0))\|}$$

Una tasa de aprendizaje (demasiado) pequeña es ineficiente lejos del mínimo local. Por otro lado, si es demasiado grande, las oscilaciones pueden afectan a la convergencia. Por ello, optamos por una tasa variable proporcional a la norma del gradiente.

$$\eta_t = \eta \|\nabla E_{in}\|$$

De esta manera, se realizan grandes pasos lejos del mínimo y pasos de menor tamaño cerca del mínimo (donde la norma del gradiente es menor). Además, η_t cancela con el denominador de $\hat{v}$ definido anteriormente lo que se sintetiza redefiniendo $\eta = \eta_t$ y hallando el opuesto del vector gradiente pero sin normalizar.

Como la convergencia no está garantizada en tiempo finito, necesitamos una condición de parada como que la función a minimizar quede por debajo de cierto umbral de error (i.e $E_{in} \leq \epsilon$) o simplemente un número máximo de iteraciones. Combinamos ambas en la implementación ofrecida an el archivo de código:

1.2 | Ejercicio 1

Implementar el algoritmo de gradiente descendente

Analizamos la definición de la función descrita en el código:

```
def gradient_descent(w_ini, lr, grad_fun, fun, epsilon, max_iters,
                     stop_cond,  hist=False):
```

Sobre la cabecera:

- `w_ini`: vector de pesos inicial
- `grad_fun`: gradiente de la función a minimizar
- `fun`: función a minimizar
- `epsilon`: cota de error para parar
- `max_iters`: número máximo de iteraciones para parar
- `stop_cond(it, w, epsilon, max_iters, grad_fun, fun)` : función que indica cuando parar el algoritmo

    `stop_cond_maxIter`: para cuando `it <= max_iters`

`stop_cond_error`: para cuando `fun(w[0], w[1]) \leq epsilon`

- `hist`: True si se desea devolver el histórico para visualización

1.3 | Ejercicio 2.

Considerar la función $E(u,v) = \left(uv \cdot e^{-u^2-v^2}\right)^2$ Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto $(u,v) = (0.5,-0.5)$ y usando una tasa de aprendizaje $\eta = 0,1$.

1.3.1 | Cálculo analítico de $\nabla E(u,v)$

Hallamos la primera derivada parcial:

$$\begin{aligned}\frac{\partial E}{\partial u}(u,v) = 2uve^{-u^2-v^2} \cdot v\left[e^{-u^2-v^2} + u \cdot e^{-u^2-v^2} \cdot 2(-u)\right] = \\ = 2uv^2e^{2(-u^2-v^2)} - 4u^3v^2e^{2(-u^2-v^2)} = -2u(2u^2-1)v^2e^{-2(u^2+v^2)}\end{aligned} \tag{1.2}$$

Y análogamente, $\frac{\partial E}{\partial u}(u,v) = -2v(2v^2-1)u^2e^{-2(u^2+v^2)}$. Luego,

$$\nabla E(u,v) = \begin{pmatrix}\frac{\partial E}{\partial u}(u,v) \\ \frac{\partial E}{\partial v}(u,v)\end{pmatrix} = \begin{pmatrix}-2u(2u^2-1)v^2e^{-2(u^2+v^2)} \\ -2v(2v^2-1)u^2e^{-2(u^2+v^2)}\end{pmatrix} \tag{1.3}$$

1.3.2 | Número de iteraciones con cota de error 10^{-8}

1.3.3 | Coordenadas obtenidas con cota de error 10^{-8}

Ejecutamos el algoritmo anteriormente descrito con los siguiente parámetros:

```
eta = 0.1
error2get = 1e-8
initial_point = np.array([0.5,-0.5])

w, it = gradient_descent(initial_point, eta, gradE, E,
                         error2get, maxIter, stop_cond_error)
```

La condición de parada fijada es que la función de error alcance un valor menor o igual a 10^{-8}, que se indica en `stop_cond_error`

- Numero de iteraciones: 25117

- Coordenadas obtenidas: $(0.0100, -0.0100)$

En la siguiente figura visualizamos el desplazamiento hacia el mínimo:

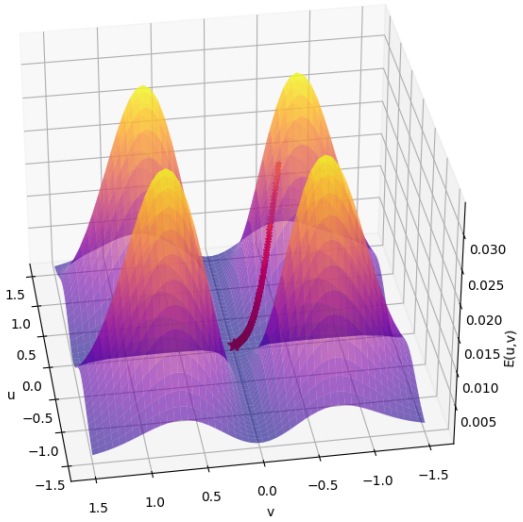

Figura 1.1: Descenso de gradiente sobre E

1.4 | Ejercicio 3

Consideramos la función $f(x,y) = x^2 + 2y^2 + 2\sin(2\pi x)\sin(\pi y)$. **Aplicar gradiente descendente para minimizar** f

1.4.1 | Dependencia de la tasa de aprendizaje

Usar como punto inicial $(x_0 = -1, y_0 = 1)$, tasa de aprendizaje $\eta = 0,01$ 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 $\eta = 0,1$, comentar las diferencias y su dependencia de η.

En primer lugar, calculamos analíticamente el gradiente de f.

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{pmatrix} = \begin{pmatrix} 2x + 4\pi sen(\pi y) cos(2\pi x) \\ 4y + 2\pi sen(\pi x) cos(\pi y) \end{pmatrix} \tag{1.4}$$

En este caso la condición de parada es el máximo de iteraciones, por lo que que ejecutamos el algoritmo con la función de condición de parada `stop_cond_maxIter`

```
eta = 0.01
maxIter = 50
initial_point = np.array([-1, 1])

ws, it = gradient_descent(initial_point, eta, gradf, f, None, maxIter,
                          stop_cond_maxIter, hist=True)
```

Así, para $\eta = 0.01$ y un máximo de 50 iteraciones, las coordinadas finales son $(-1.2178, 0.4134)$ donde f toma el valor -0.0623

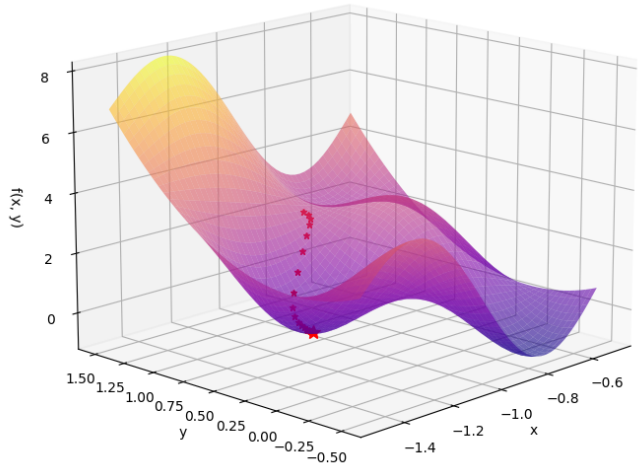

Figura 1.2: Descenso del gradiente sobre f con $\eta = 0.01$

De igual manera para $\eta = 0.1$ las coordenadas finales obtenidas son $(-0.1155, 0.1610)$ y visualizamos el descenso en la siguiente figura:

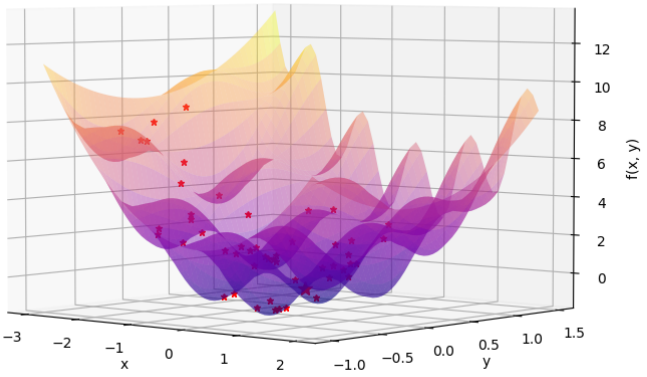

Figura 1.3: Descenso del gradiente sobre f con $\eta = 0.1$

Como se observa en la figura, la tasa de aprendizaje es demasiado alta, provocando oscilaciones que saltan el mínimo. De hecho, una inspección del vector de puntos `ws` revela que en la 43-ésima iteración alcanza el mínimo valor de las 50 iteraciones, pero que luego sigue oscilando y termina en la última iteración con un valor mayor.

La comparación entre las oscilaciones con $\eta = 0.1$ y el descenso adecuado con $\eta = 0.01$ quedan reflejadas en la siguiente figura.

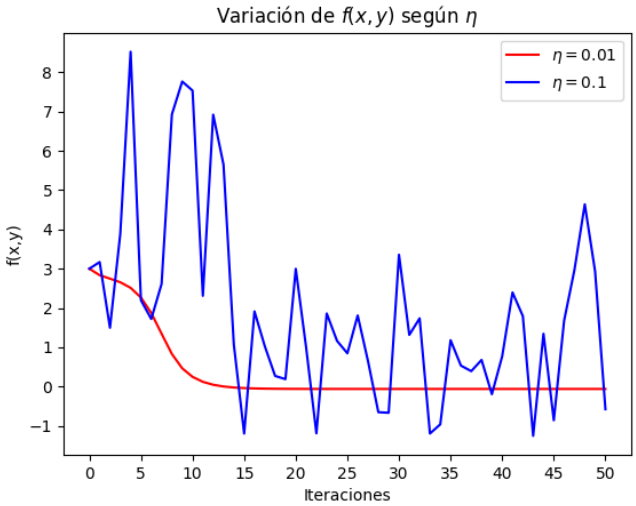

Figura 1.4: Comparación de $f(x, y)$ para $\eta = 0.01$ y $\eta = 0.1$

1.4.2 | Tabla de mínimos según punto inicial y tasa de aprendizaje

Obtener el valor mínimo y los valores de las variables (x, y) en donde se alcanzan cuando el punto de inicio se fija en: $(-0.5, -0.5)$, $(1, 1)$, $(2.1, -2.1)$, $(-3, 3)$, $(-2, 2)$. Generar una tabla con los valores obtenidos, empleando el máximo número de iteraciones (50) y las tasas de aprendizaje del anterior apartado ($\eta = 0.01$ y $\eta = 0.1$). Comentar la dependencia del punto inicial.

Cuadro 1.1: Valor mínimo para $\eta = 0.1$

Punto inicial	Coordenadas mínimo	Valor mínimo f(x,y)
(-0.5, -0.5)	(0.2848,-0.5553)	-1.2250
(1, 1)	(0.3018,-0.4257)	-1.3902
(2.1, -2.1)	(-0.2320,0.2831)	-1.3293
(-3, 3)	(0.1816,-0.3152)	-1.2883
(-2, 2)	(0.2428,-0.3121)	-1.4061

Cuadro 1.2: Valor mínimo para $\eta = 0.01$

Punto inicial	Coordenadas mínimo	Valor mínimo f(x,y)
(-0.5, -0.5)	(-0.7308,-0.4144)	-1.0366
(1, 1)	(0.7308,0.4144)	-1.0366
(2.1, -2.1)	(1.6651,-1.1728)	4.6338
(-3, 3)	(-2.1888,0.5868)	3.6942
(-2, 2)	(-1.6643,1.1713)	4.6337

Es claro a partir de estos datos que el algoritmo de gradiente Descendente tiende hacia un mínimo local que dependerá del punto inicial. En general, en este caso, la tasa de aprendizaje mayor $\eta = 0.1$ proporciona un valor mínimo más regular (en torno a -1.25) que la tasa $\eta = 0.01$.

1.5 | Ejercicio 4

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

A partir del experimento anterior, confirmamos empíricamente que el mínimo local que encuentra el algoritmo de gradiente Descendente depende fuertemente tanto del punto inicial como de la tasa de aprendizaje. No podemos garantizar que este mínimo sea global a no ser que la función sea convexa. Esto último sucede en los casos de la

función de error cuadrático medio de la regresión lineal así como en el error de entropía cruzada de la regresión logística. Esto significa que al minimizar estas medidas de error convexas con este algoritmo, no se estancará en un mínimo local.

2

Regresión Lineal

2.1 | Algoritmo de la Pseudo-inversa

El algoritmo de la Pseudo-inversa o mínimos cuadrados ordinarios consiste en minimizar el error cuadrático entre la estimación $h(x) = w^T x$ y el vector objetivo y

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2 = \frac{1}{N}\|Xw - y\|^2 \tag{2.1}$$

Basta encontrar w que anule $\nabla E_{in}(w)$. Esto se verifica cuando $w_{lin} = X^\dagger y$ Yaser S. Abu-Mostafa (2012). Donde $X^\dagger = (X^T X)^{-1} X^T$ es la **matriz pseudo-inversa** de X

Como el cálculo de la inversa $(X^T X)^{-1}$ puede ser costoso, aplicamos previamente la descomposición en valores singulares a X. Es decir, escribimos $X = UDV^T$ con $U \in \mathbb{R}^{N\times N}$, $V \in \mathbb{R}^{(d+1)\times(d+1)}$ matrices ortogonales y $D \in \mathbb{R}^{N\times(d+1)}$ matriz diagonal rectangular. Entonces $X^T X = VD^T DV^T$ y la pseudo-inversa se calculará como sigue:

$$\begin{aligned} X^\dagger = (X^T X)^{-1} X^T = (VD^T DV^T)^{-1} VD^T U^T = \\ = V(D^T D)^{-1} V^T VD^T U^T = V(D^T D)^{-1} D^T U^T = VD^\dagger U^T \end{aligned} \tag{2.2}$$

2.2 | Gradiente Descendente Estocástico

El algoritmo de gradiente descendente estocástico (SGD) es una versión secuencial del algoritmo de gradiente descendente descrito en el primer capítulo. En esta versión, usamos sólo una parte de la muestra para calcular el gradiente. Para ello, dividimos la muestra aleatoriamente en **mini-batches**. Recorreremos esta secuencia de lotes, actualizando los pesos en cada mini-batch de manera que sólo uno participa en cada actualización.

Formalmente, para cada mini-batch B_i

$$\frac{\partial E_{in}(w)}{\partial w_j} = \frac{2}{M} \sum_{n \in B_i} x_{nj}(h(x_n) - y_n)$$

2.3 | Ejercicio 1

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

Estimar un modelo de regresión lineal, a partir de los datos proporcionados por los vectores de características dados, usando tanto el algoritmo de la pseudo-inversa como el gradiente descendente estocástico (SGD). Las etiquetas serán -1,1, una por 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** E_{in} **y** E_{out} (para E_{out} calcular las predicciones usando los datos del fichero de test).

En el caso de la pseudo-inversa, se ha implementado usando la descomposición en valores singulares anteriormente descrita en `def pseudoinverse(x):` y se ha aplicado al problema de regresión con cierto vector objetivo y en `def regresion_pinv(x, y)`.

En cuanto al algoritmo de gradiente descendente estocástico (SGD), se ha implementado en su versión *mini-batch* de forma general mediante una condición genérica de parada, como se hizo con el no estocástico. Esta se ha particularizado para el caso de un batch de tamaño 32 y una condición de parada por máximo de iteraciones en

```
def sgd_maxIter(x, y, lr, max_iters, epsilon=None,
                batch_size=32, hist=False):
```

También existe otra versión (`sgd_maxIter_error()`) con condición de parada mixta por máximo de iteraciones o cota de error.

Tras cierta experimentación con los parámetros, se ha decidido exponer la comparación entre SGD con un 500 iteraciones, SGD con 20000 iteraciones y el algoritmo de la pseudoinversa. En particular, se ha escogido una tasa de aprendizaje $\eta = 0.01$ por la estabilidad proporcionada.

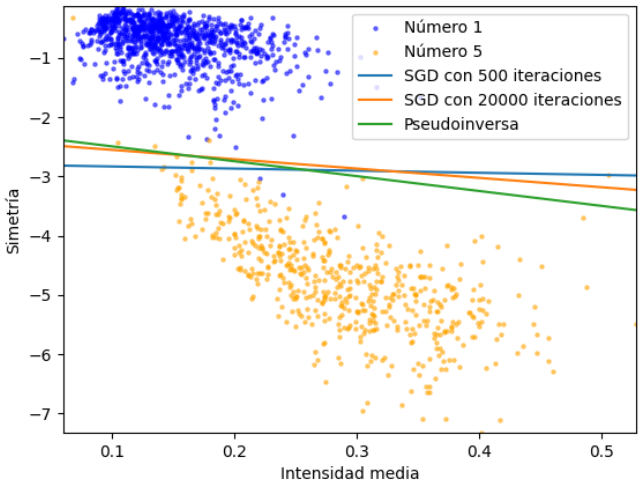

Figura 2.1: Descenso del gradiente sobre f con $\eta = 0.01$

Algoritmo	Iteraciones	E_{in}	E_{out}
SGD (32)	500	0.083469	0.132706
SGD (32)	20000	0.080445	0.135549
Pseudoinversa	—-	0.079187	0.130954

Se observa que E_{in} producido por la pseudoinversa es menor que el del Gradiente Descendente Estocástico. Esto se debe a que el algoritmo de la pseudoinversa se basa en la minimización de este error. Ahora bien, a partir cierto tamaño de datos, este algoritmo se puede volver costoso por las operaciones matriciales involucradas (si bien reducimos esta complejidad mediante la descomposición en valores singulares). Además, el componente aleatorio de SGD permite escapar de mínimos locales al contrario del algoritmo normal tratado en el capítulo 1.

En cuanto a las E_{out} originados por las predicciones en test, vemos que son mayores que los E_{in}. Esto sucede porque el algoritmo no se adapta a los datos de test igual que a los datos con los que ha sido entrenado.

Finalmente, podemos notar a partir de la gráfica junto con experimentaciones con mayores valores de iteraciones que el algoritmo de gradiente descendente estocástico genera una recta de regresión que se acerca a la producida por el algoritmo de la pseudoinversa conforme se aumentamos el número de iteraciones.

2.4 | Ejercicio 2

En este apartado exploramos cómo se transforman los errores E_{in} y E_{out} 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 $[-size, size] \times [-size, size]$. Se debe realizar el siguiente experimento:

2.4.1 | Generar muestra de entrenamiento

Generar una muestra de entrenamiento de $N = 1000$ puntos en el cuadrado $\chi = [-1, 1] \times [-1, 1]$. Pintar el mapa de puntos 2D.

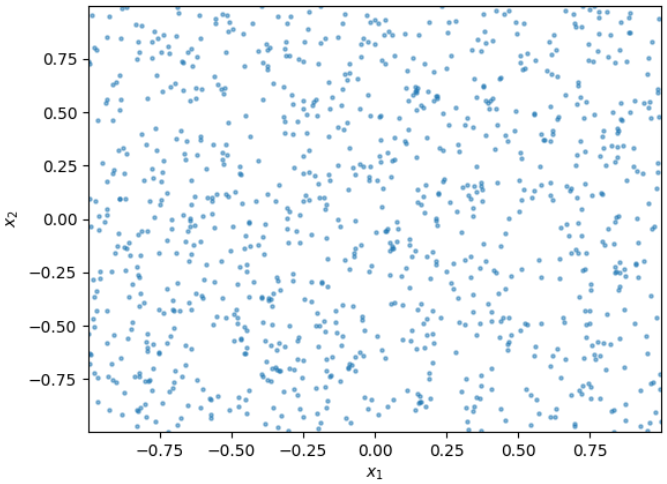

Figura 2.2: Muestra aleatoria uniforme con $N = 1000$ en $\chi = [-1, 1] \times [-1, 1]$

2.4.2 | Etiquetado de la muestra

Consideremos la función $f(x_1, x_2) = sign\left((x_1 - 0,2)^2 + x_2^2 - 0,6\right)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando el signo de un 10 % de las mismas elegido aleatoriamente. **Pintar el mapa de etiquetas obtenido.**

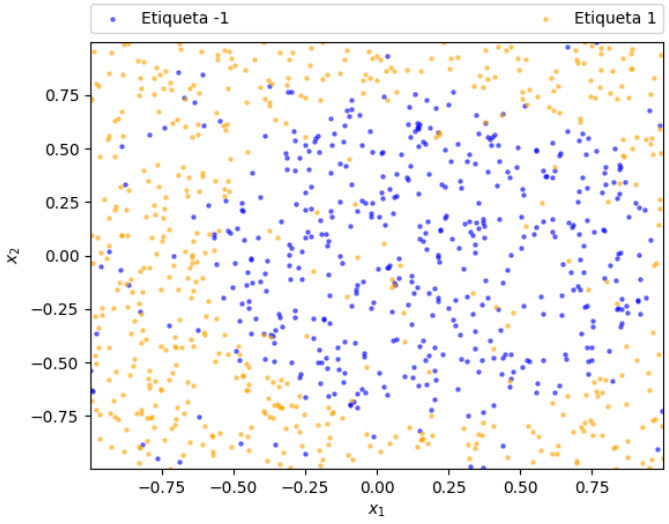

Figura 2.3: Etiquetado de la muestra según f con ruido de 10 %

Una vez generado el vector de etiquetas y aplicando f a la muestra, se ha añadido el ruido con `y[noise] = -y[noise]` siendo `noise` un vector aleatorio uniforme de $0.1 \cdot N$ índices entre 0 y N.

2.4.3 | Modelo de regresión Lineal.

Usando como vector de características $(1, x_1, x_2)$, ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. **Estimar el error de ajuste E_{in} usando SGD.**

Usando SGD con $\eta = 0.01$ y 200 iteraciones, se ha obtenido:

- $E_{in} = 0.9293368608301051$
- $w = [0.02527404, -0.35022946, -0.01136838]$

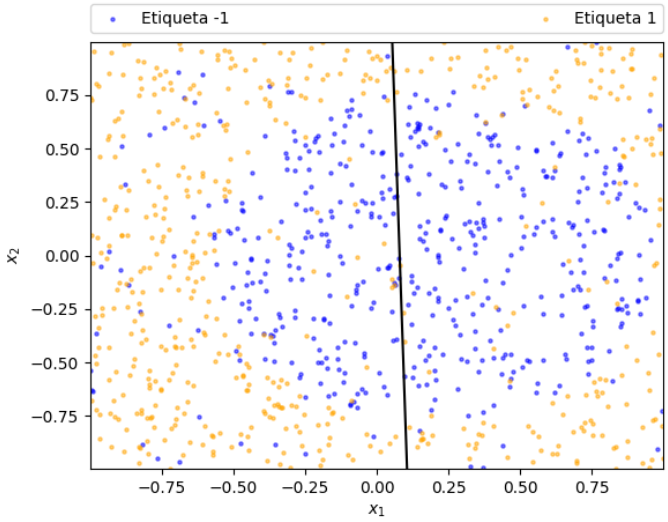

Figura 2.4: Modelo de regresión lineal a la muestra generada

En efecto, el error de ajuste es altísimo debido a que los valores se han generado aleatoriamente con distribución uniforme y no se acercan a ningún modelo lineal.

2.4.4 | Repetir experimento 1000 veces

Ejecutar todo el experimento definido en los apartado anteriores 1000 veces (generamos 1000 muestras diferentes) y

- **Calcular el valor medio de los errores E_{in} de las 1000 muestras.**
- **Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E_{out} en dicha iteración.**
- **Calcular el valor medio de E_{out} en todaslas iteraciones.**

Se ha decidido ejecutar por cada repetición, SGD con 200 iteraciones para obtener un resultado en no más de 10 segundos. Los promedios de E_{in} y E_{out} son respectivamente:

- E_{in} promedio: 0.9325394003097736
- E_{out} promedio: 0.9357118616492284

Valore qué tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}.

En efecto, los erores de ajuste obtenidos en promedio para 1000 iteraciones del experimento anterior no han producido una mejora significativa. Es claro de las figuras obtenidas anteriormente, que un modelo lineal no separa etiquetas acumuladas en circunferencia.

2.5 | Repetición del experimento con características no lineales

Repetir el mismo experimento anterior pero usando características no lineales. Ahora usaremos el siguiente vector de características:

$$\phi_2(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

Ajustar el nuevo modelo de regresión lineal y calcular el nuevo vector de pesos $\hat{w}$. Calcular los errores promedio de E_{in} y E_{out}.

Para la implementación, se ha decidido modificar la función `generar_muestra_2D_uniforme(N, no_lineal)` donde el parámetro `no_lineal` es un booleano que indica si agregar las nuevas columnas.

Los errores promedio obtenidos en este caso, han sido los siguientes:

- E_{in} promedio: 0.7736520275418629
- E_{out} promedio: 0.7785544126191773

Y la figura asociada a la transformación es la siguiente:

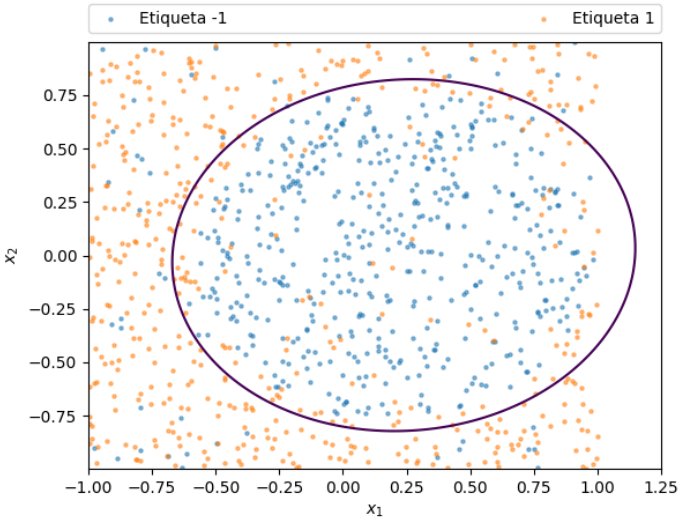

Figura 2.5: Regresión con características no lineales $\Phi(x)$

2.6 | Comparación entre los experimentos

A la vista de los resultados de los errores promedios E_{in} y E_{out} obtenidos en los dos experimentos, **¿qué modelo considera que es el más adecuado? Justifique la respuesta.**

La figura generada encaja con los resultados. El modelo con características no lineales aplica una transformación donde la regresión lineal calculada con el algoritmo de gradiente descendente estocástico con 200 iteraciones y $\eta = 0.01$ arroja unos errores de ajustes menores ($E_{in} \approx 0.7738$, $E_{out} \approx= 0.778$), que sin la transformación ($E_{in} \approx 0.9320$, $E_{out} \approx 0.9357$).

Bibliografía

Hsuan-Tien Lin Yaser S. Abu-Mostafa, Malik Magdon-Ismail. *Learing From Data. A Short Course*. AMLbook, 2012. URL `AMLbook.com`.