

Prácticas de Aprendizaje Automático

Trabajo 1: Búsqueda Iterativa de Óptimos y Regresión Lineal

Pablo Mesejo y Jesús Giráldez

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Recordatorio normas (1). Informe.

- Presentad un **informe escrito** con las valoraciones y **decisiones** adoptadas en cada apartado
 - No es solo hacer algo → hay que argumentar el por qué
- Incluid en el informe
 - los **gráficos** generados.
 - **valoración/discusión de los resultados obtenidos.**
- Recordad que tenéis dos opciones:
 - a) .zip = Código (.py) + Informe (.pdf)
 - b) .ipynb = Código, informe y resultados integrados en un notebook de Google Colab
- Si no hay informe/análisis/discusión → se considera que el trabajo no se ha presentado

Recordatorio normas (2). Código.

- **Estructurad adecuadamente el código.**
 - Los distintos ejercicios van en apartados comentados dentro del fichero
- **Todos los resultados numéricos o gráficas serán mostrados por pantalla,** parando la ejecución después de cada apartado.
 - No escribir nada en el disco
- El path que se use en la lectura de cualquier fichero auxiliar de datos debe ser siempre **"datos/nombre_fichero"**.
 - Crear directorio llamado "datos" dentro del directorio donde se desarrolla y se ejecuta la práctica
 - Objetivo: que no empleéis nombres rebuscados o rutas complicadas

Recordatorio normas (3). Código.

- El código **debe ejecutarse de principio a fin sin errores.**
- No es válido usar opciones en las entradas.
 - **Fijar al comienzo los valores para los parámetros** que se consideren óptimos.
- El código debe estar obligatoriamente **comentado** explicando lo que realizan los distintos apartados
 - **Id comentando el código** que hagáis: sirve para que entendáis mejor lo que habéis hecho, y facilita mi trabajo a la hora de corregir los ejercicios.
- Entregad **solo el código fuente, nunca los datos.**

Recordatorio normas (y 4)

.zip = Código (.py) + Informe (.pdf)

o

.ipynb = Código, informe y resultados integrados en un Colab notebook

Subid la entrega a PRADO, a la actividad creada para ello.

Fecha de entrega: 3 de Abril

Template

- Podéis partir, si queréis, del template inicial que hemos preparado y que tenéis en PRADO (template_trabajo1.py)
- Es un template para ser ejecutado en Spyder pero, si queréis emplear Colab, basta con copiar su contenido en una celda de código.

```
# -*- coding: utf-8 -*-
"""
TRABAJO 1.
Nombre Estudiante:
"""

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1)

print('EJERCICIO SOBRE LA BUSQUEDA ITERATIVA DE OPTIMOS\n')
print('Ejercicio 1\n')

def E(u,v):
    return #function

#Derivada parcial de E con respecto a u
def dEu(u,v):
    return #Derivada parcial de E con respecto a u

#Derivada parcial de E con respecto a v
def dEv(u,v):
    return #Derivada parcial de E con respecto a v

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

def gradient_descent(w_ini, lr, grad_fun, fun, epsilon, max_iters):
    #
    # gradiente descendente
    #
    return w, iterations

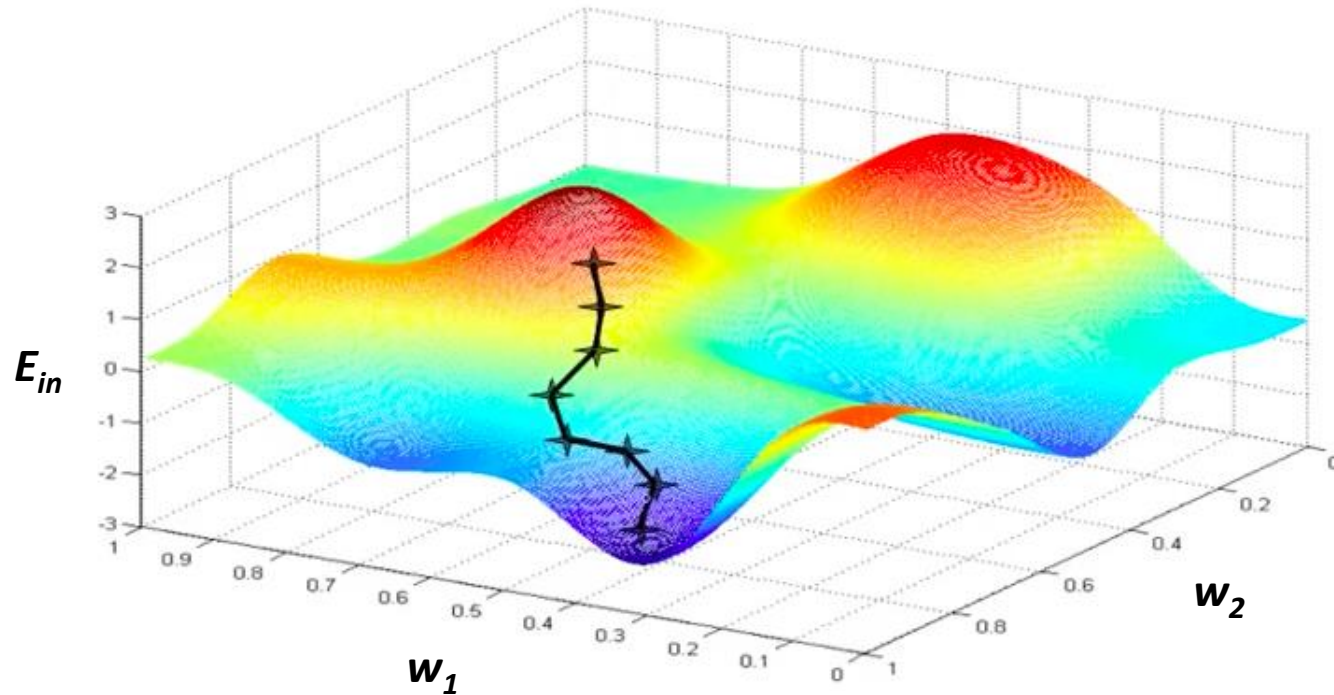
eta = 0.1
maxIter = 10000000000
error2get = 1e-8
initial_point = np.array([0.5,-0.5])
```

1. Búsqueda iterativa de óptimos

- Implementar el algoritmo de gradiente descendente
 - Algoritmo para minimizar funciones
 - Requiere una función derivable a minimizar
 - Es un algoritmo local: empieza en un punto y va descendiendo por la pendiente más pronunciada
 - El gradiente apunta en la dirección de mayor crecimiento de la función, y su magnitud es la pendiente en dicha dirección
 - Como estamos minimizando, se emplea el signo contrario al gradiente

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j}$$

1. Búsqueda iterativa de óptimos



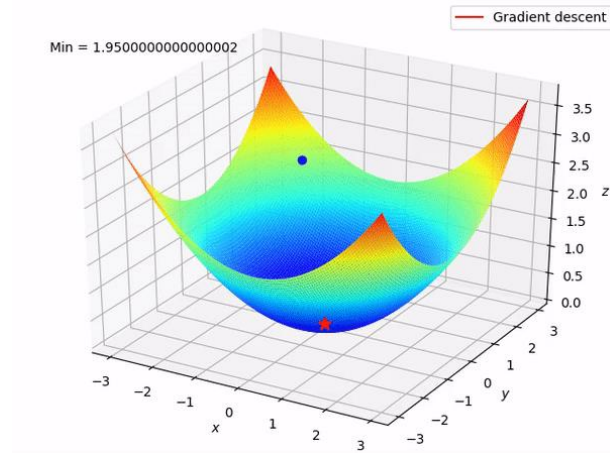
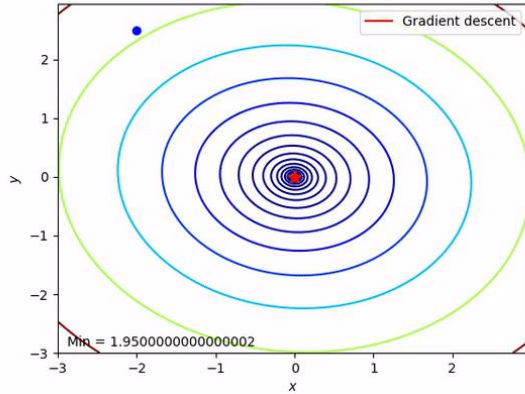
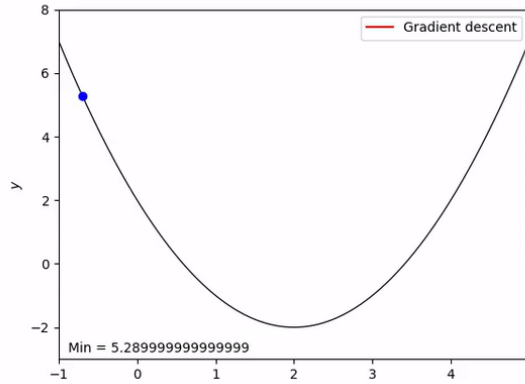
Ejemplo:
Función con dos
pesos/parámetros

Se busca minimizar el
error E_{in}

Partiendo de un punto
inicial

Se desciende por la
dirección de mayor
pendiente

1. Búsqueda iterativa de óptimos



Animaciones extraídas de https://jed-ai.github.io/py1_gd_animation/

1. Búsqueda iterativa de óptimos

- We want to choose \mathbf{w} so as to minimize $E_{in}(\mathbf{w})$
- Gradient Descent (GD):
 - Gradient descent is a general iterative optimization technique that reach a local optimum following the direction of the gradient vector on each point.

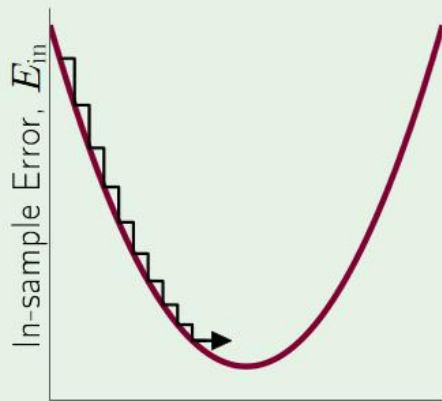
It starts on some initial value \mathbf{w} and repeatedly perform the update ,

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j} \quad (\text{GENERAL EQUATION})$$

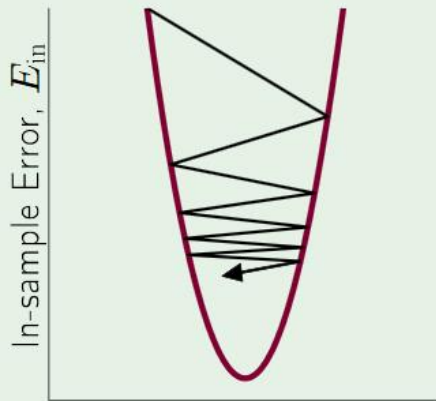
(This update is simultaneously performed for all values of $j = 0, \dots, n$). Here, η is called the learning rate.

1. Búsqueda iterativa de óptimos

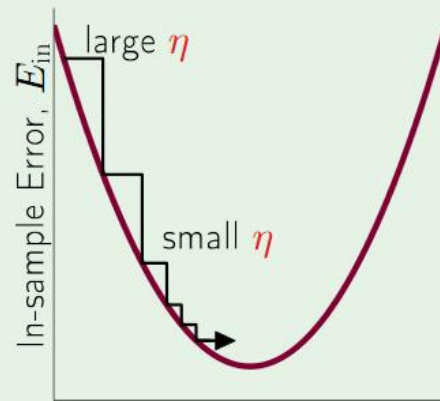
How η affects the algorithm:



η too small



η too large



variable η – just right

η should increase with the slope

1. Búsqueda iterativa de óptimos

- Implementar el algoritmo de gradiente descendente

1. Una función que implemente el gradiente descendente

```
def gradient_descent(?):
```

2. ¿Cuál es el cuerpo de la función?

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j}$$

3. ¿Qué argumentos se le pasan a la función?

- En el template os proponemos una serie de argumentos que podéis emplear, si así lo consideráis.

1. Búsqueda iterativa de óptimos

- Recomendaciones
 - Imprimid el valor de la función en cada punto del descenso de gradiente → **verificad que los valores van disminuyendo**
 - Si tenéis problemas con los resultados proporcionados por el gradiente descendente, **revisad las derivadas** (probablemente no estén bien calculadas)

1. Búsqueda iterativa de óptimos

- Limitaciones del gradiente descendente
 - Necesidad de función derivable
 - Importancia del punto inicial (es una búsqueda local)
 - Un mínimo local de una función convexa es un mínimo global
 - Importancia del learning rate
 - Demasiado grande → podríamos no converger
 - Demasiado pequeño → llevaría demasiado tiempo

1. Más ideas y consejos (1)

- Con respecto al encaje del Ejercicio 1 en Aprendizaje Automático:
 - En este primer ejercicio, no hay datos de entrenamiento, ni debéis calcular el error cuadrático medio.
 - Son solo dos funciones E y f , que tenéis que minimizar.
 - Recordad que gradiente descendente es una técnica de optimización, de modo que se puede emplear en cualquier contexto (siempre que la función sea diferenciable).

1. Más ideas y consejos (2)

- Con respecto a la implementación del gradiente descendente:
 - Tenéis que calcular, a mano, las derivadas parciales con respecto a u y v , en el caso de E , y con respecto a x e y con respecto a f .
 - En concreto, lo que tenéis que entender es que el gradiente descendente se implementa así de fácil: $\mathbf{w} = \mathbf{w} - \mathbf{lr} * \text{grad}E(\mathbf{w})$

- Y eso, es lo mismo que decir:

$$\mathbf{w}[0] = \mathbf{w_old}[0] - \mathbf{lr} * E_u(\mathbf{w_old})$$

$$\mathbf{w}[1] = \mathbf{w_old}[1] - \mathbf{lr} * E_v(\mathbf{w_old})$$

En donde $\mathbf{w}[0]$ sería u , y $\mathbf{w}[1]$ sería v . Cada peso o cada variable se actualiza con el valor de su derivada parcial.

Nota: E_u y E_v representan las derivadas parciales de la función E con respecto a u y v , respectivamente.

1. Más ideas y consejos (y 3)

- Sobre normalizar el gradiente:

- Es lo que haríamos si implementamos



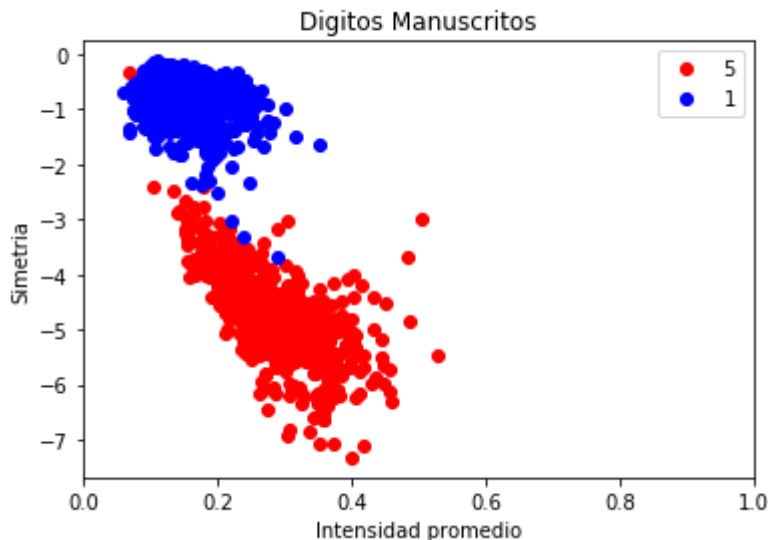
```
w = w - lr * (grad_fun(w) / np.linalg.norm(grad_fun(w)))
```

- De este modo, en la optimización se emplea solo la dirección (al utilizar un vector unitario).
 - Tardamos mucho más en alcanzar el óptimo, dado que la magnitud del gradiente no modula la longitud de los saltitos, y tenemos que dar muchos más saltitos/pasos.
- Al no normalizar el gradiente, el avance ya es adaptativo en cierto sentido. Véase, por ejemplo, <https://youtu.be/qSTHZvN8hzs?t=3810>

2. Ejercicio sobre Regresión Lineal

- En el template tenéis una función para leer los datos: `def readData(file_x, file_y)`
- La idea es usar regresión lineal para clasificación de dígitos

$$y = w_0 + w_1x_1 + w_2x_2$$



2. Ejercicio sobre Regresión Lineal

- Al final, todo consiste en estimar los \mathbf{w} 's
 - Gradient Descent
 - Stochastic Gradient Descent
 - Pseudoinversa (*one-step learning*)
 - BONUS: Método de Newton

Gradient Descent vs Stochastic Gradient Descent

Gradient Descent

It starts on some initial value \mathbf{w} and repeatedly perform the update ,

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j} \quad (\text{GENERAL EQUATION})$$

(This update is simultaneously performed for all values of $j = 0, \dots, n$). Here, η is called the learning rate.

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n - y_n)^2 = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{w}^T \mathbf{x}_n - y_n) = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

Each point (\mathbf{x}_n, y_n) contributes to the update by an amount proportional to its prediction error

In this case all points are used to compute the gradient: **BATCH GRADIENT DESCENT**

Gradient Descent vs Stochastic Gradient Descent

Stochastic Gradient Descent

- An alternative is to use a **stochastic estimation** using only a part of the sample to compute the gradient, $M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$


- Higher variability in the gradient estimation (less examples in the average)
 - Very fast of computing
 - In non-convex funtions empirical evidence of getting good local minimum
- Although an only item could be used on each iteration, a minibatch of items is the accepted rule (size: 32-128)

Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent

- Given the data set $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$
 - Fix $\mathbf{w}=0, \eta = \eta_0$
 - Iterate
For $j=0, \dots, K$:
$$w_j := w_j - \eta \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n) \text{ (all sample participate)}$$
 - Until $E_{in}(\mathbf{w}) < \text{epsilon}$

Stochastic Gradient Descent

- Fix $\mathbf{w}=0, \eta = \eta_0$
- Iterate:
- Shuffle and Split the sample into a sequence of mini-batches**  los minibatches deben ser disjuntos
- Iterate on mini-batches
For $j=0, \dots, K$:
$$w_j := w_j - \eta \sum_{n \in \text{Minibatch}} x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n) \text{ (only a mini-batch participate)}$$
- Until $E_{in}(\mathbf{w}) < \text{epsilon}$

Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent

Vs

Stochastic Gradient Descent

all points are used to compute the gradient

$$\frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

$M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent

Vs

Stochastic Gradient Descent

all points are used to compute the gradient

$$\frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

¿Qué es el x_{nj} ? Es el componente j del patrón de entrada n . En nuestro caso, j va de 0 a 2.

$M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

¿Qué es $h(x_n)$? En nuestro caso, es la salida de regresión lineal ($y = w_0 + w_1 x_1 + w_2 x_2$). Es como si tenemos: $y(x_n) = h(x_n) = w_0 x_{n0} + w_1 x_{n1} + w_2 x_{n2}$

2. Ejercicio sobre Regresión Lineal: Pseudoinversa

A Linear Regression Algorithm

$$E_{in}(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{in}(\mathbf{w}) = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = \mathbf{X}^\dagger \mathbf{y} \quad \text{where} \quad \mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$$

\mathbf{X}^\dagger is the 'pseudo-inverse' of \mathbf{X}

- 1: Construct the matrix \mathbf{X} and the vector \mathbf{y} from the data set $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1^T- \\ -\mathbf{x}_2^T- \\ \vdots \\ -\mathbf{x}_N^T- \end{bmatrix}}_{\text{input data matrix}}, \quad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

- 2: Compute the pseudo-inverse $\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$.
- 3: Return $\mathbf{w} = \mathbf{X}^\dagger \mathbf{y}$.



Can we always compute $(\mathbf{X}^T \mathbf{X})^{-1}$?

- Let consider the Singular Value Decomposition (SVD) : $\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}^T$
- $\mathbf{X}^T \mathbf{X} = \mathbf{V} \mathbf{D} \mathbf{D}^T \mathbf{V}^T$

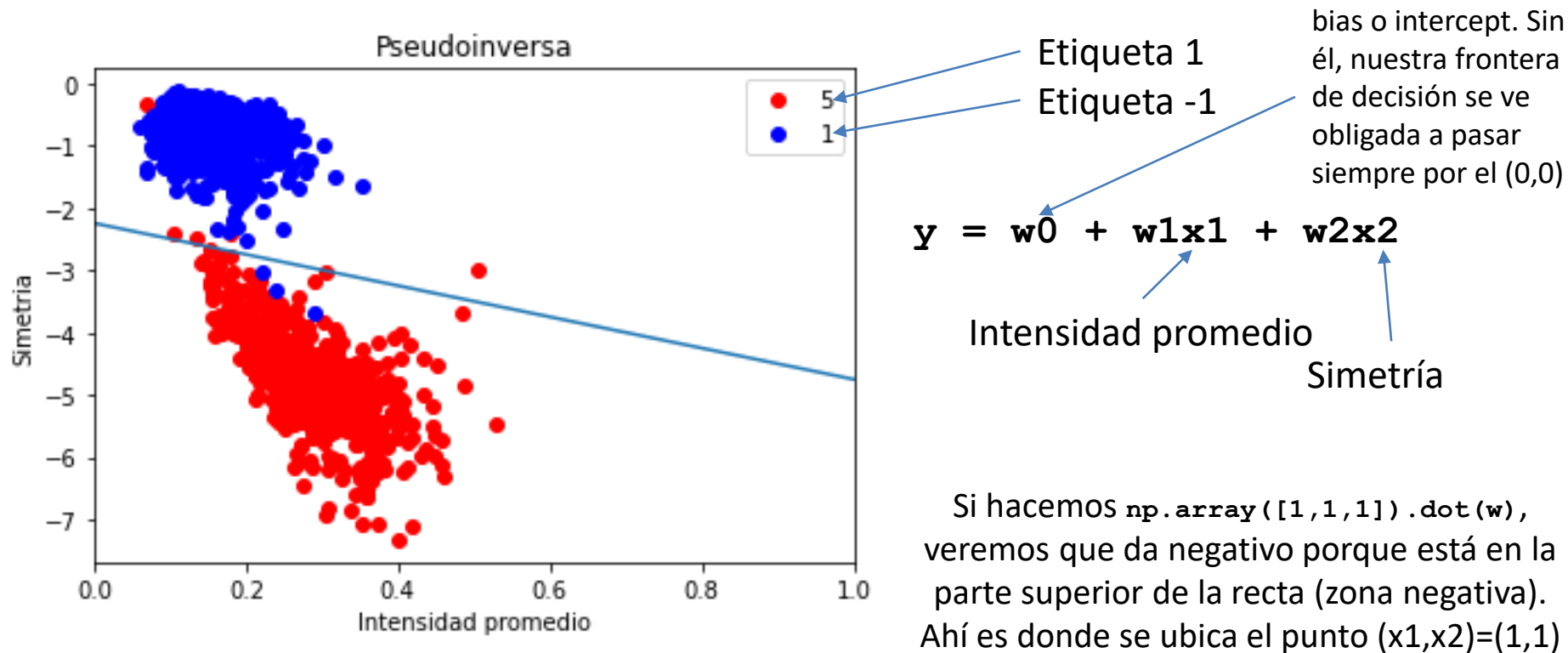
Nota: es improbable que \mathbf{X} sea cuadrada (ejemplos x features) \rightarrow no invertible

2. Ejercicio sobre Regresión Lineal

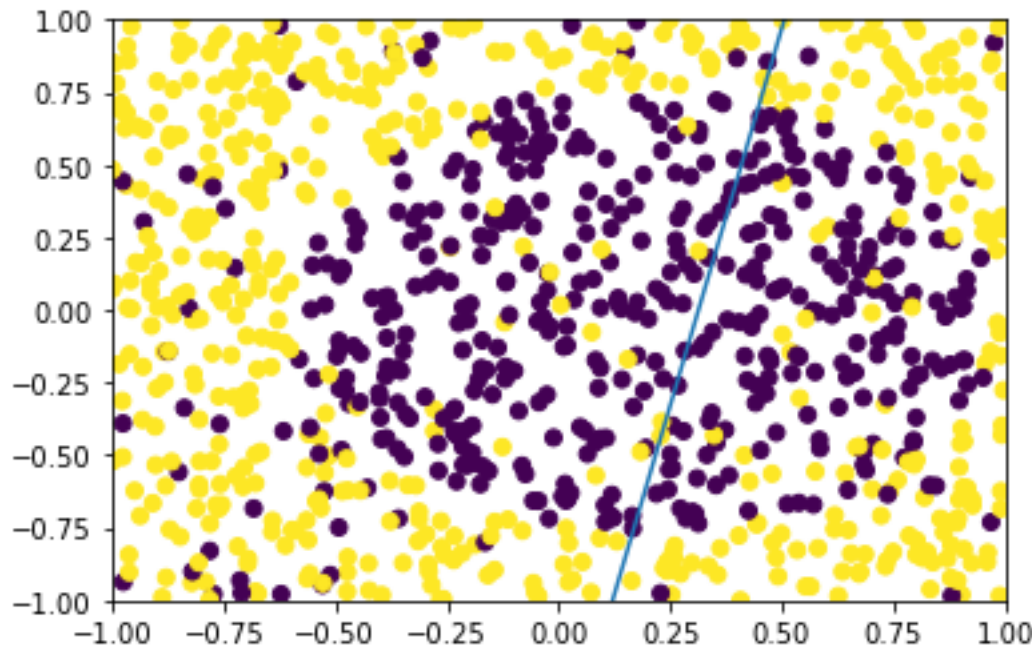
	Descenso de Gradiente	Pseudo-inversa (o ecuación normal)
Pros	<ul style="list-style-type: none">• Funciona bien aunque nuestros ejemplos de entrada tengan muchas <i>features</i> (columnas).¹• Es un método más general. Es decir, la existencia de una aproximación analítica para encontrar el óptimo, sirve para regresión lineal. Por ejemplo, no existen ecuaciones normales para regresión logística.	<ul style="list-style-type: none">• No necesita <i>learning rate</i> (η).• No necesita iterar \rightarrow resuelve el problema analíticamente.
Contras	<ul style="list-style-type: none">• Tenemos que escoger un <i>learning rate</i> (η).• Puede necesitar muchas iteraciones.	<ul style="list-style-type: none">• Necesita calcular $(X^T X)^{-1}$; lo cual puede ser costoso si la dimensionalidad de $(X^T X)$ es alta (es decir, si nuestros ejemplos de entrada tienen muchas <i>features</i>; en el caso de los dígitos solamente sería 3×3).• Son aplicables al caso de regresión lineal, pero no es extrapolable a otros modelos.

¹ Consejo: si el número de características es 1.000 o 10.000 podéis usar la pseudoinversa, pero si es más de eso yo usaría descenso de gradiente.

2. Ejercicio sobre Regresión Lineal



2. Ejercicio sobre Regresión Lineal

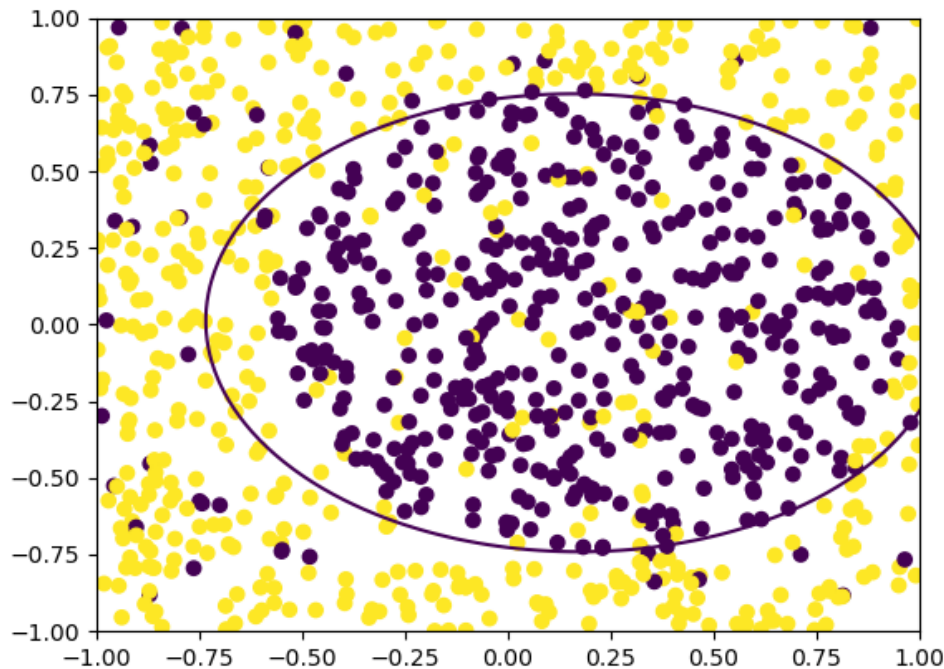


Referencias interesantes:

<https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182>

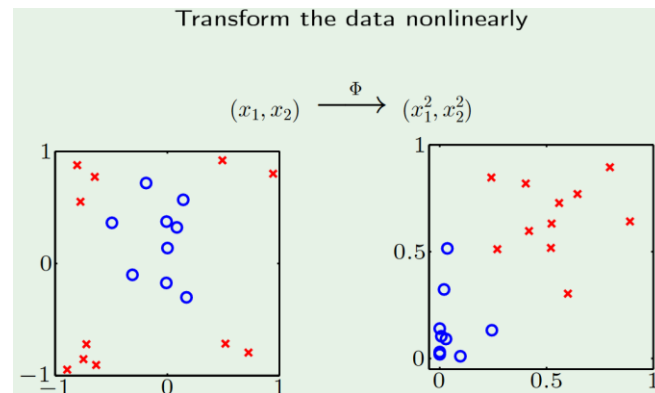
<http://work.caltech.edu/slides/slides03.pdf> (slides 19-23)

2. Ejercicio sobre Regresión Lineal



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

Usamos características no lineales para comprobar cómo, transformando los datos, podemos resolver nuestro problema empleando modelos lineales.



Referencias interesantes:

<https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182>

<http://work.caltech.edu/slides/slides03.pdf> (slide 25)

2. Más ideas y consejos (1)

- Con respecto al número de iteraciones:
 - es un hiperparámetro que podéis escoger a través de un proceso de evaluación experimental.
 - No obstante, en este caso, 200 iteraciones deberían ser suficientes para obtener buenos resultados.

2. Más ideas y consejos (2)

- Con respecto al impacto de los Ws iniciales:
 - los Ws iniciales tienen un impacto en el resultado final.
 - En este caso, el efecto no es dramático en absoluto.
 - Os invito a que probéis distintos valores iniciales, aunque $(0,0,0)$ es un valor razonable, y es el que se indica de hecho en el pseudocódigo de la teoría.

2. Más ideas y consejos (3)

- Con respecto a qué resultados mostrar:
 - regresión lineal emplea el error cuadrático medio como función de pérdida a ser minimizada por el gradiente descendente.
 - No obstante, a la hora de evaluar y mostrar vuestros resultados, mi consejo es que
 - visualicéis el modelo lineal entrenado (scatter plots con los datos de entrenamiento y test relativos a los dígitos 5 y 1, junto con la recta resultante del entrenamiento),
 - el valor de E_{in} y E_{out} ,
 - y el error de clasificación (es decir, el porcentaje de ejemplos de entrenamiento y test mal clasificados por vuestro modelo).

2. Más ideas y consejos (4)

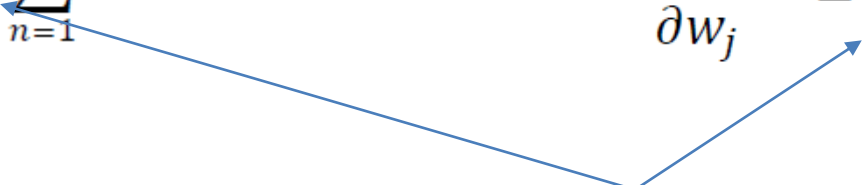
- Evolución de los errores:
 - gradiente descendiente se emplea para minimizar funciones de modo iterativo.
 - En el ejercicio 1, en cada iteración el valor de la función debería ser menor.
 - En el segundo ejercicio, al usar SGD no tiene por qué ser así.
 - Pensad que, al usar minibatches, aunque la tendencia global también tiene que ser ir mejorando, puede que no vayamos a mejorar siempre en cada iteración (dado que puede haber unos minibatches más "difíciles" que otros).
 - En cambio, si usamos todos los datos de entrenamiento (1561), sí deberíamos ir descendiendo progresivamente en el error.

2. Más ideas y consejos (5)

- Posible Overflow:

$$\frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

$M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$


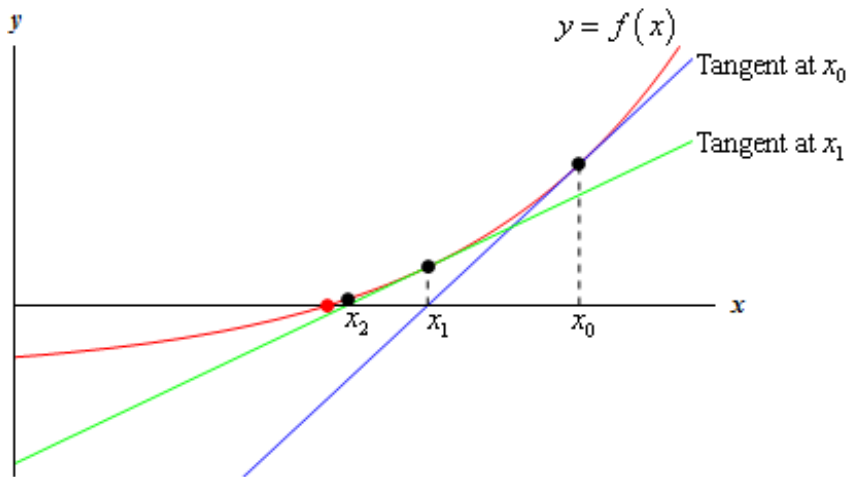
Si nos olvidamos de dividir por N o M nos puede dar un error de Overflow, es decir, **los Ws pueden llegar a ser demasiado grandes.**

2. Más ideas y consejos (y 6)

- ¿Las etiquetas siempre tienen que ser -1 y 1?
 - No, pueden ser las que queráis. Pero debéis ser conscientes de cómo esto afecta a la frontera/borde de decisión.
 - Si escogemos las etiquetas $\{-1,1\}$, la frontera de decisión la tendríamos en $0 = \mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2$
 - Si escogemos las etiquetas $\{1,5\}$, la frontera de decisión la tendríamos en $((1+5)/2) = 3 = \mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2$
 - Se podría aplicar SGD o la pseudoinversa para optimizar la función. Pero, luego, de cara a calcular los pesos adecuados, deberíamos hacer $\{(\mathbf{w}_0 - 3), \mathbf{w}_1, \mathbf{w}_2\}$

BONUS: Método de Newton

El método de Newton, o Newton-Raphson, es un método para encontrar sucesivamente mejores aproximaciones a las raíces, los ceros, de una función real.



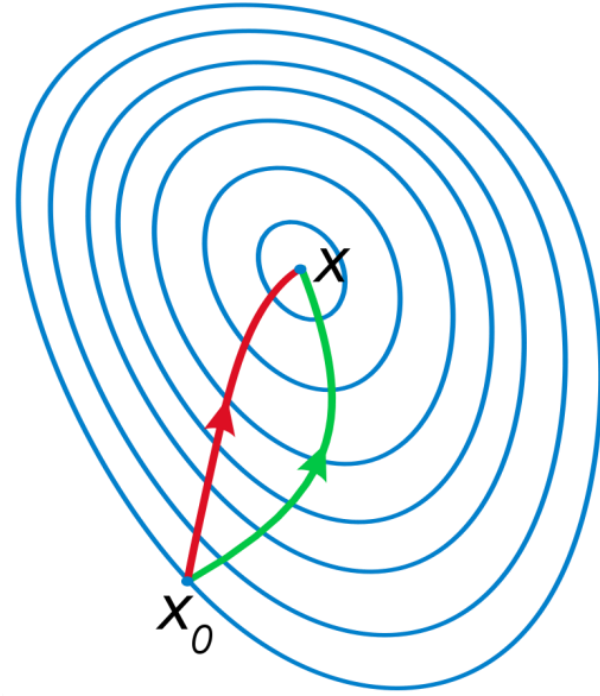
Ajusta iterativamente rectas tangentes en los puntos en donde las tangentes intersecan el cero. En el ejemplo de la izquierda vemos que la tangente en x_0 corta el eje de las Xs en x_1 ; calculamos la tangente a la curva en ese punto; y así sucesivamente.

Cuando se usa la primera derivada, estamos haciendo una aproximación lineal; mientras que con la segunda derivada la aproximación es cuadrática (utilizando información de la curvatura).

BONUS: Método de Newton

Comparativa entre **gradiente descendente**, en verde, y el **método de Newton**, en rojo, a la hora de minimizar una función.

El método de Newton usa información de la curvatura, a través de la segunda derivada o Hessiana, **para tomar un camino más directo al óptimo**.



BONUS: Método de Newton

A new update rule for \mathbf{w} based on the **second order derivatives** (Hessian)

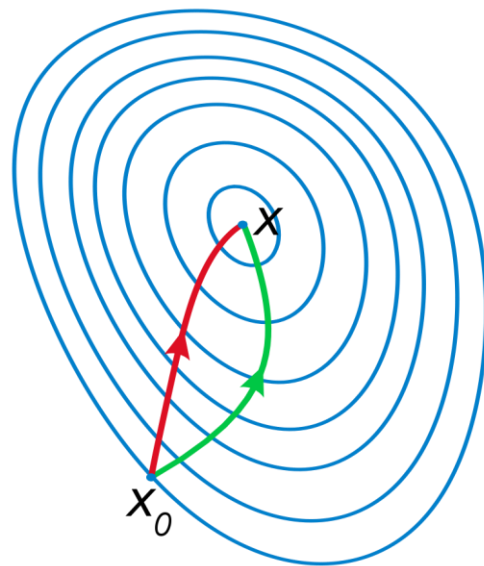
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

$$f''(x) = \nabla^2 f(x) = H_f(x) \in \mathbb{R}^{d \times d}$$

$$x_{k+1} = x_k - \gamma [f''(x_k)]^{-1} f'(x_k).$$

$$\Delta \mathbf{w} = -\mathbf{H}^{-1} \nabla E_{\text{in}}(\mathbf{w}_0)$$

Se le puede poner un learning rate para ayudar a la convergencia. Es lo que se llama “**relaxed Newton-Raphson method**”, y es la versión del método de Newton que podéis implementar en la práctica.



BONUS: Método de Newton

Consejos e ideas:

- 1) **El descenso de gradiente suele necesitar más iteraciones** que el método de Newton, pero cada iteración no es muy costosa. En el caso del método de Newton, **las iteraciones son más costosas porque requieren el cálculo e inversión de la Hessiana**.
- 2) Tened en cuenta que **si la derivada segunda es muy pequeña**, al estar en el denominador, **la iteración os llevaría a un lugar muy lejano del mínimo buscado**.
- 3) De cara a entender mejor lo que pasa cuando ejecutéis el algoritmo:
 - a. Os recomiendo **dibujar la función** en 3D para visualizar la curvatura.
 - b. Si la **Hessiana de la función no es positiva definida**, puede provocar que Newton no vaya en la dirección correcta.

Enlaces recomendados

- Materiales de Andrew Ng sobre *linear regression* y *gradient descent* (lectures 2 and 4):
https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN
- Materiales de Yaser Abu-Mostafa
(<http://work.caltech.edu/lectures.html>):
 - The linear model I: <https://www.youtube.com/watch?v=FlbVs5GbBIQ>
 - The linear model II: <https://www.youtube.com/watch?v=qSTHZvN8hzs>

Prácticas de Aprendizaje Automático

Trabajo 1: Búsqueda Iterativa de Óptimos y Regresión Lineal

Pablo Mesejo y Jesús Giráldez

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA

