

Aprendizaje Automático

Clase 7:

Algoritmos de Regresión
Descenso por el Gradiente
Regularización

Aprendizaje Automático - 2do bimestre 2025

Horario y lugar: 17:00 a 21:00. **Aula 1401**(Pabellón 0+inf)

Campus virtual: <https://campus.exactas.uba.ar/course/view.php?id=192> (mantenemos el de AA1)

Importante: Tienen que estar **matriculados** en la materia. La comunicación será a través del **campus virtual**.

- **2 puntos** de optativa para **Licenciatura en Computación (plan 93)**.
- **64 créditos** para **Licenciatura en Computación (plan 2023)**.
- **64 horas** para **Licenciatura en Ciencia de Datos**.
- **Puntos a confirmar** para el **Doctorado en Computación**.



Noé Hsueh (Ay2)



Cecilia Bolaños (Ay1)



Gastón Bujía (JTP)



Luciano del Corro (Prof)



Pablo Brusco (Prof)

Dinámica de la materia

- **Clases teóricas.** Subiremos las diapos instantes antes de la clase.
- **Guías prácticas:** Cada tema tendrá asociada una guía práctica que subiremos al campus al finalizar la teórica del tema.
- **Clases prácticas:** Durante cada clase práctica trabajaremos con los ejercicios de las guías. Se recomienda fuertemente ir al día con las guías.
- **Parciales:** Los armaremos suponiendo que hicieron las guías.
- **Notebooks:** Serán publicados junto con los cuestionarios asociados.
- **Lecturas obligatorias semanales:** Aprox 1 cap. de un libro por semana.
- **Cuestionarios obligatorios:** Semanalmente habrá cuestionarios en formato de google forms sobre la clase, los notebooks y las lecturas obligatorias. Por defecto, tendrán poco más de 1 semana para resolverlos. Ej, si se presentan el Martes luego de la clase, tendrán tiempo hasta el miércoles de la siguiente a noche.

Régimen de Aprobación

- Para aprobar la cursada deben aprobar el **parcial**, el **TP**, leer la **bibliografía obligatoria** que se presenta al final de cada clase y **completar los cuestionarios semanales obligatorios**.
- Se puede **promocionar** obteniendo nota 7 o superior, aprobando el TP, participando de la competencia del TP y habiendo aprobado 5 de los 6 cuestionarios.
- Para quienes **recuperen** un parcial (habiéndolo aprobado o no), la nota que cuenta es la del recuperatorio.
- Quienes obtengan nota inferior a 7 el parcial deben dar el **final**.

Fechas AA2

TP-AA2:

- Presentación A confirmar
- Envío slides (entrega): A confirmar
- Presentación alumnos: A confirmar

Parcial-AA2:

- Fecha: Jueves **26/06**
- Recu: Jueves **17/07**

Aprendizaje Automático II

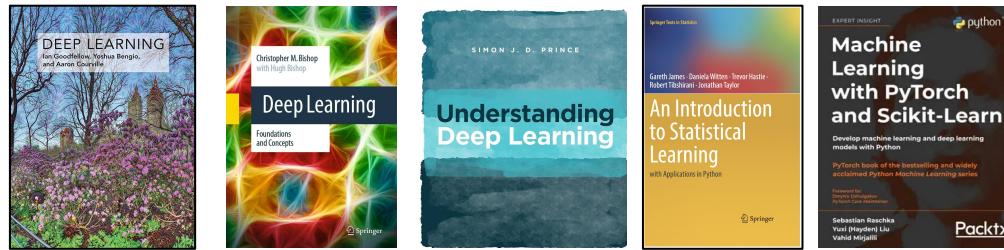
Optativa del Departamento de Computación.

Objetivos de la materia:

Una **introducción abarcativa** de los principales **conceptos, decisiones algorítmicas** y **efectos** de distintos **métodos** de aprendizaje automático.

Correlativas:

- Compu: AED 3 + MetNum + Proba
- Datos: AED 3 + ALC + Proba
- Física: Labo 5 (TPs), Teo 3, Cálculo N.



Bibliografía (AA1):

- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). **Deep learning**.
- Bishop, C. M., & Bishop, H. (2023). **Deep learning: Foundations and concepts**.
- Prince, S. J. (2023). **Understanding deep learning**.
- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). **An introduction to statistical learning: With applications in python**.
- Raschka, S., Liu, Y. H., & Mirjalili, V. (2022). **Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python**.

Herramientas



PyTorch

Programa (AA 2)

- Métodos de **Regresión** y **Descenso por Gradiente**.
- **Redes neuronales. Backpropagation.**
- **Predicción de Secuencias.** Redes neuronales recurrentes, Mecanismos de atención. Transformers.
- **Visión artificial.** Redes neuronales convolucionales.
- **Modelos Generativos.** Modelos de difusión. GANs.
- **Aprendizaje de Representaciones.** Autoencoders
- (quizás) clase colgada de Ingeniería de Atributos.
- (quizás) clase de calibración de modelos probabilísticos.

Regresión



Aprendizaje Supervisado

Objetivo del aprendizaje supervisado

Estimar la **función determinista $f(\mathbf{X})$** que determina la relación $\mathbf{X} \rightarrow \mathbf{Y}$:

Es decir, estimar f tal que $\mathbf{Y} = f(\mathbf{X}) + \varepsilon$ a través de un modelo $\hat{h}_D(\mathbf{X})$. En donde ε es el error irreducible.

Regresión

Objetivo del aprendizaje supervisado (caso regresión)

Estimar la **función determinista $f(\mathbf{X})$** que determina la relación $\mathbf{X} \rightarrow \mathbf{Y}$:

Es decir, estimar f tal que $\mathbf{Y} = f(\mathbf{X}) + \epsilon$ a través de un modelo $\hat{h}_D(\mathbf{X})$. En donde ϵ es el error irreducible.

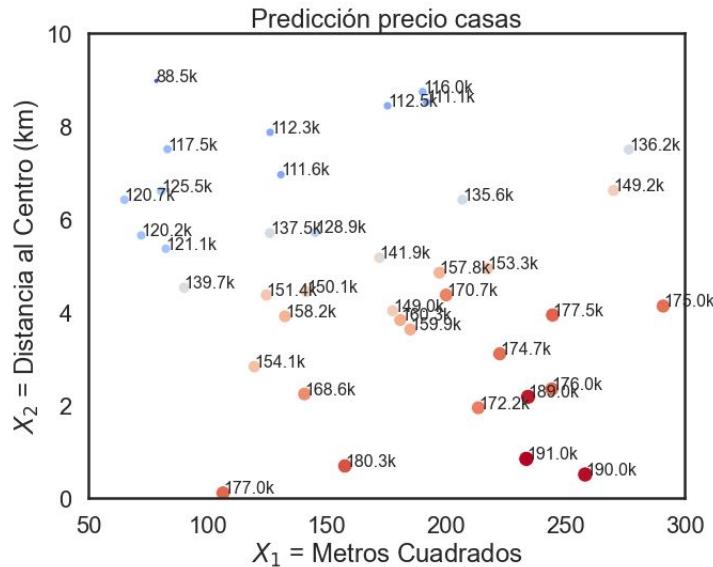
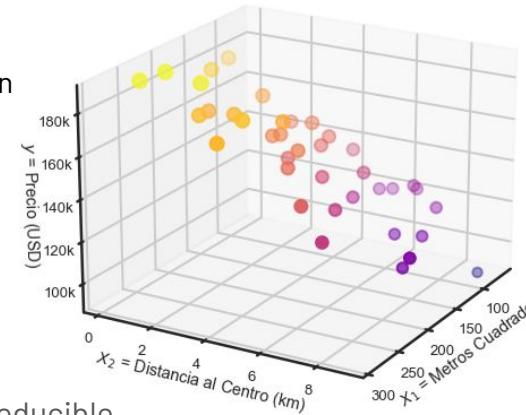
En donde $\mathbf{Y} \in \mathbb{R}$.

Ejemplo

Estimar el \mathbf{Y} = **precio de una casa** según
 X_1 = los metros cuadrados cubiertos,
 X_2 = su distancia al centro de la ciudad.

Nota: Otros atributos podrían haber sido la *cantidad de visitas al sitio de publicación* y el *color de la pintura*, el *nombre de la calle*, etc. Es decir, los atributos siguen teniendo la propiedad de ser atributos continuos, categóricos, nominales, etc, pero cambia lo que queremos predecir lo cual es determinado por **las etiquetas asociadas**.

Otra visualización (que a mi parecer confunde)



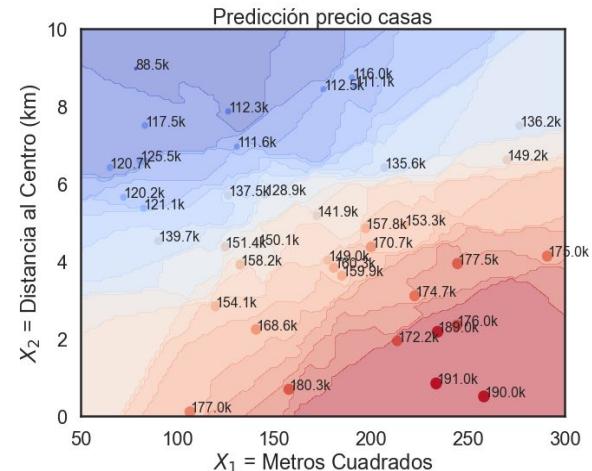
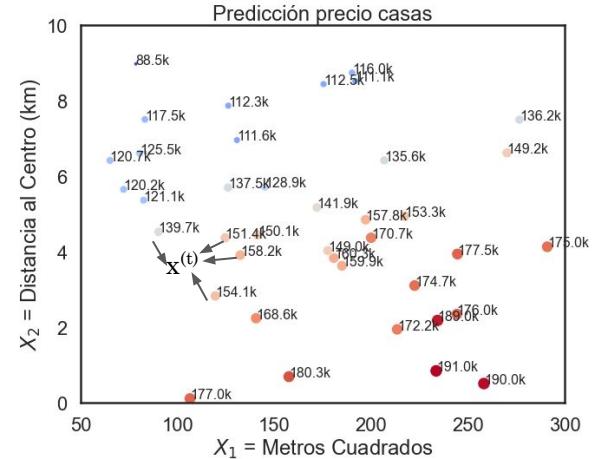
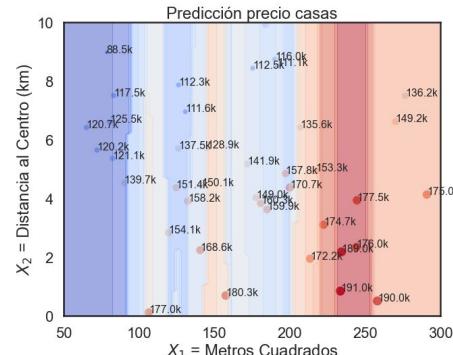
Regresión

KNN

Algoritmo de asignación a la instancia $x^{(t)}$:

1. Computar la distancia $D(x^{(t)}, x^{(j)})$ para todo punto de entrenamiento $x^{(j)}$.
2. Seleccionar las K instancias más cercanas y sus etiquetas.
3. Devolver la etiqueta más frecuente el promedio de los valores de los vecinos

Nota, como en clasificación, si el algoritmo utiliza distancias entre los atributos, primero reescalar para evitar



Regresión

Árboles de decisión

Sea \mathbf{S} una muestra de instancias con atributos \mathbf{A} . Para construir un árbol de decisión ejecutamos:

1. Elegimos el par $a \in A, c \in \mathbb{R}$ entre los posibles pares **<atributo, corte>**, que mejor divida a \mathbf{S} para **nodo_actual**.
2. Crear dos hijos del **nodo_actual**.
3. Dividir las instancias de \mathbf{S} en los nuevos nodos, según **<a,c>**:

$$\begin{aligned} S_{\leq} &\leftarrow \{x \mid x \in S \wedge x[a] \leq c\} \\ S_{>} &\leftarrow \{x \mid x \in S \wedge x[a] > c\} \end{aligned}$$

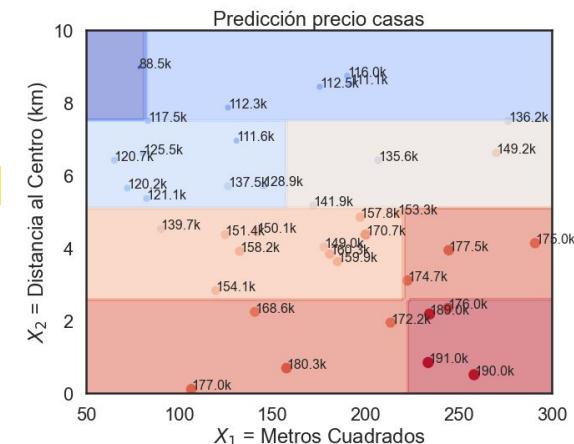
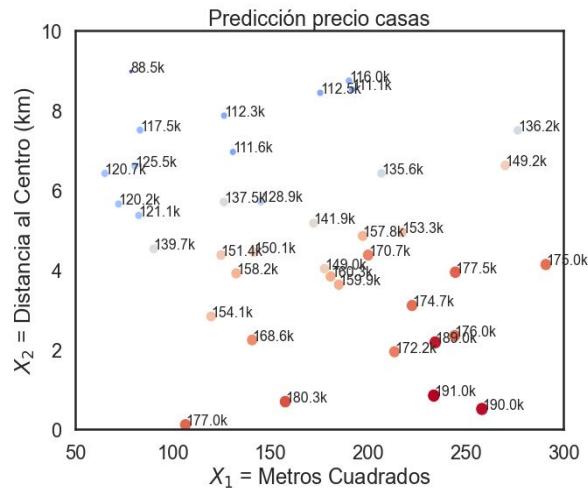
4. **repetir** para cada hijo en el que haya suficientes instancias.

El valor de una hoja será ~~la etiqueta más frecuente~~ el promedio de las instancias de la región

$$\Delta M(S, \langle a, c \rangle) = M(S) - (Prop_{\leq} * M(S_{\leq}) + Prop_{>} * M(S_{>}))$$

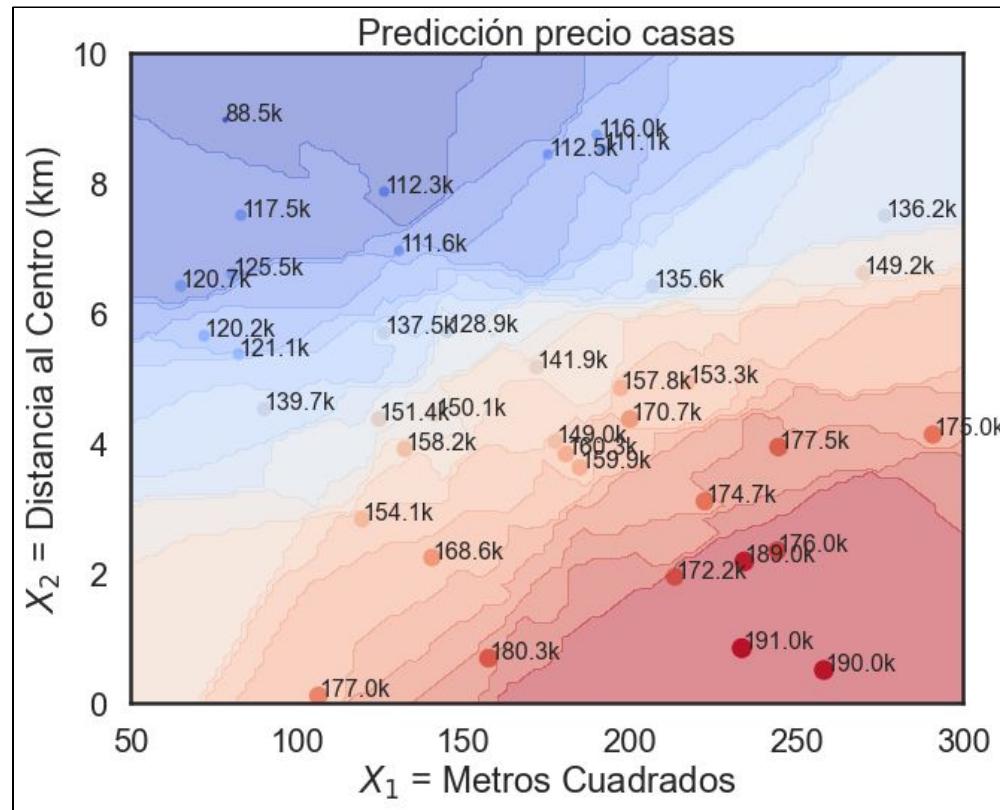
$M(S) :=$ podemos usar $VAR(S)$ por ejemplo.

(Equivalente a MSE tomando como valor de la región al promedio)



Regresión

Comparación de Algoritmos



Regresión Lineal



Regresión

Regresión Lineal

$$Y = f(X) + \varepsilon$$

Pensemos un nuevo algoritmo con el siguiente **sesgo inductivo**:

La función a estimar $f(X)$ es una combinación lineal de los atributos:

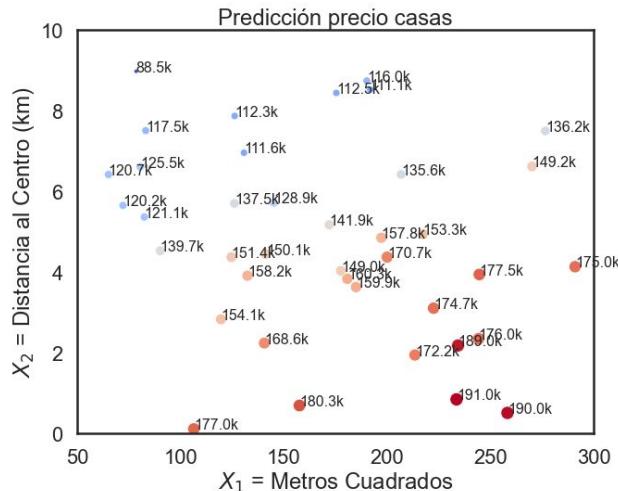
$$Y^{\text{sup}} = \text{comb_lineal}_w(X') + \varepsilon \quad \text{en donde } X' = [1, X_1, X_2, \dots, X_p]$$

$$Y = X'w + \varepsilon$$

$$Y = w_0 + w_1X_1 + w_2X_2 + \dots + w_pX_p + \varepsilon \quad \text{en donde } w_i \in \mathbb{R}$$

$w_1 \dots w_p$ son "pesos" (números) que no conocemos (y desearíamos **aprender**), que multiplican a cada atributo.

w_0 es un agregado (que también queremos aprender) para poder representar funciones que no pasen por el origen, y es una manera de sumar un valor independiente de los atributos



También llamado "intercept", de intersección en inglés, o "término bias".

Regresión

Regresión Lineal

$$Y = f(X) + \varepsilon$$

Pensemos un nuevo algoritmo con el siguiente **sesgo inductivo**:

La función a estimar $f(X)$ es una combinación lineal de los atributos:

$$Y^{\text{sup}} = \text{comb_lineal}_w(X') + \varepsilon \quad \text{en donde } X' = [1, X_1, X_2, \dots, X_p]$$

$$Y = X'w + \varepsilon$$

$$Y = w_0 + w_1X_1 + w_2X_2 + \dots + w_pX_p + \varepsilon \quad \text{en donde } w_i \in \mathbb{R}$$

$w_0, w_1 \dots w_p$ son "pesos" (números) que se desearíamos **aprender**

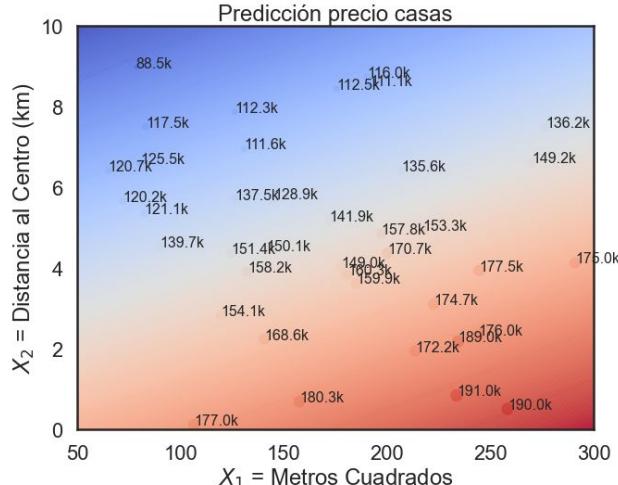
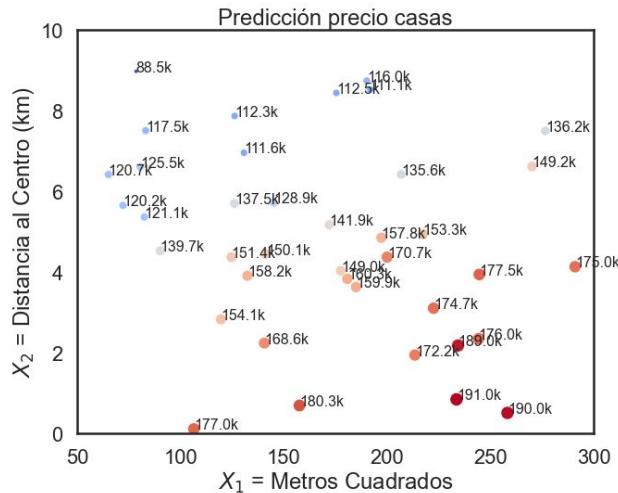
Un modelo ya entrenado tendrá la siguiente pinta:

$$\hat{h}_{\hat{w}_0, \dots, \hat{w}_p}(x^{(i)}) = \hat{w}_0 + \hat{w}_1x_1^{(i)} + \hat{w}_2x_2^{(i)} + \dots + \hat{w}_px_p^{(i)}$$

Llamamos a estos pesos estimados, **parámetros** del modelo (de ahora en más sin sombrerito para simplificar la notación).

Ejercicio: estimar (hipotéticamente) w_0, w_1, w_2 para el siguiente problema:

- Y : precio casas en CABA (en USD) según:
- X_1 : metros cuadrados cubiertos y X_2 : kilómetros al obelisco.



Notar que en este caso no hay "regiones" como tal. Si no una asignación continua.

MSE (Error Cuadrático Medio)

Repaso:

$$\text{MSE}_{X,y} = \frac{1}{n} \sum_{i=1}^n (\hat{h}(x^{(i)}) - y^{(i)})^2$$

Para el caso de un modelo de regresión lineal

$$\begin{aligned}\text{MSE}_{X,y} &= \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_p x_p^{(i)} - y^{(i)})^2\end{aligned}$$

Ejercicio

- Y: precio casas (en USD):
 - X_1 : metros cuadrados cubiertos
 - X_2 : distancia al obelisco (km).
- ★ $w_0 = 80000$
- ★ $w_1 = 150$
- ★ $w_2 = -5000$

Estos valores estimados w_0 , w_1 y w_2 "sugieren" que el precio base de una casa es de \$80 000 y que el precio de una casa aumenta \$150 por cada metro cuadrado adicional de superficie y disminuye \$5000 por cada kilómetro adicional desde el obelisco.

Calcular el MSE para este dataset:

- $x^{(1)} : (100 \text{ mts}^2, 5 \text{ km}) y^{(1)} : \72.000
- $x^{(2)} : (300 \text{ mts}^2, 7 \text{ km}) y^{(2)} : \89.000
- $x^{(3)} : (100 \text{ mts}^2, 5 \text{ km}) y^{(3)} : \68.000
- $x^{(4)} : (50 \text{ mts}^2, 2 \text{ km}) y^{(4)} : \77.500

MSE (Error Cuadrático Medio)

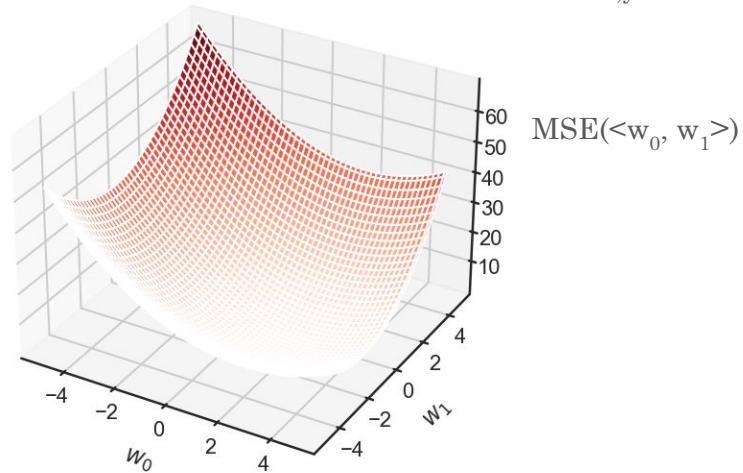
Reaso:

$$\text{MSE}_{X,y} = \frac{1}{n} \sum_{i=1}^n (\hat{h}(x^{(i)}) - y^{(i)})^2$$

Para el caso de un modelo de regresión lineal

$$\begin{aligned}\text{MSE}_{X,y} &= \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_p x_p^{(i)} - y^{(i)})^2\end{aligned}$$

Un nuevo gráfico. Pensemos al MSE como **una función** de los parámetros de un modelo $\text{MSE}_{X,y}(w)$.



$$\text{MSE}_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2$$

"Entrenar el modelo": encontrar los w . ¿Cuáles?

Regresión

Regresión Lineal

"Entrenar el modelo": encontrar los w que minimicen la siguiente expresión:

$$\text{MSE}_{\mathbf{X}, \mathbf{y}}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2$$

¿Cómo podemos **estimar los pesos** a partir de datos en regresión lineal?

- Cuadrados mínimos
 - Solución **analítica** para minimizar el MSE
- Descenso de gradiente (método iterativo)
 - Solución **iterativa** para minimizar el MSE.
 - Menos utilizado en la práctica
(salvo **datasets muy grandes**, lo veremos por cuestiones pedagógicas)
- Entre otros

$$\mathbf{w} = \left(\mathbf{X}'_{\text{train}} {}^T \mathbf{X}'_{\text{train}} \right)^{-1} \mathbf{X}'_{\text{train}} {}^T \mathbf{y}_{\text{train}}$$

No lo veremos en la materia

$$w_j^{\text{paso}=k} \leftarrow w_j - \alpha \times \text{update}(X'_{\text{train}}, y_{\text{train}}, w^{\text{paso}=k-1})$$

A continuación

Abrimos paréntesis: Descenso por el Gradiente



[https://beta.peakd.com/hive-148441/@ellenripley/a-walk
to-find-the-brick-ball](https://beta.peakd.com/hive-148441/@ellenripley/a-walk-to-find-the-brick-ball)

Cómo encontrar el mínimo de una función: $z_{min} = \min_z(g(z))$

Dada una función arbitraria $g(z)$, diferenciable, el **gradiente** en un punto $z^{(i)}$ es un vector que indica la dirección de mayor crecimiento de la función. La magnitud del gradiente representa la tasa de cambio en esa dirección.

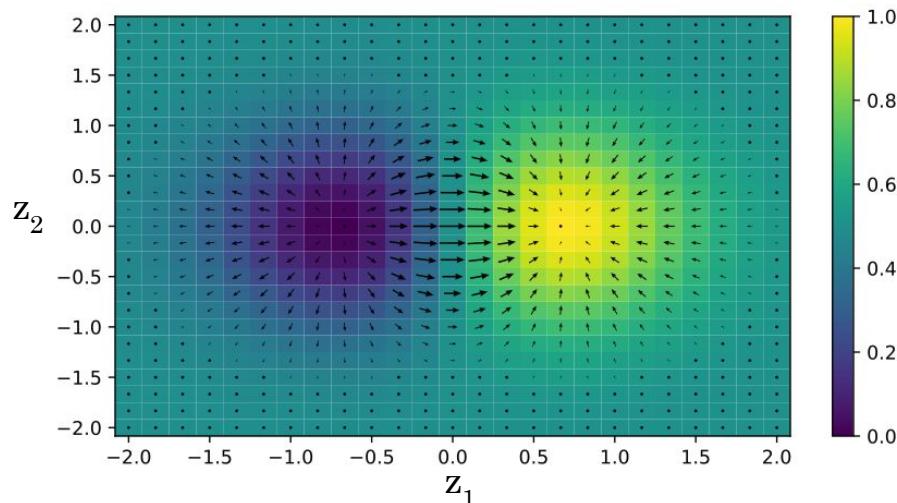
Nota: Llamé a la función $g(z)$ en vez de $f(x)$ o $f(w)$ sólo para evitar asociaciones confusas con lo que venimos viendo. Lo mismo \mathbf{R}^k en vez de \mathbf{R}^n

Para encontrar el **mínimo** de una función simplemente calculamos cuándo

$$\nabla g(z) = 0 \quad \text{¿No?}$$

- Podrían ser mínimos locales, puntos de silla, o máximos.
- No siempre es posible derivar una solución de forma cerrada $\nabla g(z)$.

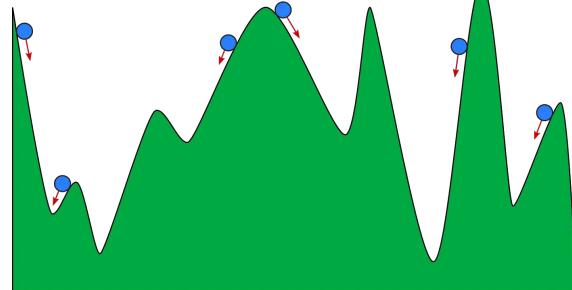
$$g: \mathbb{R}^k \rightarrow \mathbb{R}$$
$$\nabla g: \mathbb{R}^k \rightarrow \mathbb{R}^k$$
$$\nabla g(z) = \begin{bmatrix} \frac{\partial g}{\partial z_1}(z) \\ \vdots \\ \frac{\partial g}{\partial z_k}(z) \end{bmatrix} \text{ en donde } z = (z_1, \dots, z_k)$$



Cómo encontrar el mínimo de una función: $z_{min} = \min_z(g(z))$

Dada una función arbitraria $g(z)$, diferenciable, el **gradiente** en un punto $z^{(i)}$ es un vector que indica la dirección de mayor crecimiento de la función. La magnitud del gradiente representa la tasa de cambio en esa dirección.

Idea, si nos parásemos en un z arbitrario, y nos movemos en la dirección contraria a la dirección de mayor crecimiento (que es la de mayor descenso), encontraríamos un mínimo (local)

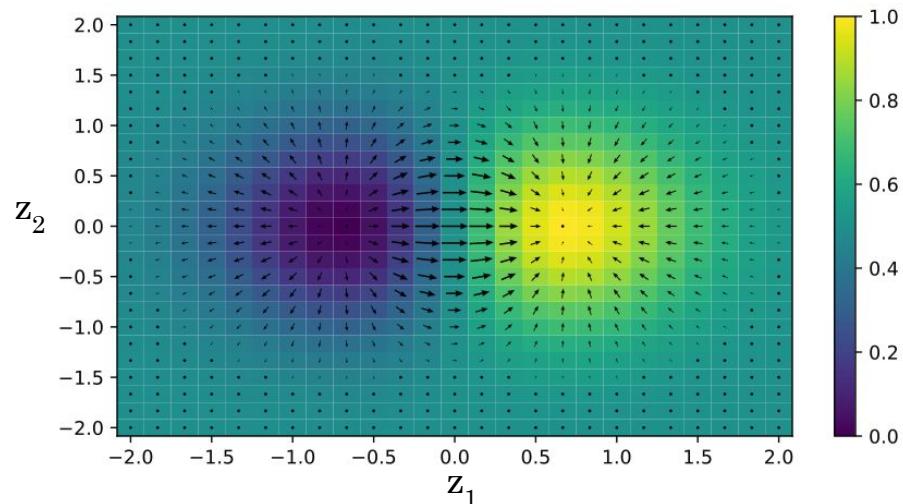


<https://towardsdatascience.com/machine-learning-101-an-intuitive-introduction-to-gradient-descent-366b77b52645>

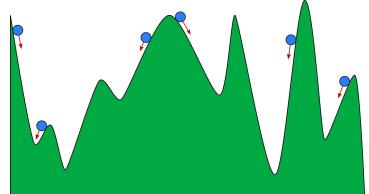
$$g: \mathbb{R}^k \rightarrow \mathbb{R}$$

$$\nabla g: \mathbb{R}^k \rightarrow \mathbb{R}^k$$

$$\nabla g(z) = \begin{bmatrix} \frac{\partial g}{\partial z_1}(z) \\ \vdots \\ \frac{\partial g}{\partial z_k}(z) \end{bmatrix} \text{ en donde } z = (z_1, \dots, z_k)$$



<https://en.wikipedia.org/wiki/Gradient>



Descenso de Gradiente

(Cauchy, 1847)

Algoritmo de optimización utilizado para **minimizar el valor de una función arbitraria diferenciable**, ajustando **iterativamente** sus parámetros en la dirección del descenso más pronunciado de la función.

Dada una función $g(\mathbf{z})$, donde \mathbf{z} es un vector:

- 1) Inicializar \mathbf{z} con un valor al azar.
- 2) Repetir:
 - a) calcula el gradiente de la función con respecto a \mathbf{z} , denotado como $\nabla g(\mathbf{z})$, que da la dirección del ascenso más pronunciado de la función.
 - b) Actualizar
 $\mathbf{z} \leftarrow \mathbf{z} - \alpha * \nabla g(\mathbf{z})$ en donde α es el learning rate

Learning rate α (Tasa de aprendizaje):

Es un hiperparámetro que determina el tamaño del paso que se toma en cada iteración para actualizar los parámetros del modelo.

```
def descenso_gradiente(g, dg, z_init, alpha, num_iterations, tol):
    """
    Descenso de gradiente para minimizar g.
    Args:
        g: La función a optimizar.
        dg: El gradiente de la función.
        z_init: Valor inicial.
        alpha: El "learning rate".
        num_iterations: Máx iteraciones.
        tol: Tolerancia para la convergencia.
    """
    z = z_init
    for i in range(num_iterations):
        gradient = dg(z) # gradient vale por ej <0.5, -0.2, -3, 0>
        z_new = z - alpha * gradient
        if abs(g(z_new) - g(z)) < tol:
            break
        z = z_new
    return z
```

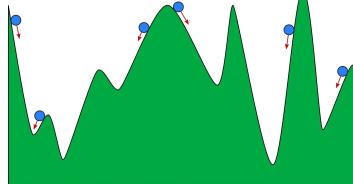
Descenso de Gradiente

(Cauchy, 1847)

Algoritmo de optimización utilizado para **minimizar el valor de una función arbitraria diferenciable**, ajustando **iterativamente** sus parámetros en la dirección del descenso más pronunciado de la función.

Dada una función $g(z)$, donde z es un vector:

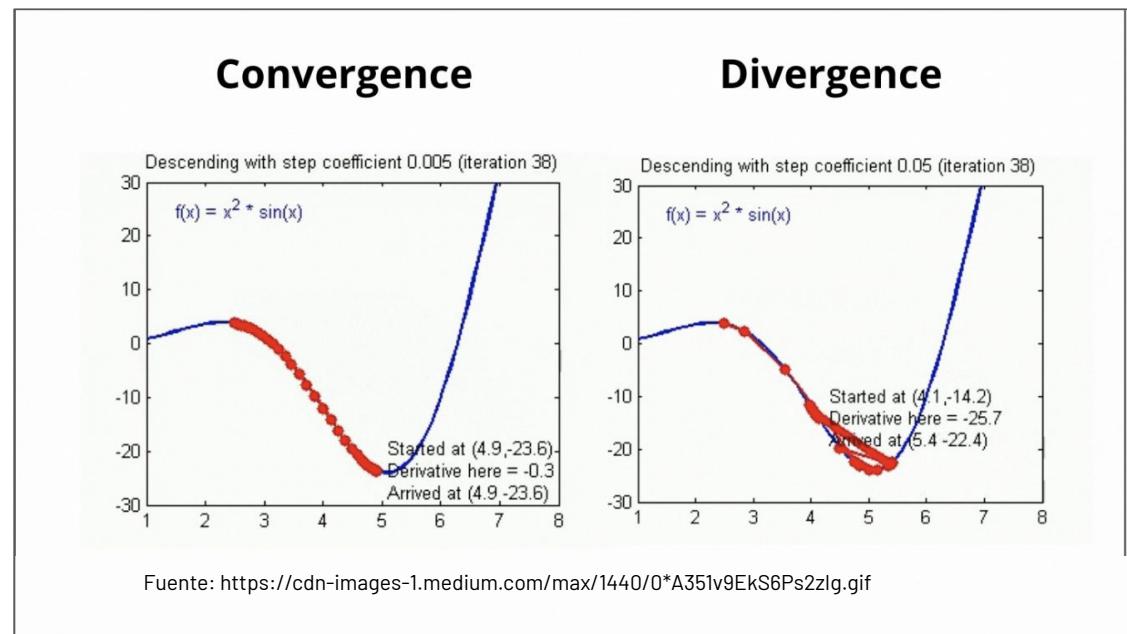
- 1) Inicializar z con un valor al azar.
- 2) Repetir:
 - a) calcula el gradiente de la función con respecto a z , denotado como $\nabla g(z)$, que da la dirección del ascenso más pronunciado de la función.
 - b) Actualizar
$$z \leftarrow z - \alpha * \nabla g(z)$$
 en donde α es el learning rate



Learning rate α (Tasa de aprendizaje):

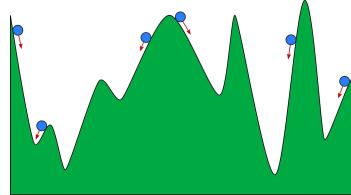
Es un hiperparámetro que determina el tamaño del paso que se toma en cada iteración para actualizar los parámetros del modelo.

Veamos cómo el Learning rate puede afectar a la convergencia del método:



Descenso de Gradiente

(Cauchy, 1847)



Algoritmo de optimización utilizado para **minimizar el valor de una función arbitraria diferenciable**, ajustando **iterativamente** sus parámetros en la dirección del descenso más pronunciado de la función.

Dada una función $\mathbf{g}(\mathbf{z})$, donde \mathbf{z} es un vector:

- 1) Inicializar \mathbf{z} con un valor al azar.
- 2) Repetir:
 - a) calcula el gradiente de la función con respecto a \mathbf{z} , denotado como $\nabla \mathbf{g}(\mathbf{z})$, que da la dirección del ascenso más pronunciado de la función.
 - b) Actualizar
$$\mathbf{z} \leftarrow \mathbf{z} - \alpha * \nabla \mathbf{g}(\mathbf{z})$$
 en donde α es el learning rate

El descenso de gradiente es especialmente útil, ya que permite la optimización de funciones complejas con muchos parámetros, lo que puede ser computacionalmente costoso o incluso imposible de hacer analíticamente. Estas funciones suelen ser, en el campo de aprendizaje automático, las llamadas "[funciones de costo](#)".

Funciones de costo (o funciones de pérdida).

Medida cuantitativa de [cuán bien](#) un modelo de aprendizaje automático se [ajusta a los datos de entrenamiento](#).

Mide la diferencia entre las predicciones del modelo y los valores reales de los datos de entrenamiento.

[Algunos ejemplos](#): MSE (hoy), cross-entropy (proximamente), etc.

Cerramos paréntesis

Minimizando el MSE

Para encontrar los pesos óptimos w_0, w_1, \dots, w_p según un dataset, podemos considerar $MSE(w)$ como la función que deseamos optimizar.

Veamos cómo utilizar Descenso por el Gradiente, con $\mathbf{g(z)} = \mathbf{MSE(w)}$. Necesitamos como vimos $\nabla g(z)$. **Por suerte**, no es muy difícil encontrar esta fórmula, es decir (ej práctica):

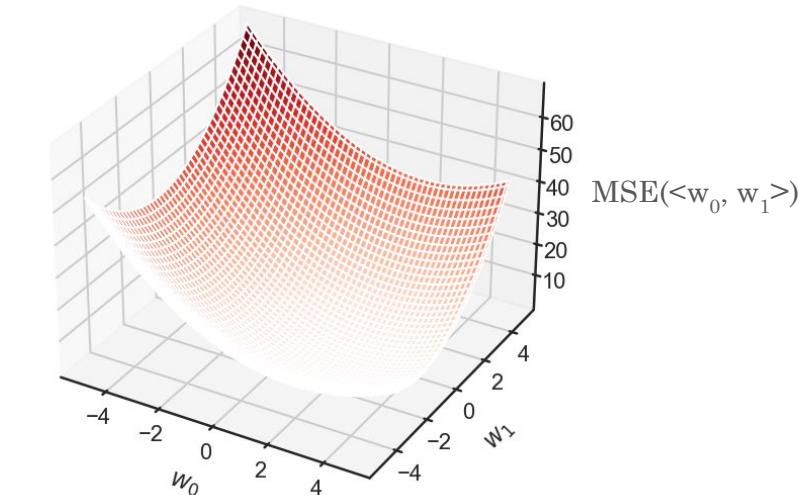
$$\nabla_w MSE_{X,y}(w) = \frac{2}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)}) * x^{(i)}$$

```
def fit(self, X_train, y_train):
    X_train_ext = np.hstack((ones_column, X_train))

def mse_reg_lin(w):
    return (1 / len(y_train)) * np.sum((np.dot(X_train_ext, w) - y_train)**2)

def grad_mse_reg_lin(w):
    return (2 / len(y_train)) * np.dot(X_train_ext.T, np.dot(X_train_ext, w) - y_train)

self.w = descenso_gradiente(mse_reg_lin, grad_mse_reg_lin, z_init=zeros, alpha=0.01, num_iterations=1000, tol=0.01)
```



Minimizando el MSE

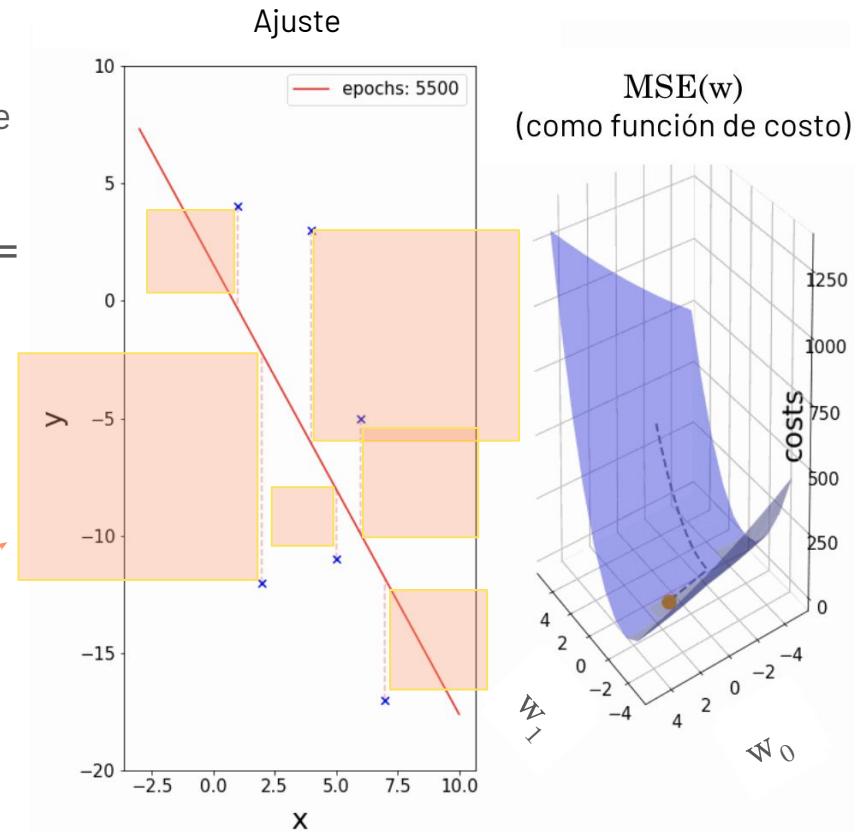
$$\text{MSE}_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2$$

Para encontrar los pesos óptimos w_0, w_1, \dots, w_p según un dataset, podemos considerar $\text{MSE}(w)$ como la función que deseamos optimizar.

Veamos cómo utilizar Descenso por el Gradiente, con $g(z) = \text{MSE}(w)$. Necesitamos como vimos $\nabla g(z)$. **Por suerte**, no es muy difícil encontrar esta fórmula, es decir (ej práctica):

$$\nabla_w \text{MSE}_{X,y}(w) = \frac{2}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)}) * x^{(i)}$$

Los rectángulos sombreados muestran lo que se está minimizando. La suma de los cuadrados del error cometido por el modelo



Abrimos otro paréntesis:
Descenso por el Gradiente con Mini Batches

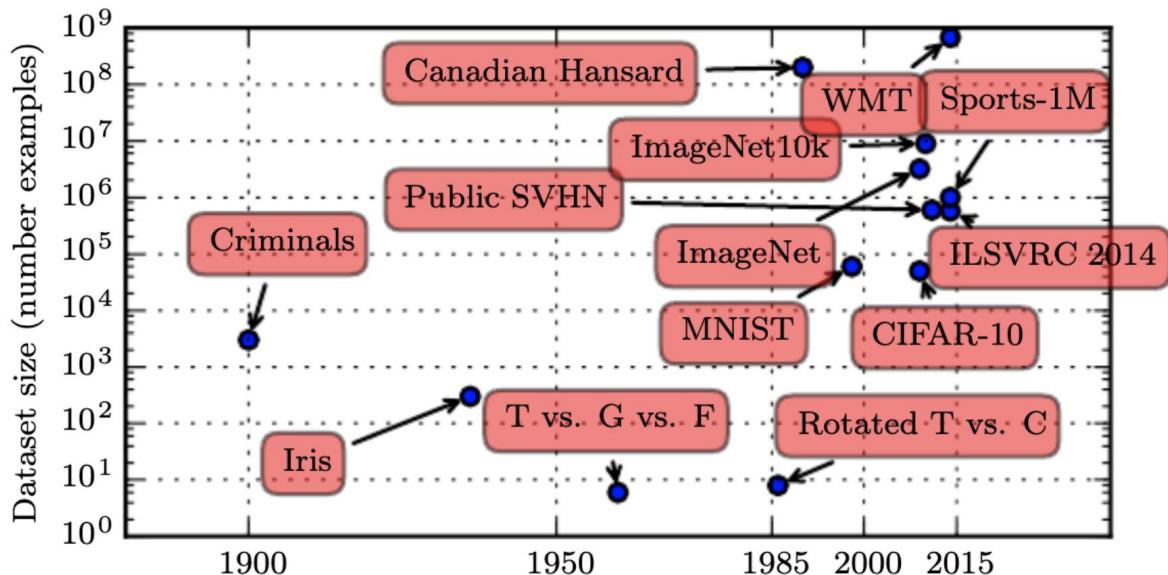
Descenso por el Gradiente con Mini Batches

Dijimos que el algoritmo Descenso por el Gradiente es especialmente útil para datasets muy grandes.

¿Por qué?

Para computar $MSE_{x,y}(w)$ necesitamos cargar en memoria al dataset completo. Si n y p son muy grandes, es muy posible que la información para un paso de descenso de gradiente

no entre en la memoria!



Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Descenso por el Gradiente con Mini Batches

“mini-batches”

Hacemos el update de los parámetros **sólo para con un subconjunto de los datos de train.**

- Si **batch_size = n**: batch gradient descent (el que ya vimos)
- Si **1 < batch_size < n**: mini-batch GD
- Si **batch_size = 1**: stochastic GD.

Terminología

Iteración (step, update): Una iteración se refiere a un **ciclo de actualización de los parámetros del modelo** utilizando un mini-batch específico. En cada iteración, **se calcula el gradiente utilizando el mini-batch actual** y se actualizan los parámetros del modelo en la dirección opuesta al gradiente.

Época (epoch): Una época se define como un **ciclo completo de entrenamiento** en el que **se utilizan todos los mini-batches disponibles en el conjunto de datos de entrenamiento**. En cada época, se realizan múltiples iteraciones utilizando diferentes mini-batches hasta que todos los datos de entrenamiento se hayan utilizado al menos una vez.

Algorithm: Algoritmo Mini Batch Gradient Descent

Inputs: X_{train} , y_{train} , α (learning rate), z_{init} , g , ∇g

Iniciar: $z = z_{init}$

while no se cumpla criterio detención **do**

// corrida para una epoch:

while haya datos para el mini batch **do**

1) Muestrear el mini batch:

$X_{batch}, y_{batch} = muestrear_sin_repo(X_{train}, y_{train})$;

2) Computar el gradiente para el minibatch

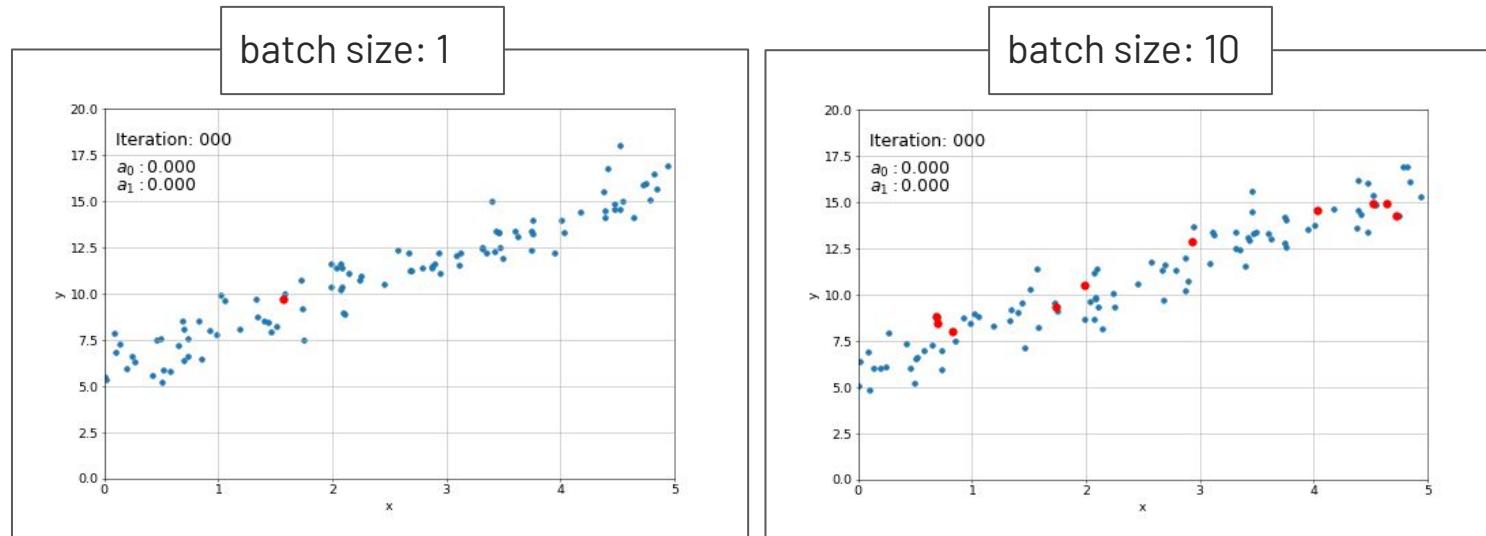
$grad = \nabla g_{(X_{batch}, y_{batch})}(z)$;

3) Actualizar los parámetros:

$z = z - \alpha * grad$;

Output: z

Descenso por el Gradiente con Mini Batches



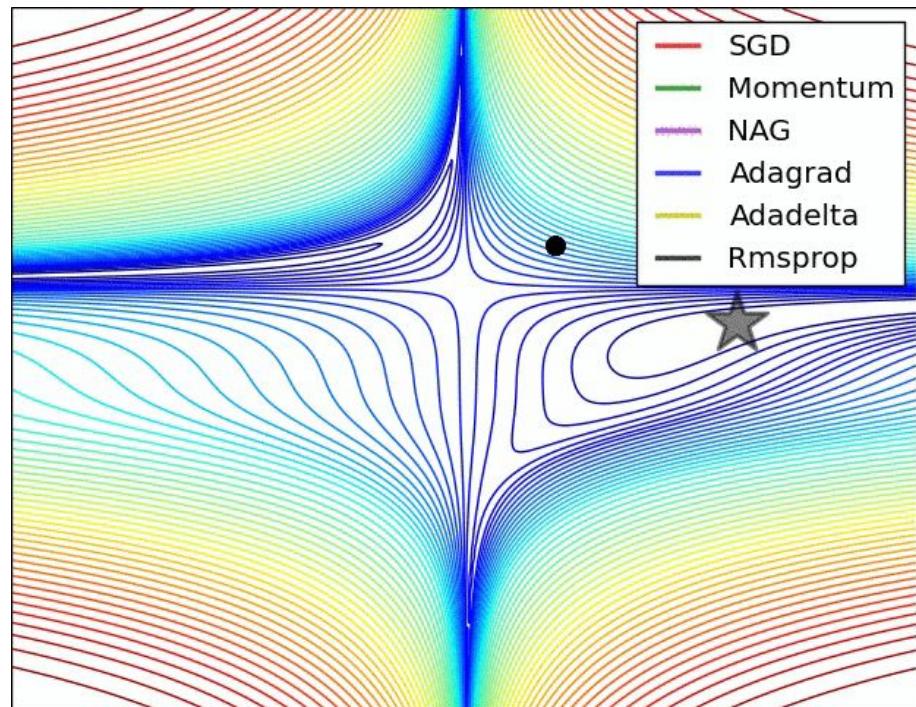
<https://towardsdatascience.com/batch-mini-batch-and-stochastic-gradient-descent-for-linear-regression-9fe4eefa637c>

¿Qué uso entonces?

Gradient Descent (GD, SGD, mini-batch GD) son herramientas básicas que han sido superadas en la práctica por otras técnicas (tales como):

- Adam
- RMSprop
- Adagrad.
- NAG
- etc

Para ver más métodos, pueden empezar por **Deep Learning (sec 8.3)** o blogs tal como <https://www.ruder.io/optimizing-gradient-descent/>



<https://www.ruder.io/optimizing-gradient-descent/>

Cerramos paréntesis

Siguiendo con Regresión Lineal

Antes de continuar, cabe aclarar que Regresión Lineal ha sido una de las herramientas principales de la estadística durante los últimos >50 años.

Es todo un mundo, hay libros completos al respecto. **Aquí estamos arañando la superficie del tema** y viéndola desde el punto de vista de Aprendizaje Automático.



<https://i.gifer.com/g20B.gif>

Ingeniería de atributos: “Funciones base”

La regresión lineal modela la relación entre las variables independientes (atributos) y la variable dependiente (target) **de forma lineal**. Pero en muchos casos del mundo real, estas relaciones **son no lineales**.

Al agregar **versiones transformadas de los atributos existentes**, se puede ayudar al modelo a aproximar esas relaciones no lineales, manteniendo el modelo lineal.

Por ejemplo:

Si el atributo original es X_1 , y la relación real es cuadrática ($Y = aX_1^2 + bX_1 + c + \epsilon$), una regresión lineal estándar no podrá capturarla correctamente.

Idea: Si agregamos X_1^2 como un nuevo atributo, el modelo ahora puede aprender coeficientes para X_1 y X_1^2 , permitiéndole **“ajustar una curva”**.

¿Ajusta una curva realmente? (**pensar en SVM**)

Las **funciones base** son funciones que transforman los atributos de entrada para que un modelo lineal pueda representar relaciones no lineales.

Atributo original	Atributo transformado	Nombre de la transformación
x	x^2, x^3, \dots	Polinomial
x	$\log(x), \sqrt{x}$	Transformación logarítmica / raíz
x_1, x_2	$x_1 \cdot x_2$	Término de interacción
x	$\sin(x), \cos(x)$	Transformación trigonométrica

Potencial problema: Sobreajuste. **¿por qué?**

Ingeniería de atributos: “Funciones base”

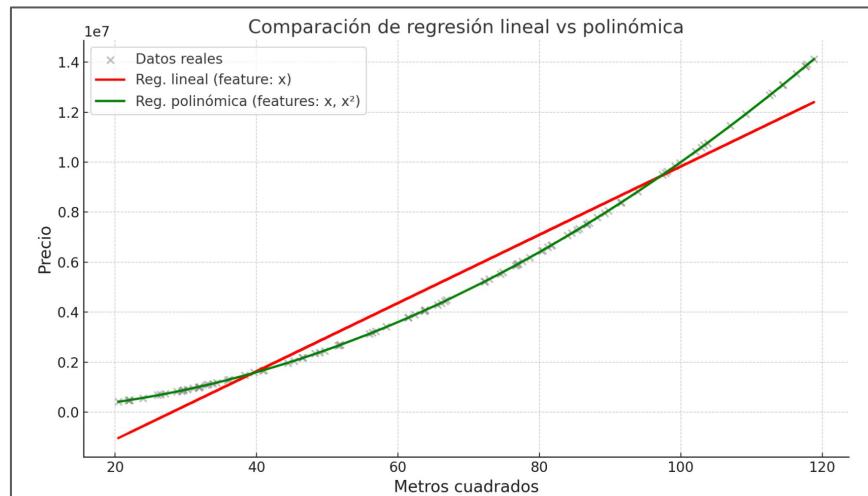
La regresión lineal modela la relación entre las variables independientes (atributos) y la variable dependiente (target) **de forma lineal**. Pero en muchos casos del mundo real, estas relaciones **son no lineales**.

Al agregar **versiones transformadas de los atributos existentes**, se puede ayudar al modelo a aproximar esas relaciones no lineales, manteniendo el modelo lineal.

Por ejemplo:

Si el atributo original es X_1 , y la relación real es cuadrática ($Y = aX_1^2 + bX_1 + c + \epsilon$), una regresión lineal estándar no podrá capturarla correctamente.

```
x = df[['metros cuadrados']]  
y = df['precio']  
  
poly = PolynomialFeatures(degree=2, include_bias=False)  
  
x_poly = poly.fit_transform(X)  
  
model = LinearRegression()  
model.fit(x_poly, y)
```



Regularización

Regularización

Muchas veces es necesario “restringir” los pesos que un algoritmo (en este caso regresión lineal) aprende.

¿Por qué?

- a) A veces tenemos atributos generados a partir de otros, por ej [$X_1, X_2, X_1^2, X_2^2, X_1 * X_2, \log(X_1), \dots$] lo cual le dan flexibilidad al modelo, pero más **chances de overfitting**.
- b) Cuando tenemos una gran cantidad de atributos, **una dimensión muy alta**, queremos focalizar el efecto en unos pocos atributos.
- c) Evitar problemas de **matrices no invertibles** por colinearidades (motivación original para regresión ridge) [Hoerl and Kennard, 1970].
- d) etc

$$\text{MSE}_{X,y}(w_0, w_1, \dots, w_p) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2$$

Nuevas funciones de costo

(no influyen al w_0)

$$J_{\text{Ridge}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_2^2 \text{ en donde } w' = (w_1, w_2, \dots)$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p w_j^2$$

$$J_{\text{Lasso}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_1$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p |w_j|$$

$$J_{\text{ElasticNet}}(w) = \text{MSE}_{X,y}(w) + \lambda_1 \|w'\|_1 + \lambda_2 \|w'\|_2^2$$

$$J_{\text{Ridge}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_2^2 \text{ en donde } w' = (w_1, w_2, \dots)$$

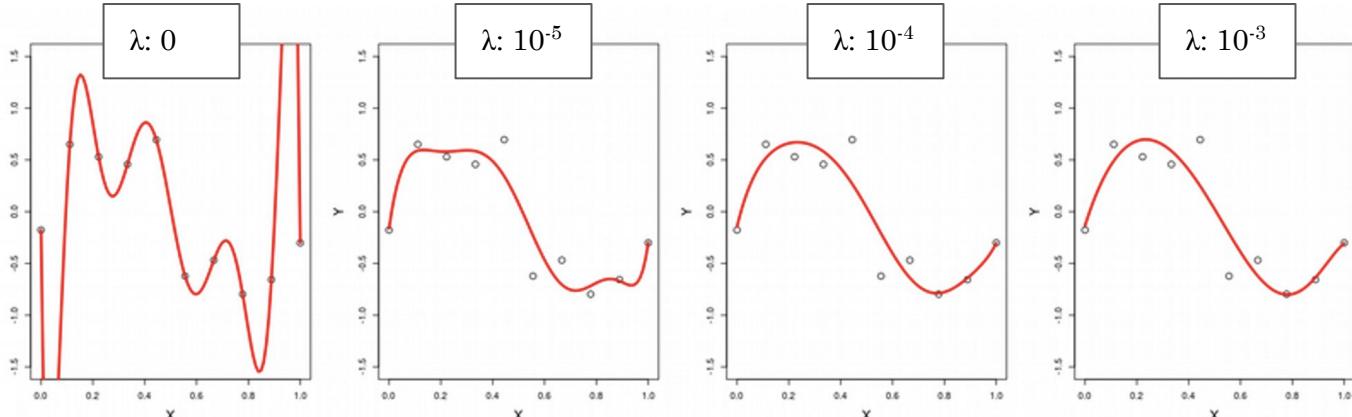
$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p w_j^2$$

Regularización

Ejemplo (ridge)

Ej

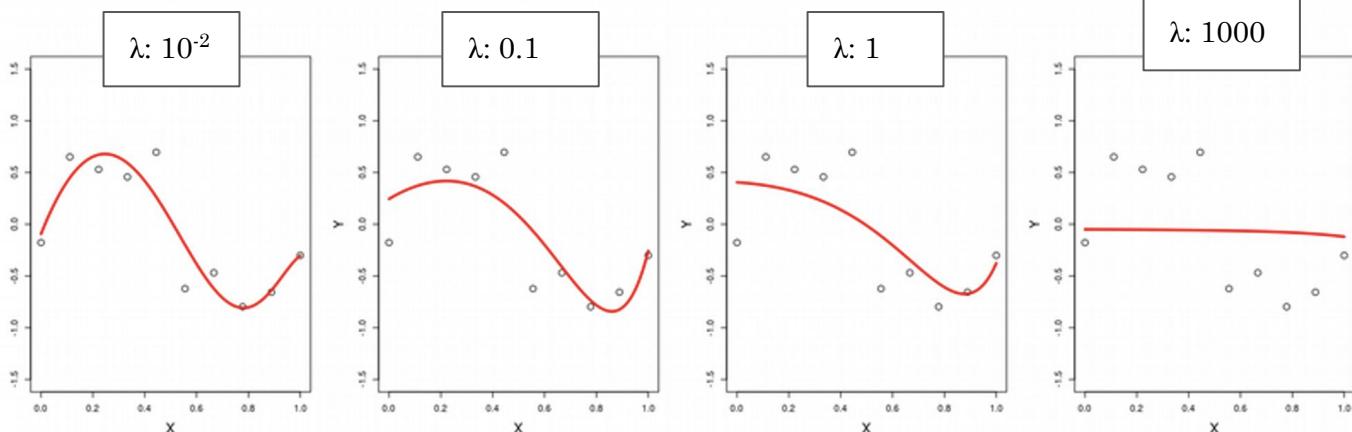
$X_{\text{poly}} = [X_1, X_1^2, X_1^3, \dots]$
 $\text{LR.fit}(X_{\text{poly}}, y)$



Nota:

¿Por qué no son líneas?

Incrementamos la dimensión, ajustamos en la dimensión alta, proyectamos las fronteras en el espacio original \Rightarrow la frontera de decisión en los atributos originales deja de ser un hiperplano (a lo SVM).



Regularización

Como filtro de atributos

Se puede ver que minimizar J_{Ridge} y J_{Lasso} es equivalente a resolver los siguientes problemas (respectivamente):

$$\min_w \left\{ \sum_{i=1}^n \left(\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)} \right)^2 \right\} \quad \text{sujeto a } \|w'\|_2^2 \leq s$$

$$\min_w \left\{ \sum_{i=1}^n \left(\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)} \right)^2 \right\} \quad \text{sujeto a } \|w'\|_1 \leq s$$

Visualmente

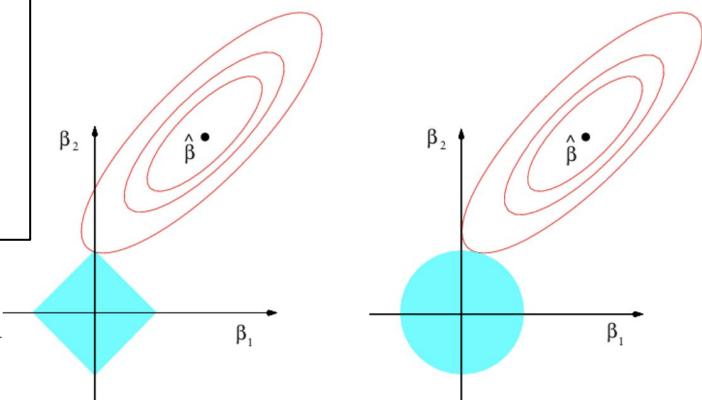
¿Dónde encontraremos el mínimo de la función restringido a las zonas azules?

$$J_{\text{Ridge}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_2^2 \text{ en donde } w' = (w_1, w_2, \dots)$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p w_j^2$$

$$J_{\text{Lasso}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_1$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p |w_j|$$

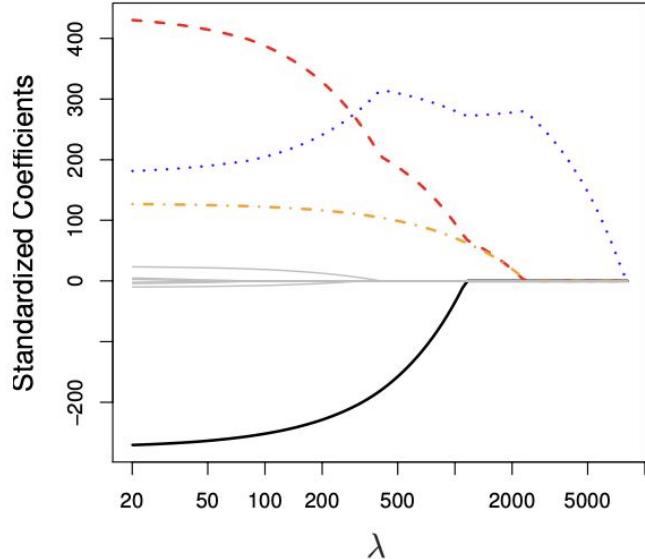


Fuente: **ISLR** Fig 6.7, (donde $\beta = w$)
En donde $\hat{\beta}$ representa el mínimo sin restricciones

Regularización

Como filtro de atributos

Lasso



Fuente: ISLR

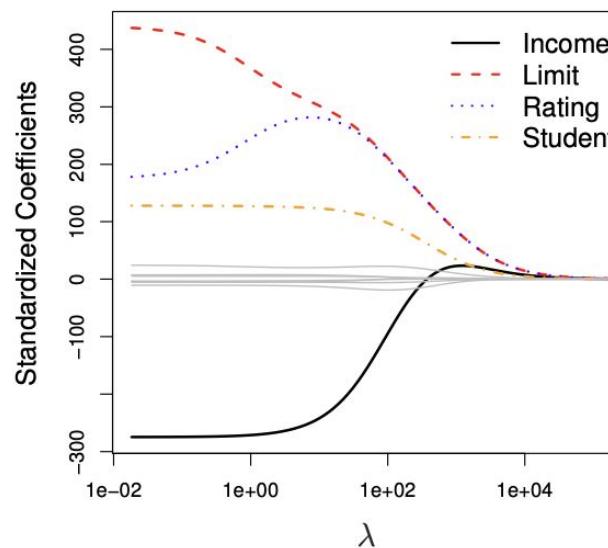
$$J_{\text{Ridge}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_2^2 \text{ en donde } w' = (w_1, w_2, \dots)$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p w_j^2$$

$$J_{\text{Lasso}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_1$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p |w_j|$$

Ridge



Fuente: ISLR

Regularización Gradientes

Utilizando las siguientes definiciones, tendrán todo lo necesario para utilizar descenso del gradiente con estas nuevas funciones de pérdida:

$$J_{\text{Ridge}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_2^2 \text{ en donde } w' = (w_1, w_2, \dots)$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p w_j^2$$

$$J_{\text{Lasso}}(w) = \text{MSE}_{X,y}(w) + \lambda \|w'\|_1$$

$$= \text{MSE}_{X,y}(w) + \lambda \sum_{j=1}^p |w_j|$$

$$\nabla_w \text{MSE}_{X,y}(w) = \frac{2}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)}) * x^{(i)}$$

$$\nabla_w J_{\text{Lasso}}(w) = \nabla_w \text{MSE}_{X,y}(w) + \lambda * signos(w)$$

$$\nabla_w J_{\text{Ridge}}(w) = \nabla_w \text{MSE}_{X,y}(w) + \lambda * w * 2$$

Sin embargo, puede que algunos valores no den como esperan.

Sugerencia, buscar "Subgradient Descent for Lasso"

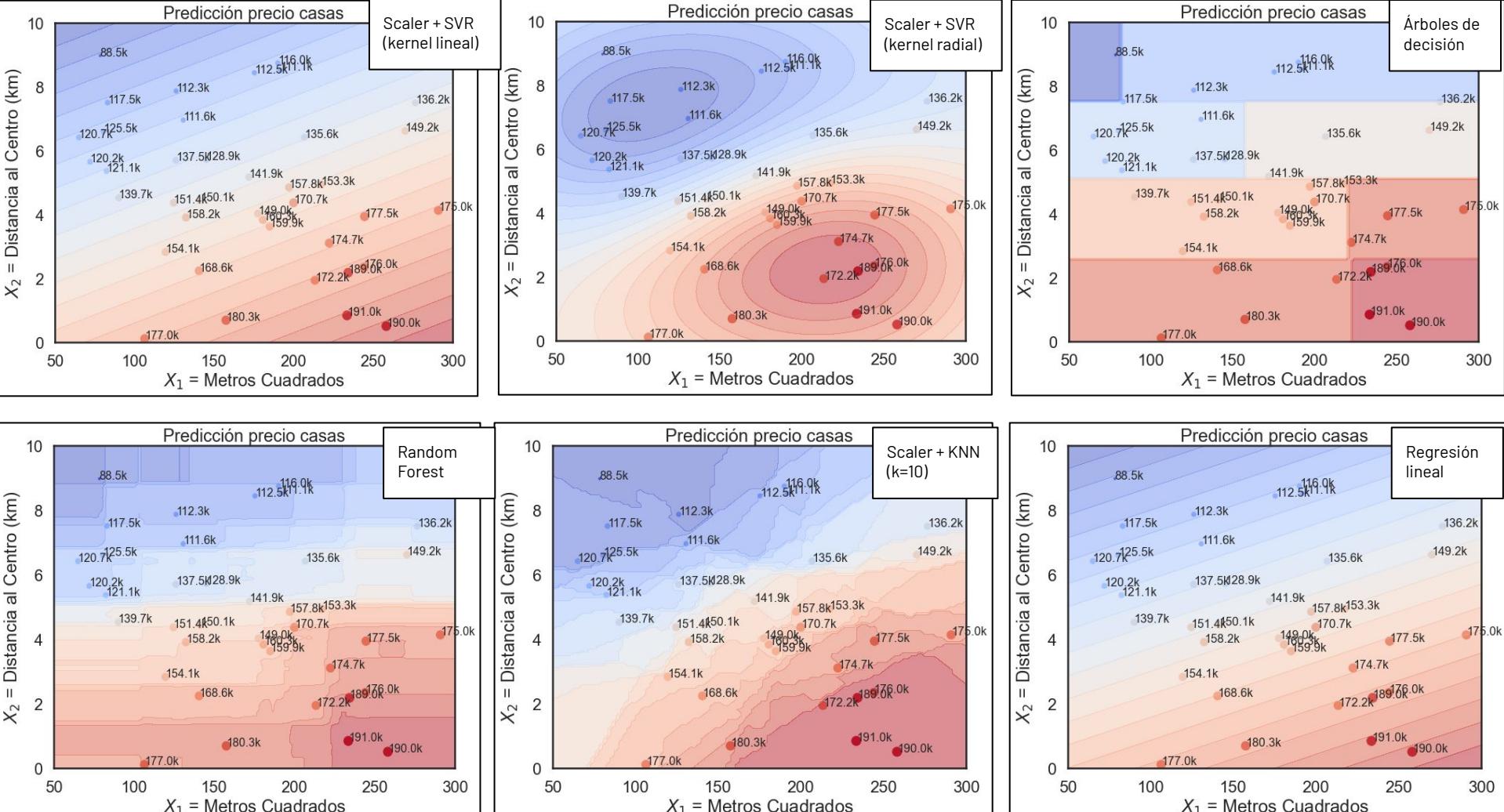
Vector de +1 y -1 por cada w_i

Resumen

Resumen

Vimos:

- que regresión es **similar a clasificación** en donde las **instancias tienen valores numéricos asociados** en vez de clases.
- cómo **adaptar algoritmos ya conocidos** a esta nueva configuración (KNN, árboles)
- cómo se puede pensar a la **regresión lineal** desde un punto de vista de sesgo inductivo.
- que entrenar un modelo en regresión es ajustar los pesos asociados w de manera de minimizar alguna **función de costo**.
- **Descenso por el gradiente** como algoritmo independiente al de regresión
- Descenso por el gradiente con **Mini Batches**
- **Regularización** y su utilización para el filtrado de atributos.



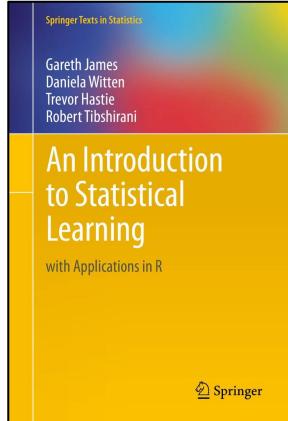
TAREA

- Sección 4.3 del **Deep Learning** (saltar todo lo que quieran de 4.3.1)
- Sección 3.5 del **ISLR** (Comparison of Linear Regression with K-Nearest Neighbors).
- Sección 6.2 (Shrinkage Methods) del **ISLR** (hasta la parte de “A simple Special Case...” p. 247).
- Completar el Notebook y el Cuestionario.

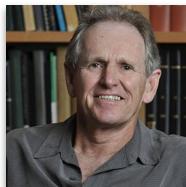
Completamente opcional:

- Tanto como quieran del capítulo 3 del **ISLR**: Linear Regression
- Sección 8.3 del **Deep Learning** (Optimización: Basic Algorithms)
- Investigar el tema “Subgradient Descent for Lasso”

ISLR



Gareth James



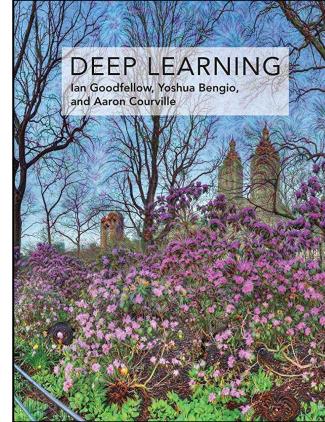
Trevor Hastie

Daniela Witten



Robert Tibshirani

Deep Learning



Ian Goodfellow



Yoshua Bengio



Aaron Courville