

Introducción a la Inteligencia Artificial

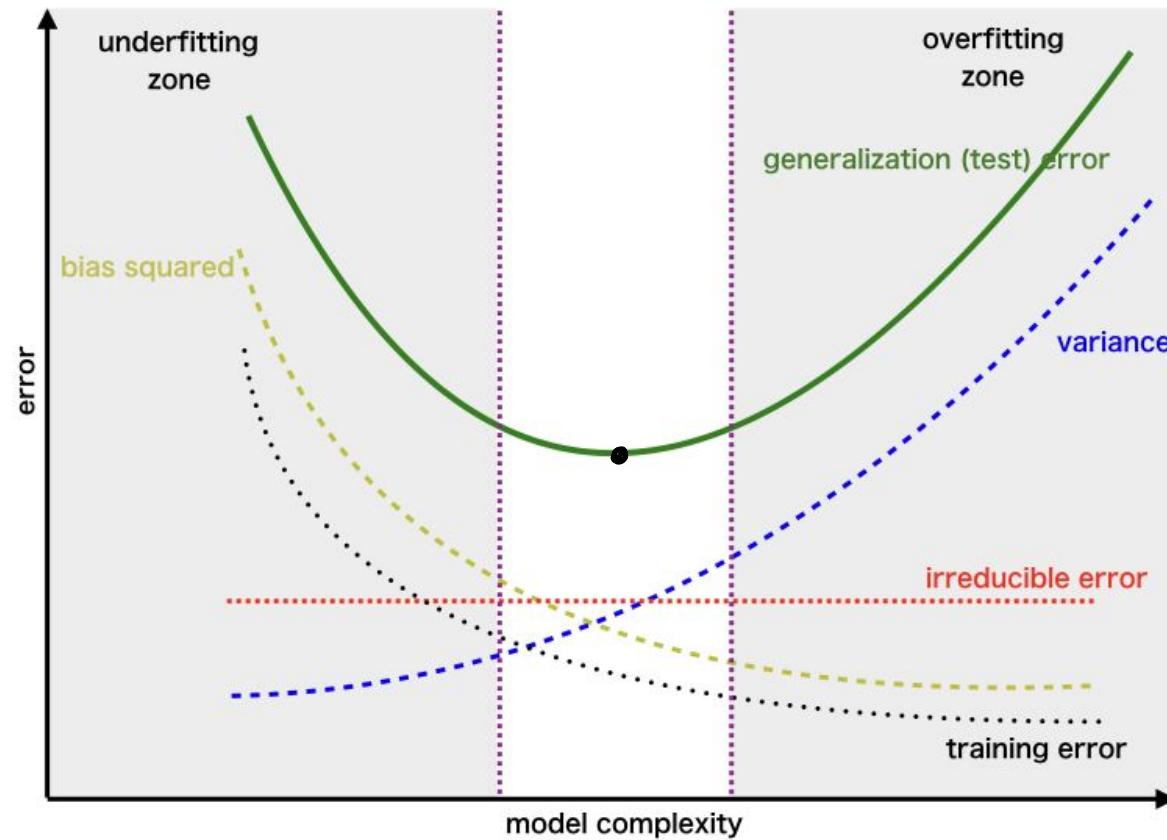
Clase 4

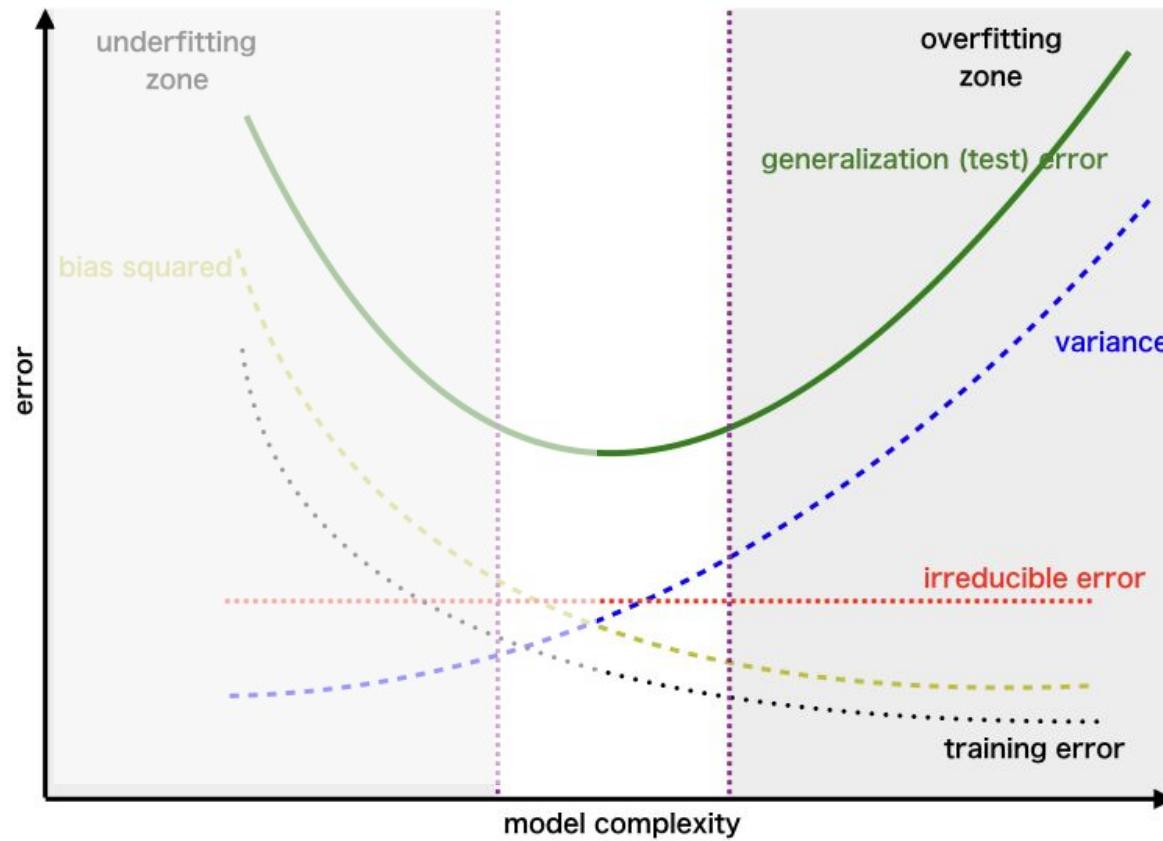


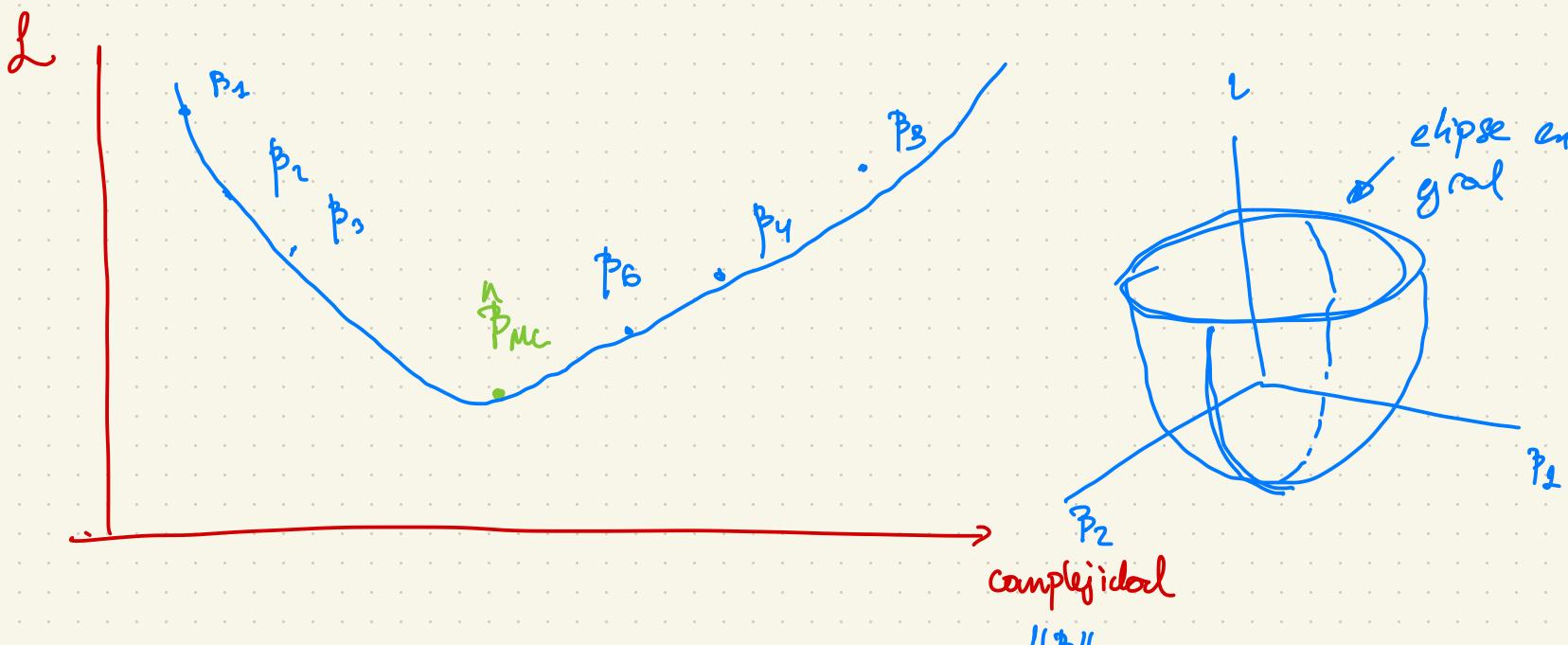
Clase 4

0. Riesgo Empírico

1. Regularización
 - a. Caso general
 - b. Ridge
 - c. Lasso
2. Gradient descent
 - a. GD
 - b. GD Estocástico
 - c. GD Mini-Batch
3. Entrenamiento de modelos
 - a. Selección de modelos
 - b. Cross-Validation

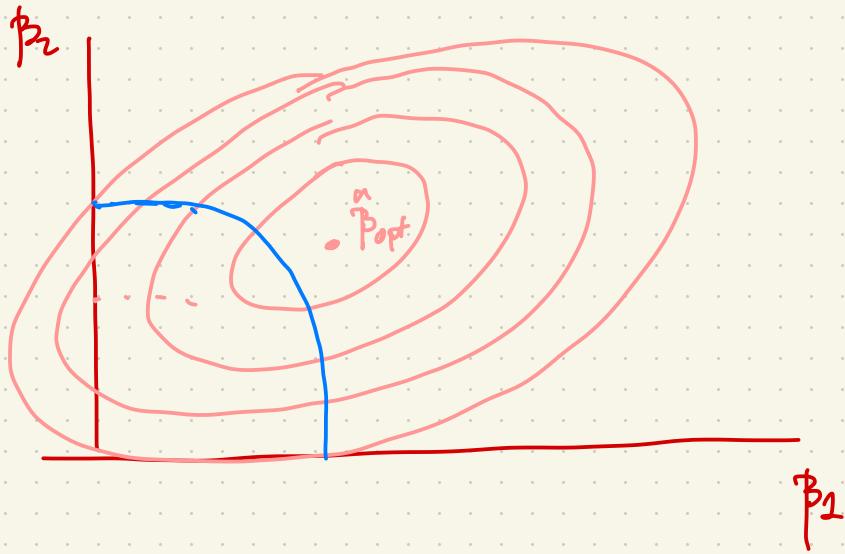




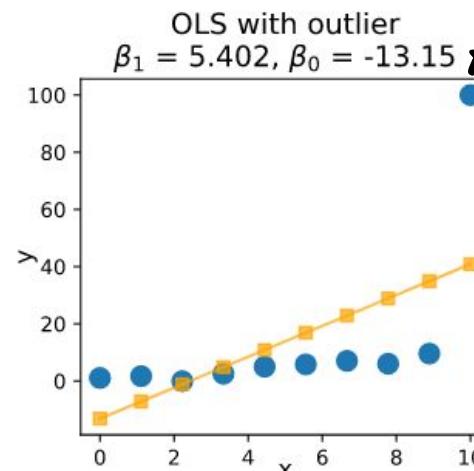
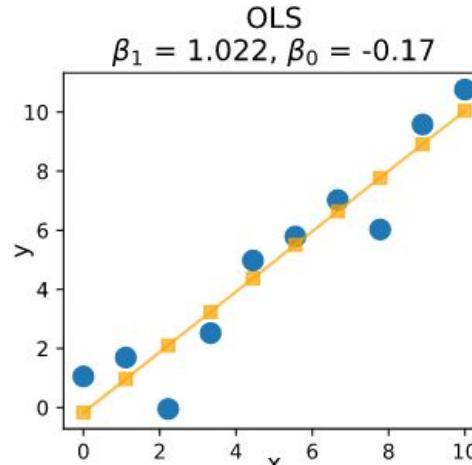


calculos $\hat{\beta}_{ML}$ obtengo un punto

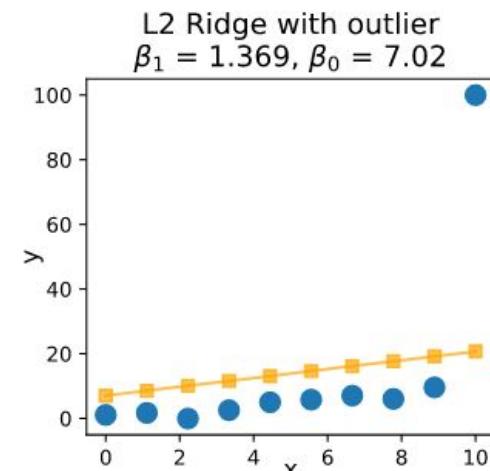
si yo planteo $\beta_i \in [-1, 1]$ pruebo con 100 β 's en ese intervalo



Regularización - Motivación



outliers
 peso(s)
 ó
 high leverage
 points

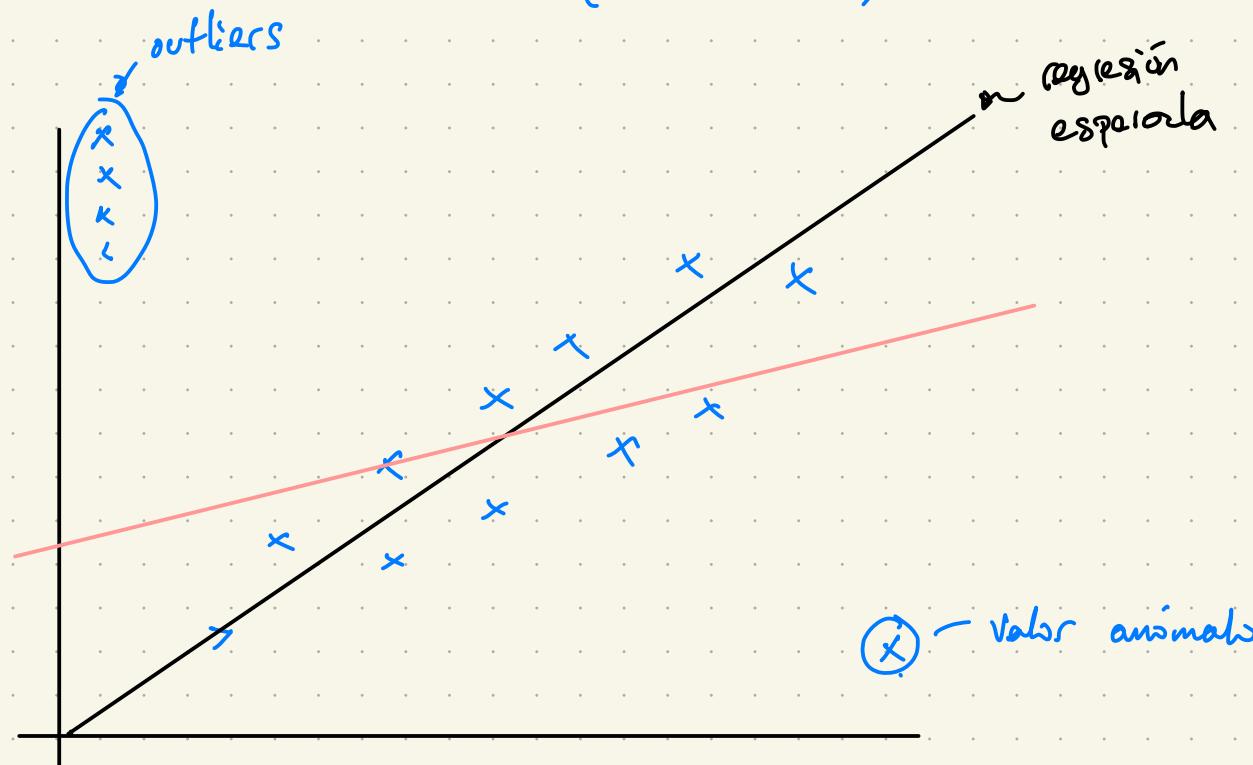


regularización: analiza $\hat{\beta}$'s. \rightarrow mejora* por restricción

Riesgo Empírico: analiza la pericia. \rightarrow mejora* por construcción

Riesgo Empírico

$$R(f) = \mathbb{E}(\underbrace{L(f, \hat{f})}_{\text{función de pérdidas}}) \quad \rightsquigarrow L(f, \hat{f}) = (f - \hat{f})^2 \quad \text{fn. de pérdida L2}$$



$$L' = \begin{cases} (f - \hat{f})^2 & \text{si } |f - \hat{f}| \leq c \\ \alpha & \text{o.w.} \end{cases}$$

¿Cómo podemos mejorar mi reg. lineal?

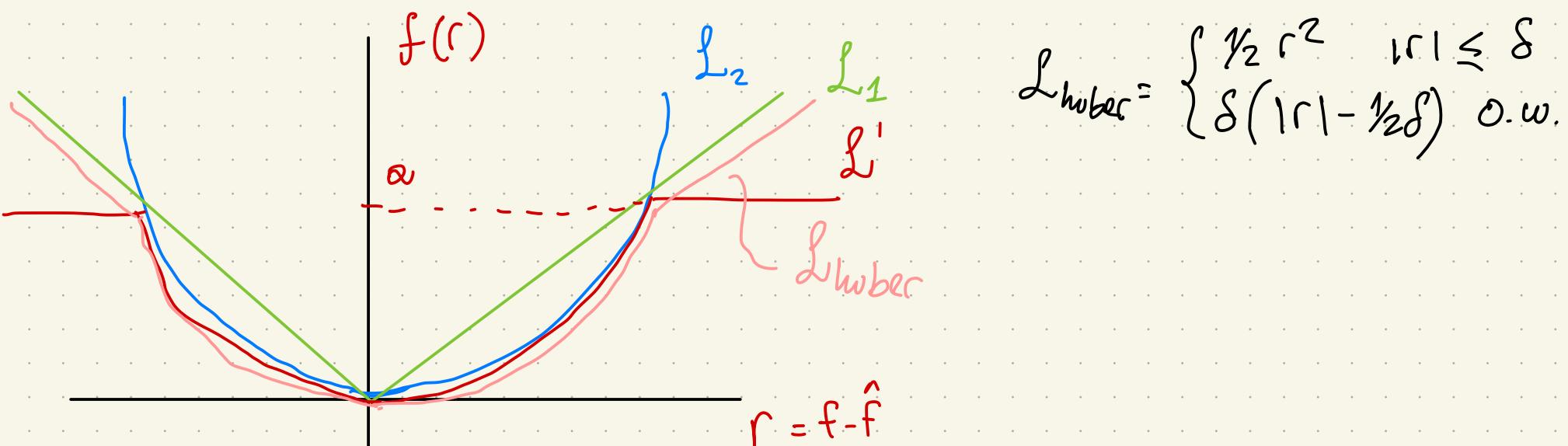
1. Cambiar L

$$- L_1 = |f - \hat{f}|$$

$$- L_2 = (f - \hat{f})^2$$

$$- L_{II} = \sum I_{|f - \hat{f}| \leq c}$$

- L_{huber}
 - L_{Tukey}
- Estimadores robustos de la pérdida (multiparamétricos)



$$L_{\text{tukey}} = \begin{cases} c^2/\alpha \left(1 - \left[1 - (\lvert r \rvert/c)^2 \right]^3 \right) & \text{para } \lvert r \rvert \leq c \\ c^2/\alpha & \text{o.w.} \end{cases}$$

'huber regressor' en sklearn

robust regressor en stat models.

nosotros siempre buscamos minimizar R de la forma más sencilla, a veces cambiar L no es barato o peor no es útil y probaremos regularizar

2. Regularización.

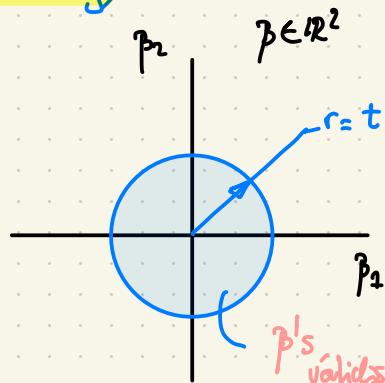
Con reg. buscamos min R al mismo tiempo que restringimos (limitamos) el comp. de los parámetros (**parameter shrinking**).

partimos de

$$\left\{ \begin{array}{l} \hat{y} = \beta_0 + \sum_i \beta_i x_i \\ \hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_i y_i - \sum_j \beta_j x_{ij} \right)^2 \\ \frac{1}{2} \sum_j \beta_j^2 \leq t \quad (\|\beta\|^2 \leq t) \end{array} \right.$$

$$q=2 \rightarrow \hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_i y_i - \sum_j \beta_j x_{ij} \right)^2 - \lambda \sum_j \beta_j^2$$

parámetro de complejidad



Espacio de parámetros

Termino de regularización
(Weight decay)

Expectation - maximization
Algorithm (E-M)

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\}^2$$

Observado - Predicción

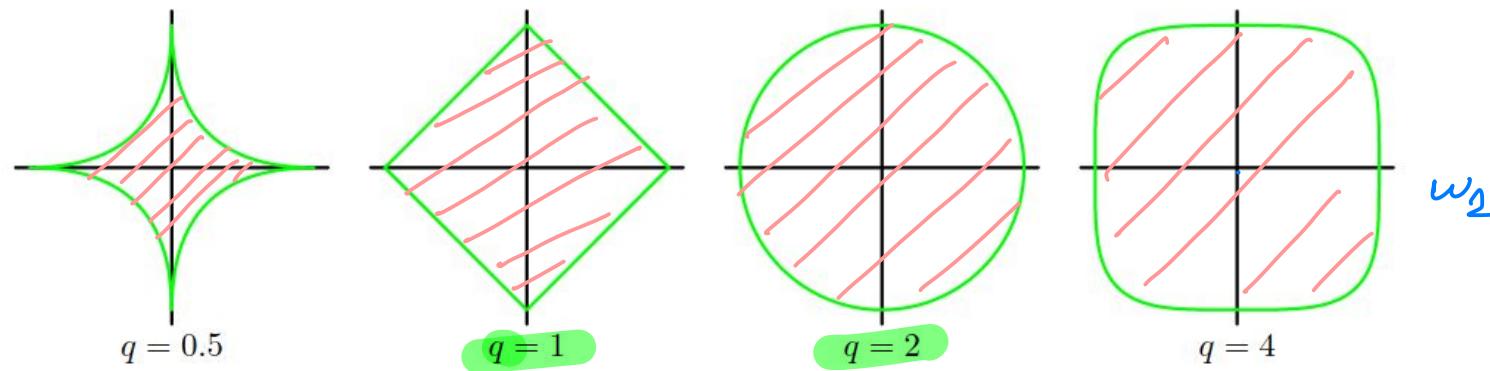


w está “libre”

en este caso $\bar{\mathbf{w}} = \bar{\mathbf{p}}$

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \boxed{\frac{\lambda}{2} \sum_{j=1}^M |w_j|^q}$$

Término de regularización “weight decay” → w afecta la pérdida



Valores válidos
de $\|\mathbf{w}\|_q$

Lasso
penalidad L_1

Ridge
pen L_2

(w_1, w_2)

$$\mathbf{w} = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}$$

Maximum A Posteriori como regularización

Distribución “a priori” de los parámetros \longrightarrow Observar data \longrightarrow Actualizar distribución (Posterior)

$$p(w) \sim D(\theta)$$

$$\dagger(w)$$

$$\dagger(\beta)$$

$$(\mathcal{X}, \mathcal{Y})$$

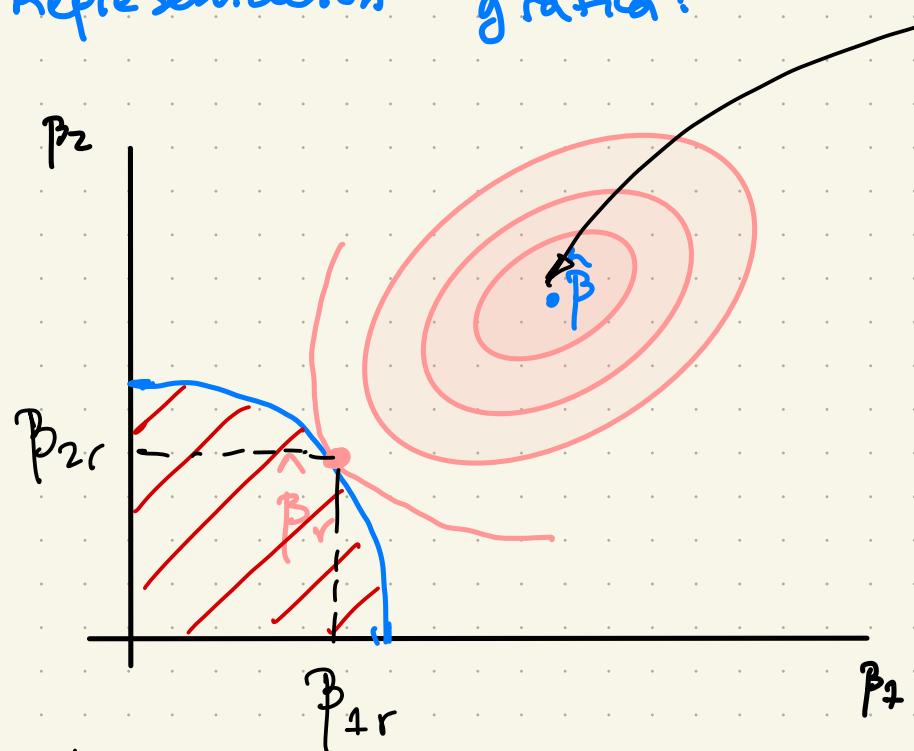
$$p(w|\mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y}|\mathcal{X}, w)p(w)}{p(\mathcal{Y}|\mathcal{X})}$$

$$w_{map} = (\Phi^T \Phi + \frac{\sigma^2}{b^2} I)^{-1} \Phi^T y$$

Gaussian prior con varianza b2

Representación gráfica:

$\hat{\beta}$: Es el mejor $\hat{\beta}$ posible

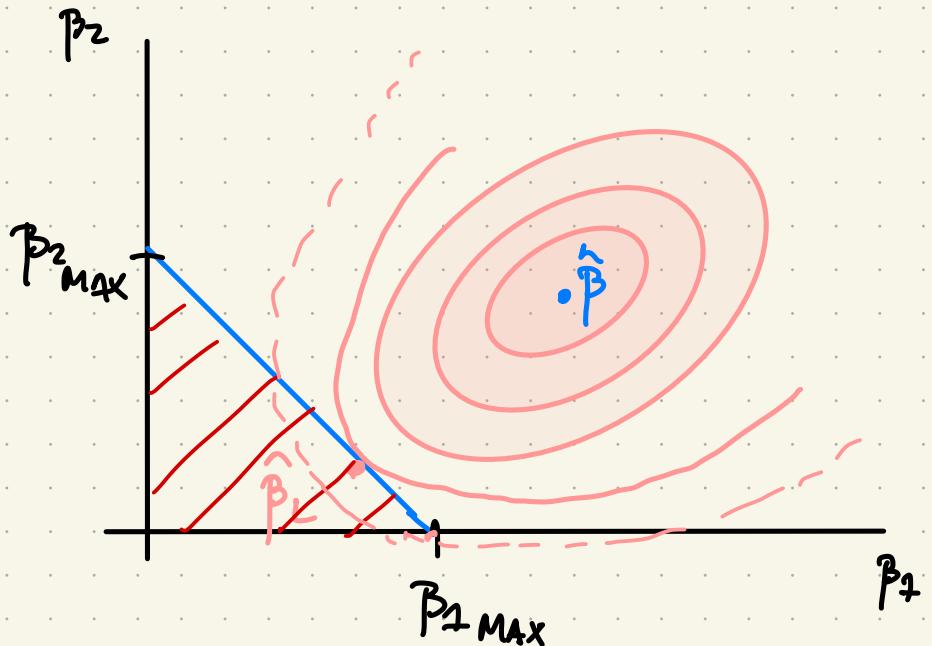


β 's vehículos

Ridge

porque enrobustece
mi sist.

$$-\lambda \sum_j |\beta_j|^2$$



Lasso

porque hace un
proceso de selección
de variables

Recordemos que $\hat{\beta}$ en general tiene curvas de nivel elípticas (por construcción del estimador).

Pero, regularizar viene a costo de alejarnos del óptimo.

¿Cómo funciona?

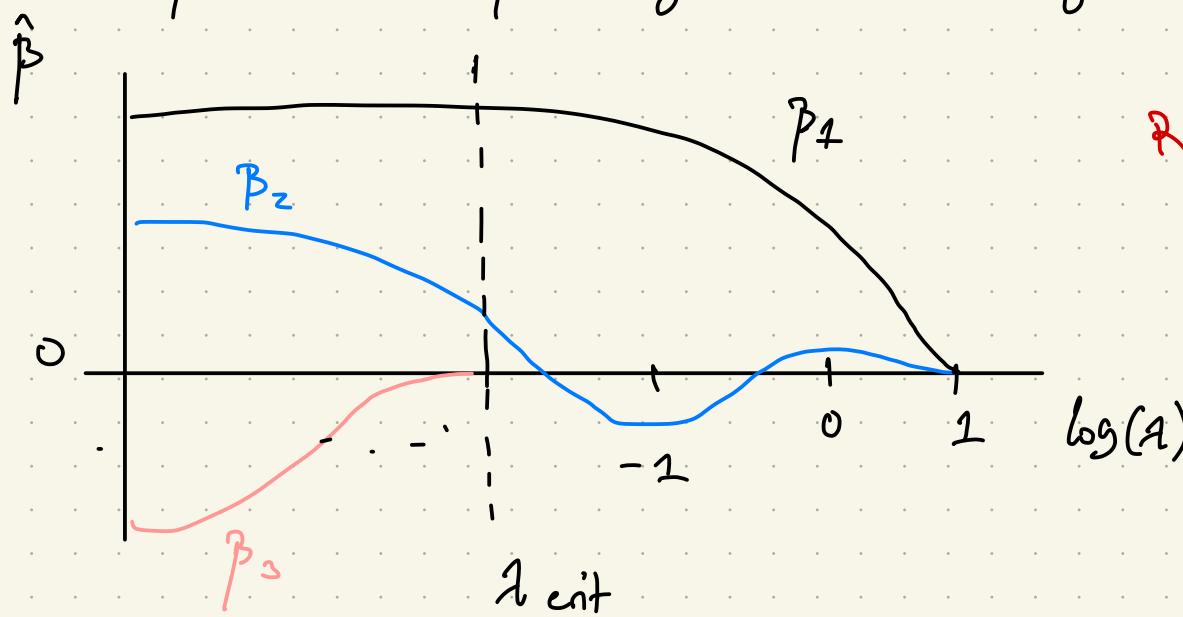
1. Elegir q (Lasso: 1, Ridge: 2)

2. Voy a elegir un vector de λ 's "apropiados" (λ 's no \neq penalidad)

3. Optimizo para β_i ms $RSS(\lambda; q) = \|y - X\beta\|^q + \lambda \|\beta\|_q$

4. Calcular las métricas (Error de representación, bondad, algún criterio externo).

5. Comparamos y elegimos el mejor:



Regresión
Lasso típica

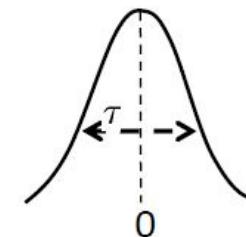
Maximum A Posteriori como regularización - Ridge (L2)

$$\hat{\beta}_{\text{MAP}} = \arg \max_{\beta} \underbrace{\log p(\{Y_i\}_{i=1}^n | \beta, \sigma^2, \{X_i\}_{i=1}^n)}_{\text{Conditional log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

I) Gaussian Prior

$$\beta \sim \mathcal{N}(0, \tau^2 \mathbf{I})$$

$$p(\beta) \propto e^{-\beta^T \beta / 2\tau^2}$$



$$\hat{\beta}_{\text{MAP}} = \arg \min_{\beta} \sum_{i=1}^n (Y_i - X_i \beta)^2 + \lambda \|\beta\|_2^2$$

↓
constant(σ^2, τ^2)

Ridge Regression

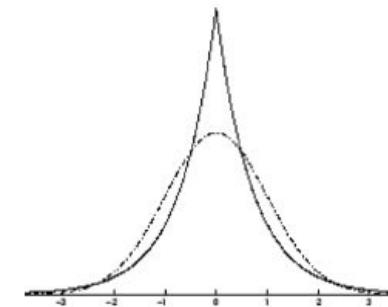
$$\hat{\beta}_{\text{MAP}} = (\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^\top \mathbf{Y}$$

Maximum A Posteriori como regularización - LASSO (L1)

$$\hat{\beta}_{\text{MAP}} = \arg \max_{\beta} \underbrace{\log p(\{Y_i\}_{i=1}^n | \beta, \sigma^2, \{X_i\}_{i=1}^n)}_{\text{Conditional log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

II) Laplace Prior

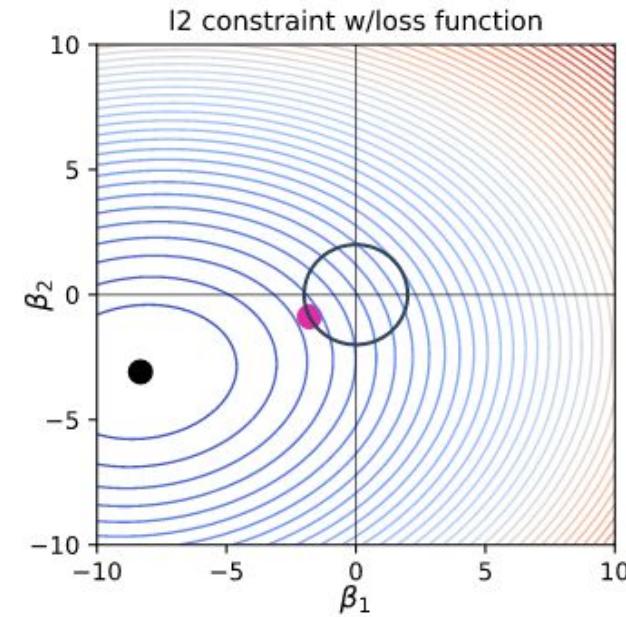
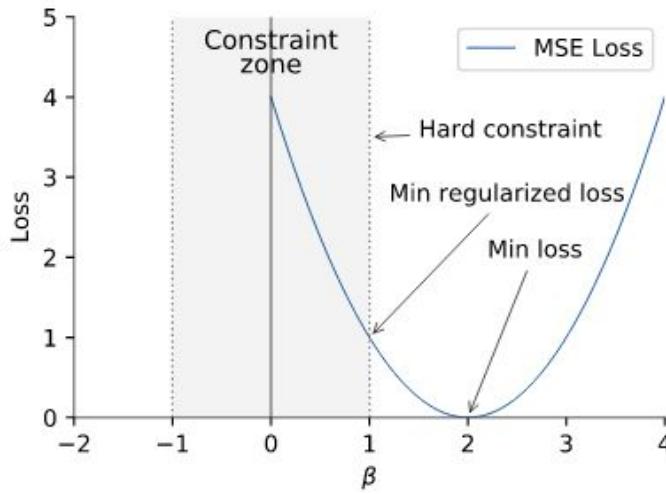
$$\beta_i \stackrel{iid}{\sim} \text{Laplace}(0, t) \quad p(\beta_i) \propto e^{-|\beta_i|/t}$$



$$\hat{\beta}_{\text{MAP}} = \arg \min_{\beta} \sum_{i=1}^n (Y_i - X_i \beta)^2 + \lambda \|\beta\|_1$$

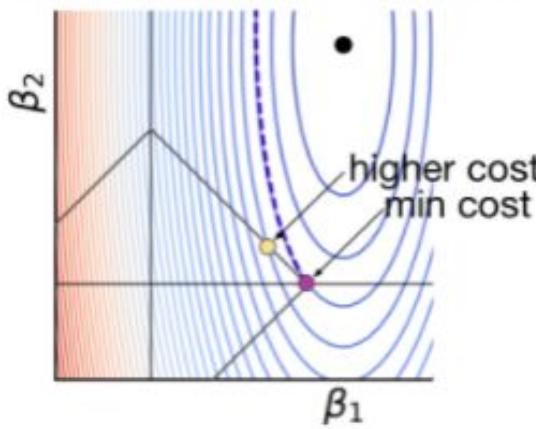
↓
constant(σ^2, t)

Regularización

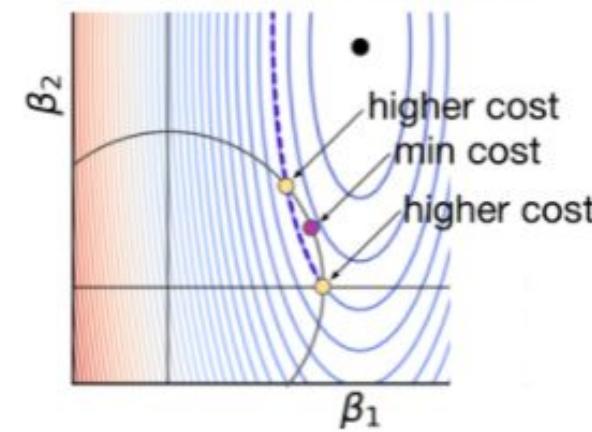


Regularización

(a) L1 Constraint Diamond



(b) L2 Constraint Circle



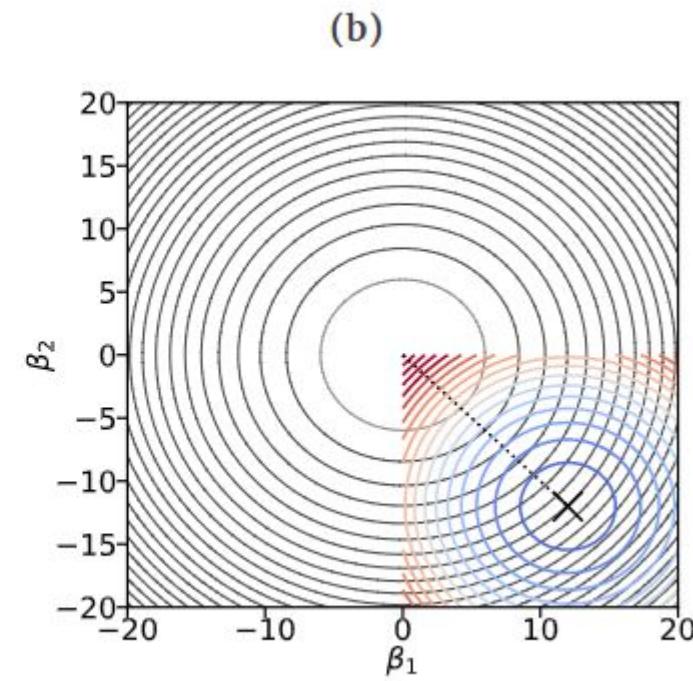
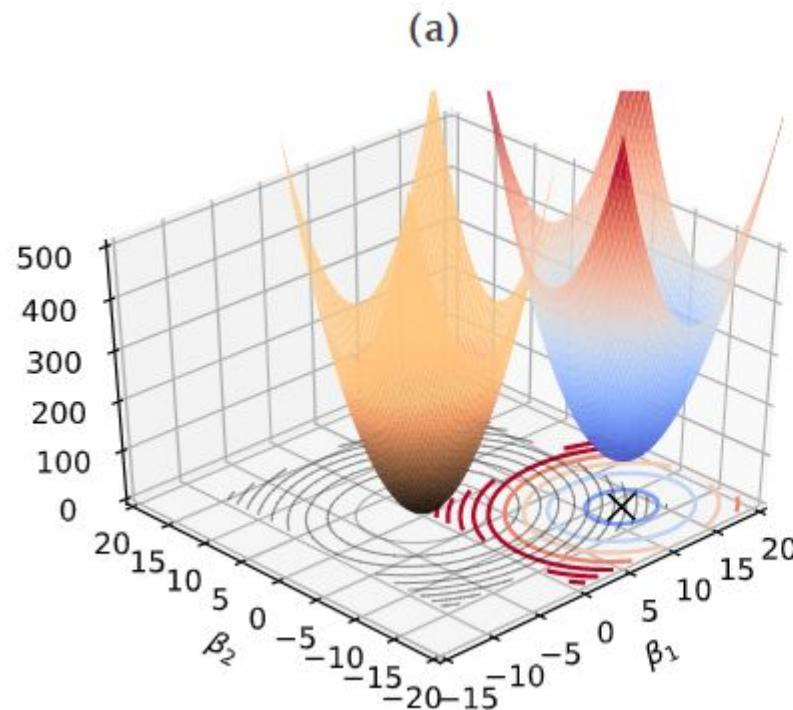
ElasticNet *2 técnicas de complejidad*

¿Qué β se reduce más?

$$(\alpha\lambda\|\beta\|_1 + \frac{1}{2}(1-\alpha)\|\beta\|_2^2)$$

α parámetro de elasticnet

Regularización



Entrenamiento de modelos - Cross-Validation

Cross-Validation

parametros n_folds folds en general son subdivisiones random del dataset



Modelo \rightarrow tiene hiperáram. $\bar{\varphi} = (\varphi_0, \varphi_1, \dots, \varphi_n)$

\rightarrow es iterativo \rightarrow tiene asociado un solver y un conjunto de parámetros de conv. ($n\text{-it}$, tol , ...)

Datos \rightarrow separar en train 80%, test 20%.

en el caso de CV. fijo el nr de particiones (folds) K y separo train en K folds.

F ₁		F ₂		F ₃		F ₄		F ₅
----------------	--	----------------	--	----------------	--	----------------	--	----------------

. planteamos el espacio de hp. de busqueda:

$\left\{ \begin{array}{l} \varphi_0 : [0,1] \\ \varphi_1 : [1,5] \\ \vdots \\ \varphi_n : [a,b] \end{array} \right.$ en gen la cant de valors per hp. no suele ser igual. vamos a generar totes las permutaciones possibles de modelitos. (Grid Search)

$$m_1 : \varphi_0 = 0, \varphi_1 = 1, \dots, \varphi_n = a$$

$$m_2 : \varphi_0 = 1, \varphi_1 = 1, \dots, \varphi_n = a$$

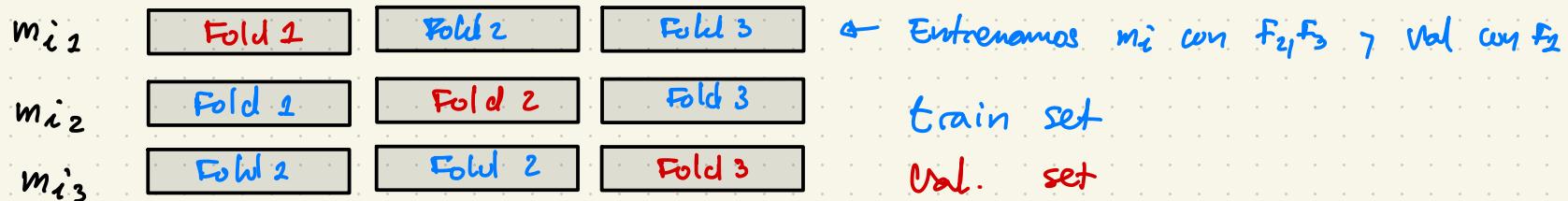
;

con Cross-val voy a probar cada m_i y voy a analizar su performance.

- K-fold cross validation \rightarrow tomo K folds y entreno mult. mi ①
- hold-out: fijo un fold que se usa para validar (lo mismo q' train-test-val)
- leave-one-out: tomo K folds con $K = \text{len}(\text{data}) - 1$ y hago ②

② ¿cómo func. K-fold w?

para el ej. $K=3$



métrica (m_{i_1}) puede no ser igual a métrica(m_{i_2}) y así con todo fold.

una vez calculados m_{ij} para un doblet i , calculo la métrica promedio:

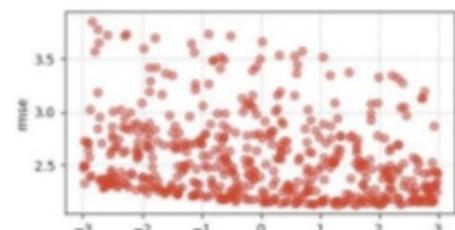
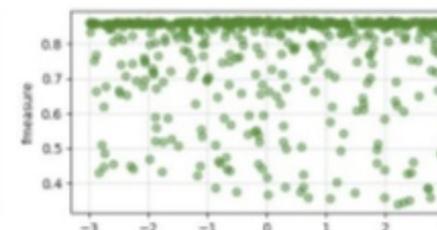
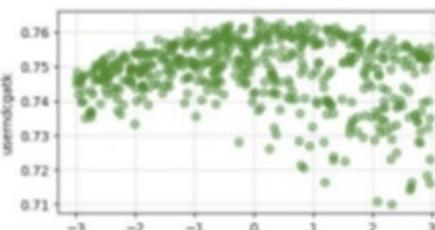
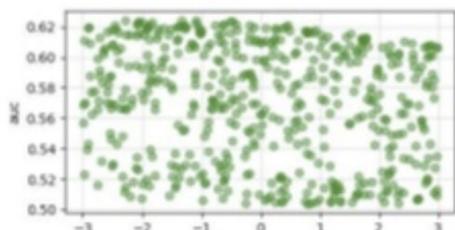
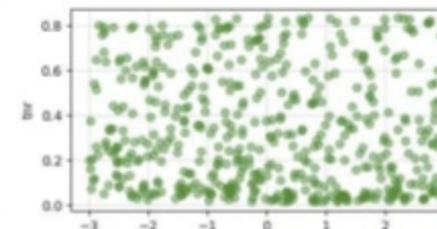
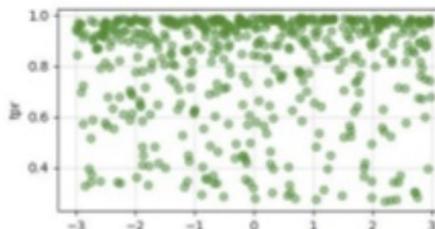
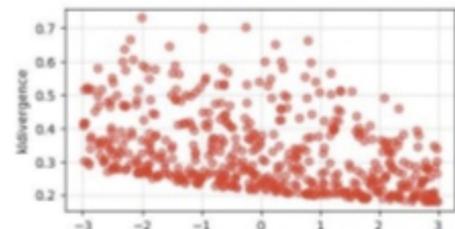
$$\text{metrica}(m_i) = E(\text{metrica}(m_{ij})) \pm \text{Var}(\text{metrica}(m_{ij}))$$

Al final del proceso tengo una tabla:

	metrica prom	var	ranking
m_0	5	0,2	$n-2$
m_1	6	0,3	$n-1$
:	:		
m_n	2	0,5	1

el ranking se basa en que modelos performan mejor.

Selección de los hiper parámetros



- Metric to **maximize**
- Metric to **minimize**

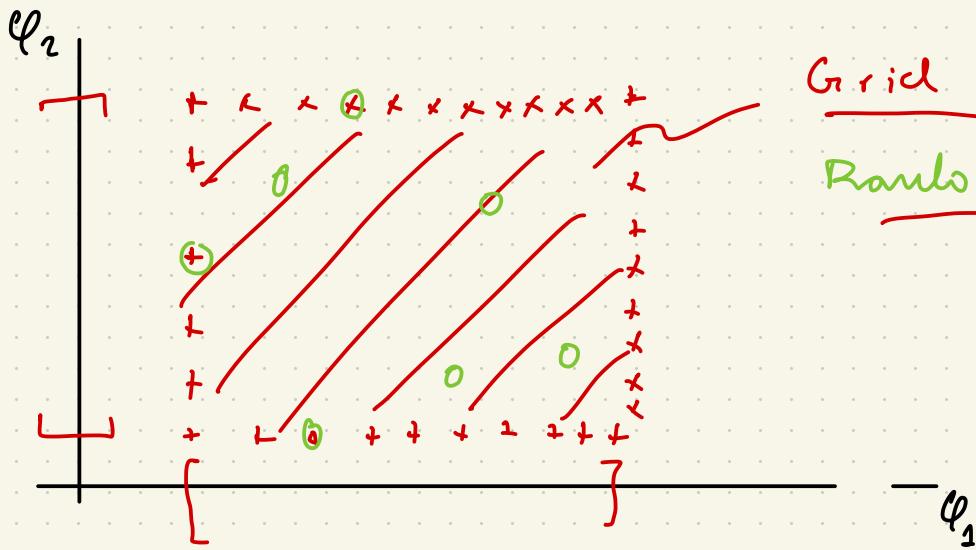
método de
fuerza bruta



Grid Search

Random Search

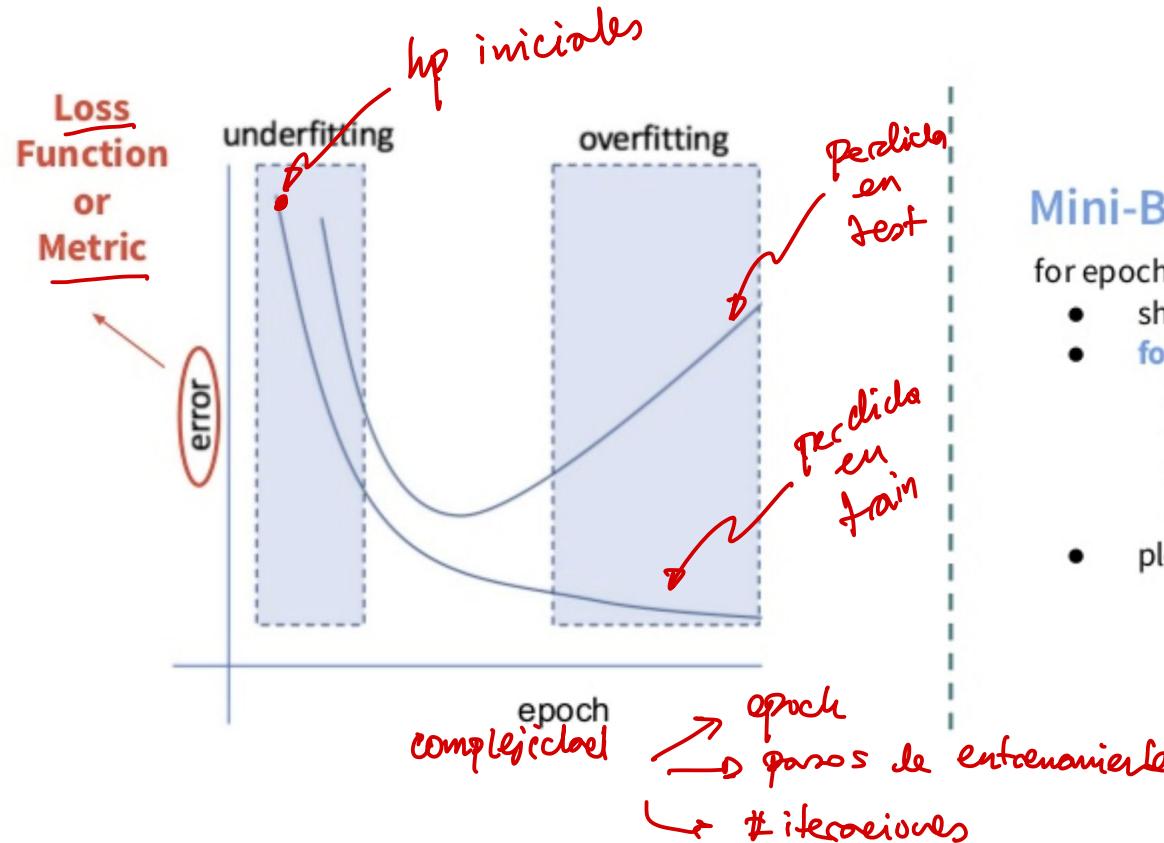
Ramdomized Grid search.



Grid search

Randomized Grid Search

Entrenamiento numérico del modelo seleccionado - Obtención de parámetros



Mini-Batch Gradient Descent

- ```
for epoch in n_epochs:
 • shuffle the batches
 • for batch in n_batches:
 ○ compute the predictions for the batch
 ○ compute the error for the batch
 ○ compute the gradient for the batch
 ○ update the parameters of the model
 • plot error vs epoch
```

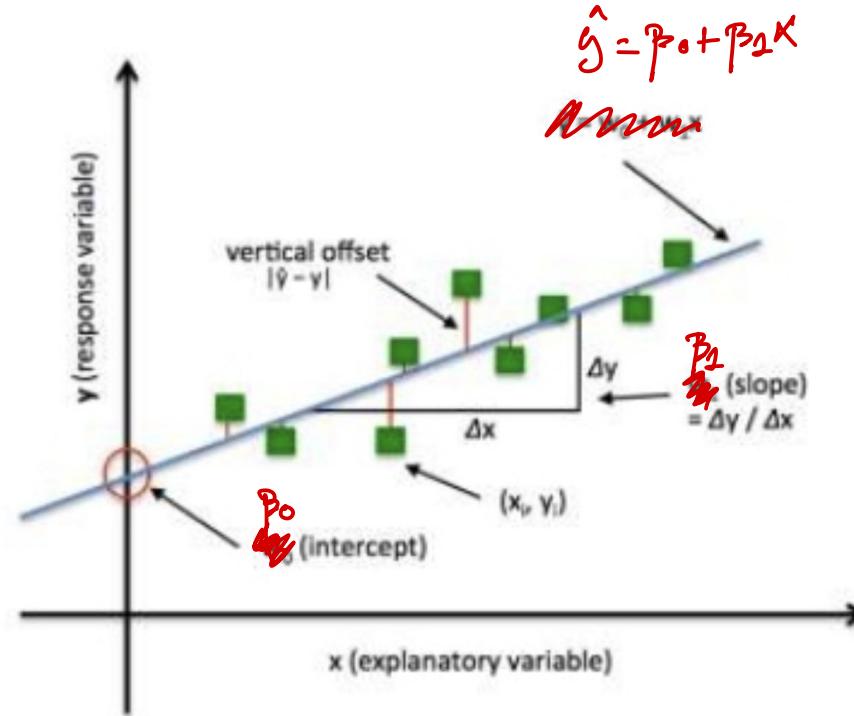
## Gradiente Descendente

## Implementación de Gradiente Descendente

Solucion analitica

$$\min \|Y - X\beta\|_2^2$$

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$



# Gradient Descent

## Implementación de Gradiente Descendente

GD ( $\alpha, w_0$ )

Solución numérica

$$\min_{\beta} \|Y - X\beta\|_2^2 \Rightarrow \min_{\beta} \sum_i (y_i - X_i \cdot \beta)^2$$



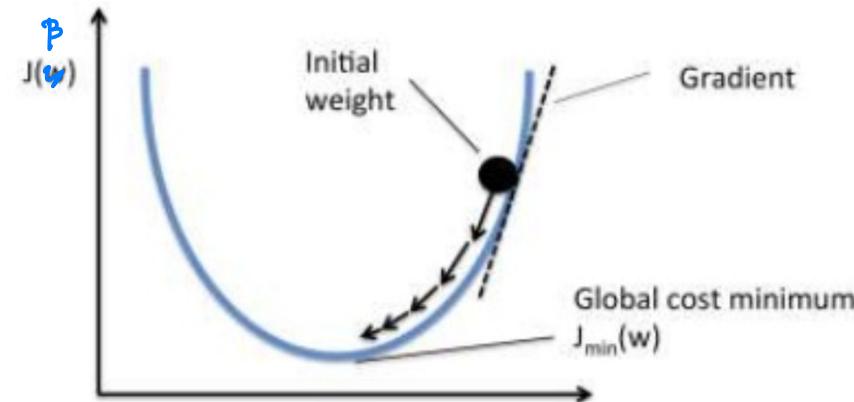
$$\bar{\beta}_1 \rightarrow \bar{\beta}_0 \rightarrow \text{cole inicial}$$
$$\bar{\beta}_0 = (\text{coleom}(), \text{coleom}())$$

$$W_{n+1} \leftarrow W_n - \alpha \nabla \left( \sum_i (y_i - X_i \cdot W)^2 \right)$$

learning rate

$$\alpha \in \mathbb{R}^+$$

$$\bar{\beta}_2 = \bar{\beta}_0 - \alpha \nabla(\text{MSE})$$



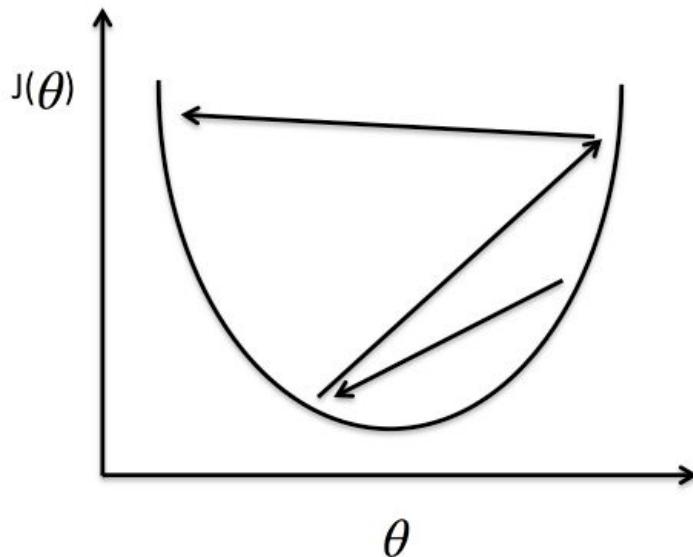
mínimo

# Gradient Descent

## Gradiente Descendente

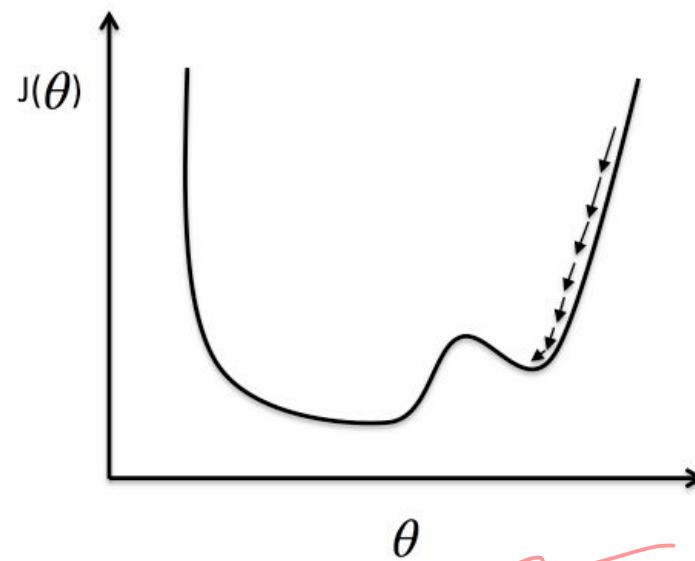
Problema ②

undershooting  $\neq$  Vanishing gradient.



Large learning rate: Overshooting.

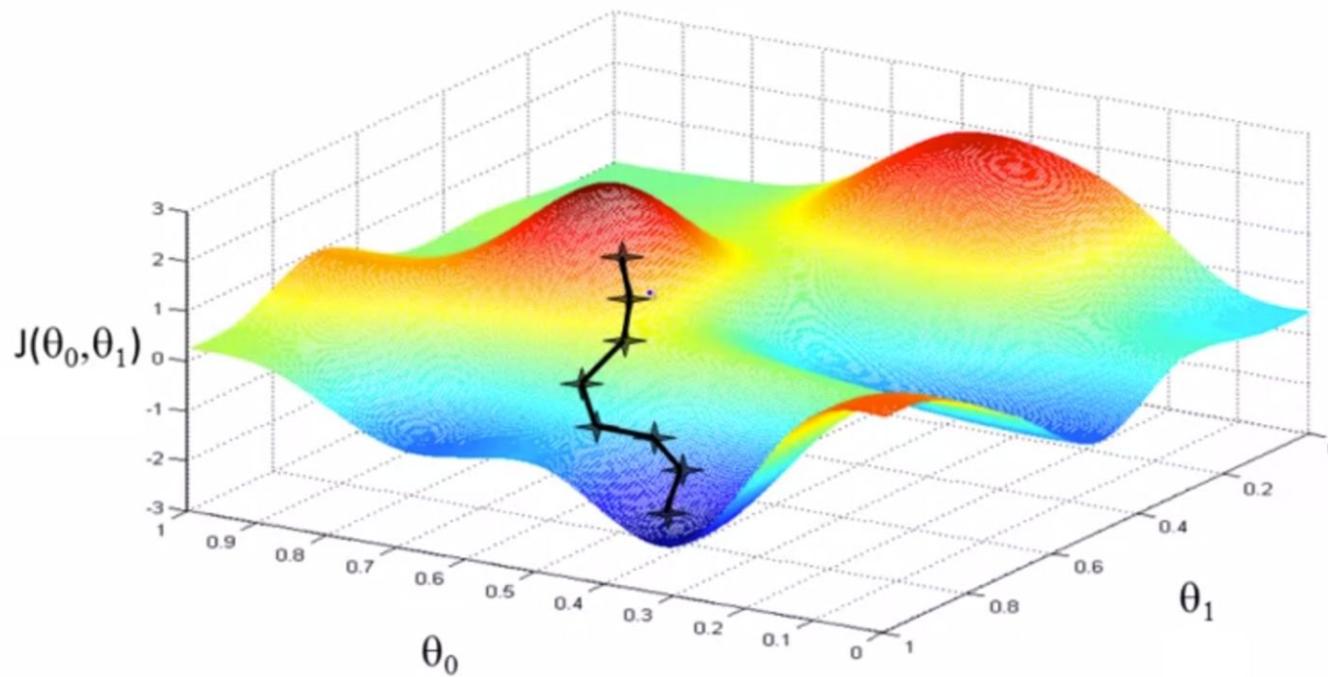
overshooting puede relacionarse con explodding gradients.



Small learning rate: Many iterations until convergence and trapping in local minima.

undershooting

## Gradiente Descendente



Andrew Ng

## Implementación de Gradiente Descendente

Solución numérica

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \left( \sum_i (y_i - X_i W)^2 \right) \\ &= \sum_i \left( \nabla_w (y_i - X_i W)^2 \right) \\ &= \sum_i \left( \nabla_w (y_i - (x_{i1}w_1 + x_{i2}w_2 + \cdots + x_{im}w_m))^2 \right) \\ &= \sum_i \left( -2(y_i - \hat{y}_i) \underbrace{x_{ij}}_{y_i - \hat{y}_i = r_i} \right) \quad \underbrace{\forall j \in (1 \cdots m)}\end{aligned}$$

## Implementación de Gradiente Descendente

Solución numérica

$$\nabla \left( \sum_{\text{all samples}} (y_i - f_W(X_i))^2 \right)$$

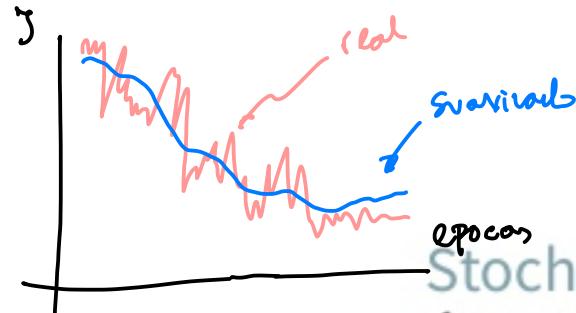
### Gradient Descent algorithm

for epoch in n\_epochs:  
    Alternativa    while e < tol    || n\_epoch < n\_epochs:

- compute the predictions for **all the samples**
- compute the error between truth and predictions
- compute the gradient using **all the samples**
- update the parameters of the model

## Implementación de Gradiente Descendente Estocástico

Solución numérica



$$\nabla ((y_i - f_W(X_i))^2)$$

$i \in K$  → es la cant. total de muestras  
es un subset

Stochastic Gradient Descent algorithm

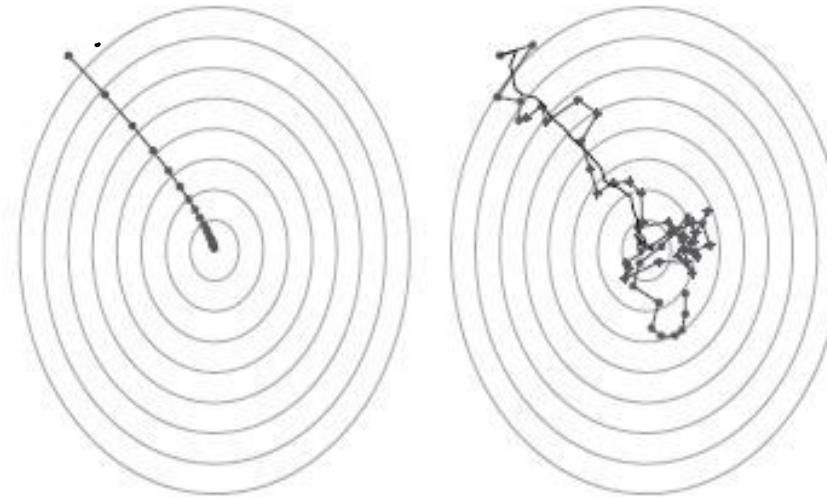
for epoch in n\_epochs:

- shuffle the samples
- for sample in n\_samples:
  - compute the predictions for the sample
  - compute the error between truth and predictions
  - compute the gradient using the sample
  - update the parameters of the model

esto condiciona las muestras.

## Implementación de Gradiente Descendente Estocástico

Solución numérica



## Implementación de Gradiente Descendente Mini-Batch

Solución numérica

$$\nabla \left( \sum_{\text{batch samples}} (y_i - f_W(X_i))^2 \right)$$

|                |  |                |  |                |  |                |  |                |
|----------------|--|----------------|--|----------------|--|----------------|--|----------------|
| B <sub>1</sub> |  | B <sub>2</sub> |  | B <sub>3</sub> |  | B <sub>4</sub> |  | B <sub>5</sub> |
|----------------|--|----------------|--|----------------|--|----------------|--|----------------|

e<sub>1</sub>    B<sub>3</sub> B<sub>3</sub> B<sub>1</sub> B<sub>4</sub> B<sub>2</sub>  
e<sub>2</sub>    B<sub>1</sub> B<sub>4</sub> B<sub>3</sub> B<sub>5</sub> B<sub>2</sub>

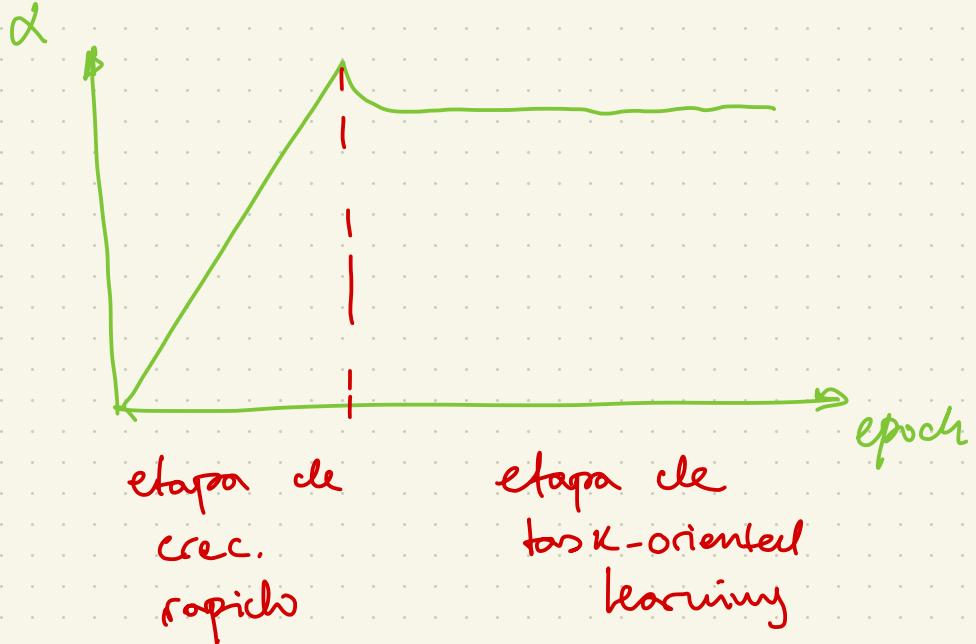
Mini-Batch Gradient Descent algorithm *n\_epoch, n\_batch, batch\_size*  
for epoch in n\_epochs:

- shuffle the batches
- **for batch in n\_batches:**
  - compute the predictions for **the batch**
  - compute the error for the batch
  - compute the gradient for **the batch**
  - update the parameters of the model

$$\sum_i \text{Big}(-2(y_i - \hat{y}_i) x_{ij})$$

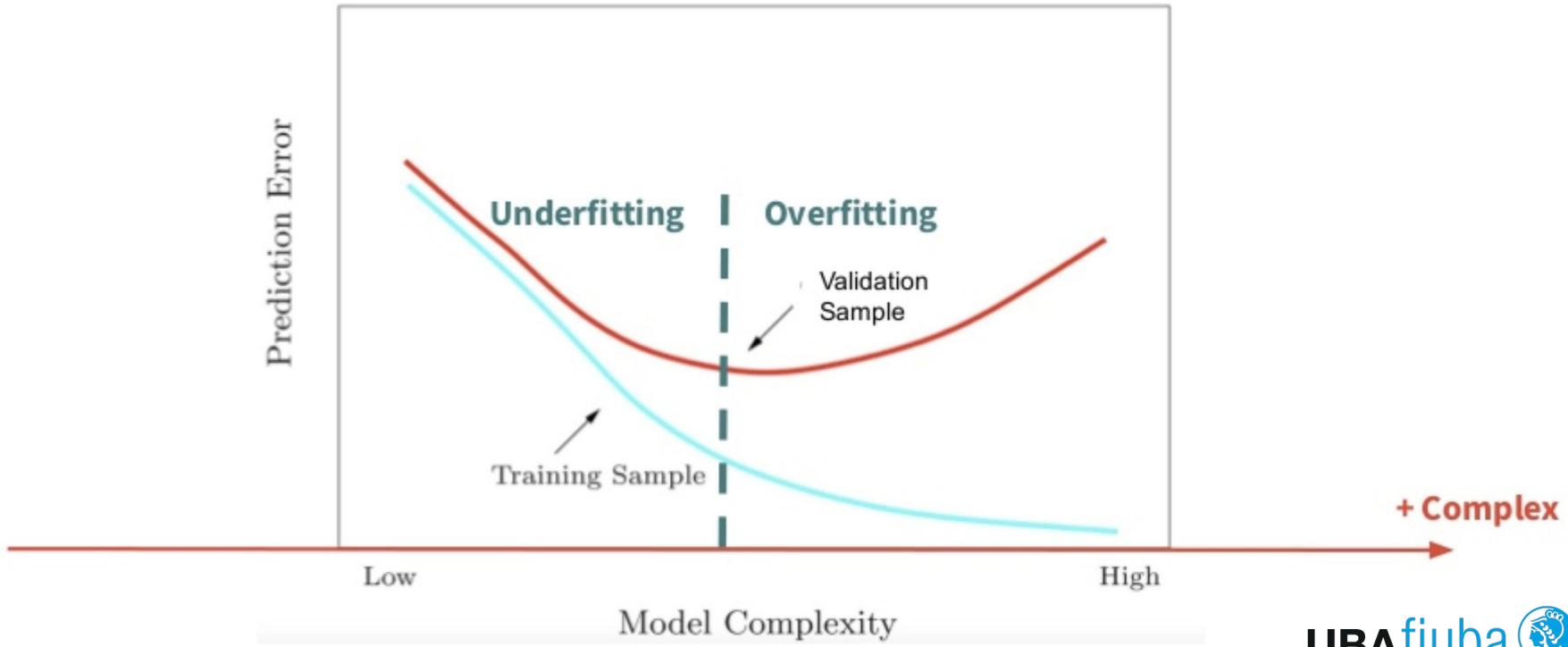
## Comparativa de gradientes

|                           | Gradient Descent                                                     | Stochastic Gradient Descent     | Mini-Batch Gradient Descent                                            |
|---------------------------|----------------------------------------------------------------------|---------------------------------|------------------------------------------------------------------------|
| <b>Gradient</b>           | $\nabla \left( \sum_{\text{all samples}} (y_i - f_W(X_i))^2 \right)$ | $\nabla ((y_i - f_W(X_i))^2)$   | $\nabla \left( \sum_{\text{batch samples}} (y_i - f_W(X_i))^2 \right)$ |
| <b>Speed</b>              | Very Fast (vectorized)                                               | Slow (compute sample by sample) | Fast (vectorized)                                                      |
| <b>Memory</b>             | $O(\text{dataset})$                                                  | $O(1)$                          | $O(\text{batch})$                                                      |
| <b>Convergence</b>        | Needs more epochs                                                    | Needs less epochs               | Middle point between GD and SGD                                        |
| <b>Gradient Stability</b> | Smooth updates in params                                             | Noisy updates in params         | Middle point between GD and SGD                                        |

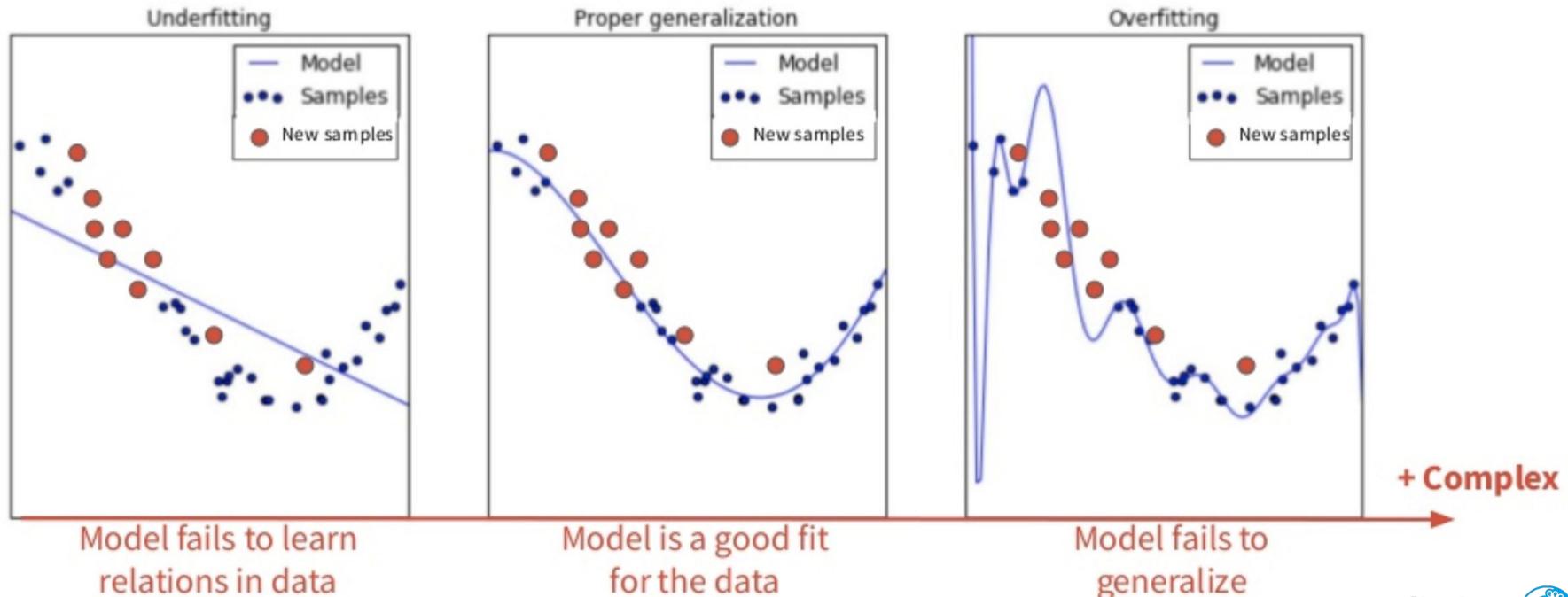


## Selección de modelos

## Selección de modelos



## Selección de modelos



## Bibliografía

- The Elements of Statistical Learning | Trevor Hastie | Springer
- An Introduction to Statistical Learning | Gareth James | Springer
- Deep Learning | Ian Goodfellow | <https://www.deeplearningbook.org/>
- Mathematics for Machine Learning | Deisenroth, Faisal, Ong
- Artificial Intelligence, A Modern Approach | Stuart J. Russell, Peter Norvig
- A visual explanation for regularization of linear models - Terence Parr
- A Complete Tutorial on Ridge and Lasso Regression in Python - Aarshay Jain