Introducción a la Inteligencia Artificial

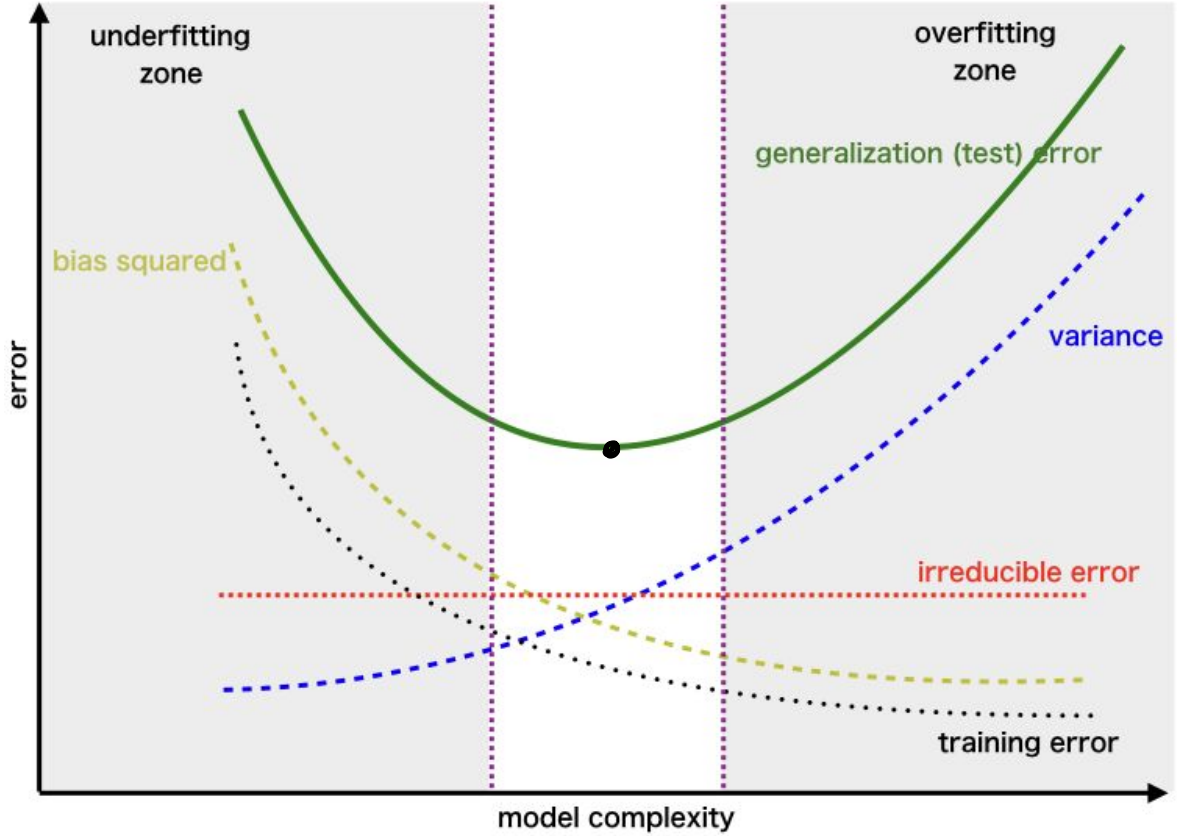
Clase 4

Índice

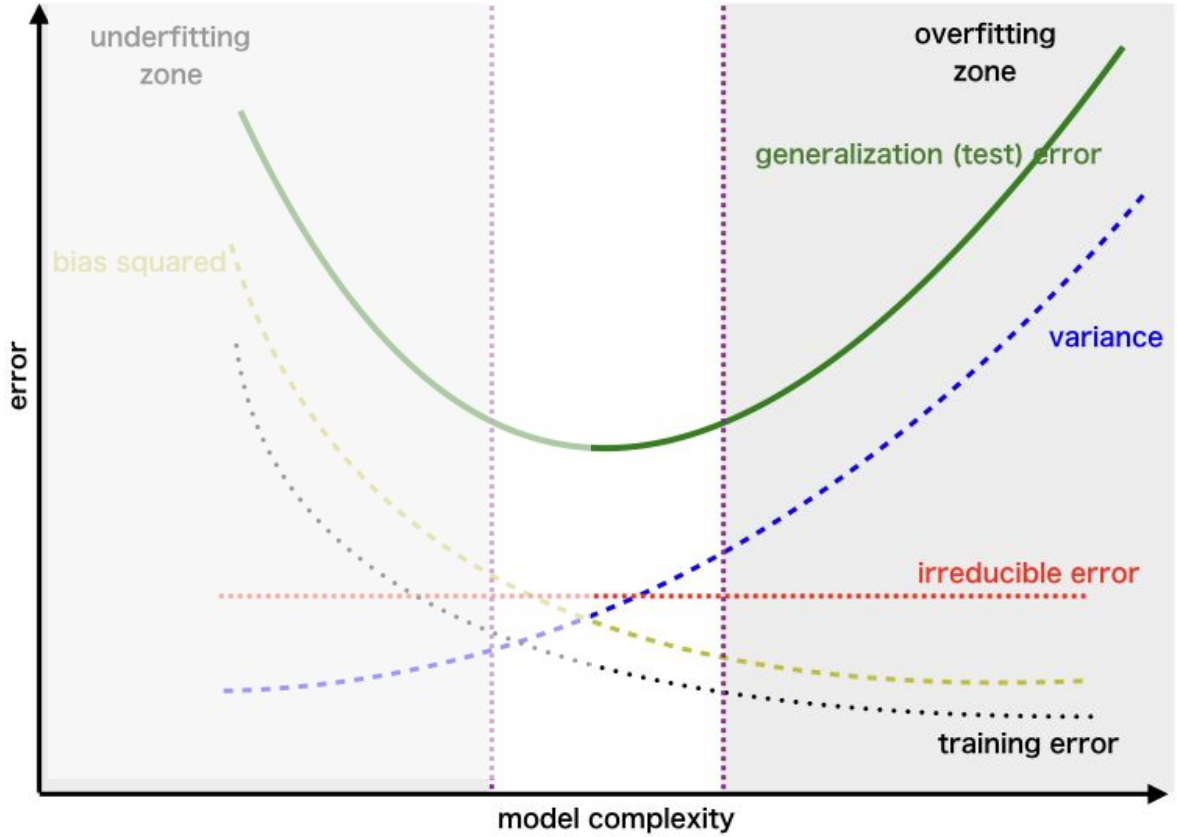
Clase 4

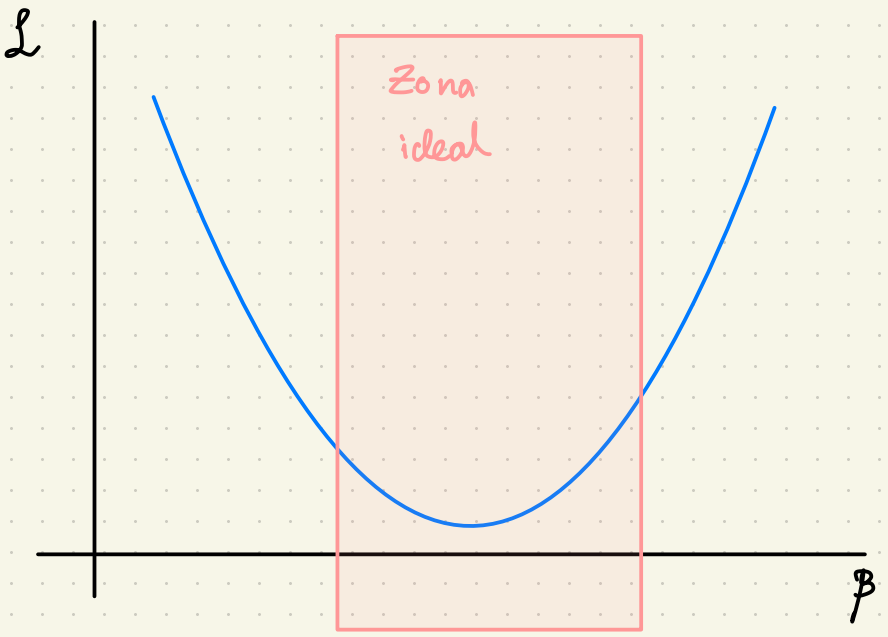
0. Riesgo Empírico
1. Regularización
   a. Caso general
   b. Ridge
   c. Lasso
2. Gradient descent
   a. GD
   b. GD Estocástico
   c. GD Mini-Batch
3. Entrenamiento de modelos
   a. Selección de modelos
   b. Cross-Validation

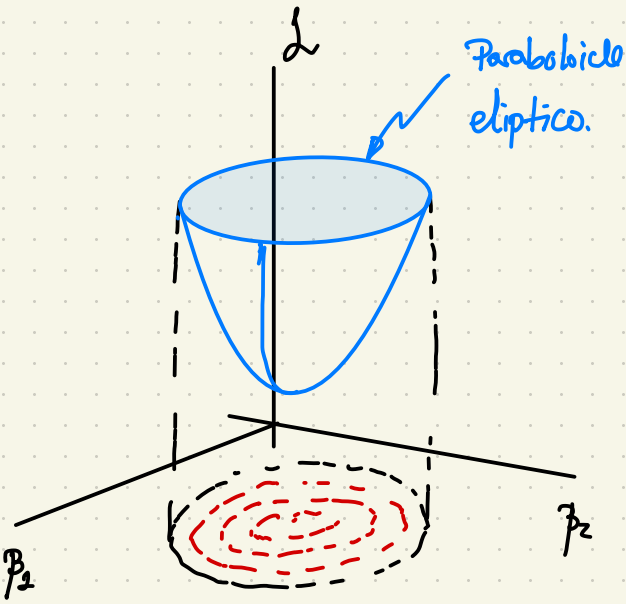
Regularización



Regularización





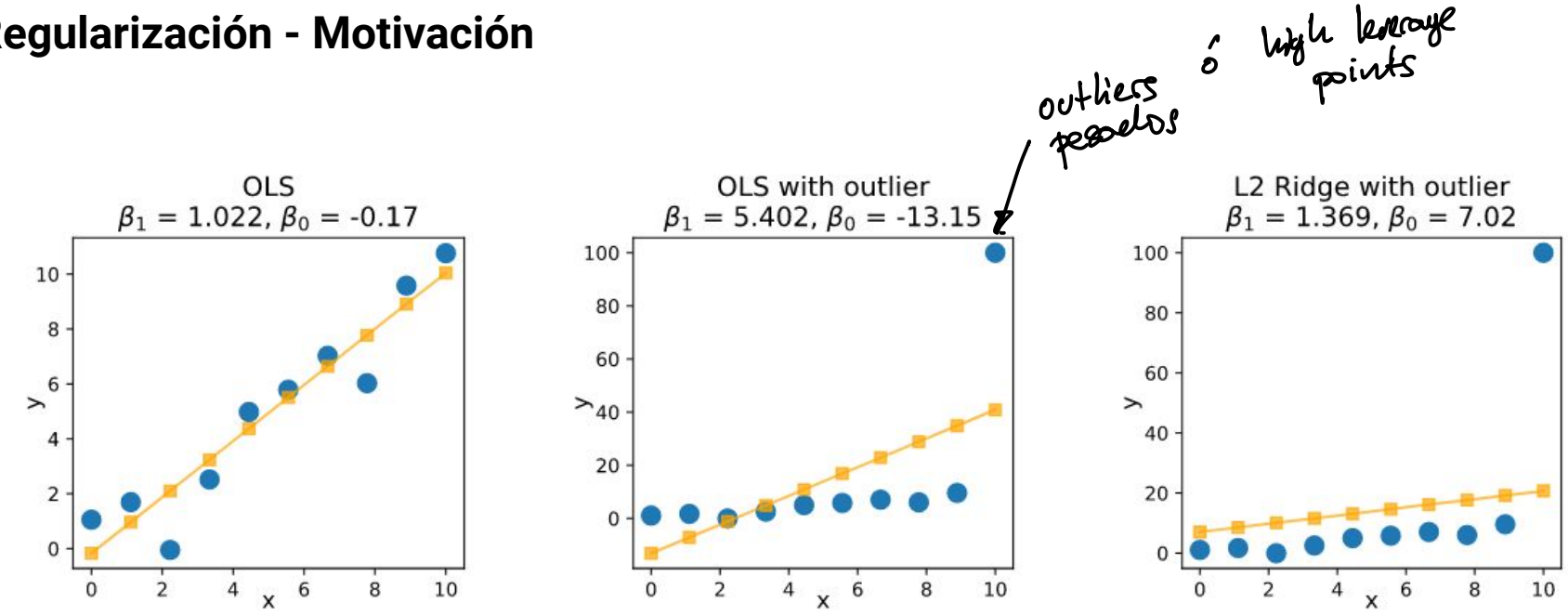


calculando $\hat{\beta}_{ML}$ obtengo un punto

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

$$\mathcal{L} \propto (y - \hat{y})^2$$

Regularización - Motivación



regularización: analiza β's. → mejora* por restricción

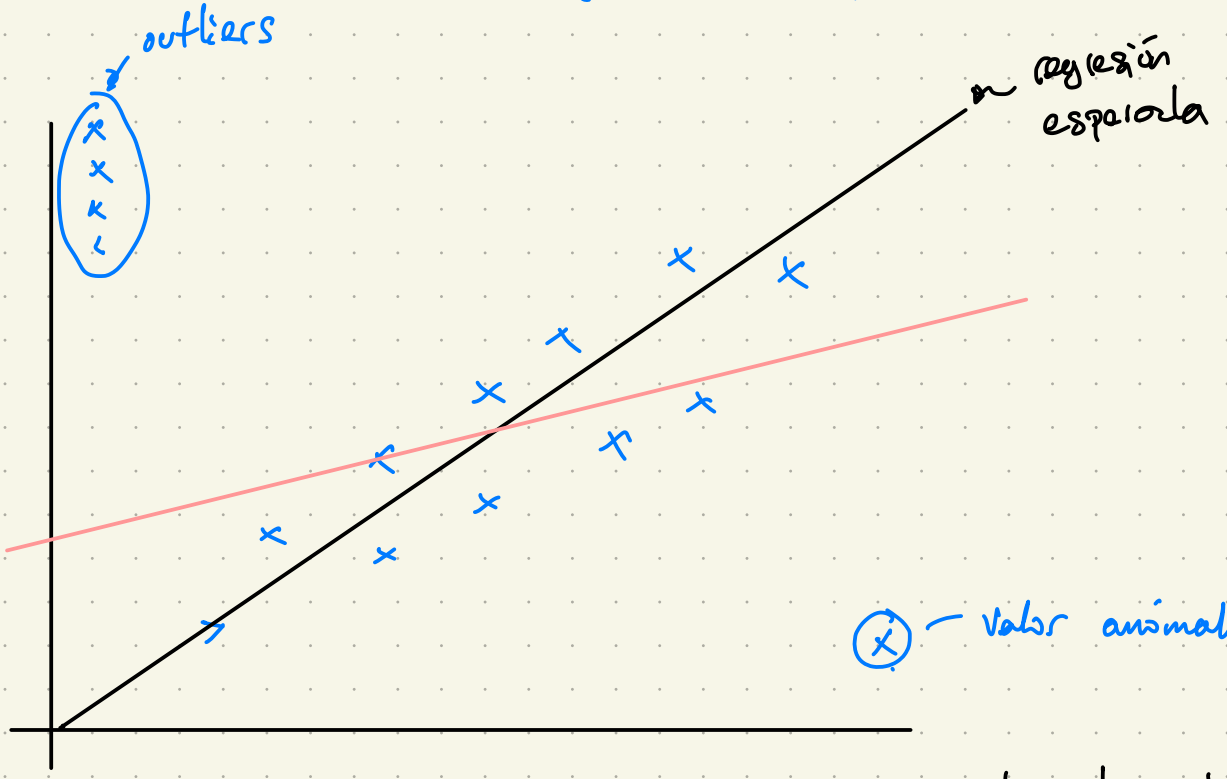
Riesgo Empírico: analiza la predicha. → mejora* por construcción

Riesgo Empírico

$$R(f) = \mathbb{E}\left(L(f, \hat{f})\right) \rightsquigarrow L(f, \hat{f}) = (f - \hat{f})^2$$

fn. de pérdida L2

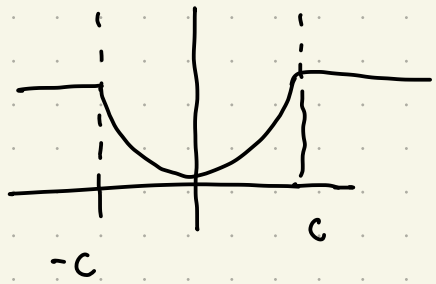
función de pérdida (loss function)



¿Cómo podemos mejorar mi reg. lineal?

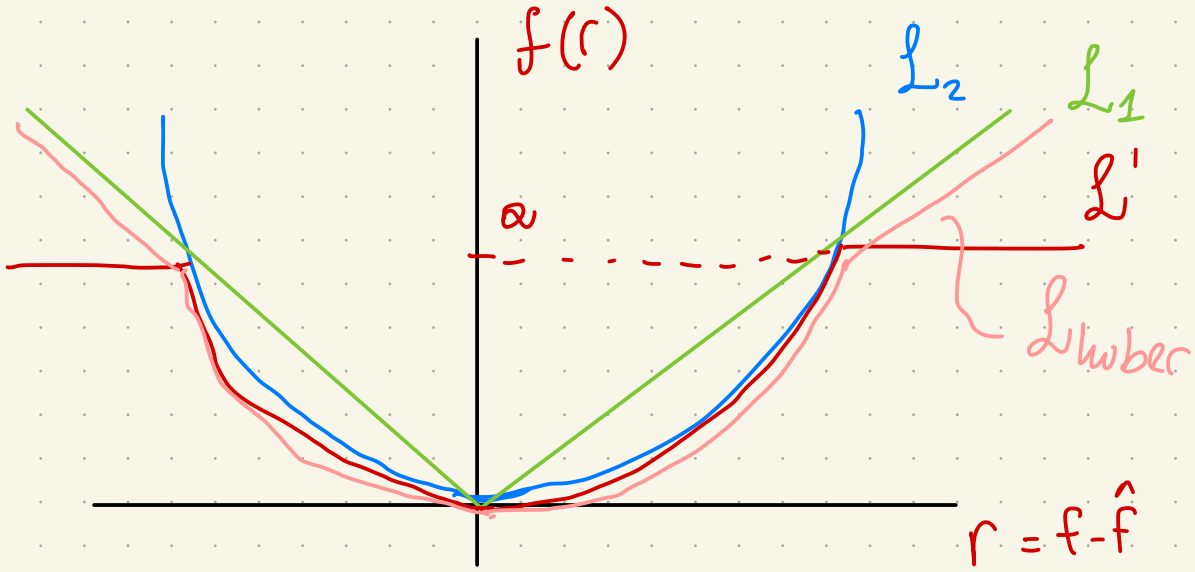
1. Cambiar L
   - $L_1 = |f - \hat{f}|$
   - $L_2 = (f - \hat{f})^2$
   - $L_{II} = \mathbb{I}_{|f-\hat{f}| \leq c}$

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

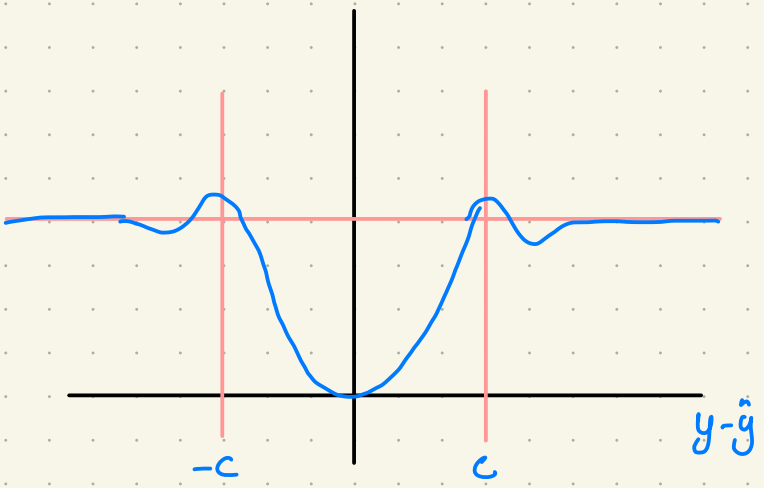


- L_{huber}
- L_{Tukey}

} Estimadores robustos de la pérdida (multiparamétricos)



$$\mathcal{L}_{huber} = \begin{cases} \frac{1}{2} r^2 & |r| \leq \delta \\ \delta\left(|r| - \frac{1}{2}\delta\right) & \text{o.w.} \end{cases}$$



$$\mathcal{L}_{Tukey} = \begin{cases} c^2/6 \left(1 - \left[1 - \left(r/c\right)^2\right]^3\right) & \text{para } |r| \leq c \\ c^2/6 & \text{o.w.} \end{cases}$$

'huber regressor' en sklearn

robust regressor en statmodels.

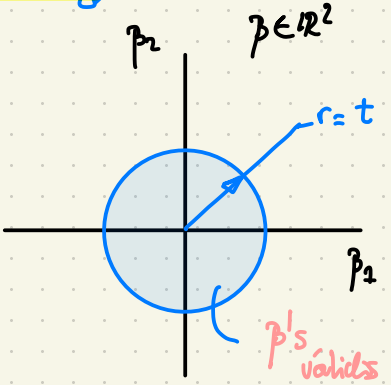
nosotros siempre buscamos minimizar R de la forma más sencilla, a veces cambiar L no es barato o peor no es útil => probamos **regularizar**

2. Regularización.

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

partimos de

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



Espacio de parámetros

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

λ: parámetro de complejidad

Término de regularización (**Weigth decay**)

Regularización

Expectation - maximization Algoritmo (E-M)

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

Observado - Predicción

↓

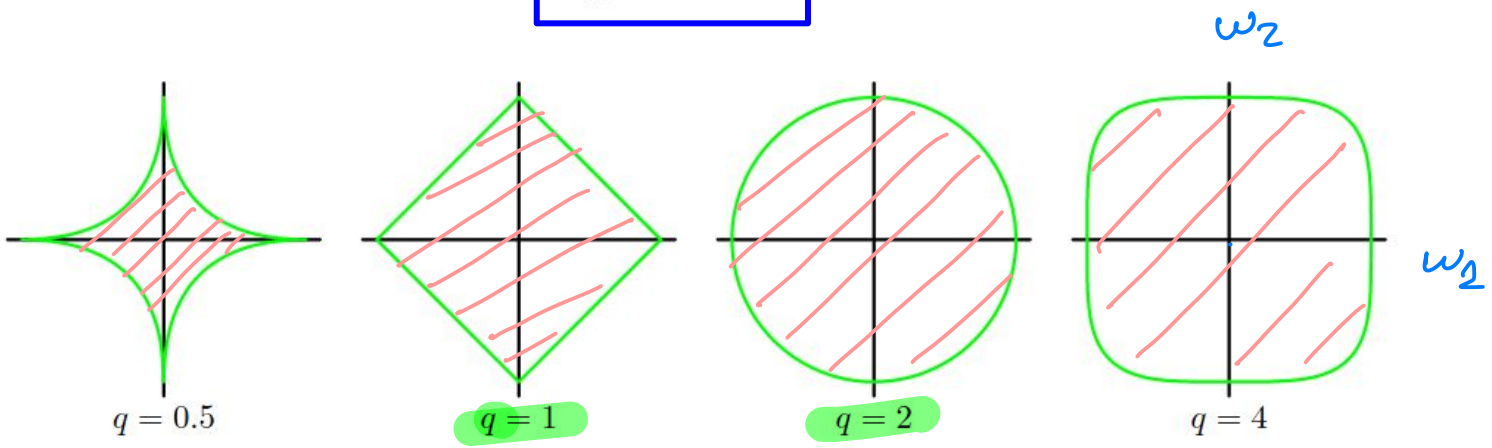
w está “libre”

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

Regularización

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

Término de regularización "weight decay" → w afecta la pérdida



Valores validos de $\|w\|_q$

Lasso

penalidad L1

Ridge

pen L2

(w_1, w_2)

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

Maximum A Posteriori como regularización

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

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

$p(w)$

$p(\beta)$

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

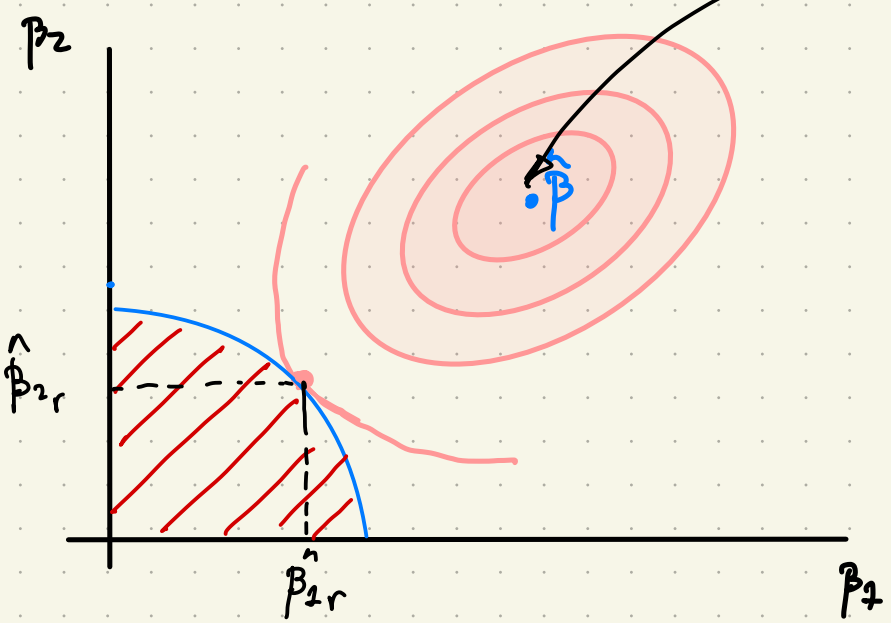
$$p(w|\mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y}|\mathcal{X}, w)p(w)}{p(\mathcal{Y}|\mathcal{X})}$$

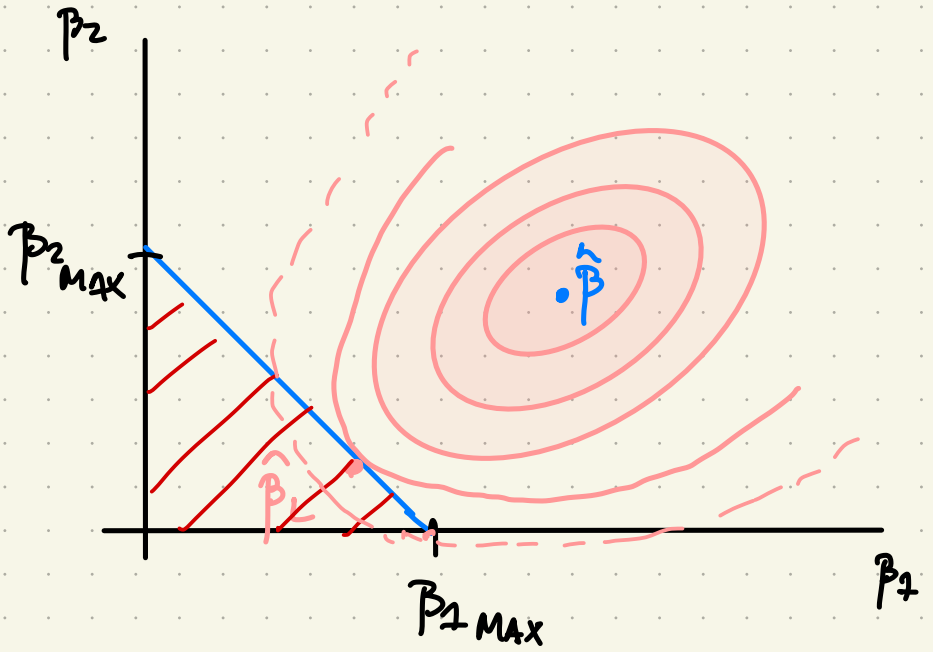
$$w_{map} = (\Phi^T\Phi + \frac{\sigma^2}{b^2}I)^{-1}\Phi^T y$$

Gaussian prior con varianza b2

Representación gráfica:

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





☑ β's válidos

Ridge $\quad - \lambda \sum_j |\beta_j|^2$ $\quad$ Lasso

↓ $\qquad$ ↓

porque enrobustece mi sist. $\qquad$ porque hace un proceso de selección de variables

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

Pero, regularizar viene a costo de alejarnos del óptimo.

¿Cómo funciona?

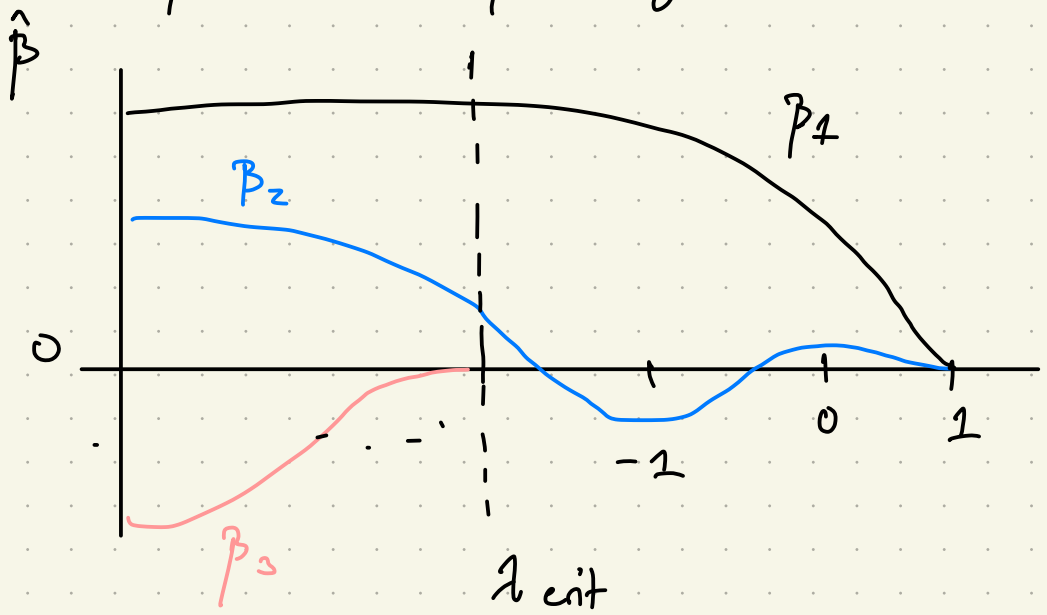
1. Elegir q (Lasso: 1, Ridge: 2)

2. Voy a elegir un vector de λ's "apropiado" ($\lambda \uparrow \Rightarrow \uparrow$ penalidad)

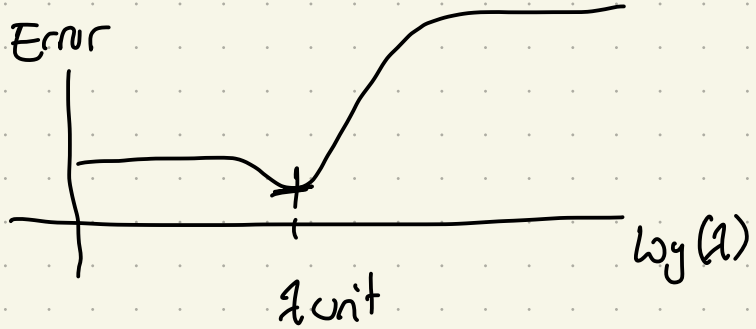
3. optimizo para λ_i ms $RSS(\lambda; q) = \|y - X\beta\|^q + \lambda \|\beta\|_q$

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

5. Comparamos y elegimos el mejor:



Regresión Lasso típica

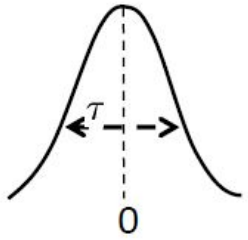


Maximum A Posteriori como regularización - Ridge (L2)

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

I) Gaussian Prior

$$\beta \sim \mathcal{N}(0, \tau^2 \mathbf{I}) \qquad p(\beta) \propto e^{-\beta^T \beta / 2\tau^2}$$



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

Ridge Regression

$\lambda = \text{constant}(\sigma^2, \tau^2)$

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

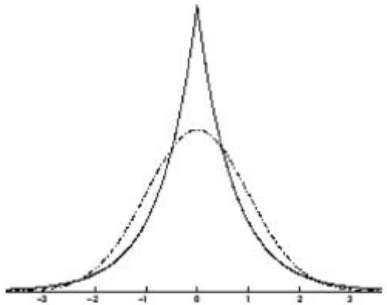
Machine Learning 10-315, Carnegie Mellon

Maximum A Posteriori como regularización - LASSO (L1)

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

II) Laplace Prior

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



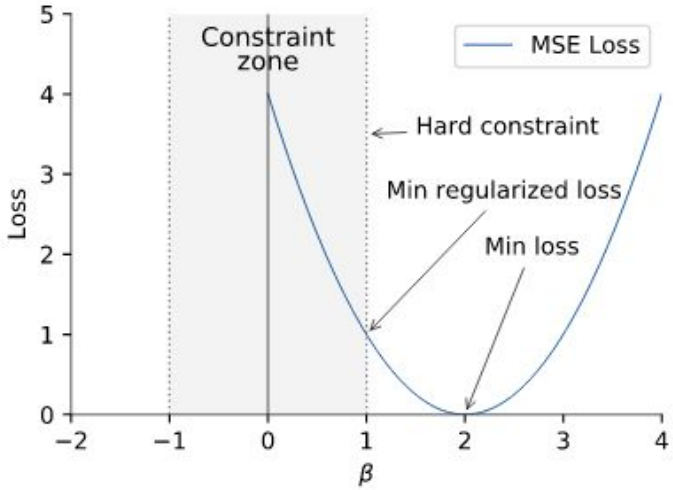
$$\hat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda \|\beta\|_1$$

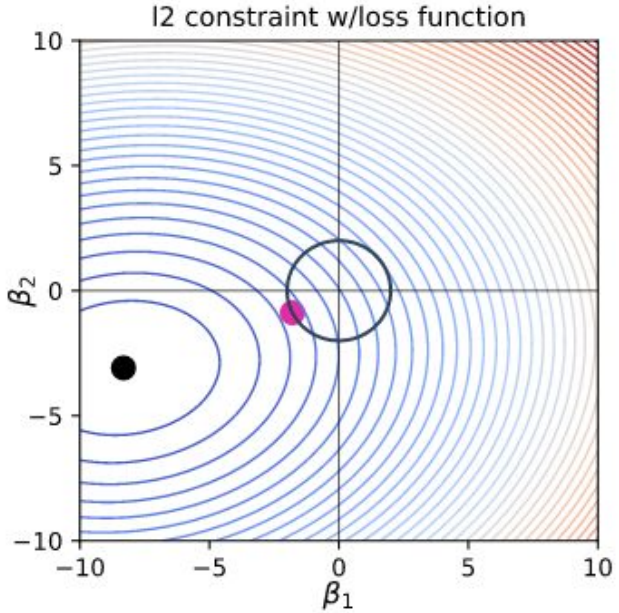
Lasso

$\downarrow$

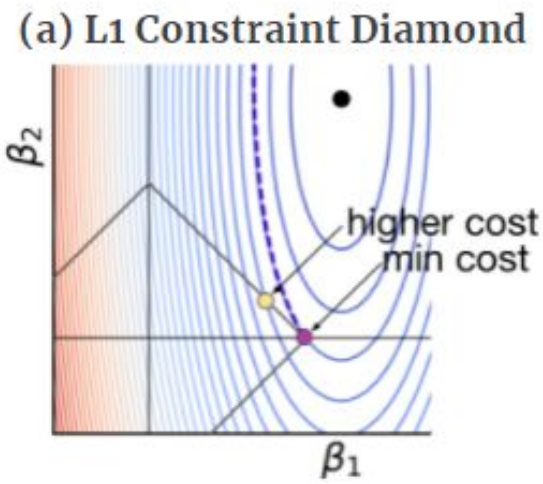
$\text{constant}(\sigma^2, t)$

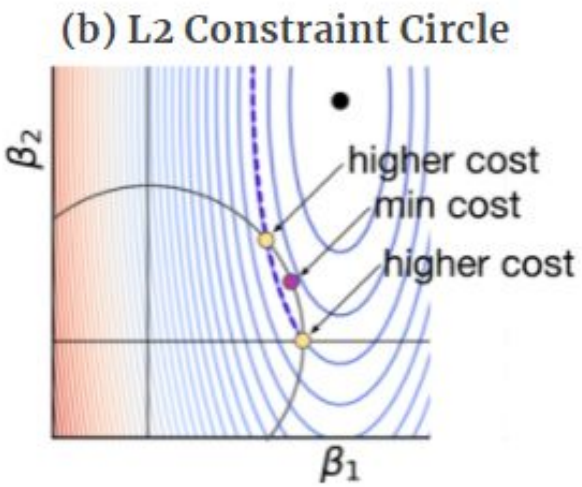
Regularización





Regularización





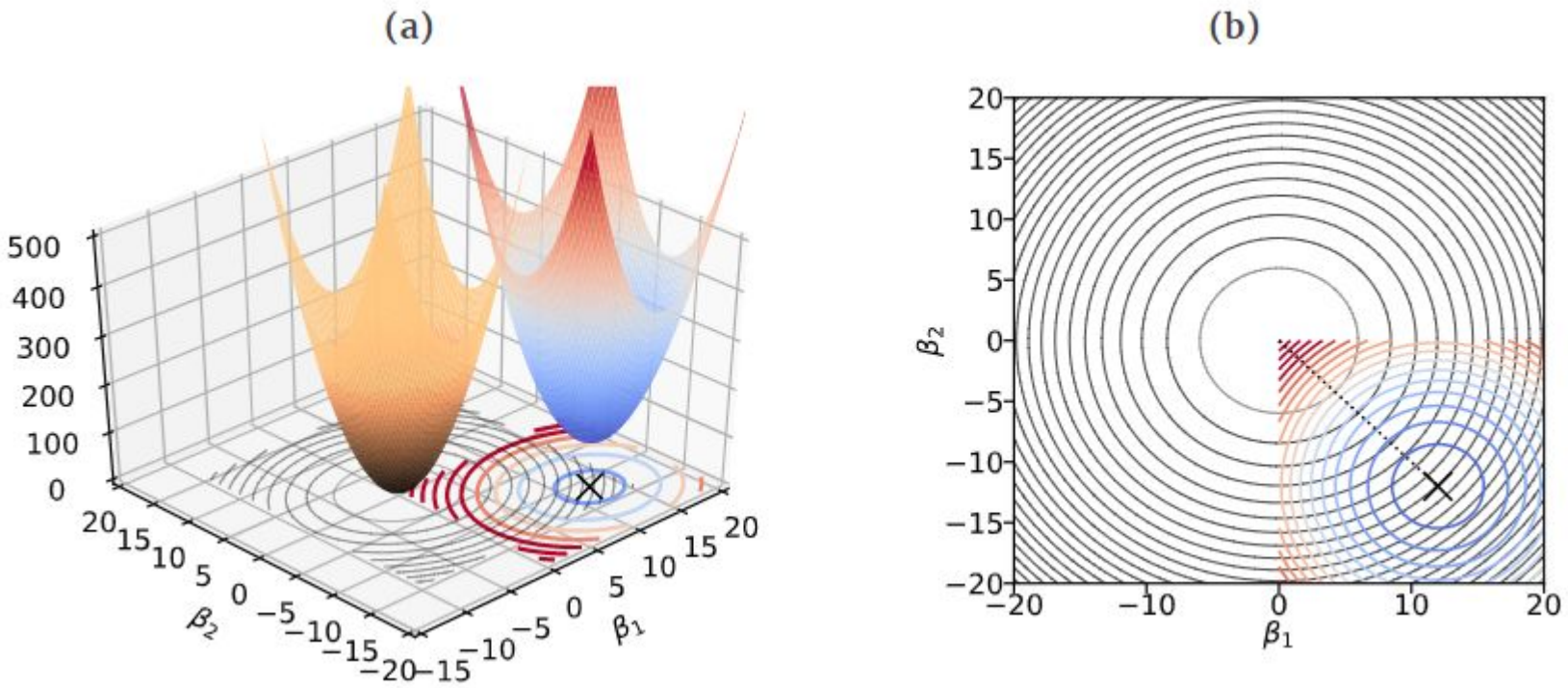
ElasticNet λ término de complejidad

¿Qué β se reduce más?

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

α parametro de elasticnet

Regularización



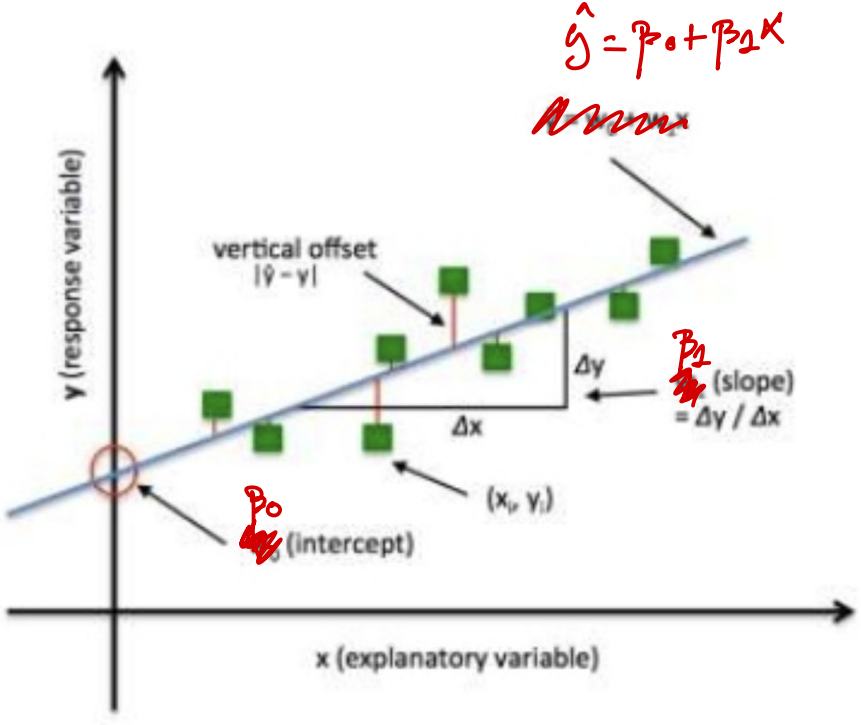
Gradiente Descendente

Implementación de Gradiente Descendente

Solucion analitica

$$\min_{\beta} \|Y - X\beta\|_2^2$$

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



betas = np.arange(-10.10, step=0,2)

↳ busqueda por fuerza bruta (manera)

Implementación de Gradiente Descendente

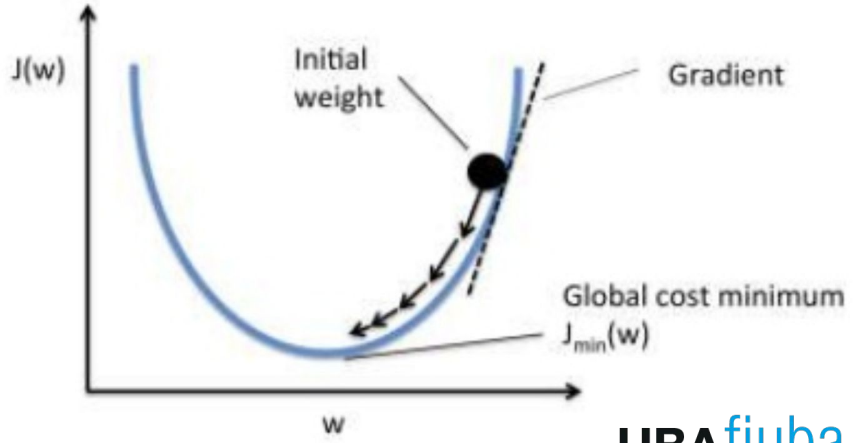
Solución numérica

$$\min_W \|Y - XW\|_2^2 \implies \min_W \sum_i (y_i - X_i \cdot W)^2$$

peso inicial w_0

$$W \leftarrow W - \alpha \nabla \left(\sum_i (y_i - X_i \cdot W)^2 \right)$$

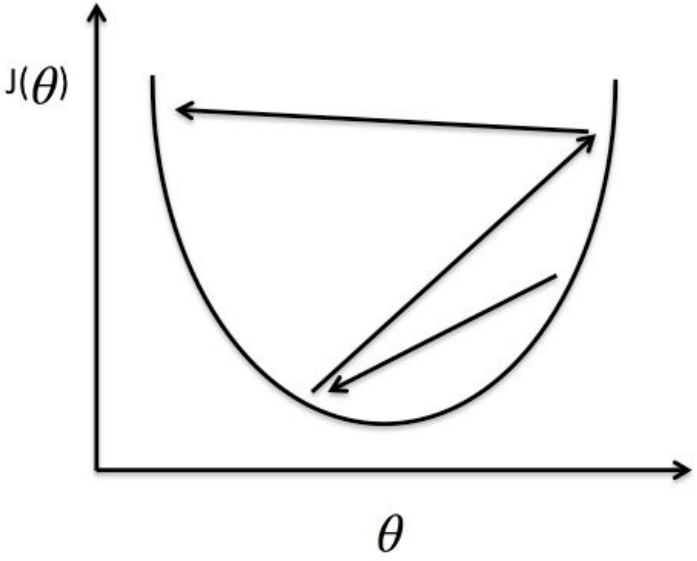
learning rate



Gradiente Descendente

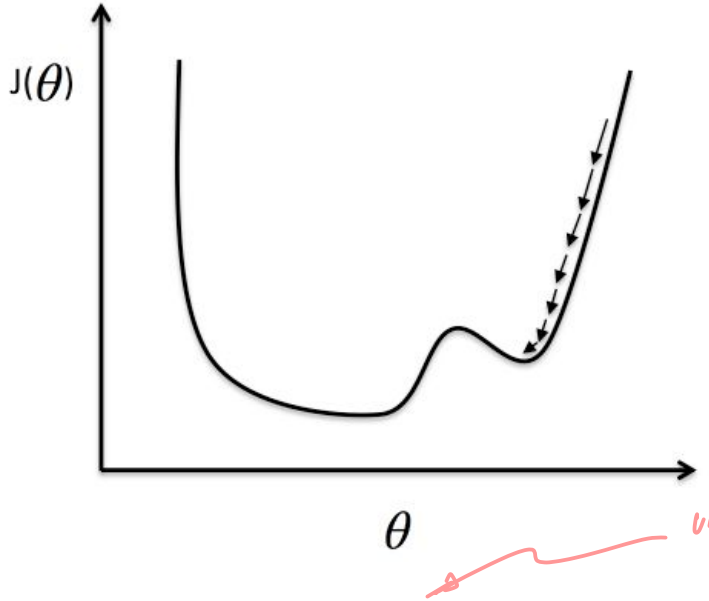
Problema ②

undershooting ≠ Vanishing gradient.



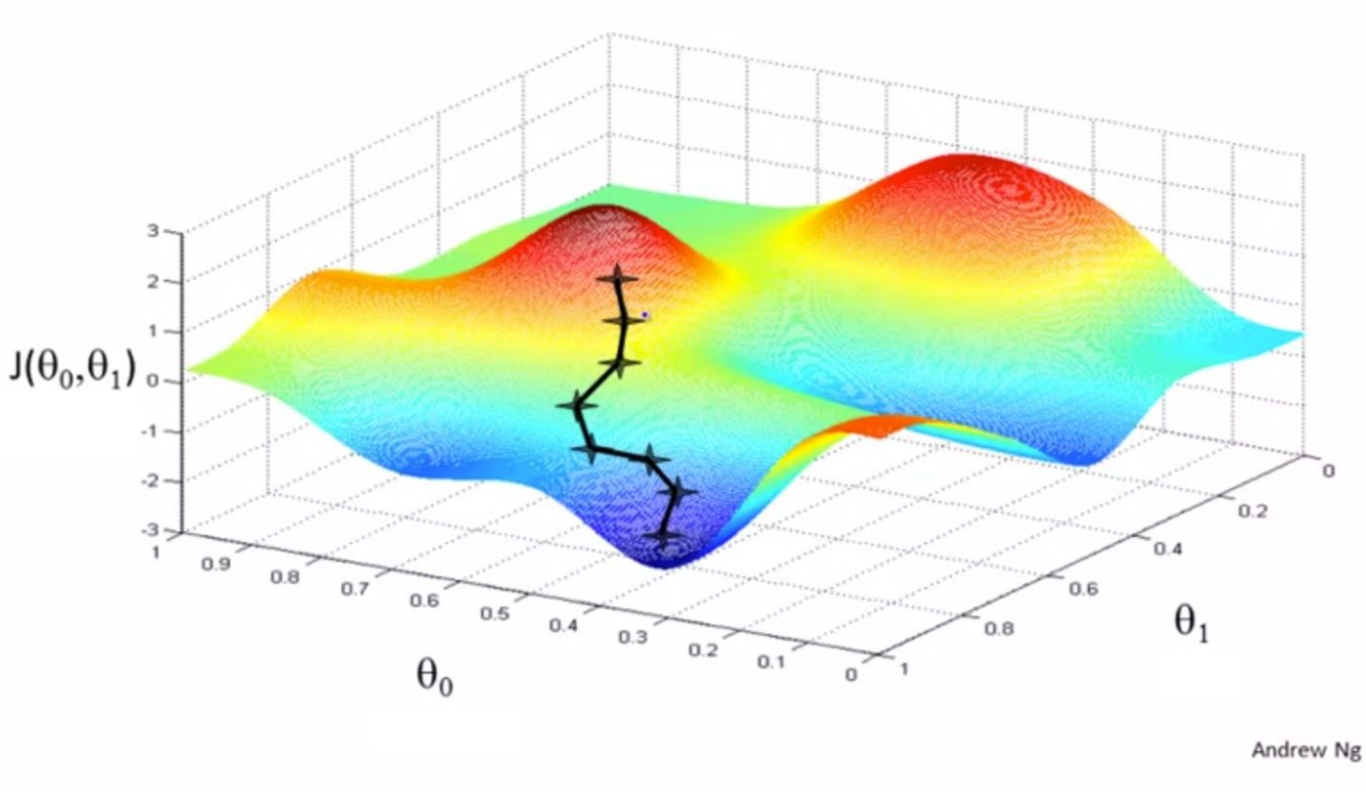
Large learning rate: Overshooting.

overshooting puede relacionarse con exploding gradients.



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

Gradiente Descendente



Implementación de Gradiente Descendente

Solución numérica

$\beta = w$

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

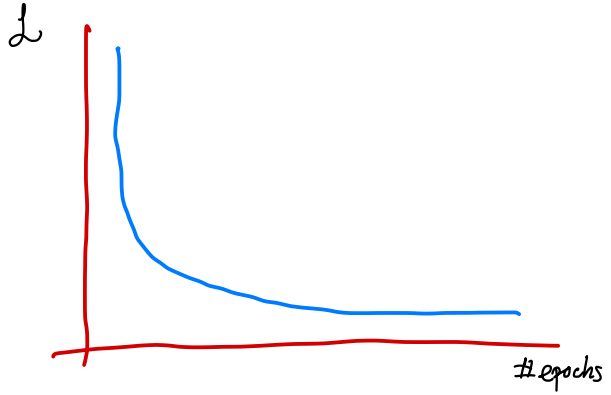
$y_i - \hat{y}_i = r_i$

$\hat{y}_i \propto \beta_t$

Implementación de Gradiente Descendente

Solución numérica

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



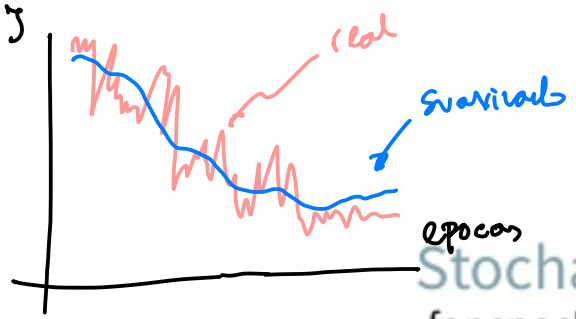
Gradient Descent algorithm

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



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

$i \subset k$

es la cant. total de muestras

es un subset

Stochastic Gradient Descent algorithm

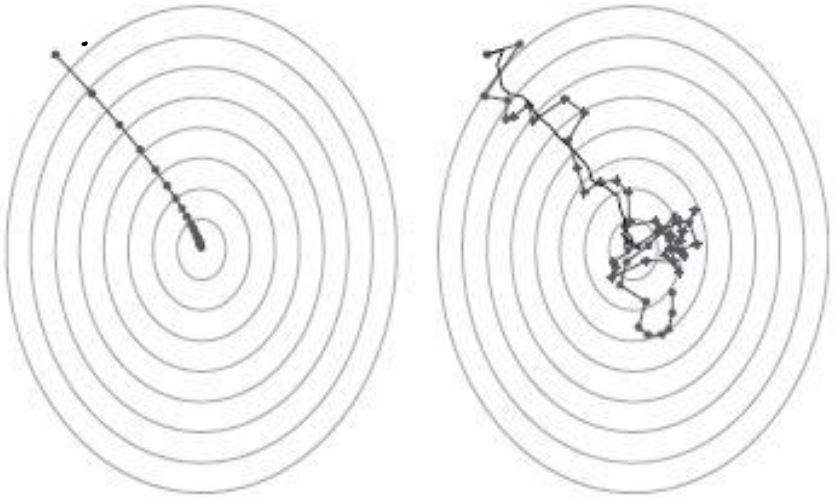
for epoch in n_epochs: (Alternativamente uso tol)

- 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

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

e1 B3 B3 B1 B4 B2

e2 B1 B4 B3 B5 B2

Mini-Batch Gradient Descent algorithm

n_epoch, n_batch, batch_size

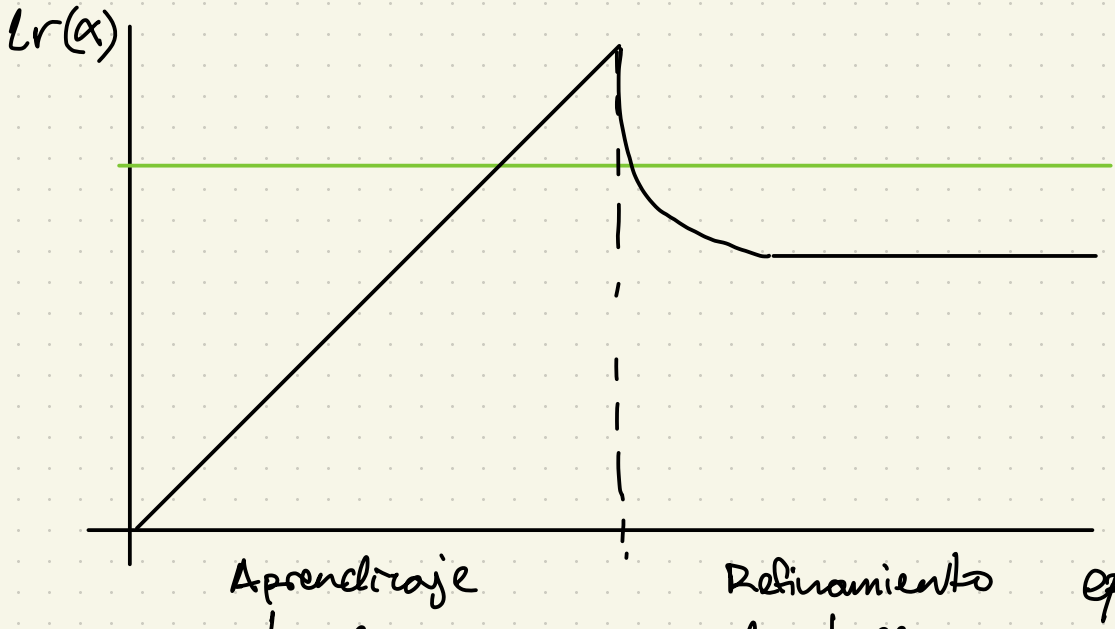
for epoch in n_epochs:

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

Comparativa de gradientes

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



α = cte (caso típico)

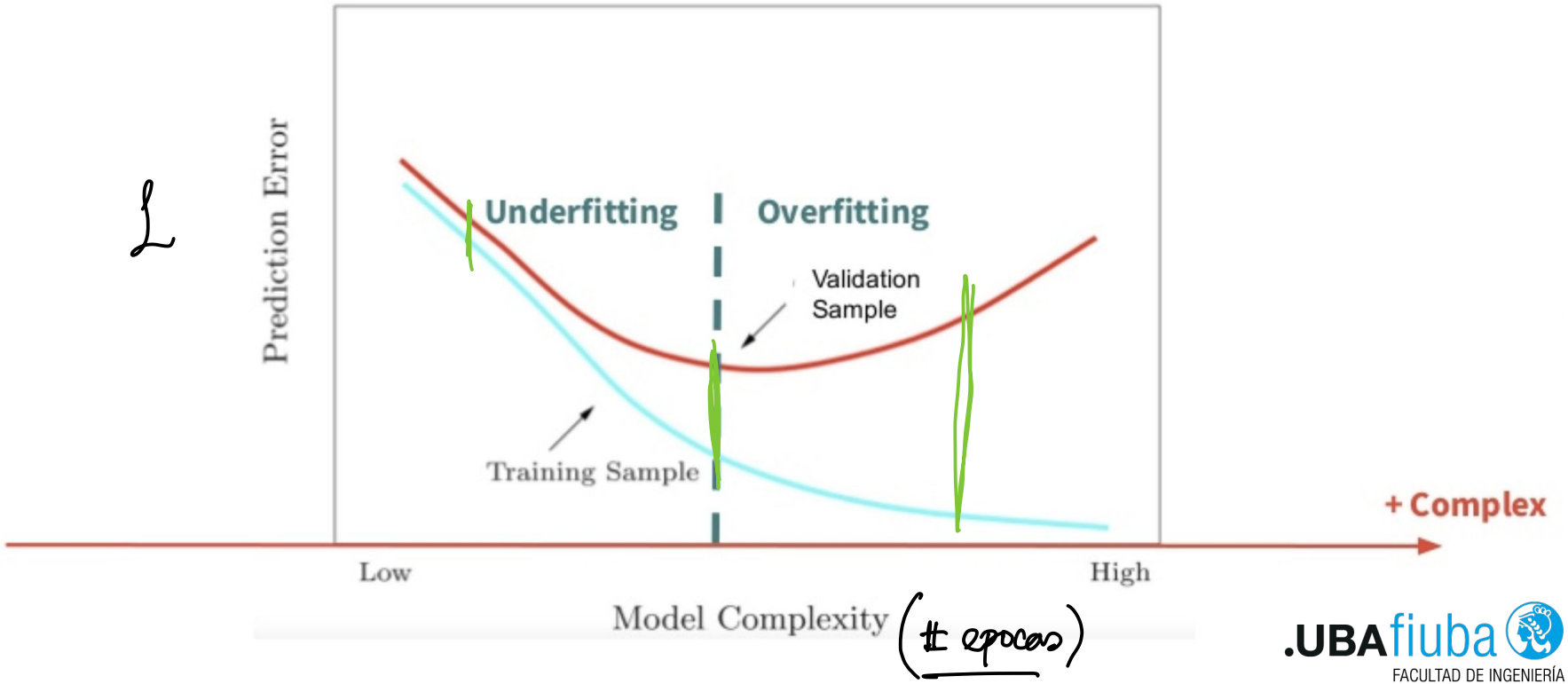
α con dependencia funcional (caso clásico transfer learning)
optimizadores adaptativos

tipicamente $\alpha \sim 0,1$ para aprendizaje

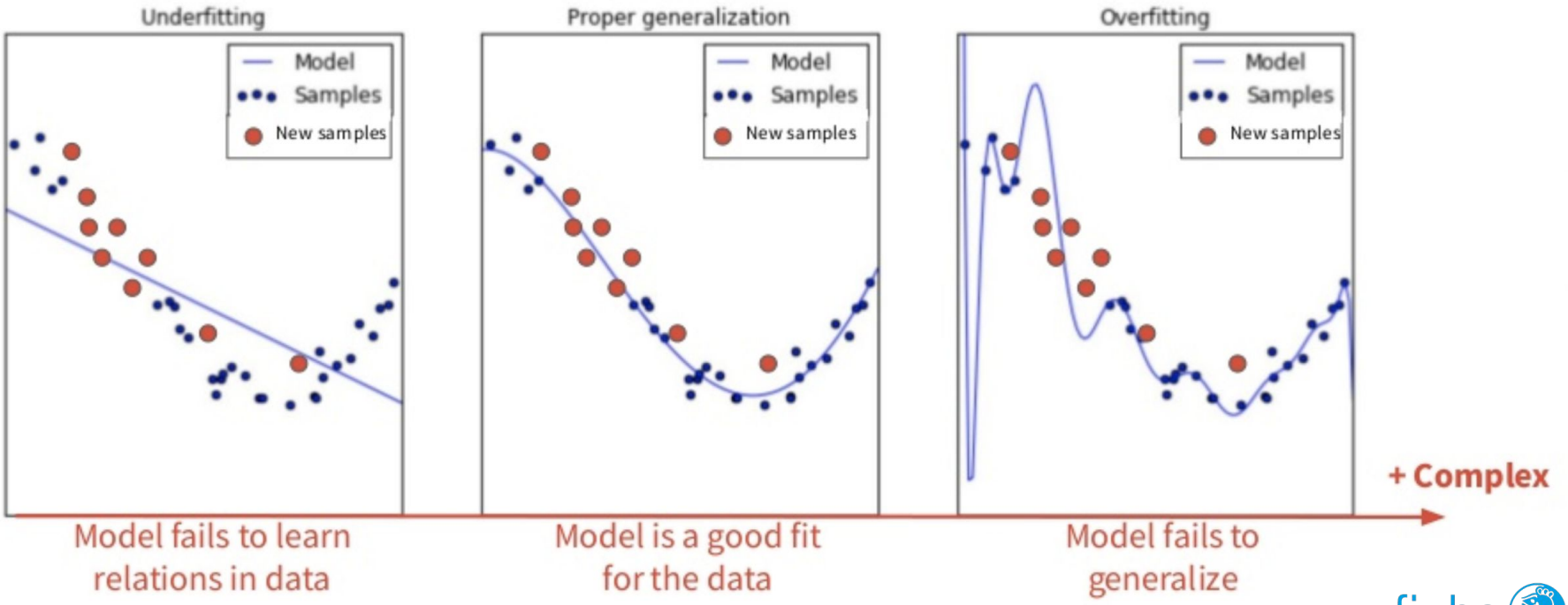
$\alpha \sim 0,0001$ para refinamiento.

Selección de modelos

Selección de modelos

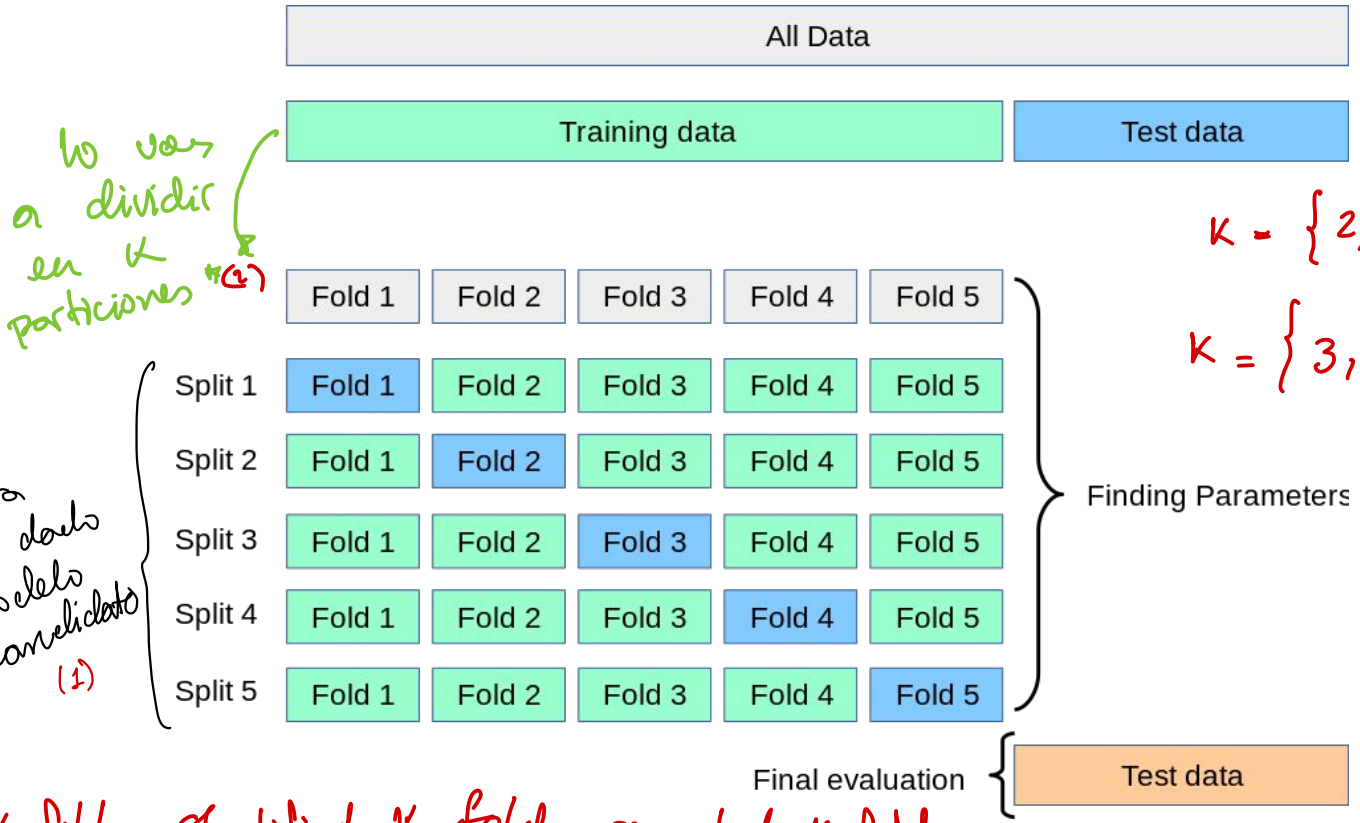


Selección de modelos



Cross-Validation

(2) separar en post ⇒ muestreo representativo.



separamos 70 - 30

lo voy a dividir en K particiones (1)

Para un dado modelo candidato (1)

$K = \{2, \dots, n\_samples - 1\}$

$K = \{3, 5, 10, n\_samples - 1\}$

↓ Validación loo

leave one out

* K-fold, stratified K-fold, repeated K-fold

Selección de los hiper parámetros

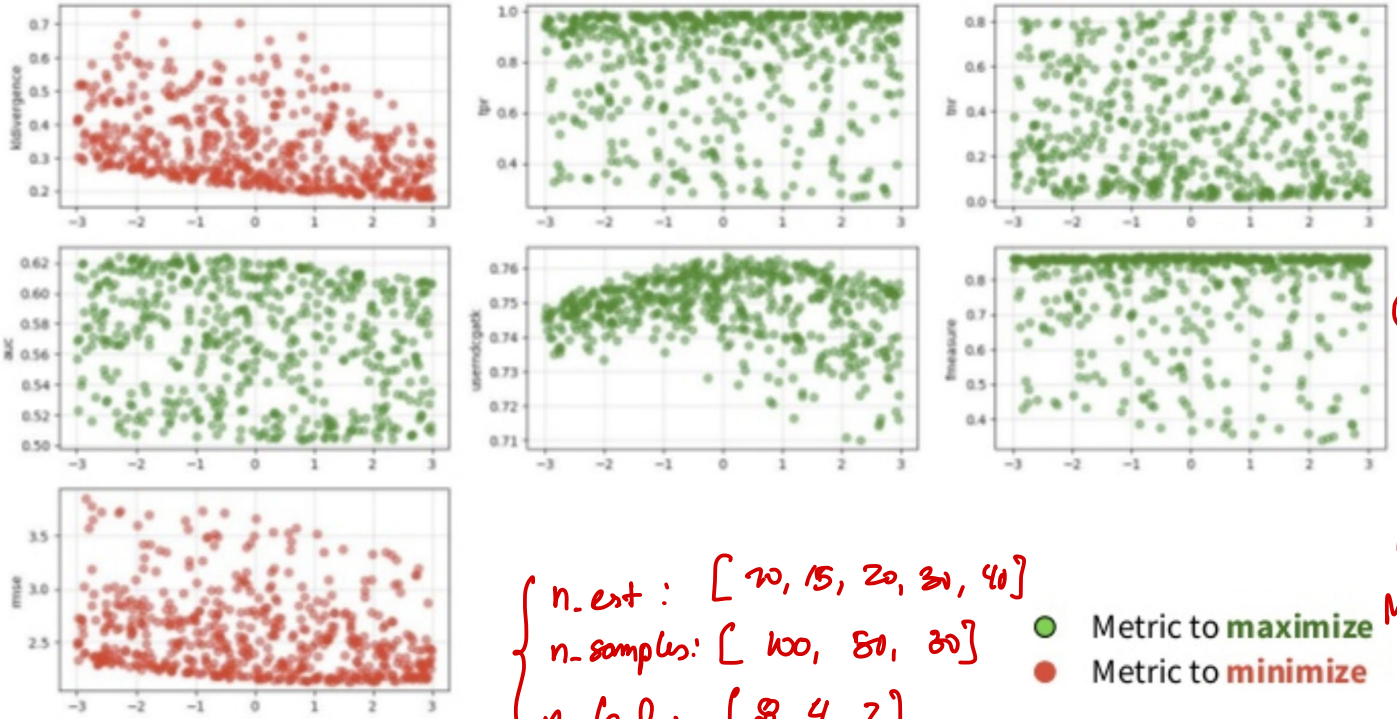
Modelo → Random Forest
hyp RF → n_samples, n_est, leaf_size, width, ...

método de fuerza bruta

Grid Search ③

↓ Alternativa

Random Search



③ GS(X_train, Y_train
CV = K-fold,
Modelo = ...,
Grilla = {...})

MC1: RF(10, 100, 8)
MC2: RF(10, 100, 4)
⋮

{ n_est: [10, 15, 20, 30, 40]
n_samples: [100, 80, 20]
n_leaf: [8, 4, 2]

● Metric to maximize
● Metric to minimize

Pasos van a ser:

- Separar train - test
- Armar las K particiones (CV)
- Elegir el espacio de búsqueda (3) (elegimos los modelos cand.)
- for combinacion in espacio_de_busqueda:

    MC = modelo (** combinación) ← inicializamos el modelo en una clase conf.

    for fold in folds:

        test ← fold

        train ← folds - {fold}

        MC_f.fit(train)

        Metricas(MC_f(test))

        return métrica

modelo	split	métrica	...
MC_1^1	1	0,86	
MC_1^2	2	0,71	
⋮	⋮	⋮	
MC_1^5	5	0,92	

Métrica (MC_1) = promedio de todos estos.

Modelo candidato: modelo elegido de antemano con una dada conf.

MC_1 : RL (pen = L_1)
MC_2 : RL (pen = L_2)

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

→ es iterativo → tiene asociado un Solver y un conjunto de parametros de conv. (n_it, tol, ...)

Dataset → separo en train 80%, test 20%

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

- planteamos el espacio de hp. de busqueda:

$$\begin{cases} \varphi_0 : [0,1] \\ \varphi_1 : [1,5] \\ \vdots \\ \varphi_n : [a,b] \end{cases}$$

en grid la cant de valores por hp. no suele ser igual. vamos a generar todas las permutaciones posibles de modelitos. (Grid Search)

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

m_2: $\varphi_0 = 1, \ \varphi_2 = 1, \ \dots, \ \varphi_n = a$

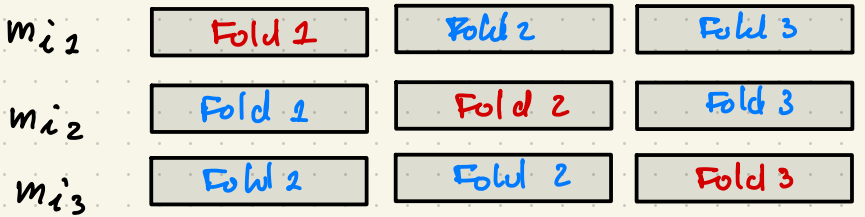
⋮

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

- K-fold cross validation → tomo K folds y entreno mult. m_i ①
- hold-out: fijo un fold que se usa para validar (lo mismo q' train-test-val)
- leave-one-out: toma K folds con K = len(data) - 1 y luego ②

② ¿cómo func. K-fold CV?

para el ej. K=3



← Entrenamos m_i con F_2, F_3 y val con F_1

train set

Val. set

métrica(m_{i1}) puede no ser igual a métrica(m_{i2}) y así para todo fold.

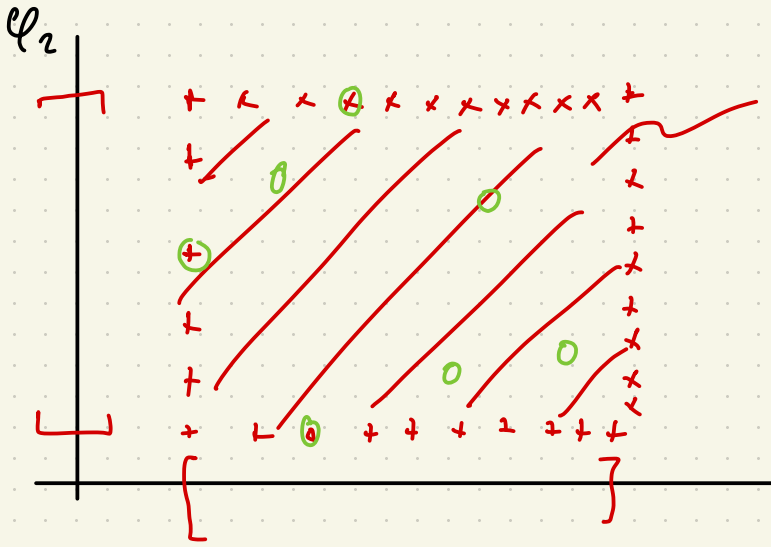
una vez calculados m_{ij} para un dado i, calculo la métrica promedio:

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

Al final del proceso tengo una tablita:

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

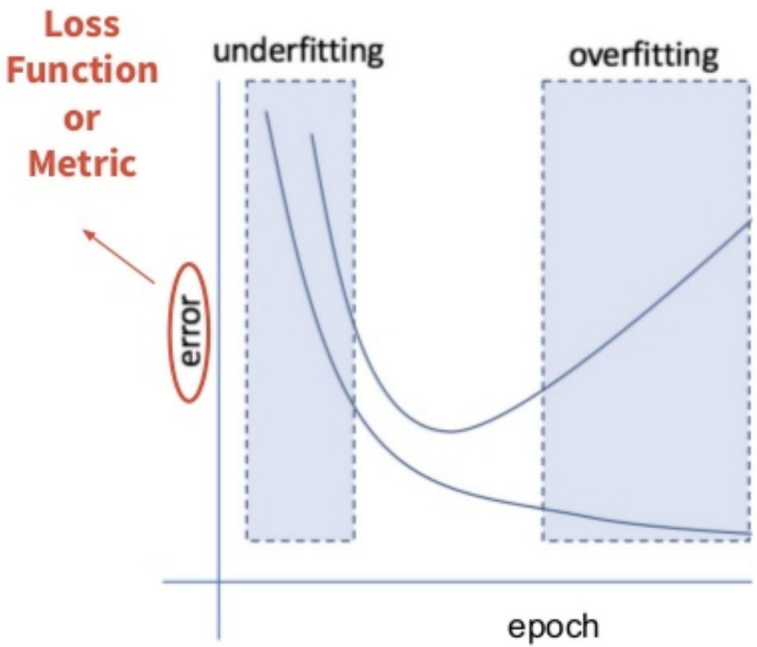
el ranking se basa en que modelo performa mejor.



Grid search

Randomized GridSearch

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

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