


2023/


# ITBA

# Maestría en Management & Analytics


## Clase 4 - Optimización de Hiperparámetros



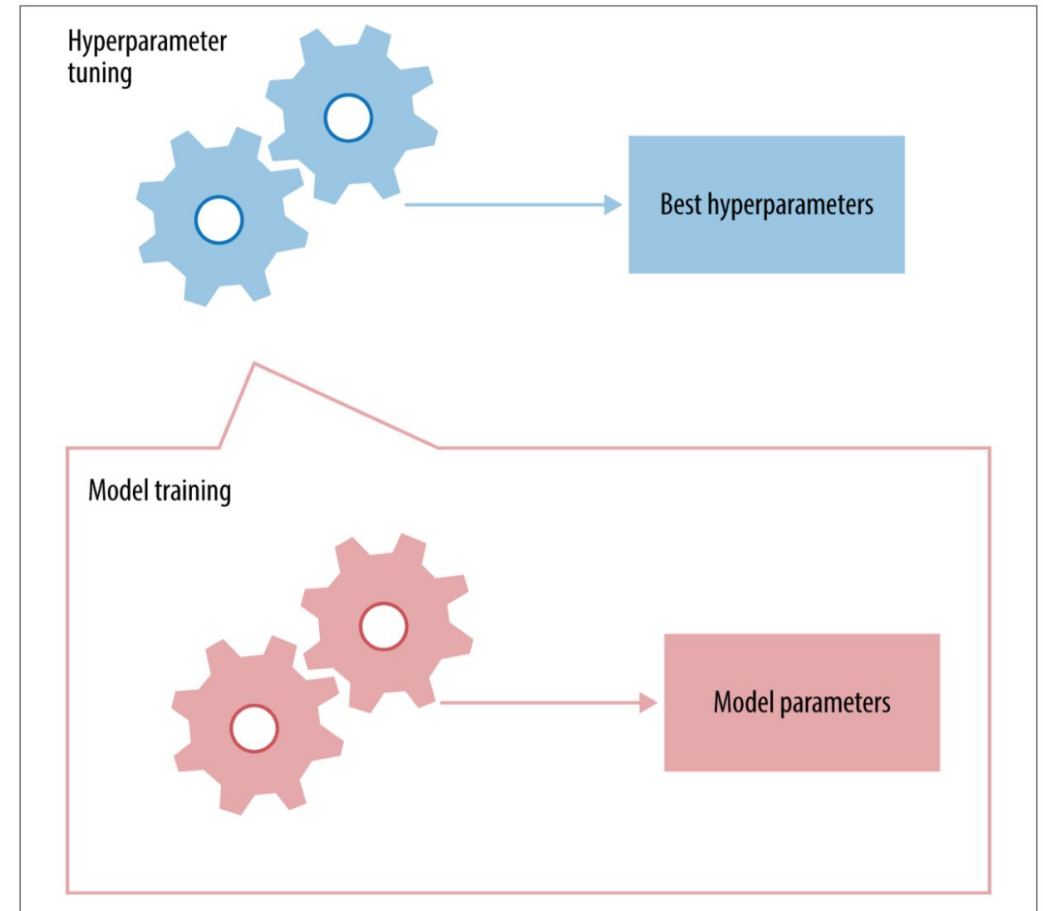
En esta clase vamos a profundizar sobre la diferencia entre los **parámetros** y los **hiperparámetros**.



Luego vamos a ver técnicas para buscar **optimizar** los hiperparámetros de un algoritmo de machine learning siguiendo una metodología que nos permita evaluar correctamente su poder de generalización.



En Machine Learning, tenemos **dos tipos de parámetros**: los que se aprenden de los datos de entrenamiento, por ejemplo, los pesos en la regresión lineal, y los parámetros de un algoritmo de aprendizaje que se optimizan por separado, es decir por fuera le procedimiento de ajuste (o fit) de los datos. Estos últimos son los parámetros de ajuste, también llamados **hiperparámetros**, de un modelo, por ejemplo, el parámetro de regularización en la regresión Ridge o Lasso.



## Set de validación o desarrollo

Cuando optimizamos hiperparámetros, como por ejemplo el valor de regularización  $\alpha$  o la cantidad de vecinos en KNN, tenemos que entrenar los algoritmos con diferentes valores y evaluar sus resultados.

¿Evaluamos sobre el set de testeo?

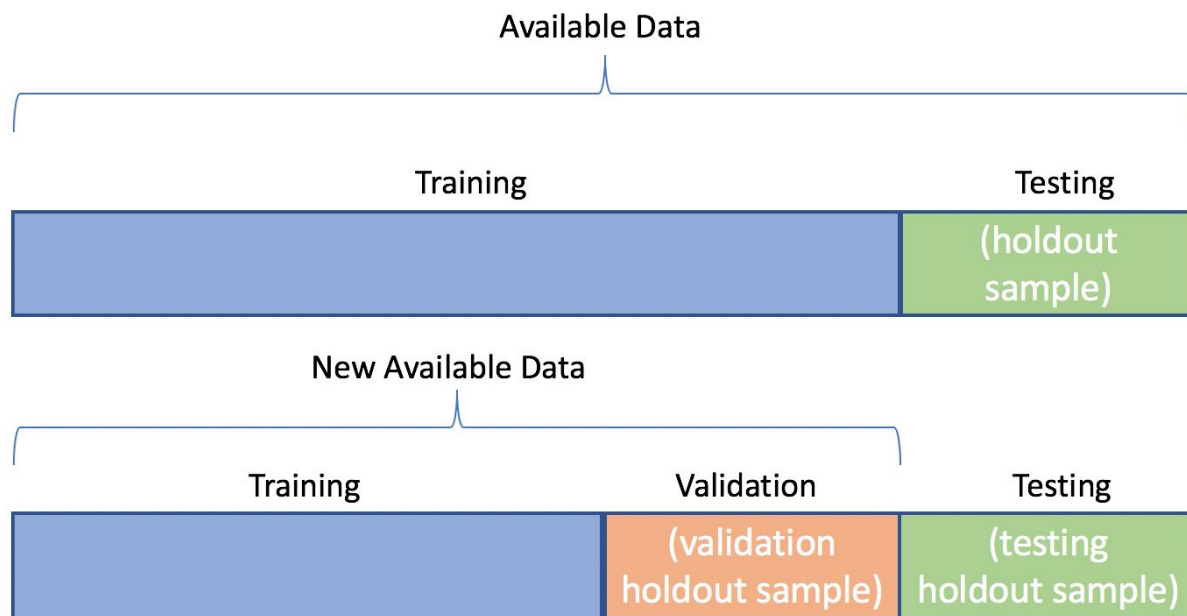


### ¡Atención!

Si evaluamos varios modelos contra el set de testeo y elegimos el que performa mejor en ese set, entonces el set de testeo ya no ofrece una estimación insesgada del poder de generalización del modelo.

Entonces lo que hacemos es dividir los datos en los siguientes sets:

- **Training set:** para entrenar el modelo.
- **Validation set:** para seleccionar los mejores hiperparámetros.
- **Test set:** para evaluar el poder de generalización del modelo final.



### ¡Atención!

Estamos dejando muchos datos fuera del set de entrenamiento. Esto, además de resignar datos de los que podría aprender el modelo, puede generar problemas de varianza en los sets de validación y testeo, especialmente con datasets pequeños. Una forma de resolver este problema es usar **cross-validation**.

**Cross-validation** es un método estadístico para evaluar el rendimiento de la generalización que es más estable y exhaustivo que el split entre train y test. Los datos se dividen repetidamente y se entrenan múltiples modelos. Luego se calcula el promedio de la performance en los diferentes *folds*.

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

La versión más utilizada de la validación cruzada es ***k-fold cross-validation***, en la que  $k$  es un número especificado por el usuario, normalmente 5 o 10.

**Ventajas de Cross-validation:**

- Resultados más confiables y estables cuando el dataset es chico, ya que se evalúa la performance sobre todos los datos
- Uso más efectivo de los datos, porque utilizamos todos los datos para entrenar el modelo.

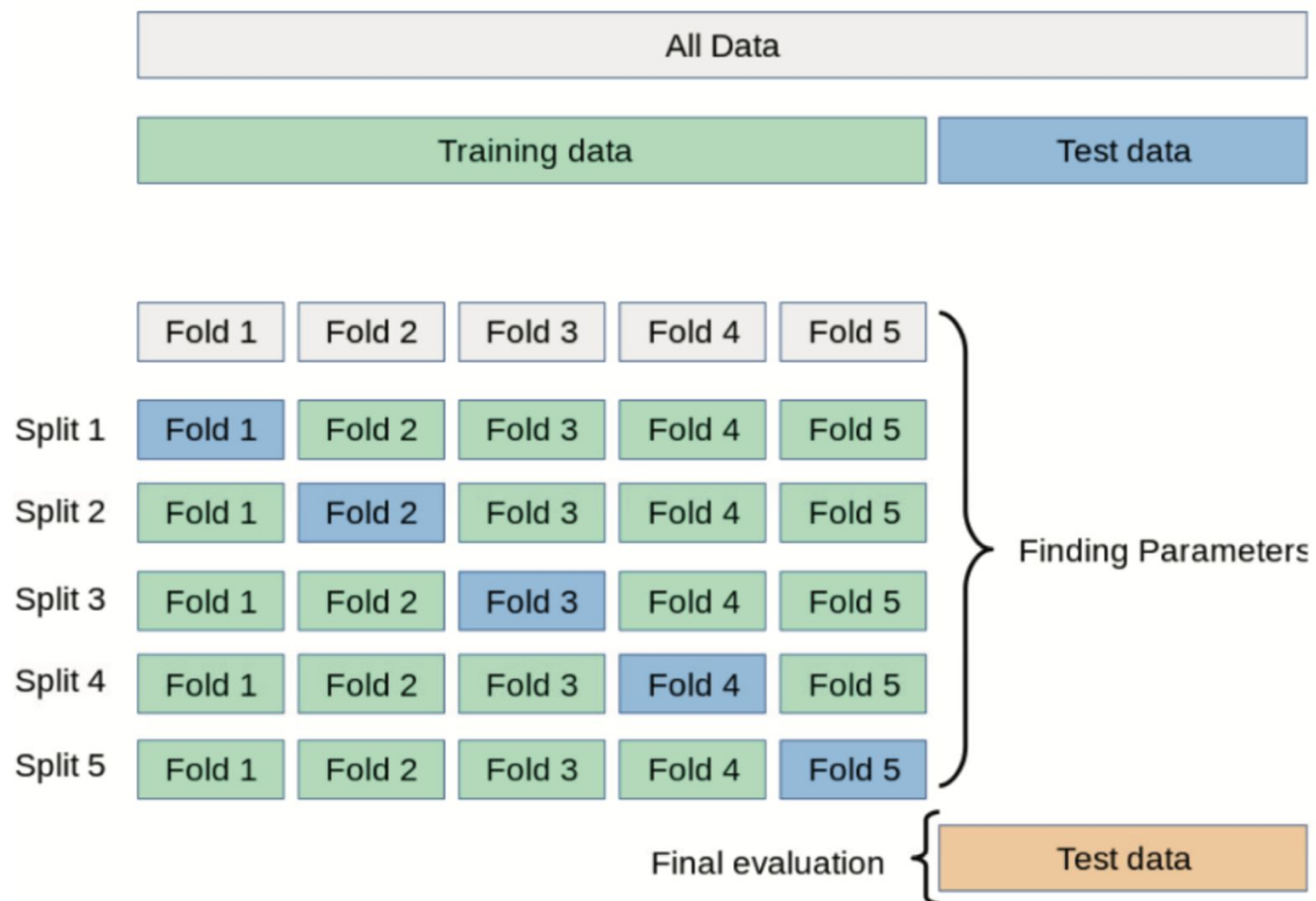
**Desventaja de Cross-validation:**

- La mayor desventaja es el costo computacional, ya que hay que entrenar  $k$  modelos en lugar de un solo modelo.



**Esquema completo:**

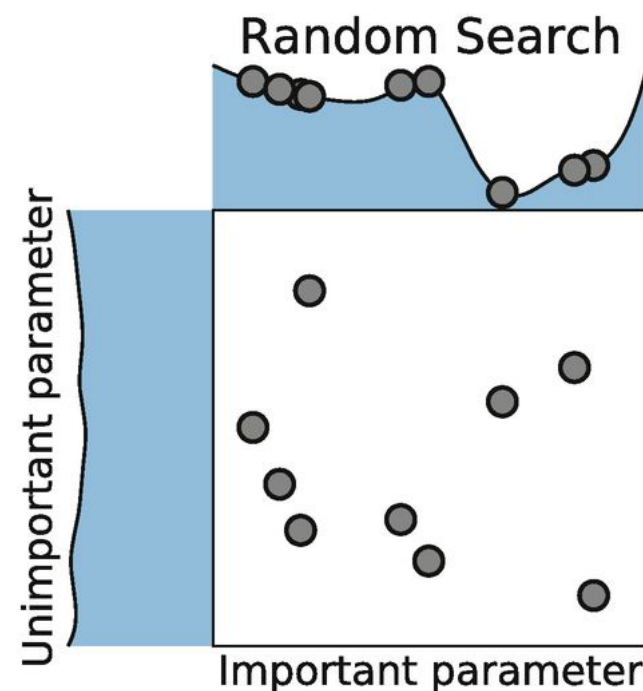
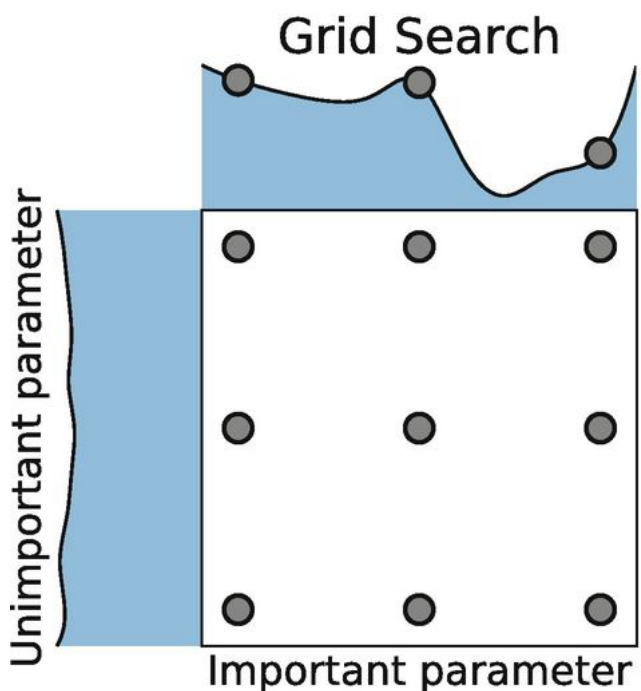
Se valida de forma cruzada el conjunto de entrenamiento para buscar los mejores parámetros y, luego se utiliza el conjunto de test para evaluar el rendimiento de la configuración de parámetros elegida en los nuevos datos.



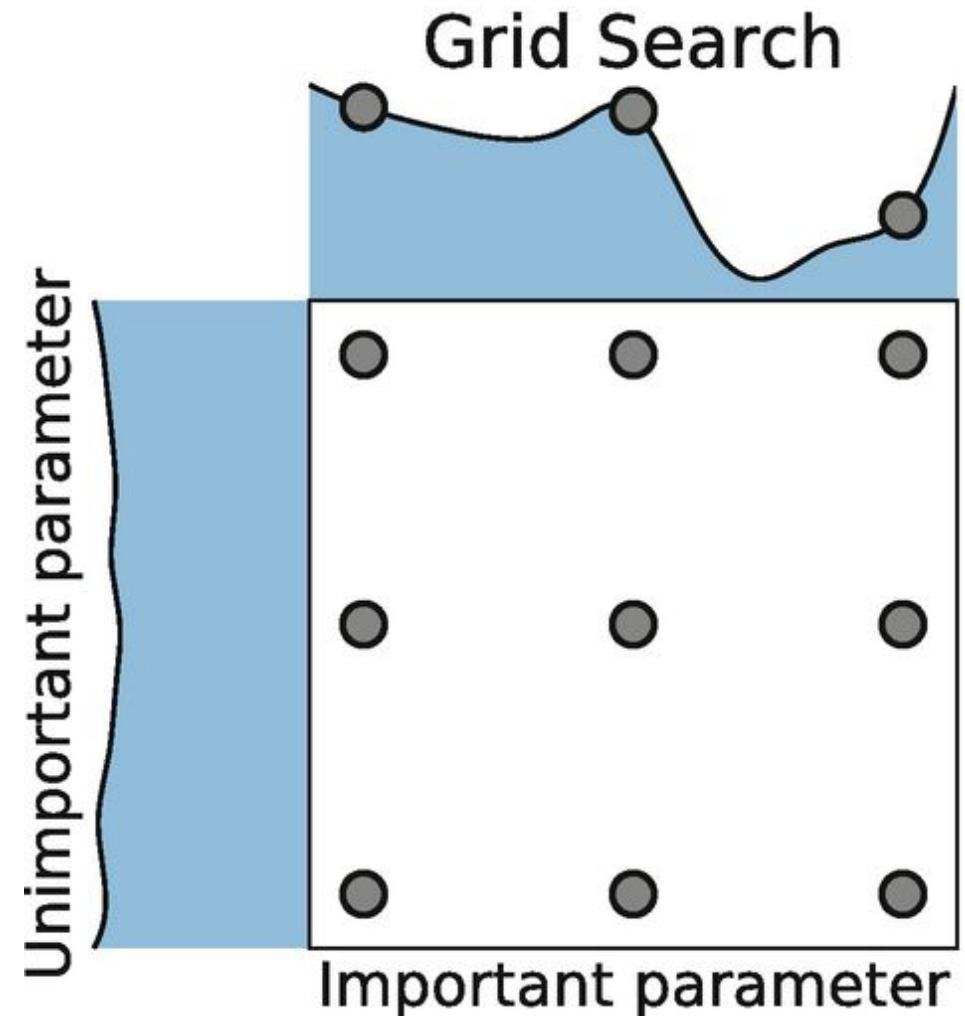
Normalmente un algoritmo de machine learning requiere que **optimicemos** los valores de **una gran cantidad de hiperparámetros**.

Existen diversos algoritmos para tunear los hiperparámetros. En esta clase vamos a ver a los dos algoritmos más clásicos:

- **Grid Search**
- **Random Search**

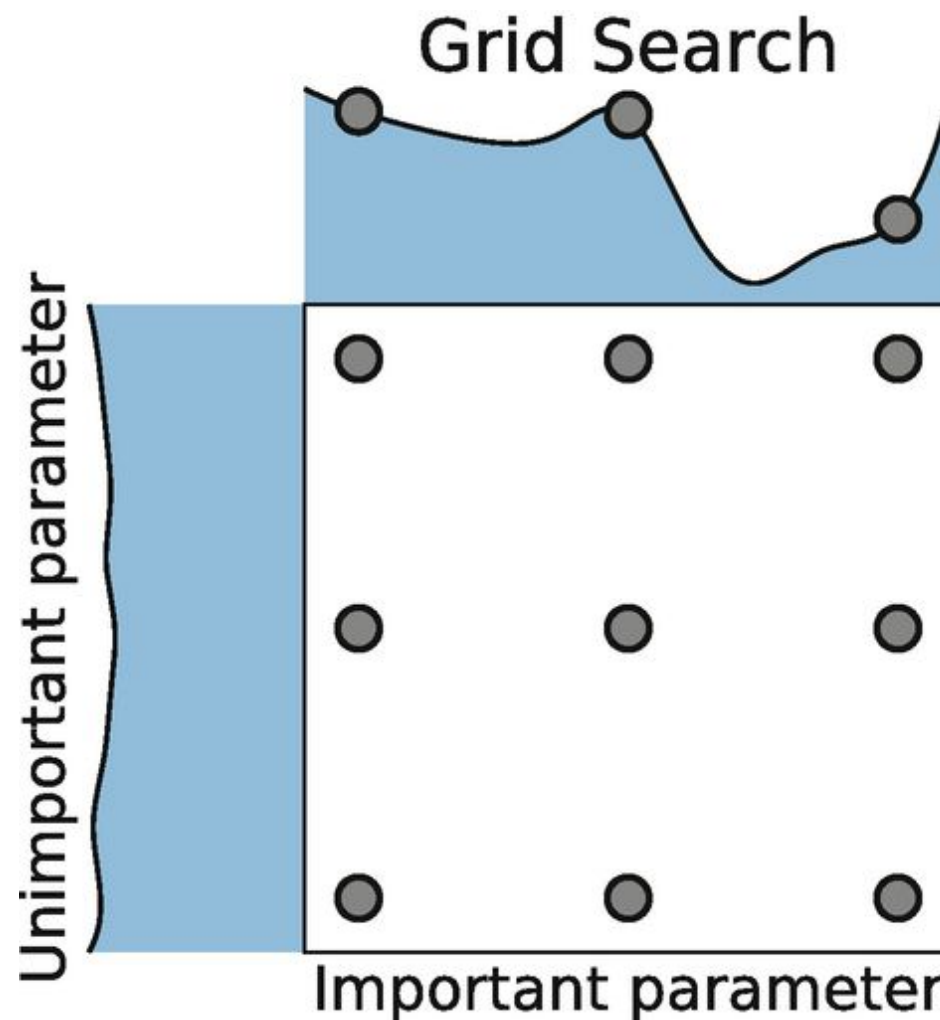


**Grid Search** busca encontrar la mejor combinación de hiperparámetros dentro de una “grilla” (grid) especificada de forma manual. Se realiza una búsqueda exhaustiva para cada valor de la grilla y se elige la combinación que minimiza una determinada métrica de error (generalmente calculada mediante cross-validation)

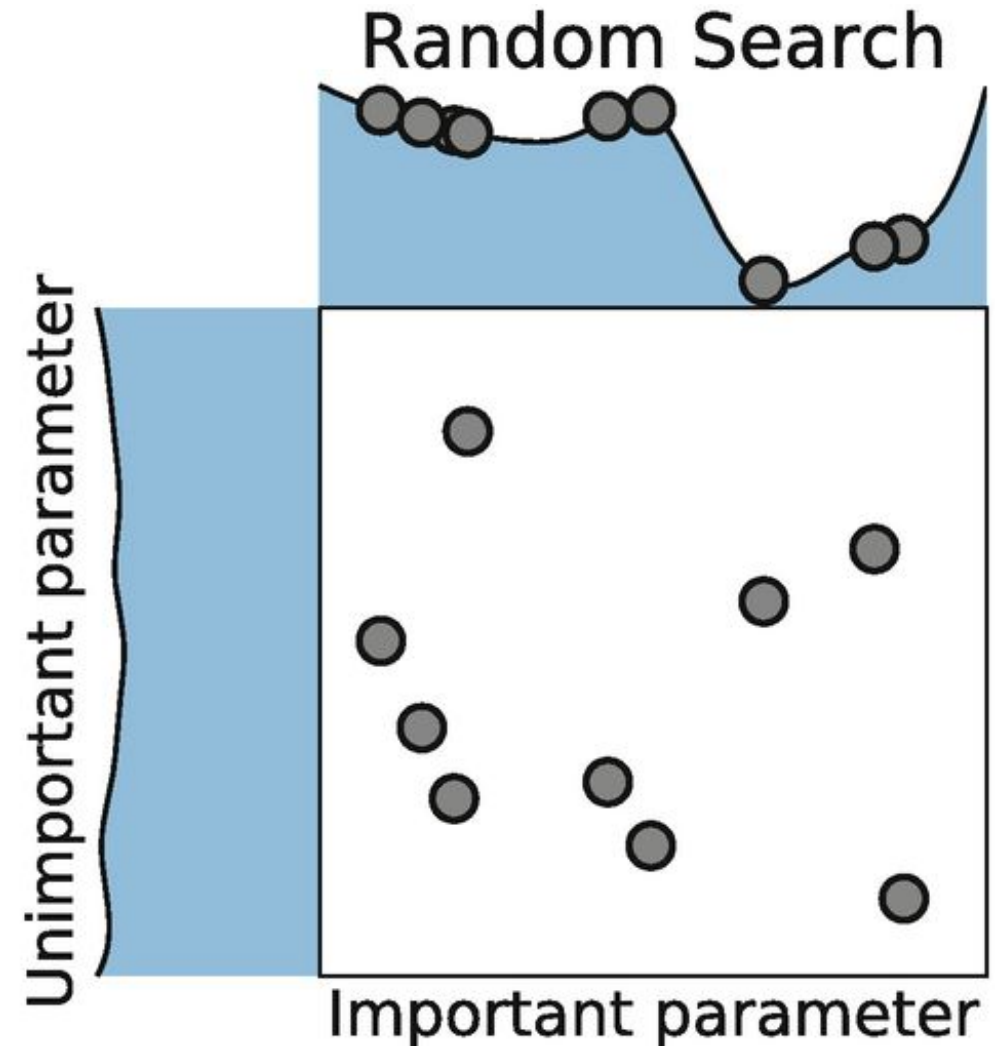


Es el algoritmo más costoso computacionalmente.

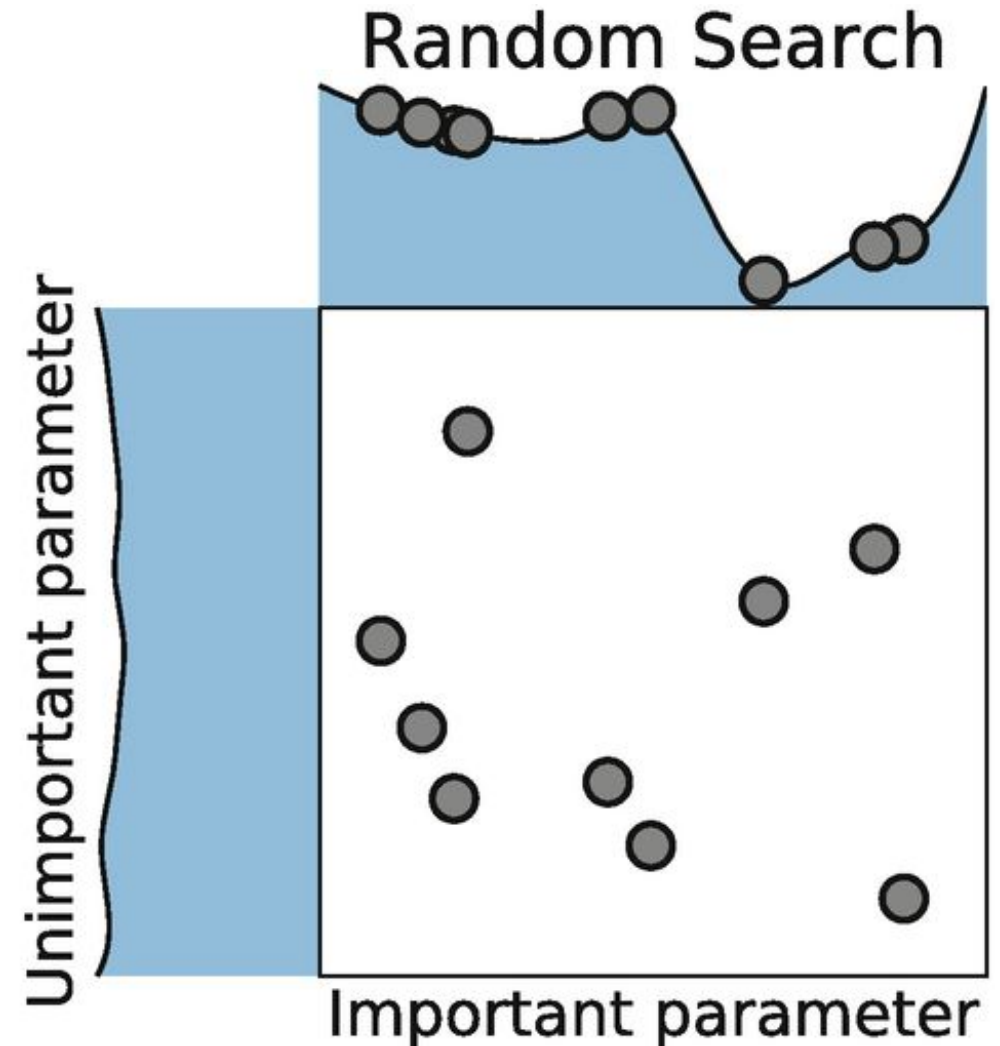
Sin embargo, es trivialmente paralelizable.



**Random Search** es una ligera variación de Grid Search. En lugar de buscar en toda la cuadrícula, Random Search sólo evalúa una muestra aleatoria de puntos de la cuadrícula. Esto hace que la búsqueda aleatoria sea mucho más económica computacionalmente que la búsqueda en la cuadrícula.



Bergstra y Bengio mostraron que en muchas ocasiones Random Search performa de forma similar a Grid Search. Estimaron que probar 60 puntos random tomados de la grilla de hiperparámetros debería dar resultados cercanos al óptimo.



## Recapitulando:

- Presentamos la diferencia entre **parámetros** e **hiperparámetros**.
- Presentamos el split entre **train, validation y test sets**
- Presentamos el método de **cross-validation**
- Presentamos los algoritmos **Grid Search** y **Random Search** para la optimización de hiperparámetros

GRACIAS