

# SELECCIÓN DE MODELOS

## OPTIMIZACIÓN DE HIPERPARÁMETROS

DANA REYES - PABLO GONZÁLEZ - ANGÉLICA AGUDELO



# ÍNDICE

1 Introducción

2 Pasos previos

3 Búsqueda de hiperparámetros por grilla

4 Búsqueda de hiperparámetros aleatoria

5 Búsqueda de hiperparámetros con optimización bayesiana

6 Conclusiones

7 Ejercicio



# INTRODUCCIÓN

- Los hiperparámetros de un modelo son valores que deben especificarse de antemano y generalmente no se obtienen de los datos por lo que deben ser seleccionados por el científico de datos.
- Una correcta selección de dichos parámetros evita inconvenientes en el ajuste del modelo que se desee implementar
- Se presentan tres métodos distintos para realizar el ajuste de hiperparámetros que nos permitirán encontrar el conjunto más óptimo para cualquier modelo de aprendizaje supervisado o no supervisado.



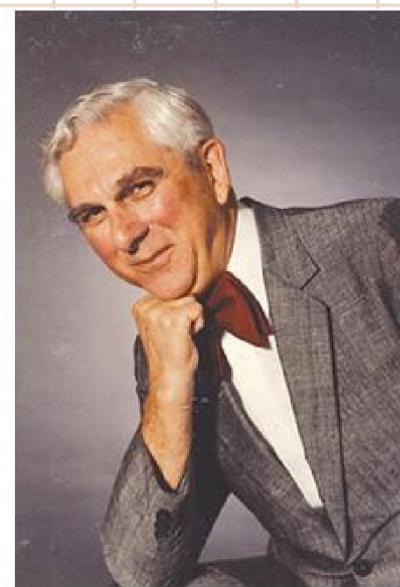
# NOTA HISTÓRICA

BÚSQUEDA DE  
GRILLA

BÚSQUEDA  
ALEATORIA

OPTIMIZACIÓN  
BAYESIANA

1953



R.L. ANDERSON

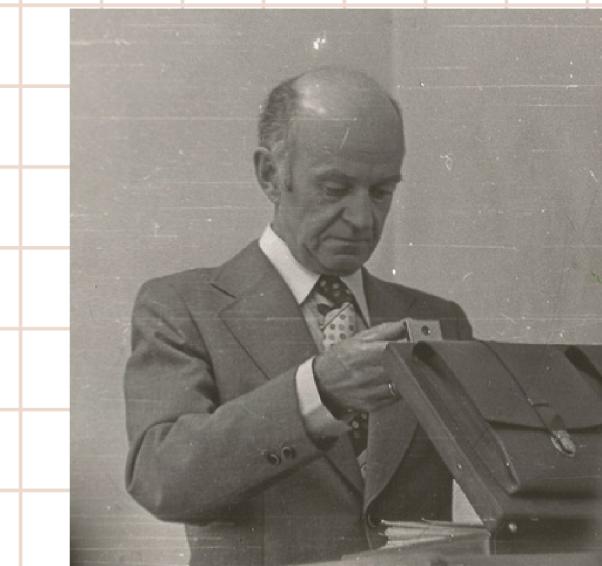
1964

L. A. RASTRIGIN

1965

J. MÁTYÁS

1970 - 1980



JONAS MOCKUS

# PASOS PREVIOS

1

## IMPORTAR DATOS

Vamos a utilizar un conjunto de datos de Kaggle para predecir la prima de seguro de vida.

2

## ENTRENAR MODELO BASE

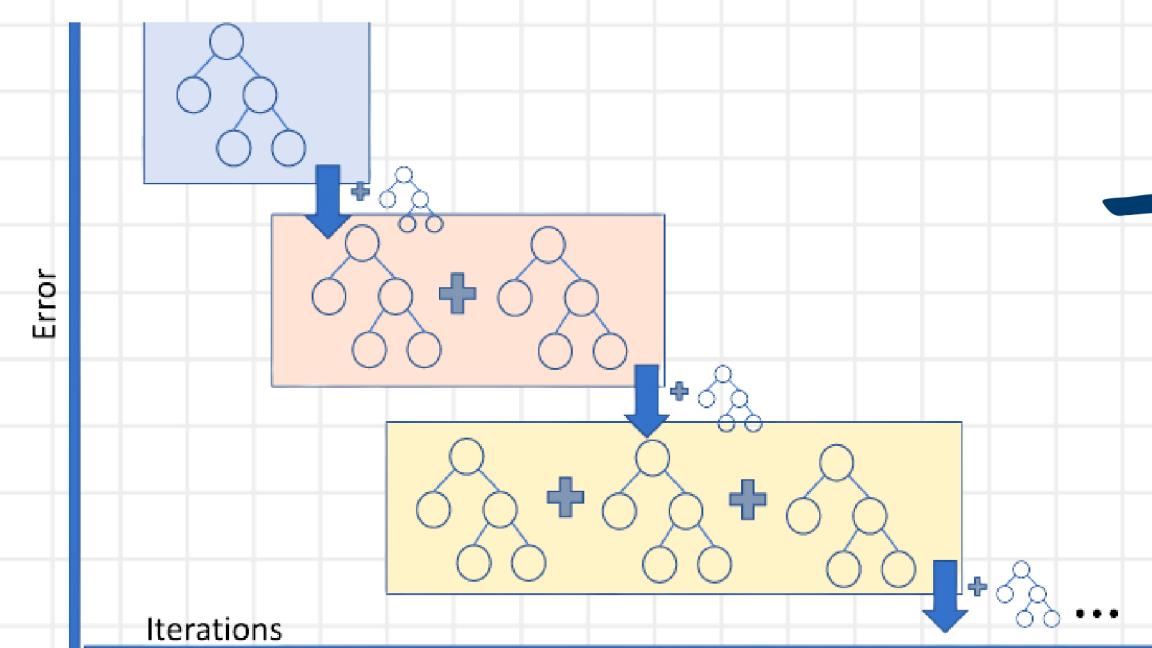
Entrenamos un modelo GradientBoostingRegressor con hiperparametros por defecto.

3

## EVALUAR MODELO BASE

Evaluamos el rendimiento del modelo base utilizando el coeficiente de determinación ( $R^2$ ).

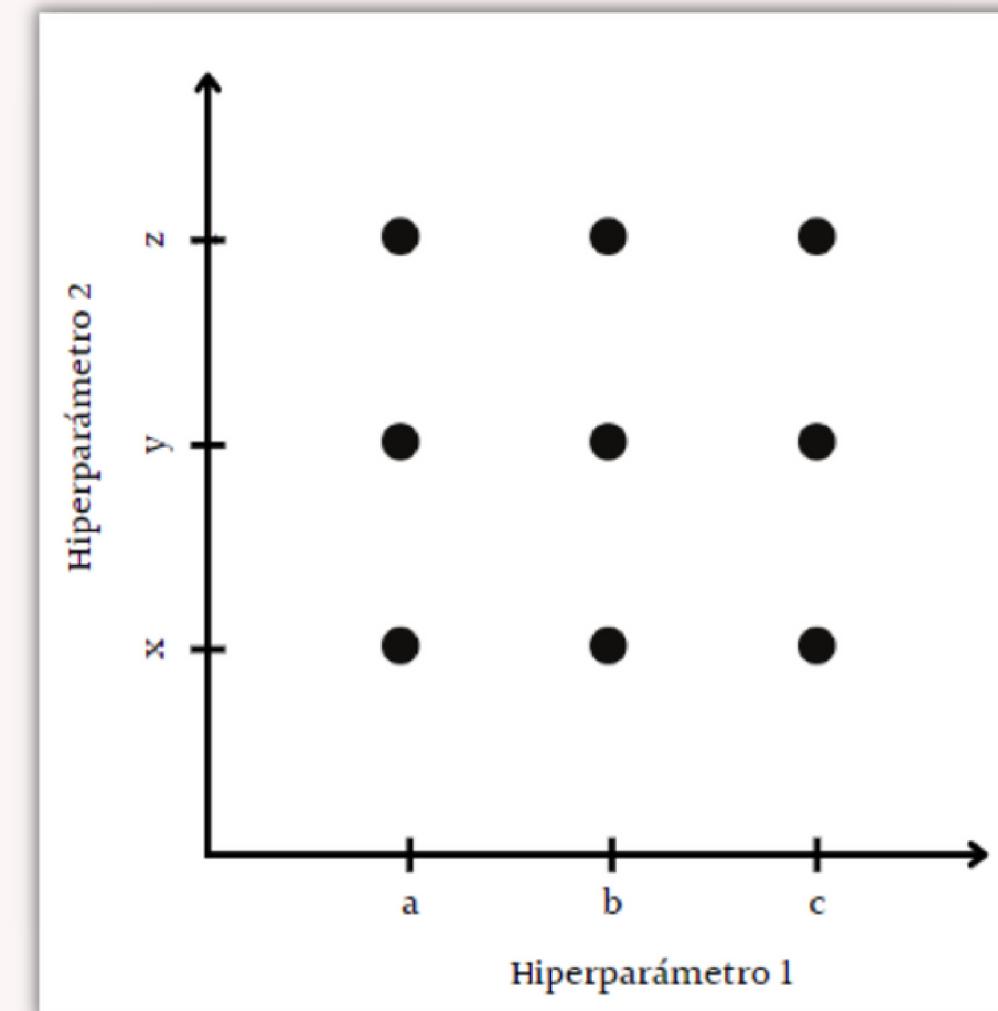
kaggle



$$R^2 = \frac{\sum_{t=1}^T (\hat{Y}_t - \bar{Y})^2}{\sum_{t=1}^T (Y_t - \bar{Y})^2}$$

# BÚSQUEDA DE GRILLA

- Conocido como búsqueda exhaustiva o en inglés "Grid Search" consiste en poner a prueba todas las posibles combinaciones de valores que se le proporcione en los parámetros.
- Esta prueba se realiza en puntos del espacio repartidos en forma de cuadrícula donde cada eje corresponde a los posibles valores de un hiperparámetro



# FUNDAMENTO MATEMÁTICO

Consideramos que se desean hallar  $m$  hiperparámetros

$$V = (v_1, v_2, \dots, v_m)$$

El método de búsqueda de grilla define un vector de límites inferiores  $a$  y un vector de límites superiores  $b$  para cada componente de  $V$

$$a = (a_1, a_2, \dots, a_m) \quad b = (b_1, b_2, \dots, b_m)$$

Se define una cantidad  $n$  de puntos contenidos en el intervalo  $[a_i, b_i]$  que serán equidistantes.

De esta manera se obtienen  $mn$  puntos en la cuadrícula y se calcula el modelo una vez para cada punto.

# EJEMPLO

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import ParameterGrid
param_grid = {'loss': ['squared_error', 'absolute_error', 'huber', 'quantile'],
              'learning_rate': [0.05, 0.1, 0.15],
              'n_estimators': [50, 100, 150],
              'min_samples_split': [2, 3],
              'min_samples_leaf': [1, 2],
              'max_depth': [1, 2],
              'max_features': ['sqrt', 'log2']}

from sklearn.ensemble import GradientBoostingRegressor
GBR = GradientBoostingRegressor()
grid_GBR = GridSearchCV(GBR, param_grid, refit = True, verbose = 0, n_jobs=-1)

X = data.iloc[:, :-1].values
Y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, shuffle=True, random_state=2)

grid_GBR.fit(X_train, y_train)
```

Los mejores parámetros encontrados en la búsqueda son:

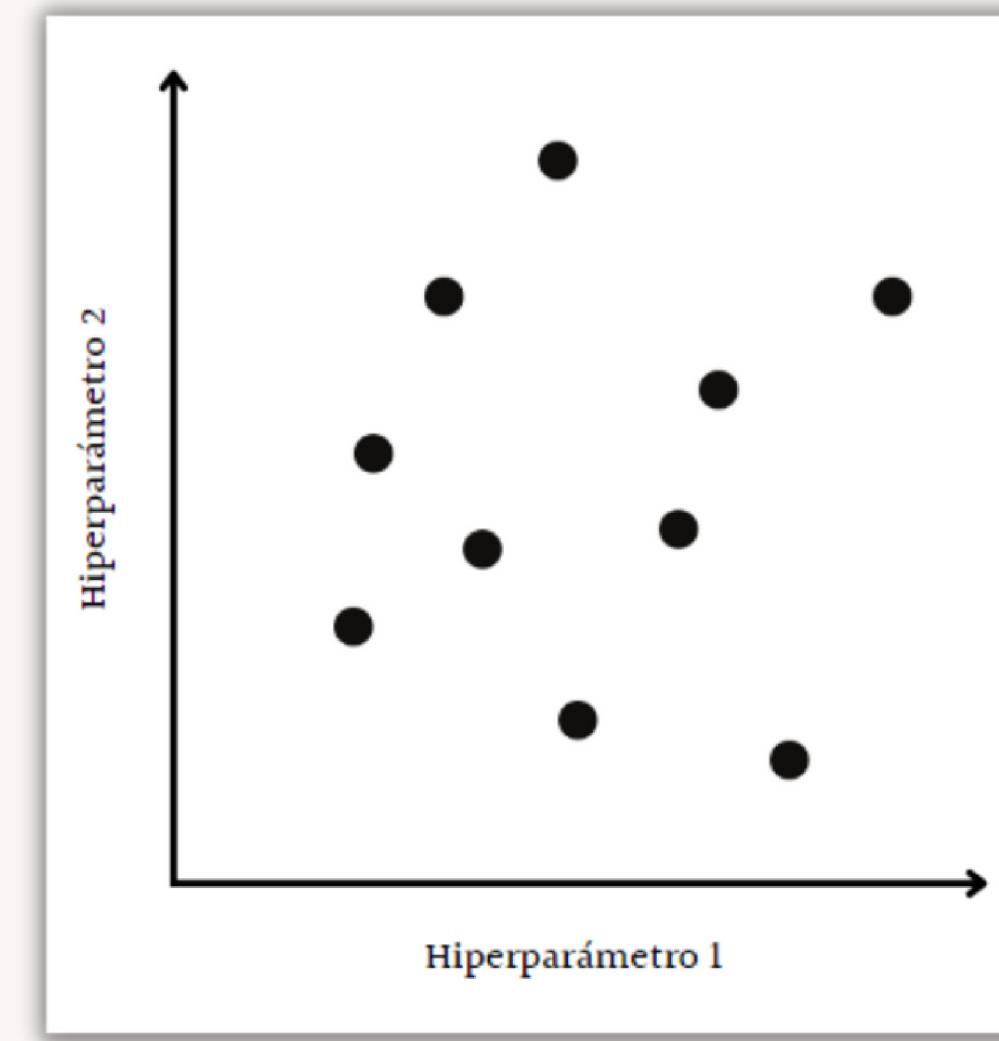
```
{'learning_rate': 0.15, 'loss': 'huber', 'max_depth': 2, 'max_features': 'log2', 'min_samples_
```

El coeficiente de determinación del modelo es:

```
0.8650136879445166
```

# BÚSQUEDA ALEATORIA

- Conocido en inglés como "Random Search" consiste en hacer una búsqueda aleatoria sobre los hiperparámetros, donde se sacan muestras independientes a partir de una distribución uniforme sobre los posibles valores para los parámetros.
- El espacio de configuración es el mismo que para la búsqueda de grilla.



# FUNDAMENTO MATEMÁTICO

Sea  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  la función a ser minimizada y sea  $x \in \mathbb{R}^n$  un valor candidato a ser solución en el espacio de búsqueda entonces:

- Se inicia  $x$  en una posición aleatoria en el espacio de búsqueda.
- Hasta que se cumplan un número de iteraciones o se cumpla un criterio de finalización
  - Se muestrea una nueva posición agregando un vector aleatorio con distribución normal a la actual posición de  $x$ .
  - Si  $f(y) < f(x)$  entonces la nueva posición de  $x$  es la de  $y$ .
- Ahora  $x$  ocupa la mejor posición encontrada.

# EJEMPLO

```
from sklearn.model_selection import RandomizedSearchCV
```

```
hyperparameters = dict(loss = ['squared_error', 'absolute_error', 'huber', 'quantile'],
                       learning_rate = [0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2],
                       min_samples_split = [2, 3, 4, 5],
                       min_samples_leaf = [1, 2, 3, 4],
                       max_depth = [1, 2, 3, 4, 5, 6, 7],
                       max_features = ['sqrt', 'log2'],
                       n_estimators = [20, 40, 60, 80, 100, 120, 140, 160, 180, 200])

clf = RandomizedSearchCV(reg, hyperparameters, random_state=0)

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

search = clf.fit(X_train, y_train)
parameters=search.best_params_
print('Mejores parámetros:', parameters)

Mejores parámetros: {'n_estimators': 180, 'min_samples_split': 4, 'min_samples_leaf': 3,
                     'max_features': 'sqrt', 'max_depth': 4, 'loss': 'huber', 'learning_rate': 0.2}
```

# ● EJEMPLO

```
from sklearn.model_selection import RandomizedSearchCV
```

```
reg = GradientBoostingRegressor(loss=parameters['loss'],
                                 learning_rate = parameters['learning_rate'],
                                 min_samples_split = parameters['min_samples_split'],
                                 min_samples_leaf = parameters['min_samples_leaf'],
                                 max_depth = parameters['max_depth'],
                                 max_features = parameters['max_features'],
                                 n_estimators = parameters['n_estimators'])

reg.fit(X_train, y_train)

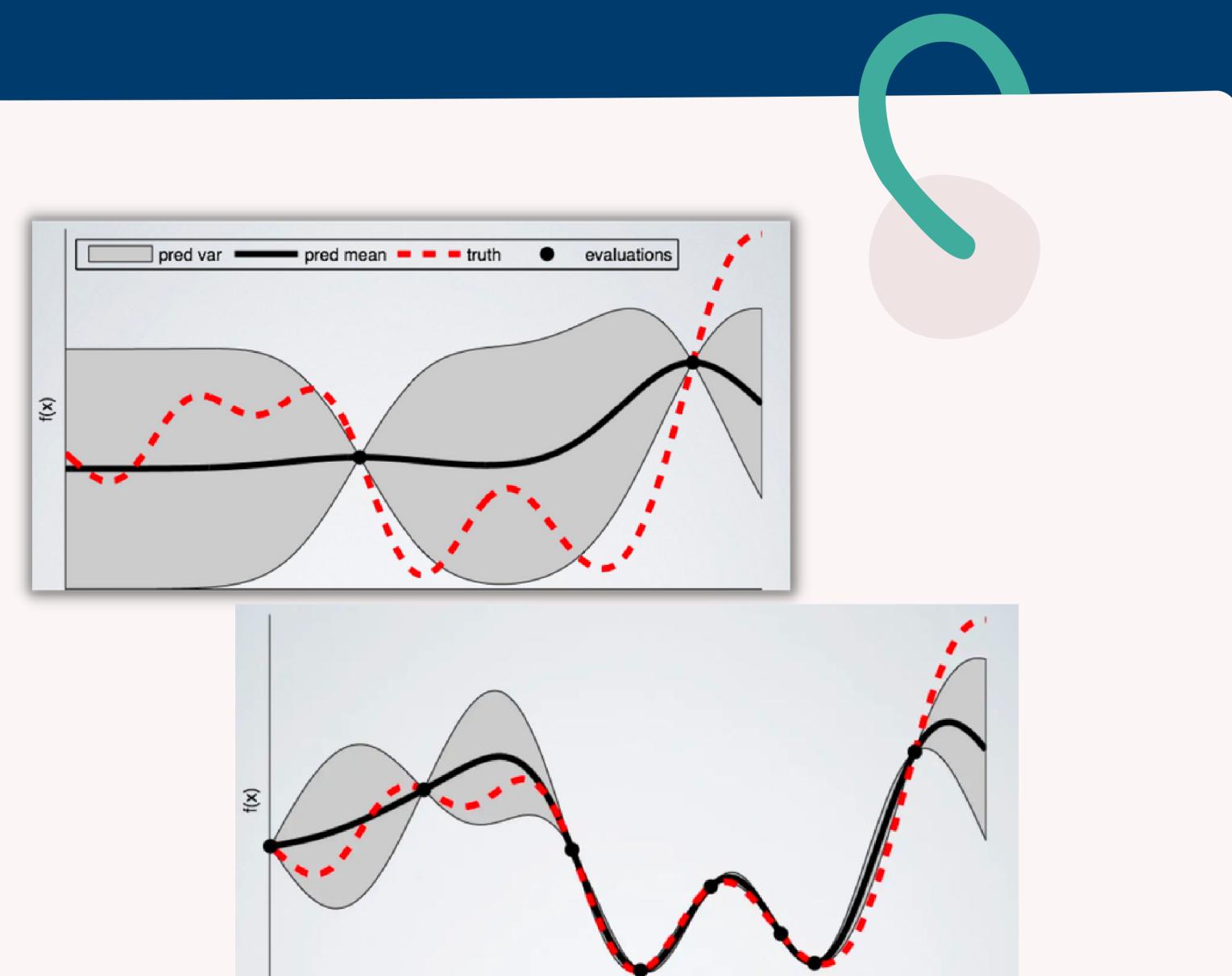
y_pred = reg.predict(X_test)

r2 = r2_score(y_test, y_pred)
print('R2:', r2)

R2: 0.7952619140358745
```

# OPTIMIZACIÓN BAYESIANA

La optimización bayesiana, tiene en cuenta las evaluaciones anteriores al elegir el conjunto de hiperparámetros para evaluar a continuación. Al elegir sus combinaciones de parámetros de manera informada, permite enfocarse en aquellas áreas del espacio de parámetros que cree que traerán los puntajes de validación más prometedores.



**Construir un modelo de probabilidad de la función objetivo y utilizarlo para seleccionar los hiperparámetros más prometedores para evaluar en la verdadera función objetivo.**

# FUNDAMENTO MATEMÁTICO

Sea  $f(x)$  la función objetivo, entonces definimos una función sustituta de la función objetivo como

$$p(score | \text{hyperparameters})$$

Esta aproximación es un modelo probabilístico que dado un conjunto de hiperparámetros, se evalúa la función objetivo en este conjunto y se retorna un score que es usado para actualizar la distribución del modelo probabilístico, entonces con cada iteración, este modelo probabilístico se vuelve un predictor mas robusto de scores.

La distribución se actualiza usando el teorema de Bayes, ya que se calcula de la siguiente forma:

$$p(score | \text{hyperparameters}) = \frac{p(\text{hyperparameters} | score) \cdot p(score)}{p(\text{hyperparameters})}$$

$$p(\text{hyperparameters} | score) = \begin{cases} l(x), & \text{if } \text{-score} < \text{-score threshold} \\ g(x), & \text{if } \text{-score} \geq \text{-score threshold} \end{cases}$$

Cada vez que un conjunto de hiperparámetros se evalúa en la función objetivo, se obtiene una mezcla gaussiana  $l(x)$  ó  $g(x)$  que depende del score obtenido.

# EJEMPLO

```
import optuna
```

```
def objective(trial):
    # set hyperparameters
    _loss = trial.suggest_categorical('loss', {'squared_error', 'absolute_error', 'huber'})
    _learning_rate = trial.suggest_float('learning_rate', low = 0.025, high = 0.2, step = 0.025)
    _min_samples_split = trial.suggest_int('min_samples_split', low = 2, high = 5, step = 1)
    _min_samples_leaf = trial.suggest_int('min_samples_leaf', low = 1, high = 4, step = 1)
    _max_depth = trial.suggest_int('max_depth', low = 1, high = 7, step = 1)
    _max_features = trial.suggest_categorical('max_features', {'sqrt', 'log2'})
    _n_estimators = trial.suggest_int('n_estimators', low = 20, high = 200, step = 20)

    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

    reg = GradientBoostingRegressor(loss=_loss,
                                    learning_rate = _learning_rate,
                                    min_samples_split = _min_samples_split,
                                    min_samples_leaf = _min_samples_leaf,
                                    max_depth = _max_depth,
                                    max_features = _max_features,
                                    n_estimators = _n_estimators)

    reg.fit(X_train, y_train)

    y_pred = reg.predict(X_test)

    r2 = r2_score(y_test, y_pred)

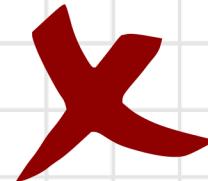
    return r2
```

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100, n_jobs=-1)
```

# CONCLUSIONES



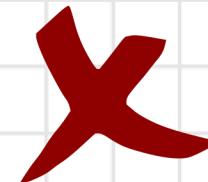
OPTIMIZACIÓN BAYESIANA ES EL MAS EFECTIVO



BUSQUEDA POR GRILLA PUEDE SER COMPUTACIONALMENTE COSTOSO



EN OCASIONES EL MODELO BASE ES LA MEJOR OPCIÓN



BUSQUEDA ALEATORIA NO GARANTIZA EXPLORAR TODAS LAS POSIBILIDADES



# ● EJERCICIO PROPUESTO

Ajusta un arbol de decisión (**DecisionTreeRegressor**) de scikit-learn al conjunto de datos usado en esta exposición, y utilizar el método de optimización bayesiana para tuning de hiperparámetros.

## ● OBJETIVO

Obtener un coeficiente de determinación ( $R^2$ ) mayor a 0.89

## DATOS

<https://www.kaggle.com/datasets/mirichoi0218/insurance>

# REFERENCIAS



- **Selección de modelos:**

[https://scikit-learn.org/stable/model\\_selection.html#model-selection](https://scikit-learn.org/stable/model_selection.html#model-selection)

- **Nota histórica:**

- [https://en.wikipedia.org/wiki/Random\\_search](https://en.wikipedia.org/wiki/Random_search)
- [https://en.wikipedia.org/wiki/Random\\_optimization](https://en.wikipedia.org/wiki/Random_optimization)
- [https://en.wikipedia.org/wiki/Bayesian\\_optimization](https://en.wikipedia.org/wiki/Bayesian_optimization)

# REFERENCIAS



- **Librerías:**
  - **Búsqueda de grilla:** [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
  - **Búsqueda aleatoria:** [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.RandomizedSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)
  - **Optimización bayesiana:** <https://optuna.readthedocs.io/en/stable/>