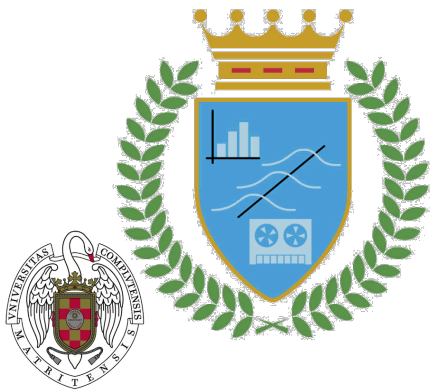
Bagging, Random Forest, Gradient Boosting y XGboost

MÁSTER EN BIG DATA,

DATA SCIENCE E INTELIGENCIA ARTIFICIAL



0. Recordatorio.

Ventajas de los árboles:

- Gran potencia descriptiva, se comprende muy bien el resultado. Resultados a menudo simples.
- Sobre todo en clasificación, pero también en regresión, se descubren interacciones y reglas muy difíciles de encontrar con otros métodos. Estas reglas se pueden utilizar como variables dummy para utilizar en otros métodos predictivos.
- Las relaciones no lineales no afectan tanto al comportamiento de los árboles como a otros métodos.
- No hay asunciones teóricas sobre los datos
- Aportan medidas de importancia de las variables
- Manera propia y eficiente de tratar los missings, incorporada al proceso
- Incorporación automática de interacciones, detectan relaciones por regiones que ningún otro método puede encontrar

0. Recordatorio.

Desventajas de los árboles:

- Poca fiabilidad y mala generalización: cada hoja es un parámetro y esto provoca modelos sobreajustados e inestables para la predicción. Añadir una variable nueva o un nuevo conjunto de observaciones puede alterar mucho el árbol.

- Complejidad en la construcción del árbol y casuística: dos plataformas (programas) diferentes dan dos árboles diferentes

- Poca eficacia predictiva, sobre todo en regresión: toscos en los valores de predicción. Por ello raramente son el modelo final. Se suelen usar como apoyo a otros modelos.

0. Recordatorio.

- Afortunadamente, se ha conseguido mantener las ventajas de los árboles mejorando el poder predictivo mediante la combinación de muchos árboles:
  - Bagging.
  - Random Forest.
  - Gradient Boosting..

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

❖**Bagging**

Dados los datos de tamaño N,

1) Repetir *m* veces i) y ii):

(i) Seleccionar N observaciones con reemplazamiento de los datos originales.

(ii) Aplicar un árbol y obtener predicciones para todas las observaciones originales N.

2) Agregar las *m* predicciones obtenidas en el apartado 1)

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

- Dependiendo del tamaño del conjunto de datos, habrá que ensamblar más o menos árboles, *m*.

- Hay que tomar distintos datos de entrada.

- Aunque el tamaño original, N (número de filas del conjunto de datos), y el tamaño de observaciones seleccionadas es el mismo, NO TIENEN POR QUÉ SER LOS MISMOS DATOS: **selección con reemplazamiento**, puedo tener observaciones repetidas.

- Si algo se repite, gana importancia. La gran sensibilidad de los árboles provoca que, ante el cambio de pesos de los distintos individuos, varíe el resultado.

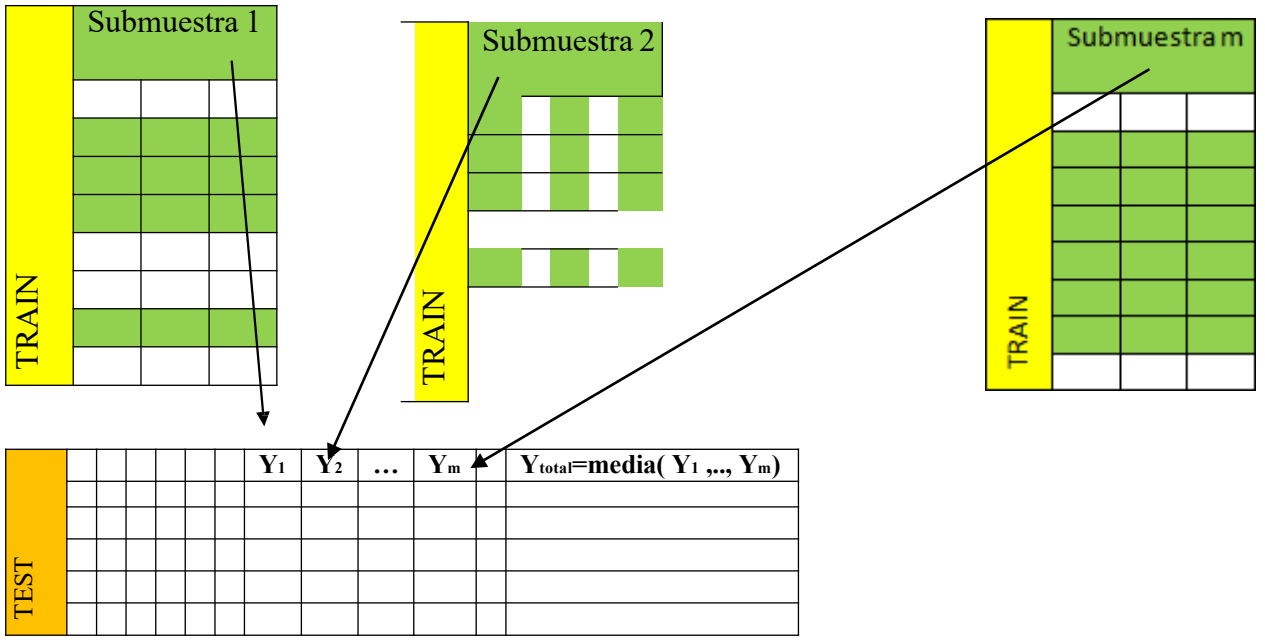
1. Bagging (Bootstrap Averaging) (Breiman, 1996).

- El resultado del bagging será la agregación de las *m* predicciones: en general **promedio del valor o de la probabilidad**

- **Filosóficamente,** este proceso de ensamblado se podría hacer para cualquier tipo de modelos, pero se ha demostrado que sólo es útil con árboles.

- **CUIDADO:** con la complejidad computacional. Para ahorrar tiempo, es importante tener ajustado **el número mínimo de observaciones en una hoja** en árboles, para ahorrarse esas pruebas en bagging.

1. Bagging (Bootstrap Averaging) (Breiman, 1996).



- Con cada submuestra se genera un modelo con el que se predicen los datos test. La predicción final será la media (u otra agregación) de las *m* diferentes predicciones.

- Al utilizarse diferentes submuestras, se reduce la dependencia de la estructura de los datos completos para construir el modelo, y como consecuencia se reduce la varianza del modelo (y a menudo también el sesgo).

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

Dados los datos de tamaño N,

1) Repetir *m* veces i) y ii):

(i) Seleccionar N observaciones con reemplazamiento de los datos originales.

(ii) Aplicar un árbol y obtener predicciones para todas las observaciones originales N.

2) Agregar las *m* predicciones obtenidas en el apartado 1)

- El apartado (i) admite todo tipo de variaciones: tomar n<N (para ganar velocidad computacional) **con** o **sin** reemplazamiento, estratificación, etc. **CONSEJO:** probar con un modelo grande, una vez, ver cuánto tardaría, y calcular si es factible mantener el tamaño en este tipo de métodos.

- En cuanto a (ii), la complejidad del árbol a utilizar es un tema a debate. Inicialmente se propuso árboles débiles (pocas hojas finales) y muchas iteraciones *m.* Pero en algunas versiones se utilizan árboles desarrollados hasta el final sin prefijar el número de hojas o profundidad.

- Si se trata de un problema de clasificación, dos estrategias pueden ser utilizadas:
  a) Promediar las probabilidades estimadas y obtener una clasificación a partir de un punto de corte.
  b) Clasificar en cada iteración y asignar a cada observación la clasificación mayoritaria entre todas las iteraciones (majority voting).

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

- La idea del submuestreo es **controlar el sobreajuste** implícito en la selección rígida de variables y estimación fija de parámetros que caracteriza a los métodos directos.

- En general, bagging funciona bien:

1. Cuando los modelos no están claros (muchas variables con relación débil pero estable con la variable dependiente, multiplicidad de modelos-opciones).

2. Cuando existen relaciones no lineales (regresión) o separaciones no lineales (clasificación)

3. Cuando existen interacciones ocultas, muchas variables categóricas, etc.

OJO: no tiene sentido aplicar bagging cuando hay POCAS variables.

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

❖PRINCIPALES PARÁMETROS A CONTROLAR EN BAGGIN.

❖ Para obtener óptimos resultados, es necesario tunear los parámetros del bagging. Los parámetros más importantes son:

- El tamaño de las muestras m y si se va a utilizar bootstrap (con reemplazo) o sin reemplazamiento. Si el número de observaciones es pequeño, mejor utilizar con reemplazamiento. Si es grande, es indiferente pues el resultado con o sin reemplazamiento es muy similar.
- El número de iteraciones m a promediar (no importante, no se produce sobreajuste por demasiadas iteraciones y en general a partir de un cierto momento temprano ya no se mejora nada)

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

❖PRINCIPALES PARÁMETROS A CONTROLAR EN BAGGIN.

❖ Para obtener óptimos resultados, es necesario tunear los parámetros del bagging. Los parámetros más importantes son:

➢ Características de los árboles. Son bastante influyentes:

- El número de hojas final o, en su defecto, la profundidad del árbol
- El max_Depth -en Python, con sklearn- (número de divisiones máxima en cada nodo. Por defecto se dejará en 2, árboles binarios)
- El número de observaciones mínimo en una rama-nodo. Se puede ampliar para evitar sobreajuste (reducir varianza) o reducir para ajustar mejor (reducir sesgo).
- Otros, como el parámetro de complejidad (que no suele estar incluido en muchos paquetes), p-valor, etc.

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

❖PRINCIPALES PARÁMETROS A CONTROLAR EN BAGGIN.

❖ CONTROLAR LA ALEATORIEDAD → La reproducibilidad es fundamental.

Cuando se plantean modelos en machine learning se usan a menudo técnicas de remuestreo para evaluar bien los modelos (training-test, validación cruzada, etc.). Estas técnicas conllevan sorteos, ordenaciones aleatorias, etc. Para ello se utilizan semillas (seed) de aleatorización.

Si el proceso depende de una sola semilla que se suele poner al principio del código, no hay problema con la reproducibilidad del método.

También hay que recordar que la semilla de aleatorización no es un parámetro del modelo, la usamos como control y podemos jugar con ella, variándola como hacemos en validación cruzada repetida, para observar la sensibilidad del modelo y sus errores ante un esquema de selección de observaciones ligeramente diferente.

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

- PRINCIPALES PARÁMETROS A CONTROLAR EN BAGGIN.

- **Out of Bag**

- Las **instancias que no se incluyen en la muestra de entrenamiento de un modelo específico** se consideran como instancias "out-of-bag" (fuera de la bolsa) para ese modelo en particular. Estas instancias OOB proporcionan una evaluación adicional del rendimiento del modelo, ya que no se utilizaron durante su entrenamiento. Por lo tanto, se pueden utilizar para estimar la precisión fuera de la muestra del modelo sin la necesidad de un conjunto de prueba adicional.

- En resumen, el "out-of-bag" (OOB) es un conjunto de muestras que no se incluyen en la muestra de entrenamiento de un modelo específico en Bagging y se utiliza para evaluar el rendimiento del modelo en instancias no vistas durante su entrenamiento (conociendo los errores cometidos en esos casos).

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

```
# Crear el modelo de Bagging
# n_estimators int, default=10: Número de modelos a aplicar.
# max_samples int or float, default=1.0: número de valores máximos a extraer para cada modelo.
# max_features int or float, default=1.0: número de variables a utilizar para cada modelo.
# boostrap bool, default=True: con o sin reemplazo aplicado a las observaciones.
# bootstrap_features bool, default=False: con o sin reemplazo aplicado a las variables.
# oob_score bool, default=False: Out of Bag. model.oob_score_ devuelve un error medio #cometido en los casos fuera de la bolsa. Para todos los errores: #model.oob_decision_function_
bagging_model = BaggingClassifier(modelo_base, max_samples = 300, max_features = 9,n_estimators=250, random_state=123, oob_score = True)
bagging_model.fit(X_train, y_train)
```

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

EJERCICIO: Con el código anterior, y la BBDD “arboles.csv”.

(1) Crear un modelo de árboles (sin CV ni GridSearchCV, sólo ilustrativo).
(2) Crear un modelo de bagging basado en el anterior.
(3) Aplicar un GridSearchCV con validación cruzada.

Comparar accuracy de los tres modelos.

Téngase en cuenta que, - dado el carácter ilustrativo – al no buscar los parámetros óptimos del primer árbol, no se garantiza la obtención del mejor modelo ni en (2) ni en (3).

Sin embargo, no tener el mejor modelo en (1) enfatiza aún más la mejora en (2) y en (3).

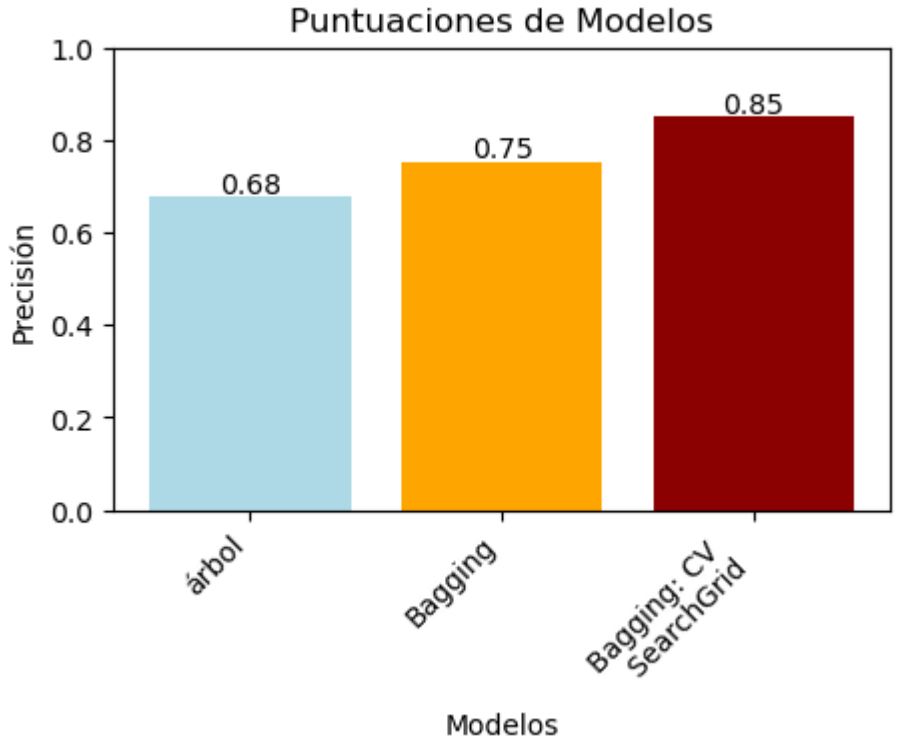
1. Bagging (Bootstrap Averaging) (Breiman, 1996).

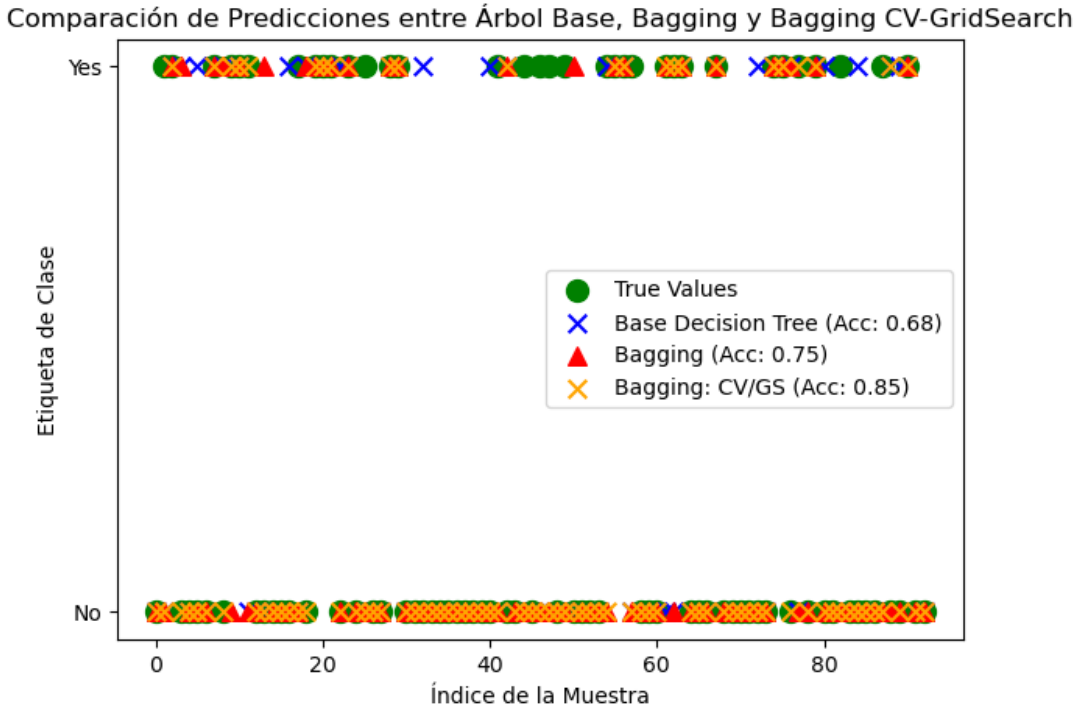
EJERCICIO: Con el código anterior, y la BBDD “arboles.csv”.

```
DecisionTreeClassifier(min_samples_split=10, criterion='gini', max_depth = 5)
BaggingClassifier(base_classifier, max_samples = 300, max_features = 9,n_estimators=250, random_state=123, oob_score = True)
param_grid = {
    'n_estimators': [10, 50, 100,250],
    'max_samples': [1,75,150,300],
    'max_features': [1,4,7,9],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False]
}
GridSearchCV(bagging_model, param_grid, cv=5, scoring='accuracy')
```

1. Bagging (Bootstrap Averaging) (Breiman, 1996).

EJERCICIO: Con el código anterior, y la BBDD “arboles.csv”.





1. Bagging (Bootstrap Averaging) (Breiman, 1996).

- Bagging soluciona gran parte del problema de sobreajuste de los árboles **pero** tiene un problema importante: a veces es complicado **obtener modelos nuevos**. El remuestreo no siempre garantiza que el modelo obtenido vaya a cambiar.

- Según los datos disponibles, si hay algún subconjunto de variables muy predominantes, siempre se obtienen los mismos árboles: estas variables importantes van a hacer las particiones de las primeras ramas, que siempre van a quedar iguales.

2. Random Forest (Breiman, 2001).

- Random Forest es un algoritmo de combinación de árboles predictores, ofreciendo un único ouput fruto de la combinación de estos.

- Es una modificación del Bagging que consiste en incorporar aleatoriedad en las variables utilizadas para segmentar cada nodo del árbol. Mientras que los árboles de decisión consideran todas las divisiones posibles de características, random forest solo seleccionan un subconjunto de esas características, encontrando aquí la principal diferencia.

- Random Forest es una técnica de agregación que mejora la predicción con respecto a otros métodos mediante la inclusión de aleatorización en la construcción de las distintas estructuras.

2. Random Forest (Breiman, 2001).

Es una modificación del bagging que consiste en incorporar aleatoriedad en las variables utilizadas para segmentar cada nodo del árbol.

❖ **Random Forest**

Dados los datos de tamaño N,

1) Repetir *m* veces i), ii), iii):

(i) Seleccionar N observaciones con reemplazamiento de los datos originales

(ii) Aplicar un árbol de la siguiente manera:

❖ **En cada nodo, seleccionar *p* variables de las *k* originales y de las *p* elegidas, escoger la mejor variable para la partición del nodo. (iii) Obtener predicciones para todas las observaciones originales N**

2) Promediar las *m* predicciones obtenidas en el apartado 1)

2. Random Forest (Breiman, 2001).

La aleatoriedad de características, también conocida como "feature bagging", genera un subconjunto aleatorio de características, lo que garantiza una baja correlación entre los árboles de decisión empleados.

- Random Forest tiene una muy **buena capacidad explicativa:** no es tan caja negra
- **Diferencia con bagging:** bagging coge la mejor variable de entre todas;
- **Random forest** escoge la mejor variable de entre un subconjunto ***p***
- **Esto garantiza la obtención de nuevos modelos**
- ***p*** pequeño para aliviar los problemas de bagging (más variabilidad de variables).

2. Random Forest (Breiman, 2001).

- El algoritmo Random Forest da un paso más en soslayar el problema de selección de variables, evitando decidirse rígidamente por un set de variables y aprovechando a la vez las ventajas del bagging.
- Se trata de **incorporar dos fuentes de variabilidad** (remuestreo de observaciones y de variables) para ganar en capacidad de generalización, y **reducir el sobreajuste** conservando a la vez la facultad de ajustar bien relaciones particulares en los datos (interacciones, no linealidad, cortes, problemas de extrapolación, etc.)
- **Random Forest evita también el problema de variables predictoras muy dominantes**. En bagging, si un par de variables muy dominantes los árboles serían todos muy parecidos.

2. Random Forest (Breiman, 2001).

Aquí el pseudoalgoritmo del Random Forest:

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

2. Random Forest (Breiman, 2001).

❖**Principales parámetros a controlar en Random Forest (sklearn).**

- El tamaño o % de las muestras **n** y si se va a utilizar bootstrap (con reemplazo) o sin reemplazamiento.
- El número de iteraciones **m** a promediar
- El número de variables **p** a muestrear en cada nodo (si es igual al número inicial de variables **k** el Random Forest es equivalente al Bagging. Dicho de otra manera, **Bagging es un caso particular de Random Forest**). **p** puede ser un porcentaje o un número.

2. Random Forest (Breiman, 2001).

❖**Principales parámetros a controlar en Random Forest (sklearn).**

- Parámetros de los árboles. Son bastante influyentes:
  - El número de hojas final o, en su defecto, la profundidad del árbol
  - El maxbranch (número de divisiones máxima en cada nodo. Por defecto se dejará en 2, árboles binarios)
  - El p-valor para las divisiones en cada nodo. Más alto, árboles menos complejos (**más sesgo, menos varianza**, es más exigente en la clasificación)
  - El número de observaciones mínimo en una rama-nodo. Se puede ampliar para evitar sobreajuste (reducir varianza) o reducir para ajustar mejor (reducir sesgo).

2. Random Forest (Breiman, 2001).

❖ **Principales parámetros a controlar en Random Forest (sklearn).**

from sklearn.ensemble import RandomForestClassifier

 # n_estimators: int, default=100. Número de árboles en el bosque.

 # criterion{“gini”, “entropy”, “log_loss”}, default=”gini”

 # max_depth = Profundidad máxima del árbol. En caso de no especificar, el clasificador sigue segmentando hasta que las hojas son puras, o se alcanza el min_sample_split. Con caracter ilustrativo, se selecciona bajo.

 # min_samples_split: int or float, default=2. Mínimo número de casos para dividir un nodo interno.

 # min_samples_leaf: int or float, default=1. Mínimo número de casos pertenecientes a una hoja-nodo.

 # min_weight_fraction_leaf: float, default=0.0. Un valor de peso o importancia mínima a establecer para determinadas clases. Es especialmente útil cuando se tienen clases desbalanceadas. Una fracción mínima ponderada de las muestras.

2. Random Forest (Breiman, 2001).

❖ **Principales parámetros a controlar en RandomForestClassifier()**

criterion{“gini”, “entropy”, “log_loss”}, default=”gini”

n_estimators: *int, default=100*

max_Depth: *int, default=None*

min_samples_Split: *int or float, default=2*

min_samples_leaf: *int or float, default=1*

max_features: *{“sqrt”, “log2”, None}, int or float, default=1.0*

max_leaf_nodes: *int, default=None*

Bootstrap: *bool, default=True*

random_state

class_weight: {“balanced”, “balanced_subsample”}, dict or list of dicts, default=None

ccp_alphanon-negative float, default=0.0

max_samples: int or float, default=None

2. Random Forest (Breiman, 2001).

❖ **Principales parámetros a controlar en RandomForestRegressor()**

Criterion: {“squared_error”, “absolute_error”, “friedman_mse”, “poisson”}, default=”squared_error”

n_estimators: *int, default=100*

max_Depth: *int, default=None*

min_samples_Split: *int or float, default=2*

min_samples_leaf: *int or float, default=1*

max_features: *{“sqrt”, “log2”, None}, int or float, default=1.0*

max_leaf_nodes: *int, default=None*

Bootstrap: *bool, default=True*

random_state

ccp_alphanon-negative float, default=0.0

max_samples: int or float, default=None

2. Random Forest (Breiman, 2001).

❖**EJERCICIO:**

Con la metodología aplicada anteriormente, aplicar a la BB.DD. El algoritmo de random forest para un problema de clasificación y para un problema de regresión.

- Probar distintos parámetros en GridSearchCV y observar detenidamente la performance para cada una de las soluciones. Puede haber casos que interese apostar por una mayor sensibilidad, o especificidad…

- El mejor modelo es aquel que, no sólo presenta un buen ajuste, sino que se comporta de acuerdo a los criterios de la tarea concreta.

2. Random Forest (Breiman, 2001).

❖ EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
df = pd.read_csv(file_path_2)

print(df.head())

print(f'\nLa frecuencia de cada clase es: \n{df.chd.value_counts(normalize=True)}')
```

```
   sbp  tobacco   ldl  adiposity  famhist  typea  obesity  alcohol  age  chd
0  160    12.00  5.73      23.11  Present     49    25.30    97.20   52    1
1  144     0.01  4.41      28.61   Absent     55    28.87     2.06   63    1
2  118     0.08  3.48      32.28  Present     52    29.14     3.81   46    0
3  170     7.50  6.41      38.03  Present     51    31.99    24.26   58    1
4  134    13.60  3.50      27.78  Present     60    25.99    57.34   49    1

La frecuencia de cada clase es:
0    0.65368
1    0.34632
```

```
#es importante tratar de forma adecuada las variables categóricas. Se convierten en numéricas con la regla: one hot encoding.

df[['famhist']] = pd.get_dummies(df[['famhist']],drop_first=True)

# Separar las variables predictoras y la variable de respuesta.

X = df.drop('chd', axis=1)

y = df['chd']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

```
La frecuencia de cada clase en train es:
No     0.655827
Yes    0.344173
Name: chd, dtype: float64

La frecuencia de cada clase en test es:
No     0.645161
Yes    0.354839
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
RF_model = RandomForestClassifier(n_estimators = 60,bootstrap = True, max_depth = 20, min_samples_split=10, criterion='entropy',min_samples_leaf = 10,random_state=123)

RF_model.fit(X_train, y_train)

y_pred_rf = RF_model.predict(X_test)


# Evaluar el rendimiento del modelo

accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f'Precisión del modelo con RF estándar: {accuracy_rf}')
```

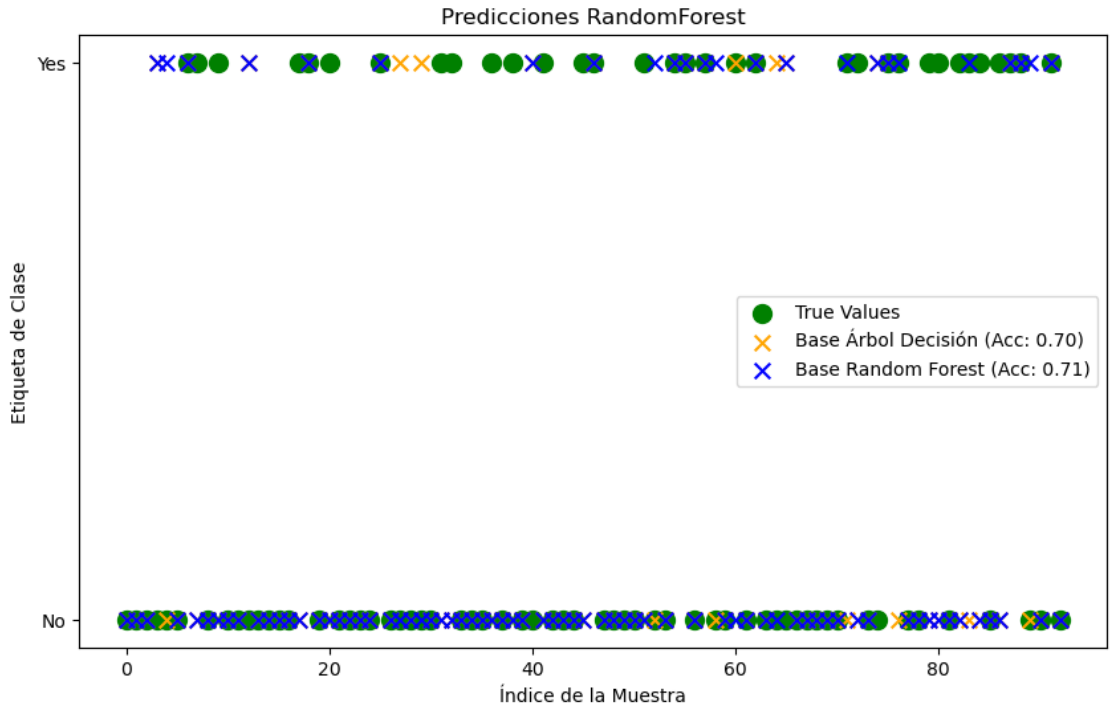
```
Precisión del modelo con RF estándar: 0.7096774193548387
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

Se puede entrenar un árbol de decisión para comprar y representar los resultados predichos:

```
# Crear un gráfico de dispersión para comparar las predicciones
plt.figure(figsize=(10, 6))


plt.scatter(np.arange(len(y_test)), y_test, color='green', label='True Values', marker='o', s=100)
plt.scatter(np.arange(len(y_test)), y_pred_base, color='orange’,
 label=f'Base Árbol Decisión (Acc: {accuracy_a:.2f})', marker='x', s=70)
plt.scatter(np.arange(len(y_test)), y_pred_rf, color='blue’,
 label=f'Base Random Forest (Acc: {accuracy_rf:.2f})', marker='x', s=70)


plt.title('Predicciones RandomForest')
plt.xlabel('Índice de la Muestra')
plt.ylabel('Etiqueta de Clase')
plt.legend()
plt.show()
```



2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

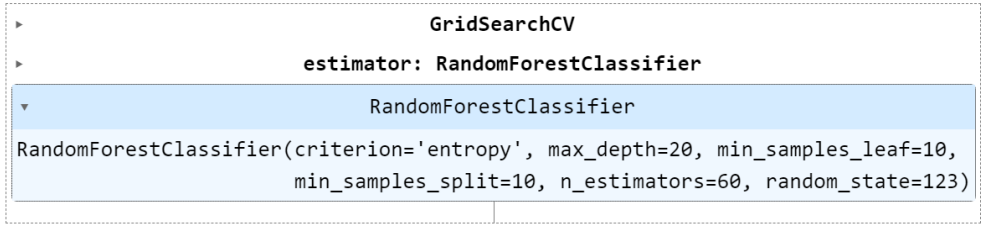
```
# se procede a observar el posible sobreajuste comparando predicciones en train y test.

# predicciones significativamente mayores en train que en test puede indicar sobreajuste.

# Predicciones en conjunto de entrenamiento y prueba

y_train_pred = RF_model.predict(X_train)

y_test_pred = RF_model.predict(X_test)

print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred)}')

print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred)}')

print('Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.')
```

```
Se tiene un accuracy para train de: 0.8346883468834688
Se tiene un accuracy para test de: 0.7096774193548387
Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
# Se puede seleccionar un grill extenso y meditado sobre los distintos parámetros con los que jugar.
params = {
    'n_estimators' : [50,100,150,200,250],
    'max_depth': [2, 3, 5, 10, 20],
    'bootstrap': [True, False],
    'min_samples_leaf' : [3,10,30],
    'min_samples_split': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
scoring_metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
#recordar que arbol2 es el árbol cuyas VI son todas las variables.
# cv = crossvalidation
grid_search_RF = GridSearchCV(estimator=RF_model,
                              param_grid=params,
                              cv=4, scoring = scoring_metrics, refit='accuracy')
grid_search_RF.fit(X_train, y_train)
```



En el caso anterior, se observa que el parámetro max_depth tiene un valor de 20, indicador que de mayores valores pueden suponer mejora, por lo que modificar el grill puede ser indicado en ese parámetro. Igualmente puede ocurrir con min_samples_split, el cual, parece indicar la búsqueda de valores menores y más continuados, por ejemplo: [8,9,10,11,12]

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
# se procede a observar el posible sobreajuste comparando predicciones en train y test.

# predicciones significativamente mayores en train que en test puede indicar sobreajuste.

# Predicciones en conjunto de entrenamiento y prueba

y_train_pred = best_model_RF.predict(X_train)

y_test_pred = best_model_RF.predict(X_test)

print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred)}')

print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred)}')

print('Comprobar que la diferencia no sea muy grande por temas de sobreajuste')
```

```
Se tiene un accuracy para train de: 0.7886178861788617
Se tiene un accuracy para test de: 0.6989247311827957
Comprobar que la diferencia no sea muy grande por temas de sobreajuste
```

2. Random Forest (Breiman, 2001).

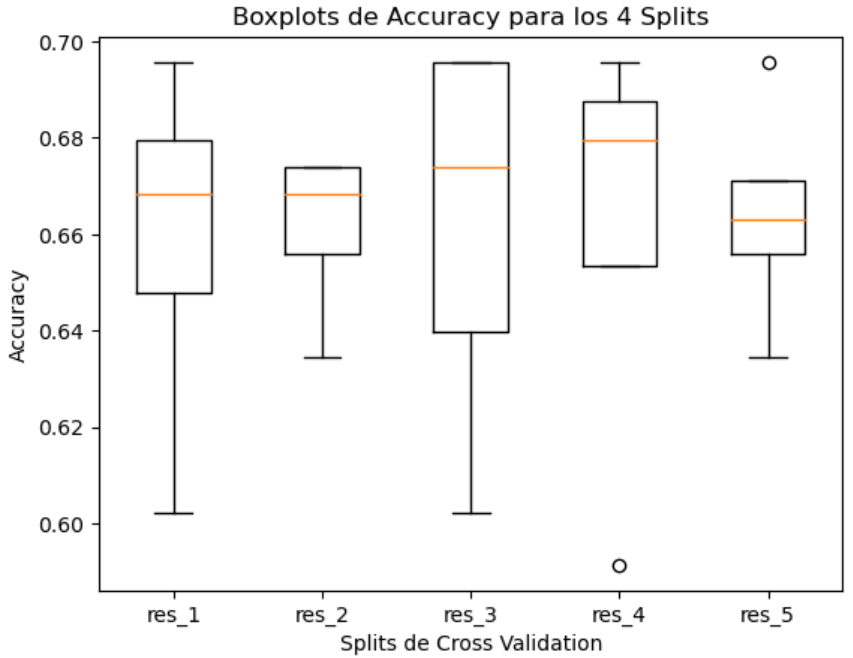
❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
# Obtener resultados del grid search

results = pd.DataFrame(grid_search_RF.cv_results_)

results.head()

# Ordenar el DataFrame por la métrica de interés (por ejemplo, accuracy)

sorted_results = results.sort_values(by='mean_test_accuracy', ascending=True).head(5)

print(sorted_results)

# se selecciona el modelo candidato, y se procede a analizar su robustez a lo largo de cross validation.

res_1 = sorted_results[['split0_test_accuracy', 'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy']].iloc[0]

res_2 = sorted_results[['split0_test_accuracy', 'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy']].iloc[1]

res_3 = sorted_results[['split0_test_accuracy', 'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy']].iloc[2]

res_4 = sorted_results[['split0_test_accuracy', 'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy']].iloc[3]

res_5 = sorted_results[['split0_test_accuracy', 'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy']].iloc[4]
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en clasificación:

```
# Crear un boxplot para los cuatro valores de accuracy

plt.boxplot([res_1.values,res_2.values,res_3.values,res_4.values,res_5.values], labels = ['res_1','res_2','res_3','res_4','res_5'])

plt.title('Boxplots de Accuracy para los 4 Splits')

plt.xlabel('Splits de Cross Validation')

plt.ylabel('Accuracy')

plt.show()

# seleccionemos el segundo modelo dada su mayor robustez con respecto al propuesto por GridSearch

# nótese que "**" es para desempaquetar una lista de valores.

random_f_2 = RandomForestClassifier(**sorted_results['params'].iloc[1],random_state=123)

random_f_2.fit(X_train, y_train)

res_rf_2 = random_f_2.predict(X_test)

accuracy_score(y_test,res_rf_2) # 0.698

# Si se quiere conocer quién tiene mayor robustez en sensibilidad por cuestiones de criterio exógeno:

sorted_results[['std_test_recall_macro']]
```



2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en regresión:

```
compress = pd.read_csv(file_path_3)

compress.head()

# Separar las variables predictoras y la variable de respuesta.

X_c = compress.drop('cstrength', axis=1)

y_c = compress['cstrength']


RF_R = RandomForestRegressor(n_estimators = 60,bootstrap = True, max_depth = 20, min_samples_split=10, criterion='absolute_error',min_samples_leaf = 10,random_state=123)

# Crear un conjunto de entrenamiento y uno de prueba

X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_c, y_c, test_size=0.2, random_state=123)

# Construir el modelo de árbol de decisiones

RF_R.fit(X_train_c, y_train_c)
```

```
▾                    RandomForestRegressor
RandomForestRegressor(criterion='absolute_error', max_depth=20,
                      min_samples_leaf=10, min_samples_split=10,
                      n_estimators=60, random_state=123)
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en regresión:

```
#se valora el posible sobreajuste

pred_rf_c_train = RF_R.predict(X_train_c)

pred_rf_c_test = RF_R.predict(X_test_c)

print(f'MAE del modelo en train:{mean_absolute_error(y_train_c,pred_rf_c_train)}')

print(f'MAE del modelo en test:{mean_absolute_error(y_test_c,pred_rf_c)}')

# se procede a validación cruzada sin un grill de parámetros:

print(RF_R.get_params())
```

```
MAE del modelo en train:4.046392597087379
MAE del modelo en test:4.916126618122975
```

```
scoring_metrics_c = {
    'MAE': make_scorer(mean_absolute_error),
    'MSE': make_scorer(mean_squared_error)}
# los parámetros necesitan presentar formato lista.
params = { 'bootstrap': [True],
    'criterion': ['absolute_error'],
    'max_depth': [20],
    'max_features': [1.0],
    'min_samples_leaf': [10],
    'min_samples_split': [10],
    'n_estimators': [60],
    'random_state': [123]}
```

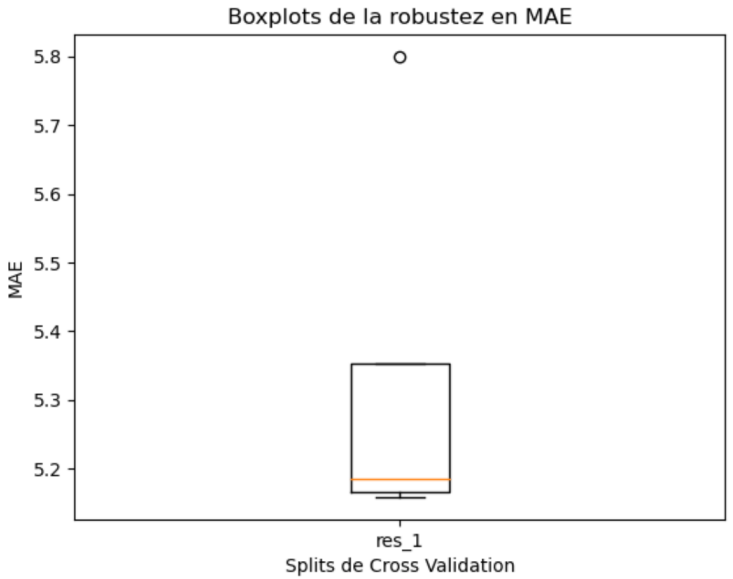
```
grid_search_rf_c = GridSearchCV(estimator=RF_R,
                                param_grid=params,
                                cv=4, scoring = scoring_metrics_c, refit='MAE')
grid_search_rf_c.fit(X_train_c, y_train_c)
```

2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en regresión:

```
res_1 = pd.DataFrame(grid_search_c.cv_results_)[['split0_test_MAE', 'split1_test_MAE','split2_test_MAE','split3_test_MAE']].iloc[0]

print(res_1)

# Crear un boxplot para los cuatro valores de accuracy

plt.boxplot([res_1.values], labels = ['res_1'])

plt.title('Boxplots de la robustez en MAE')

plt.xlabel('Splits de Cross Validation')

plt.ylabel('MAE')

plt.show()
```

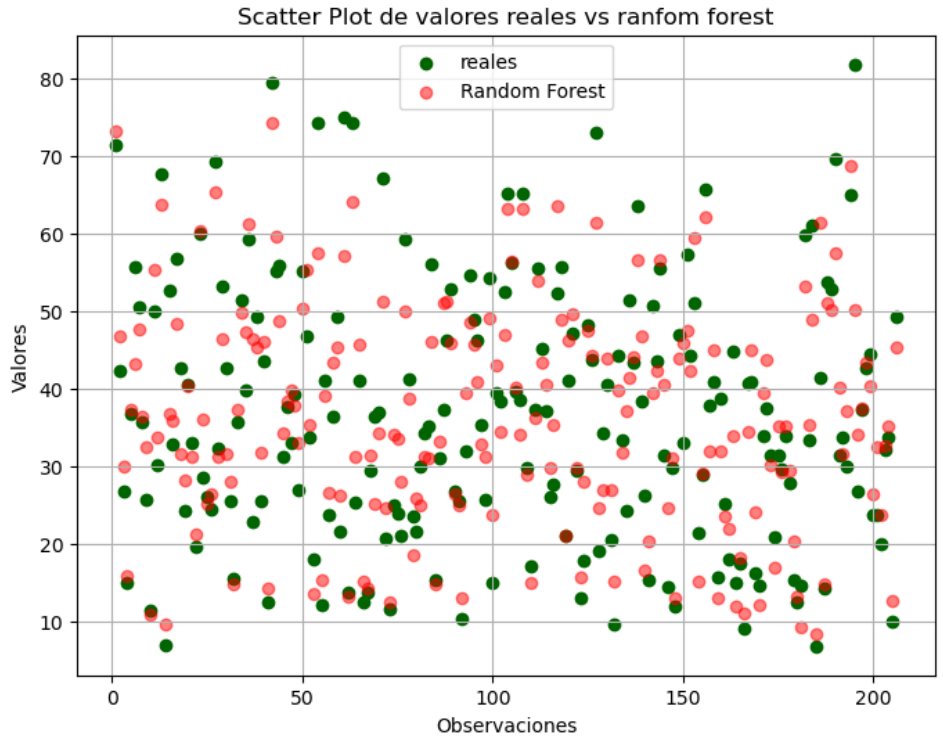
```
split0_test_MAE    5.202713
split1_test_MAE    5.168623
split2_test_MAE    5.158319
split3_test_MAE    5.799694
Name: 0, dtype: float64
```



2. Random Forest (Breiman, 2001).

❖EJERCICIO: ejemplo de pasos a tomar en regresión:

```
indices = np.arange(1, len(y_test_c) + 1)


plt.figure(figsize=(8, 6))

plt.scatter(indices, y_test_c, color='darkgreen', label='reales')

plt.scatter(indices, pred_rf_c_test, color='red', alpha=0.5, label='Random Forest

plt.title('Scatter Plot de valores reales vs ranfom forest')

plt.xlabel('Observaciones')

plt.ylabel('Valores')

plt.legend()  # Agregar leyenda

plt.grid(True)

plt.show()
```




```
# Calcular diferentes medidas de bondad de ajuste
mae = mean_absolute_error(y_test_c, pred_rf_c_test)
mse = mean_squared_error(y_test_c, pred_rf_c_test)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_c, pred_rf_c_test)

# Imprimir las métricas
print(f'MAE (Error Absoluto Medio): {mae:.2f}')
print(f'MSE (Error Cuadrático Medio): {mse:.2f}')
print(f'RMSE (Raíz del Error Cuadrático Medio): {rmse:.2f}')
print(f'R2: {r2}')
```

```
MAE (Error Absoluto Medio): 4.92
MSE (Error Cuadrático Medio): 43.83
RMSE (Raíz del Error Cuadrático Medio): 6.62
R2: 0.8402963970008323
```

3. Gradient Boosting (Friedman, 2001).

- El Gradient Boosting funciona añadiendo secuencialmente predictores a un conjunto y cada uno de ellos corrige los errores de su predecesor. Este modelo se entrena con los errores residuales del predictor anterior.

- Se basa en ir actualizando las predicciones en la **dirección de decrecimiento dada por el negativo del gradiente, de la función de error** $L(y_i, f(x_i))$, donde y_i es la etiqueta real y $f(x_i)$ el valor predicho por el modelo.

- La actualización de las predicciones se realiza en la dirección de decrecimiento de la función de error.

3. Gradient Boosting (Friedman, 2001).

IDEA: buscar la pendiente de crecimiento.

NOTA: tanto el gradient boosting como el xgboost son modelos sin NINGUNA EXPLICABILIDAD, totalmente caja negra.

Consejo: usarlos sólo cuando la ganancia en eficacia sea muy significativa.
Cuando se pierde la idea intuitiva del origen de una ineficiencia, es muy difícil resolver los problemas.

En cada iteración se calcula la pendiente

En cada árbol calculo los parámetros que lo mejorarían, y actualizo

Esquema *Gradient Boosting*

1) Inicialmente, $f_0(x_0) = argmin \sum_{i=1}^{N} L(y_i, \gamma)$
2) $For\ m = 1\ to\ M$
   a. $For\ i = 1,2 \dots N$:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   b. Crear árboles de regresión, recalcular los residuos r_{im}, a partir de las regiones temporales $R_{jm}, j = 1,2, \dots J_m$.
   c. $For\ j = 1, 2, \dots, J_m$,

$$\gamma_{jm} = argmin_{\gamma} \sum_{X_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   d. Actualizar $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3) Finalmente, $\hat{f}(x) = f_M(x)$

3. Gradient Boosting (Friedman, 2001).

1. Comienza con un modelo inicial simple, como un solo árbol de decisión.
2. Calcula las predicciones iniciales y los errores residuales (diferencias entre las predicciones y las etiquetas reales).
3. Ajusta un nuevo modelo para predecir estos errores residuales. Este nuevo modelo se agrega al modelo existente, corrigiendo los errores.
4. Repite los pasos 2 y 3, iterativamente, para mejorar el modelo en cada paso. Cada nuevo modelo se enfoca en corregir los errores restantes del conjunto anterior.

3. Gradient Boosting (Friedman, 2001).

A continuación, se presentan algunas de las funciones de error para ajustar el modelo:

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

En la tabla anterior se ve que la derivada (el gradiente) de la función de error clásica en regresión es el residuo o diferencia entre el verdadero valor de la *y* y su predicción: r= y-f(x).

3. Gradient Boosting (Friedman, 2001).

Entonces básicamente, en problemas de regresión, el algoritmo Gradient Boosting consiste en modificar las predicciones en la dirección de decrecimiento del gradiente (en este caso el residuo).

Si el residuo sale negativo en una observación (estamos prediciendo valores más altos que la realidad), se actualiza la predicción en la dirección de decrecimiento, es decir se reduce el valor predicho.

- **NOTA:** gradient boosting se suele usar con árboles, pero no es más que un tipo de ensamblado que se podría aplicar en cualquier modelo.

- **OJO:** a diferencia de bagging/random forest, gradient boosting no calcula un montón de árboles y después los agrega, sino que, **para cada árbol construido, analiza cómo se ha equivocado y trata de aliviar el error** (por efecto o por defecto) de cara a la construcción del siguiente: cada árbol es una mejora del anterior. El último árbol no es la agregación de todos, sino el mejor de los conseguidos.

3. Gradient Boosting (Friedman, 2001).

Algoritmo Gradient Boosting para regresión (función de error SCE):

1) Dar como valor predictivo de la variable *y*, para cada observación, la **media** de los valores de la variable *y*. Este será el punto de partida, y en cada iteración del algoritmo la predicción de *y* para cada observación será actualizada de manera individual.

$$\hat{y}_i^{(0)} = \bar{y}$$

2) Repetir los pasos siguientes para cada iteración *m*:

i) Calcular el residuo actual $r_i^{(m)} = y_i - \hat{y}_i^{(m)}$

ii) Construir un árbol de regresión para predecir los residuos, tomando $r_i^{(m)}$ como variable dependiente, y el conjunto de las variables X input como independientes. Esto nos dará como resultado un residuo "predicho" $\hat{r}_i^{(m)}$, que no es exactamente igual que el real, pero nos sirve para actualizar las observaciones test, para las que no hay residuo real al no haber *y*.

De este modo se obtienen predicciones de los residuos para datos train y para datos test.

3. Gradient Boosting (Friedman, 2001).

Algoritmo Gradient Boosting para regresión (función de error SCE):

iii) Actualizar la predicción de *y* para cada observación (incluidas las observaciones de datos test), en la dirección de decrecimiento.

- ¿Cuánto se modifica la predicción de cada observación?
- Lo normal es que sea en una cantidad proporcional al valor del residuo, así si se comete mucho error se modifica mucho la predicción y si es pequeño, menos.
- No conviene hacer grandes modificaciones, con el fin de que el algoritmo las haga de forma gradual.

Aquí interviene la constante de regularización ν que toma valores pequeños (entre 0.0001 y 0.2 por ejemplo), para que en cada paso se modifique poco la predicción, en la buena dirección, pero poco.

$$\hat{y}_i^{(m+1)} = \hat{y}_i^{(m)} + \nu \cdot \hat{r}_i^{(m)}$$

3. Gradient Boosting (Friedman, 2001).

Algoritmo Gradient Boosting para regresión (función de error SCE):

3) El proceso se detiene cuando se llega al número de iteraciones final deseado.

- Es conveniente señalar que los datos train convergen al verdadero valor de la y, así que **no se debe tomar en ningún caso como referencia la performance del gradient boosting sobre datos train**, sino solamente sobre datos test.

3. Gradient Boosting (Friedman, 2001).

Parámetros a modificar para Gradient Boosting:

- La constante de regularización ν (shrink) . Normalmente entre (0.0001 y 0.2). Cuanto más alta, más rápido converge, pero demasiado alta es poco preciso. Si se pone muy baja (la recomendación teórica) hay que poner muchas iteraciones para que converja. En la práctica se comienza con valores altos para observar resultados básicos y cuando se controla bien el proceso el modelo final se realiza con valores bajos de v y muchas iteraciones.
- El número final de iteraciones-árboles **M.** A menor ν, serán necesarias más iteraciones M. Es un parámetro a monitorizar (con validación cruzada y gráficos preferentemente), pues teoría y práctica coinciden en que a partir de un punto se puede producir sobreajuste. La decisión sobre el valor M se denomina **early stopping**.

La idea básica es monitorear el rendimiento del modelo en un conjunto de datos de validación durante el proceso de entrenamiento y detener el entrenamiento una vez que el rendimiento deja de mejorar o comienza a empeorar.

- Características de los árboles. Son bastante influyentes:
  - El número de hojas final o, en su defecto, la profundidad del árbol
  - El maxbranch (número de divisiones máxima en cada nodo. Por defecto se dejará en 2, árboles binarios)
  - El número de observaciones mínimo en una rama-nodo. Se puede ampliar para evitar sobreajuste (reducir varianza) o reducir para ajustar mejor (reducir sesgo).

3. Gradient Boosting (Friedman, 2001).

Parámetros a modificar para Gradient Boosting:

❖Adicionalmente, en búsquedas paramétricas sofisticadas se podría considerar:

- **Tasa de aprendizaje** (*learning rate*): controla la contribución de cada árbol al modelo general. Un valor más bajo de tasa de aprendizaje hará que cada árbol tenga una contribución menor, lo que puede hacer que el modelo sea más robusto, pero requerirá más árboles para alcanzar un rendimiento óptimo. Similar al parámetro de regularización.
- **Submuestreo** de características (subsample): este parámetro controla la proporción de características que se seleccionan aleatoriamente en cada iteración para construir un árbol. Un valor menor que 1.0 hará que cada árbol se construya con un subconjunto de características, lo que puede ayudar a reducir el sobreajuste y mejorar la generalización.

3. Gradient Boosting (Friedman, 2001).

Parámetros a modificar para *GradientBoostingClassifier*:

loss: ***{'log_loss', 'exponential'}, default='log_loss'*** Función de pérdida para optimizar.

learning_rate: ***float, default=0.1*** Parámetro shrink de ratio de aprendizaje.

n_estimators: ***int, default=100***

Subsample: ***float, default=1.0*** Proporción de la muestra para entrenar cada uno de los modelos base.

Criterion: ***{'friedman_mse', 'squared_error'}, default='friedman_mse'***

random_state

warm_start*bool, default=False*

validation_fraction: Proporción de training para validación en early stopping.

n_iter_no_change: Número de iteracones en la que parar si no se encuentran mejoras en la función de validación.

tol: Parámetro para la tolerancia al earlystopping. Entre 0 e inf. Cuando la función de pérdida no mejora en n_iter_no_change, se para**.**

ccp_Alpha: Parámetro de complejidad

Más los referentes a árboles ya conocidos.

3. Gradient Boosting (Friedman, 2001).

Parámetros a modificar para *GradientBoostingRegressor*:

loss: ***{'squared_error', 'absolute_error', 'huber', 'quantile'}, default='squared_error'*** Función de pérdida para optimizar.

learning_rate: ***float, default=0.1*** Parámetro shrink de ratio de aprendizaje.

n_estimators: ***int, default=100***

Subsample: ***float, default=1.0*** Proporción de la muestra para entrenar cada uno de los modelos base.

Criterion: ***{'friedman_mse', 'squared_error'}, default='friedman_mse'***

random_state

warm_start*bool, default=False*

alpha*float, default=0.9*

validation_fraction: Proporción de training para validación en early stopping.

n_iter_no_change: Número de iteracones en la que parar si no se encuentran mejoras en la función de validación.

tol: Parámetro para la tolerancia al earlystopping. Entre 0 e inf. Cuando la función de pérdida no mejora en n_iter_no_change, se para**.**

ccp_Alpha: Parámetro de complejidad

Más los referentes a árboles ya conocidos.

3. Gradient Boosting (Friedman, 2001).

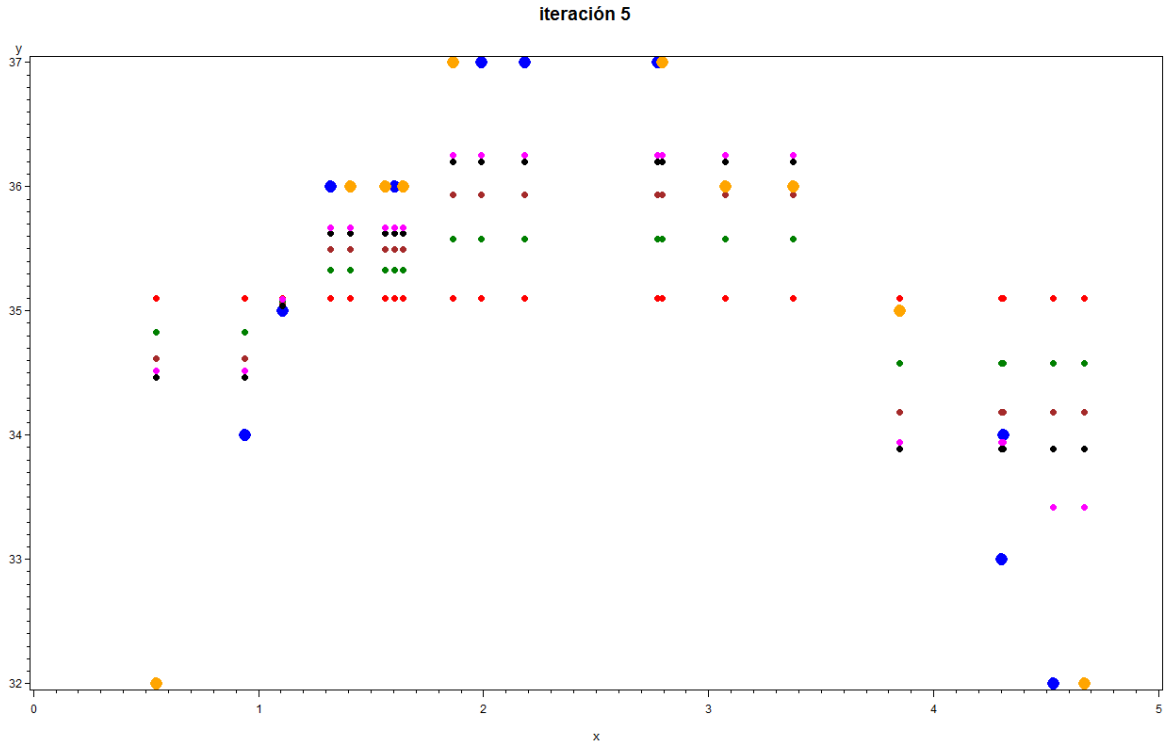
Ejemplo de construcción manual:

- 1) En la primera iteración (puntos pequeños color rojo en el gráfico) se fijan los valores iniciales de las predicciones de la variable y como su media (35.1) , para todas las observaciones.
- 2) Se calcula el residuo real (resi1) que para la primera observación train toma valor -1.1 y para las observaciones test no existe al no existir la y.
- 3) Se construye un árbol de regresión, con resi1 como variable objetivo, x como variable input. Esto da una predicción para resi1 (resi1_est) que no es exactamente igual que resi1: en la observación train nº9 resi1 toma valor -1.1 y su predicción -2.1; además las observaciones test tienen valor predicho resi1_est, al disponer de la variable predictora x.
- 4) Se actualiza la predicción de y (y1). En la primera observación train, de predecir con 35.1, se ha reducido la predicción en la buena dirección a 34.825; en las observaciones test se ha pasado a 34.825 y 34.575. En el gráfico aparece en color verde la predicción al final de esta primera iteración.
- 5) El proceso continúa: se calcularían los residuos, se predecirían, se actualizaría la y en cada iteración. Se observa cómo las observaciones reales train (puntos grandes azules) tienden a clavar perfectamente su predicción pero las observaciones reales test (puntos grandes naranjas), que son las que importan, también se predicen bastante bien. La quinta iteración está representada por los puntos pequeños rosa.

3. Gradient Boosting (Friedman, 2001).

Ejemplo de construcción manual:

y	ytest	x	media	resi1	resi1_est	y1
34	.	0.93843	35.1	-1.1	-1.1	34.825
35	.	1.10557	35.1	-0.1	-0.1	35.075
36	.	1.31851	35.1	0.9	0.9	35.325
36	.	1.60456	35.1	0.9	0.9	35.325
37	.	1.99041	35.1	1.9	1.9	35.575
37	.	2.18037	35.1	1.9	1.9	35.575
37	.	2.77429	35.1	1.9	1.9	35.575
33	.	4.30212	35.1	-2.1	-2.1	34.575
34	.	4.30672	35.1	-1.1	-2.1	34.575
32	.	4.53017	35.1	-3.1	-2.1	34.575
.	32	0.54394	35.1	.	-1.1	34.825
.	32	4.66699	35.1	.	-2.1	34.575



3. Gradient Boosting (Friedman, 2001).

Gradient Boosting para Clasificación:

La función de error en este caso es la deviance:

$$L\big(y_i, f(x_i)\big) = \log(1 + e^{-2y_i f(x_i)})$$

donde la variable dependiente es binaria: $y_i = [1,0]$

La función $f(x_i)$ es la función logit y se define como $f(x_i) = \frac{1}{2}\log(\frac{\hat{p}_i^{(m}}{1-\hat{p}_i^{(m}})$ con $p_i = P(y_i = 1)$

3. Gradient Boosting (Friedman, 2001).

Gradient Boosting para Clasificación:

- **1)** Se toma como valor inicial para la probabilidad predicha de 1 en todas las observaciones el porcentaje de 1 en la muestra:
- $\hat{p}_i^{(0)} = \%\ de\ observaciones\ con\ y = 1;\ \hat{f}_i^{(0)} = \frac{1}{2}\log(\frac{\hat{p}_i^{(0)}}{1-\hat{p}_i^{(0)}})$
- **2)** Repetir los pasos siguientes para cada iteración *m*:

Recordar: constante de regularización ν

  - **i)** Calcular el residuo actual $r_i^{(m)} = y_i - \hat{p}_i^{(m)}$
  - **ii)** Construir un árbol de regresión para predecir los residuos, tomando $r_i^{(m)}$ como variable dependiente u objetivo, y el conjunto de las variables X input como independientes.
  - **iii)** Actualizar la predicción de la función logit f para cada observación: $\hat{f}_i^{(m+1)} = \hat{f}_i^{(m)} + \nu \cdot \hat{r}_i^{(m)} = \frac{1}{2}\log(\frac{\hat{p}_i^{(m)}}{1-\hat{p}_i^{(m)}}) + \nu \cdot \hat{r}_i^{(m)}$
  - iv) Actualizar la predicción de las probabilidades mediante: $\hat{p}_i^{(m+1)} = \frac{1}{1+e^{-2\hat{f}_i^{(m+1)}}}$
- **3)** El proceso se detiene cuando se llega al número de iteraciones final deseado.

3. Gradient Boosting (Friedman, 2001).

Stochastic Gradient Boosting:

Es una pequeña modificación para luchar

contra el sobreajuste y alta varianza, en la línea de bagging-random forest: Se selecciona en cada iteración (cada árbol creado), una muestra diferente de los datos de entrenamiento para construir el árbol.

3. Gradient Boosting (Friedman, 2001).

Ventajas de Gradient Boosting:

- Invariante frente a transformaciones monótonas: no es necesario realizar transformaciones logarítmicas, etc.
- Buen tratamiento de missing, variables categóricas, etc. Universalidad.
- Muy fácil de implementar, relativamente pocos parámetros a monitorizar (número de hojas o profundidad del árbol, tamaño final de hojas, parámetro de regularización…).
- Gran eficacia predictiva, algoritmo muy competitivo. En Kaggle, aproximadamente el 80% de los concursantes lo usa, exclusivamente o combinado con otras técnicas. Supera a menudo al algoritmo Random Forest.
- Robusto respecto a variables irrelevantes. Robusto respecto a colinealidad. Detecta interacciones ocultas.
- Permite representar la importancia de variables

3. Gradient Boosting (Friedman, 2001).

Desventajas de Gradient Boosting:

- Como todos los métodos basados en árboles, dependiendo de los datos puede ser superado por otras técnicas más sencillas.

- En datos relativamente **sencillos** (pocas variables, no missing, no interacciones, linealidad (regresión) o separabilidad lineal (clasificación)), el gradient boosting (o random forest) no tiene nada nuevo que aportar y pueden ser preferibles modelos sencillos (regresión, regresión logística, discriminante) o modelos ad-hoc que adapten aspectos concretos como la no linealidad (redes por ejemplo).

- Es una caja negra.

3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

Con la metodología aplicada anteriormente, aplicar a la BB.DD. El algoritmo de Gradient Boosting para un problema de clasificación y para un problema de regresión.

- Probar distintos parámetros en GridSearchCV y observar detenidamente la performance para cada una de las soluciones. Puede haber casos que interese apostar por una mayor sensibilidad, o especificidad…

- El mejor modelo es aquel que, no sólo presenta un buen ajuste, sino que se comporta de acuerdo a los criterios de la tarea concreta.

3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

```
gb_classifier = GradientBoostingClassifier(n_estimators = 450, subsample = 1, random_state = 123,n_iter_no_change = 10)

gb_classifier.fit(X_train, y_train)

y_pred_base = gb_classifier.predict(X_test)

# Evaluar el rendimiento del modelo

accuracy_a = accuracy_score(y_test, y_pred_base)

print(f'Precisión de gradient boosting: {accuracy_a}')
```

Precisión de gradient boosting: 0.72

3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

```
# se procede a observar el posible sobreajuste comparando predicciones en train y test.
# predicciones significativamente mayores en train que en test puede indicar sobreajuste.
# Predicciones en conjunto de entrenamiento y prueba
y_train_pred = gb_classifier.predict(X_train)
y_test_pred = gb_classifier.predict(X_test)
print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred)}')
print('Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.')
```

```
Se tiene un accuracy para train de: 0.8617886178861789
Se tiene un accuracy para test de: 0.7204301075268817
Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.
```

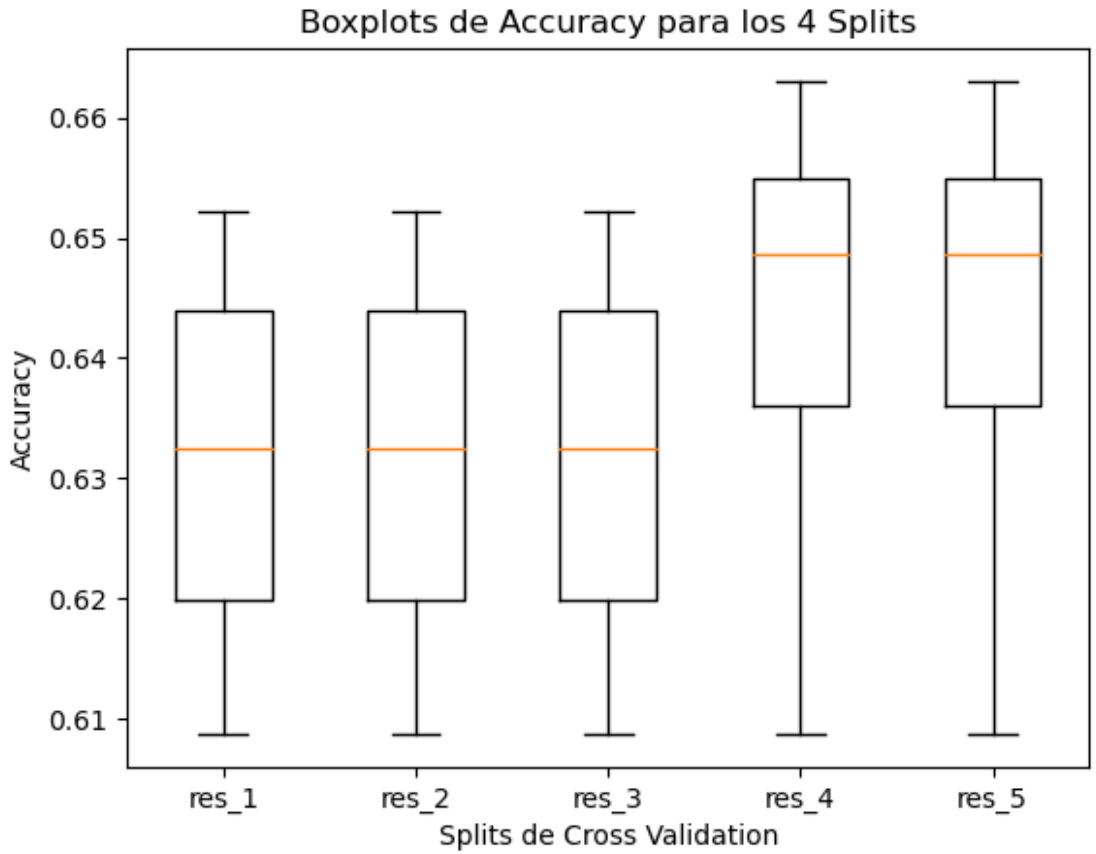
3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

```
# Se puede seleccionar un grill extenso y meditado sobre los distintos parámetros con los que jugar.
#params = {
#    'loss': ["log_loss", "exponential"],
#    'n_estimators': [200,400,600],
#    'n_iter_no_change': [None,5,10,20],
#    'criterion': ["friedman_mse", "squared_error"],
#    'max_depth': [2, 3, 5, 10, 20],
#    'min_samples_leaf' : [3,10,30],
#    'min_samples_split': [5, 10, 50, 100],
#}
params = {
    'n_estimators': [400,600],
    'n_iter_no_change': [None,5,10],
    'max_depth': [5, 10],
    'min_samples_leaf' : [30],
    'min_samples_split': [5, 10, 50],
}

scoring_metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
#recordar que arbol2 es el árbol cuyas VI son todas las variables.
# cv = crossvalidation
grid_search_GB = GridSearchCV(estimator=gb_classifier,
                              param_grid=params,
                              cv=4, scoring = scoring_metrics, refit='accuracy')
grid_search_GB.fit(X_train, y_train)
```

3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**



3. Gradient Boosting (Friedman, 2001).

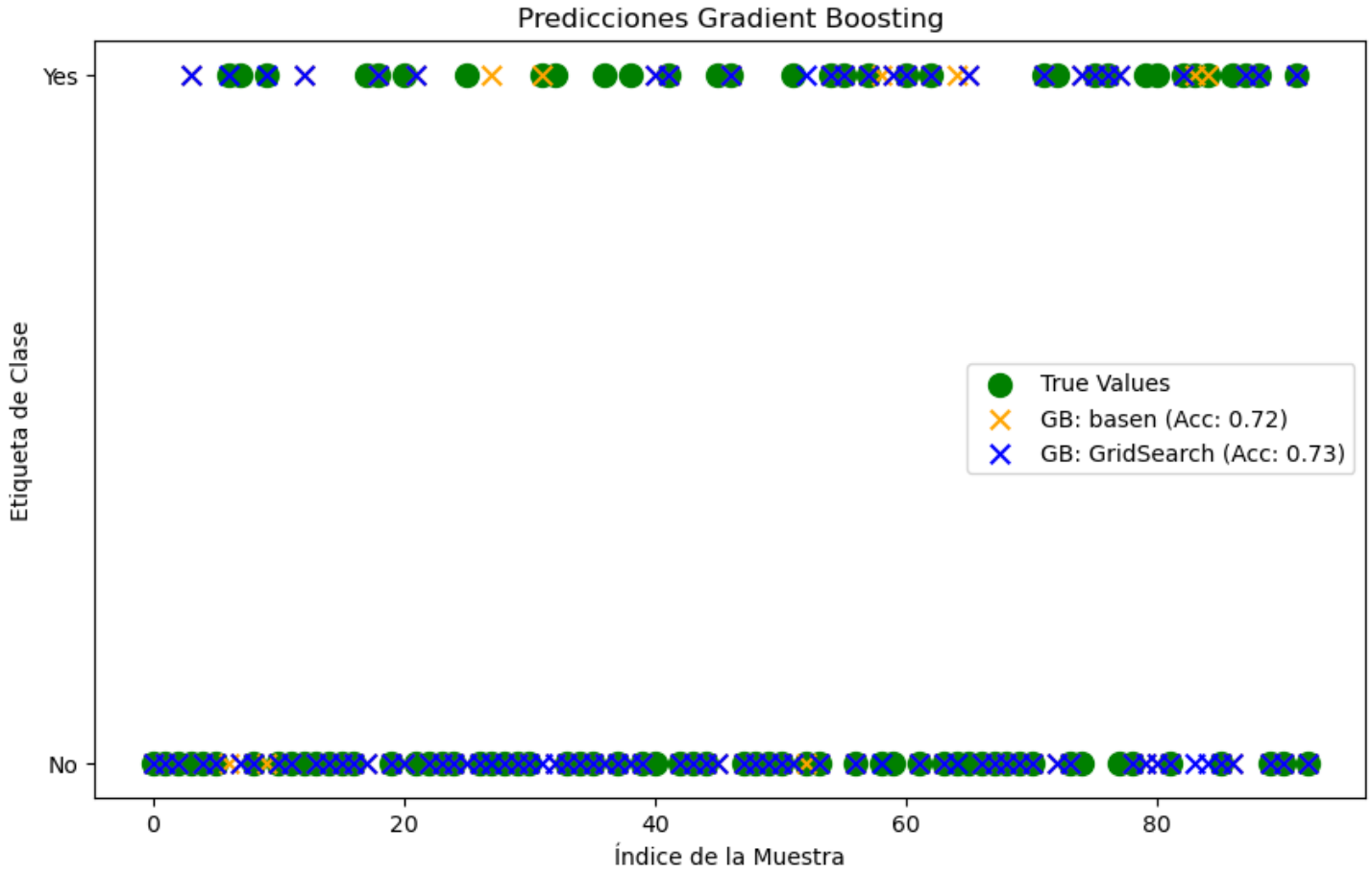
❖**EJERCICIO:**

```
# modelo_GB = RandomForestClassifier(**sorted_results['params'].iloc[N],random_state=123) para el modelo N deseado
modelo_GB = grid_search_GB.best_estimator_

y_train_pred_gb = modelo_GB.predict(X_train)
y_test_pred_gb = modelo_GB.predict(X_test)
print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred_gb)}')
print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred_gb)}')
print('Nótese la diferencia en accuracy para ambos conjuntos de datos se ha reducido en gran medida.')
```

```
Se tiene un accuracy para train de: 0.7940379403794038
Se tiene un accuracy para test de: 0.7311827956989247
Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.
```

3. Gradient Boosting (Friedman, 2001).

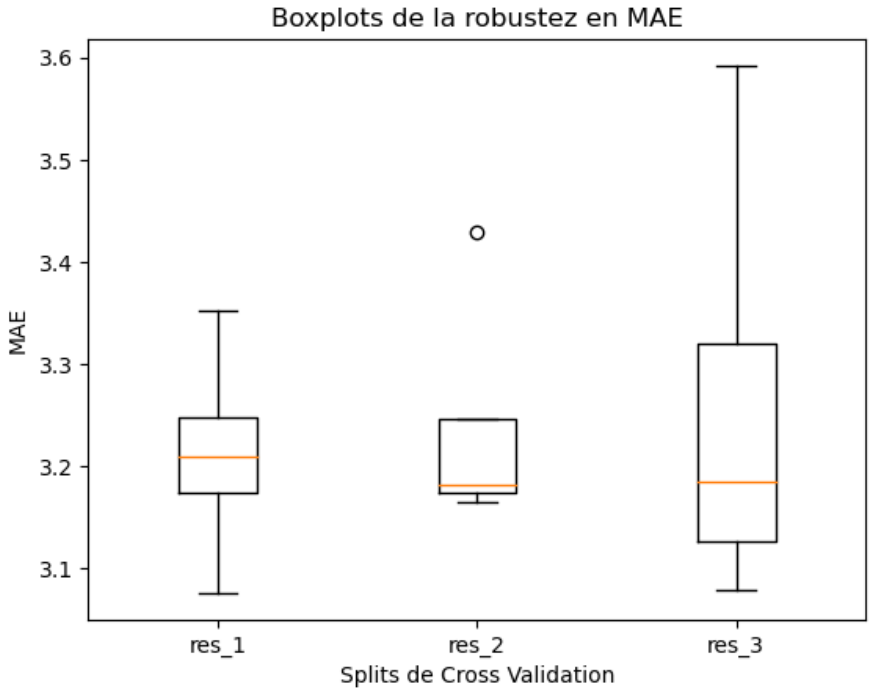
❖**EJERCICIO:**

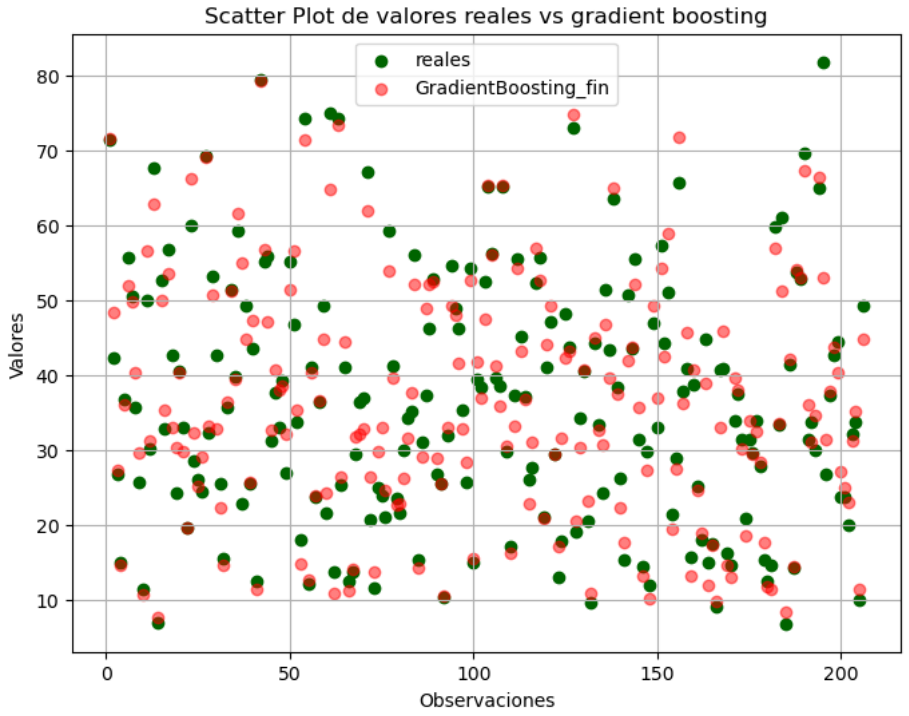


3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

SE REPITE EL PROCESO PARA EL CASO CONTÍNUO CON “compress.csv”:





3. Gradient Boosting (Friedman, 2001).

❖**EJERCICIO:**

SE REPITE EL PROCESO PARA EL CASO CONTÍNUO CON “compress.csv”:

```
# Calcular diferentes medidas de bondad de ajuste
mae = mean_absolute_error(y_test_c, pred_gb_c_test_fin)
mse = mean_squared_error(y_test_c, pred_gb_c_test_fin)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_c, pred_gb_c_test_fin)

# Imprimir las métricas
print(f'MAE (Error Absoluto Medio): {mae:.2f}')
print(f'MSE (Error Cuadrático Medio): {mse:.2f}')
print(f'RMSE (Raíz del Error Cuadrático Medio): {rmse:.2f}')
print(f'R2: {r2}')
```

```
MAE (Error Absoluto Medio): 3.01
MSE (Error Cuadrático Medio): 22.42
RMSE (Raíz del Error Cuadrático Medio): 4.73
R2: 0.9183196289923017
```

4. Extreme Gradient Boosting (XGBOOST)

- La principal aportación del algoritmo con respecto a Gradient Boosting es la regularización.
- Proporciona mayor eficiencia y rendimiento.

CARACTERÍSTICAS:

- **Optimización de Gradient Boosting**: XGBoost utiliza una implementación optimizada del algoritmo de Gradient Boosting, que lo hace más rápido y eficiente.
- **Características esparsas**: XGBoost puede manejar conjuntos de datos con características esparsas (conjuntos de datos en los que la mayoría de los valores de las características son cero o muy cercanos a cero, lo que implica que la mayoría de las características tienen poco impacto en el resultado final del modelo).
- **Regularización**: incluye técnicas de regularización para controlar el sobreajuste, como la penalización L1 y L2 en los pesos de los árboles.

4. Extreme Gradient Boosting (XGBOOST)

CARACTERÍSTICAS:

- **Selección automática de variables**: permite a identificar automáticamente las variables más importantes para el modelo.
- **Valores missing**: puede manejar valores faltantes sin necesidad de realizar imputaciones.
- **Paralelización**: aprovecha el procesamiento paralelo en múltiples núcleos de CPU, lo que acelera el entrenamiento y la predicción, especialmente en conjuntos de datos grandes.
- **Optimización hiperparamétrica**: proporciona herramientas para la optimización hiperparamétrica, como la búsqueda en cuadrícula (optimización de la combinación de modelos con hiperparámetros) y la búsqueda aleatoria, para ayudar a encontrar la combinación óptima de parámetros para el modelo.

4. Extreme Gradient Boosting (XGBOOST)

FILOSOFÍA:

- Se basa en la técnica de aumento de gradiente .
- Combina múltiples **modelos débiles** (normalmente árboles de decisión, pero no es la única opción) para crear un modelo más fuerte y preciso.
- **IDEA PRINCIPAL:** cada **nuevo modelo** se construye para **corregir** los **errores** cometidos por los modelos **anteriores**.
  - Comienza con un modelo inicial simple y luego construye modelos adicionales para enfocarse en los casos que fueron mal predichos anteriormente.
  - Cada modelo se ajusta a los datos de entrenamiento calculando los gradientes de una función de pérdida específica. Luego, el modelo intenta minimizar esta función de pérdida encontrando la mejor combinación de características y parámetros para hacer predicciones más precisas.
  - Utiliza **técnicas de regularización** para evitar el sobreajuste y mejorar la generalización del modelo. Esto incluye términos de penalización en la función de pérdida para controlar la complejidad del modelo y limitar la profundidad de los árboles de decisión.

4. Extreme Gradient Boosting (XGBOOST)

REGULARIZACIÓN:

- Es una técnica utilizada en el aprendizaje automático para evitar el sobreajuste (overfitting) y mejorar la capacidad de generalización de un modelo.
- Consiste en agregar términos adicionales a la función de pérdida o error del modelo durante el proceso de entrenamiento, con el objetivo de penalizar los valores excesivamente grandes de los parámetros del modelo.
- La regularización ayuda a controlar la complejidad del modelo, evitando que se ajuste demasiado a los datos de entrenamiento.

Esto es especialmente útil cuando se trabaja con conjuntos de datos pequeños o ruidosos, donde el riesgo de sobreajuste es mayor. Es una técnica orientada a la reducción de varianza de los errores (sobreajuste).

4. Extreme Gradient Boosting (XGBOOST)

Otros métodos para controlar el sobreajuste:

- seleccionar modelos más sencillos.
- early stopping (que no haga demasiadas iteraciones).
- utilizar validación cruzada para controlar la varianza del error.
- Ensamblados.

❖La diferencia con la **regularización** es que ésta **interviene en la optimización** interna del algoritmo (en el proceso de estimación de parámetros).

4. Extreme Gradient Boosting (XGBOOST)

- **Regularización Lasso (L1):** funciona al agregar una penalización proporcional al valor absoluto de los coeficientes del modelo. Esto significa que algunos de los coeficientes pueden hacerse exactamente cero. L1 tiende a eliminar variables irrelevantes del modelo, seleccionando solo las más importantes. Es útil cuando se desea realizar una **selección automática** de **variables** ante la sospecha de que sólo algunas son importantes.

- **Regularización Ridge (L2):** funciona al agregar una penalización proporcional al cuadrado de los coeficientes del modelo. A diferencia de la regularización Lasso, la regularización Ridge no hace que los coeficientes sean exactamente cero. En cambio, reduce la magnitud de todos los coeficientes, lo que ayuda a evitar que los coeficientes tengan valores extremadamente grandes y controla la complejidad del modelo. Es útil cuando se desea **reducir** la **influencia** de variables menos importantes en el modelo, **sin eliminarlas** por completo.

4. Extreme Gradient Boosting (XGBOOST)

❖El caso más conocido y antiguo de regularización es la **regresión Ridge**, que utiliza la regularización L2 para evitar el sobreajuste y mejorar la estabilidad del modelo. Se agrega un término de penalización a la función de pérdida del modelo, que es proporcional a la suma de los cuadrados de los coeficientes del modelo. Este término de penalización se controla mediante un hiperparámetro llamado "parámetro de regularización" o "factor de regularización

❖**IDEA**: tener parámetros con valores bajos para evitar el sobreajuste.

4. Extreme Gradient Boosting (XGBOOST)

❖Cuando hay colinealidad (variables independientes muy correladas entre ellas) la estimación de los parámetros puede ser muy errática. Por ello se introduce un término de penalización:

En regresión normal:

$$min_\beta \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\}$$

En regresión Ridge se fija primero un parámetro lambda y se penalizan los valores altos de la suma de parámetros al cuadrado:

$$\hat{\beta}^{ridge} = argmin_\beta \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

De este modo, el carácter errático de los parámetros se ve corregido, obteniendo valores bajos y **más estables** para los betas

4. Extreme Gradient Boosting (XGBOOST)

- En este algoritmo se modifica el gradient boosting a la hora de construir cada árbol con una función de penalización basada en el número de hojas y el score- predicción en cada hoja (que sería análogo al valor del parámetro en regresión):
- Árboles más complejos = más hojas, más suma de cuadrados.
- El algoritmo Xgboost prefija **dos parámetros principales de regularización**, **lambda** y **alpha**, que penalizan por los pesos w, score-predicción en cada hoja. Un tercero **gamma** penaliza por el número de hojas Q.

$$\Omega(f_t) = \frac{1}{2}\lambda\left(\sum_{j=1}^{Q} w_j^2\right) + \alpha\left(\sum_{i=1}^{Q} |w_i|\right) + \gamma Q$$

4. Extreme Gradient Boosting (XGBOOST)

Ventajas:

- **Regularización**. Una gran novedad, aunque hay que monitorizar los parámetros lambda, alpha, lambda_bias. Sirven sobre todo para corregir la varianza del modelo. **A mayores valores, más conservador: más sesgo, menos varianza.**

- El paquete está elaborado más universalmente y permite utilizar diferentes funciones objetivo. Es **muy rápido** y esa es otra razón por su uso en grandes bases de datos.

- El algoritmo implementado **sigue dividiendo un nodo aunque parezca malo**, y después evalúa el árbol final**. El gradient boosting normal se para en un nodo si es malo.** Esto puede significar una gran diferencia.

- El programa Xgboost incorpora también, aparte de la regularización, el **control del sobreajuste** utilizando las ideas de remuestreo del randomforest: tiene un parámetro de % de sorteo de observaciones y otro de sorteo de variables.

4. Extreme Gradient Boosting (XGBOOST)

Desventajas:

- Es otra manera de construir los árboles, no hay mucha teoría al respecto. ¡Ha nacido de la práctica! Tal vez llegue el momento en que se le detecten problemas.
- **Búsquedas exhaustivas de hiperparámetros:** hay que monitorizar los parámetros de regularización, asumiendo el riesgo de que sean muy dependientes de los datos utilizados. Los resultados de utilizar regularización a menudo no tienen efecto o son demasiado erráticos.
- **Sensible** al ruido y datos atípicos: más propenso a sobreajustar en presencia de datos ruidosos o valores atípicos.

4. Extreme Gradient Boosting (XGBOOST)

Desventajas:

- **Mayor consumo de memoria**, lo que ser un problema cuando se trabaja con conjuntos de datos grandes o en entornos con recursos limitados.
- **Dificultad en interpretación**
- Sensibilidad a la **selección de características**: puede ser sensible a la selección de características, lo que significa que el rendimiento del modelo puede depender en gran medida de las características elegidas. Si las características no están correctamente seleccionadas o se incluyen características irrelevantes, puede haber un impacto negativo en la precisión y la generalización del modelo.

4. Extreme Gradient Boosting (XGBOOST)

from xgboost import XGBClassifier, XGBRegressor

PARÁMETROS A MODIFICAR:

Antes de ejecutar XGBoost, debemos establecer tres tipos de parámetros: parámetros generales, parámetros del potenciador (booster) y parámetros de la tarea.

- Los parámetros generales se relacionan con el tipo de potenciador que estamos utilizando para realizar el aumento, comúnmente un modelo de árbol o lineal.
- Los parámetros del potenciador dependen del tipo de potenciador que hayas elegido.
- Los parámetros de la tarea de aprendizaje determinan el escenario de aprendizaje. Por ejemplo, las tareas de regresión pueden utilizar diferentes parámetros que las tareas de clasificación o de ranking.

4. Extreme Gradient Boosting (XGBOOST)

https://xgboost.readthedocs.io/en/stable/parameter.html#

Son múltiples parámetros a seleccionar. Aquí se muestran algunos de ellos. Ver el enlace.

Parámetros generales (algunos):

booster [default= gbtree]: gbtree, gblinear …

validate_parameters: Saber si un parámetro está siendo usado o no.

num_feature Número de variables a usar.

n_estimators.

4. Extreme Gradient Boosting (XGBOOST)

https://xgboost.readthedocs.io/en/stable/parameter.html#

Parámetros al elegir un árbol como '*booster*' (algunos):

- eta [default=0.3, alias: learning_rate]: parámetro v: controla el peso de las variables explicativas para hacer el modelo más o menos conservativo en cada iteración.

- gamma [default=0, alias: min_split_loss] Mínima reducción de pérdida para dividir un nodo.

- Los relacionados con árboles, e.g.: max_depth, max_leaves, min_child_weight…

- subsample: Ratio de instancias a ser muestreadas para cada iteración en el trianing set. Si 0.5, escogerá aleatoriamente la mitad de todas las instancias.

- sampling_method [default= uniform]: uniform, subsample, gradient_based (La probabilidad de selección para cada instancia de entrenamiento es proporcional al valor absoluto regularizado de los gradientes.)

4. Extreme Gradient Boosting (XGBOOST)

https://xgboost.readthedocs.io/en/stable/parameter.html#

Parámetros al elegir un árbol como '*booster*' (algunos):

- colsample_bytree, colsample_bylevel, colsample_bynode [default=1]. (Sub-muestreo de columnas)

- colsample_by* parameters: Controlan la fracción de columnas a submuestrear durante la construcción de árboles.
- colsample_bytree: Define la proporción de submuestreo de columnas para cada árbol.
- colsample_bylevel: Establece la proporción de submuestreo de columnas para cada nivel en un árbol.
- colsample_bynode: Determina la proporción de submuestreo de columnas para cada nodo (división) en un árbol.
- Los parámetros colsample_by* trabajan juntos, acumulativamente, afectando la selección de columnas en cada etapa del proceso.

4. Extreme Gradient Boosting (XGBOOST)

https://xgboost.readthedocs.io/en/stable/parameter.html#

Parámetros al elegir un árbol como '*booster'* (algunos):

tree_method string [default= auto]: Choices: auto, exact, approx, hist, gpu_hist

- auto: fasted method.

- exact: enumera todos los candidatos a división.

- approx: aproxima el 'exact' utilizando de cuantiles y un histograma de gradientes. Más rápido.

- hist: Algoritmo aproximado optimizado con histograma más rápido.

- gpu_hist: el anterior mejorado con uso de GPU.

- grow_policy [default= depthwise]. Depthwise (divide nodos cerca de la raíz), lossguide (divide los nodos con el cambio de pérdida más alto)

4. Extreme Gradient Boosting (XGBOOST)

https://xgboost.readthedocs.io/en/stable/parameter.html#

Otros parámetros:

- lambda [default=1, alias: reg_lambda]: Regularización L2.

- alpha [default=0, alias: reg_alpha]: Regularización L1.

- scale_pos_weight: Si los datos están desbalanceados.

- max_cat_to_onehot: Categorías a convertir a través del método one hot encoding.

- max_cat_threshold: Número máximo de categorías a escoger por cada división del árbol.

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Clasificación.

```
xgb_classifier = XGBClassifier(booster = 'gbtree', n_estimators = 200,
                               eta = 0.1, gamma = 1, random_state=123, max_depth = 15, tree_method = 'hist')

xgb_classifier.fit(X_train, y_train)

y_pred_base = xgb_classifier.predict(X_test)

# Evaluar el rendimiento del modelo

accuracy_a = accuracy_score(y_test, y_pred_base)

print(f'Precisión de gradient boosting: {accuracy_a}')
```

PRECISIÓN = 0.72

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Clasificación.

```
y_train_pred = xgb_classifier.predict(X_train)

y_test_pred = xgb_classifier.predict(X_test)

print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred)}')

print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred)}')

print('Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.')
```

```
Se tiene un accuracy para train de: 0.983739837398374

Se tiene un accuracy para test de: 0.7204301075268817

Nótese la diferencia en accuracy para ambos conjuntos de datos y el posible sobreajuste.
```

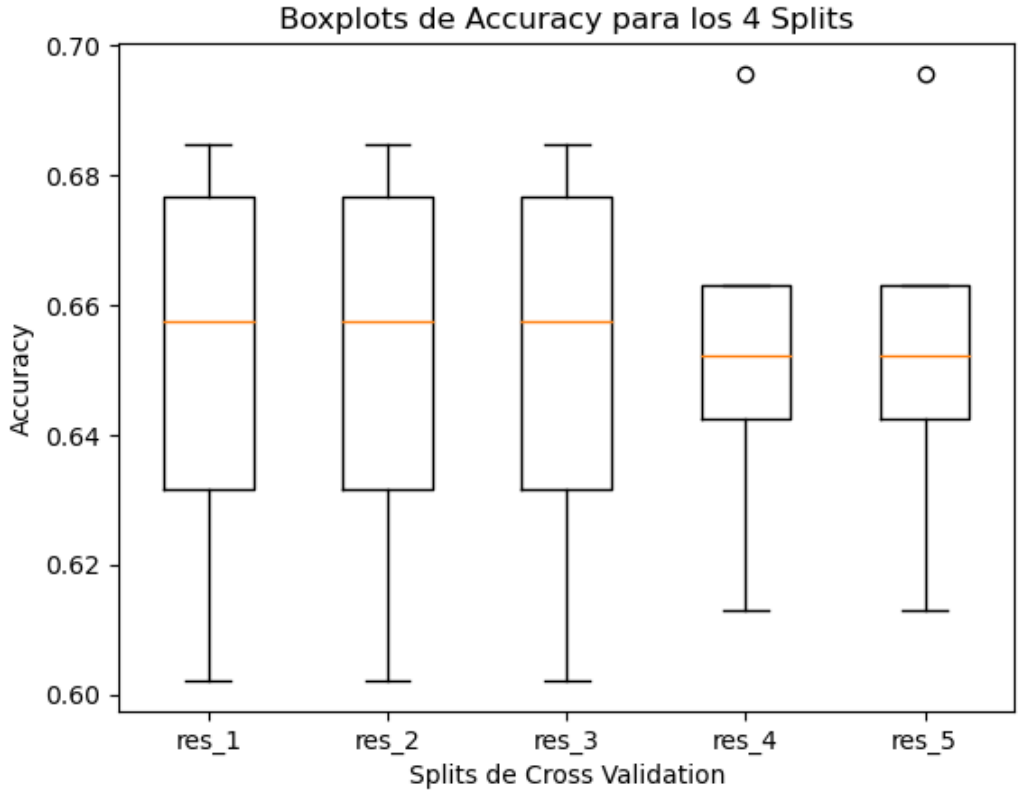
4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Clasificación.

```
params = {
    'n_estimators': [100,200,300],
    'eta' : [0.1,0.4,0.7],
    'gamma' : [0.1,0.5,1],
    'max_depth': [5, 10]
}

scoring_metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
#recordar que arbol2 es el árbol cuyas VI son todas las variables.
# cv = crossvalidation
grid_search_XGB = GridSearchCV(estimator=xgb_classifier,
                               param_grid=params,
                               cv=4, scoring = scoring_metrics, refit='accuracy')
grid_search_XGB.fit(X_train, y_train)
```

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Clasificación.




```
modelo_XGB = grid_search_GB.best_estimator_

y_train_pred_xgb = modelo_XGB.predict(X_train)
y_test_pred_xgb = modelo_XGB.predict(X_test)
print(f'Se tiene un accuracy para train de: {accuracy_score(y_train,y_train_pred_xgb)}')
print(f'Se tiene un accuracy para test de: {accuracy_score(y_test,y_test_pred_xgb)}')
print('Nótese la diferencia en accuracy para ambos conjuntos de datos sigue alta con estos parámetros.')
```

```
Se tiene un accuracy para train de: 0.948509485094851
Se tiene un accuracy para test de: 0.7311827956989247
Nótese la diferencia en accuracy para ambos conjuntos de datos sigue alta con estos parámetros.
```

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Clasificación.

print('Resultados para Modelo')

print(classification_report(y_test, y_test_pred_xgb))

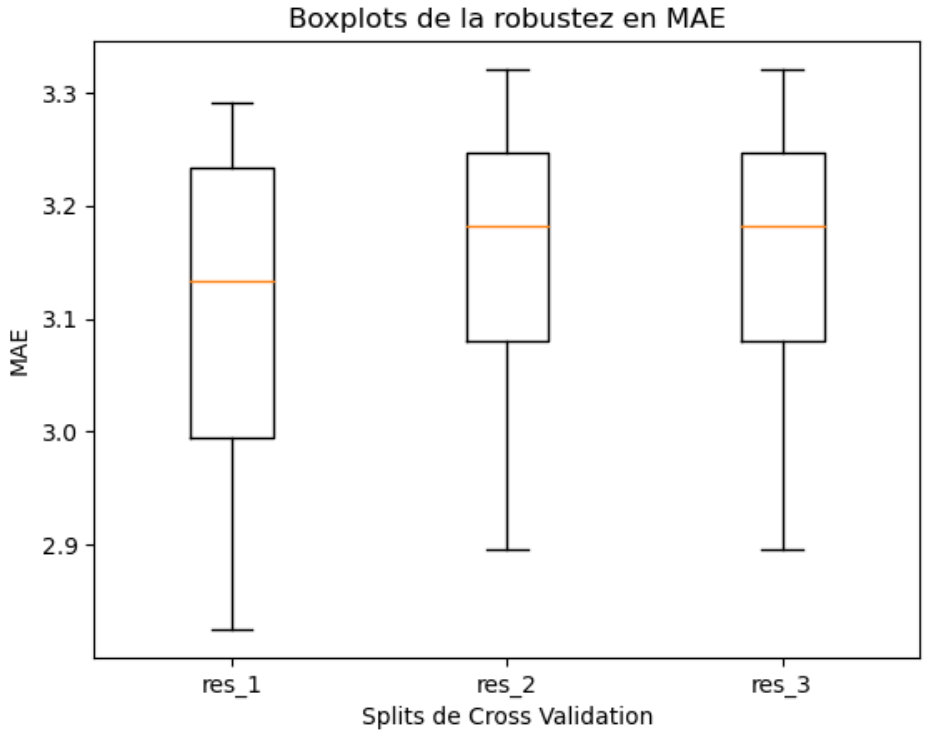
```
Resultados para Modelo
              precision    recall  f1-score   support

           0       0.77      0.83      0.80        60
           1       0.64      0.55      0.59        33

    accuracy                           0.73        93
   macro avg       0.71      0.69      0.70        93
weighted avg       0.72      0.73      0.73        93
```

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.

El script es el mismo: sólo cambia los datos de entrada y las medidas de bondad de ajuste.

```
scoring_metrics_c = {
    'MAE': make_scorer(mean_absolute_error),
    'MSE': make_scorer(mean_squared_error)
}
# los parámetros necesitan presentar formato lista.
params = {
    'n_estimators': [100,200,300],
    'eta' : [0.1,0.4,0.7],
    'gamma' : [0.1,0.5,1],
    'max_depth': [5, 10]
}
grid_search_xgb_c = GridSearchCV(estimator=xgb_regressor,
                           param_grid=params,
                           cv=4, scoring = scoring_metrics_c, refit='MAE')
grid_search_xgb_c.fit(X_train_c, y_train_c)
```



4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.

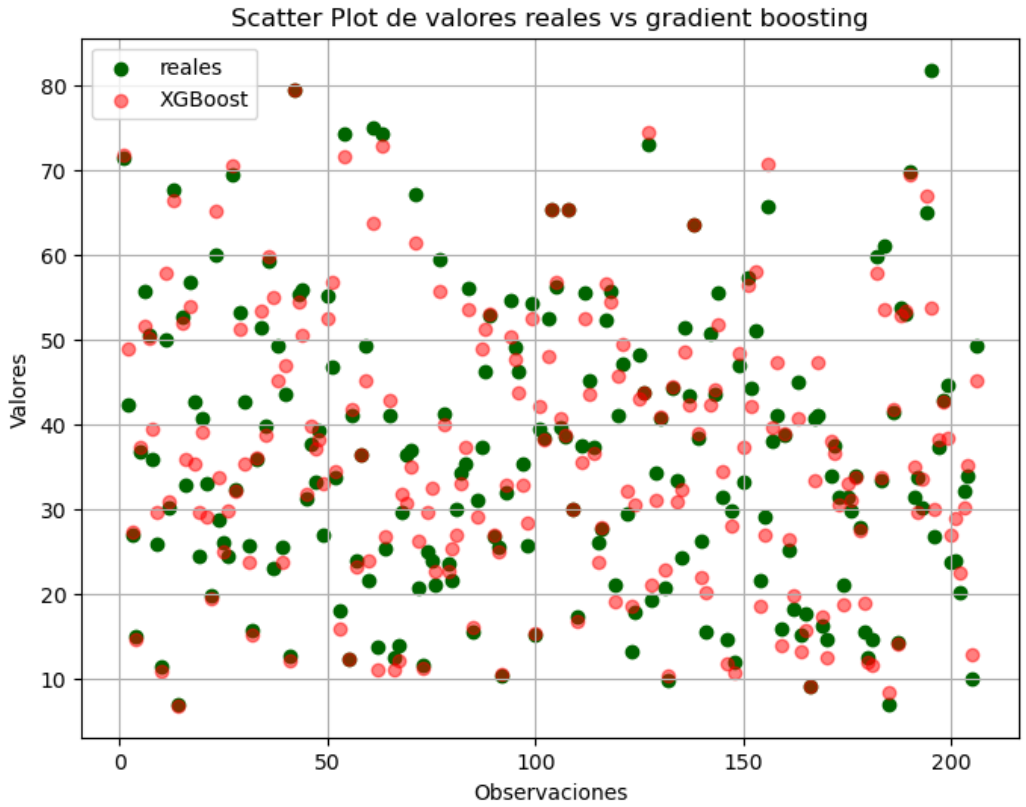
```
pred_xgb_c_train_fin = grid_search_xgb_c_fin.predict(X_train_c)

pred_xgb_c_test_fin = grid_search_xgb_c_fin.predict(X_test_c)

print(f'MAE del modelo en train:{mean_absolute_error(y_train_c,pred_xgb_c_train_fin)}')

print(f'MAE del modelo en test:{mean_absolute_error(y_test_c,pred_xgb_c_test_fin)}’)
```

MAE del modelo en train:0.7014421172859598

MAE del modelo en test:2.778095599794851

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.




```
MAE (Error Absoluto Medio): 2.78
MSE (Error Cuadrático Medio): 20.58
RMSE (Raíz del Error Cuadrático Medio): 4.54
R2: 0.9249970628297842
```

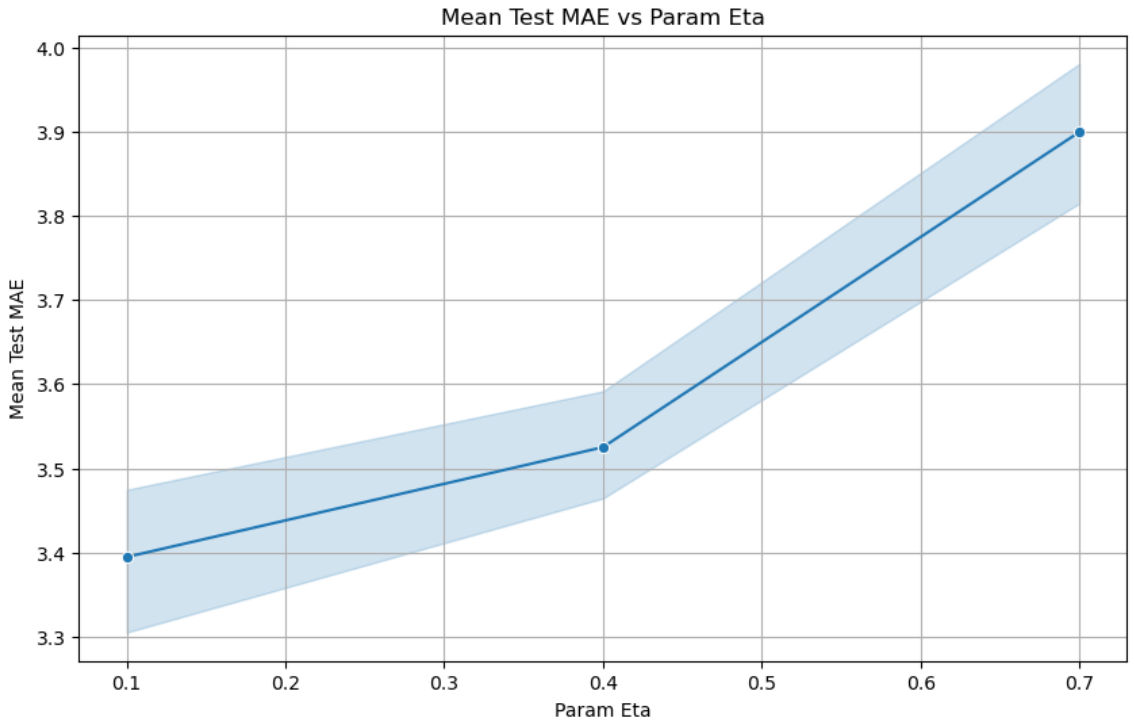
Téngase en cuenta que, dado el ejemplo propuesto, el sobreajuste es muy alto como para determinar que el ajuste del modelo es óptimo

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.

Si, por ejemplo, se quiere saber la relación entre el parámetro eta y el MAE:

```
results['param_eta'] =
results['param_eta'].astype(float)

results['mean_test_MAE'] =
results['mean_test_MAE'].astype(float)

plt.figure(figsize=(10, 6))

sns.lineplot(x='param_eta',
y='mean_test_MAE', data=results, marker='o')
```

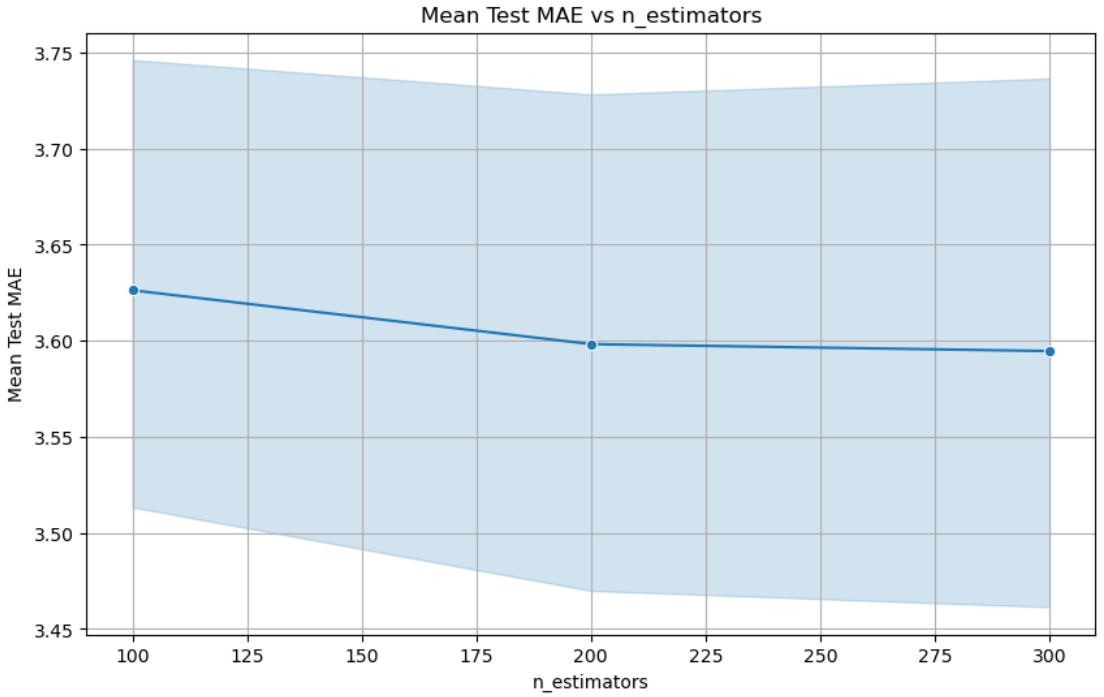


4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.

Si, por ejemplo, se quiere saber la relación entre el parámetro n_estimators y el MAE:

```
results['param_n_estimators'] =
results['param_n_estimators'].astype(float)plt.fig
ure(figsize=(10, 6))

sns.lineplot(x=param_n_estimators',
y='mean_test_MAE', data=results, marker='o')
```



4. Extreme Gradient Boosting (XGBOOST)

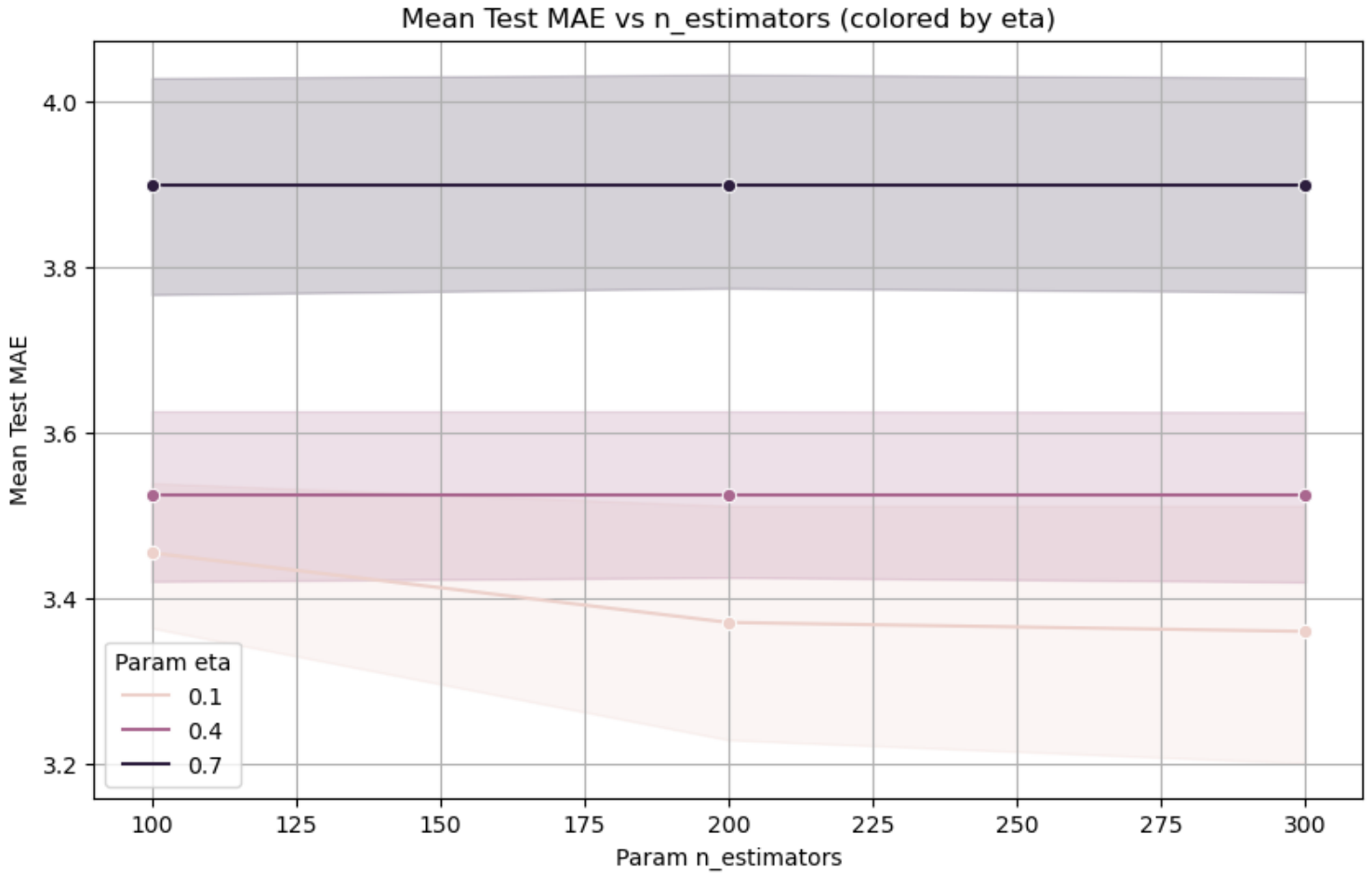
EJEMPLO de Regresión.

Así mismo, se pueden combinar ambos criterios:

```
results['param_eta'] = results['param_eta'].astype(float)
results['param_n_estimators'] = results['param_n_estimators'].astype(float)
results['mean_test_MAE'] = results['mean_test_MAE'].astype(float)

# Crear el lineplot con diferentes líneas para cada valor de 'param_n_estimators'
plt.figure(figsize=(10, 6))
sns.lineplot(x='param_n_estimators', y='mean_test_MAE', hue='param_eta', data=results, marker='o')

# Personalizar el gráfico
plt.title('Mean Test MAE vs n_estimators (colored by eta)')
plt.xlabel('Param n_estimators')
plt.ylabel('Mean Test MAE')
plt.legend(title='Param eta')
plt.grid(True)
plt.show()
```

4. Extreme Gradient Boosting (XGBOOST)

EJEMPLO de Regresión.



9. Referencias.

1. Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. 1nd ed. Boca Raton, Florida: Chapman; https://www.crcpress.com/Classification-and-Regression-Trees/Breiman-Friedman-Stone-Olshen/p/book/9780412048418.

2. Hastie, Tibshirani: *The Elements of Statistical Learning* (PDF) (En la web hay más información) http://statweb.stanford.edu/~tibs/ElemStatLearn/

3. https://scikit-learn.org/

4. https://xgboost.readthedocs.io/