XGBoost: Singularidades y parámetros

GBM es bueno pero, Por qué XGBOOST es mejor?

1. El proceso de GBM es muy lento debido a que el método de separación de nodos es "exhaustive search", Además, no seleciona las features de una manera inteligente. En cambio, XGBoost aplica:
    a. Método de muestreo para la selección de features,
    b. Control del threshold de la división de la ganancia por cada división de nodo,
    c. Utiliza un algoritmo de aproximación (de forma cuantitativa ponderada) y un histograma más rápido para la búsqueda de divisiones.

2. GBM no considera regularización para el modelo. XGBoost usa tanto L1 ó L2 mientras minimiza la función de pérdida.

3. XGBoost calcula gradientes de segundo orden (como por ejemplo, derivadas parciales de segundo orden de la función de perdida - similar al método de Newton -, lo cual provee de información acerca de la dirección del gradiente y como llegar al mínimo de nuestra función de pérdida).

4. XGBoost usa el método de ‘early stop’ para mejorar la eficiencia.

Otras singularidades de XGBoost son:

- Implementación por GPU, puede hacer el proceso de entrenamiento muy rápido.

- Bloque de columnas para el aprendizaje paralelo; Para que GBM encuentre la mejor división sobre una característica continua, los datos necesitan ser ordenados y ajustados enteramente en memoria, xgboost en cambio, para ordenar los bloques de información, se puede hacer independientemente y se puede dividir entre hilos paralelos de la CPU. La búsqueda de la división puede paralelizarse, ya que la recopilación de estadísticas para cada columna se realiza en paralelo.

- Utiliza un algoritmo que tiene en cuenta la dispersión. La entrada de Xgboost puede ser dispersa debido a razones como la codificación de un solo punto, los valores perdidos y las entradas nulas. XGboost es consciente del patrón de dispersión de los datos y sólo visita la dirección predeterminada (entradas no ausentes) en cada nodo.

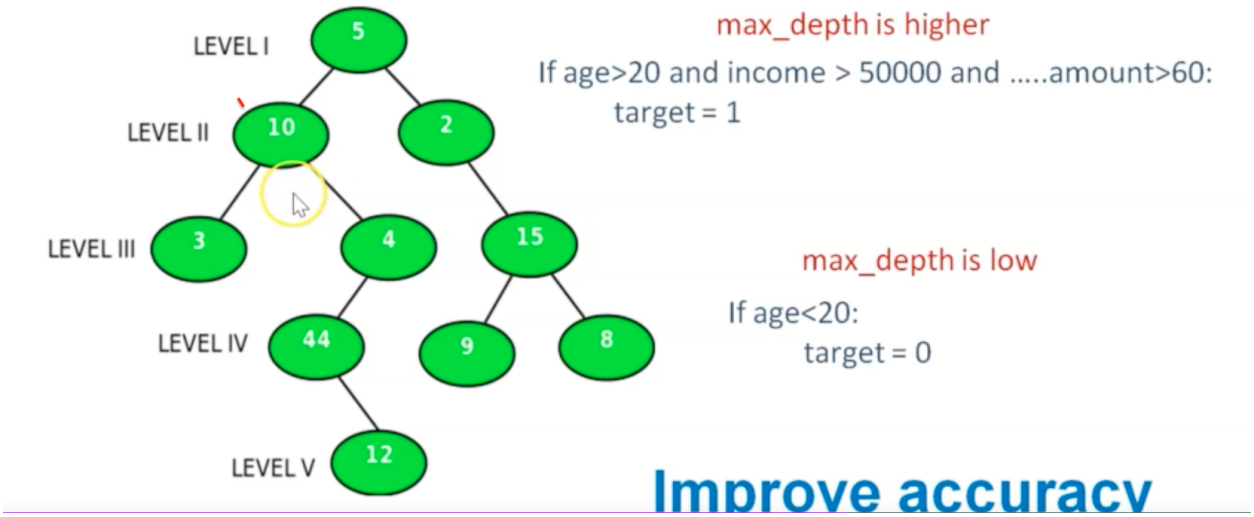
- Acceso consciente de la caché; Para evitar la pérdida de caché durante la búsqueda de divisiones y garantizar la paralelización, escoge 2ˆ16 ejemplos por bloque.

- Computación fuera del núcleo: Para los datos que no caben en la memoria principal, los divíde en varios bloques y los almacena en el disco. Comprime cada bloque por columnas y descomprime sobre la marcha por un hilo independienteh mientras se lee el disco.

Parámetros importantes en la construcción de un modelo XGBoost (clasificador o regresor)

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
              validate_parameters=False, verbosity=None)
```

- **max_depth [default = 6]:** Especifíca la profundidad de los árboles, mientras más alto el nivel más se aumenta el efecto 'cross', resuelve interacciones no lineales. Valores típicos van de 3 a 12. Se recomienda determinar este valor usando cross validation.



- **objective [default = reg:squarederror]**: Esto es para definir el tipo de target que xgboost predecirá.

- reg:squarederror:

$$\sqrt{\frac{\sum_{t=1}^{T}(\hat{y}-y_t)^2}{T}}$$

  - reg: squaredlogerror: Regresión con pérdida logaritmica cuadráda.

$$\frac{1}{2}[log(actual+1)-log(predict+1)]^2$$

  - reg:logistic: Regresión logística.

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

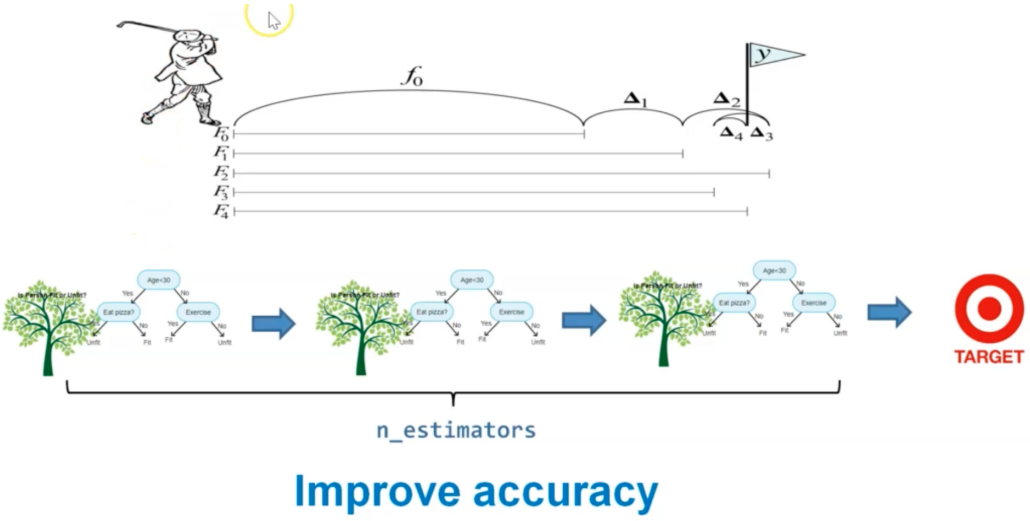
  - binary:logistic: Regresión logística para clasificación binaria, su salida es una probabilidad, métrica por defecto para evaluación la tasa de error en clasificación binaria.

$$H_p(q) = -\frac{1}{N}\sum_{i=1} y_i * log(p(y_i)) + (1-y_i) * log(1-p(y_i))$$

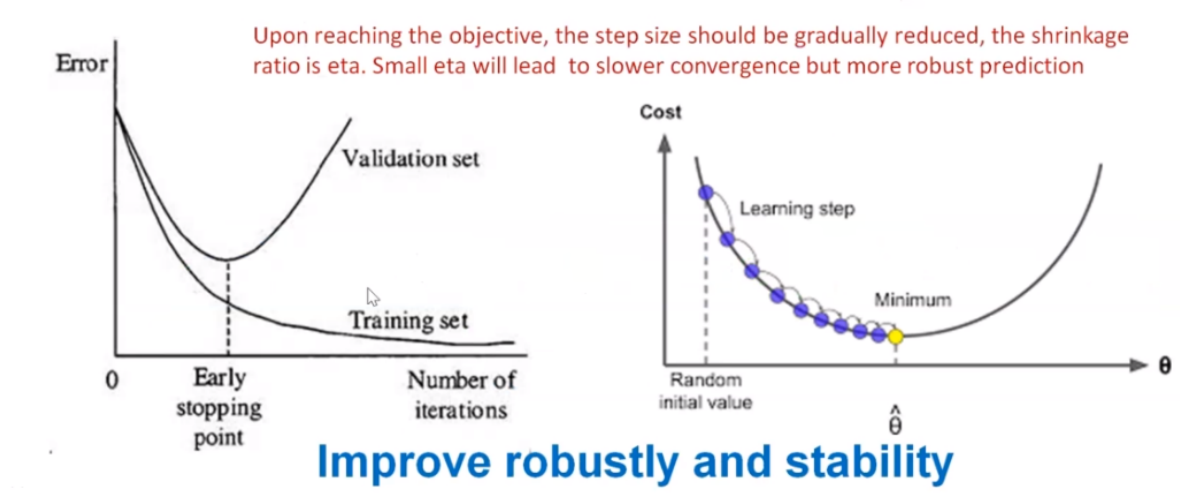
  - multi:softmax: Generalmente usada para clasificación multiclase. Retorna la clase predecida. Asegurarse de fijar el parámetro *num_class* para definir el número total de clases únicas.

  - multi:softprob: Al igual que softmax, pero returna la probabilidad predecida de cada data point a ser pertenenciente a cada clase.

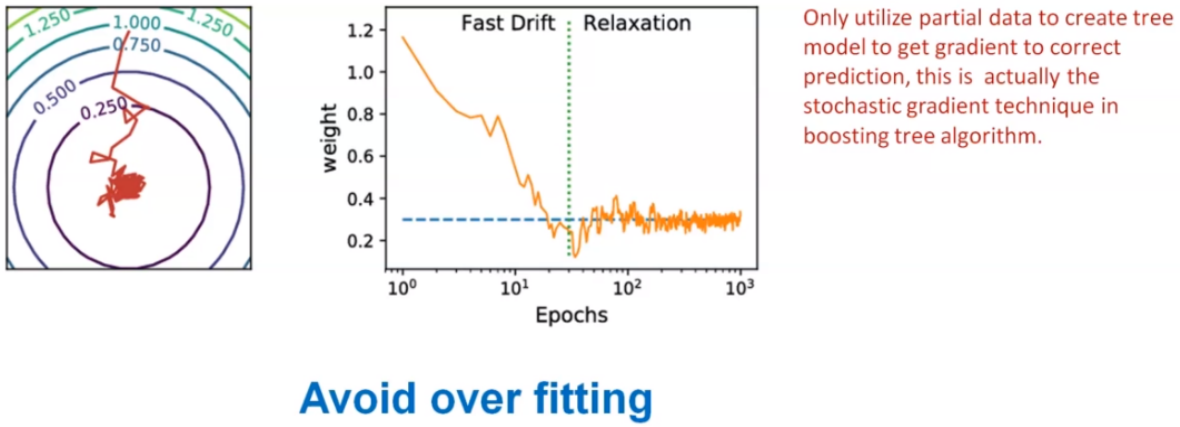
- **n_estimators [default = 100]**: El número de modelos de árboles usados en el proceso iterativo. También es el número de pasos de iteración en el método boost. También se recomienda fijarlo posterior a un proceso de cross validation.



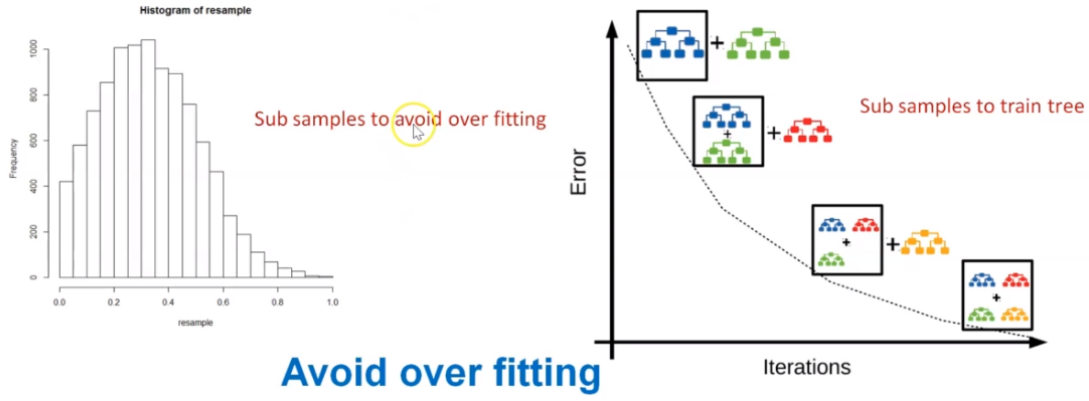
- **eta / learning rate [default=0.3]:** Parámetro muy importante en xgboost, es la tasa de aprendizaje para cada paso de corrección de predicción en xgboost. Se refiere a la tasa de aprendizaje utilizada por el algoritmo para actualizar los pesos de los árboles de decisión en el modelo. Este hiperparámetro le permite a XGBoost controlar la velocidad a la que se actualizan los pesos del modelo durante el entrenamiento. Cuando se establece en un valor bajo, como 0.1, el algoritmo actualiza los pesos del modelo de forma más lenta, lo que puede ayudar a mejorar la precisión del modelo pero también aumentar la cantidad de tiempo necesario para entrenar el modelo. Por otro lado, cuando se establece en un valor alto, como 0.9, el algoritmo actualiza los pesos del modelo de forma más rápida, lo que puede ayudar a reducir el tiempo de entrenamiento pero también aumentar el riesgo de no converger a una solución óptima. En resumen, el hiperparámetro learning rate en XGBoost le permite controlar la velocidad a la que se actualizan los pesos del modelo durante el entrenamiento.. Debe existir un equilibrio entre eta y n_estimators.



- **subsample [default=1]:** El hiperparámetro subsample en XGBoost se refiere a la fracción de muestras de entrenamiento que se utilizan para entrenar cada árbol de decisión en el modelo. Este hiperparámetro le permite a XGBoost controlar la complejidad del modelo y reducir el sobreajuste. Cuando se establece en un valor bajo, como 0.5, solo se utilizan la mitad de las muestras de entrenamiento para entrenar cada árbol de decisión, lo que puede ayudar a reducir el sobreajuste y mejorar la capacidad de generalización del modelo. Por otro lado, cuando se establece en un valor alto, como 1.0, se utilizan todas las muestras de entrenamiento para entrenar cada árbol de decisión, lo que puede ayudar a mejorar la precisión del modelo pero también aumentar el riesgo de sobreajuste. En resumen, el hiperparámetro subsample en XGBoost le permite controlar la complejidad del modelo y reducir el sobreajuste. Intente empezar con 0,8.



- **colsample_bytree [default=1]:** Este valor es la fracción de selección de features para construir el árbol de decisión en el paso de boosting. Tenga en cuenta que el submuestreo se realiza una vez por cada árbol construido. Se refiere a la fracción de columnas de datos que se utilizan para entrenar cada árbol de decisión en el modelo. Este hiperparámetro le permite a XGBoost controlar la complejidad del modelo y reducir el sobreajuste. Cuando se establece en un valor bajo, como 0.5, solo se utilizan la mitad de las columnas de datos para entrenar cada árbol de decisión, lo que puede ayudar a reducir el sobreajuste y mejorar la capacidad de generalización del modelo. Por otro lado, cuando se establece en un valor alto, como 1.0, se utilizan todas las columnas de datos para entrenar cada árbol de decisión, lo que puede ayudar a mejorar la precisión del modelo pero también aumentar el riesgo de sobreajuste. En resumen, el hiperparámetro colsample_bytree en XGBoost le permite controlar la complejidad del modelo y reducir el sobreajuste. Los valores típicos van de 0,5 a 1. Pruebe a empezar con 0,9.



  - **lambda [default=1]:** Término de regularización L2 en los pesos (regresión Ridge).

    La regresión Ridge es una técnica de aprendizaje automático utilizada para regularizar un modelo de regresión. La regularización es un proceso que se utiliza para evitar el sobreajuste del modelo, lo que puede mejorar su capacidad de generalización. La regresión Ridge funciona agregando una

penalización a la función de costo del modelo de regresión para suavizar los coeficientes del modelo. Esto ayuda a evitar que los coeficientes del modelo se vuelvan muy grandes, lo que puede mejorar la precisión del modelo y evitar el sobreajuste. En resumen, la regresión Ridge es una herramienta útil para regularizar un modelo de regresión y mejorar su capacidad de generalización. Se recomienda empezar con 0.5 y luego ir ajustando.

$$Cost = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

- **alpha [default=0]:** Término de regularización L1 en el peso (regresión Lasso).

  La regresión Lasso es una técnica de aprendizaje automático utilizada para seleccionar las variables más importantes en un modelo de regresión. La palabra "Lasso" proviene de la expresión en inglés "least absolute shrinkage and selection operator", que significa "operador de selección y encogimiento absoluto mínimo". La regresión Lasso funciona agregando una penalización al modelo de regresión para obligar a algunos coeficientes a ser cero, lo que ayuda a eliminar las variables menos importantes del modelo. Esto puede ayudar a mejorar la precisión del modelo y hacerlo más fácil de interpretar. En resumen, la regresión Lasso es una herramienta útil para seleccionar las variables más importantes en un modelo de regresión y mejorar su precisión y facilidad de interpretación. Se recomienda empezar con 0.5 y luego ir ajustando.

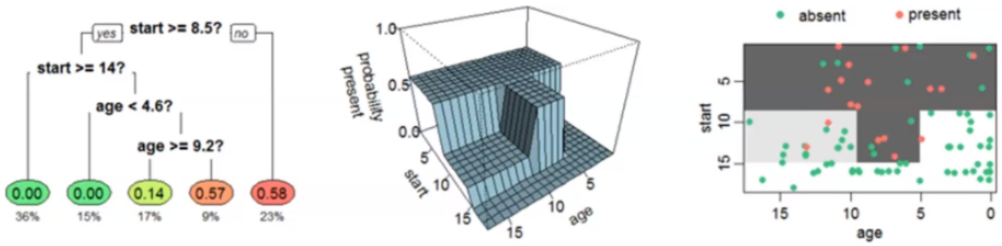
$$Cost = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

En XGBoost, la regresión Ridge y la regresión Lasso son técnicas de regularización utilizadas para mejorar la precisión y la capacidad de generalización de los modelos de árboles de decisión. Ambas técnicas utilizan

una penalización en la función de costo del modelo de árboles de decisión para evitar el sobreajuste. Sin embargo, la regresión Ridge (L2) utiliza una penalización basada en la suma de los cuadrados de los coeficientes del modelo, mientras que la regresión Lasso (L1) utiliza una penalización basada en la suma absoluta de los coeficientes del modelo. Esta diferencia en el tipo de penalización utilizada puede dar lugar a resultados ligeramente diferentes en términos de la selección de variables y la precisión del modelo en XGBoost. En resumen, la regresión Ridge y la regresión Lasso son técnicas de regularización ligeramente diferentes utilizadas en XGBoost para mejorar la precisión y la capacidad de generalización de los modelos de árboles de decisión.

- **min_child_weight [defaul=1]**

  El hiperparámetro min_child_weight en XGBoost se refiere al peso mínimo que debe tener un nodo hijo en un árbol de decisión para que se realice una división en el árbol. Este hiperparámetro se utiliza para controlar la complejidad del modelo y evitar el sobreajuste. Cuando se establece en un valor bajo, como 1, el árbol de decisión puede realizar divisiones en nodos hijos con un peso muy pequeño, lo que puede resultar en un modelo más complejo y con mayor riesgo de sobreajuste. Por otro lado, cuando se establece en un valor alto, como 10, el árbol de decisión solo puede realizar divisiones en nodos hijos con un peso relativamente alto, lo que puede resultar en un modelo más simple y con menor riesgo de sobreajuste. En resumen, el hiperparámetro min_child_weight en XGBoost le permite controlar la complejidad del modelo y reducir el riesgo de sobreajuste. Por lo general, se recomienda empezar con un valor moderado para este hiperparámetro, como 1, y luego ajustarlo en iteraciones posteriores basándose en los resultados obtenidos en el conjunto de datos de prueba y ajustarlo en iteraciones posteriores..



Este parámetro simplemente coincide con el número mínimo de instancias necesarias para estar en cada nodo. Cuanto mayor sea min_child_weight, más conservador será el algoritmo.

- **tree_method [default = auto]:**

  El hiperparámetro tree_method en XGBoost se refiere al algoritmo utilizado para construir los árboles de decisión en el modelo. Este hiperparámetro le permite a XGBoost controlar la velocidad y la precisión del entrenamiento del modelo. Hay varios valores posibles para este hiperparámetro, como 'auto', 'exact' y 'approx', cada uno de los cuales utiliza un algoritmo diferente para construir los árboles de decisión.

  - 'auto' utiliza un algoritmo automático que selecciona el mejor algoritmo en función del conjunto de datos y del tipo de problema que se está tratando de resolver.
  - exact' utiliza un algoritmo exacto que construye los árboles de decisión de forma precisa pero puede ser lento para conjuntos de datos grandes.
  - 'approx' utiliza un algoritmo aproximado que construye los árboles de decisión de forma rápida pero con menor precisión.
  - ‘hist’ algoritmo greedy aproximado optimizado para histogramas más rápido.
  - ‘hist_gpu’ implementación de GPU de ‘hist’.

En resumen, el hiperparámetro tree_method en XGBoost le permite controlar la velocidad y la precisión del entrenamiento del modelo.

- **importance_type** [defualt='gain']:

  El hiperparámetro importance_type en XGBoost se refiere al tipo de importancia que se calcula para cada característica en el modelo de árboles de decisión. Este hiperparámetro se utiliza para evaluar qué tan importantes son las características individuales en el modelo. Hay varios valores posibles para este hiperparámetro, como 'weight', 'gain' y 'cover', cada uno de los cuales calcula la importancia de las características de forma diferente.

  - 'weight' calcula la importancia de las características en función del peso de los nodos que contienen esa característica en los árboles de decisión.
  - 'gain' calcula la importancia de las características en función del aumento en la precisión del modelo que se obtiene al utilizar esa característica.
  - 'cover' calcula la importancia de las características en función del número de veces que se utiliza esa característica para realizar divisiones en los árboles de decisión.

  En resumen, el hiperparámetro importance_type en XGBoost le permite evaluar qué tan importantes son las características individuales en el modelo. se recomienda utilizar el valor 'gain' para este hiperparámetro, ya que este valor calcula la importancia de las características en función del aumento en la precisión del modelo que se obtiene al utilizar esa característica. Esto puede ser útil para identificar las características más importantes en el modelo y mejorar su precisión. En resumen, aunque no hay un valor recomendado específico para el hiperparámetro importance_type en una primera iteración, se recomienda utilizar el valor 'gain' para calcular la importancia de las características en el modelo.

- **scale_post_weight** [default=1]

  El hiperparámetro scale_post_weight en XGBoost se refiere a la forma en que se escalan los pesos de los árboles de decisión en el modelo después de cada iteración de entrenamiento. Este hiperparámetro se utiliza para mejorar la precisión del modelo y evitar el sobreajuste. Cuando se establece en un valor bajo, como 0.1, los pesos de los árboles de decisión se escalan de forma más lenta después de cada iteración de entrenamiento, lo que puede ayudar a mejorar la precisión del modelo pero también aumentar la cantidad de tiempo necesario para entrenar el modelo. Por otro lado, cuando se establece en un valor alto, como 0.9, los pesos de los árboles de decisión se escalan de forma más rápida después de cada iteración de entrenamiento, lo que puede ayudar a reducir el tiempo de entrenamiento pero también aumentar el riesgo de no converger a una solución óptima. Un valor mayor a 0 es recomendado en caso de tener clases muy imbalanceadas.

- **random_state** [default=0]

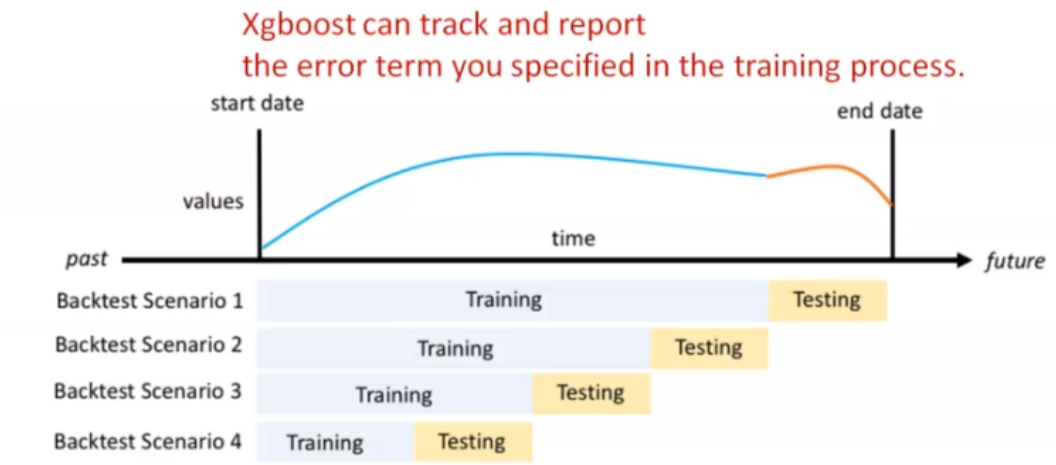
  El número del seed, usado para generara resultados reproducibles.

- **nthread** [default to max number of threads available if not set]

  Se utiliza para el procesamiento paralelo y debe introducirse el número de núcleos del sistema. Si desea ejecutar en todos los núcleos, no debe introducir el valor y el algoritmo lo detectará automáticamente.

- **eval_metric:**

  Se pueden especificar métricas de evaluación en fit() o train(), una métrica por defecto será asignada de acuerdo al objetivo (rmse para regresión, error para clasificación, precisión media promedio para ranking). El usuario también puede añadir múltiples métricas de evaluación.



- **early_stopping_rounds:**

  Se utiliza para determinar el número de iteraciones de entrenamiento que se llevarán a cabo antes de detener el entrenamiento si no se observa una mejora en la función de pérdida. Esto se conoce como detención temprana y se utiliza para evitar el sobreentrenamiento.