XGBoost

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.

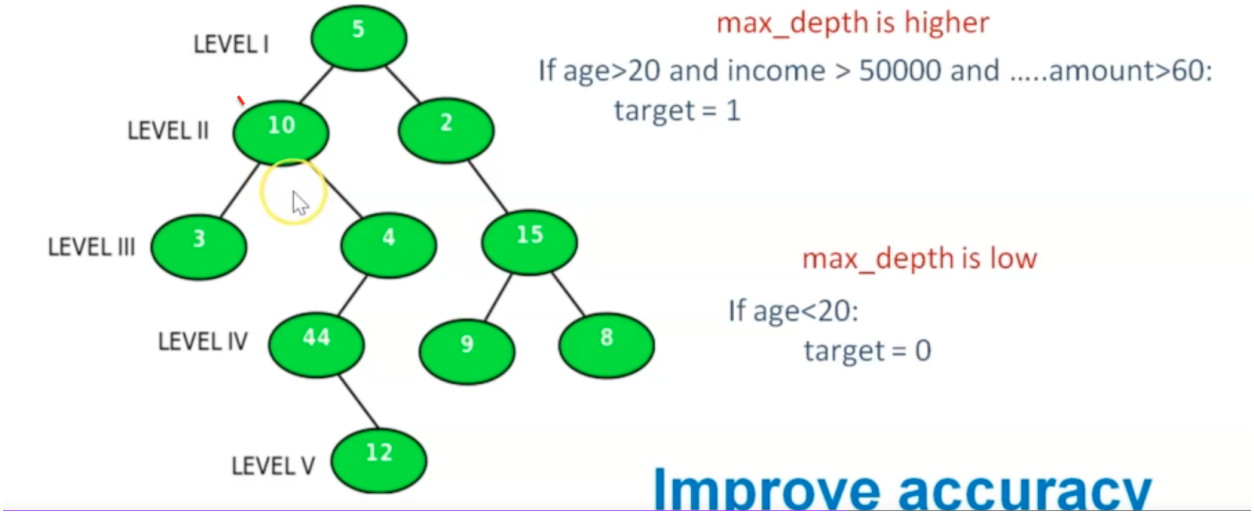
- Acceso consciente de la caché; Para evitar la pérdida de caché durante la búsqueda de divisiones y garantizar la paralelización, escoge 2ˆ16 ejemplos por bloque.

- 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.

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

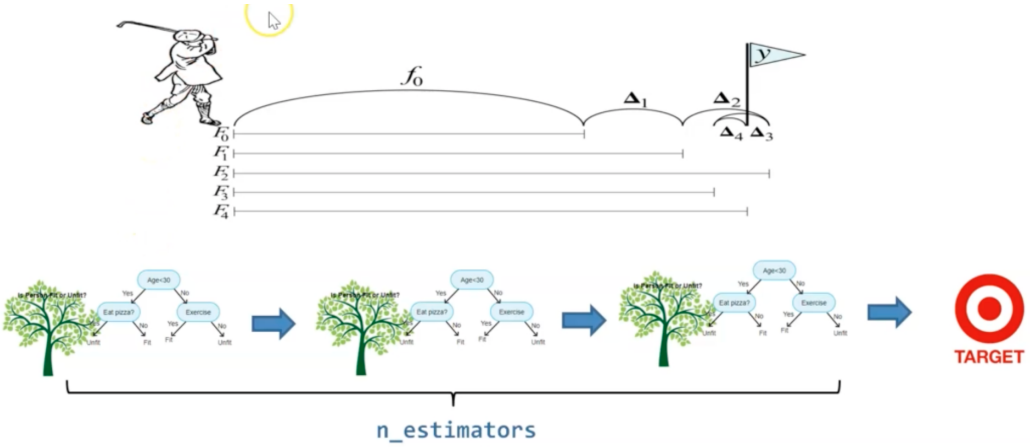
- binary:logistic: Regresión logística para clasificación binaria, su salida es una probabilidad, métrica por defecto para evaluación la tasa de error en clasificación binaria.

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

- 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.



Improve accuracy