

# Entendiendo las debilidades de un árbol de decisión & GBM

## De árboles de decisión hasta un GBM (gradient boosting model)

Si un nodo de un árbol es particionado en hojas muy granulares, entonces el modelo de árbol es muy probable que se sobreajuste (overfitting), por lo que debo tener en cuenta lo siguiente:

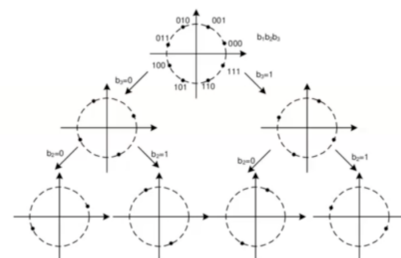
- El número de observaciones por hoja no debe ser muy bajo,
- La profundidad del árbol no debe ser muy grande,
- El número de hojas puede causar un problema de sobreajuste,
- Tener en cuenta la regularización (prune).

Un modelo tipo GBM ataca los problemas anteriores reduciendo la complejidad

# From decision tree to GBM



Increase efficiency



- 2 If you use and split all the features in decision tree, then training process is **time consuming** – you must know how to choose a candidate to split nodes
  - use sampling ways to select features in building a tree
  - control threshold of split gini, entropy, mse or other gain measures
  - use non- exhaustive search (sorting is slow) way to find split points, such as Histogram-based methods (e.g. in xgboost method)
  - consider weighted sampling on target

`sum(negative instances) / sum(positive instances)`

Si usamos y dividimos todas las variables de entrada al modelo (features), entonces el proceso de entrenamiento es demasiado costoso en tiempo, debo saber como escoger un candidato para dividir los nodos.

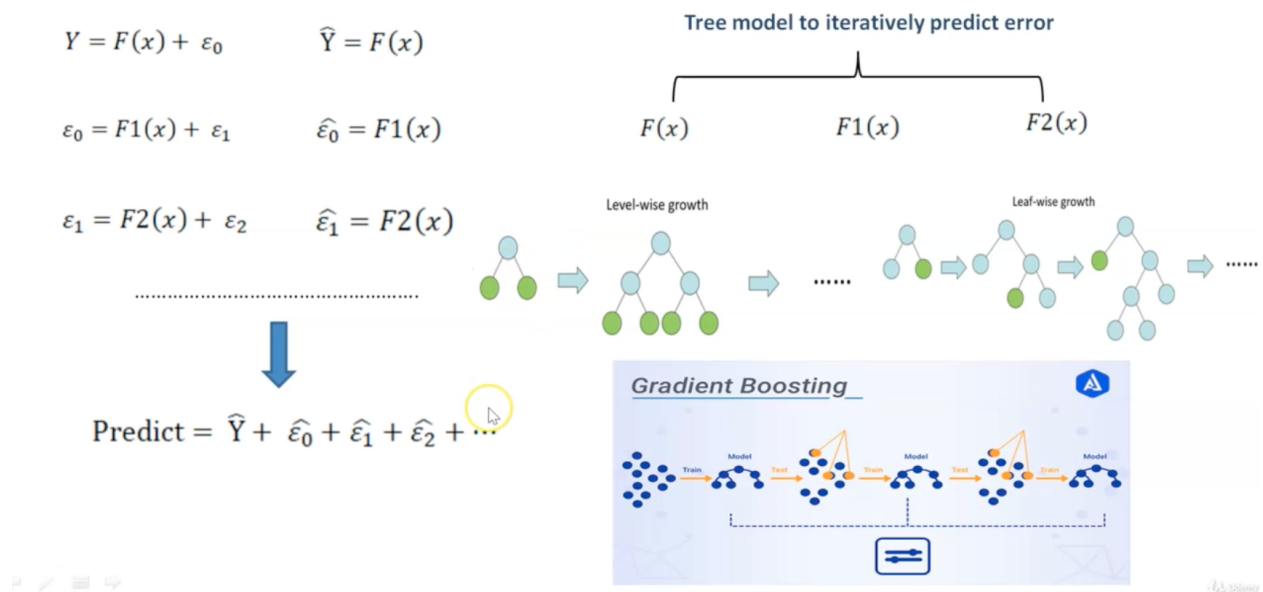
- Utilizar formas de muestreo para seleccionar características en la construcción de un árbol,
- Umbral de control del threshold del gini, entropía, mse u otras medidas de ganancia,
- Usa un método de búsqueda no exhaustivo para encontrar los puntos de separación de los nodos (métodos sort son lentos), como los métodos basados en histogramas (e.g. in xgboost method).
- Considerar un muestreo ponderado en el target.  $\frac{\sum negative\ instances}{\sum positive\ instances}$

## Construir un GBM desde un árbol de decisión

En secciones anterior, se definió detalladamente el como es la lógica de construcción de un modelo GBM, como lo muestra la siguiente imagen ya vista anteriormente, en donde se puede resumir como:

1. Construir un primer modelo el cual prediga la target definida,
2. Construir un segundo modelo que prediga el error del j-1 modelo, siendo j el modelo actual,
3. Repetir el paso 2 iterativamente.

## From decision tree to GBM



En pseudo-código, construimos un modelo de árbol de clasificación llamada  $T_0$  para predecir una target, declaramos que el modelo ensamble:  $GBM \text{ Model} = T_0$ , usamos el modelo clasificador para predecir (con el método `predict_proba()`), para obtener  $p\_target$ , por cada observación calculamos un error  $error_0 = target - p\_target$ , una vez hecho eso, hacemos lo siguiente:

- Construimos un modelo tipo árbol de **regresión** el cual será  $T_{(j+1)}$  para predecir  $error_j$  y así obtener  $p\_error_j$ ,
- $GBM\_Model = GBM\_Model + T_{(j+1)}$ ,

- $error_{(j+1)} = error_j - p\_error_j$ ,
- $j = j + 1$

En código

```
it_times = 10
tree_entropy = DecisionTreeClassifier(criterion = 'entropy', random_state = 12,
                                     max_depth = 3, min_samples_leaf = 5)

tree_entropy.fit(X_train, y_train)

prediction = tree_entropy.predict_proba(X_train)[:, 1]

prediction_test = tree_entropy.predict(X_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, prediction_test)
roc_auc = metrics.auc(fpr, tpr)
print ("auc at 0 step: ", roc_auc)

err = y_train - prediction
booster = [tree_entropy]

for j in range(it_times):
    j = j + 1
    tree_model = DecisionTreeRegressor(criterion = 'mse', random_state = 12,
                                       max_depth = 3, min_samples_leaf = 5)

    tree_model.fit(X_train, err)
    err_predict = tree_model.predict(X_train)
    prediction = prediction + err_predict
    err = err - err_predict
    booster.extend([tree_model])

    err_predict_test = tree_model.predict(X_test)
    prediction_test = prediction_test + 0.02*err_predict_test
    fpr, tpr, thresholds = metrics.roc_curve(y_test, prediction_test)
    roc_auc = metrics.auc(fpr, tpr)
    print ("auc at iteration " + str(j) + " step: ", roc_auc)
```

Output:

```
course_code — ~/Documents/Educational/xgboost/course_code — zsh — 68...

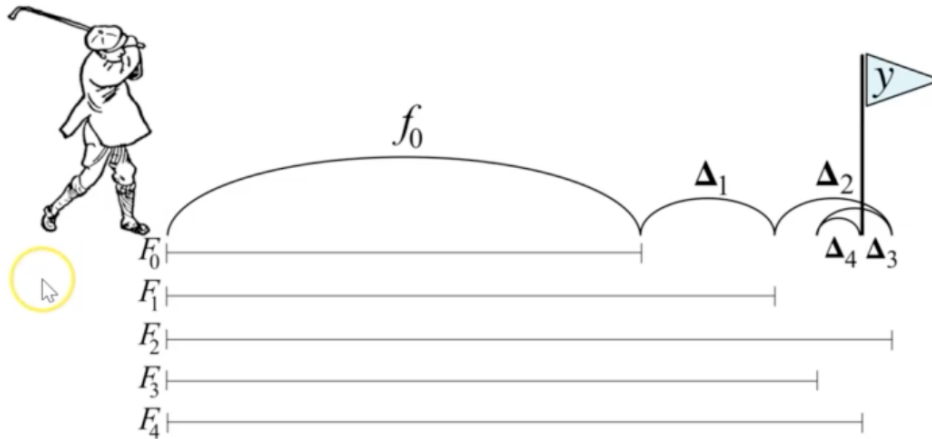
~/Documents/Educational/xgboost/course_code main*
[xgboost_env > python3 yourbooster.py
features:
['variance', 'skewness', 'curtosis', 'entropy', 'class']
row and column number
(1372, 5)
data / feature types:
variance      float64
skewness      float64
curtosis      float64
entropy      float64
class        int64
dtype: object
missing values:
variance      0
skewness      0
curtosis      0
entropy      0
class        0
dtype: int64
auc at 0 step: 0.9290452499407723
auc at iteration 1 step: 0.9698629866540315
auc at iteration 2 step: 0.9775625839058674
auc at iteration 3 step: 0.9933566295506594
auc at iteration 4 step: 0.994027876490563
auc at iteration 5 step: 0.9882136934375741
auc at iteration 6 step: 0.9904840874990128
auc at iteration 7 step: 0.9927939666745637
auc at iteration 8 step: 0.992379372976388
auc at iteration 9 step: 0.9924188580904999
auc at iteration 10 step: 0.9907604832977968
```

## Por qué una estrategia GBM es mejor que un árbol de decisión?

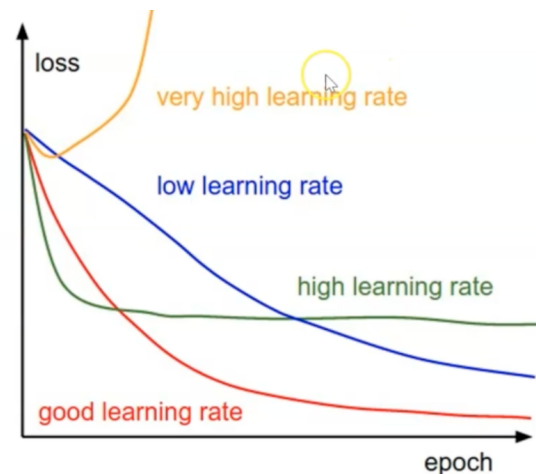
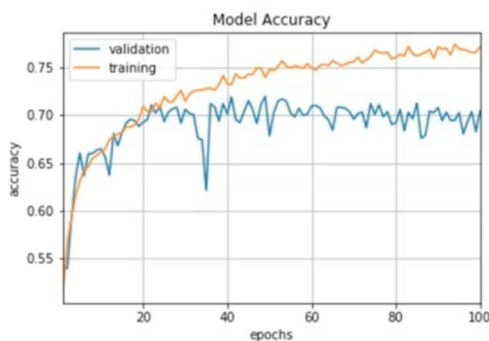
1. Dado que GBM es un tipo de ensamble aditivo de modelos de árbol, en otras palabras, usa el error de la predicción original para corregir la predicción del árbol anterior, esto incrementará el accuracy del modelo.

Try to extract useful stuff from garbage

Try complete whole assignment from leftover



2. Porque el GBM es un modelado iterativo y considera el 'learning rate', subsamples, etc, esto mejorará la eficiencia del algoritmo en términos de convergencia y velocidad.



3. GBM considera el número de iteraciones (el número de modelos de árbol) a ser usados, esto afecta tanto el accuracy, la estabilidad y la eficiencia (velocidad).

