Boosting Algorithm

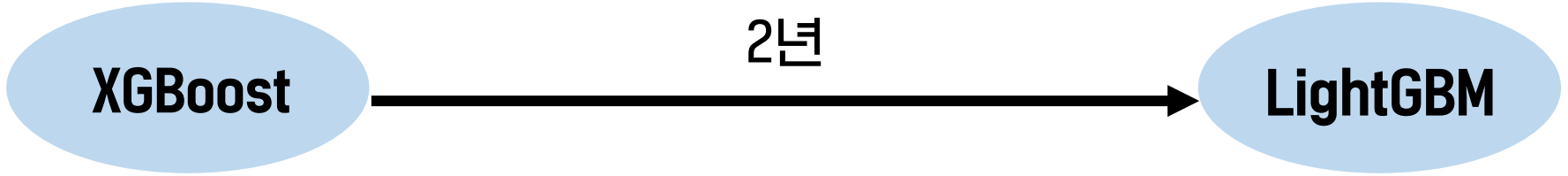
LightGBM

2022.02.14

황성아

Abstract

“XGBoost와 함께 부스팅 계열 알고리즘에서 각광 받고 있음”



- 학습시간이 너무 오래 걸림

+

- XGBoost와 예측 성능 비슷
- 학습에 걸리는 시간이 훨씬 적음
- 메모리 사용량도 상대적으로 적음

Leaf-wise tree growth
GOSS(Gradient Based One Side Sampling)
EFB(Exclusive Feature Bundling)

LightGBM

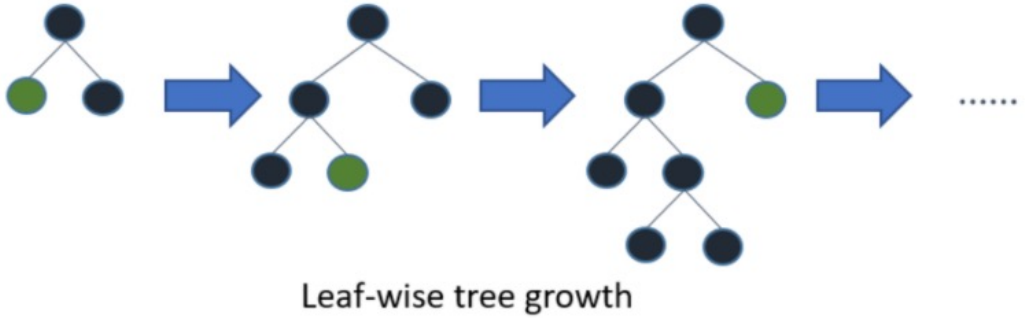
Leaf-wise tree growth

<일반GBM>

균형 트리 분할

- Root Node와 가까운 Node를 우선적으로 대칭 분할
- 트리의 깊이가 최소화 될 수 있음
- 균형 잡힌 트리 생성 가능
- 과적합에 강함
- 균형을 맞추기 위한 시간이 필요

<LightGBM>



리프 중심 트리 분할

- 최대 손실 값을 가지는 Leaf Node를 지속적으로 분할
- 깊은 비대칭 트리 형성
- 균형 트리 분할 방식보다 예측 오류 손실 최소화

GOSS (Gradient Based One Side Sampling)

XGBoost

Algorithm 1: Histogram-based Algorithm

Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    for $k = 1$ ***to*** m **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      for j ***in*** *usedRows* **do**
        bin $\leftarrow I$.f[k][j].bin
        H[bin].y $\leftarrow H$[bin].y + I.y[j]
        H[bin].n $\leftarrow H$[bin].n + 1
      Find the best split on histogram H.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

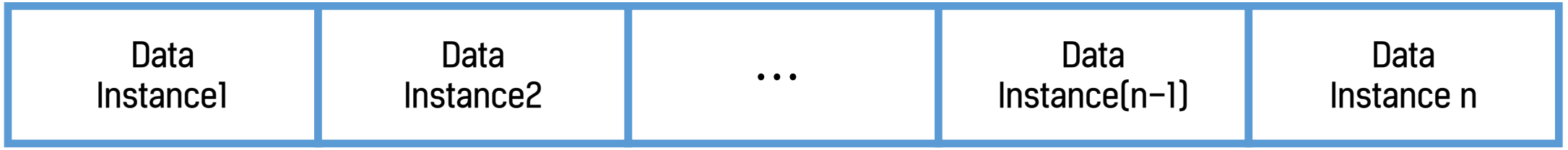
LightGBM

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models $\leftarrow$ { }, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len(I) , randN $\leftarrow$ b $\times$ len(I)
for $i = 1$ ***to*** d **do**
  preds $\leftarrow$ models.predict(I)
  g $\leftarrow loss(I,$ preds), w $\leftarrow$ {1,1,...}
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times =$ fact ▷ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

GOSS (Gradient Based One Side Sampling)

"Gradient가 큰 데이터들을 그대로 유지, 작은 데이터들은 랜덤 샘플링"



Small Gradient

random b ×100 %

$* \frac{1-a}{b} > 1$ *일 때, 효과가 좋음*

ex) a=0.1, b =0.9 -> 1
a = 0.05, b = 0.6 -> 0.95/0.5 > 1

Large Gradient

상위 a ×100 %: 모두 사용

그래디언트가 크다는 것은 잔차(residual)가 크다는 것이고, 결국 GOSS는 잔차가 큰 관측치는 그대로 두고,
잔차가 작은 관측치의 수를 줄여 학습시간과 well-trained 관측치가 학습에 미치는 영향을 모두 줄임
-> 올바르게 훈련되지 않은(under-trained) 관측치에 더 높은 가중치를 부여하여 다음 학습에서 보다 더 많은 focus를 받도록 함

EFB (Exclusive Feature Bundling)

$$\{x_1, x_4, x_7\} \rightarrow y_1$$
$$\{x_2, x_5, x_6, x_9\} \rightarrow y_2$$

"Sparse Data 에서 상호 배타적인 Feature들을 하나의 Feature로 묶어 속도를 높이는 방법"

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow$ G.sortByDegree()
bundles $\leftarrow$ { }, bundlesConflict $\leftarrow$ { }
for i ***in*** *searchOrder* **do**
  needNew $\leftarrow$ True
  for $j = 1$ ***to*** *len(bundles)* **do**
    cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
    if *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add(F[i]), needNew $\leftarrow$ False
      break
  if *needNew* **then**
    Add $F[i]$ as a new bundle to *bundles*
Output: *bundles*

지금 현재 존재하는 피쳐 셋들에 대해 어떤 피쳐들을 하나의 번들로 묶을 것인지 결정

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow$ {0}, totalBin $\leftarrow$ 0
for f ***in*** F **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
  newBin[i] $\leftarrow$ 0
  for $j = 1$ ***to*** *len*(F) **do**
    if $F[j].bin[i] \neq 0$ **then**
      newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]
Output: $newBin, binRanges$

실질적으로 번들링이 되어야 하는 변수들을 이용해서 하나의 변수로 값을 표현하고자 하는 과정

EFB (Exclusive Feature Bundling)

Greedy Bundling example

동시에 0이 아닌 부분의 개수

	x_1	x_2	x_3	x_4	x_5
I_1	1	1	0	0	1
I_2	0	0	1	1	1
I_3	1	2	0	0	2
I_4	0	0	2	3	1
I_5	2	1	0	0	3
I_6	3	3	0	0	1
I_7	0	0	3	0	2
I_8	1	2	3	4	3
I_9	1	0	1	0	0
I_{10}	2	3	0	0	2

	x_1	x_2	x_3	x_4	x_5
x_1	-	6	2	1	6
x_2	6	-	1	1	6
x_3	2	1	-	3	4
x_4	1	1	3	-	3
x_5	6	6	4	3	-

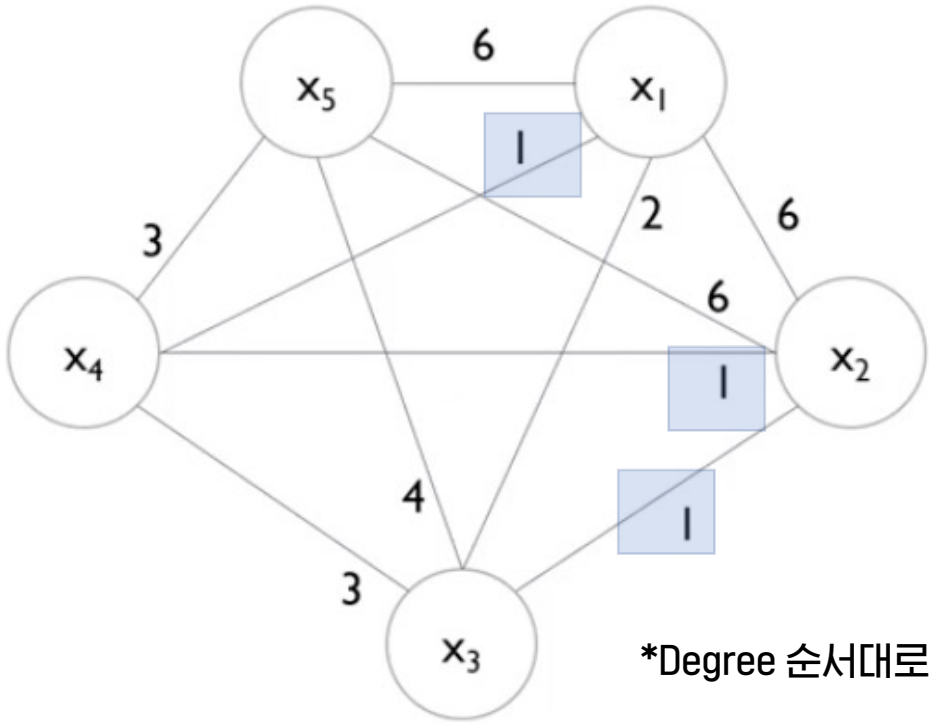
	x_5	x_1	x_2	x_3	x_4
d	19	15	14	10	8

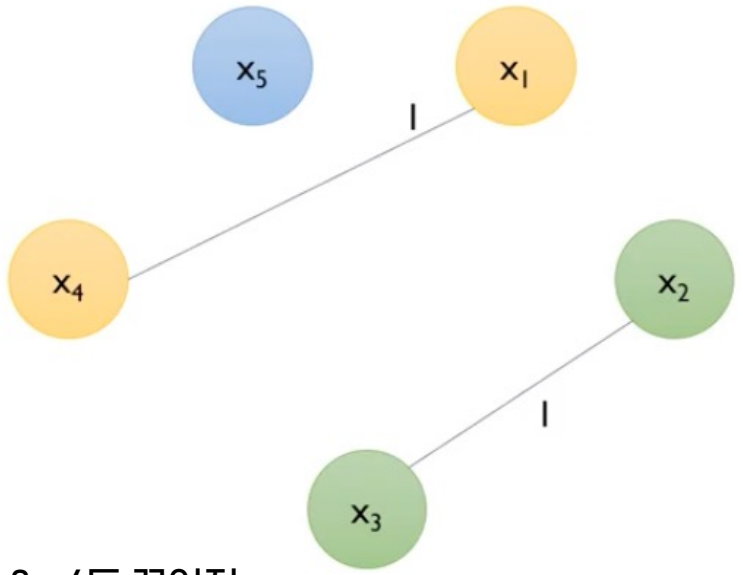
*Degree: 강도

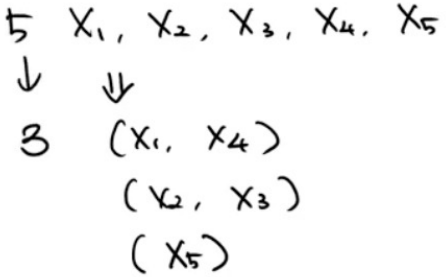
EFB (Exclusive Feature Bundling)

Greedy Bundling example

Cut off =0.2 -> n=10 , 2 이상







*Degree 순서대로 끊기 때문에 x2, x4도 끊어짐

EFB (Exclusive Feature Bundling)

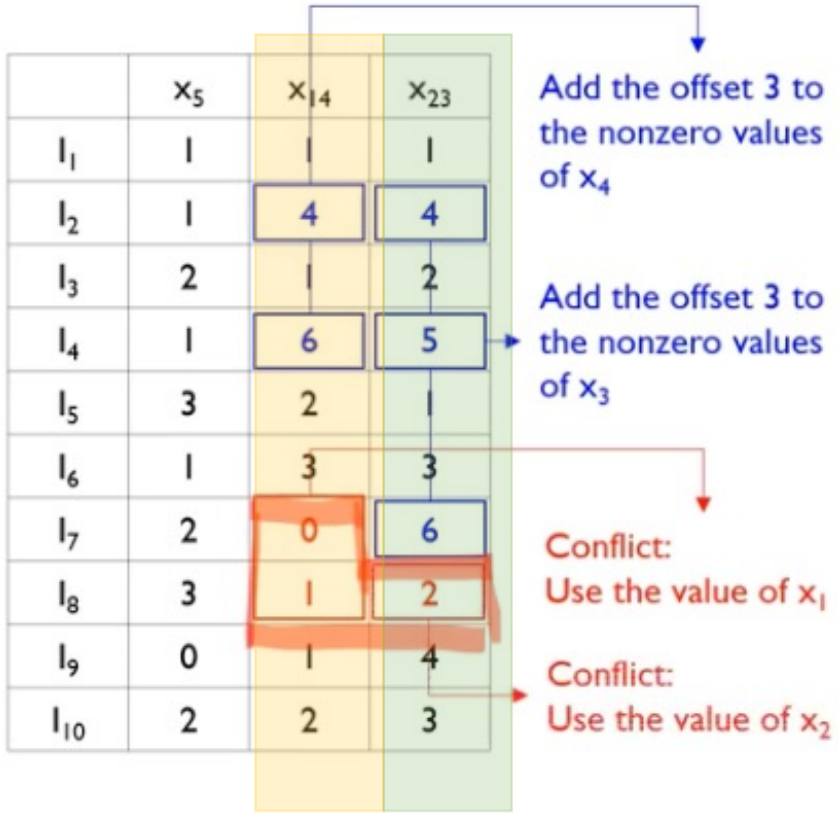
Exclusive Feature Merging

기준 기준

	x_5	x_1	x_4	x_2	x_3
I_1	1	1	0	1	0
I_2	1	0	1	0	1
I_3	2	1	0	2	0
I_4	1	0	3	0	2
I_5	3	2	0	1	0
I_6	1	3	0	3	0
I_7	2	0	0	0	3
I_8	3	1	4	2	3
I_9	0	1	0	0	1
I_{10}	2	2	0	3	0

Max:3 Max:3

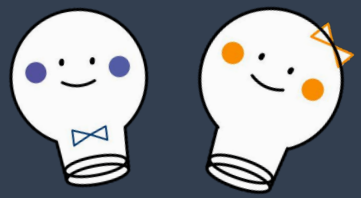
Add offset: 번들링을 하기 위한 대상이 되는 변수에다가,
원래 기준이 되는 변수가 가질 수 있는 최대값 더해 줌



Parameter

num_iterations	Default: 100 반복 수행 트리개수 지정 너무 크면 과적합
learning_rate	학습률
max_depth	최대 깊이 너무 깊어지면 과적합
min_data_in_leaf	최종 리프 노드가 되기 위한 레코드 수 과적합 제어
boosting	Gbdt: 그래디언트 부스팅 rf: 랜덤포레스트

Bagging_fraction	데이터 샘플링 비율 과적합 제어
feature_fraction	학습시 선택되는 피쳐 비율 과적합 제어
lambda_L2	L2 정규화 적용 값
lambda_L1	L1 정규화 적용 값
objective	reg:linear 회귀 Binary: logistic 이진분류 Multi: softmax 다중분류, 클래스 반환 Multi: softprob 다중분류, 확률 반환



우리만 따라와 Follow ADS

이상 ADS 황성아 였습니다. 감사합니다.