D&A

ML Session 7차시

부스팅(Boosting).

2022 / 11 / 08

D&A 학회장 권유진

CONTENTS.

01. Boosting

배깅 vs 부스팅

배깅(Bagging)

같은 알고리즘을 사용해 훈련 세트의 서브셋을 무작위로 구성해 분류기를 각기 다르게 학습

→ 병렬 처리 가능

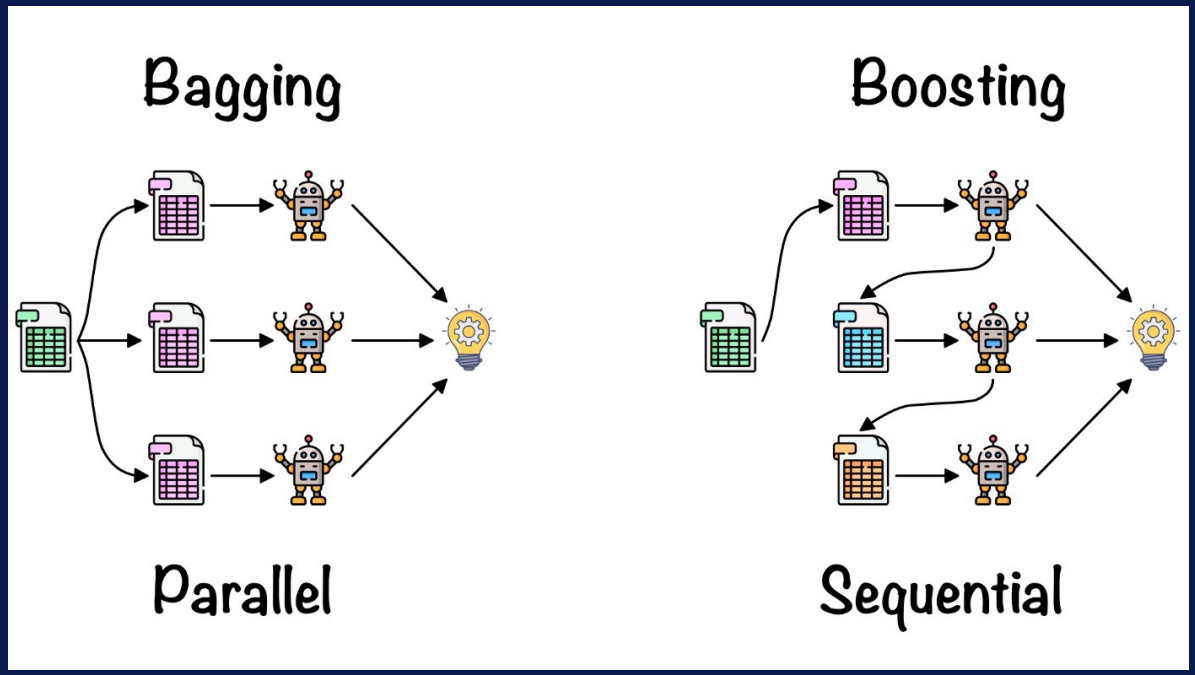
부스팅(Boosting)

약한 학습기 여러 개를 연결해 강한 학습기를 만드는 방법

앞의 모델을 보완해 나가면서 일련의 예측기 학습

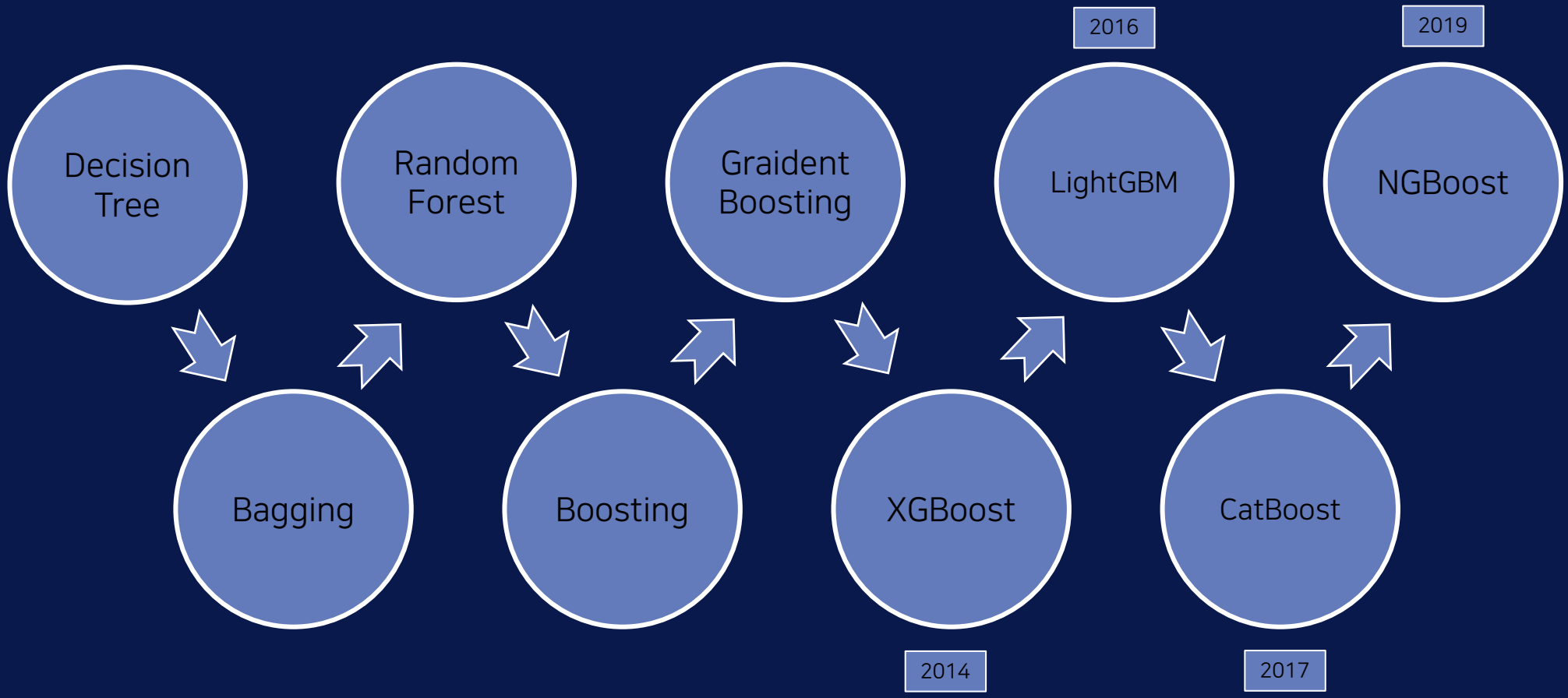
→ 순차적 처리

→ 앞의 모델을 보완해 나가기 때문에 병렬 불가!!!!



01. Boosting

Boosting History

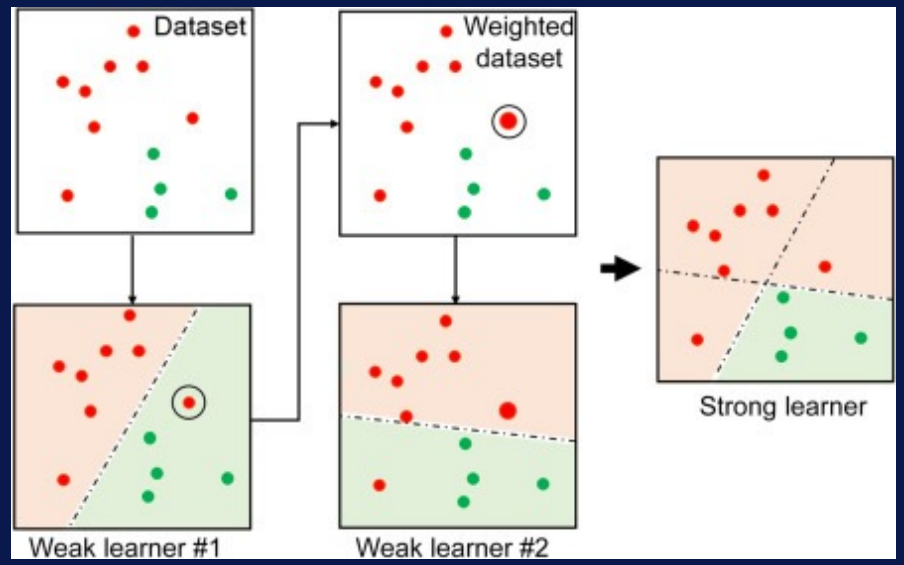


02. AdaBoost

에이다부스트(AdaBoost)

이전 모델이 과소적합된 훈련 샘플의 가중치를 더욱 높이는 방법으로 이전 예측기 보완해 나가는 방법

→ 학습하기 어려운 샘플에 더욱 맞춰짐



학습 과정

1. 첫 번째 분류기를 훈련하고 예측
2. 잘못 분류된 훈련 샘플의 가중치를 상대적으로 높임
3. 두 번째 분류기는 업데이트된 가중치를 사용해 훈련하고 예측 생성
4. 다시 가중치 업데이트
5. 위 과정 반복

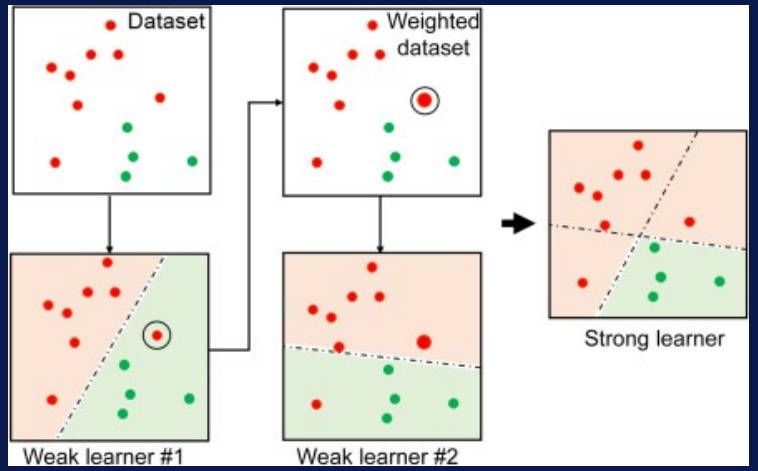
경사 하강법이 손실 함수 최소화를 위해 학습했다면, 에이다부스트는 성능이 점차 더욱 좋아지기 위해 학습

예측 할 때, 학습된 모든 분류기를 활용

→ 하지만 가중치가 적용된 훈련 세트의 전반적인 정확도에 따라 예측기마다 다른 가중치 적용

02. AdaBoost

에이다부스트(AdaBoost)



1. 각 샘플의 가중치 $w^{\{(i)\}}$를 $\frac{1}{m}$로 초기화
   → m: 샘플 수

2. 이전 예측기가 학습 되면 가중치가 적용된 에러율 r_j가 훈련 세트에 대해 계산

$$r_j = \frac{\sum_{i=1, \hat{y}_j^{(i)} \neq y^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$ 에러율

$$\alpha_j = \eta \log \frac{1-r_j}{r_j}$$ 가중치 계산

→ η:학습률(학습되는 정도, 이번 학습을 통한 결과를 얼마나 반영할 지)

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & if, \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} e^{\alpha_j}. & if, \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$ 잘못 예측된 샘플에 가중치 부여

3. 그 후, 모든 샘플의 가중치를 $\sum_{j=1}^{m} w^{(i)}$로 나눔 (정규화)

4. 2~3 과정 반복

5. 모든 예측기의 예측을 계산하고, 예측기 가중치 α_j를 더해 예측 결과 만듦

$$\hat{y}(x) = \operatorname{argmax}_k \sum_{j=1, \hat{y}_j(x)=k}^{N} \alpha_j$$ 가중치 합이 가장 큰 클래스가 예측값이 됨

03. GBM

GBM(Gradient Boosting Machine)

Gradient Boosting = Gradient Descent + Boosting

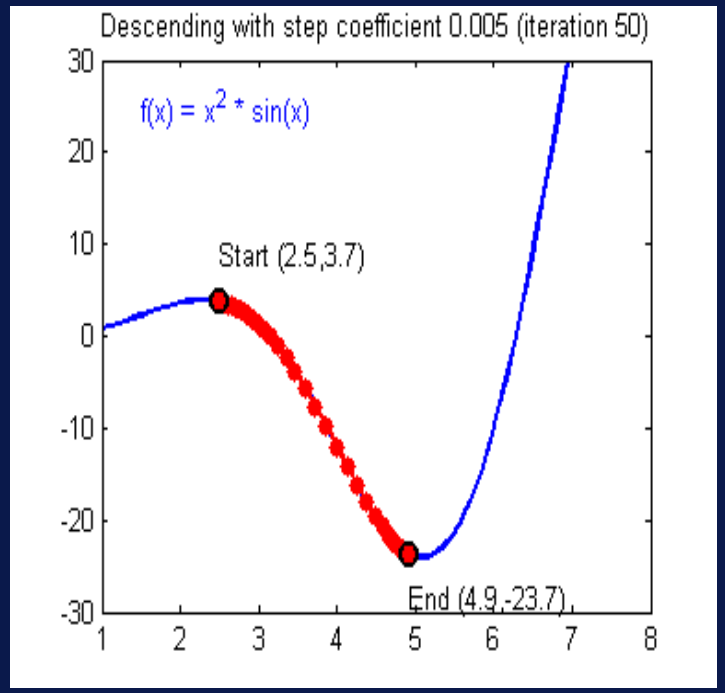
경사하강법(Gradient Descent)

손실 함수의 기울기를 구하고 경사의 반대 방향으로 계속 이동시켜 최소값에 도달할 때 까지 반복

$$\min L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2$$ 손실 함수 계산

$$\frac{\partial L}{\partial f(x_i)} = f(x_i) - y_i$$ 경사(기울기) 계산

$$y_i - f(x_i) = -\frac{\partial L}{\partial f(x_i)}$$ 잔차는 기울기의 음수



03. GBM

GBM(Gradient Boosting Machine)

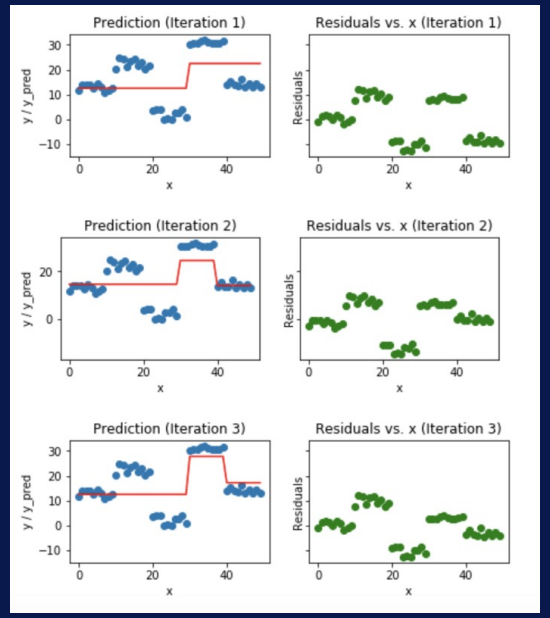
Gradient Boosting = Gradient Descent + Boosting

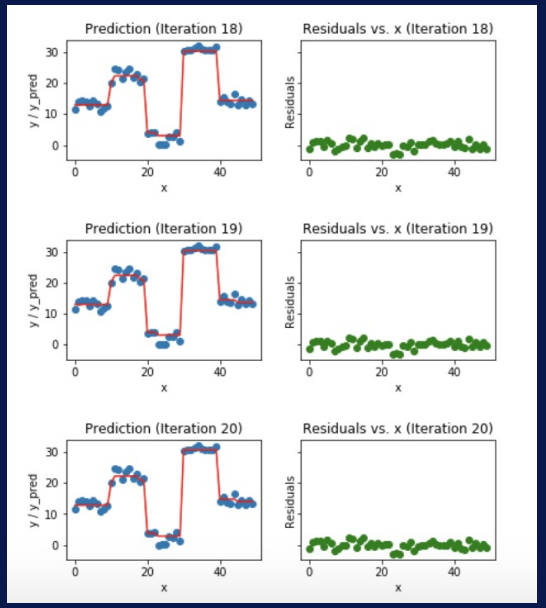
이전 예측기가 만든 잔여 오차를 통해 새로운 예측기를 학습하는 방식으로 이전 예측기를 보완

너무 과도하게 잔차를 학습하면 과대적합

학습 과정

1. 첫 번째 예측기 학습
2. 잔차 계산
3. 잔차를 통해 다음 예측기 학습
4. 2~3 과정 반복





03. GBM

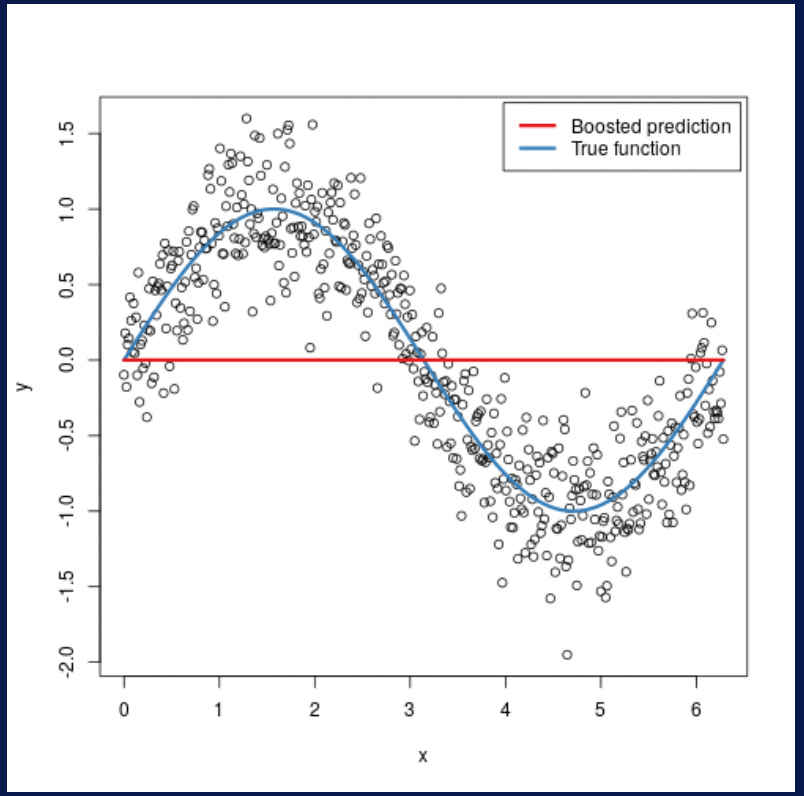
GBM(Gradient Boosting Machine)

$\hat{y} = f_1(x)$ 첫 번째 예측기 학습

$y - \hat{y} = y - f_1(x)$ 잔차 계산

$y - f_1(x) = r_1 = f_2(x)$ 잔차를 통해 두 번째 예측기 학습

$y - f_1(x) - f_2(x) = r_2 = f_3(x)$ 두 번째 잔차를 통해 예측기 학습

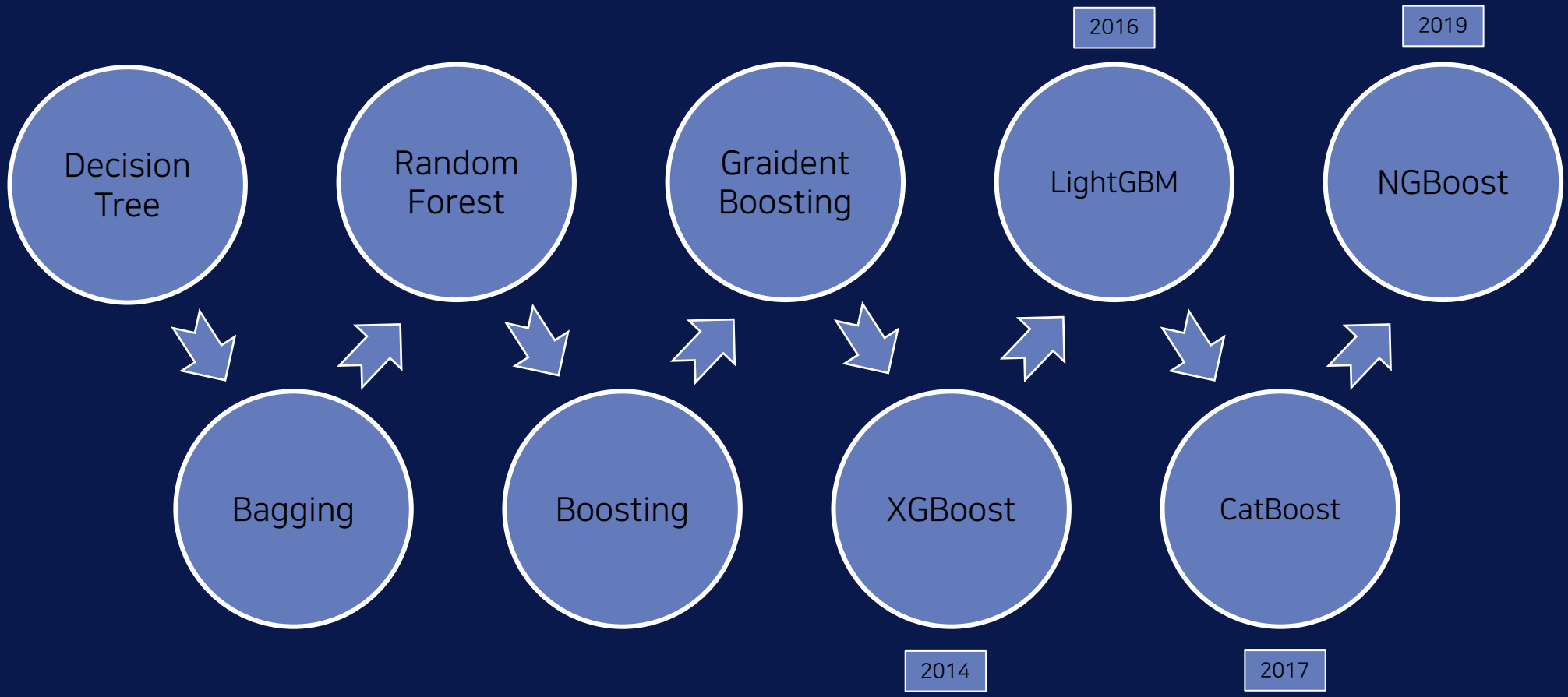


잔차를 학습하기 때문에 예측 시에는 모든 예측기의 결과 값을 합

$$y = f_1(x) + f_2(x) + f_3(x)$$
$$= f_1(x) + f_2(x) + (y - f_1(x) - f_2(x))$$

03. GBM

Boosting History



03. GBM

XGBoost(eXtreme Gradient Boost)

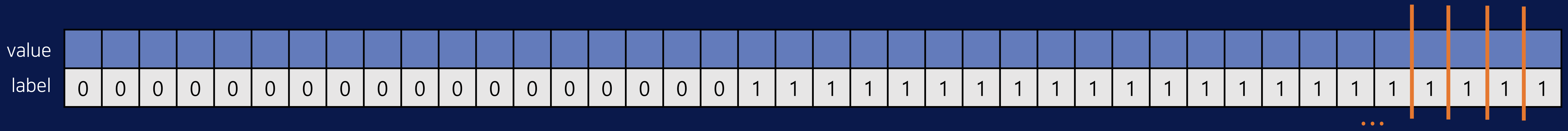
Graident Boosting에 Extreme한 요소들을 얹은 모델(GBM을 발전시킨 모델)
→ 속도 및 성능 향상

개선 사항

Regularized boosting (prevents overfitting)	정규화
Can handle missing value automatically	자동으로 결측값 처리
Parallel processing	병렬 처리
Can cross-validate at each iteration	교차 검증, 조기 종료
→ Enables early stopping, finding optimal number of iterations	
Incremental training	연이어 추가 학습
Can plug in your own optimization objectives	Objective 설정
Tree pruning	가지치기

03. GBM

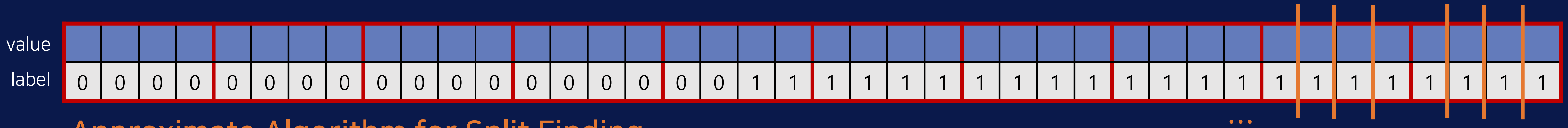
XGBoost(eXtreme Gradient Boost)



총 39회 계산

Extact Greedy Algorithm for Split Finding

기존 트리 알고리즘에서 사용하는 Split Finding 방법
전수 조사를 해 최적의 Split point 탐색
항상 optimal solution 을 찾는다.
→ 전수 조사로 인해 분산 처리 불가
→ 모든 데이터가 메모리에 올라가지 않으면 처리 불가능



총 30회 계산
10×3 (버켓 수×탐색 수)

Approximate Algorithm for Split Finding

XGBoost에서 사용하는 Split Finding 방법
데이터를 정렬한 후, 버켓을 기준으로 최적의 split point 탐색
→ 버켓 마다 split point를 탐색하므로 병렬 처리 가능!!!

03. GBM

XGBoost(eXtreme Gradient Boost)

종류	파라미터명	default값	설명
부스터 파라미터	learning_rate	0.1	학습률 (범위: 0~1)
	n_estimators	100	생성할 약한 학습기 개수
	max_depth	3	트리의 최대 깊이 (보통 3~10 적용)
	min_child_weight	1	트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 가중치 총합 과적합 조절 용도(범위: 0~inf)
	gamma	0	트리의 리프 노드를 추가적으로 나눌지 결정할 최소 손실 감소값 해당 값보다 큰 손실이 감소된 경우에 리프 노드 분리 값이 클수록 과적합 방지
	early_stopping_rounds	None	조기 중단을 위한 반복 횟수 n번 반복하는 동안 성능 평가 지표가 향상되지 않으면 반복이 멈춤
	subsample	1	트리가 커져 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정 보통 0.5~1 사이의 값 적용 (범위: 0~1)
	colsample_bytree	1	트리 생성에 필요한 피쳐를 임의로 샘플링
	res_lambda	1	L2 Regularization 적용 값
	reg_alpha	0	L1 Regularization 적용 값
	scale_pos_weight	1	특정값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위한 파라미터
학습 테스크 파라미터	objective	회귀:linear	손실함수 (Binary: logistic, Multi: softmax, Multi: softprob)
	eval_metric	회귀:rmse 분류:error	검증에 사용하는 함수 정의 (rmse,mae, logloss, merror, mlogloss, auc)
	eval_set	None	성능 평가에 사용하는 데이터셋

03. GBM

LGBM

속도 느림
오버피팅 가능성

속도 보통
오버피팅 규제

속도 빠름
데이터셋이 적으면 오버피팅

03. GBM

LGBM

기존 트리 모델들은 모든 feature와 data를 모두 scan해 information gain을 측정해야 했다.
→ 속도가 오래 걸림
→ 해당 문제 해결을 위해 LGBM은 GOSS, EFB 적용

GOSS(Gradient-based One-Side Sampling)

→ 모든 데이터는 경사의 크기가 다르다
→ 경사가 큰 데이터 유지, 작은 데이터 무작위로 드랍
→ 경사가 큰 데이터는 잘 학습되지 못한, 경사가 작은 데이터는 잘 학습된 데이터이다.
→ 잘 학습되지 못한 경사가 큰 데이터 위주로 학습

random sampling

RowID	1	5	6	2	3	4
경사	0	0.1	0.2	0.5	3	-5

RowID	1	6	3	4
경사	0	0.2	3	-5

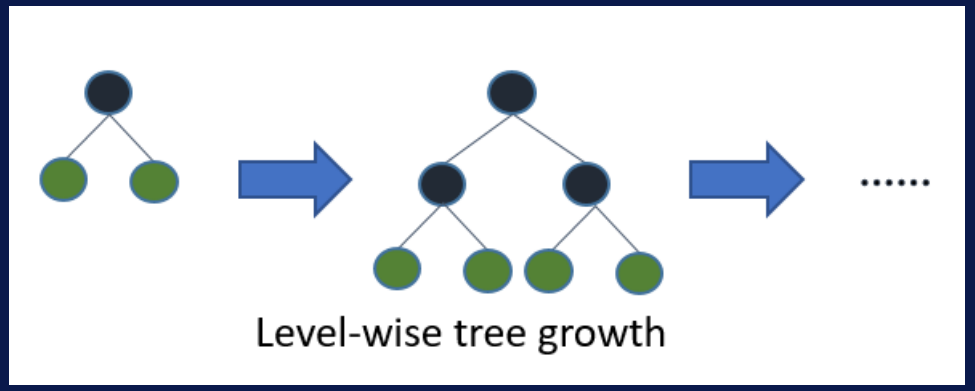
EFB(Exclusive Feature Bundling)

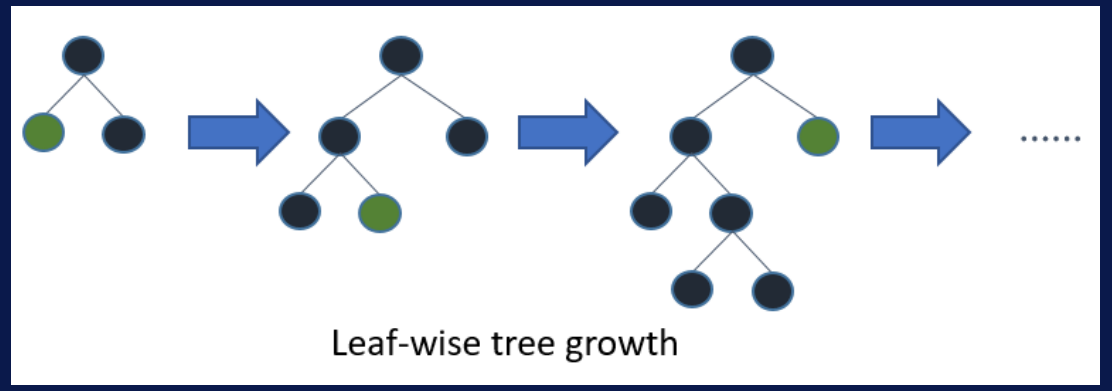
→ 상호 배제적인 feature를 bundling(모음)
→ feature가 서로 exclusive 하지 않을 경우의 risk 감수하고 시행

$$[x_1, x_2, x_3, x_4, x_5]$$
$$\downarrow$$
$$[x_{1,5}, x_{2,3}, x_4]$$

03. GBM

LGBM





Level-wise tree: 균형 트리 분할

사용 모델: RF, XGB

성장방법: 균형을 맞춤, 연산 추가, 수평 성장

Leaf-wise tree: 리프 중심 트리 분할

사용 모델: LGBM

성장 방법: 균형을 맞추지 않음, 손실 값이 큰 리프 노드에서 분할, 수직 성장

03. GBM

LGBM

- 속도가 빠르고, 성능이 좋다.

  → GOSS, EFB

- 저장 공간을 덜 차지한다.
- 병렬적인 학습(동시적)이 가능하다.
- Overfitting에 민감하다.

  → 대용량 데이터셋에 적합

03. GBM

LGBM

종류	파라미터명	default값	설명
부스터 파라미터	learning_rate	0.1	학습률 (범위: 0~1)
	n_estimators	100	생성할 약한 학습기 개수
	max_depth	3	트리의 최대 깊이 (보통 3~10 적용)
	min_child_weight	1	트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 가중치 총합 과적합 조절 용도(범위: 0~inf)
	num_leaves	31	하나의 트리가 가질 수 있는 최대 리프 개수
	subsample	1	트리가 커져 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정 보통 0.5~1 사이의 값 적용 (범위: 0~1)
	colsample_bytree	1	트리 생성에 필요한 피쳐를 임의로 샘플링
	res_lambda	1	L2 Regularization 적용 값
	reg_alpha	0	L1 Regularization 적용 값
학습 테스크 파라미터	objective	회귀:linear	손실함수 (Binary: logistic, Multi: softmax, Multi: softprob)
	eval_metric	회귀:rmse 분류:error	검증에 사용하는 함수 정의 (rmse,mae, logloss, merror, mlogloss, auc)
	eval_set	None	성능 평가에 사용하는 데이터셋

03. GBM

CatBoost

범주형 feature를 처리하는데 중점을 둔 알고리즘
→ feature에 범주형 변수가 많을 경우 효율적
→ 반대로 수치형 feature가 대부분일 경우 비효율적으로 작동

기존 GBM기반 알고리즘이 갖고 있는 Target Leakage, Prediction Shift 문제 해결
→ Ordering Principle을 제안하여 해결

Target Leakage

예측 시점에 사용할 수 없는 정보가 데이터셋에 포함된 상황 ex) 예측값 y가 데이터셋 X에 포함
GBM에서는 모델 학습 중 잔차를 데이터셋으로 활용하므로 Target Leakage 발생

$$r_i = y_i - \hat{y}_i$$

Prediction Shift

학습 데이터셋에서는 y를 알기 때문에 y를 활용해 학습
평가 데이터셋에서는 학습 데이터셋의 y를 활용한 값을 통해 계산
→ 하지만 평가 데이터셋과 학습 데이터셋의 분포는 다르므로 문제가 됨.

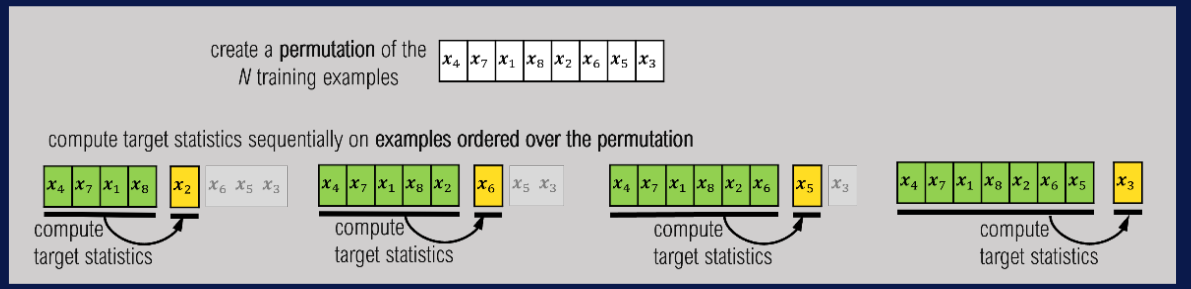
03. GBM

CatBoost

Ordered Target Statistics

범주형 변수의 값을 계산할 때 이전 데이터 까지만 활용하여 계산 과정

1. Boosting 단계마다 무작위 순열 활용해 정렬
2. $i+1$번째 값을 계산할 때는 i번째 까지의 값을 기준으로 계산

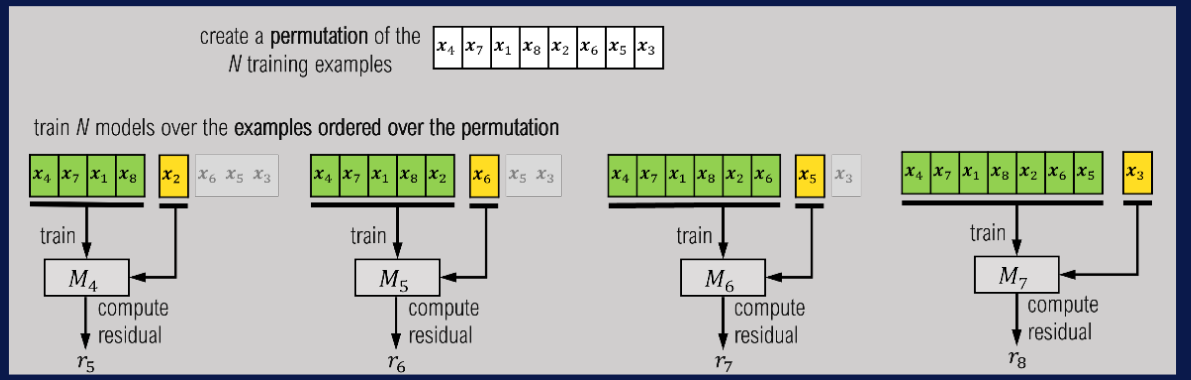


Ordered Boosting

잔차를 구하는 과정에서 이전 데이터 까지만 활용하여 계산

과정

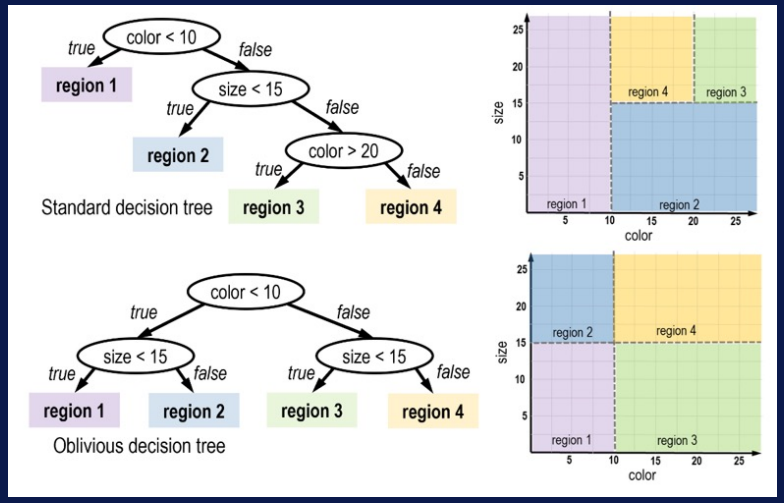
1. Boosting 단계마다 무작위 순열 활용해 정렬
2. $i+1$번째 샘플의 잔차를 계산할 때 i번째 까지 사용한 데이터로 학습



Oblivious Trees

같은 레벨에서는 동일한 분할 기준 사용

→ 계산량 감소



04. Boosting 정리

전체 Boosting 정리

모델	AdaBoost	GBM	XGBoost	LGBM	CatBoost
부스팅 방식	데이터 가중치	잔차	잔차	잔차	잔차
속도	느림	느림	보통	빠름	빠름
병렬 처리	불가	불가	가능	가능	가능
Tree	Level-wise + Asymmetric	Level-wise + Asymmetric	Level-wise + Asymmetric	Leaf-wise + Asymmetric	Level-wise + Oblivious
특징				대용량 데이터셋에 적합	범주형 변수에 특화

첨부자료 출처

01. Boosting

Bagging vs Boosting_이미지출처: https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422

02. AdaBoost

AdaBoost_이미지출처: https://www.sciencedirect.com/topics/engineering/adaboost

03. GBM

경사하강법_gif출처: https://angeloyeo.github.io/2020/08/16/gradient_descent.html
GBM_이미지출처: http://uc-r.github.io/gbm_regression
GBM_gif출처: https://blog.mlreview.com/gradient-boosting-from-scratch-1e317ae4587d
LGBM_이미지출처: https://github.com/Microsoft/LightGBM/blob/master/docs/Features.rst#references
CatBoost_이미지출처: https://livebook.manning.com/book/ensemble-methods-for-machine-learning/chapter-8/v-5/171

폰트

네이버 글꼴 모음 _ 나눔 스퀘어 사용
출처 : https://hangeul.naver.com/font

D&A

ML Session 7차시 부스팅(Boosting)

Thank You.

2022 / 11 / 08

D&A 학회장 권유진