



# CatBoost: unbiased boosting with categorical features

고급심화 차수빈

# 1. Catboost Overview

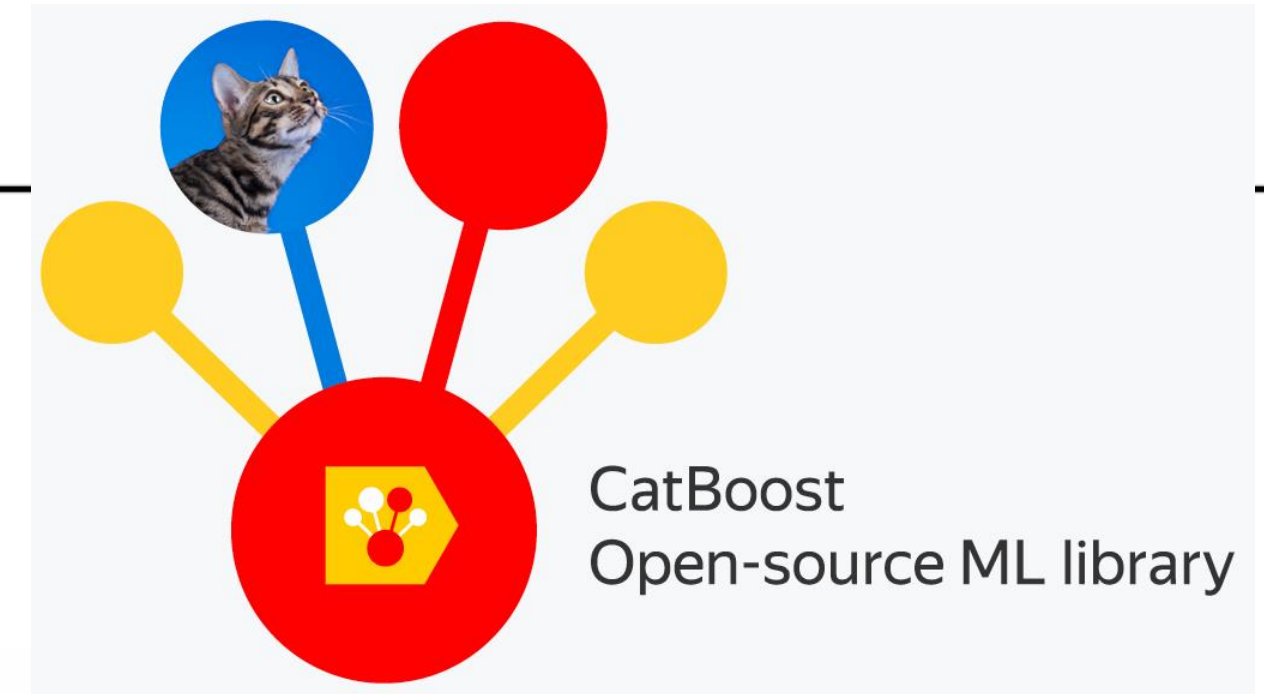
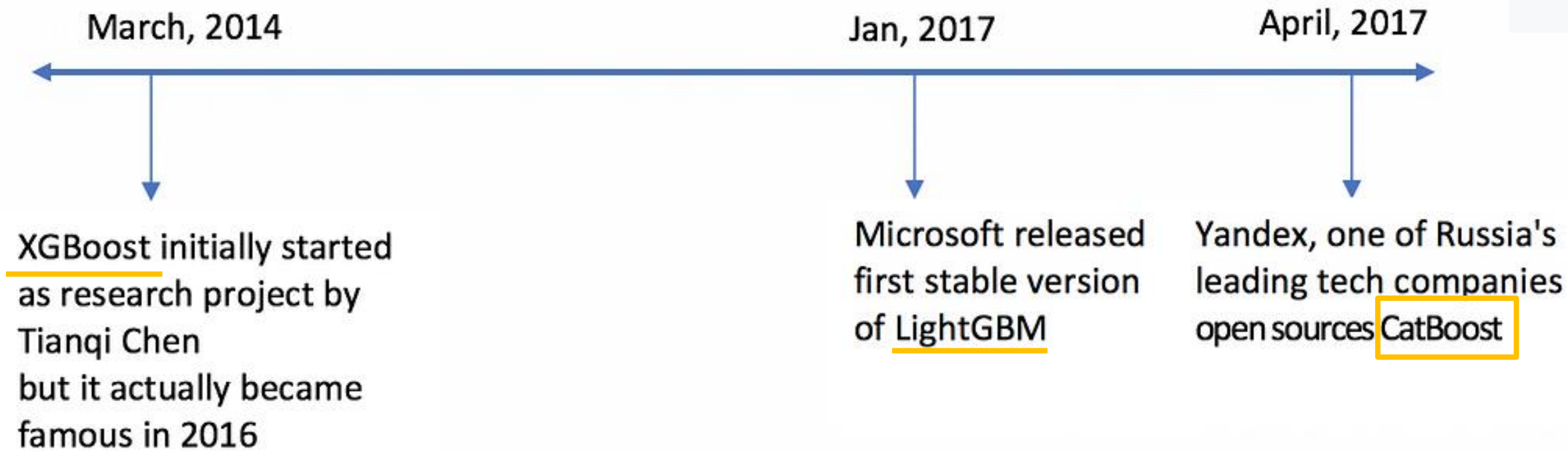


# #1. CatBoost Overview

- Catboost?

꽁꽁 얼어붙은 강의실 위를 고양이가 걸어갑니다 🐈

- \* History – Gradient Boosting



- 주어진 functional space에서 경사 하강법을 수행하며 앙상블 예측기를 구성해 나가는 알고리즘
- 약한 모델(= base predictors)을 탐욕적으로 반복 결합함으로써 강한 모델을 만들어 나가는 방식
- 다양한 작업에서 SOTA를 달성하였음

# #1. CatBoost Overview

그러나, Gradient Boosting의 기존 구현은 두 가지 통계적 문제에 직면함

1. Target Leakage

2. Prediction Shift

**범주형(Categorical) 데이터에 특화된 부스팅 알고리즘을 고안해 보자!**

CatBoost = **C**ategorical **B**oosting

## 2. Background



# #2. Background

- GradientBoost의 작동 원리
  - Dataset에 대한 가정
    - $D = (x_k, y_k)_{k=1 \dots n}$ : data
    - $x_k = (x_k^1, \dots, x_k^m)$ : r.v of m features
    - $y_k \in \mathbb{R}$ : target, binary(분류) / numerical(회귀)
    - $(x_k, y_k) \sim P(\cdot, \cdot)$  (P: unknown distribution)
  - 목표: loss  $L(F) := E L(y, F(x))$ 를 최소화하는 함수  $F: \mathbb{R}^m \rightarrow \mathbb{R}$ 을 훈련시키는 것
  - Gradient Boosting은 greedy한 방법으로 일련의 근사치  $F^t: \mathbb{R}^m \rightarrow \mathbb{R}, t = 0, 1, \dots$ 를 반복적으로 구축해 나가는 알고리즘
    - $F^t = F^{t-1} + \alpha h^t$  ( $\alpha$ : step size,  $h^t: \mathbb{R}^m \rightarrow \mathbb{R}$ , 손실을 최소화하기 위한 함수)
  - Catboost는 기본적으로 GradientBoosting 방식을 채택
    - 기본 예측기(base predictor)로 binary decision tree를 활용

# 3. Categorical Features



# #3. Categorical Feature

- Related Work

- 범주형 변수는 비교할 수 없는 범주라 불리는 이산형(discrete) 값의 집합을 가지는 변수들임
  - 이러한 특성을 처리하는 가장 기본적인 접근법이 One-hot Encoding임
  - 그러나 one-hot Encoding은 범주의 개수만큼 새로운 변수를 생성해 냄
    - high cardinality 특성("user ID" feature 등)의 경우 너무 많은 새로운 feature의 수를 초래

id	color
1	red
2	blue
3	green
4	blue



id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0



# #3. Categorcal Feature

- Related Work

- 대안으로, 범주를 제한된 클러스터로 그룹화하고 목표 통계량(Target Statistics, TS)을 사용하여 수치적 특성으로 각 범주의 예상 목표 값을 추정하는 방법이 제안되었음
  - 각 카테고리의 기대되는 타겟 값으로 클러스터링하여 각 카테고리를 새로운 수치 특성으로 변환
  - 정보 손실을 최소화하면서도 효율적인 처리를 가능하게 함

$k$ 번째 훈련 예제의  $i$ 번째 feature의 범주 값  $x_k^i$ 를  
목표 통계량(TS)  $\hat{x}_k^i$ 라는 하나의 **수치형** feature로 대체하자.

- 다음과 같이 범주에 의해 조건화 되는 예상 target  $y$ 를 추정

$$\hat{x}_k^i \approx E(y|x_i = x_k^i)$$

# #3. Categorcal Feature

- Target Statistics

## 1. Greedy TS

- 동일한 범주  $x_k^i$ 를 가지고 훈련 예제들에 대한  $y$ 의 평균값으로 추정하는 방법

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	A	...	0
$l_5$	...	B	...	1
$l_6$	...	C	...	1
$l_7$	...	B	...	0
$l_8$	...	C	...	1
$l_9$	...	C	...	1
$l_{10}$	...	C	...	0

	...	$x^i(TS)$	...	$y$
$l_1$	...	0.50	...	1
$l_2$	...	0.67	...	1
$l_3$	...	0.80	...	1
$l_4$	...	0.50	...	0
$l_5$	...	0.67	...	1
$l_6$	...	0.80	...	1
$l_7$	...	0.67	...	0
$l_8$	...	0.80	...	1
$l_9$	...	0.80	...	1
$l_{10}$	...	0.80	...	0

# #3. Categorcal Feature

- Target Statistics

- 1. Greedy TS

- smoothing을 적용하지 않는 경우 noisy category 문제가 발생할 수 있음

	y=1	y=0	TS
A	10	10	0.5
B	40	10	0.8
C	10	40	0.2
D	25	25	0.5
E	1	0	1

← 매우 희소하게 나오는 값의 TS가 매우 커지는 문제

# #3. Categorical Features

- Target Statistics

1. Greedy TS

- Greedy TS With Smoothing

$$\hat{x}_k^i = \frac{\sum_{j=1}^n \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{j=1}^n \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

- $\sum_{j=1}^n \mathbb{1}_{\{x_j^i = x_k^i\}}$ :  $k$ 번째 feature의 category 값이  $i$ 인 객체의 총 수
- $\sum_{j=1}^n \mathbb{1}_{\{x_j^i = x_k^i\}} * y_j$ :  $k$ 번째 feature의 category 값이  $i$ 인 객체 중  $y$ (target) 값이 1인 객체의 수
- $a(>0)$ 와  $p$ 는 hyper-parameter
- $p$ 는 주로 target의 평균값을 취함

=> 자주 등장하지 않는 noisy category의 부정적 영향을 줄여주는 역할

# #3. Categorical Features

- Greedy TS with smoothing example
  - ✓  $a = 0.1$  (parameter) ,  $p = 0.7$  (computed from the training dataset)
  - ✓ For category A

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	A	...	0
$l_5$	...	B	...	1
$l_6$	...	C	...	1
$l_7$	...	B	...	0
$l_8$	...	C	...	1
$l_9$	...	C	...	0
$l_{10}$	...	C	...	1

$$\hat{x}_k^A = \frac{\sum_{j=1}^n \mathbf{1}_{\{x_j^A = x_k^A\}} \cdot y_j + ap}{\sum_{j=1}^n \mathbf{1}_{\{x_j^A = x_k^A\}} + a}$$

$$= \frac{1 + 0.1 \times 0.7}{2 + 0.1} = 0.5095$$

# #3. Categorical Features

- Greedy TS with smoothing example

- ✓  $a = 0.1$  (parameter) ,  $p = 0.7$  (computed from the training dataset)

- ✓ For category B

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	A	...	0
$l_5$	...	B	...	1
$l_6$	...	C	...	1
$l_7$	...	B	...	0
$l_8$	...	C	...	1
$l_9$	...	C	...	0
$l_{10}$	...	C	...	1

$$\hat{x}_k^B = \frac{\sum_{j=1}^n \mathbb{1}_{\{x_j^B = x_k^B\}} \cdot y_j + ap}{\sum_{j=1}^n \mathbb{1}_{\{x_j^B = x_k^B\}} + a}$$

$$= \frac{2 + 0.1 \times 0.7}{3 + 0.1} = 0.6677$$

# #3. Categorical Features

- Greedy TS with smoothing example

- ✓  $a = 0.1$  (parameter) ,  $p = 0.7$  (computed from the training dataset)

- ✓ For category C

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	A	...	0
$l_5$	...	B	...	1
$l_6$	...	C	...	1
$l_7$	...	B	...	0
$l_8$	...	C	...	1
$l_9$	...	C	...	0
$l_{10}$	...	C	...	1

$$\hat{x}_k^C = \frac{\sum_{j=1}^n \mathbb{1}_{\{x_j^C = x_k^C\}} \cdot y_j + ap}{\sum_{j=1}^n \mathbb{1}_{\{x_j^C = x_k^C\}} + a}$$

$$= \frac{4 + 0.1 \times 0.7}{5 + 0.1} = 0.7980$$

# #3. Categorical Features

- Target Statistics

1. Greedy TS

- 문제점: **Target Leakage** 발생
  - feature  $\hat{x}_k^i$ 의 값을 계산하는 데 target인  $y_k$ 값이 사용되는 현상
  - 이는 **conditional shift**를 유발
- **Conditional Shift**
  - 학습용(train) 데이터와 테스트용(test) 데이터에서의  $\hat{x}^i|y$ 의 분포가 달라지는 문제
  - $E(\hat{x}^i|y) \neq E(\hat{x}_k^i|y_k)$



# #3. Categorical Features

- Target Statistics

- 1. Greedy TS

- Conditional Shift Example(in the paper)
  - i번째 feature는 범주형이고, 모든 category가 독립적이라고 가정

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	D	...	0
$l_5$	...	E	...	0
$l_6$	...	F	...	0



	...	$x^i$	...	$y$
$l_1$	...	$\frac{1 + ap}{1 + a}$	...	1
$l_2$	...	$\frac{1 + ap}{1 + a}$	...	1
$l_3$	...	$\frac{1 + ap}{1 + a}$	...	1
$l_4$	...	$\frac{0 + ap}{1 + a}$	...	0
$l_5$	...	$\frac{0 + ap}{1 + a}$	...	0
$l_6$	...	$\frac{0 + ap}{1 + a}$	...	0

- 이러한 경우, 분할 경계(결정 경계)를 다음과 같이 설정 가능:  $x^i = \frac{0.5 + ap}{1 + a}$

# #3. Categorical Features

- Target Statistics

- 1. Greedy TS


- Conditional Shift Example(in the paper)

Training Set

	...	$x^i$	...	$y$
$l_1$	...	A	...	1
$l_2$	...	B	...	1
$l_3$	...	C	...	1
$l_4$	...	D	...	0
$l_5$	...	E	...	0
$l_6$	...	F	...	0

Test Set

$l_7$	...	G	...	1
$l_8$	...	H	...	1
$l_9$	...	I	...	0
$l_{10}$	...	J	...	0



	...	$x^i$	...	$y$
$l_1$	...	$\frac{1+ap}{1+a}$	...	1
$l_2$	...	$\frac{1+ap}{1+a}$	...	1
$l_3$	...	$\frac{1+ap}{1+a}$	...	1
$l_4$	...	$\frac{0+ap}{1+a}$	...	0
$l_5$	...	$\frac{0+ap}{1+a}$	...	0
$l_6$	...	$\frac{0+ap}{1+a}$	...	0

$l_7$	...	p	...	1
$l_8$	...	p	...	1
$l_9$	...	p	...	0
$l_{10}$	...	p	...	0

- 테스트 데이터에서의 TS 값이 모두  $p$ 가 되어버림 → 결정 경계로서의 의미가 상실됨

# #3. Categorical Features

- Conditional shift
  - 이러한 문제점에 대해, 논문의 저자들은 TS가 가져야 할 속성으로 두 가지를 제시
- Property 1
  - › target 값이 동일하다면 학습 데이터와 테스트 데이터에서의 TS의 기댓값(expectation)은 동일해야 한다.
  - ›  $E(\hat{x}^i | y = v) = E(\hat{x}_k^i | y_k = v)$ ,  $(x_k, y_k)$ :  $k$ th training example
- Property 2
  - › 모델 학습 시에는 training data를 최대한 활용할 수 있는 방향으로 진행해야 한다.
  - › 최대한 모든 정보 활용

# #3. Categorical Features

- Conditional shift 문제를 어떻게 해결해야 할까?
  - $x_k$ 의 TS 계산 시  $y_k$ 가 사용되는 것이 문제이니  $x_k$ 만 제외하고 계산해 보자!
    - ↳  $D_k \subset D \setminus \{x_k\}$

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in D_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in D_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

## 2. Holdout TS

- 학습용 데이터를 두 부분으로 분리하자:  $D = \widehat{D}_0 \cup \widehat{D}_1$ 
  - ↳ TS 계산 시에는  $D_0$  활용
  - ↳ 학습 시에는  $D_1$  활용

⇒ Property 2 위반

## 3. Leave-one-out TS

- 학습:  $D_k \subset D \setminus \{x_k\}$  활용
- 평가:  $D_k = D$  활용(학습 데이터 전체)

⇒ 여전히 target leakage 문제는 발생

# #3. Categorical Features

- Conditional shift 문제를 어떻게 해결해야 할까?

## 4. Ordered TS

- └ Catboost에서 Target Leakage 문제를 해결하기 위해 제안하는 방식
- 임의로 시간 개념을 도입해 보자.  $\Rightarrow$  **artificial time**



- 학습 예제에 대해 random permutation 수행
  - 각각의 예제에 대해 접근 가능한 모든 "history" (이전 정보) 활용
- $D_k = \{x_j : \sigma(j) < \sigma(k)\}$
- high variance 문제를 피하기 위해 permutation을 여러 번 수행

# #3. Categorical Features

- Ordered TS - Example

출처: 고려대학교 DSBA 연구실 - 강필성 교수님 강의자료

- a random permutation  $\sigma$  of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

- $a = 0.1$  (parameter) ,  $p = 0.667$  (computed from the training dataset)

	...	$x^i$	...	TS	$y$
$I_1$	...	A	...	0.000	1
$I_2$	...	B	...	1.000	1
$I_3$	...	C	...	1.000	1
$I_4$	...	A	...	1.000	0
$I_5$	...	B	...	0.977	1
$I_6$	...	C	...	0.982	1
$I_7$	...	B	...	0.992	0
$I_8$	...	C	...	0.986	1
$I_9$	...	C	...	0.992	0
$I_{10}$	...	C	...	0.748	1

$$\begin{aligned}\hat{x}_k^i &= \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a} \\ &= \frac{3 + 0.1 \times 0.667}{4 + 0.1} = 0.748\end{aligned}$$

## 4. Prediction Shift



# #4. Prediction Shift

- Gradient Boosting Procedure

$$h^t = \arg \min_{h \in H} \mathbb{E} \left( -g^t(\mathbf{x}, y) - h(\mathbf{x}) \right)^2 \quad \longrightarrow \quad h^t = \arg \min_{h \in H} \boxed{\frac{1}{n} \sum_{k=1}^n} \left( -g^t(\mathbf{x}, y) - h(\mathbf{x}) \right)^2$$

- 보통 기댓값은 알려져 있기 않기때문에, 기존에 가지고 있는 (유한한) 데이터셋  $D$ 를 활용하여 평균 loss를 계산
- 이때, 학습 데이터인  $x_k$ 가 주어졌을 때의 gradient의 조건부 분포인  $g^t(x_k, y_k)|x_k$ 와 일반적인 테스트 데이터인  $x$ 가 주어졌을 때의 gradient의 조건부 분포인  $g^t(x, y)|x$ 가 다름( $\rightarrow$  shift)
  - $\Rightarrow$  base predictor인  $h^t$ 는 원래 solution과 달라지게 됨( $\rightarrow$  biased)
  - $\Rightarrow$  학습된 모델  $F^t (= F^{t-1} + \alpha * h^t)$ 의 일반화 성능에 영향

Prediction Shift 발생!



# #4. Prediction Shift

- Analysis of Prediction Shift

- prediction shift는 근본적으로 target leakage로부터 발생
- 이를 방지하기 위해 Catboost는 트리의 매 학습 step마다 다른 데이터셋(independent samples)을 활용해야 한다고 주장

**Theorem 1** 1. If two *independent* samples  $\mathcal{D}_1$  and  $\mathcal{D}_2$  of size  $n$  are used to estimate  $h^1$  and  $h^2$ , respectively, using Equation (6), then  $\mathbb{E}_{\mathcal{D}_1, \mathcal{D}_2} F^2(\mathbf{x}) = f^*(\mathbf{x}) + O(1/2^n)$  for any  $\mathbf{x} \in \{0, 1\}^2$ .

2. If the *same* dataset  $\mathcal{D} = \mathcal{D}_1 = \mathcal{D}_2$  is used in Equation (6) for both  $h^1$  and  $h^2$ , then  $\mathbb{E}_{\mathcal{D}} F^2(\mathbf{x}) = f^*(\mathbf{x}) - \underbrace{\frac{1}{n-1} c_2 \left( x^2 - \frac{1}{2} \right)}_{\text{bias}} + O(1/2^n)$ .

$$h^t = \arg \min_{h \in H} \frac{1}{n} \sum_{k=1}^n \left( -g^t(\mathbf{x}, y) - h(\mathbf{x}) \right)^2$$

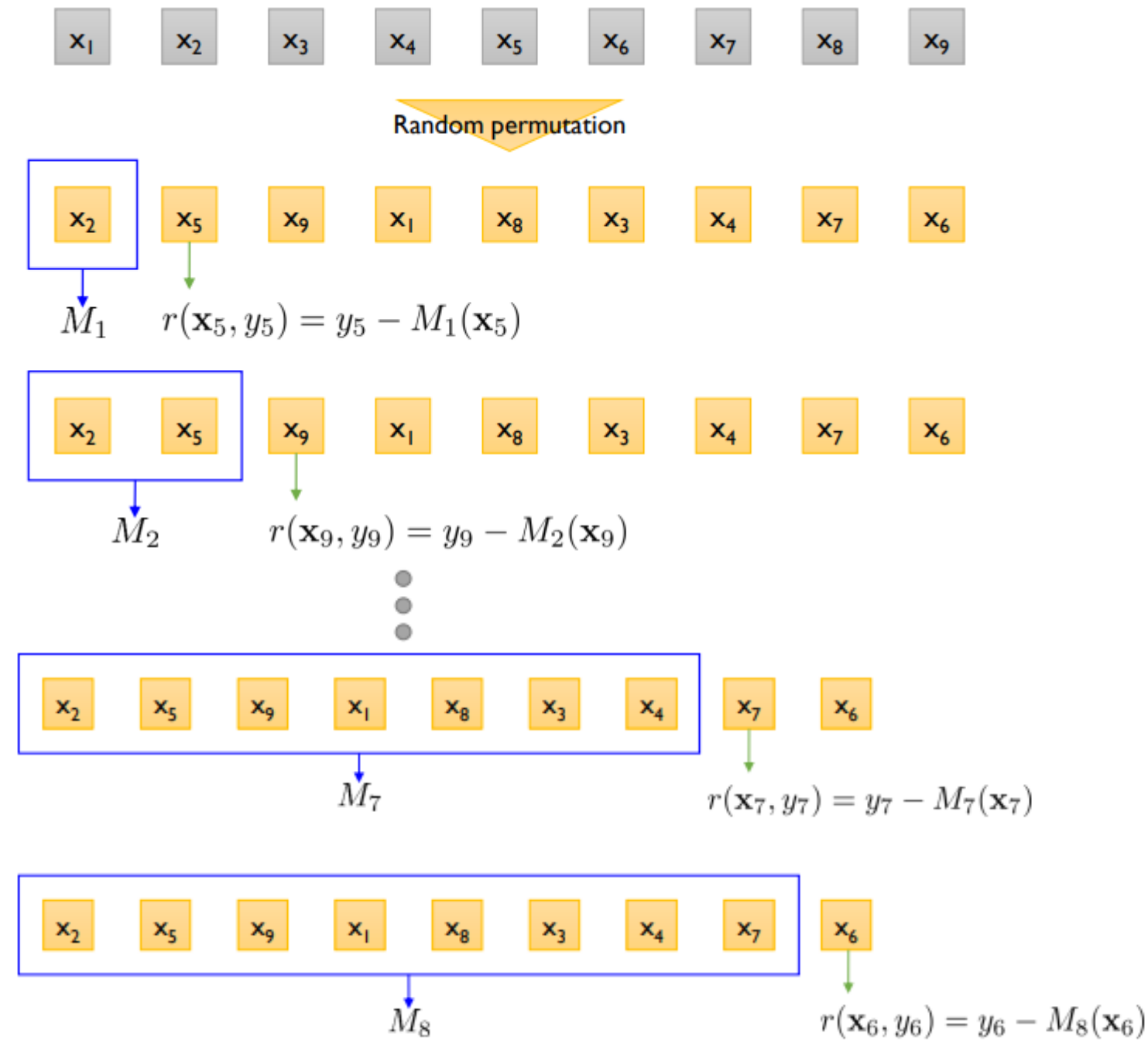
⇒ 독립된 데이터셋을 활용할 때에 비해 같은 데이터셋을 사용하는 경우 추가적인 bias가 발생하게 된다.

## 5. Ordered Boosting



# #5. Ordered Boosting

- Overview



출처: 고려대학교 DSBA 연구실

⇒ 모델(트리)를 계속 생성해 가며 다음 데이터의 잔차를 계산해 나가는 방식

# #5. Ordered Boosting

- Overview

- 임의의 permutation( $\sigma$ )을 만들고, 순차적으로 잔차를 계산하여 트리를 학습하면서 target leakage를 방지하는 방식
  - 이때 TS 계산에 사용되는 permutation( $\sigma_{cat}$ )과 ordered boosting 시 사용되는 permutation( $\sigma_{boost}$ )은 동일하게 설정
    - ⇒ for prediction shift 방지
- 그러나 이러한 방식은 데이터 개수만큼의 서로 다른 학습모델을 필요로 하고, 이는 복잡도와 메모리 요구량을 상승시킴
  - ⇒ 이를 방지하기 위해 그래디언트 부스팅 알고리즘을 일부 수정

# #5. Ordered Boosting

- Practical Implementation

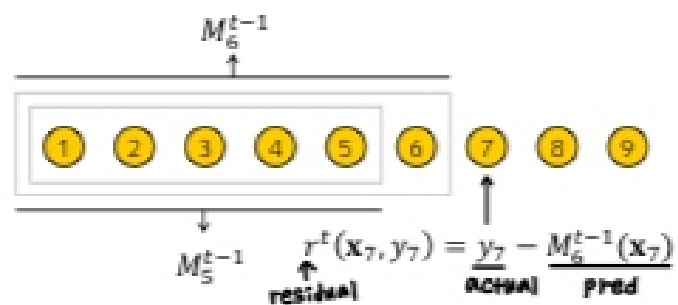


Figure 1: Ordered boosting principle, examples are ordered according to  $\sigma$ .

---

## Algorithm 1: Ordered boosting

---

**input** :  $\{(\mathbf{x}_k, y_k)\}_{k=1}^n, I$ ;  
 $\sigma \leftarrow$  random permutation of  $[1, n]$ ;  
 $M_i \leftarrow 0$  for  $i = 1..n$ ;  
**for**  $t \leftarrow 1$  **to**  $I$  **do**  
    **for**  $i \leftarrow 1$  **to**  $n$  **do**  
         $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$ ;  
    **for**  $i \leftarrow 1$  **to**  $n$  **do**  
         $\Delta M \leftarrow$   
             $\text{LearnModel}((\mathbf{x}_j, r_j) :$   
                 $\sigma(j) \leq i)$ ;  
         $M_i \leftarrow M_i + \Delta M$ ;  
**return**  $M_n$

---



---

## Algorithm 2: Building a tree in CatBoost

---

**input** :  $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, Mode$

- ①  $grad \leftarrow \text{CalcGradient}(L, M, y)$ ;
- ②  $r \leftarrow \text{random}(1, s)$ ;  
        **if**  $Mode = Plain$  **then**  
             $G \leftarrow (grad_r(i) \text{ for } i = 1..n)$ ;
- ③ **if**  $Mode = Ordered$  **then**  
             $G \leftarrow (grad_{r, \sigma_r(i)-1}(i) \text{ for } i = 1..n)$ ;  
         $T \leftarrow$  empty tree;
- ④ **foreach** *step of top-down procedure* **do**  
        **foreach** *candidate split*  $c$  **do**  
             $T_c \leftarrow$  add split  $c$  to  $T$ ;  
            **if**  $Mode = Plain$  **then**  
                 $\Delta(i) \leftarrow \text{avg}(grad_r(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i))$  for  $i = 1..n$ ;  
            **if**  $Mode = Ordered$  **then**  
                 $\Delta(i) \leftarrow \text{avg}(grad_{r, \sigma_r(i)-1}(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i), \sigma_r(p) < \sigma_r(i))$  for  $i = 1..n$ ;  
             $loss(T_c) \leftarrow \text{cos}(\Delta, G)$   
             $T \leftarrow \arg \min_{T_c} (loss(T_c))$
- if**  $Mode = Plain$  **then**  
         $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \text{avg}(grad_{r'}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i))$  for  $r' = 1..s, i = 1..n$ ;
- if**  $Mode = Ordered$  **then**  
         $M_{r', j}(i) \leftarrow M_{r', j}(i) - \alpha \text{avg}(grad_{r', j}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i), \sigma_{r'}(p) \leq j)$  for  $r' = 1..s, i = 1..n, j \geq \sigma_{r'}(i) - 1$ ;
- return**  $T, M$

---

# #5. Ordered Boosting

- Practical Implementation

## 1. Initialization

- 학습 데이터셋으로부터  $s+1$ 개의 독립적인 random permutation을 생성
  - $s$ 개의 permutation( $\sigma_1, \dots, \sigma_s$ )은 split을 계산하는 데 사용
  - 1개의 permutation( $\sigma_0$ )은  $s$ 개의 permutation을 통해 얻어진 트리로부터 leaf value( $b_j$ )를 계산하기 위해 사용
- 하나의 permutation만 사용하는 경우 최종 모델 예측에서 분산이 증가하는 문제가 발생
  - 여러 permutation을 적용하여 분산이 커지는 것을 방지

# #5. Ordered Boosting

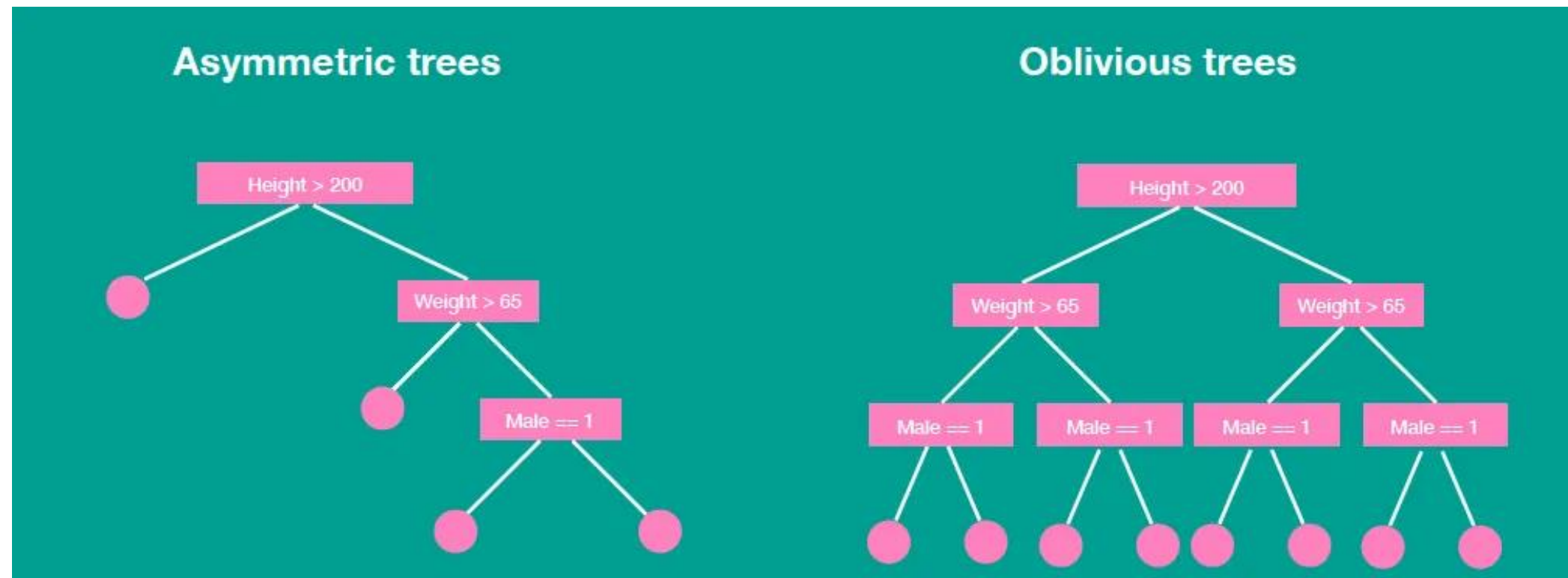
- Practical Implementation

- 2. Ordered Boosting 작동 방식

- └ Ordered 위주로..

- base predictors: oblivious decision tree

- └ oblivious decision tree: 트리의 level마다 동일한 조건(same splitting critetion)을 부여하여 좌우 대칭 형태로 만든 decision tree



- supporting models  $M_{r,j}$ 를 유지

- $M_{r,j}$ : permutation  $\sigma_r$ 에서 처음  $j$ 개의(처음 ~  $j$ 번째) 데이터로 학습된 모델의  $i$ 번째 데이터에 대한 현재 예측값

- gradient는 다음과 같이 계산됨

$$grad_{r,j}(i) = \left. \frac{\partial L(y_i, s)}{\partial s} \right|_{s=M_{r,j}(i)}$$



# #5. Ordered Boosting

- Practical Implementation

## 2. Ordered Boosting 작동 방식

└ Ordered 위주로..

1) 주어진  $L$ (손실함수),  $M$ (모델),  $y$ (target value)를 활용하여 계산

2) 각 반복( $t$ )마다  $\{\sigma_1, \dots, \sigma_s\}$  중에서 permutation  $\sigma_r$  sampling

- 범주형 변수들에 대한 TS는  $\sigma_r$ 을 통해 계산됨
- permutation은 학습 과정에 영향

---

### Algorithm 2: Building a tree in CatBoost

---

```
input :  $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, Mode$   
 $grad \leftarrow CalcGradient(L, M, y);$   
 $r \leftarrow random(1, s);$   
if  $Mode = Plain$  then  
└  $G \leftarrow (grad_r(i) \text{ for } i = 1..n);$   
if  $Mode = Ordered$  then  
└  $G \leftarrow (grad_{r, \sigma_r(i)-1}(i) \text{ for } i = 1..n);$   
 $T \leftarrow \text{empty tree};$   
foreach step of top-down procedure do  
└ foreach candidate split  $c$  do  
└└  $T_c \leftarrow \text{add split } c \text{ to } T;$   
└└ if  $Mode = Plain$  then  
└└└  $\Delta(i) \leftarrow \text{avg}(grad_r(p) \text{ for } p : leaf_r(p) = leaf_r(i)) \text{ for } i = 1..n;$   
└└└ if  $Mode = Ordered$  then  
└└└└  $\Delta(i) \leftarrow \text{avg}(grad_{r, \sigma_r(i)-1}(p) \text{ for } p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i)) \text{ for } i = 1..n;$   
└└└  $loss(T_c) \leftarrow \text{cos}(\Delta, G)$   
└  $T \leftarrow \arg \min_{T_c} (loss(T_c))$   
if  $Mode = Plain$  then  
└  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \text{avg}(grad_{r'}(p) \text{ for } p : leaf_{r'}(p) = leaf_{r'}(i)) \text{ for } r' = 1..s, i = 1..n;$   
if  $Mode = Ordered$  then  
└  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha \text{avg}(grad_{r',j}(p) \text{ for } p : leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \leq j) \text{ for } r' = 1..s, i = 1..n, j \geq \sigma_{r'}(i) - 1;$   
return  $T, M$ 
```

---



# #5. Ordered Boosting

- Practical Implementation

## 2. Ordered Boosting 작동 방식

└ Ordered 위주로..

3)  $\sigma_r$ 에 따라 각 예제( $i$ )마다  $\text{gradient}(\text{grad}_{r,\sigma_r(i)-1}(i))$  계산

---

**Algorithm 2:** Building a tree in CatBoost

---

**input** :  $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, \text{Mode}$

$\text{grad} \leftarrow \text{CalcGradient}(L, M, y);$

$r \leftarrow \text{random}(1, s);$

**if**  $\text{Mode} = \text{Plain}$  **then**

└  $G \leftarrow (\text{grad}_r(i) \text{ for } i = 1..n);$

**if**  $\text{Mode} = \text{Ordered}$  **then**

└  $G \leftarrow (\text{grad}_{r,\sigma_r(i)-1}(i) \text{ for } i = 1..n);$

$T \leftarrow \text{empty tree};$

**foreach** *step of top-down procedure* **do**

└ **foreach** *candidate split*  $c$  **do**

└└  $T_c \leftarrow \text{add split } c \text{ to } T;$

└└ **if**  $\text{Mode} = \text{Plain}$  **then**

└└└  $\Delta(i) \leftarrow \text{avg}(\text{grad}_r(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i)) \text{ for } i = 1..n;$

└└ **if**  $\text{Mode} = \text{Ordered}$  **then**

└└└  $\Delta(i) \leftarrow \text{avg}(\text{grad}_{r,\sigma_r(i)-1}(p) \text{ for } p : \text{leaf}_r(p) = \text{leaf}_r(i), \sigma_r(p) < \sigma_r(i)) \text{ for } i = 1..n;$

└└  $\text{loss}(T_c) \leftarrow \text{cos}(\Delta, G)$

└  $T \leftarrow \arg \min_{T_c} (\text{loss}(T_c))$

**if**  $\text{Mode} = \text{Plain}$  **then**

└  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \text{avg}(\text{grad}_{r'}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i)) \text{ for } r' = 1..s, i = 1..n;$

**if**  $\text{Mode} = \text{Ordered}$  **then**

└  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha \text{avg}(\text{grad}_{r',j}(p) \text{ for } p : \text{leaf}_{r'}(p) = \text{leaf}_{r'}(i), \sigma_{r'}(p) \leq j) \text{ for } r' = 1..s, i = 1..n, j \geq \sigma_{r'}(i) - 1;$

**return**  $T, M$

---

# #5. Ordered Boosting

- Practical Implementation

## 2. Ordered Boosting 작동 방식

└ Ordered 위주로..

### 4) 트리 $T_t$ 생성

- candidate split  $c$  계산
  - 각 예제에 대해 leaf value와 gradient 간의 코사인 유사도를 기반으로 계산됨
- 각 candidate split  $c$ 마다..
  - $i$ 와 동일한 leaf node에 있는 이전의  $p$ 개의 예제의 average gradient를  $i$ 번째 예제에 대한 leaf value( $\Delta(i)$ )로 할당
- 사전에 계산한 그래디언트와 새로 할당한  $\Delta$  사이의 cosine similarity( $\cos(\Delta, G)$ )가 가장 적은 split point  $c$  선정

### 5) 1 ~ 4 반복

---

**Algorithm 2:** Building a tree in CatBoost

---

**input** :  $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, Mode$

$grad \leftarrow CalcGradient(L, M, y);$

$r \leftarrow random(1, s);$

**if**  $Mode = Plain$  **then**

┌  $G \leftarrow (grad_r(i) \text{ for } i = 1..n);$

**if**  $Mode = Ordered$  **then**

┌  $G \leftarrow (grad_{r, \sigma_r(i)-1}(i) \text{ for } i = 1..n);$

$T \leftarrow \text{empty tree};$

**foreach** step of top-down procedure **do**

┌ **foreach** candidate split  $c$  **do**

┌  $T_c \leftarrow \text{add split } c \text{ to } T;$

┌ **if**  $Mode = Plain$  **then**

┌  $\Delta(i) \leftarrow \text{avg}(grad_r(p) \text{ for } p : leaf_r(p) = leaf_r(i)) \text{ for } i = 1..n;$

┌ **if**  $Mode = Ordered$  **then**

┌  $\Delta(i) \leftarrow \text{avg}(grad_{r, \sigma_r(i)-1}(p) \text{ for } p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i)) \text{ for } i = 1..n;$

┌  $loss(T_c) \leftarrow \cos(\Delta, G)$

┌  $T \leftarrow \arg \min_{T_c} (loss(T_c))$

**if**  $Mode = Plain$  **then**

┌  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \text{avg}(grad_{r'}(p) \text{ for } p : leaf_{r'}(p) = leaf_{r'}(i)) \text{ for } r' = 1..s, i = 1..n;$

**if**  $Mode = Ordered$  **then**

┌  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha \text{avg}(grad_{r',j}(p) \text{ for } p : leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \leq j) \text{ for } r' = 1..s, i = 1..n, j \geq \sigma_{r'}(i) - 1;$

**return**  $T, M$

---

# #3. Categorical Features

- Ordered Boosting - Example

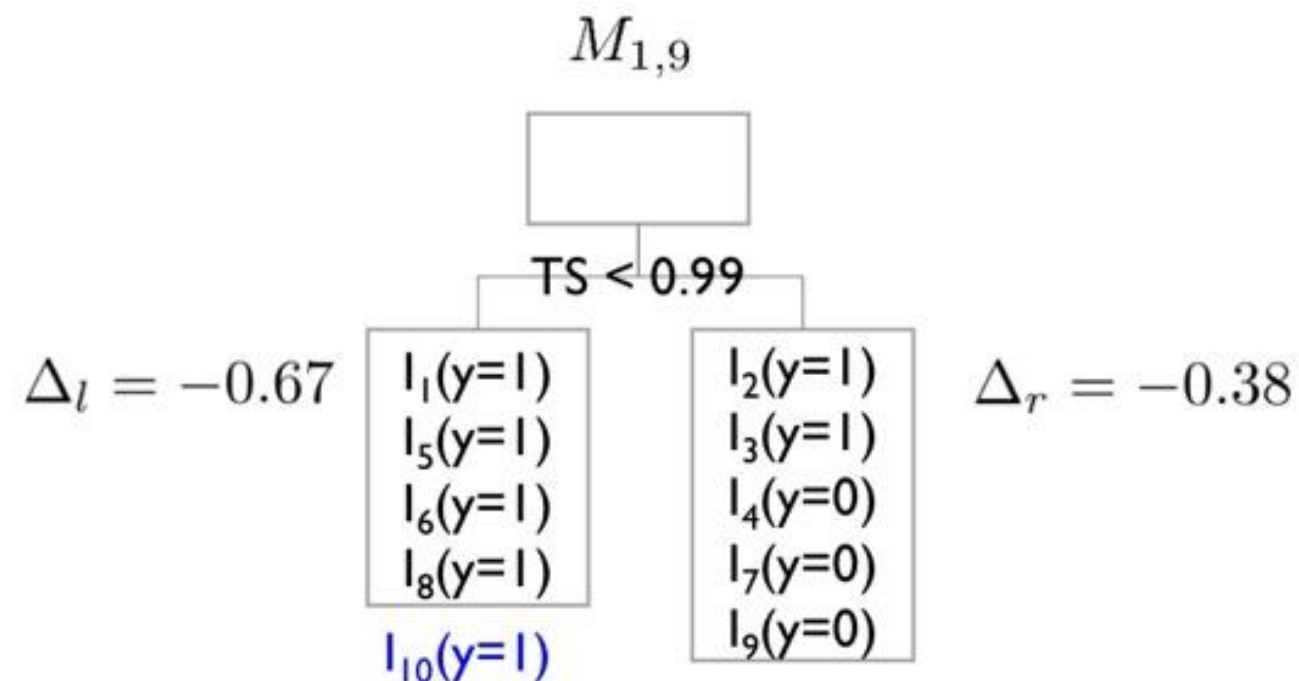
출처: 고려대학교 DSBA 연구실 - 강필성 교수님 강의자료

- Ordered Boosting Example

- ✓ Assumption: squared loss (gradient:  $f(x) - y$ )

- ✓ random permutation is conducted

	...	$x^i$	...	TS	$y$	G	$\Delta$
$I_1$	...	A	...	0.000	1	0	0
$I_2$	...	B	...	1.000	1	0	0
$I_3$	...	C	...	1.000	1	-1	0
$I_4$	...	A	...	1.000	0	0	-0.5
$I_5$	...	B	...	0.977	1	-1	0
$I_6$	...	C	...	0.982	1	-1	-0.5
$I_7$	...	B	...	0.992	0	-0.5	-0.33
$I_8$	...	C	...	0.986	1	-1.5	-0.67
$I_9$	...	C	...	0.992	0	-0.33	-0.38
$I_{10}$	...	C	...	0.748	1	-1.67	-0.58



$$G(I_{10}) = grad_{1,\sigma_1(10)-1}(I_{10})$$

$$= \Delta_l - y_{10} = -0.67 - 1 = -1.67$$

$$\Delta(I_{10}) = \frac{1}{4} (grad_{1,\sigma_1(10)-1}(I_1) + grad_{1,\sigma_1(10)-1}(I_5)$$

$$+ grad_{1,\sigma_1(10)-1}(I_6) + grad_{1,\sigma_1(10)-1}(I_8))$$

$$= \frac{1}{4} (0 - 1 - 1 - 0.33) = -0.58$$

# #5. Ordered Boosting

- Practical Implementation

## 3. Choosing leaf values

- 구성된 모든 tree가 주어지면, 최종 모델  $F$ 의 leaf value들은 표준 gradient boosting 과정에 의해 계산됨
- 학습 예제  $i$ 는  $leaf_0(i)$ 에 대응됨
  - TS 계산을 위해 permutation  $\sigma_0$  활용
- 테스트 시에 새로운 예제에 대해 최종 모델  $F$ 가 적용되면, TS는 전체 training data에 대해 계산되게 됨
  - ⇒ Property 2 만족

# 6. Experiments





# #6. Experiments

- Settings

- 모든 학습 알고리즘에 대해 범주형 특성을 ordered TS 방법을 사용하여 전처리 하였음
- 약 80%의 데이터로 매개변수 튜닝 및 훈련을 하고, 나머지 20% 데이터로 테스트 수행
- 성능 평가 지표: logloss, zero-one loss

- Comparison with baseline

Table 8: Comparison with baselines: logloss / zero-one loss, relative increase is presented in the brackets.

	CatBoost	LightGBM	XGBoost
Adult	<b>0.2695 / 0.1267</b>	0.2760 (+2.4%) / 0.1291 (+1.9%)	0.2754 (+2.2%) / 0.1280 (+1.0%)
Amazon	<b>0.1394 / 0.0442</b>	0.1636 (+17%) / 0.0533 (+21%)	0.1633 (+17%) / 0.0532 (+21%)
Click	<b>0.3917 / 0.1561</b>	0.3963 (+1.2%) / 0.1580 (+1.2%)	0.3962 (+1.2%) / 0.1581 (+1.2%)
Epsilon	<b>0.2647 / 0.1086</b>	0.2703 (+1.5%) / 0.114 (+4.1%)	0.2993 (+11%) / 0.1276 (+12%)
Appetency	<b>0.0715 / 0.01768</b>	0.0718 (+0.4%) / 0.01772 (+0.2%)	0.0718 (+0.4%) / 0.01780 (+0.7%)
Churn	<b>0.2319 / 0.0719</b>	0.2320 (+0.1%) / 0.0723 (+0.6%)	0.2331 (+0.5%) / 0.0730 (+1.6%)
Internet	<b>0.2089 / 0.0937</b>	0.2231 (+6.8%) / 0.1017 (+8.6%)	0.2253 (+7.9%) / 0.1012 (+8.0%)
Upselling	<b>0.1662 / 0.0490</b>	0.1668 (+0.3%) / 0.0491 (+0.1%)	0.1663 (+0.04%) / 0.0492 (+0.3%)
Kick	<b>0.2855 / 0.0949</b>	0.2957 (+3.5%) / 0.0991 (+4.4%)	0.2946 (+3.2%) / 0.0988 (+4.1%)

- LightGBM과 XGBoost 모두 CatBoost에 비해 더 큰 loss를 가짐
- CatBoost에서 개선된 loss에 대해 Paired one-tail t-test를 진행한 결과 Appetency, Churn, Upselling 데이터셋을 제외하고 모두 유의수준 0.01 하에서 통계적 유의성이 입증됨

# #6. Experiments

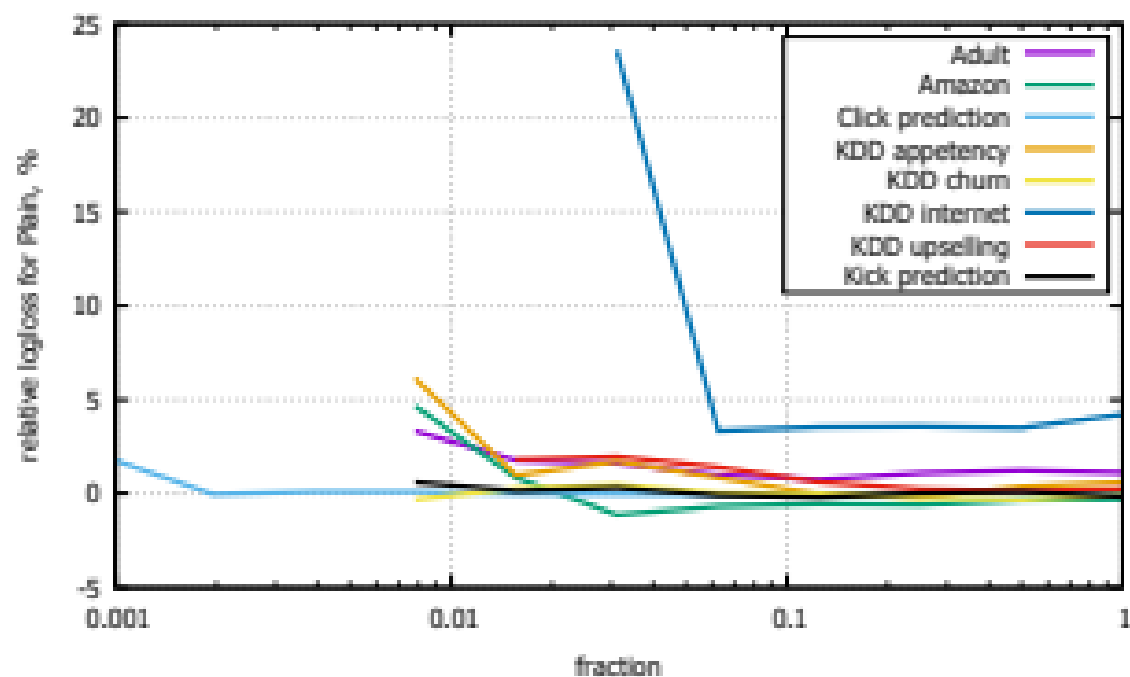
- Ordered and Plain modes

- CatBoost의 두 가지 필수적인 boosting 모드를 비교함
  - Plain
  - Ordered
- 작은 dataset일수록 ordered mode의 성능이 plain mode보다 좋음
  - Adult, Internet 데이터셋

Table 3: Plain boosting mode: logloss, zero-one loss and their change relative to Ordered boosting mode.

	Logloss	Zero-one loss
Adult	0.272 (+1.1%)	0.127 (-0.1%)
Amazon	0.139 (-0.6%)	0.044 (-1.5%)
Click	0.392 (-0.05%)	0.156 (+0.19%)
Epsilon	0.266 (+0.6%)	0.110 (+0.9%)
Appetency	0.072 (+0.5%)	0.018 (+1.5%)
Churn	0.232 (-0.06%)	0.072 (-0.17%)
Internet	0.217 (+3.9%)	0.099 (+5.4%)
Upselling	0.166 (+0.1%)	0.049 (+0.4%)
Kick	0.285 (-0.2%)	0.095 (-0.1%)

- 데이터셋의 일부분만을 가지고 학습할수록(= 데이터셋의 크기가 작아질수록) ordered mode에 비해 plain mode의 error가 더 큰 것을 확인할 수 있음



# #6. Experiments

- Analysis of target statistics
  - Ordered TS가 모든 데이터셋에서 가장 좋은 성능을 보임
  - Ordered TS를 제외한 나머지 3개 방법 중에서는 Holdout 방식의 성능이 가장 좋았음
  - 거의 모든 경우에서 Leave-one-out TS가 Greedy TS에 비해 좋은 성능을 보였으나, categorical feature의 각 category의 빈도가 높은 Adult dataset의 경우 반대 결과가 도출됨

Table 4: Comparison of target statistics, relative change in logloss / zero-one loss compared to ordered TS.

	Greedy	Holdout	Leave-one-out
Adult	+1.1% / +0.8%	+2.1% / +2.0%	+5.5% / +3.7%
Amazon	+40% / +32%	+8.3% / +8.3%	+4.5% / +5.6%
Click	+13% / +6.7%	+1.5% / +0.5%	+2.7% / +0.9%
Appetency	+24% / +0.7%	+1.6% / -0.5%	+8.5% / +0.7%
Churn	+12% / +2.1%	+0.9% / +1.3%	+1.6% / +1.8%
Internet	+33% / +22%	+2.6% / +1.8%	+27% / +19%
Upselling	+57% / +50%	+1.6% / +0.9%	+3.9% / +2.9%
Kick	+22% / +28%	+1.3% / +0.32%	+3.7% / +3.3%



# #6. Experiments

- Feature combinations

- 조합할 수 있는 최대 feature의 수인  $c_{max}$ 에 대해..
  - $1 < c_{max} \leq 2$ : feature combination을 수행하지 않았을 때에 비해 평균적으로 logloss가 1.86% 개선되었음
  - $1 < c_{max} \leq 3$ : 평균적으로 logloss가 2.04% 개선되었음
  - $c_{max} > 3$ : 유의미한 성능 향상을 보이지 않았음

- Number of permutations

- model 학습 시 사용되는 random permutation  $\sigma_r$ 의 개수  $s$ 에 대해..
  - $s = 1$ 일 때보다  $s = 3$ 일 때 logloss 0.19% 개선됨
  - $s = 1$ 일 때보다  $s = 9$ 일 때 logloss 0.38% 개선됨
  - ⇒  $s$ 가 증가할수록 성능이 개선되는 경향을 보임

# 7. Conclusion



# #7. Conclusion

- Gradient Boosting이 가지고 있던 두 가지 Statistical Issue를 해결

## 1. Target Leakage → Ordered Target Statistics

- feature  $\hat{x}_k^i$ 의 값을 계산하는 데 target인  $y_k$ 값이 사용되는 현상
- training 시점에 알 수 없는 target 값을 활용

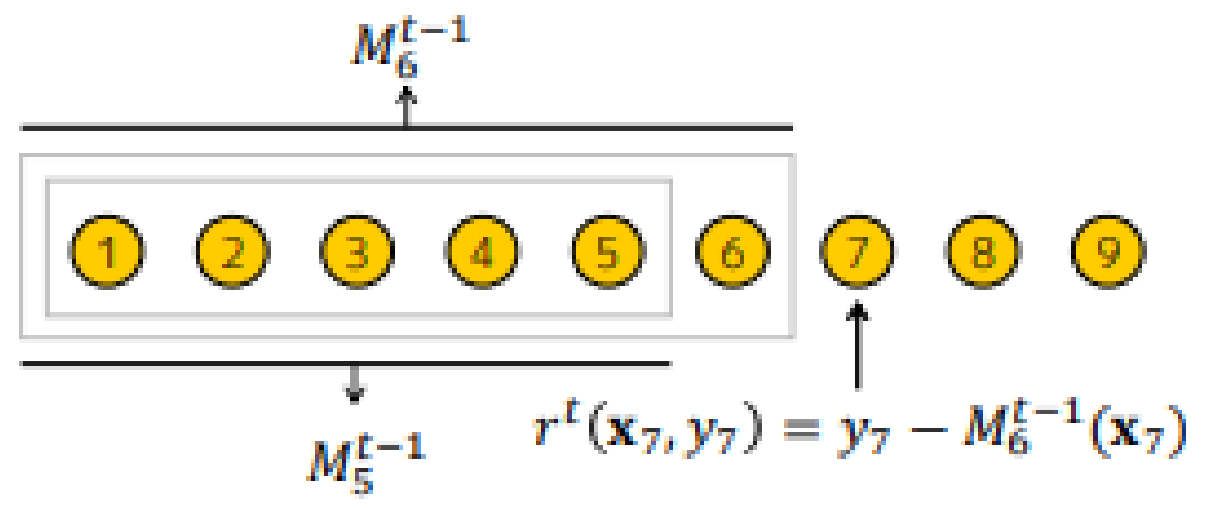
⇒ 해결



## 2. Prediction Shift → Ordered Boosting

$$h^t = \underset{h \in H}{\operatorname{argmin}} \mathbb{E}(-g^t(x, y) - h(x))^2 \approx \underset{h \in H}{\operatorname{argmin}} \frac{1}{n} \sum_{k=1}^n (-g^t(x_k, y_k) - h(x_k))^2$$

⇒ 해결



# #7. Conclusion

---

- CatBoost는..
    - 여러 benchmark data에 대해 SOTA 달성
    - High Cardinality(범주가 많은) categorical feature를 다루는 데 용이
    - model tuning 간소화, 높은 예측 성능
- ⇒ 많은 예측 task에서 널리 사용됨

# THANK YOU

