CatBoost: unbiased boosting with categorical features

Kobe University Engineering
M2 201T264T
Fuki Yamamoto

Flow of the presentation

I. Background

II. Preliminaries

III. Ordered TS

IV. Ordered Boosting

Main Section

V. Experiments & Comparison

VI. Conclusion

Flow of the presentation

I. **Background**

II. Preliminaries

III. Ordered TS

IV. Ordered Boosting

V. Experiments & Comparison

VI. Conclusion

What is CatBoost?

- ✓ CatBoost is **one of the implementations of GBDT** (Most recent)

dmlc XGBoost → LightGBM → CatBoost

- ✓ Implemented algorithm to **pre-process categorical features** for GBDT
  - Ordered TS

- ✓ Implemented some mechanisms to train model
  - Ordered Boosting
  - Symmetric Tree (Oblivious Tree)

- ✓ Implemented several advanced functions (Presented in another papers)
  - Stochastic Gradient Langevin Boosting
  - Estimate uncertainty

What is CatBoost?

- ✓ CatBoost is **one of the implementations of GBDT** (Most recent)

dmlc XGBoost → LightGBM → CatBoost

Scope of this paper & today's presentaion

- ✓ Implemented algorithm to **pre-process categorical features** for GBDT
  - **Ordered TS**
- ✓ Implemented some mechanisms to train model
  - **Ordered Boosting**
  - Symmetric Tree (Oblivious Tree)

- ✓ Implemented several advanced functions (Presented in another papers)
  - Stochastic Gradient Langevin Boosting
  - Estimate uncertainty

Flow of the presentation

I. Background & Purpose

II. Preliminaries

III. Ordered TS

IV. Ordered Boosting

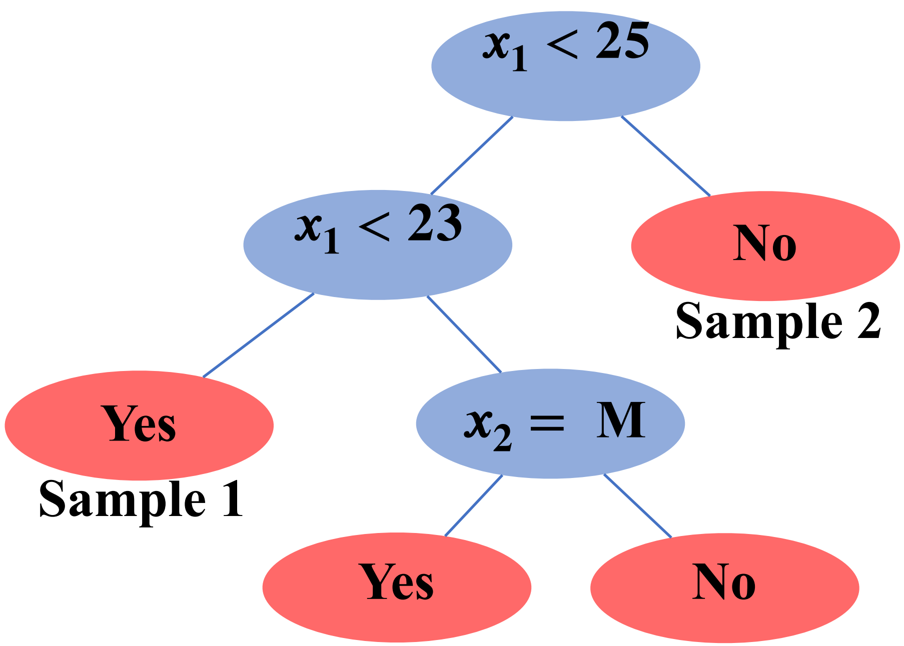
V. Experiments & Comparison

VI. Conclusion

Decision Tree

✓ One of machine learning models with a **tree structure**

✓ Output the leaf weights that match the thresholds



Task: Student or not
x_1: Age
x_2: Gender

Sample1: $x_1 = 15$, $x_2 =$ M
→ Yes

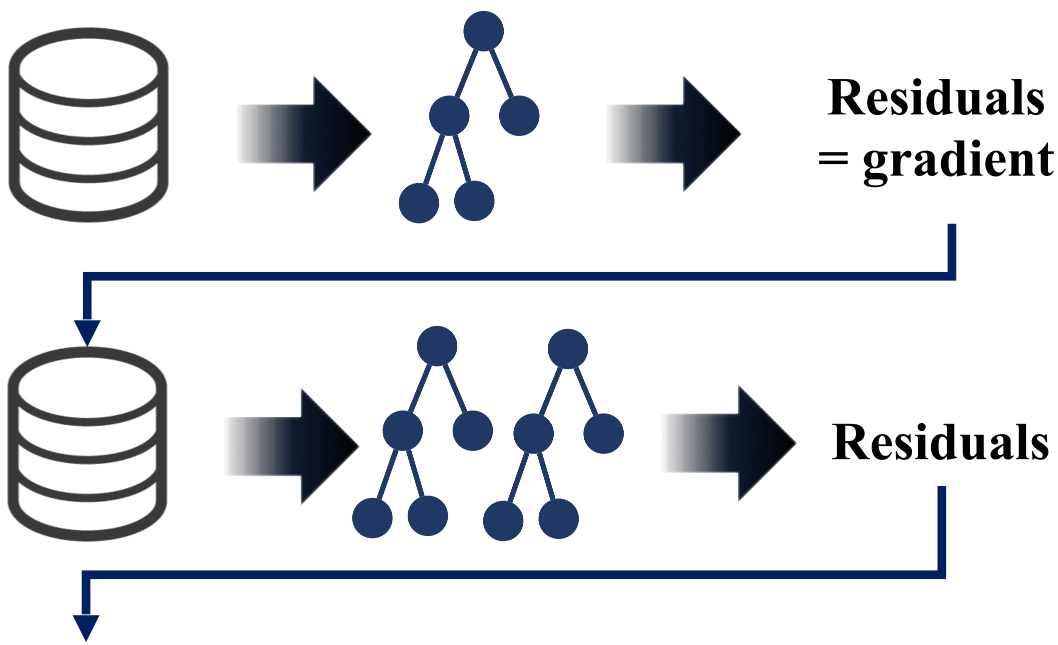
Sample2: x_1=30, $x_2 =$ F
→ No

Gradient Boosting

✓ A technique that **sequentially learns weak learners to fit the residuals**

✓ In many cases, **decision trees** are used as weak learners

✓ High performance with tabular data



Major Libraries

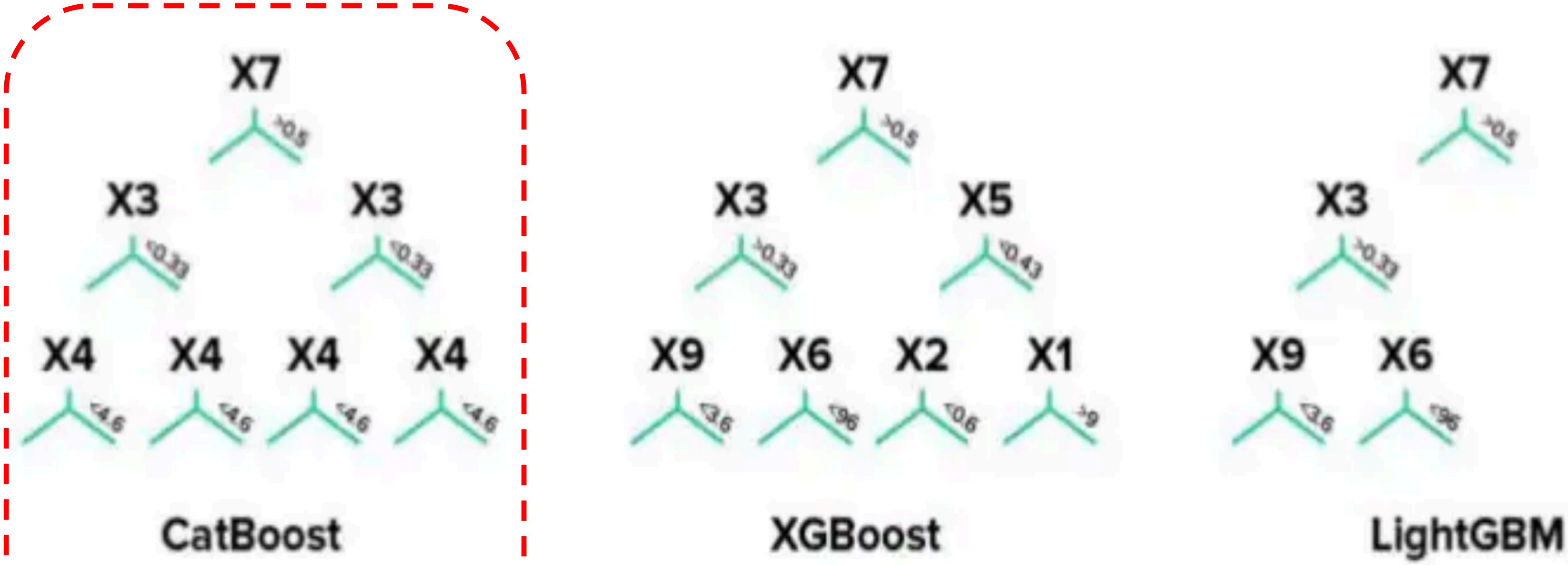
dmlc XGBoost

LightGBM

CatBoost

Gradient Boosting Decision Trees

- ✓ Each library uses a **different structure of trees**
- ✓ CatBoost uses **symmetric trees with the same threshold at each depth**
- ✓ Symmetric trees is suitable for parallel processing



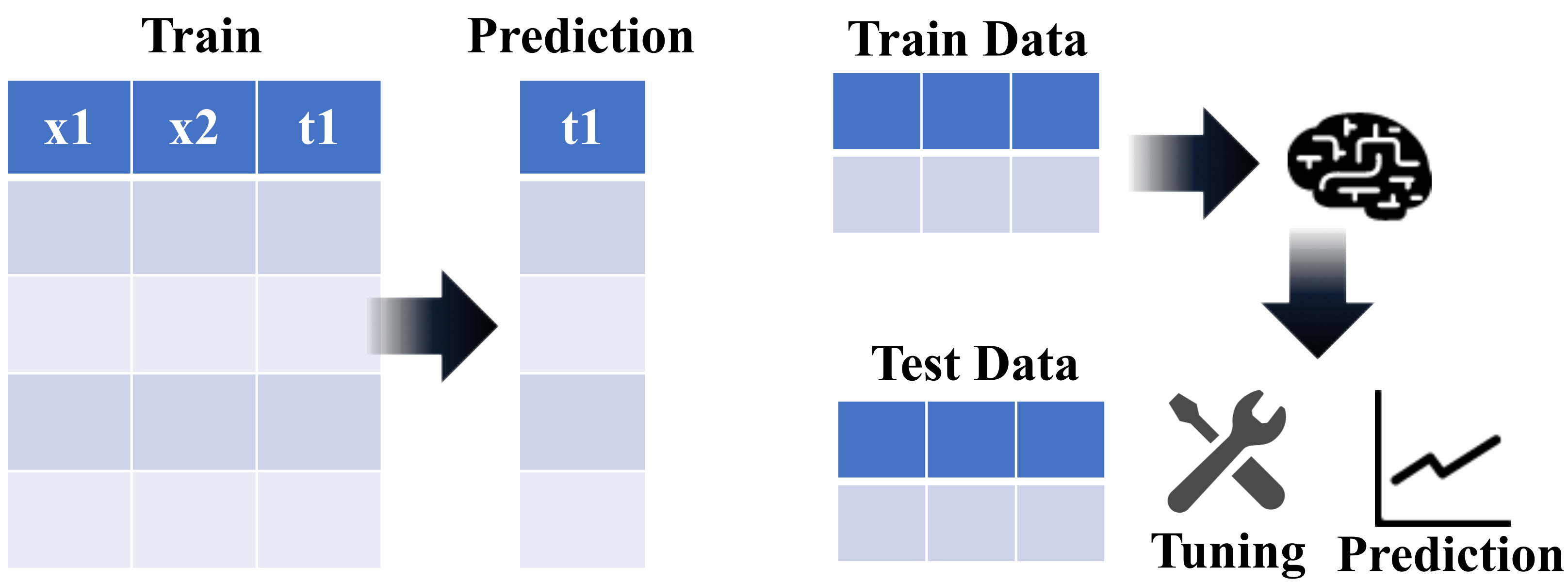
Symmetric **Depth-wise** **Leaf-wise**

Gradient Boosting Decision Trees

model	Tree	Special Features
XGBoost	**Depth-wise**	• **Sparsity-aware**
Light GBM	**Leaf-wise**	• **TS (use grad info)** • **EFB** • **GOSS**
CatBoost	**Symmetric**	• **Ordered Boosting** • **Ordered TS** • **Estimate uncertainty** • **SGLB** • **Mono Forest**

Target Leakage (Data Leakage)

- ✓ Use **information not available at the prediction** in the learning process
- ✓ Causes **overfitting and over-estimation** of model performance
- ✓ E.g., include targets in the features, tune hyper-parameters with test data



Pre-processing of categorical features

✓ **One Hot Encoding**

- Converted to vectors that indicate if it belongs to each category
- In high cardinality cases, **number of features increases** significantly

✓ **Target Encoding (Target Statistics, TS)**

- Replace categories with statistics **calculated from the target**
- Prone to cause **Target Leakage** (Especially categories with little data)

One Hot Encoding

x1	t1
A	1
A	1
B	0
C	1

→

xA	xB	xC
1	0	0
1	0	0
0	1	0
0	0	1

Target Encoding (TS)

x1	t1
A	1
A	1
B	0
B	1

→

x1	t1
1.0	1
1.0	1
0.5	0
0.5	1

Existing TS methods

✓ **Leave One Out TS**

- Calculate TS at training with data other than self
- Calculate TS at inference for all data
- **Causes Target Leakage**

✓ **Hold Out TS**

- Calculate TS at training with Hold-out data
- Calculate TS at inference for all data
- **Decrease the amount of training data**

✓ **Light GBM TS**

- Calculate (simple) TS by gradient for each node
- Process categorical features automatically
- **Causes Target Leakage**

Leave-one-out TS

x1	t1
A	1
A	1
A	0
B	1

x1	t1
0.5	1
0.5	1
1.0	0
0	1

Hold-out TS

x1	t1
A	1

x1	t1
A	0
A	1

x1	t1
A	0.5

Existing Problems

✓ **Existing Pre-processing for categorical features**

- Large increase in the number of features (One Hot Encoding)
- Causes **Target Leakage** (TS)

✓ **Existing GBDT**

- Building all trees and calculation of all residuals **both use** $\boldsymbol{D \subset D_{train}}$

→ Causes **Prediction Shift** (Overfitting)

→ Degraded performance on small datasets

So, CatBoost propose **Ordered TS + Ordered Boosting**

Flow of the presentation

I. Preliminaries

II. Background & Purpose

III. Ordered TS

IV. Ordered Boosting

V. Experiments & Comparison

VI. Conclusion

Algorithm of Ordered TS

- ✓ Calculate TS based on **the order of randomly sorted data**
- ✓ Calculates TS **from data before itself**
- ✓ Simulate sequential **sampling of data from true distribution**

id	x1	t1
1	A	1
2	A	1
3	A	0
4	B	1
5	B	0
6	B	0
7	B	1

Random Sort

id	x1	t1
7	B	1
5	B	0
3	A	0
1	A	1
4	B	1
2	A	1
6	B	0

Calculate TS

id	x1	t1
7	0	1
5	1.0	0
3	0	0
1	0	1
4	0.5	1
2	0.5	1
6	0.33	0

Ordered TS

Benefits

✓ No need to decrease the amount of training data

✓ Computation based on random order **reduces Target Leakage**

However, TS varies greatly depending on the result of sorting

How to use it in GBDT

Training: Randomly sort the data and calculate TS **for each tree build**

Test: Calculate TS with all training data

Flow of the presentation

I. Preliminaries

II. Background & Purpose

III. Ordered TS

IV. Ordered Boosting

V. Experiments & Comparison

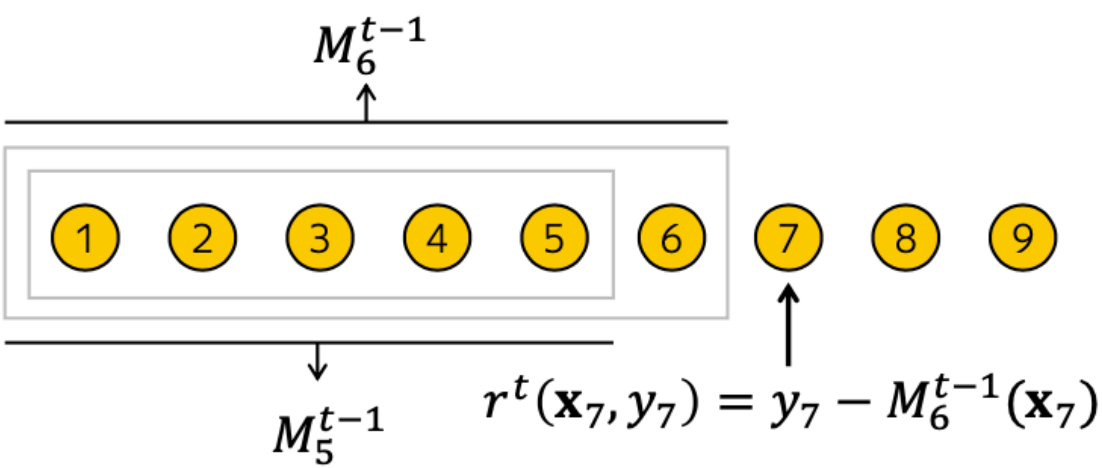
VI. Conclusion

Exact Algorithm of Ordered Boosting

✓ Use **different data** for gradient calculation and tree build

✓ Compute **gradient from the model trained on dataset before itself**

✓ Need to train *n*(the number of data) models(GBDT)
→ Difficult → CatBoost propose more practical algorithm



M_i^t: Model trained using first i examples for t-th iteration

y_i: i-th target

r^t: residuals (gradient) for t-th iteration

Exact Algorithm of Ordered Boosting

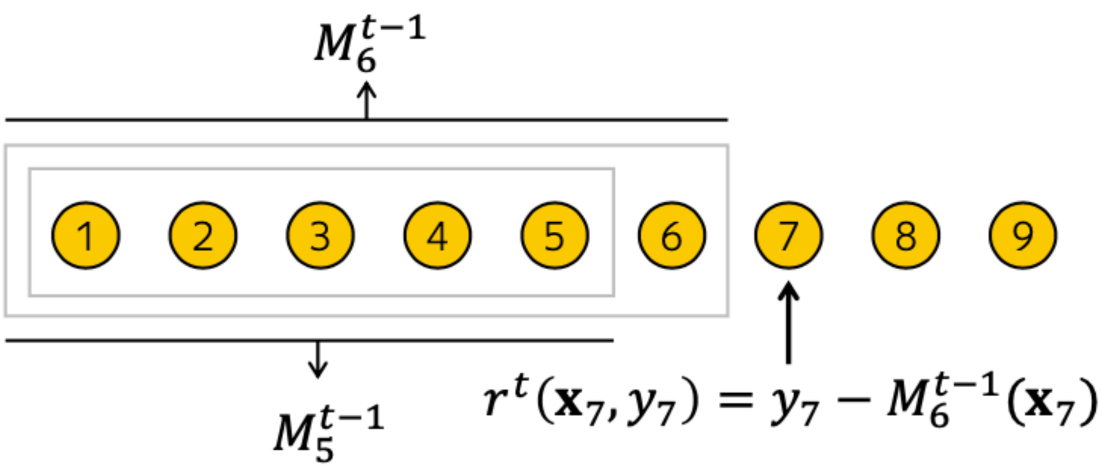
M_i^t: GBDT trained on first i examples for t-th iteration

ΔM: Decision Tree

y_i: i-th target

r^t: residuals (gradient) for t-th iteration

I: the number of trees



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**
    $\Delta M \leftarrow$
     $LearnModel((\mathbf{x}_j, r_j) :$
     $\sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Practical Algorithm of Ordered Boosting

Changes

✓ **Definition of M_i:**

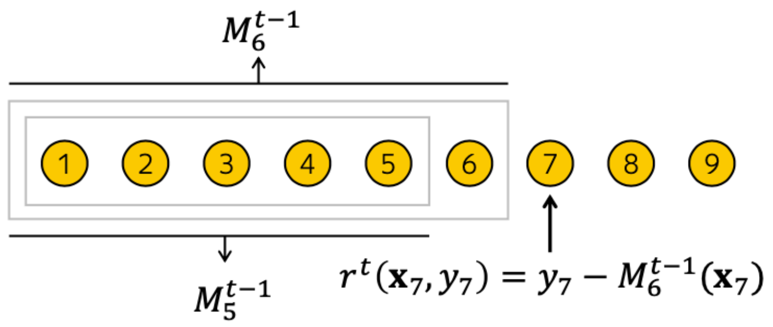
Model trained on first i examples

→ **Outputs** calculate & update by first i examples

(**Tree structure is common**, independent of order)

✓ **Prediction**

Use M_n → Use model that calculate leaf weights for all data



Practical Algorithm of Ordered Boosting

✓ **Use malti-random order $\boldsymbol{\sigma_r}$** $(\boldsymbol{r} = \mathbf{0},\ \ldots,\ \boldsymbol{s})$ (1 based):

$\boldsymbol{r} = \mathbf{0}$ use to calculate TS at prediction and standard order

$\boldsymbol{r} = \mathbf{1} \sim \boldsymbol{s}$ use to build each trees & calculate Ordered TS

✓ **Number of models to train:** $n \to 1$

Randomly select r from $1 \sim s$ **for each tree build**

Compute **leaf weights and split scores** based on order σ_r

✓ **Use ordered output instead of $\boldsymbol{M_i}$** (Model trained on the data **up to $\boldsymbol{i}$-th)**

$\to M_{r,j}$: i-th output calculated by first 2^j examples $(i \leq 2^{j+1},\ \ j\!: 1, \ldots, \log_2 n)$

$\to grad_{r,j}$: i-th gradient calculated by first 2^j examples

$(i \leq 2^{j+1},\ \ j\!: 1, \ldots, \log_2 n)$

Practical Algorithm of Ordered Boosting

$\boldsymbol{Update\_M(M_r,\ D_{train},\ \sigma)}$:

$M_{r,j}(i) = avg(grad_{r,j}(k)$

k in $same\ leaf\ as$ the $i - th\ data)$

return $M_{r,j}$

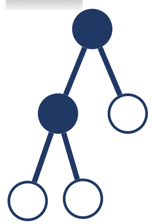
$\boldsymbol{Build\_tree(M_{r,j},\ D_{train},\ \sigma)}$:

$r \leftarrow random(1,s)$

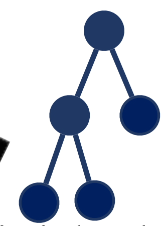
$grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)$

$T_t \leftarrow tree.fit\left(X, grad_{r,j}\right)$

return T_t



Tree Structure



Finished Tree

$\boldsymbol{Calc\_leaf\_weight(T_t,\ D_{train})}$:

Calculate leaf weights
based on tree structure for all data
return w^t

$\boldsymbol{Prediction(T,\ D_{test})}$:

$Ordred\_TS(D_{test},\ \sigma_0)$

return $\sum \alpha w^t$

(α: learning rate)

Practical Algorithm of Ordered Boosting

$Init()$:

$M_{r,j}(i) \leftarrow 0$

$\sigma_r \leftarrow random\_permutaion()$

return $M_{r,j},\ \sigma$

$\boldsymbol{Build\_tree(M_{r,j},\ D_{train},\ \sigma)}$:

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

$\boldsymbol{grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)}$

$\boldsymbol{T_t \leftarrow tree\,.\,fit(X, grad_{r,j})}$

return $\boldsymbol{T_t}$

Each factor's shape & definition

✓ $\boldsymbol{\sigma = (s,\ n),\ \sigma_r = (n)}$

✓ $\boldsymbol{M = grad = (r, n)}$

✓ $\boldsymbol{M_{r,j}(i) = (n)}$

i-th output calculated by first 2^j examples ($i \leq 2^{j+1},\ \ j:\ 1,\ldots,\log_2 n$)

✓ $\boldsymbol{grad_{r,j}(i) = (n)}$

i-th gradient calculated by first 2^j examples ($i \leq 2^{j+1},\ \ j:\ 1,\ldots,\log_2 n$)

Practical Algorithm of Ordered Boosting

$\boldsymbol{Update\_M(M_r,\ D_{train},\ \sigma)}$:

$M_{r,j}(i) = avg(grad_{r,j}\ (k)$

k in $same\ leaf\ as\ the\ i-th\ data)$

return $M_{r,j}$

$\boldsymbol{Build\_tree(M_{r,j},\ D_{train},\ \sigma)}$:

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

$\boldsymbol{grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)}$

$\boldsymbol{T_t \leftarrow tree\,.\,fit(X, grad_{r,j})}$

return $\boldsymbol{T_t}$

Each factor's shape & definition

✓ $\boldsymbol{\sigma = (s,\ n),\ \sigma_r = (n)}$

✓ $\boldsymbol{M = grad = (r, n)}$

✓ $\boldsymbol{M_{r,j}(i) = (n)}$

i-th output calculated by data up to 2^j $(i \leq 2^{j+1},\ \ j:\ 1,\ldots,\log_2 n)$

✓ $\boldsymbol{grad_{r,j}(i) = (n)}$

i-th gradient calculated by data up to 2^j $(i \leq 2^{j+1},\ \ j:\ 1,\ldots,\log_2 n)$

Practical Algorithm of Ordered Boosting

Update_M $(M_r,\ D_{train},\ \sigma)$**:**

$M_{r,j}(i) = avg(grad_{r,j}\ (k)$

k in $same\ leaf\ as\ the\ i-th\ data)$

return $M_{r,j}$

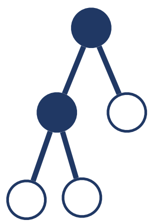
Build_tree $(M_{r,j},\ D_{train},\ \sigma)$**:**

$r \leftarrow random(1,s)$

$grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)$

$T_t \leftarrow tree\ .\ fit(X, grad_{r,j})$

return T_t



Tree Structure

Calc_leaf_weight $(T_t,\ D_{train})$**:**

Calculate leaf weights

based on tree structure for all data

return w^t

Prediction $(T,\ D_{test})$**:**

$Ordred\_TS(D_{test},\ \sigma_0)$

return $\sum \alpha w^t$

(α: learning rate)

Practical Algorithm of Ordered Boosting

Update_M$(M_r,\ D_{train},\ \sigma)$**:**

$M_{r,j}(i) = avg(grad_{r,j}\ (k)$

k in $same\ leaf\ as\ the\ i-th\ data)$

return $M_{r,j}$

Calculate $M_{r,j}(i)$ $(2^j < i \leq 2^{j+1})$

with the first 2^j **data**

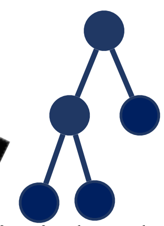
Build_tree$(M_{r,j},\ D_{train},\ \sigma)$**:**

$r \leftarrow random(1,s)$

$grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)$

$T_t \leftarrow tree.fit(X, grad_{r,j})$

return T_t



Finished Tree

Calc_leaf_weight$(T_t,\ D_{train})$**:**

Calculate leaf weights

based on tree structure for all data

return w^t

Prediction$(T,\ D_{test})$**:**

$Ordred\_TS(D_{test},\ \sigma_0)$

return $\sum \alpha w^t$

(α: learning rate)

Practical Algorithm of Ordered Boosting

$\boldsymbol{Update\_M(M_r,\ D_{train},\ \sigma)}$:

$M_{r,j}(i) = avg(grad_{r,j}(k)$

k in $same\ leaf\ as\ the\ i-th\ data)$

return $M_{r,j}$

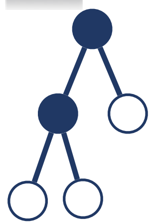
$\boldsymbol{Build\_tree(M_{r,j},\ D_{train},\ \sigma)}$:

$r \leftarrow random(1,s)$

$grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)$

$T_t \leftarrow tree.fit(X, grad_{r,j})$

return T_t



Tree Structure

$\boldsymbol{Calc\_leaf\_weight(T_t,\ D_{train})}$:

Calculate leaf weights

based on tree structure for all data

return w^t

$\boldsymbol{Prediction(T,\ D_{test})}$:

$Ordred\_TS(D_{test},\ \sigma_0)$

return $\sum \alpha w^t$

(α: learning rate)

Practical Algorithm of Ordered Boosting

$\boldsymbol{Update\_M(M_r,\ D_{train},\ \sigma)}$:

$M_{r,j}(i) = avg(grad_{r,j}\,(k)$

k in $same\ leaf\ as\ the\ i-th\ data)$

return $M_{r,j}$

$\boldsymbol{Build\_tree(M_{r,j},\ D_{train},\ \sigma)}$:

$r \leftarrow random(1,s)$

$grad_{r,j}(i) \leftarrow y_{r,i} - M_{r,j}(i)$

$T_t \leftarrow tree.fit(X, grad_{r,j})$

return T_t

$\boldsymbol{Calc\_leaf\_weight(T_t,\ D_{train})}$:

Calculate leaf weights

based on tree structure for all data

return w^t

$\boldsymbol{Prediction(T,\ D_{test})}$:

$Ordred\_TS(D_{test},\ \sigma_0)$

return $\sum \alpha w^t$

(α: learning rate)

Computation Complexity (for one iteration)

✓ **Calculate gradient: $O(n) \to O(s \cdot n)$ (s: number of random orders)**

Compute the gradient corresponding to s random order

✓ **Build Tree (histogram method): $O(f \cdot n)$ (f: number of features)**

Same computation cost as plain GBDT

✓ **Update $M_{r,j}$: $O(n) \to O(s \cdot n)$**

$$(s+1) \cdot \sum_{j=1}^{\log_2 n} 2^{j+1} = (s+1)s \cdot \left(2^2 + 2^3 + \ldots + 2^{\log_2 n} + 2^{\log_2 n+1}\right)$$

$$\Longleftrightarrow \quad = (\mathrm{s}+1) \cdot 2^{\log_2 n+1}\left(1 + \frac{1}{2} + \frac{1}{2^2} + \ldots\right) < (s+1) \cdot 2^{\log_2 n+2} = 4(s+1)n$$

✓ **Calculate Ordered TS: $O\left(N_{TS} \cdot n\right)$**

(N_{TS} : Number of TS not calculated previous iteration)

Ordered Boosting

✓ **Benefits**

- Reduce **Prediction Shift** (Fitting to training data distribution)
- Expected to improve performance on **small datasets**

However, the computational cost increases (Ordered TS)

✓ **Feature Combination**

- Automate the combination of category features
- Difficult to generate all combinations
- Combine all other categories **with the category used for previous splits**

Flow of the presentation

I. Preliminaries

II. Background & Purpose

III. Ordered TS

IV. Ordered Boosting

V. Experiments & Comparison

VI. Conclusion

V. Experiments & Comparison

Datasets (Classifficaion)

Dataset name	Instances	Features	Description
Adult[11]	48842	15	Prediction task is to determine whether a person makes over 50K a year. Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: (AAGE>16) and (AGI>100) and (AFNLWGT>1) and (HRSWK>0)
Amazon[12]	32769	10	Data from the Kaggle Amazon Employee challenge.
Click Prediction[13]	399482	12	This data is derived from the 2012 KDD Cup. The data is subsampled to 1% of the original number of instances, downsampling the majority class (click=0) so that the target feature is reasonably balanced (5 to 1). The data is about advertisements shown alongside search results in a search engine, and whether or not people clicked on these ads. The task is to build the best possible model to predict whether a user will click on a given ad.
Epsilon[14]	400000	2000	PASCAL Challenge 2008.
KDD appetency[15]	50000	231	Small version of KDD 2009 Cup data.
KDD churn[16]	50000	231	Small version of KDD 2009 Cup data.
KDD Internet[17]	10108	69	Binarized version of the original dataset. The multi-class target feature is converted to a two-class nominal target feature by re-labeling the majority class as positive ('P') and all others as negative ('N'). Originally converted by Quan Sun.
KDD upselling[18]	50000	231	Small version of KDD 2009 Cup data.
Kick prediction[19]	72983	36	Data from "Don't Get Kicked!" Kaggle challenge.

Small

Ordered TS vs. Other TS methods

- ✓ Compare **each TS methods** with CatBoost (order mode)
- ✓ The same results were obtained in Plain mode
- ✓ **Ordered TS showed best results**

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%

Symmetric Trees vs. Other Tree methods

- ✓ Compare **each GBDT methods + Ordered TS** with CatBoost (plain mode)
- ✓ Match other algorithms of each method as closely as possible
- ✓ None of the GBDT methods are clearly superior

Table 9: Comparison with baselines: logloss / zero-one loss (relative increase for baselines).

	Raw setting of CatBoost	LightGBM	XGBoost
Adult	0.2800 / 0.1288	-1.4% / +0.2%	-1.7% / -0.6%
Amazon	0.1631 / 0.0533	+0.3% / 0%	+0.1% / -0.2%
Click	0.3961 / 0.1581	+0.1% / -0.1%	0% / 0%
Appetency	0.0724 / 0.0179	-0.8% / -1.0%	-0.8% / -0.4%
Churn	0.2316 / 0.0718	+0.2% / +0.7%	+0.6% / +1.6%
Internet	0.2223 / 0.0993	+0.4% / +2.4%	+1.4% / +1.9%
Upselling	0.1679 / 0.0493	-0.7% / -0.4%	-1.0% / -0.2%
Kick	0.2955 / 0.0993	+0.1% / -0.4%	-0.3% / -0.2%
Average		-0.2% / +0.2%	-0.2% / +0.2%

Ordered Boosting vs. Other GBDT methods

- ✓ Compare each **GBDT methods + Ordered TS** with CatBoost (order)
- ✓ **CatBoost(order) showed best results for all cases**
- ✓ **Excellent results, especially on small datasets**

	CatBoost	LightGBM	XGBoost
Adult	**0.270 / 0.127**	+2.4% / +1.9%	+2.2% / +1.0%
Amazon	**0.139 / 0.044**	+17% / +21%	+17% / +21%
Click	**0.392 / 0.156**	+1.2% / +1.2%	+1.2% / +1.2%
Epsilon	**0.265 / 0.109**	+1.5% / +4.1%	+11% / +12%
Appetency	**0.072 / 0.018**	+0.4% / +0.2%	+0.4% / +0.7%
Churn	**0.232 / 0.072**	+0.1% / +0.6%	+0.5% / +1.6%
Internet	**0.209 / 0.094**	+6.8% / +8.6%	+7.9% / +8.0%
Upselling	**0.166 / 0.049**	+0.3% / +0.1%	+0.04% / +0.3%
Kick	**0.286 / 0.095**	+3.5% / +4.4%	+3.2% / +4.1%

Ordered Boosting vs. Other GBDT methods

- ✓ Use Epsilon dataset
- ✓ All models matched parameters
- ✓ Measure mean time to build 8000 trees

Table 6: Comparison of running times on Epsilon

	time per tree
CatBoost Plain	**1.1 s**
CatBoost Ordered	1.9 s
XGBoost	3.9 s
LightGBM	**1.1 s**

- ✓ Compare CatBoost modes
- ✓ "order" showed good results on average (especially on small datasets)

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%)

VI. Conclusion

✓ **CatBoost is One of major GBDT implementations released by Yandex**

✓ **Ordered TS**

- Calculate TS based on the order of randomly sorted data
- Prevent target leakage
- Ordered TS showed better results than Other TS methods

✓ **Ordered Boosting**

- Train a model based on the order of randomly sorted data
- Use different data set for building trees and calculating residuals
- Ordered Boosting showed good results on small datasets

Practical Algorithm of Ordered Boosting

Algorithm 3: CatBoost

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

$\sigma_r \leftarrow$ random permutation of $[1, n]$ for $r = 0..s$;
$M_0(i) \leftarrow 0$ for $i = 1..n$;
if $Mode = Plain$ **then**
    $M_r(i) \leftarrow 0$ for $r = 1..s$, $i : \sigma_r(i) \leq 2^{j+1}$;
if $Mode = Ordered$ **then**
    for $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
        $M_{r,j}(i) \leftarrow 0$ for $r = 1..s$, $i = 1..2^{j+1}$;
for $t \leftarrow 1$ **to** I **do**
    $T_t, \{M_r\}_{r=1}^s \leftarrow BuildTree(\{M_r\}_{r=1}^s, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, Mode)$;
    $leaf_0(i) \leftarrow GetLeaf(\mathbf{x}_i, T_t, \sigma_0)$ for $i = 1..n$;
    $grad_0 \leftarrow CalcGradient(L, M_0, y)$;
    foreach *leaf* j *in* T_t **do**
        $b_j^t \leftarrow -\mathrm{avg}(grad_0(i)$ for $i : \ leaf_0(i) = j)$;
    $M_0(i) \leftarrow M_0(i) + \alpha b_{leaf_0(i)}^t$ for $i = 1..n$;
return $F(\mathbf{x}) = \sum_{t=1}^I \sum_j \alpha\, b_j^t \mathbb{1}_{\{GetLeaf(\mathbf{x}, T_t, ApplyMode) = j\}}$;

$\sigma_r(i) = 1,\ldots 2^{j+1}$

Practical Algorithm of Ordered Boosting

Function *BuildTree*

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**
  $G \leftarrow (grad_r(i)$ for $i = 1..n)$;
if $Mode = Ordered$ **then**
  $G \leftarrow (grad_{r,\lfloor \log_2(\sigma_r(i)-1)\rfloor}(i)$ for $i = 1..n)$;
$T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do**
  foreach *candidate split* c **do**
    $T_c \leftarrow$ add split c to T;
    $leaf_r(i) \leftarrow GetLeaf(\mathbf{x}_i, T_c, \sigma_r)$ for $i = 1..n$;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \mathrm{avg}(grad_r(p)$ for $p: leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \mathrm{avg}(grad_{r,\lfloor \log_2(\sigma_r(i)-1)\rfloor}(p)$ for $p: leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
    $loss(T_c) \leftarrow \cos(\Delta, G)$
  $T \leftarrow \arg\min_{T_c}(loss(T_c))$
$leaf_{r'}(i) \leftarrow GetLeaf(\mathbf{x}_i, T, \sigma_{r'})$ for $r' = 1..s, i = 1..n$;
if $Mode = Plain$ **then**
  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha\,\mathrm{avg}(grad_{r'}(p)$ for $p: leaf_{r'}(p) = leaf_{r'}(i))$ for $r' = 1..s, i = 1..n$;
if $Mode = Ordered$ **then**
  for $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
    $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha\,\mathrm{avg}(grad_{r',j}(p)$ for $p: leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \le 2^j)$ for $r' = 1..s,\ i: \sigma_{r'}(i) \le 2^{j+1}$;
return T, M