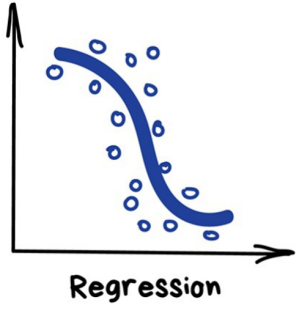
Modern Methods in Gradient Boosting. Theory and Practice

Leonid Iosipoi,
Sber AI Lab and HSE HDI Lab, Moscow

Pushkin,
June 24, 2022

Artificial Intellegence

what people think it is	what actually it is
	

Outline

1. Background on GBDT
2. Sampling Methods in GBDT
3. Fast Split Scoring for Multioutput Problems
4. Numerical Results

Background on GBDT

Background on GBDT

Gradient Boosting is one of the most powerful methods for solving prediction problems in both classification and regression domains.

It is a dominant tool today in application domains where tabular data is abundant, for example, in e-commerce, financial, and retail industries.

It has contributed countless top solutions in Kaggle competitions.

Background on GBDT

- Dataset $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^{n}$, where $x_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}^d$
- (x_k, y_k) are i.i.d. according to unknown $P(\cdot, \cdot)$
- $L(y, \hat{y})$ is a given loss function

Background on GBDT

- Dataset $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^{n}$, where $x_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}^d$
- (x_k, y_k) are i.i.d. according to unknown $P(\cdot, \cdot)$
- $L(y, \hat{y})$ is a given loss function

The goal is to construct a model $F(x)$ to minimize the aggregation of loss L,

$$\mathcal{L}_n(F) = \sum_{k=1}^{n} L\big(y_k, F(x_k)\big).$$

Background on GBDT

Gradient Boosting is an algorithm which combines weak learners into a single strong learner in an iterative and greedy fashion.

Background on GBDT

Gradient Boosting is an algorithm which combines weak learners into a single strong learner in an iterative and greedy fashion.

- Choose a set of weak learners $\mathcal{F} \subset \{f : \mathbb{R}^m \to \mathbb{R}^d\}$

Background on GBDT

Gradient Boosting is an algorithm which combines weak learners into a single strong learner in an iterative and greedy fashion.

- Choose a set of weak learners $\mathcal{F} \subset \{f : \mathbb{R}^m \to \mathbb{R}^d\}$
- At each step $t \in \mathbb{N}$
  1. Compute derivatives:

$$g_i^t = \nabla_a L(y_i, a)\big|_{a=F^{t-1}(x_i)} \quad \text{and} \quad H_i^t = \nabla_{aa}^2 L(y_i, x)\big|_{a=F^{t-1}(x_i)}.$$

Background on GBDT

Gradient Boosting is an algorithm which combines weak learners into a single strong learner in an iterative and greedy fashion.

- Choose a set of weak learners $\mathcal{F} \subset \{f : \mathbb{R}^m \to \mathbb{R}^d\}$
- At each step $t \in \mathbb{N}$
  1. Compute derivatives:

$$g_i^t = \nabla_a L(y_i, a)\big|_{a=F^{t-1}(x_i)} \quad \text{and} \quad H_i^t = \nabla^2_{aa} L(y_i, x)\big|_{a=F^{t-1}(x_i)}.$$

  2. Find an approximate minimizer $f^t \in \mathcal{F}$ using the Newton method:

$$f^t \approx \operatorname*{argmin}_{f \in \mathcal{F}} \left\{ \sum_{i=1}^{n} \left(\left(g_i^t\right)^\top f(x_i) + \frac{1}{2} \left(f(x_i)\right)^\top H_i^t f(x_i) \right) + \Omega(f) \right\},$$

     where $\Omega(f)$ is a regularization term.

Background on GBDT

Gradient Boosting is an algorithm which combines weak learners into a single strong learner in an iterative and greedy fashion.

- Choose a set of weak learners $\mathcal{F} \subset \{f : \mathbb{R}^m \to \mathbb{R}^d\}$
- At each step $t \in \mathbb{N}$
  1. Compute derivatives:

$$g_i^t = \nabla_a L(y_i, a)\big|_{a=F^{t-1}(x_i)} \quad \text{and} \quad H_i^t = \nabla^2_{aa} L(y_i, x)\big|_{a=F^{t-1}(x_i)}.$$

  2. Find an approximate minimizer $f^t \in \mathcal{F}$ using the Newton method:

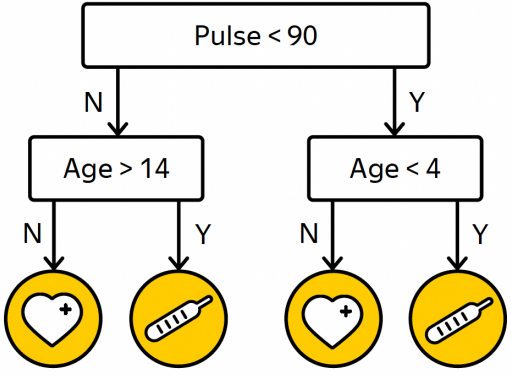
$$f^t \approx \operatorname*{argmin}_{f \in \mathcal{F}} \left\{ \sum_{i=1}^{n} \left(\left(g_i^t\right)^\top f(x_i) + \frac{1}{2} \left(f(x_i)\right)^\top H_i^t f(x_i) \right) + \Omega(f) \right\},$$

     where $\Omega(f)$ is a regularization term.

  3. Update the model: $F^{t+1} = F^t + \varepsilon f^t$ ($\varepsilon > 0$ is a learning rate).

Background on GBDT

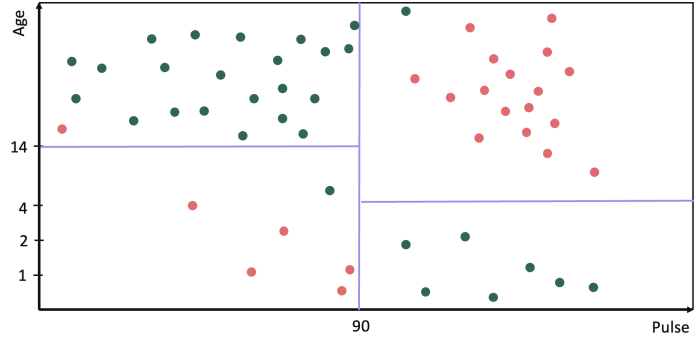
Gradient Boosted Decision Tree (GBDT) uses decision trees as weak learners.



Based on the construction mechanism, any decision tree can be expressed as

$$f(x) = \sum_{j=1}^{J} v_j \cdot [x \in R_j],$$

where J is the number of leaves, R_j is a j-th leaf, v_j is the value of j-th leaf. Here [predicate] denotes the indicator function.



Background on GBDT

The problem of learning f^t can be divided into two separate problems:

1. Finding the tree structure — division of a feature space into J leaves.
2. Finding the leaf values — finding leaf values which minimize the loss function for a tree with a given structure.

Fitting a decision tree

Since decision trees take constant values at each leaf, we can optimize the objective function from for each leaf R_j separately,

$$v_j = \underset{v \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \sum_{x_i \in R_j} \left(g_i^\top v + \frac{1}{2} v^\top H_i v \right) + \frac{\lambda}{2} \|v\|^2 \right\},$$

where we employ l_2 regularization on leaf values with $\lambda > 0$.

(we don't indicate the dependence of J, v_j, R_j, g_i, and H_i on the step t)

Background on GBDT

If the loss function L is separable w.r.t. different outputs, all Hessians $H_1, \ldots, H_n$ are diagonal. If it is not the case, it is a common practice to purposely simplify them to this extent in order to avoid matrix inversion.

For diagonal Hessians, the optimal leaf values are given by

$$v_j = -\frac{\sum_{i \in R} g_i^j}{\sum_{i \in R} h_i^j + \lambda}, \quad \text{where } g_i = \begin{pmatrix} g_i^1 \\ \vdots \\ g_i^d \end{pmatrix} \text{ and } H_i = \begin{pmatrix} h_i^1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & h_i^d \end{pmatrix}.$$

Substituting these leaf values back into the objective function, and omitting insignificant terms, we obtain

$$\text{Loss}(f_t) = -\frac{1}{2}\sum_{j=1}^{J} S(R_j), \quad \text{where} \quad S(R) = \sum_{j=1}^{d} \frac{\left(\sum_{x_i \in R} g_i^j\right)^2}{\sum_{x_i \in R} h_i^j + \lambda}.$$

The function $S(\cdot)$ will be referred to as the scoring function.

Finding the tree structure

- Commonly, a greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used.
- At a general step, to split one of existing leaves, we iterate through all leaves, features, and thresholds for each feature.

Finding the tree structure

- Commonly, a greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used.
- At a general step, to split one of existing leaves, we iterate through all leaves, features, and thresholds for each feature.
- The split of leaf R is based on a feature and threshold for this feature $R_{\text{left}} = \{x_i \in R \mid x_i^j \leq \text{threshold}\}$ and $R_{\text{right}} = \{x_i \in R \mid x_i^j > \text{threshold}\}$. To evaluate split candidates, we maximize the impurity score given by

$$S(R_{\text{left}}) + S(R_{\text{right}}).$$

This is equivalent to maximization of the information gain

$$\text{Gain} = -\frac{1}{2}\Big(S(R) - \big(S(R_{\text{left}}) + S(R_{\text{right}})\big)\Big).$$

Background on GBDT

Similar to the previous step, some simplifications to the scoring function are made to speed up its computation done a tremendous number of times.

For instance, in CatBoost, the second-order derivatives are totally ignored during the split search and are used only to compute leaf values.

We will develop this idea further in our work.

Sampling Methods in GBDT

Sampling in GBDT

A key problem

Weak learners f^t are highly correlated since they are trained on the same dataset. This leads to a limited generalization ability of GBDT.

A common approach is to use random subsampling of the training data at each boosting iteration.

Sampling in GBDT

Stochastic Gradient Boosting (SGB)

- A randomized version of gradient boosting algorithm proposed by Friedman (Computational Statistics & Data Analysis 38(4), 2002).
- At each iteration, random fraction s of objects is used to fit the model
- SGB selects random $s \cdot n$ objects uniformly and without replacement
- SGB improves the ensemble quality and reduces its training complexity

Sampling in GBDT

Gradient-based one-side sampling (GOSS)

- GOSS keeps the data instances with large gradients and randomly drops the instances with small gradients (LightGBM paper, NeurIPS, 2017).
- GOSS samples:
  - αn objects with largest absolute gradients with probability 1
  - $s(1-\alpha)n$ other objects at random
- For unbiased estimation, GOSS uses weights:
  - αn samples with largest gradients are used with weight 1
  - other samples are used with weight $(1-\alpha)/s$

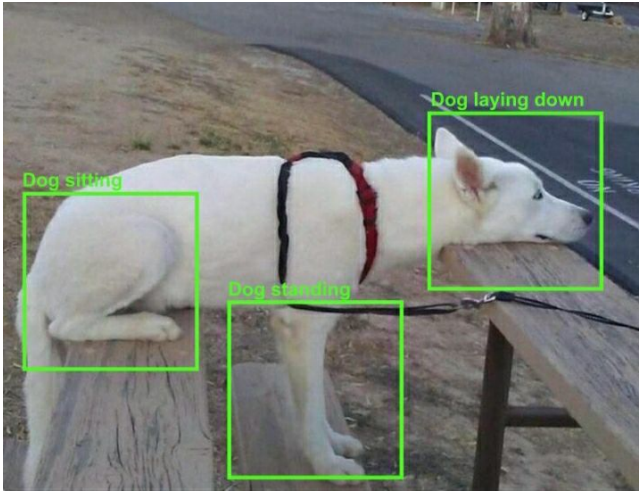
Sampling in GBDT

Minimal Variance Sampling (MVS)

- A nearly optimal non-uniform sampling method for sample instances was proposed by Bulat Ibragimov and Gleb Gusev (NeurIPS, 2019).
- MVS randomly chooses objects to maximize the estimation accuracy of split scoring used to train decision trees at each iteration.

Fast Split Scoring

Fast Split Scoring



Fast Split Scoring

Our main focus will be the scalability of GBDT to multioutput problems:

- multiclass classification
- multilabel classification
- multioutput regression

These problems arise in various areas such as Finance, Multivariate Time Series Forecasting, Recommender Systems, and others.

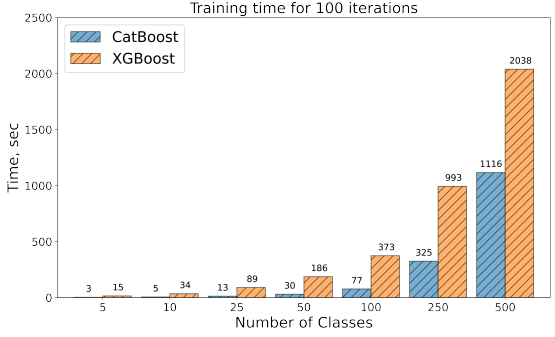
Fast Split Scoring

There are several extremely efficient, open-source, and production-ready implementations of GBDT such as

- XGBoost
- LightGBM
- CatBoost

Even for them, fitting a GBDT model for moderately large datasets with high-dimensional output can easily encounter near-forever running time.

Motivation



XGBoost and CatBoost training time on a synthetic dataset (2000k instances, 100 features) for multiclass classification.

Motivation

There are two possible strategies on how to handle a multioutput problem.

Motivation

There are two possible strategies on how to handle a multioutput problem.

- One-versus-all strategy assumes that each output is handled separately, (at each boosting step, a single decision tree is built for every output). Examples: XGBoost and LightGBM.

Motivation

There are two possible strategies on how to handle a multioutput problem.

- One-versus-all strategy assumes that each output is handled separately, (at each boosting step, a single decision tree is built for every output). Examples: XGBoost and LightGBM.
- Single-tree strategy assumes that all outputs are handled together (at each boosting step, a single multivariate tree is built for all outputs). Example: CatBoost.

Fast Split Scoring

The computational complexity of both strategies is proportional to the number of outputs. Specifically,

- One-versus-all strategy requires fitting a separate decision tree for each single output at each boosting step.
- Single-tree strategy requires scanning all the output dimensions
  (a) to estimate the information gain during the split search
  (b) to compute leaf values of a decision tree with a given structure

Fast Split Scoring

Key idea: To exclude some of the output dimensions during the split search (the most time-consuming step) for single-tree GBDT.

The methods we are going to discuss are very similar to SGB, GOSS, MVS with the only difference that they are applied to output dimensions.

Fast Split Scoring

For split search, we simplify the scoring function to

$$S_G(R) = \frac{\left\|G^\top v_R\right\|^2}{|R| + \lambda},$$

where

$$G = \begin{pmatrix} g_1^1 & \dots & g_1^d \\ \vdots & \ddots & \vdots \\ g_n^1 & \dots & g_n^d \end{pmatrix} \quad \text{and} \quad v_R = \begin{pmatrix} [x_1 \in R] \\ \vdots \\ [x_n \in R] \end{pmatrix}.$$

Fast Split Scoring

Sketching

To reduce the complexity of computing $S_G(R)$ in d, we will approximate $S_G(R)$ with $S_{G_k}(R)$ for some other matrix $G_k \in \mathbb{R}^{n \times k}$ with $k \ll d$.

We will refer to G_k as the sketch matrix.

Since there might be several good splits with almost equal impurity scores, replacing S_G with S_{G_k} might result in completely different tree structures.

Fast Split Scoring

We want the proposed approximation to be universal and uniformly accurate for all splits we will possibly iterate over. Our approximation error is given by

$$\text{Error}(S_G, S_{G_k}) = \sup_R \left| S_G(R) - S_{G_k}(R) \right|.$$

Given the two matrices G and G_k, this optimization problem is an instance of Integer Programming problem and hence is NP-complete.

Since the brute force is not an option in our case, we will replace this problem with a relaxed one and will look for nearly-optimal solutions.

Fast Split Scoring

We show that reasonably good upper bounds on the error are obtained when $GG^\top$ is well approximated with $G_k G_k^\top$ in the operator norm.

This observation reduces our problem to Approximate Matrix Multiplication (AMM) which is extensively studied.

Fast Split Scoring

Truncated SVD

Key Idea: To replace the gradient matrix G with its Truncated SVD version.

We start with Truncated SVD since, by the matrix approximation lemma, it provides the optimal deterministic solution to AMM.

Fast Split Scoring

Proposition. Let $G \in \mathbb{R}^{n \times d}$ be any matrix. Let also $G_k \in \mathbb{R}^{n \times k}$ be the best k-rank approximation of G provided by the Truncated SVD. Then

$$\mathrm{Error}(S_G, S_{G_k}) \leq \sigma_{k+1}^2(G),$$

where $\sigma_{k+1}^2(G)$ is $(k+1)$ largest singular value of G.

Top Outputs

Key Idea: To choose k columns of G with the largest Euclidian norm.

Specifically, let us denote the columns of G by $g_1, \ldots, g_d$ and let $i_1, \ldots, i_d$ be the indexes which sort the columns of G in descending order by their norm, $\|g_{i_1}\| \geq \|g_{i_2}\| \geq \ldots \geq \|g_{i_d}\|$. We consider the following sketch

$$G = \begin{pmatrix} | & | & & | \\ g_1 & g_2 & \ldots & g_d \\ | & | & & | \end{pmatrix}, \quad G_k = \begin{pmatrix} | & | & & | \\ g_{i_1} & g_{i_2} & \ldots & g_{i_k} \\ | & | & & | \end{pmatrix}.$$

Fast Split Scoring

Proposition. Let $G \in \mathbb{R}^{n \times d}$ be any matrix. Let also $G_k \in \mathbb{R}^{n \times k}$ be the sketch of G given by Top Outputs. Then

$$\mathrm{Error}(S_G, S_{G_k}) \leq \sum_{j=k+1}^{d} \|g_{i_j}\|^2.$$

Fast Split Scoring

Random Sampling

Key Idea: To sample k columns of G with optimal (non-uniform) probabilities.

Namely, we define the non-uniform sampling probabilities by

$$p_i = \frac{\|g_i\|^2}{\sum_{j=1}^{d} \|g_j\|^2}, \quad i = 1, \ldots, d.$$

Fast Split Scoring

Now we consider the following sketch

$$G_k = \begin{pmatrix} | & | & & | \\ \widehat{g}_1 & \widehat{g}_2 & \dots & \widehat{g}_k \\ | & | & & | \end{pmatrix},$$

where the columns $\widehat{g}_1, \dots, \widehat{g}_k$ are independent copies of the random vector $\widehat{g}$ such that

$$\widehat{g} = \frac{1}{\sqrt{k p_i}} g_i \quad \text{with probability } p_i.$$

Fast Split Scoring

Proposition. Let $G \in \mathbb{R}^{n \times d}$ be any matrix. Let also $G_k \in \mathbb{R}^{n \times k}$ be a sketch given by Random Sampling. Then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$,

$$\mathrm{Error}(S_G, S_{G_k}) \leq C_{G,\delta} \frac{\|G\|^2}{\sqrt{k}},$$

where $C_{G,\delta}$ is a constant depending on G and δ and is given by

$$C_{G,\delta} = 2\sqrt{\mathrm{sr}(G) \log\left(\frac{4\,\mathrm{sr}(G)}{\delta}\right)}.$$

Fast Split Scoring

Random Projections

Key Idea: To sample k random linear combinations of columns of G.

Previously, the sketch G_k was formed by sampling columns from G according to some probability distribution. This process can be viewed as multiplication of G by a matrix Π,

$$G_k = G\Pi,$$

where $\Pi \in \mathbb{R}^{d \times k}$ has independent columns, and each column is all zero except for a 1 in a random location.

Fast Split Scoring

Now we consider sampling matrices Π, every entry of which is an independently sampled random variable. Namely, we consider the sketch

$$G_k = G\Pi,$$

where $\Pi \in \mathbb{R}^{d \times k}$ is a random matrix filled with independent $\mathcal{N}(0, k^{-1})$ entries.

This approach is based on the Johnson-Lindenstrauss lemma. In fact, this lemma is true for many other distributions, but there was no significant difference between them in our numerical experiments.

Fast Split Scoring

Proposition. Let $G \in \mathbb{R}^{n \times d}$ be any matrix. Let also $\Pi \in \mathbb{R}^{d \times k}$ be a random matrix filled with independently sampled $\mathcal{N}(0, k^{-1})$ entries. Set $G_k = G\Pi$. Then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$,

$$\mathrm{Error}(S_G, S_{G_k}) \leq C_{G,\delta} \frac{\|G\|^2}{\sqrt{k}},$$

where $C_{G,\delta}$ is a constant depending on G and δ and is given by

$$C_{G,\delta} = c\sqrt{\mathrm{sr}(G) + \ln\left(\frac{1}{\delta}\right)}.$$

for some absolute constant $c > 0$.

Numerical Results

Numerical Results

The goal of out numerical study is to compare

- proposed methods for fast split scoring in GBDT
- existing state-of-art boosting toolkits supporting multioutput tasks

We implemented a version of the GBDT algorithm in Python – SketchBoost. It follows the classic scheme as XGBoost does, works only on GPU, and uses Python GPU libraries such as CuPy and Numba.

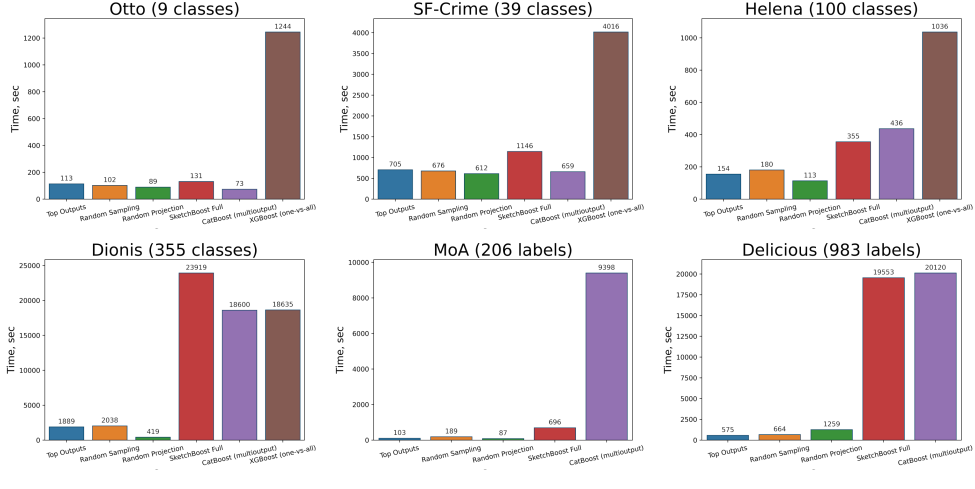
Numerical Results

The experiments were conducted on datasets from Kaggle, OpenML, and Mulan website for multiclass/multilabel classification and multitask regression.

Dataset	Task	Rows	Features	Classes/Labels/Targets
Otto	multiclass	61 878	93	9
SF-Crime	multiclass	878 049	10	39
Helena	multiclass	65 196	27	100
Dionis	multiclass	416 188	60	355
Mediamill	multilabel	43 910	120	101
MoA	multilabel	23 814	876	206
Delicious	multilabel	16 110	500	983
RF1	multitask	9125	64	8
SCM20D	multitask	8966	61	16

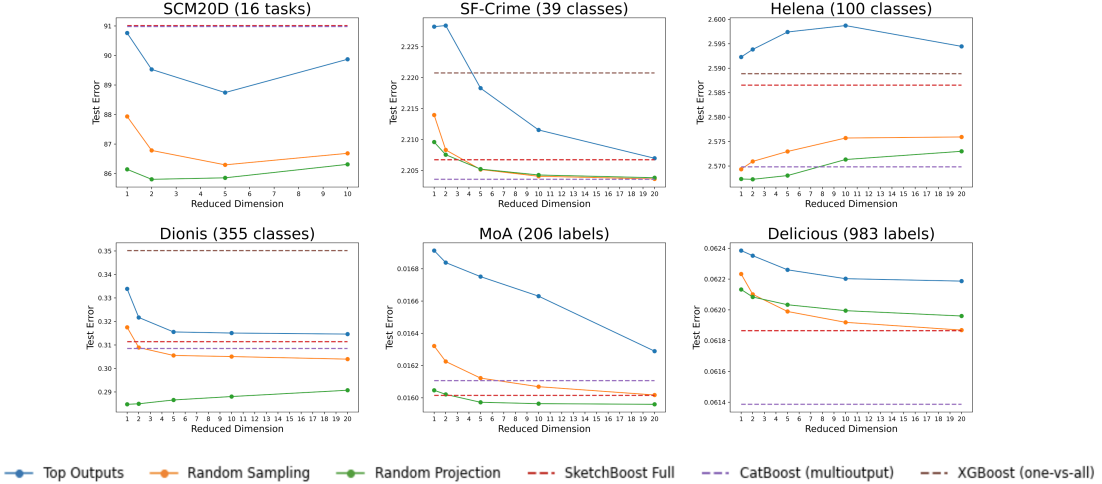
Numerical Results

Training time per fold in seconds.



Numerical Results

Dependence of test errors on sketch size $k \in \{1, 2, 5, 10, 20\}$.



Numerical Results

Test errors (cross-entropy for classification and RMSE for regression) ± their STD.
(XGBoost supports only multiclass classification tasks and hence has missing values.)

Dataset	SketchBoost				Baselines	
	Top Outputs (for the best k)	**Random Sampling** (for the best k)	**Random Projection** (for the best k)	**SketchBoost Full** (multioutput)	**CatBoost** (multioutput)	**XGBoost** (one-vs-all)
Multiclass classification						
Otto (9 classes)	0.4715±0.0035	0.4636±0.0026	**0.4566**±0.0023	0.4697±0.0030	0.4658±0.0033	0.4599±0.0028
SF-Crime (39 classes)	2.2070±0.0005	2.2037±0.0004	2.2038±0.0004	2.2067±0.0003	**2.2036**±0.0005	2.2208±0.0008
Helena (100 classes)	2.5923±0.0024	2.5693±0.0022	**2.5673**±0.0026	2.5865±0.0025	2.5698±0.0025	2.5889±0.0032
Dionis (355 classes)	0.3146±0.0011	0.3040±0.0014	**0.2848**±0.0012	0.3114±0.0009	0.3085±0.0010	0.3502±0.0020
Multilabel classification						
Mediamill (101 labels)	0.0745±1.3e-05	0.0745±1.3e-05	**0.0743**±1.1e-05	0.0747±1.3e-05	0.0754±1.1e-05	–
MoA (206 labels)	0.0163±2.2e-05	0.0160±1e-05	**0.0160**±6e-06	0.0160±9e-06	0.0161±2.6e-05	–
Delicious (983 labels)	0.0622±6.2e-05	0.0619±5.9e-05	0.0620±6.2e-05	0.0619±5.5e-05	**0.0614**±5.2e-05	–
Multitask regression						
RF1 (8 tasks)	1.1860±0.1366	0.9944±0.1015	0.9056±0.0582	1.1687±0.0835	**0.8975**±0.0384	–
SCM20D (16 tasks)	88.7442±0.6346	86.2964±0.4398	**85.8061**±0.5534	91.0142±0.3397	90.9814±0.3652	–

Numerical Results

Training time per fold in seconds.
(XGBoost supports only multiclass classification tasks and hence has missing values.)

Dataset	SketchBoost (GPU)				Baseline	
	Top Outputs (for the best k)	**Random Sampling** (for the best k)	**Random Projection** (for the best k)	**SketchBoost Full** (multioutput)	**CatBoost** (multioutput)	**XGBoost** (one-vs-all)
Multiclass classification					**GPU**	**GPU**
Otto (9 classes)	113	102	89	131	**73**	1244
SF-Crime (39 classes)	705	676	**612**	1146	659	4016
Helena (100 classes)	154	180	**113**	355	436	1036
Dionis (355 classes)	1889	2038	**419**	23919	18600	18635
Multilabel classification					**CPU**	
Mediamill (101 labels)	**251**	263	294	1777	10164	–
MoA (206 labels)	103	189	**87**	696	9398	–
Delicious (983 labels)	**575**	664	1259	19553	20120	–
Multitask regression					**CPU**	
RF1 (8 tasks)	369	396	**340**	413	804	–
SCM20D (16 tasks)	499	528	**479**	597	798	–

Conclusions

- Four different methods to accelerate the training time of GBDT on multioutput tasks. They are generic and can be easily integrated into any GBDT realization that uses the single-tree strategy.
- Empirical study shows that these methods achieve comparable or even better performance compared to the existing state-of-the-art boosting toolkits but in remarkably less time.

Thank you for listening!