High-Order Optimization of Gradient Boosted Decision Trees

Jean Pachebat JPACHEBAT@GMAIL.COM
Sergey Ivanov IVANOVSERG990@GMAIL.COM
Paris, France

Abstract

Gradient Boosted Decision Trees (GBDTs) are dominant machine learning algorithms for modeling discrete or tabular data. Unlike neural networks with millions of trainable parameters, GBDTs optimize loss function in an additive manner and have a single trainable parameter per leaf, which makes it easy to apply high-order optimization of the loss function. In this paper, we introduce high-order optimization for GBDTs based on numerical optimization theory which allows us to construct trees based on high-order derivatives of a given loss function. In the experiments, we show that high-order optimization has faster per-iteration convergence that leads to reduced running time. Our solution can be easily parallelized and run on GPUs with little overhead on the code. Finally, we discuss future potential improvements such as automatic differentiation of arbitrary loss function and combination of GBDTs with neural networks.

1. Introduction

Gradient boosted decision trees (GBDT) [10] are state-of-the-art ML models for tabular datasets. Many variants of GBDTs have been proposed in the recent years that achieve top performance in classification [15, 24], regression [7, 18], and ranking [14, 27] tasks with applications where data often contain missing or noisy features, complex relationships, and heterogeneity such as in recommender systems, information retrieval, and many others [8, 17, 20, 23, 28].

Let's consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|i \in \{1, ..., n\}\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$. A GBDT model is a sum of K additive functions, each of which represents a decision tree:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \tag{1}$$

where $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is a space of regression trees. Each regression tree f_k maps m-dimensional feature vector $\mathbf{x}$ to one of its leaves (or regions) with index $q(\mathbf{x})$ that has a corresponding weight value $w_{q(\mathbf{x})}$.

One of the most important differences between GBDT models and other tree-based models such as Random Forest [5] or Extremely Randomized Trees [12] is that GBDT model learns the trees in an iterative fashion by adding a new tree with respect to the performance of previous trees, while the latter models grow trees independently from each other. In particular, in GBDT the loss objective $\mathcal{L}^{(t)}$ at every iteration t and regularization term Ω is minimized by adding a new tree f_t:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) \tag{2}$$

The regularization term Ω often penalizes the complexity of the tree function and we consider it to be $\Omega(f_t) = \frac{1}{2}\lambda\|w\|^2$, where λ is a hyperparameter.

In the seminal paper [11] Friedman proposed to learn new trees such that they produce the steepest-descent step direction given by the first-order gradient of the loss function with respect to the last prediction of the model:

$$f_t = -\partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{3}$$

Thus, each new tree gets an updated version of the data $\{(\mathbf{x}_i, \tilde{y}_i)\,|i \in \{1, ..., n\}\}$, where $\tilde{y}_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ is a "pseudoresponse" that the tree is fitting.

More than a decade after Chen and Guestrin proposed XGBoost [9] that popularized second-order optimization of gradient boosting. Instead of fitting the first-order gradient, each new tree decomposes the loss function using second-order approximation and derives the closed-form formulas for assigning the leaf weights and building a regression tree.

Let $g_i^{(k)}$ be the k-th order derivative of the loss function with respect to the last model's prediction $\hat{y}^{(t-1)}$, i.e. $g_i^{(k)} = \partial^k_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$. Then the Eq. 2 can be approximated by second-order Taylor expansion:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i^{(1)} f_t(\mathbf{x}_i) + \frac{1}{2} g_i^{(2)} f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{4}$$

Let's first assume that the tree structure is given to us and we want to find the leaf weights that minimize the Eq. 4. Let $I_j = \{i|q(\mathbf{x}_i) = j\}$ be a set of all data instances that fall into the leaf j. Then, removing the term that does not depend on the tree f_t and regrouping the data for each leaf we obtain:

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i^{(1)} f_t(\mathbf{x}_i) + \frac{1}{2} g_i^{(2)} f_t^2(\mathbf{x}_i)] + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i^{(1)}) w_j + \frac{1}{2}(\sum_{i \in I_j} g_i^{(2)} + \lambda) w_j^2] \end{aligned} \tag{5}$$

For each leaf j let's denote $G_j^{(k)} = \sum_{i \in I_j} g_i^{(k)}$ be the sum of k-th order gradients of data instances that fall into that leaf. Then the minimum of Eq. 5 for each leaf j is given by

$$w_j^* = -\frac{G_j^{(1)}}{G_j^{(2)} + \lambda}, \tag{6}$$

and by plugging it back to Eq.5 we get the optimal loss value:

$$\mathcal{L}_j^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{[G_j^{(1)}]^2}{G_j^{(2)} + \lambda}. \tag{7}$$

To date, the power of second-order GBDT models [9, 19, 22, 25, 26] has been demonstrated across a range of tasks and baselines including modern neural networks [3, 13]. However, to the best

of our knowledge, there are no works that study high-order gradient information of the loss function during optimization. In this work, we consider a high-order Taylor expansion of the loss function (Eq. 2) and derive closed-form formulas for arbitrary order gradient statistics and compare the results of these higher-order methods in the experiments.

2. High-Order Optimization of GBDT

We start by noting that XGboost decomposition of the loss function Eq. 4 provides two advantages. First, it allows us to derive optimal leaf weights for arbitrary differentiable loss functions. Second, the closed-form Eq. 6 provides a greedy algorithm to construct a tree. However, the second-order Taylor approximation may lead to inaccurate estimation of the loss function and therefore longer convergence. Next, we show an example of the optimal leaf weights that use third-order approximation, which is followed by the general k-th order methods.

2.1. Cubic Optimization of GBDT

Let's start with a third-order Taylor expansion of the loss Eq. 2 (after removing constant terms):

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T} [G_j^{(1)} w_j + \frac{1}{2}(G_j^{(2)} + \lambda) w_j^2 + \frac{1}{6} G_j^{(3)} w_j^3] \tag{8}$$

Here $G_j^{(k)} = \sum_{i \in I_j} g_i^{(k)}$ be the sum of k-th order gradients of data instances in the leaf j. By equating the derivative to zero, we can find the optimal weights w_j for each leaf j:

$$w_j^* = -\frac{G_j^{(2)} + \lambda}{G_j^{(3)}} \left(1 \pm \sqrt{1 - \frac{2 G_j^{(1)} G_j^{(3)}}{(G_j^{(2)} + \lambda)^2}} \right) \tag{9}$$

Note that at the minimum the first-order gradients $G_j^{(1)}$ are zero and therefore for a small enough $\sum_{i \in I_j} g_i^{(1)}$ we can use an expansion $1 - \sqrt{1 - \alpha} = \frac{\alpha}{2} + \frac{\alpha^2}{8} + O(\alpha^3)$ to simplify the terms of this equation . The optimal weights become:

$$w_j^* = -\frac{G_j^{(1)}}{G_j^{(2)} + \lambda} \left(1 + \frac{G_j^{(1)} G_j^{(3)}}{2(G_j^{(2)} + \lambda)^2} \right) \tag{10}$$

By plugging the weights back into the Eq. 8 we can compute the optimal loss value:

$$\mathcal{L}_j^* = -\frac{[G_j^{(1)}]^2}{G_j^{(2)} + \lambda} (\frac{1}{2} + \frac{1}{6}\varepsilon + 2\varepsilon^3 + \frac{2}{3}\varepsilon^4) \tag{11}$$

Here, $\varepsilon = \dfrac{G_j^{(1)} G_j^{(3)}}{G_j^{(2)} + \lambda}$. By comparing it to the Eq. 7 we can observe additional terms in Eq. 11 which correct the loss value estimation. Similarly, to XGBoost approach we can design an efficient

greedy algorithm (Alg. 1) that estimate the goodness of each split by computing the loss reduction score $\mathcal{L}_{split} = \mathcal{L}_{node} - (\mathcal{L}_{left} + \mathcal{L}_{right})$ and then selecting the split with the maximum score. To make it efficient, the algorithm first sorts the data on that node based on the feature values and then computes the loss based on the Eq. 11. Similar algorithms can be designed for arbitrary order k.

2.2. Householder Optimization of GBDT

Recap that the GBDT model is an additive process of adding new trees, each of which approximates some function of the loss objective $\phi(\mathbf{x}) \leftarrow \phi(\mathbf{x}) + f_t(\mathbf{x})$. In the case of Friedman's first-order method $f_t(\mathbf{x})$ approximates the gradient Eq. 3. In the second and third-order methods, each tree $f_t(\mathbf{x})$ approximates functions that involve higher-order gradient statistics, given by Eqs. 6 and 9. Furthermore, under some conditions of the regularity of the loss function, Householder [16] gave the general formula for arbitrary high order:

$$\phi(\mathbf{x}) \leftarrow \phi(\mathbf{x}) + (p+1)\left(\frac{(1/g)^{(p)}}{(1/g)^{(p+1)}}\right)_{x_n}, \tag{12}$$

where g is the gradient of the loss function and $(1/g)^{(p)}$ is the derivative of order p of inverse of g. The convergence has an order $(p+2)$. For example, for $p=0$ Eq. 12 results in Newton-Raphson update of XGBoost (Eq. 6). For $p=1$, Householder equation gives an update of order 3, also known as Halley's method [4]:

$$w_j^* = -\frac{G_j^{(1)}}{G_j^{(2)}+\lambda}\left(1 - \frac{G_j^{(1)}G_j^{(3)}}{2(G_j^{(2)}+\lambda)^2}\right)^{-1} \tag{13}$$

Given the approximation $(1-\alpha)^{-1} = 1 + \alpha + O(\alpha^2)$ for small α, we can recover third-order update in Eq. 9. Finally, for $p=2$ we obtain the following fourth-order update:

$$w_j^* = -G_j^{(1)}\left(\frac{(G_j^{(2)}+\lambda)^2 - G_j^{(1)}G_j^{(3)}/2}{(G_j^{(2)}+\lambda)^3 - G_j^{(1)}(G_j^{(2)}+\lambda)G_j^{(3)} + G_j^{(1)}G_j^{(4)}/6}\right). \tag{14}$$

3. Experiments

In the experiments, we use Eqs. 6, 13, 14 for the second, third, and fourth-order models respectively. Our implementation is based on the open-source package Py-Boost [1]. We used four binary classification datasets [13] for which we optimize binary cross-entropy loss. For each dataset, we use the validation part to select the best hyperparameters and use the test part to measure the accuracy. To compare models' efficiency we first trained the second-order model (GBDT-2) for 10000 trees and then recorded the best test accuracy it achieves. Then for each model, we measured the running time it takes to reach 99% of the best accuracy so that the small perturbations in the loss value can be discarded. The training was performed on GPU, with 120GB memory, 7.8Gbps, and 2 GPUs. We kept the maximum depth of 6 for trees. In contrary to second order, higher order GBDTs are

1. https://github.com/jpachebat/py_boost

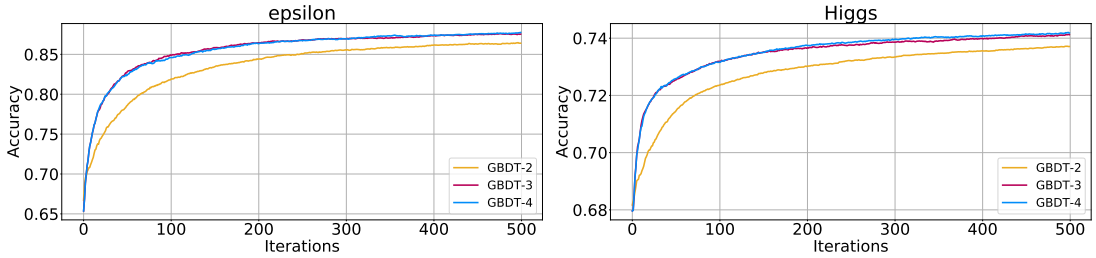

Figure 1: Test accuracy for high-order GBDT models.

more sensitive to regularization factor λ, which we selected using hyperparameter search in the range $[10^0, \ldots, 10^6]$.

Convergence of validation accuracy is presented in the Fig. 1. We can see that the third (GBDT-3) and fourth (GBDT-4) methods have much faster convergence initially. However, as we train longer, the accuracy of GBDT-2 becomes comparable to those of the higher-order methods (see Appendix B). On the other hand, the running time to reach 99% optimal accuracy for higher-order is significantly smaller as shown in Table 1. For example, on Epsilon dataset the relative difference (Gap) between GBDT-2 model and GBDT-3 model is 73%.

Table 1: Running time to achieve 99% of the optimal test accuracy for high-order GBDT models. Gap % is the relative difference w.r.t. GBDT-2 accuracy (the smaller, the better).

		Higgs		**Epsilon**		Covertype		MiniBooNE	
	Dataset	Time (s.)	Gap %	Time (s.)	Gap %	Time (s.)	Gap %	Time (s.)	Gap %
	GBDT-2	8.98 ± 0.01	0	94.80 ± 0.01	0	32.08 ± 0.01	0	1.35 ± 0.01	0
Ours	GBDT-3	4.93 ± 0.00	-45	25.00 ± 0.01	-73	18.46 ± 0.00	-42	0.59 ± 0.01	-56
Ours	GBDT-4	4.7 ± 0.1	-47	29.17 ± 0.01	-69	23.09 ± 0.00	-28	0.48 ± 0.01	-64

4. Conclusion

In this work, we proposed a high-order optimization framework to learn GBDT model, which has not been explored in the context of gradient boosting and may lead to many improvements to the existing GBDT algorithms: faster convergence, automatic regularization of the step size, and better optima. There are many exciting future directions for this research. Our framework is developed for arbitrary differentiable loss objectives; however, the user still has to provide manually-derived gradients in order to compute the optimal weights. Recent interest in automatic and symbolic differentiation [2] can come to the rescue, especially in the case when the loss objective is highly non-linear and the optimization order p is high. Note, however, that automatic differentiation additionally increases the running time so there is a trade-off between the efficiency of higher-order optimization and its versatility. Going one step further, we can implement a GBDT model directly in popular frameworks such as PyTorch [21] or TensorFlow [1] and leverage automatic differentiation, GPU acceleration, distributed training, and many other features (see [25] for an example). Finally, training GBDT model with neural networks in an end-to-end fashion has recently attracted attention [6, 18] and is worth studying in the context of high-order optimization.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[3] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

[4] John P Boyd. Finding the zeros of a univariate equation: proxy rootfinders, chebyshev interpolation, and the companion matrix. *SIAM review*, 55(2):375–396, 2013.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] Jiuhai Chen, Jonas Mueller, Vassilis N Ioannidis, Soji Adeshina, Yangkun Wang, Tom Goldstein, and David Wipf. Convergent boosted smoothing for modeling graph data with tabular node features. *arXiv preprint arXiv:2110.13413*, 2021.

[7] Jiuhai Chen, Jonas Mueller, Vassilis N. Ioannidis, Soji Adeshina, Yangkun Wang, Tom Goldstein, and David Wipf. Does your graph need a confidence boost? convergent boosted smoothing on graphs with tabular node features. In *International Conference on Learning Representations*, 2022.

[8] Mingcheng Chen, Zhenghui Wang, Zhiyun Zhao, Weinan Zhang, Xiawei Guo, Jian Shen, Yanru Qu, Jieli Lu, Min Xu, Yu Xu, et al. Task-wise split gradient boosting trees for multi-center diabetes prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2663–2673, 2021.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[11] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[12] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[13] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022.

[14] Andrey Gulin, Igor Kuralenok, and Dimitry Pavlov. Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank. Proceedings of Machine Learning Research. PMLR, 25 Jun 2011.

[15] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*, pages 1–9, 2014.

[16] A.S. Householder. *The Numerical Treatment of a Single Nonlinear Equation*. International series in pure and applied mathematics. McGraw-Hill, 1970.

[17] Md Zahidul Islam, Jixue Liu, Jiuyong Li, Lin Liu, and Wei Kang. Evidence weighted tree ensembles for text classification. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1737–1740, 2020.

[18] Sergei Ivanov and Liudmila Prokhorenkova. Boost then convolve: Gradient boosting meets graph neural networks. *arXiv preprint arXiv:2101.08543*, 2021.

[19] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[20] Cagri Ozcaglar, Sahin Geyik, Brian Schmitz, Prakhar Sharma, Alex Shelkovnykov, Yiming Ma, and Erik Buchanan. Entity personalized talent search models with tree interaction features. In *The World Wide Web Conference*, pages 3116–3122, 2019.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[22] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

[23] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. Are neural rankers still outperformed by gradient boosted decision trees? In *International Conference on Learning Representations*, 2021.

[24] Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

[25] Olivier Sprangers, Sebastian Schelter, and Maarten de Rijke. Probabilistic gradient boosting machines for large-scale probabilistic regression. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 1510–1520, 2021.

[26] Aleksei Ustimenko and Liudmila Prokhorenkova. Sglb: Stochastic gradient langevin boosting. In *International Conference on Machine Learning*, pages 10487–10496. PMLR, 2021.

[27] Aleksei Ustimenko, Aleksandr Vorobev, Gleb Gusev, and Pavel Serdyukov. Learning to select for a predefined ranking. In *International Conference on Machine Learning*, pages 6477–6486. PMLR, 2019.

[28] Yanru Zhang and Ali Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015.

Appendix A. Third-Order Split Selection Algorithm

Algorithm 1 Selecting optimal split for a tree node

Input: I, instances of the node

$score \leftarrow 0$

$G^{(i)} \leftarrow \sum_{i \in I} g_i^{(1)}$, $G^{(2)} \leftarrow \sum_{i \in I} g_i^{(2)}$, $G^{(3)} \leftarrow \sum_{i \in I} g_i^{(3)}$

for $k = 1$ ***to*** m **do**

  $G_{left}^{(i)} \leftarrow 0$ for $i = 1, 2, 3$

  for j *in sorted(*I*, by* $\mathbf{x}_{jk}$*)* **do**

    $G_{left}^{(i)} \leftarrow G_{left} + g_j^{(i)}$ for $i = 1, 2, 3$

    $G_{right}^{(i)} \leftarrow G^{(i)} - G_{left}^{(i)}$ for $i = 1, 2, 3$

    Compute $\mathcal{L}_{node}$, $\mathcal{L}_{left}$, $\mathcal{L}_{right}$ Eq. 11

    $score \leftarrow \max(score, \mathcal{L}_{node} - (\mathcal{L}_{left} + \mathcal{L}_{right}))$

  end

end

Output: Split with max score

Appendix B. Full traning curves

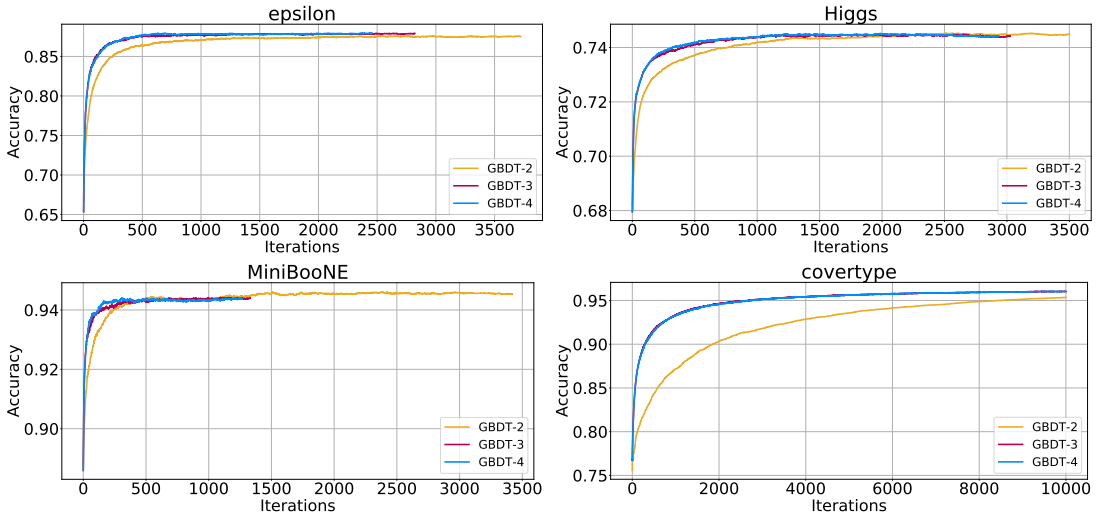

Figure 2: Test accuracy for high-order GBDT models (full training curves).