Oblivious piecewise-linear decision trees

A. Gurianov
Lomonosov Moscow State University
Moscow, Russia
guryanov93@gmail.com

***Abstract*—Gradient boosting ensembles of decision trees are a very popular type of machine learning model, with several popular implementations. Some of those implementations utilize symmetric decision trees - decision trees of a specific structure that improve regularization and speed predictions. Over the last years, several research works have been published related to piecewise-linear decision trees and their utilization in gradient boosting ensembles. In this paper, symmetric piecewise-linear decision trees were introduced, and it was shown that it is possible to efficiently train such trees in gradient boosting ensembles. It was shown that such symmetric piecewise-linear decision trees have significantly faster prediction time compared to regular decision trees and piecewise-linear decision trees of similar depths and that ensembles of symmetric piecewise-linear decision trees achieve competitive quality on open datasets.**

Keywords— machine learning, decision trees, piecewise-linear decision trees, gradient boosting.

I. Introduction

Gradient boosting over decision trees is one of the most popular machine learning algorithms. It was initially proposed in [1], and it can achieve excellent quality on a large variety of datasets out of the box, has built-in feature selection and regularization, and also has several efficient implementations [2], [3], [4]. Improving efficiency and performance of gradient boosting ensembles are popular areas of scientific research. One of the possible improvements to those algorithms is the usage of oblivious (or symmetric) decision trees [5] that have the same split rule on each level. This decision tree structure drastically reduces prediction time of a single decision tree, which is important in many big data and highload scenarios. Another possible type of tree that possesses interesting properties is a piecewise-linear decision tree. Piecewise-linear decision trees return a linear function over features in a leaf as a prediction. This allowing to more easily model linear dependencies in the data and improves performance without increasing prediction times.

In this paper, a definition of oblivious piecewise-linear decision trees is proposed. Algorithms for training and predicting such trees were implemented using Python, and a significant boost in prediction speed is demonstrated on publicly available datasets.

This paper is organized as follows. First, a brief overview of gradient boosting over decision trees is given. Then the concept of piecewise-linear decision trees is explained. Then a definition of oblivious piecewise-linear decision tree is given, and an algorithm for efficient training of such trees within gradient boosting paradigm is described. The next section describes an experimental evaluation of prediction speed of suggested algorithms. The last chapter contains the conclusion of the work.

II. Gradient Boosting over Decision Trees

Given a dataset of samples $D = (x_i, y_i)$ with m features, where $|D| = n$ is the number of samples, and $x_i \in R^m$, gradient boosting algorithm trains a sequence of size T of decision trees $\{t_k\}_1^T$ to minimise global loss function L. The final output of gradient boosting is the summation of these trees $\hat{y} = \sum_{k=1}^{T} t_k(x_i)$. Let $l: R^2 \to R$ be the loss function for a single sample. Then the global loss function is usually represented as a summation of individual losses over training dataset D and a regularization term $\Omega(t_k)$ to prevent overfitting:

$$L = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{T} \Omega(t_k).$$

Let $\hat{y}_i^{(k)}$ be the predicted value of x_i after iteration k. At iteration $k+1$ a new tree t_{k+1} is trained to minimise the following loss

$$\begin{aligned} L^{(k+1)} &= \sum_{i=1}^{n} l(\hat{y}_i^{(k+1)}, y_i) + \sum_{k=1}^{T} \Omega(t_k) \\ &= \sum_{i=1}^{n} l(\hat{y}_i^{(k)} + t_k(x_i), y_i) + \sum_{k=1}^{T} \Omega(t_k). \end{aligned}$$

With second-order approximation, and assuming all predictors from 1 to k fixed, the loss at step k can be represented as

$$L^{(k+1)} \approx C + \Omega(t_k) + \sum_{i=1}^{n} (g_i t_k(x_i) + \frac{1}{2} h_i t_k(x_i)^2),$$

Popular libraries for gradient boosting over decision trees implement algorithms that minimize global loss in the way described above. Piecewise-linear decision trees described in [6] also fit linear coefficients for features in the split. For such trees it is possible to represent loss component of a leaf parametrized with linear coefficient w and constant value b the following way:

$$GHF_N = \sum_{i\in N} g_i + h_i f_i, GXHFX_N^j = \sum_{i\in N} g_i x_i^j + h_i f_i x_i^j$$

$$H_N = \sum_{i\in N} h_i, HX_N^j = \sum_{i\in N} h_i x_i^j, HXX_N^j = \sum_{i\in N} h_i (x_i^j)^2.$$

$$\begin{aligned} L_{leaf}^j = \frac{1}{2} w^2 (HXX_N^j + \lambda_w) + \frac{1}{2} b^2 (H_N + \lambda_b) \\ + w \cdot GXHFX_N^j + b \cdot GHF_N \\ + wb \cdot HX_N^j + \sum (g_i f_i + \frac{1}{2} h_i f_i^2). \end{aligned}$$

III. Oblivious Piecewise-Linear Decision Trees

A. Definition

Definition 1 (Symmetric piecewise-linear decision tree). A symmetric piecewise-linear decision tree is a piecewise-linear decision tree that has all the levels of the tree completely filled, all nodes on the same level have the same splitting rule and common left linear coefficient and right linear coefficient.

Such kind of trees are simpler than general piecewise-linear decision trees and can be considered as a method of regularization. They also have an increased prediction speed, having a greater degree of utilization of modern CPUs through SIMD parallelization.

B. *Efficient Learning*

Algorithm for efficient learning of oblivious piecewise-linear decision trees has a lot in common with algorithm described in [6], with several key differences. The general idea remains the same - at each step of the algorithm we choose to change the tree structure to optimize global loss function based on values of gradients and hessians in each training instance computed from gradient descent step.

First difference is that tree building is performed level-wise instead of node-wise. Second difference is that we build each level of the tree by considering every possible split over every possible feature that optimizes global loss function assuming common splitting rules and left and right linear coefficients for nodes on the same level.

$$\begin{aligned} L_{level}^{j} = \sum_{l \in Leaves} \Big(&\frac{1}{2} w^2 (HXX_N^j + \lambda_w) + \frac{1}{2} b_l^2 (H_N + \lambda_{b_l}) \\ &+ w \cdot GXHFX_N^j + b_l \cdot GHF_N \\ &+ w \cdot b_l \cdot HX_N^j \Big). \end{aligned}$$

Finding optimal values of linear coefficients, leaf values on each level and optimal value of loss function corresponding to each splitting rule is a quadratic form minimization problem. The matrix representation of this quadratic form is an arrowhead matrix, which has efficient algorithms for computing an inverse matrix [7]. This inverse algorithm is used to solve a minimization problem and find optimal values of loss, linear coefficients, and leaf values at each level. Building of levels in the oblivious piecewise-linear decision tree continues until stop condition is achieved, usually formulated as a limit on the number of levels.

IV. Experiments

Suggested algorithm for learning and predicting gradient boosting ensembles of oblivious piecewise-linear decision trees was implemented using Python programming language. To demonstrate improvements in prediction performance, ensembles of 100 decision trees with depth 5 were trained on a HIGGS dataset that consists of 11000000 objects with 28 real-valued features using XGBoost, CatBoost, Piecewise-Linear decision trees (PWL) and Oblivious Piecewise-Linear decision trees (SYMPWL). These ensembles were used to predict 5% of HIGGS dataset 100 times, and averages prediction times were reported in the table below. According to the results, oblivious piecewise-linear decision trees achieved a significant improvement in prediction time compared to regular piecewise-linear decision trees but are slower than CatBoost oblivious decision trees.

Table I. HIGGS Dataset Features Using XGBoost, CatBoost, PWL and SYMPWL

Library	Prediction time (ms)
XGBoost	280
CatBoost	90
PWL	300
SYMPWL	132

V. Conclusion

Efficient machine learning algorithms that are able to demonstrate the best quality possible in as little prediction time as possible is an important area of research relevant for many industries. This paper introduces oblivious piecewise-linear decision trees - a type of decision tree that combines advantages of oblivious decision trees and piecewise-linear decision trees. Suggested oblivious piecewise-linear decision tree is possible to efficiently train and it is possible to implement an algorithm that achieves a significant prediction speedup per tree compared to XGBoost and regular piecewise-linear decision trees.

References

[1] Friedman, J.H. Stochastic gradient boosting / J.H. Friedman // Computational statistics & data analysis. – 2002. – Vol. 38(4). – P. 367-378.

[2] Ke, G. Lightgbm: A highly efficient gradient boosting decision tree / G. Ke // Advances in neural information processing systems. – 2017. – Vol. 30.

[3] Chen, T. Xgboost: A scalable tree boosting system / T. Chen, C. Guestrin // Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. – 2016. – P. 785-794.

[4] Dorogush, A.V. CatBoost: gradient boosting with categorical features support / A.V. Dorogush, V. Ershov, A. Gulin // ArXiv preprint: 1810.11363, 2018.

[5] Kohavi, R. Oblivious decision trees, graphs, and top-down pruning / R. Kohavi, C.H. Li // IJCAI. – 1995. – P. 1071-1079.

[6] Guryanov, A. Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees / A. Guryanov // International Conference on Analysis of Images, Social Networks and Texts. – Springer, Cham, 2019. – P. 39-50.

[7] Najafi, H.S. An efficient method for computing the inverse of arrowhead matrices / H.S. Najafi, S.A. Edalatpanah, G.A. Gravvanis // Applied Mathematics Letters. – 2014. – Vol. 33. – P. 1-5.