# HistGBODT: Gradient Boosting for Ordinal Classification

by

Kemeng Zhang

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Bachelor of Science.

Baltimore, Maryland

May 14, 2020

# Abstract

Tree-based models, including Gradient Boosting Trees and Random Forests, have recently demonstrated state-of-the-art performance in a variety of machine learning tasks. Most notably, popular implementations of Gradient Boosting such as `XGBoost`, `LightGBM`, and most recently, `CatBoost` have become the go-to machine learning solutions widely adopted in industry and academia by researchers. Gradient Boosting is compatible with a wide range of loss functions, thus can perform tasks such as: regression, binary/multi-classification, and learning-to-rank. Unfortunately, in the existing gradient boosting framework, there has not been any implementation that aims to address the situation where the target variable is discrete, but ordered. This class of problem is called "Ordinal Classification". Traditionally, researchers ignore the ordered structure of the target variable and model it using multi-classification, or ignore the discrete nature of the target variable and model it using learning-to-rank. We introduce yet another extension to Gradient Boosting, called "Histogram-based Gradient Boosting Ordinal Decision Trees" (`HistGBODT`). We provide the theoretical foundation for learning discrete ordinal target variables and the algorithmic details

ABSTRACT

for achieving such goals. To illustrate how `HistGBODT` addresses ordinal classification, we conduct extensive experiments, both on simulated and real-world datasets. `HistGBODT` typically yields improved performance over Gradient Boosting with multi-classification (cross-entropy) loss, while maintaining computational efficiency and interpretability. `HistGBODT` can be adopted when it is unclear when a learning task at hand should be approached by classification or learning-to-rank.

Primary Reader: Dr. John C. Miller

# Acknowledgments

I would like to thank my advisor, Dr. Miller, for his guidance during my undergraduate years at Johns Hopkins. I will never forget the exciting discussions we had both in academics and in life. I want to thank Dr. Fishkind and Dr. Athreya not only for their insightful comments and encouragement, but also for the thought-provoking questions which motivated me to broaden my research from various perspectives, in statistics, optimization, and matrix analysis. Last but not least I want to thank my family for supporting me obtaining my undergraduate degree, both financially and spiritually.

# Dedication

This thesis is dedicated to my parents.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Gradient Boosting Decision Trees (GBDT) is a highly effective machine learning method used in a variety of tasks. It has become the de-facto choice of method for problems with noisy and high-dimensional data such as weather forecasting [3, 4], recommendation systems [5], and ad click-through rate prediction [6]. A vast majority of winning solutions to machine learning challenges hosted by Kaggle incorporated Gradient Boosting in some fashion.

The reason why GBDT is gaining popularity in predictive modeling is that it can perform some of the most important tasks such as classification, regression, and ranking. GBDT is able to capture the complex data dependencies by natively incorporating feature interactions in the process of constructing decision trees. It produces an ensemble of weak learners using decision trees, which has many desirable properties as a statistical model. Currently, there are many fast and scalable implementations

of gradient boosting: `XGBoost`, `LightGBM`, and `CatBoost` [2, 7–9] that have found a wide following among practitioners.

## 1.1 Motivation

Gradient Boosting is compatible with a wide range of problems and objectives. In this paper, we consider using Gradient Boosting for the objective of Ordinal Classification. Our aim is to learn a rule to predict labels from a discrete ordered set. For example, users generate ratings for products on Amazon from several rating "levels", e.g. one through five stars. Similar problems often arises when the target variable is human-generated ratings, severity of symptoms of diseases, or performance metrics.

Ordinal Classification is fundamentally different from traditional machine learning tasks such as classification, regression, and learning-to-rank. Similar to classification, our target variable consist of discrete values; as in regression, there is meaningful order between values; as in learning-to-rank, we aim to predict relative order between each instances. Different from classification, our label is discrete but ordered. We call this type of label "ordinal" as opposed to "nominal". Ordinal classification formalizes the notion of order by incurring a greater penalty for predictions farther from the true label than those closer. As in regression, our target variable is not on a continuous scale. As in learning-to-rank, which also outputs on a continuous scale, fails to produce a discrete label for each instance. In this sense, it is possible for a ranking

model (but not for ordinal classification) to predict the wrong label and incur no loss at all, as long as the relative order of the true labels and predicted labels are correct. Although ordinal classification and learning-to-rank are two different problems, the distinction is not always clear.

In the past, ordinal classification could be approached by all of the three tasks mentioned previously: multi-classification, regression, or learning-to-rank. However, none of these approaches reflect the specific structure of discrete ordered labels.

## 1.2 Literature

To date, the most popular approach to model discrete ordinal data is by Ordinal Logistic Regression. McCullagh (1980) introduced the Proportional Odds Model, which uses logits of cumulative probabilities, often called cumulative logits [10–12]. Recently, many algorithms for ordinal classification have been proposed from a machine learning perspective. Freund et al. [13] studied the ranking objective in collaborative filtering and proposes an AdaBoost based solution called `RankBoost`. Dembczyński et al. [14] further the boosting framework by providing a gradient descent algorithm for ordinal classification. A crucial aspect of the learning objective is the loss function. Rennie and Srebro [15] surveyed strategies for learning discrete ordered labels. They view ordinal classification as a generalization of binary classification. Thus, commonly used loss functions for binary classification can be easily

extended to multi-category. Most notably, we adopt the "All-Threshold Loss" from threshold-based approach in Rennie and Srebro. Crammer and Singer [16] suggest a generalization of the Perceptron algorithm for discrete ordinal labels. Shashua and Levin [17] suggest a similar generalization to Support Vector Machine (SVM). Chu and Keerthi [18] proposes support vector approaches for ordinal classification, which optimize multiple thresholds to define parallel discriminant hyperplanes for the ordinal scales. This approach guarantees that the thresholds are properly ordered at the optimal solution. Finally, the theoretical properties of threshold-based loss function are discussed in Lin and Li [19, 20], and Pedregosa et al. [21] characterizes the Fisher consistency of threshold-based loss functions in the context of ordinal classification.

## 1.3 Main Contribution

Our main contribution is to extend the powerful gradient boosting algorithm to learn discrete ordinal response variables using the All-Threshold Loss function. We provide an efficient algorithm for optimizing our objective using Majorization-Minimization (MM) with provable convergence guarantees.

## 1.4 Thesis Structure

In Chapter 2, I will introduce the statistical framework of Ordinal Classification and the All-Threshold loss function. In Chapter 3, I will review the Gradient Boosting

learning objective in the traditional setting, and extend the objective to All-Threshold loss. In Chapter 4, I will describe the algorithms that speed up the learning speed during the growth of each decision tree in detail. Finally, Chapter 5 will provide numerical experimentation on simulated and real-world datasets, comparing the performance of multi-classification and ordinal classification.

# Chapter 2

# Ordinal Classification: Statistical Framework

## 2.1 Problem Setting

We present the general framework learning discrete ordinal responses. For a given data set with $n$ examples and $m$ features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ ($|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \{r_1, ..., r_K\}$), it is assumed our response variable is from a set of $K$ discrete ordered class labels $r_1 \prec ... \prec r_{K-1} \prec r_K$, where $\prec$ denotes the ordering relation between the labels. Let us assume, without loss of generality, that $r_k = k - 1$.

## 2.2 Ordinal Classification

In ordinal classification, label can take one of several discrete, but ordered labels. For instance, movie ratings can go from zero to ten stars. Unlike multi-class classification or learning-to-rank, not all errors are equally bad, e.g., it is worse to mis-classify a 1-star with a 10-star than a 4-star movie with a 5-star one. Hence, we construct a loss matrix $L \in \mathbb{R}^{K \times K}$ for ordinal classification to satisfy the following properties:

1. $L(y, \hat{y}) = L_{ij}$, where $i = y, j = \hat{y} = F(\mathbf{x}) \in \{0, ...K-1\}$.

2. $L_{ii} = 0$, $\forall i$

3. $L_{ij} \leq L_{ik}$, $\forall i < j < k$, or $k < j < i$.

In other words, we incur no loss when the predicted label is the same as ground truth, and a larger loss when the absolute distance between the predicted label and the ground truth is larger.

## 2.3 Loss Function Construction

Note that in traditional multi-class classification with $K$ classes, the problem is often solved as a collection of two class sub-problems. Friedman [22], and Hastie & Tibshirani [23] considers $\binom{K}{2}$ pairwise subproblems, often referred to as the "One-vs-One" (OvO) strategy. Another strategy, often referred to as "One-vs-Rest" (OvR), considers building one classifier per class. For each classifier, the class is fitted against

all the other classes, hence $K$ classifiers in total [24]. In Rennie and Srebro [15], the authors consider the strategy for building classifiers for $K$ discrete ordered labels. They also view ordinal classification as a generalization of binary classification, and adopted the "One-vs-Rest" strategy by exploiting the ordering structure in the label when constructing the loss function. They extend the binary loss functions to the case of ordinal classification by introducing $K-1$ thresholds. We first review common loss functions used with binary responses, where $y \in \pm 1$. Then we extend this to more general loss functions for discrete ordinal labels.

### 2.3.1 Binary Classification

Consider the loss function over training observations:

$$L(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, \tilde{y}_i) \tag{2.1}$$

where $\tilde{y}_i = f(\mathbf{x}_i) \in \mathbb{R}$; $\hat{y}_i = \text{sgn}(\tilde{y}_i) \in \pm 1$. The loss function $l$ should have several desired properties. It should be close to zero when $\tilde{y}_i$ and $y_i$ agree in sign and have non-negative value when they have opposite signs. These common choices are:

- Zero-one loss, $I(\text{sgn}(y_i \tilde{y}_i) \neq 1)$, where $I$ is the indicator function;
- Hinge loss, $\max(0, 1 - \tilde{y}_i y_i)$
- Logistic loss, $\log(1 + \exp(-\tilde{y}_i y_i))$

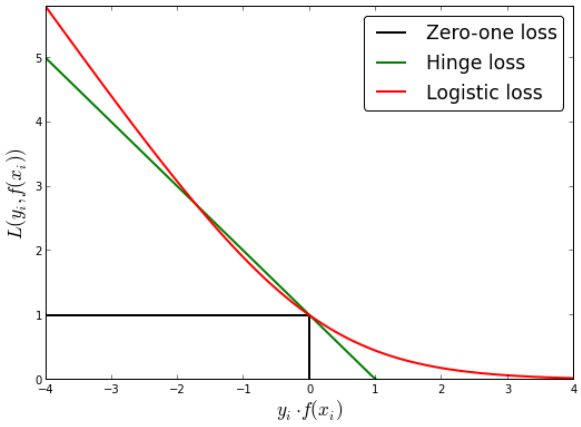



Figure 2.1: Binary Classification Loss Functions.

### 2.3.2 Extending Binary Classification - Threshold-Based Constructions

In the binary case, we use a single threshold, namely 0, to separate the real line into two distinct segments: $\tilde{y}_i = f(\mathbf{x}_i) > 0$ corresponds to $\hat{y}_i = +1$ predicted label, while $\hat{y}_i = -1$ otherwise. To extend binary loss to the case of discrete ordinal regression, [15] introduce $K - 1$ thresholds $\theta_0 < \theta_1 < ... < \theta_{K-2}$ partitioning the real line into $K$ segments. We denote this vector of threshold as $\boldsymbol{\theta} \in \mathbb{R}^{K-1}$. Each of the K segment corresponds to one of the $K$ labels. The predicted class label $\hat{y}_i$ can thus

be derived from the predicted score $\tilde{y}_i$ and thresholds vector $\boldsymbol{\theta}$:

$$\hat{y}_i = \begin{cases} 0 & \text{if } \tilde{y}_i \in (-\infty, \theta_0) \\ k & \text{if } \tilde{y}_i \in [\theta_{k-1}, \theta_k) \\ K-1 & \text{if } \tilde{y}_i \in [\theta_{K-2}, \infty) \end{cases} \tag{2.2}$$

### 2.3.3 All-Threshold Loss

The idea behind [15] all-threshold loss is to sum up all threshold violation penalties. Here, not only the function increases when thresholds are violated, but the slope of the loss function increases each time a threshold is crossed.

$$l_{ord}(y_i, \tilde{y}_i, \boldsymbol{\theta}) = \sum_{k=0}^{y_i-1} l(1, \tilde{y}_i - \theta_k) + \sum_{k=y_i}^{K-2} l(-1, \tilde{y}_i - \theta_k) \tag{2.3}$$

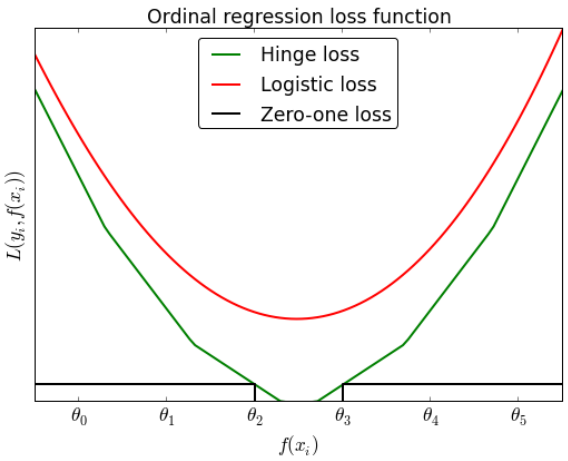



Figure 2.2: Ordinal Classification Loss Functions when $y_i = 3$.

In Figure 2.2, consider the case $K = 7, y_i = 3$. The x-axis denotes $\tilde{y}_i = f(\mathbf{x}_i)$, and the y-axis denotes the loss incurred by $\tilde{y}_i$. All-threshold loss checks if $\tilde{y}_i > \theta_k$ for $k = 0, 1, 2$; if any threshold is violated, a loss is incurred $l(1, \tilde{y}_i - \theta_k)$, where $l$ is the binary classification loss mentioned previously. Similarly, it also checks if $\tilde{y}_i < \theta_k$ for $k = 3, 4, 5$; a loss $l(-1, \tilde{y}_i - \theta_k)$ is incurred if any threshold value is violated. We can make our expression more compact by defining the following binary function:

$$\mathrm{s}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases} \tag{2.4}$$

$$l_{ord}(y_i, \tilde{y}_i, \boldsymbol{\theta}) = \sum_{k=0}^{K-2} l(\mathrm{s}(y_i - k), \tilde{y}_i - \theta_k) \tag{2.5}$$

A problem arises if in the learned threshold $\boldsymbol{\theta}$, there exists $k$ such that $\theta_k > \theta_{k+1}$. In Li et. al [20] Theorem 2, the authors proved that under this framework as long as $l$ is convex there exists optimal solution $(\tilde{\boldsymbol{y}}^*, \boldsymbol{\theta}^*)$ such that $\boldsymbol{\theta}^*$ is ordered.

**Probabilistic Interpretation of the All-Threshold Loss**

Define

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\psi(x) = \log(1 + e^{-x}) = -\log(\sigma(x)) \text{ (Negative log-likelihood)}$$

$$\mathbb{P}(y_i \leq k) = \sigma(\mathrm{s}(y_i - k)(\tilde{y}_i - \theta_k)) = \sigma(\theta_k - \tilde{y}_i)$$

# Chapter 3

# Gradient Boosted Trees

We review the gradient tree boosting algorithms [2,25]. Furthermore, we introduce learning rules under the context of the statistical framework of threshold-based ordinal classification.

## 3.1 Building Block: Decision Trees

The most popular base learner in gradient boosting is the decision tree, also known as CART [26]. It is represented as a directed acyclic graph data structure (See Fig. 3.1). Given a training instance $\mathbf{x}_i \in \mathbb{R}^m$ to be predicted, we traverse the tree from top to bottom, performing a series of boolean operations stored in each intermediate node, comparing a threshold value against a feature in $\mathbf{x}_i$ in the process. We proceed to either the left child or the right child depending on the result of the comparison.

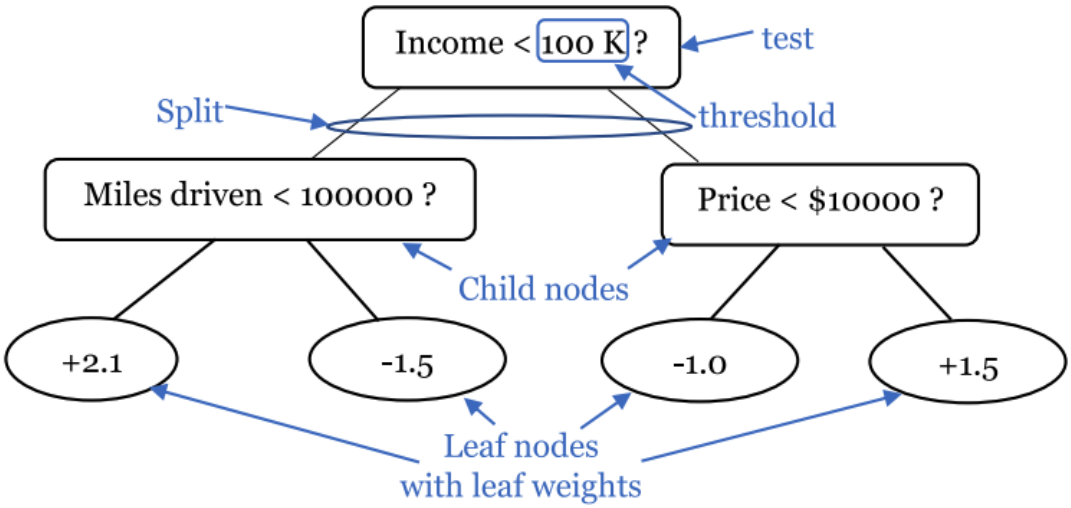



Figure 3.1: A visualization of decision tree. [1].

The intermediate node with a feature and a threshold value represents a "split" on the dataset. We repeat the splitting process until we reach the terminal leaf region, where the node has no left or right child, and stores an output value called the "leaf weight". Each leaf node gives rise to an instance set $I_j$, the set of all data points for which traversal ended at that leaf node [1].

Formally, the predicted output of a decision tree can be expressed as:

$$\tilde{y}_i = \sum_{j=1}^{J} w_j \mathbf{1}_{\{\mathbf{x}_i \in I_j\}} \tag{3.1}$$

The strategy for growing a decision tree will be discussed in Chapter 4 where the algorithmic aspect of `HistGBODT` will be explained in detail.

## 3.2 Regularized Learning Objective

A tree ensemble model uses T additive functions to predict output.

$$\tilde{y}_i = \sum_{t=1}^{T} f_t(\mathbf{x}_i), f_t \in \mathcal{F}, \tag{3.2}$$

where $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}$ is the space of regression trees (also known as CART). Here $q : \mathbb{R}^m \to \{1, 2, ..., J\}$ is the mapping from the feature space to the terminal leaf assignment, i.e. the tree structure; define instance set $I_j = \{i|q(\mathbf{x}_i) = j\}$ as the set of examples that get assigned to leaf $j$; $f_t(\mathbf{x}_i) = w_j$ if $q(\mathbf{x}_i) = j$, i.e. $i \in I_j$; $J$ is the number of leaves in a tree; $w_j \in \mathbb{R}$ is the predicted output on the $j$-th leaf. For a given example, we will use decision rules $q$ in each tree to assign each example into the leaves and calculate the final prediction by summing up the scores (given by $w$) in the corresponding leaves. To learn the set of functions in the ensemble model, we minimize the following *regularized* objective:

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^{n} l_{ord}(y_i, \tilde{y}_i, \boldsymbol{\theta}) + \sum_{t=1}^{T} \Omega(f_t) \\ &\text{where } \Omega(f) = \gamma J + \frac{1}{2}\lambda ||w||^2 \end{aligned} \tag{3.3}$$

Here the second term $\Omega$ penalizes the complexity of the regression trees. The additional term regularizes the model to smooth out the final prediction in order to avoid over-fitting. Intuitively, the regularized objective tends to select simpler and predictive regression trees.

## 3.3 Gradient Tree Boosting

The tree ensemble model in Eq. (3.3) includes functions as parameters and cannot be optimized directly. Instead, the model is trained in an additive manner. Formally, let $\tilde{y}_i^{(t)}$ be the prediction of the $i$-th example at the $t$-th iteration, we need to add $f_t$, and $\boldsymbol{\delta}^{(t)}$ to minimize the following objective:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l_{ord}(y_i, \tilde{y}_i^{(t-1)} + f_t(\mathbf{x}_i), \boldsymbol{\theta}^{(t-1)} + \boldsymbol{\delta}^{(t)}) + \Omega(f_t) \tag{3.4}$$

where $\tilde{y}_i^{(t-1)}$ is the combined output of the previous $t-1$ trees with the $i$-th data point as input.

We greedily add new regression tree $f_t$ and threshold improvement $\boldsymbol{\delta}^{(t)}$ that most improve our model according to Eq. (3.3). To quickly optimize the objective using second-order approximation, we choose the logistic loss in Sec. 2.3.1 that is convex and twice differentiable.

We initialize at $t = 0$:

$$\begin{aligned} \tilde{y}^{(0)}, \boldsymbol{\theta}^{(0)} &= \underset{\tilde{y}, \boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} l_{ord}(y_i, \tilde{y}, \boldsymbol{\theta}) \\ \tilde{y}_i^{(0)} &= \tilde{y}^{(0)} \quad \forall i = \{1, ..., n\} \end{aligned} \tag{3.5}$$

$$\begin{aligned} \mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \Big[ l_{ord}(y_i, \tilde{y}_i^{(t-1)}, \boldsymbol{\theta}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \Big] + \\ \left( \sum_{i=1}^{n} \frac{\partial l_i}{\partial \boldsymbol{\theta}} \right)^{\intercal} \boldsymbol{\delta}^{(t)} + \frac{1}{2} \boldsymbol{\delta}^{(t)\intercal} \left( \sum_{i=1}^{n} \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^{\intercal}} \right) \boldsymbol{\delta}^{(t)} + \left( \sum_{i=1}^{n} f_t(\mathbf{x}_i) \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \tilde{y}} \right)^{\intercal} \boldsymbol{\delta}^{(t)} + \Omega(f_t) \end{aligned} \tag{3.6}$$

Exploiting the tree structure $q_t$ of $f_t$, we can re-write $\mathcal{L}^{(t)}$ in terms of instance sets $I_j$. We drop all constant term to get a simplified objective:

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} = &\sum_{j=1}^{J}\Big[\Big(\sum_{i\in I_j}(g_i + \frac{\partial^2 l_i}{\partial\boldsymbol{\theta}\partial\tilde{y}}^{\intercal}\boldsymbol{\delta}^{(t)})\Big)w_j + \frac{1}{2}\big(\sum_{i\in I_j}h_i + \lambda\big)w_j^2\Big] + \\ &\left(\sum_{i=1}^{n}\frac{\partial l_i}{\partial\boldsymbol{\theta}}\right)^{\intercal}\boldsymbol{\delta}^{(t)} + \frac{1}{2}\boldsymbol{\delta}^{(t)\intercal}\left(\sum_{i=1}^{n}\frac{\partial^2 l_i}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^{\intercal}}\right)\boldsymbol{\delta}^{(t)} + \gamma J\end{aligned} \tag{3.7}$$

where

$$\begin{aligned}l_i &= l_{ord}(y_i, \tilde{y}, \boldsymbol{\theta})|_{\tilde{y}=\tilde{y}_i^{(t-1)}, \boldsymbol{\theta}=\boldsymbol{\theta}^{(t-1)}} \\ g_i &= \frac{\partial}{\partial\tilde{y}}l_{ord}(y_i, \tilde{y}, \boldsymbol{\theta})|_{\tilde{y}=\tilde{y}_i^{(t-1)}, \boldsymbol{\theta}=\boldsymbol{\theta}^{(t-1)}} \\ h_i &= \frac{\partial^2}{\partial\tilde{y}^2}l_{ord}(y_i, \tilde{y}, \boldsymbol{\theta})|_{\tilde{y}=\tilde{y}_i^{(t-1)}, \boldsymbol{\theta}=\boldsymbol{\theta}^{(t-1)}}\end{aligned}$$

Note that our simplified loss function could also be rewritten in matrix form for simplicity. We define

Trainable parameters (descent direction in a Newton-Raphson step):

$$\boldsymbol{\beta} = \begin{bmatrix} w_1 & ... & w_J & \delta_0 & ... & \delta_{K-2}\end{bmatrix}^{\intercal}$$

Full gradients:

$$\boldsymbol{g} = \begin{bmatrix} G_1 & ... & G_J & \boldsymbol{u}^{\intercal}\end{bmatrix}^{\intercal}$$

Hessian matrix:

$$\boldsymbol{H} = \begin{bmatrix} H_1 + \lambda & & 0 & — & \boldsymbol{L}_1^\intercal & — \\ & \ddots & & & \vdots & \\ 0 & & H_J + \lambda & — & \boldsymbol{L}_J^\intercal & — \\ | & & | & & & \\ \boldsymbol{L}_1 & \dots & \boldsymbol{L}_J & & \bigvee & \\ | & & | & & & \end{bmatrix} \tag{3.8}$$

where $\tilde{\mathcal{L}}^{(t)}$ can be re-expressed in matrix form as:

$$\tilde{\mathcal{L}}^{(t)} = \boldsymbol{g}^\intercal \boldsymbol{\beta} + \frac{1}{2} \boldsymbol{\beta}^\intercal \boldsymbol{H} \boldsymbol{\beta} + \gamma J \tag{3.9}$$

here

$$G_j = \sum_{i \in I_j} g_i \qquad \boldsymbol{u} = \sum_{i=1}^{n} \frac{\partial l_i}{\partial \boldsymbol{\theta}} \qquad H_j = \sum_{i \in I_j} h_i$$

$$\boldsymbol{L}_j = \sum_{i \in I_j} \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \tilde{y}} \qquad \boldsymbol{V} = \sum_{i=1}^{n} \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^\intercal}$$

See Appendix B for full expressions of the gradient statistics above. We can easily solve for the optimal parameter from the quadratic program above:

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\ \tilde{\mathcal{L}}^{(t)} = -\boldsymbol{H}^{-1} \boldsymbol{g} \tag{3.10}$$

The corresponding value for $\tilde{\mathcal{L}}^{(t)}$ is

$$\tilde{\mathcal{L}}^{(t)*} = -\frac{1}{2}\boldsymbol{g}^\intercal \boldsymbol{H}^{-1}\boldsymbol{g} + \gamma J \tag{3.11}$$

### 3.3.1 Analysis of Hessian Matrix

Computing the optimal paramter $\boldsymbol{\beta}^*$ requires inverting the Hessian matrix $\boldsymbol{H}$, which runs in $\mathcal{O}\big((J+K)^3\big)$ time. Moreover, note when the $\ell_2$ regularization parameter $\lambda = 0$, the rows and columns of $\boldsymbol{H}$ sum to 0, which means it has an eigenvalue 0 and associated eigenvector $\mathbf{1}$. The Hessian matrix is thus singular and we cannot apply Newton-Raphson method directly to find the minimum of our objective.

It turns out we could both get the inversion time complexity down to $\mathcal{O}\big(K^3\big)$, and construct a "majorizing" Hessian matrix that is positive definite regardless of the value of $\ell_2$ regularization parameter $\lambda$.

#### 3.3.1.1 Inversion of Block Matrices

The special structure of the Hessian matrix can be exploited. Note that $\boldsymbol{H}$ is a 2×2 block matrix, with two diagonal matrices along its diagonal entries. Inverting diagonal matrices is trivial. Inverse formulae also exists for 2×2 block matrices [27]:

$$\boldsymbol{H}^{-1} = \begin{bmatrix} \boldsymbol{H}_\lambda^{-1} + \boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal(\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal)^{-1}\boldsymbol{L}\boldsymbol{H}_\lambda^{-1} & -\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal(\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal)^{-1} \\ -(\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal)^{-1}\boldsymbol{L}\boldsymbol{H}_\lambda^{-1} & (\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal)^{-1} \end{bmatrix} \tag{3.12}$$

where

$$\boldsymbol{H}_\lambda = \begin{bmatrix} H_1 + \lambda & & 0 \\ & \ddots & \\ 0 & & H_J + \lambda \end{bmatrix}$$

$\boldsymbol{H}_\lambda$ is nonsingular, and Hessian $\boldsymbol{H}$ is invertible whenever the Schur complement $\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal$ of $\boldsymbol{H}_\lambda$ is also invertible. In other words, in order to compute the inverse of the Hessian matrix we only need to invert $\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal$, in addition to matrix multiplication. Hence, we are able to reduce the computational cost such that the inversion does not scale with decision tree growth, as the number of leaves grow larger and larger.

For clarity, we can now re-express the optimal value for each parameter:

$$\begin{aligned} \boldsymbol{\delta}^{(t)*} &= -\Big(\boldsymbol{V} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{L}^\intercal\Big)^{-1}\Big(\boldsymbol{u} - \boldsymbol{L}\boldsymbol{H}_\lambda^{-1}\boldsymbol{g}\Big) \\ &= -\Big(\boldsymbol{V} - \sum_{j=1}^{J} \frac{\boldsymbol{L}_j \boldsymbol{L}_j^\intercal}{H_j + \lambda}\Big)^{-1}\Big(\boldsymbol{u} - \sum_{j=1}^{J} \frac{G_j}{H_j + \lambda}\boldsymbol{L}_j\Big) \\ w_j^* &= -\frac{G_j + \boldsymbol{L}_j^\intercal \boldsymbol{\delta}^{(t)*}}{H_j + \lambda} \end{aligned} \tag{3.13}$$

#### 3.3.1.2 Majorization Minimization: The Construction of Invertible Hessian Matrix

We observe that gradient boosting is, in its essence, a Newton-Raphson optimization algorithm, where we leverage the second order Taylor expansion to the original objective function, and at each boosting iteration train a decision tree to minimize

the objective function in the direction of steepest descent. However, there are cases where it falls apart:

- The objective function is not twice differentiable;
- The objective function is twice differentiable, but the Hessian is singular;

We encountered the second case. The true condition behind gradient boosting to converge is to derive an upper bound of original function. This works exactly like Majorization-Minimization (Expectation-Maximization is its special case), where we try to find a quadratic upper bound of the objective function, and minimize the upper bound [28,29]. In order to find that, we just need to find a majorizing Hessian matrix $\tilde{\boldsymbol{H}} \succ \boldsymbol{H}$. MM is worth considering since we have access to an invertible surrogate hessian matrix, making the objective function much easier to optimize.

Formally, we are minimizing a complicated objective function $\mathcal{L}$ iteratively. We construct a majorizing function by second-order Taylor expansion around the current best solution $\mathbf{x}^{(t)}$, replacing the original hessian by its upper bound. We then find a new solution $\mathbf{x}^{(t+1)}$ by minimizing the majorizing function. Then we construct a new majorizing function around $\mathbf{x}^{(t+1)}$, and so on.

**Definition 1.** *(Majorization) Suppose $f$ and $g$ are twice differentiable at $\mathbf{x}^{(t)}$. If $g$ majorizes $f$ at $\mathbf{x}^{(t)}$ then*

- $g(\mathbf{x}^{(t)}) = f(\mathbf{x}^{(t)})$,

- $\nabla_{\mathbf{x}} g(\mathbf{x}^{(t)}) = \nabla_{\mathbf{x}} f(\mathbf{x}^{(t)})$
- $\nabla^2_{\mathbf{x}} g(\mathbf{x}^{(t)}) \succ \nabla^2_{\mathbf{x}} f(\mathbf{x}^{(t)})$

*Proof.* If $g$ majorizes $f$ at $\mathbf{x}^{(t)}$ then $d = g - f$ has a minimum at $\mathbf{x}^{(t)}$. Now use the familiar necessary conditions for the minimum of a differentiable function, which says that the derivative at the minimum is zero and the hessian is symmetric positive definite. Then we have $g(\mathbf{x}) > f(\mathbf{x})\ \forall \mathbf{x} \neq \mathbf{x}^{(t)}$. □

**Majorization Minimization (MM) Algorithm** Suppose our current best approximation to the minimum of $f$ is $\mathbf{x}^{(t)}$, and we have a $g$ that majorizes $f$ at $\mathbf{x}^{(t)}$. If $\mathbf{x}^{(t)}$ already minimizes $g$ we stop, otherwise we update $\mathbf{x}^{(t)}$ to $\mathbf{x}^{(t+1)}$ by minimizing $g$. If we do not stop we have the sandwich inequality:

$$f(\mathbf{x}^{(t+1)}) < g(\mathbf{x}^{(t+1)}) < g(\mathbf{x}^{(t)}) = f(\mathbf{x}^{(t)}) \tag{3.14}$$

Repeating these steps produces a decreasing sequence of function values, and under appropriate additional compactness and continuity conditions this guarantees convergence of the algorithm.

Now we can consider the critical question: how can we construct $\tilde{\boldsymbol{H}} \succ \boldsymbol{H}$? One way to achieve this is to find a sharp upper bound along the diagonal entries of $\boldsymbol{H}$, such that $\tilde{\boldsymbol{H}} - \boldsymbol{H}$ is a diagonal matrix with positive entries. This operation shifts the eigenvalues of $\boldsymbol{H}$ to the positive direction. As a result, $\tilde{\boldsymbol{H}}$ becomes invertible.

Let's analyze the diagonal entries of $\boldsymbol{H}_\lambda$ and $\boldsymbol{V}$

$$\begin{aligned} H_j + \lambda &= \sum_{i \in I_j} \sum_{k=0}^{K-2} \sigma(s_{ik} d_{ik}) \big(1 - \sigma(s_{ik} d_{ik})\big) + \lambda \\ \boldsymbol{V}_{kk} &= \sum_{i=1}^{n} \sigma(s_{ik} d_{ik}) \big(1 - \sigma(s_{ik} d_{ik})\big) \end{aligned} \tag{3.15}$$

where

$$\begin{aligned} \sigma(x) &= \frac{1}{1 + e^{-x}} \\ s_{ik} &= \mathrm{s}(y_i - k) \\ d_{ik} &= \tilde{y}_i - \theta_k \end{aligned} \tag{3.16}$$

We observe it is sufficient to find a sharp majorizer to $\sigma(x)(1 - \sigma(x))$. Sec. 4.1 of [29] provides a procedure for constructing a sharp majorizer in the differentiable case. Sec 5.1 of [29] conveniently provided us with our desired majorizer $\frac{2\sigma(x)-1}{2x}$ to $\sigma(x)(1 - \sigma(x))$.

**Sharp Quadratic Majorization in One Dimension** Since we would like $\tilde{\boldsymbol{H}}$ to majorize $\boldsymbol{H}$ at each of its diagonal entry, it is sufficient to consider the one-dimensional case. Consider twice differentiable $f(x) \leq g(x) = f(x^{(t)}) + f'(x^{(t)})(x - x^{(t)}) + \frac{1}{2} h(x^{(t)})(x - x^{(t)})^2$. $f(x) = \log(1 + \exp(-x))$ is strongly convex, and is majorized by $g$ at $x^{(t)}$. Hence by the inequality $h$ must satisfy:

$$h(x^{(t)}) \geq \delta(x; x^{(t)}) = \frac{f(x) - f(x^{(t)}) - f'(x^{(t)})(x - x^{(t)})}{\frac{1}{2}(x - x^{(t)})^2} \quad \forall x \neq x^{(t)} \tag{3.17}$$

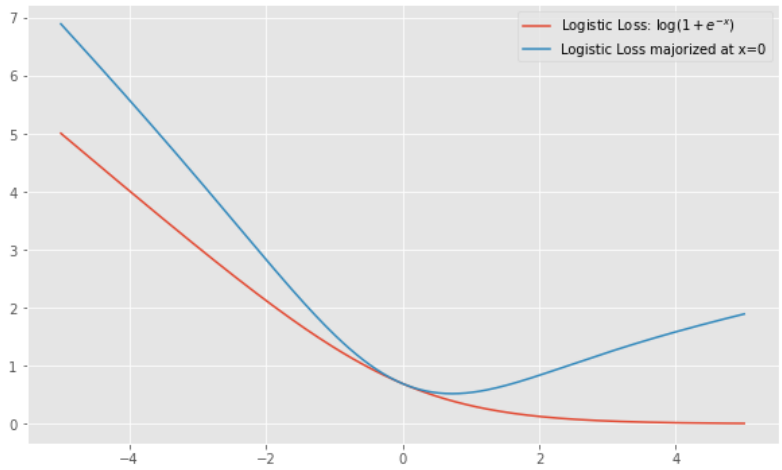



Figure 3.2: Sharp Majorizing function at $x = 0$.

We can find such $h$ if and only if

$$\sup_x \delta(x; x^{(t)}) < \infty$$

And we achieve **sharp quadratic majorization** if we set $h(x^{(t)}) = \sup_x \delta(x; x^{(t)})$ since we have found the tightest lower bound of $h(x^{(t)})$ in order to achieve majorization conditions and the convergence rate will also be the fastest.

**Proposition:** We achieve sharp quadratic majorization when $x = -x^{(t)}$

*Proof.*

$$\begin{aligned}
\frac{\partial}{\partial x}\delta(x;x^{(t)}) &= \frac{\frac{1}{2}(x-x^{(t)})^2\big(f^{'}(x)+f^{'}(x^{(t)})\big)-(x-x^{(t)})\big(f(x)-f(x^{(t)})\big)}{\frac{1}{4}(x-x^{(t)})^4} \\
\frac{\partial}{\partial x}\delta(x;x^{(t)})\Big|_{x=-x^{(t)}} &= \frac{-\frac{1}{2}(-2x^{(t)})^2-(-2x^{(t)})x^{(t)}}{\frac{1}{4}(-2x^{(t)})^4}=0 \\
\frac{\partial^2}{\partial x^2}\delta(x;x^{(t)})\Big|_{x=-x^{(t)}} &= \frac{\frac{1}{2}\Big((x-x^{(t)})^2f^{''}(x)-(x-x^{(t)})\big(f^{'}(x)-f^{'}(x^{(t)})\big)\Big)}{\frac{1}{4}(x-x^{(t)})^4} \\
&= \frac{2x^{(t)^2}\sigma(x^{(t)})\big(1-\sigma(x^{(t)})\big)+x^{(t)}\big(\sigma(-x^{(t)})-\sigma(x^{(t)})\big)}{4x^{(t)^4}} < 0 \quad \forall\ x^{(t)}! = 0
\end{aligned} \tag{3.18}$$

□

Plug in $x=-x^{(t)}$ and we get

$$h(x^{(t)}) = \delta(-x^{(t)};x^{(t)}) = \frac{2\sigma(x^{(t)})-1}{2x^{(t)}} \tag{3.19}$$

In that case, we have found our modified hessian matrix $\tilde{\boldsymbol{H}}$ with diagonal entries:

$$\begin{aligned}
\tilde{H}_j+\lambda &= \sum_{i\in I_j}\sum_{k=0}^{K-2}\frac{2\sigma(s_{ik}d_{ik})-1}{2s_{ik}d_{ik}}+\lambda \\
\tilde{\boldsymbol{V}}_{kk} &= \sum_{i=1}^{n}\frac{2\sigma(s_{ik}d_{ik})-1}{2s_{ik}d_{ik}}
\end{aligned} \tag{3.20}$$

**Convergence Properties of Quadratic Majorization Minimization** Suppose $\mathcal{L}:\mathcal{X}\to\mathbb{R}$ a convex objective function to be minimized, with gradient $\boldsymbol{g}=\nabla_{\mathbf{x}}\mathcal{L}$ and hessian matrix $\boldsymbol{H}=\nabla^2_{\mathbf{x}}\mathcal{L}$,

**Theorem 3.3.1.1.** *(Böhning & Lindsay 1988 [30]) Let* $\mathbf{x}^{(0)}\in\mathcal{X}$ *and suppose that* $(\mathbf{x}^{(t)})_{t\geq 1}$ *is defined by the Majorization Minimization algorithm* $\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\tilde{\boldsymbol{H}}(\mathbf{x}^{(t)})^{-1}\boldsymbol{g}(\mathbf{x}^{(t)})$, *where* $\tilde{\boldsymbol{H}}(\mathbf{x})\succeq\boldsymbol{H}(\mathbf{x})\ \forall\mathbf{x}\in\mathcal{X}$. *The sequence* $(\mathbf{x}^{(t)})_{t\geq 1}$ *has the following properties:*

*(a) Monotonicity:* $\mathcal{L}(\mathbf{x}^{(t+1)}) \geq \mathcal{L}(\mathbf{x}^{(t)})$*, with strict inequality if* $\mathbf{x}^{(t+1)} \neq \mathbf{x}^{(t)}$*.*

*(b) Guaranteed convergence: The sequence* $\left(\boldsymbol{g}(\mathbf{x}^{(t)})\right)_{t\geq 1}$ *converges to 0 if* $\mathcal{L}$ *is bounded below.*

*(c) Rate of convergence: The algorithm converges with rate* $||\mathbb{I} - \tilde{\boldsymbol{H}}(\mathbf{x}^*)^{-1}\boldsymbol{H}(\mathbf{x}^*)||_2$

*Proof.* To prove (a), let the descent direction $\boldsymbol{d} = -\tilde{\boldsymbol{H}}(\mathbf{x}^{(t)})^{-1}\boldsymbol{g}(\mathbf{x}^{(t)})$. At $\mathbf{x}^{(t)}$ construct majorizing function $\mathcal{M}$ of $\mathcal{L}$:

$$\mathcal{M}(\mathbf{x}) = \mathcal{L}(\mathbf{x}^{(t)}) + \boldsymbol{g}(\mathbf{x}^{(t)})^\intercal(\mathbf{x} - \mathbf{x}^{(t)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(t)})^\intercal\tilde{\boldsymbol{H}}(\mathbf{x}^{(t)})(\mathbf{x} - \mathbf{x}^{(t)}) \tag{3.21}$$

By Definition 1, $\mathcal{M}(\mathbf{x}) \geq \mathcal{L}(\mathbf{x})\ \forall x \in \mathcal{X}$, $\mathcal{M}(\mathbf{x}^{(t)}) = \mathcal{L}(\mathbf{x}^{(t)})$. By Taylor expansion we have:

$$\begin{aligned}
\mathcal{L}(\mathbf{x}^{(t+1)}) - \mathcal{L}(\mathbf{x}^{(t)}) &\leq \mathcal{M}(\mathbf{x}^{(t+1)}) - \mathcal{L}(\mathbf{x}^{(t)}) \\
&= \boldsymbol{g}(\mathbf{x}^{(t)})^\intercal\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^\intercal\tilde{\boldsymbol{H}}(\mathbf{x}^{(t)})\boldsymbol{d} \\
&= -\frac{1}{2}\boldsymbol{g}(\mathbf{x}^{(t)})^\intercal\tilde{\boldsymbol{H}}(\mathbf{x}^{(t)})^{-1}\boldsymbol{g}(\mathbf{x}^{(t)}) \\
&\leq 0
\end{aligned}$$

To prove part (b), suppose for sake of contradiction $||\boldsymbol{g}(\mathbf{x}^{(t)})||_2$ is bounded away from 0. But the increment above is clearly bounded above by 0, contradicting the boundedness of $\mathcal{L}$.

Part (c) was proved by Ostrowski's Theorem [Ortega & Rheinboldt, 1970, Theorem 10.1.3] [31]. □

At sharp quadratic majorization minimization, the convergence rate, similar to

that of Newton-Raphson, is still quadratic in our case. As a rough sketch, in the one-dimensional case, plug in $\sigma(x)(1-\sigma(x))$ and $\frac{2\sigma(x)-1}{2x}$ we get $1-\frac{2xe^{-x}}{1-e^{-2x}}=\frac{x^2}{6}+\mathcal{O}(x^4)$.

# Chapter 4

# Faster Training: Algorthmic Speed-up

In this section, we will discuss several algorithmic tricks that make Gradient Boosting fast to train. Also, techniques that enhance the generalizability of boosted trees will be introduced.

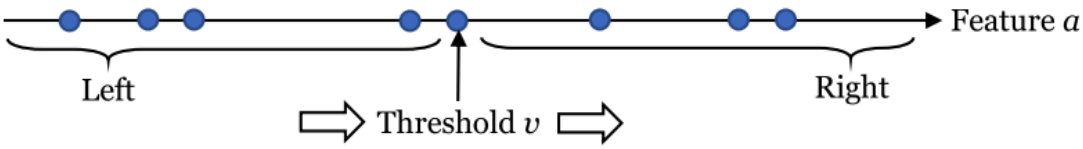



Figure 4.1: We scan the training data from left to right, considering each value as a threshold candidate. [1]

## 4.1 Weighted Quantile Sketch and Feature Quantization

### 4.1.1 Approximate splitting with Quantile Sketch

During one boosting step, at each node we will propose a split candidate by scanning through each feature and each threshold value. The reduction in loss function as a result of the split will be evaluated to assess whether the split candidate should be kept or discarded.

The naive approach is highly inefficient. We would first sort the feature and linear scan all possible unique values. It is unnecessary to do so, since there is not much difference splitting at $x = 5$ and $x = 5.00001$.

To speed up the algorithm, instead of considering all possible unique values to split on, we only consider a fixed number of quantile points as our splitting candidates. We will build separate quantile sketches for each feature.

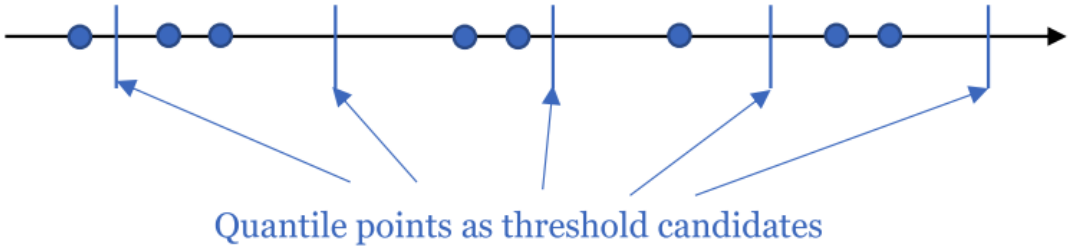



Figure 4.2: We restrict splitting candidates to only quantile points [1]

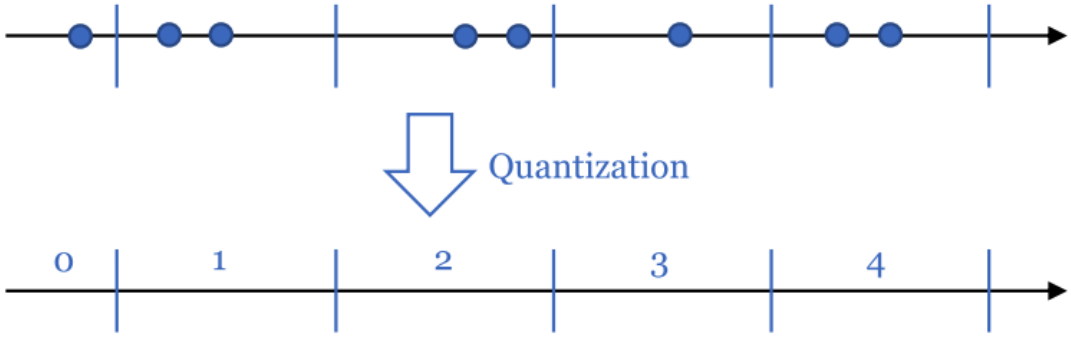



Figure 4.3: Quantize numerical feature values into quantiles. [1]

### 4.1.2 Feature Quantization

By leveraging quantile sketch, we can now place values of each feature to which quantile it belongs. All data points whose values for a particular feature fall between two consecutive quantile points will go to the same quantile index. As long as the value of the same feature of two data points are both placed between two consecutive quantile points, they are considered the same. As a result, we only need to remember the quantile indices instead of the actual feature value. Also, comparison at each split is faster, since it is faster to compare integers than to compare floats.

### 4.1.3 Weighted Quantile

We introduce sample weights to encourage hard-to-learn samples to be placed to the same quantile. As a result, a tree is more likely to make a split and isolate samples where the loss is high. In the original paper [2]:

$$\begin{aligned}\tilde{\mathcal{L}} + \text{constant} &= \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i) + \frac{g_i^2}{2h_i}] \\ &= \sum_{i=1}^{n}\frac{1}{2}h_i[f_t^2(\mathbf{x}_i) + 2\frac{f_t(\mathbf{x}_i)g_i}{h_i} + (g_i/h_i)^2] \\ &= \sum_{i=1}^{n}\frac{1}{2}h_i[f_t(x_i) - (-g_i/h_i)]^2\end{aligned} \tag{4.1}$$

where $-g_i/h_i$ is exactly the prediction to $f_t(x_i)$ when $\lambda = 0$, hence the sum could be thought of as the weighted mean squared error (MSE), where the weight is determined by the hessian evaluated at each data point.

**Definition 2.** *(Weighted Quantile) Let the multi-set* $\mathcal{D} = \{(x_{1k}, h_1), (x_{2k}, h_2), ..., (x_{nk}, h_n)\}$ *represent the k-th feature and hessian evaluated at each data point. We can define a quantile function* $r_k : \mathbb{R} \to [0, 1]$ *as*

$$r_k(\hat{x}) = \frac{1}{\sum_{(x,h)\in\mathcal{D}} h} \sum_{(x,h)\in\mathcal{D}, x<\hat{x}} h$$

## 4.2 Histogram Aggregation of Gradient Statistics

This is also where the `hist` in `histGBODT` comes from. One crucial step for determining leaf weight is the computation of sum of gradients and hessians $G_j, H_j$, and $\boldsymbol{L}_j$. Recall that, for the purpose of split finding, all data points in the same quantile are considered as identical. To speed up split finding process, we should sum up their first and second order gradients and store them into the histogram bin associated with the particular quantile. At each split, the left and right child will inherit parent node's histogram of gradient statistics for subsequent splitting. We can build two histogram with the effort of one, since we can get the other one by subtracting the histogram already built from parent node's histogram.

## 4.3 Tree-growing Strategies

We maintain a priority queue of nodes that will be split next. The node that provides the greatest global reduction in loss will be at the top of the priority queue. When the node at the top of the priority queue is popped, we propose the best splitting candidate and evaluate the reduction in loss after the split. If there is no positive reduction, the node will be finalized as a leaf node and no splitting will be made. Otherwise, calculate the loss reduction after splitting for each of the child

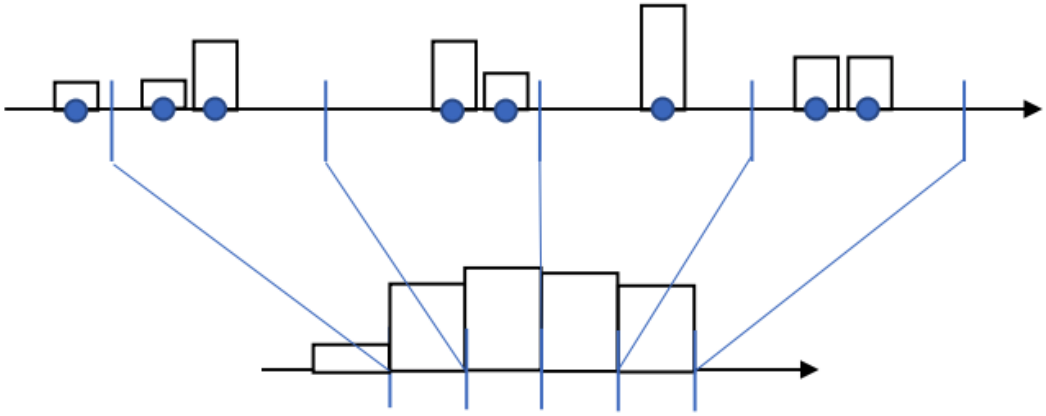


Figure 4.4: Aggregation of gradient statistics along one feature. Only one bar is displayed for simplicity. In reality, there are three bars, storing sum of gradients and hessians w.r.t. $w$ (scalars) and second-order mixed gradient $\nabla_{\boldsymbol{w},\boldsymbol{\delta}}\tilde{\mathcal{L}}$ (a vector) at each bar.

node, decide whether they will be added to the priority queue.

**Sparsity-Aware Split Finding**: When the training data contains missing values, we cannot compare the threshold hold value at each tree node to the missing value. When splitting the instance set into two, the data point is attempted to split to the left and split to the right. The direction that gives the greater amount of loss reduction is selected. See Appendix A for full split-finding algorithm detail.

We have found that, to compute the loss function evaluated at the optimal parameter, we need to invert the Hessian matrix, which implies that we need to have access to the inverse of the Schur complement $\boldsymbol{V} - \boldsymbol{L}\tilde{\boldsymbol{H}}_{\boldsymbol{\lambda}}\boldsymbol{L}^{\intercal}$. Computing the inverse would be costly, since we will calculate the inverse at each candidate split point for each

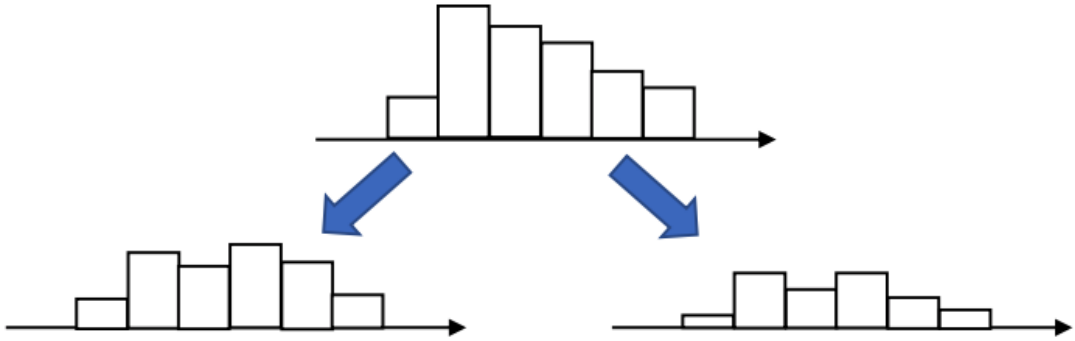


Figure 4.5: Splitting gradient statistics to left and right child. The sum of histograms of left and right child equals the histogram of parent node.

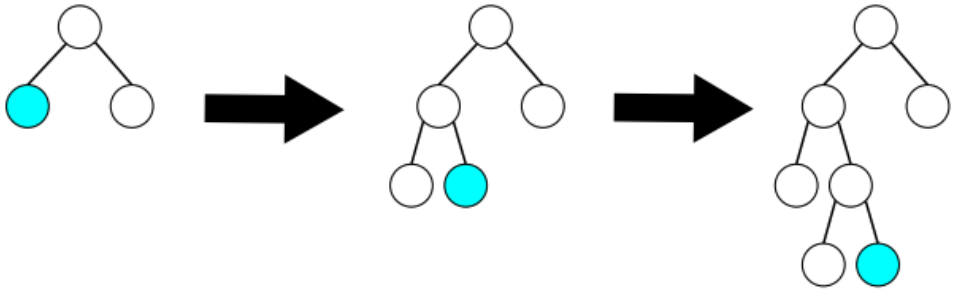


Figure 4.6: Loss-optimal tree growth strategy

feature. Instead, we could leverage the Sherman-Morrison Formula to compute the inverse, which only consists of inverting a diagonal matrix and matrix multiplication. Suppose we are proposing a split candidate at leaf index $k$:

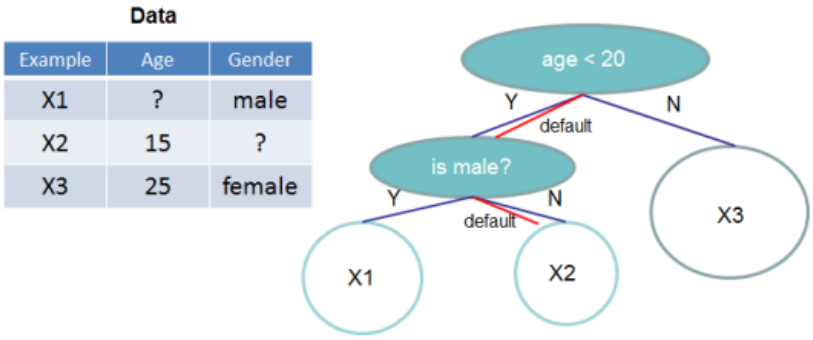



Figure 4.7: When a data point has missing values and they are compared against tree node thresholds, it will be split to the direction that gives rise to greater loss reduction. [2]

Before split:

$$
\begin{aligned}
(\boldsymbol{V} - \boldsymbol{L}\tilde{\boldsymbol{H}}_{\boldsymbol{\lambda}}\boldsymbol{L}^{\intercal})^{-1} &= (\boldsymbol{V} - \sum_{j=1}^{J} \frac{\boldsymbol{L}_j \boldsymbol{L}_j^{\intercal}}{H_j + \lambda})^{-1} \\
&= (\boldsymbol{V} - \sum_{j \neq k}^{J} \frac{\boldsymbol{L}_j \boldsymbol{L}_j^{\intercal}}{H_j + \lambda} - \frac{\boldsymbol{L}_k \boldsymbol{L}_k^{\intercal}}{H_k + \lambda})^{-1} \\
&= (\boldsymbol{A} - \frac{\boldsymbol{L}_k \boldsymbol{L}_k^{\intercal}}{H_k + \lambda})^{-1} \\
&= \boldsymbol{A}^{-1} + \frac{\boldsymbol{A}^{-1} \boldsymbol{L}_k \boldsymbol{L}_k^{\intercal} \boldsymbol{A}^{-1}}{H_k + \lambda - \boldsymbol{L}_k^{\intercal} \boldsymbol{A}^{-1} \boldsymbol{L}_k}
\end{aligned}
\tag{4.2}
$$

After split:

$$
\begin{aligned}
(\boldsymbol{V} - \boldsymbol{L}\tilde{\boldsymbol{H}}_{\boldsymbol{\lambda}}\boldsymbol{L}^{\intercal})^{-1} &= (\boldsymbol{V} - \sum_{j \neq k}^{J} \frac{\boldsymbol{L}_j \boldsymbol{L}_j^{\intercal}}{H_j + \lambda} - \frac{\boldsymbol{L}_{k_1} \boldsymbol{L}_{k_1}^{\intercal}}{H_{k_1} + \lambda} - \frac{\boldsymbol{L}_{k_2} \boldsymbol{L}_{k_2}^{\intercal}}{H_{k_2} + \lambda})^{-1} \\
&= (\boldsymbol{A} - \frac{\boldsymbol{L}_{k_1} \boldsymbol{L}_{k_1}^{\intercal}}{H_{k_1} + \lambda} - \frac{\boldsymbol{L}_{k_2} \boldsymbol{L}_{k_2}^{\intercal}}{H_{k_2} + \lambda})^{-1}
\end{aligned}
\tag{4.3}
$$

where

$$\boldsymbol{L}_k = \boldsymbol{L}_{k_1} + \boldsymbol{L}_{k_2}$$

$$H_k = H_{k_1} + H_{k_2}$$

Note that at root node, the matrix $\boldsymbol{A}$ above is just the diagonal matrix $\tilde{\boldsymbol{V}}$ in our hessian matrix, which is trivial to invert. Hence we iteratively have access to the Schur complement by invoking Sherman-Morrison each time. To compute the inverse after the split, invoke Sherman-Morrison twice.

## 4.4 Gradient-Aware Pruning and Adaptive Learning Rate

We leverage out-of-bag samples to make our learned model more robust to overfitting. The idea is to reduce model complexity when the model does not generalize well to validation samples not used during training.

### 4.4.1 Gradient-Aware Pruning Using Out-of-Bag Samples

While growing a decision tree is a top-down procedure, pruning a tree is bottom-up. Without pruning, decision trees will be grown to the maximum height, unless there is not enough samples at each leaf, or splitting does not reduce loss evaluated

in-sample. A loss-optimal greedy approach to tree growth is prone to over-fitting. Instead, after a decision tree has been built, we introduce validation samples to prune leaves that do not generalize well to new samples.

During pruning, the loss will be computed on the validation samples and compared between child nodes and parent node. The leaf will be pruned if keeping the left and right child does not decrease the out-of-bag loss. The deeper the decision tree grows, the fewer samples a leaf will contain, thus making the leaf weight estimates have higher variances. The pruning procedure robustify the training process by reducing the model complexity.

When implementing gradient-aware pruning, the histograms of gradient statistics are built on the validation samples top-down. Then, at each pair of children leaf nodes, from the out-of-bag gradient statistics we can evaluate loss before splitting, and after splitting respectively.

### 4.4.2 Adaptive Learning Rate Using Out-of-Bag Samples

Traditionally, in the most popular gradient boosting implementations, learning rate is set to a fix number and it is treated as a hyperparameter. Its optimal value is found via grid-search or random-search. Usually, smaller learning rate gives better performance. Inspired by adaptive learning rate commonly used in deep learning, the

learning rate is reduced after every few iterations. During training, the benefit of each tree provides will diminish. Thus quite often the last few trees do not provide much predictive power.

Line search in optimization provides us with a solution to picking the optimal learning rate. The learning rate is computed using gradient statistics on out-of-bag samples to improve generalizability. As a result, the tree that does not have much predictive power will receive a learning rate close to 0, thus encouraging pruning at the whole tree-level.

Formally, we would like to pick the learning-rate $\nu$ at each iteration right after the tree is built and pruned. $\mathcal{D}_{\text{val}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{val}}}$ is the out-of-bag (validation) samples.

$$\mathcal{L}^{\text{OOB}}(\nu, \boldsymbol{w}^{(t)}, \boldsymbol{\delta}^{(t)}) = \sum_{i=1}^{n_{\text{val}}} l_{\text{ord}}(y_i, F^{(t-1)}(\mathbf{x}_i) + \nu f_t(\mathbf{x}_i), \boldsymbol{\theta}^{(t-1)} + \nu\boldsymbol{\delta}^{(t)}) \tag{4.4}$$

For simplicity, we re-write this in terms of our previous best estimate, $\boldsymbol{z}^{(t-1)}$, consisting of the prediction of the target on out-of-bag samples, and $\boldsymbol{\theta}^{(t-1)}$; a descent direction $\boldsymbol{\beta}^{(t)}$, consisting of leaf weights $\boldsymbol{w}^{(t)}$ and $\boldsymbol{\delta}^{(t)}$; and step size $\nu$.

$$\mathcal{L}^{\text{OOB}}(\boldsymbol{z}^{(t-1)} + \nu\boldsymbol{\beta}^{(t)}) \simeq \mathcal{L}^{\text{OOB}}(\boldsymbol{z}^{(t-1)}) + \nu\boldsymbol{g}^{\text{OOB}\intercal}\boldsymbol{\beta}^{(t)} + \frac{\nu^2}{2}\boldsymbol{\beta}^{(t)\intercal}\tilde{\boldsymbol{H}}^{\text{OOB}}\boldsymbol{\beta}^{(t)} \tag{4.5}$$

$$\nu^* = -\frac{\boldsymbol{g}^{\text{OOB}\intercal}\boldsymbol{\beta}^{(t)}}{\boldsymbol{\beta}^{(t)\intercal}\tilde{\boldsymbol{H}}^{\text{OOB}}\boldsymbol{\beta}^{(t)}} \tag{4.6}$$

Here the definition of $\boldsymbol{g}^{\text{OOB}}$ and $\tilde{\boldsymbol{H}}^{\text{OOB}}$ is the same to the counterpart during training in-sample. The only difference is the set of samples being evaluated on. In practice

the over-fitting robust out-of-bag learning rate estimate is clipped to the range $[0, 1]$.

# Chapter 5

# Experiments

We conduct experiments on both simulated and real-world datasets. We start with simple synthetic datasets to understand the behavior of the algorithm. Then, we report our model's performance on real-world data sets.

We employ two evaluation metrics as measures of predictive performance in ordinal classification tasks:

- *Mean Absolute Error* (MAE): $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$
- *Mean Zero-One Error* (ZOE): $\frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i \neq y_i)$

## 5.1 Artificial Data

Figure 5.1 presents teh behavior of `HistGBODT` on 3 2-D syntehic datasets. Each dataset is generated from a Gaussian distribution divided into 3 ordinal ranks using

non-linear decision boundaries. The second dataset is labelled the same way as the first dataset, except the data points closest to the decision boundaries are randomized with probability $\frac{1}{2}$. The prediction accuracy on the three datasets are: 99.73%, 94.41%, and 98.01%, respectively. The simulation results show that the algorithm is working reasonably well on this task.

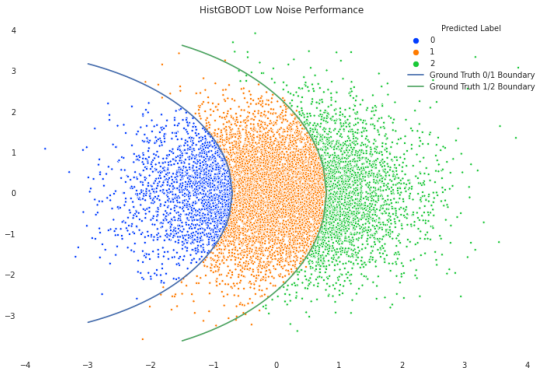
## 5.2 Benchmark Data

We performed experiments with 16 benchmark datasets that were used by Chu and Keerthi [32] [1]. The target values were discretized into 10 equally-sized quantiles. These bins thus divide the range of the target valeus into 10 ordinal ranks. The input vectors were normalized to zero mean and unit variance coordinate-wise. To show the advantage of explicitly modeling the ordinal nature of the response variables, we also employed the standard gradient boosting trained using the multi-class cross entropy loss, and using the mean-squared error with the predicted values rounded to the nearest ordinal scale. We randomly partitioned each dataset into training/testing splits as specified in table 5.1. The partition was repeated 20 times.

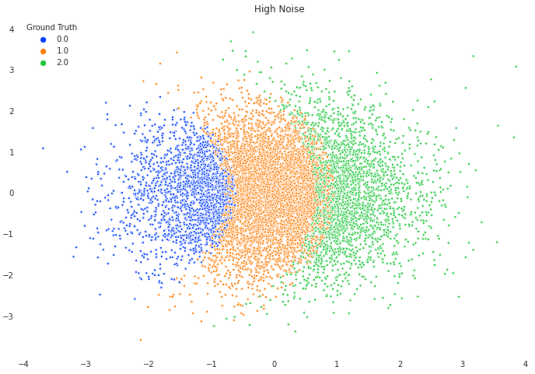
From the testing set results averaged over 20 trials, we could see a clear advantage of minimizing the All-Threshold loss over traditional methods such as multi-class cross entropy and mean-squared error. To determine statistical significance, we conduct

[1] Datasets available for download at: `http://www.gatsby.ucl.ac.uk/~chuwei/ordinalregression.html`

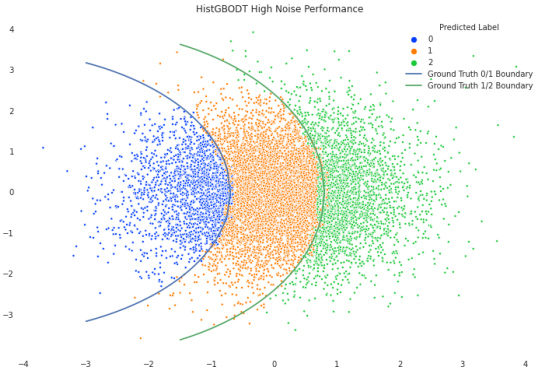

(a) Artificial dataset I with low noise



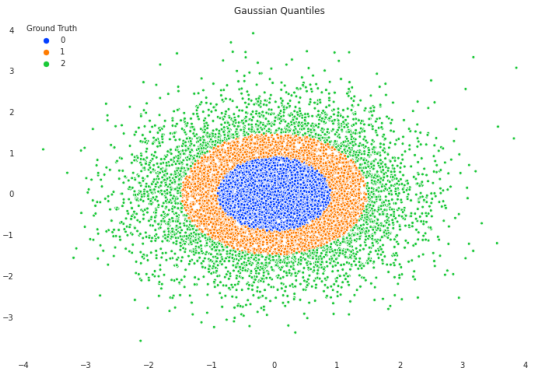

(b) Artificial dataset I prediction



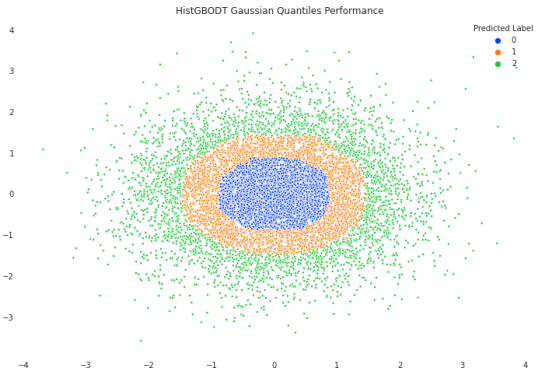

(c) Artificial dataset II with high noise




(d) Artificial dataset II prediction




(e) Artificial dataset III Gaussian Quantiles




(f) Artificial dataset III prediction

Figure 5.1: `HistGBODT` performance on synthetic datasets.

| | Datasets | Attributes(Numeric,Nominal) | Training Instances | Instances for Test |
|---|---|---|---|---|
| | Diabetes | 2(2,0) | 30 | 13 |
| | Pyrimidines | 27(27,0) | 50 | 24 |
| | Triazines | 60(60,0) | 100 | 86 |
| | Wisconsin Breast Cancer | 32(32,0) | 130 | 64 |
| Set I | Machine CPU | 6(6,0) | 150 | 59 |
| | Auto MPG | 7(4,3) | 200 | 192 |
| | Boston Housing | 13(12,1) | 300 | 206 |
| | Stocks Domain | 9(9,0) | 600 | 350 |
| | Abalone | 8(7,1) | 3177 | 1000 |
| | Bank Domains(1) | 8(8,0) | 5461 | 2731 |
| | Bank Domains(2) | 32(32,0) | 5461 | 2731 |
| | Computer Activity(1) | 12(12,0) | 5461 | 2731 |
| Set II | Computer Activity(2) | 21(21,0) | 5461 | 2731 |
| | Califomia Housing | 8(8,0) | 10427 | 5213 |
| | Census Domains(1) | 8(8,0) | 11189 | 5595 |
| | Census Domains(2) | 16(16,0) | 11189 | 5595 |

Table 5.1: Datasets and their characteristics. “Attributes” state the number of numerical and nominal attributes. “Training Instances” and “Instances for Test” specify the size of training/test partitions.

| | Mean zero-one error | | | Mean absolute Error | | |
|---|---|---|---|---|---|---|
| Data | AT | MultiClass | MSE | AT | MultiClass | MSE |
| Diabetes | 69.23% | 69.23% | 69.23% | 1.154 | 1.154 | 1.154 |
| Pyrimidines | **70.83%** | 75.62% | 77.92% | **1.240** | 1.577 | 1.383 |
| Triazines | 67.85% | **67.79%** | 71.05% | **1.112** | 1.319 | 1.153 |
| Wisconsin Breast Cancer | 86.33% | **85.78%** | 88.83% | **2.123** | 2.805 | 2.438 |
| Machine CPU | **31.69%** | 41.10% | 37.46% | **0.4483** | 0.939 | 0.5161 |
| Auto MPG | **45.83%** | 50.86% | 47.24% | **0.5464** | 0.6331 | 0.5576 |
| Boston Housing | **43.91%** | 45.02% | 44.22% | **0.5204** | 0.5680 | 0.5235 |
| Stocks Domain | 22.06% | **21.29%** | 21.84% | 0.2261 | **0.2164** | 0.2257 |
| Abalone | **42.34%** | 48.40% | 44.68% | **0.5212** | 0.7552 | 0.5384 |
| Bank Domains(1) | 72.28% | **69.17%** | 78.4% | **1.456** | 1.788 | 1.530 |
| Bank Domains(2) | **38.81%** | 44.75% | 39.47% | **0.4094** | 0.5145 | 0.4171 |
| Computer Activity(1) | **51.45%** | 52.28% | 57.66% | **0.7039** | 0.8177 | 0.7929 |
| Computer Activity(2) | **41.60%** | 47.05% | 45.18% | **0.4687** | 0.5881 | 0.5134 |
| Califomia Housing | **46.94%** | 50.68% | 49.18% | **0.5683** | 0.6928 | 0.6016 |
| Census Domains(1) | **60.58%** | 62.59% | 66.83% | **0.9296** | 1.410 | 0.998 |
| Census Domains(2) | 62.25% | **60.25%** | 66.93% | **0.9793** | 1.130 | 1.022 |

Table 5.2: Testing set results: the bold font indicates the cases in which the average value is the lowest in the results of minimizing the three loss functions.

| | Mean zero-one error | | Mean absolute error | |
|---|---|---|---|---|
| Null hypothesis | AT < MultiClass | AT < MSE | AT < MultiClass | AT < MSE |
| Test Statistic | 20 | 1 | 1 | 1 |
| P-value | 0.01155 | 0.0004026 | 0.0004026 | 0.0004026 |

Table 5.3: Nonparametric Wilcoxon signed-rank test results

Wilcoxon signed-rank tests with a p-value threshold of 0.025 (here, a Dunn–Šidák correction is applied for multiple comparison). We observe that the performance gains are statistically significant for all comparisons.

# Chapter 6

# Conclusion and Future Directions

## 6.1 Conclusion

Ordinal classification is an important supervised learning task but has not been widely implemented in popular machine learning frameworks such as `scikit-learn` or `XGBoost`, etc. In this paper, we propose minimizing the All-Threshold loss function combined with the powerful gradient boosting as a solution. To optimize the objective, we encounter the issue of non-invertibility of the Hessian matrix and adopted the Majorization-Minimization (MM) algorithm to perform Newton-Raphson updates. Experiments on benchmark datasets indicates the generalization performance is competitive and statistically significantly better than existing approaches such as minimizing the cross entropy and mean-squared error (MSE).

## 6.2 Future Directions

Our implementation is under the `scikit-learn` framework, using `Cython` for native `C` speed performance and parallel computing. However, integration with highly optimized gradient boosting package such as `XGBoost` and `CatBoost` could be even more desirable due to access to GPU computing.

Traditionally, decision trees split along directions parallel to the coordinate axes. In high-dimensional space when classes seem to be inseparable along any single coordinate direction, or decision boundaries are highly irregular and non-linear, such splits require very deep trees with complicated structures, such increasing variance and overfitting. Several work has proposed to split along a linear combination of coordinates (Oblique Decision Trees, [33–37]). Integrating All-Threshold loss with boosted oblique decision trees could be an interesting future research direction.

# Bibliography

[1] H. Cho, "Speeding up gradient boosting for training and prediction."

[2] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," p. 785–794, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[3] T. Gneiting and A. E. Raftery, "Weather forecasting with ensemble methods," *Science*, vol. 310, no. 5746, pp. 248–249, 2005. [Online]. Available: https://science.sciencemag.org/content/310/5746/248

[4] T. Gneiting and M. Katzfuss, "Probabilistic forecasting," *Annual Review of Statistics and Its Application*, vol. 1, no. 1, pp. 125–151, 2014. [Online]. Available: https://doi.org/10.1146/annurev-statistics-062713-085831

[5] J. Bennett, C. Elkan, B. Liu, P. Smyth, and D. Tikk, "Kdd cup and workshop 2007," *SIGKDD Explor. Newsl.*, vol. 9, no. 2, p. 51–52, Dec. 2007. [Online]. Available: https://doi.org/10.1145/1345448.1345459

[6] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah,

R. Herbrich, S. Bowers, and et al., "Practical lessons from predicting clicks on ads at facebook," p. 1–9, 2014. [Online]. Available: https://doi.org/10.1145/2648584.2648589

[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," pp. 3146–3154, 2017. [Online]. Available: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[8] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *CoRR*, vol. abs/1810.11363, 2018. [Online]. Available: http://arxiv.org/abs/1810.11363

[9] A. V. Dorogush, A. Gulin, G. Gusev, N. Kazeev, L. O. Prokhorenkova, and A. Vorobev, "Catboost: unbiased boosting with categorical features," *CoRR*, vol. abs/1706.09516, 2017. [Online]. Available: http://arxiv.org/abs/1706.09516

[10] P. McCullagh, "Regression models for ordinal data," *Journal of the Royal Statistical Society*, vol. 42, no. 2, pp. 109–142, 1980. [Online]. Available: http://www.jstor.org/stable/2984952

[11] J. A. Anderson, "Regression and ordered categorical variables," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 46, no. 1, pp. 1–22, 1984. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1984.tb01270.x

BIBLIOGRAPHY

[12] A. Agresti, "Modelling ordered categorical data: recent advances and future challenges," *Statistics in Medicine*, vol. 18, no. 17-18, p. 2191–2207, 1999.

[13] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, p. 933–969, Dec. 2003.

[14] K. Dembczynski, W. Kotlowski, and R. Slowinski, "Ordinal classification with decision rules," 2007.

[15] J. D. M. Rennie, "Loss functions for preference levels: Regression with discrete ordered labels," pp. 180–186, 2005.

[16] K. Crammer and Y. Singer, "Pranking with ranking," pp. 641–647, 2002. [Online]. Available: http://papers.nips.cc/paper/2023-pranking-with-ranking.pdf

[17] A. Shashua and A. Levin, "Ranking with large margin principle: Two approaches," pp. 961–968, 2003. [Online]. Available: http://papers.nips.cc/paper/2269-ranking-with-large-margin-principle-two-approaches.pdf

[18] W. Chu and S. S. Keerthi, "New approaches to support vector ordinal regression," p. 145–152, 2005. [Online]. Available: https://doi.org/10.1145/1102351.1102370

[19] H.-T. Lin and L. Li, "Large-margin thresholded ensembles for ordinal regression: Theory and practice," pp. 319–333, 2006.

[20] L. Li and H. tien Lin, "Ordinal regression by extended binary classification," pp. 865–872, 2007. [Online]. Available: http://papers.nips.cc/paper/3125-ordinal-regression-by-extended-binary-classification.pdf

[21] F. Pedregosa, F. Bach, and A. Gramfort, "On the consistency of ordinal regression methods," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 1769–1803, Jan. 2017.

[22] J. Friedman, "Another approach to polychotomous classification, technical report," 1996.

[23] T. H. . R. Tibshirani, "Classification by pairwise coupling," *Annals of Statistics*, vol. 26, no. 2, pp. 451–471, 1998.

[24] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *J. Mach. Learn. Res.*, vol. 5, p. 101–141, Dec. 2004.

[25] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[26] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," 1984.

[27] T.-T. Lu and S.-H. Shiou, "Inverses of 2 x 2 block matrices," *Computers and Mathematics with Applications*, vol. 43, no. 1-2, pp. 119–129, 2002.

[28] T. Chen, S. Singh, B. Taskar, and C. Guestrin, "Efficient Second-Order Gradient Boosting for Conditional Random Fields," vol. 38, pp. 147–155, 09–12 May 2015. [Online]. Available: http://proceedings.mlr.press/v38/chen15b.html

[29] J. de Leeuw and K. Lange, "Sharp quadratic majorization in one dimension." 2011.

[30] D. Böhning and B. Lindsay, "Monotonicity of quadratic-approximation algorithms," *Annals of the Institute of Statistical Mathematics*, vol. 40, pp. 641–663, 02 1988.

[31] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. USA: Society for Industrial and Applied Mathematics, 2000.

[32] W. Chu and Z. Ghahramani, "Gaussian processes for ordinal regression." *Journal of Machine Learning Research*, vol. 6, pp. 1019–1041, 07 2005.

[33] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A%3A1010933404324

[34] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[35] B. Menze, B. Kelm, D. Splitthoff, U. Köthe, and F. Hamprecht, "On oblique random forests," pp. 453–469, 09 2011.

[36] T. M. Tomita, M. Maggioni, and J. T. Vogelstein, "Randomer forests," *ArXiv*, vol. abs/1506.03410, 2015.

[37] T. M. Tomita, J. Browne, C. Shen, J. Chung, J. L. Patsolic, B. Falk, J. Yim, C. E. Priebe, R. Burns, M. Maggioni, and J. T. Vogelstein, "Sparse projection oblique randomer forests," *ArXiv*, vol. abs/1506.03410, 2019.

[38] L. Devroye, L. Györfi, and G. Lugosi, "A probablistic theory of pattern recognition," *NewYork: Springer Verlag*, vol. 31, 01 1996.

[39] G. Biau, L. Devroye, and G. Lugosi, "Consistency of random forests and other averaging classifiers," *Journal of Machine Learning Research*, vol. 9, pp. 2015–2033, 09 2008.

[40] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[41] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[42] P. Bickel and K. Doksum, *Mathematical Statistics: Basic Ideas and Selected Topics.*, 01 2007, vol. 56.

# Appendix A

**Algorithm 1** Histogram-based Gradient Boosting

**Input:** Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$

$L_2$ regularization $\lambda$

Learning rate $\nu$

Leaf penalty $\gamma$

Maximum iterations $T$

Maximum depth of tree $D$

Maximum number of nodes $J$

Minimum number of training example in each leaf $n_{\text{min}}$

If pruning will be performed $\mathbb{1}_{\text{prune}}$

If adaptive learning rate will be selected $\mathbb{1}_{\text{adaptive}}$

Number of training observations to be set aside for pruning/adaptive learning rate selection $n_{\text{val}}$

**Output:** Collection of trees $\{f_t\}_{t\geq 1, t\in\mathbb{N}}$, Category threshold parameter $\boldsymbol{\theta}$

Shrinkage factor on each tree $\{\nu_t\}_{t\geq 1, t\in\mathbb{N}}$

**if** $\mathbb{1}_{\text{prune}} = 1$ or $\mathbb{1}_{\text{adaptive}} = 1$ **then**

  $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}} = \textsc{TrainValidationSplit}(\mathcal{D}, n_{\text{val}})$

**else**

  $\mathcal{D}_{\text{train}} = \mathcal{D}$, $\mathcal{D}_{\text{val}} = \emptyset$

**end if**

$\dot{\mathbf{X}}_{\text{train}}, \dot{\mathbf{X}}_{\text{val}} = \textsc{FeatureQuantize}(\mathbf{X}_{\text{train}}, \mathbf{X}_{\text{val}})$

$\tilde{y}^{(0)}, \boldsymbol{\theta}^{(0)} = \underset{\tilde{y}, \boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i \in \mathcal{D}_{\text{train}}} l_{ord}(y_i, \tilde{y}, \boldsymbol{\theta})$

$\tilde{\mathbf{y}}^{(0)}_{\text{train}} = \tilde{\mathbf{y}}^{(0)}_{\text{val}} = \tilde{y}^{(0)}\mathbf{1}$

**for** $t = 1 : T$ **do**

  $\{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{train}}} = \textsc{UpdateGradientStatistics}(\mathbf{y}_{\text{train}}, \tilde{\mathbf{y}}^{(t-1)}_{\text{train}}, \boldsymbol{\theta}^{(t-1)})$

  **if** $\mathbb{1}_{\text{prune}} = 1$ or $\mathbb{1}_{\text{adaptive}} = 1$ **then**

    $\{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{val}}} = \textsc{UpdateGradientStatistics}(\mathbf{y}_{\text{val}}, \tilde{\mathbf{y}}^{(t-1)}_{\text{val}}, \boldsymbol{\theta}^{(t-1)})$

  **else**

    $\{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{val}}} = \emptyset$

  **end if**

  $f_t, \boldsymbol{\delta}^{(t)} = \textsc{GrowTree}(\dot{\mathbf{X}}_{\text{train}}, \{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{train}}}, J, D, \lambda, \gamma, n_{\min})$

  **if** $\mathbb{1}_{\text{prune}} = 1$ **then**

    $f_t, \boldsymbol{\delta}^{(t)} = \textsc{DoPrune}(f_t, \{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{val}}})$

  **end if**

  **if** $\mathbb{1}_{\text{adaptive}} = 1$ **then**

    $\nu_t = \textsc{ComputeAdaptiveLearningRate}(f_t, \{\left(g_i, h_i, (L_{ik})_{k=0}^{K-2}\right)\}_{i \in \mathcal{D}_{\text{val}}})$

$\nu_t = \textsc{Clip}(\nu_t, 0, \nu)$

**else**

$\nu_t = \nu$

**end if**

$\tilde{\mathbf{y}}_{\text{train}}^{(t)} = \tilde{\mathbf{y}}_{\text{train}}^{(t-1)} + \nu_t f_t(\dot{\mathbf{X}}_{\text{train}})$

$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} + \nu_t \boldsymbol{\delta}^{(t)}$

**if** $\mathbb{1}_{\text{prune}} = 1$ or $\mathbb{1}_{\text{adaptive}} = 1$ **then**

$\tilde{\mathbf{y}}_{\text{val}}^{(t)} = \tilde{\mathbf{y}}_{\text{val}}^{(t-1)} + \nu_t f_t(\dot{\mathbf{X}}_{\text{val}})$

**end if**

$\mathbb{1}_{\text{EarlyStop}} = \textsc{CheckEarlyStopping}(\mathbf{y}_{\text{train}}, \tilde{\mathbf{y}}_{\text{train}}, \mathbf{y}_{\text{val}}, \tilde{\mathbf{y}}_{\text{val}}, \nu_t)$

**if** $\mathbb{1}_{\text{EarlyStop}} = 1$ **then**

Break

**end if**

**end for**

**end function**

**Algorithm 2** Grow a tree at iteration $t$

**Input:** Quantized training observations $\dot{\mathbf{X}}_{\text{train}}$

Gradient Statistics $\{\big(g_i, h_i, (L_{ik})_{k=0}^{K-2}\big)\}_{i \in \mathcal{D}_{\text{train}}}$

$L_2$ regularization $\lambda$

Leaf penalty $\gamma$

Maximum depth of tree $D$

Maximum number of nodes $J$

Minimum number of training example in each leaf $n_{\min}$

**Output:** Tree $f_t$ with loss optimal structure $q_t$

Category threshold parameter Newton Raphson step $\boldsymbol{\delta}^{(t)}$

$I_1 = \{1, ..., n_{\text{train}}\}$ ▷ Initialize tree structure with a single node

$\Big(m_1, v_1, \mathbb{1}_{\text{MissingGoesLeft}}, \Delta\tilde{\mathcal{L}}^{(t)}(I_1)\Big) = \textsc{EvaluateSplit}(I_1)$

$d_1 = 1$ ▷ Depth level of node 1

$S = \texttt{PriorityQueue}(\big[\{1 : \Delta\tilde{\mathcal{L}}^{(t)}(I_1)\}\big])$ ▷ Set of leaf nodes to be split

LeafSet = `HashSet({})`

**while** $j = 1; j \leq J$ **do**

  $u = S.\texttt{pop()}$ ▷ Retrieve node with the largest loss reduction after split

  **if** $d_u < D$ **then**

    $I_{j+1} = \{i \in I_u : \dot{\mathbf{x}}_{im_u} < \tau_u\}$ ▷ Split $I_u$ using the $m_u^{\text{th}}$ feature & threshold $\tau_u$

    $I_{j+2} = \{i \in I_u : \dot{\mathbf{x}}_{im_u} \geq \tau_u\}$

    $d_{j+1}, d_{j+2} = d_u + 1, d_u + 1$ ▷ Assign depth level of new nodes

12: **if** $d_j + 1 = D$ **then**

13: LeafSet.`add`($I_{j+1}$)

14: LeafSet.`add`($I_{j+2}$)

15: **else**

16: **if** $|I_{j+1}| \geq n_{\text{min}}$ **then**

17: $\left(m_{j+1}, \tau_{j+1}, \mathbb{1}_{\text{MissingGoesLeft}}, \Delta\tilde{\mathcal{L}}^{(t)}(I_{j+1})\right) = \textsc{EvaluateSplit}(I_{j+1})$

18: $S$.`add`($\{j+1 : \Delta\tilde{\mathcal{L}}^{(t)}(I_{j+1})\}$)

19: $j = j + 1$

20: **else**

21: LeafSet.`add`($I_{j+1}$)

22: **end if**

23: **if** $|I_{j+2}| \geq n_{\text{min}}$ **then**

24: $\left(m_{j+2}, \tau_{j+2}, \mathbb{1}_{\text{MissingGoesLeft}}, \Delta\tilde{\mathcal{L}}^{(t)}(I_{j+2})\right) = \textsc{EvaluateSplit}(I_{j+2})$

25: $S$.`add`($\{j+2 : \Delta\tilde{\mathcal{L}}^{(t)}(I_{j+2})\}$)

26: $j = j + 1$

27: **else**

28: LeafSet.`add`($I_{j+2}$)

29: **end if**

30: **end if**

31: **end if**

32: **if** $j == J$ **then**

LeafSet.`add`($I_{j+1}$)

LeafSet.`add`($I_{j+2}$)

**while** $|S| \neq 0$ **do**

LeafSet.`add`($S$.`pop`())

**end while**

**end if**

**end while**

($\boldsymbol{w}^{(t)}, \boldsymbol{\delta}^{(t)}$) =FinalizeLeaves(LeafSet) ▷ Leaf weights & threshold Newton step

**Algorithm 3** EVALUATESPLIT($I_j$) ▷ Compute greatest loss reduction for a node

$\left(G_j, H_j, \boldsymbol{L}_j\right) = \sum_{i \in I_j} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

**for** feature $m$ in $\mathbf{X}_{\text{train}}$ **do**

  $M_m = \{i \in I_j : \mathbf{X}_{im} = \emptyset\}$ ▷ Provision for missing values

  $\left(G_m^{(\emptyset)}, H_m^{(\emptyset)}, \boldsymbol{L}_m^{(\emptyset)}\right) = \sum_{i \in M_m} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

  $\mathcal{T}_m = \{\text{set of quantile points for feature m}\}$

  **for** $\tau \in \mathcal{T}_m$ **do** ▷ Consider each threshold candidate

    $\text{LeftSet}_{m,\tau} = \{i \in I_j : \mathbf{X}_{im} < \tau\}$

    $\text{RightSet}_{m,\tau} = \{i \in I_j : \mathbf{X}_{im} \geq \tau\}$

    $\left(G_{m,\tau}^{(\text{left})}, H_{m,\tau}^{(\text{left})}, \boldsymbol{L}_{m,\tau}^{(\text{left})}\right) = \sum_{i \in \text{LeftSet}_{m,\tau}} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

    $\left(G_{m,\tau}^{(\text{right})}, H_{m,\tau}^{(\text{right})}, \boldsymbol{L}_{m,\tau}^{(\text{right})}\right) = \sum_{i \in \text{RightSet}_{m,\tau}} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

    $\Delta\tilde{\mathcal{L}}^{(t)}(m, \tau, \mathbb{1}_{\text{MissingGoesLeft}}) = \text{COMPUTELOSSREDUCTION}\Big($

    $\left(G_{m,\tau}^{(\text{left})}, H_{m,\tau}^{(\text{left})}, \boldsymbol{L}_{m,\tau}^{(\text{left})}\right), \left(G_{m,\tau}^{(\text{right})}, H_{m,\tau}^{(\text{right})}, \boldsymbol{L}_{m,\tau}^{(\text{right})}\right), \left(G_m^{(\emptyset)}, H_m^{(\emptyset)}, \boldsymbol{L}_m^{(\emptyset)}\right), \lambda, \gamma\Big)$

  **end for**

**end for**

$(m, \tau, \mathbb{1}_{\text{MissingGoesLeft}}) = \underset{m', \tau', \mathbb{1}'_{\text{MissingGoesLeft}}}{\text{argmax}} \Delta\tilde{\mathcal{L}}^{(t)}(m', \tau', \mathbb{1}'_{\text{MissingGoesLeft}})$

**return** $(m, \tau, \mathbb{1}_{\text{MissingGoesLeft}}, \Delta\tilde{\mathcal{L}}^{(t)}(m, \tau, \mathbb{1}_{\text{MissingGoesLeft}}))$

**Algorithm 4** DoPrune($f_t, \{(g_i, h_i, (L_{ik})_{k=0}^{K-2})\}_{i \in \mathcal{D}_{\text{val}}}$)

$\{I_h^{(left)}, I_h^{(right)}\}_{h=1}^{H} = \text{CollectChildrenPairs}(f_t)$

$S_{\text{merge}} = \texttt{PriorityQueue}([])$

**for** $h = 1 : H$ **do**

  $\big(G_h^{(left)}, H_h^{(left)}, \boldsymbol{L}_h^{(left)}\big) = \sum_{i \in I_h^{(left)}} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

  $\big(G_h^{(right)}, H_h^{(right)}, \boldsymbol{L}_h^{(right)}\big) = \sum_{i \in I_h^{(right)}} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

  $\Delta_{\text{merge}}\tilde{\mathcal{L}}_h^{(t)} = \text{EvaluateMerge}\Big(\big(G_h^{(left)}, H_h^{(left)}, \boldsymbol{L}_h^{(left)}\big), \big(G_h^{(right)}, H_h^{(right)}, \boldsymbol{L}_h^{(right)}\big)\Big)$

  **if** $\Delta_{\text{merge}}\tilde{\mathcal{L}}_h^{(t)} > \texttt{tolerance}$ **then**

    $S_{\text{merge}}\texttt{.add}\big(\{(I_h^{(left)}, I_h^{(right)}) : \Delta_{\text{merge}}\tilde{\mathcal{L}}_h^{(t)}\}\big)$

  **end if**

**end for**

**while** $S_{\text{merge}} \neq \emptyset$ **do**

  $(I_h^{(left)}, I_h^{(right)}) = S_{\text{merge}}\texttt{.pop}()$; $I_h = \text{Merge}(I_h^{(left)}, I_h^{(right)})$

  **if** $I_h\texttt{.sibling.isLeaf}() = \texttt{TRUE}$ **then**

    $\big(G_h^{(sibling)}, H_h^{(sibling)}, \boldsymbol{L}_h^{(sibling)}\big) = \sum_{i \in I_h^{(sibling)}} (g_i, h_i, (L_{ik})_{k=0}^{K-2})$

    $\Delta_{\text{merge}}\tilde{\mathcal{L}}_{h'}^{(t)} = \text{EvaluateMerge}\Big(\big(G_h, H_h, \boldsymbol{L}_h\big), \big(G_h^{(sibling)}, H_h^{(sibling)}, \boldsymbol{L}_h^{(sibling)}\big)\Big)$

    **if** $\Delta_{\text{merge}}\tilde{\mathcal{L}}_{h'}^{(t)} > \texttt{tolerance}$ **then**

      $S_{\text{merge}}\texttt{.add}\big(\{(I_h, I_h^{(sibling)}) : \Delta_{\text{merge}}\tilde{\mathcal{L}}_{h'}^{(t)}\}\big)$

    **end if**

  **end if**

**end while**

# Appendix B

$$G_j = \sum_{i \in I_j} g_i = \sum_{i \in I_j} \sum_{k=0}^{K-2} -s_{ik}\sigma(-s_{ik}d_{ik})$$

$$H_j = \sum_{i \in I_j} h_i = \sum_{i \in I_j} \sum_{k=0}^{K-2} \frac{2\sigma(s_{ik}d_{ik}) - 1}{2s_{ik}d_{ik}}$$

$$\boldsymbol{L}_j = \sum_{i \in I_j} \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \tilde{y}} = \begin{bmatrix} ... & \sum_{i \in I_j} -\sigma(s_{ik}d_{ik})\big(1 - \sigma(s_{ik}d_{ik})\big) & ... \end{bmatrix}^\intercal \in \mathbb{R}^{K-1}$$

$$\boldsymbol{u} = \sum_{i=1}^{n} \frac{\partial l_i}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{n} \begin{bmatrix} ... & s_{ik}\sigma(-s_{ik}d_{ik}) & ... \end{bmatrix}^\intercal \in \mathbb{R}^{K-1}$$

$$\boldsymbol{V} = \sum_{i=1}^{n} \frac{\partial^2 l_i}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^\intercal} = \operatorname{diag}\Big( \begin{bmatrix} ... & \sum_{i=1}^{n} \frac{2\sigma(s_{ik}d_{ik}) - 1}{2s_{ik}d_{ik}} & ... \end{bmatrix} \Big) \in \mathbb{R}^{(K-1)\times(K-1)}$$

$$\begin{aligned} \sigma(x) &= \frac{1}{1+e^{-x}} \\ s_{ik} &= \mathrm{s}(y_i - k) \\ d_{ik} &= \tilde{y}_i - \theta_k \end{aligned} \tag{B.1}$$