THE UNIVERSITY OF CHICAGO

A VARIATIONAL BAYESIAN APPROACH FOR COMBINING WEAK LEARNERS INTO A STRONG LEARNER IN REGRESSION PROBLEMS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF STATISTICS

BY
ANDREW GOLDSTEIN

CHICAGO, ILLINOIS
DECEMBER 2022

For Mickey

But this was a new year, and he felt that this time, with this new pair of shoes, he could do anything, anything at all.

– Ray Bradbury

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Matthew Stephens; without his original ideas and confidence in my abilities to bring them to fruition, none of this would have been possible. While I have often chosen to be as independent as possible in my studies, Matthew has always served as an intellectual rudder, making the necessary course corrections to keep me sailing in the right direction.

I also want to thank the other members of my dissertation committee: Rebecca Willett and Veronika Rockova. Their suggestions and questions helped me immensely with figuring out how to convey my thoughts and research.

I would also like to thank the other faculty members in the Department of Statistics; I have learned so much from taking and TA-ing your classes. In particular, I would like to thank Chao Gao and Mihai Anitescu, whose classes ignited new scholarly interests in me. I would also like to thank Rina Foygel Barber; being your TA so many times has given me an appreciation for the fundamental concepts of linear regression that no applied statistician should forget.

Next, I would like to thank all of my friends and colleagues in the Stephens lab and the Department of Statistics. Getting to know you all has made my time here at UChicago so enjoyable.

Lastly, I would like to thank all of my friends, my family, my partner Selene, and our dog Mickey. You have all brought me such joy and laughter over the years; spending time with all of you is what gives life meaning.

ABSTRACT

One of the pillars of machine learning is that of non-linear regression on tabular data. For the last few decades, the performance of ensemble methods based on a sum-of-trees model (gradient boosting and random forest methods in particular) has been state-of-the-art (Shwartz-Ziv and Armon [2022], Chen and Guestrin [2016], Fernández-Delgado et al. [2014]). However, such methods can suffer from a few weaknesses; in particular, they often require time-consuming cross-validation procedures to tune a slew of hyper-parameters, and they provide no level of uncertainty about their predictions. Bayesian methods can address both through the use of hierarchical modelling. But many methods rely on Markov chain Monte Carlo (MCMC) methods that can be slow and scale poorly, not to mention the further complications that can arise due to poor mixing of the Markov chain.

In this dissertation, we introduce a new Bayesian framework, VEB-Boost, that aims to address these challenges (implemented in our R package `VEB.Boost`). In particular, it relies on empirical Bayes and variational inference, allowing us to bypass hyper-parameter tuning while being able to scale well. In the VEB-Boost framework, we combine weak learners (*a la* boosting) by adding and/or multiplying them together in an arbitrary order. Doing so yields a modular fitting procedure that reduces to iteratively fitting a single weak learner at a time. We demonstrate the potential of VEB-Boost with a simulation study and real-dataset benchmarking analysis.

We also show how to extend the VEB-Boost model to non-Gaussian response data. We derive extensions for: logistic regression, multinomial logistic regression, negative binomial regression, accelerated failure time models, ordinal logistic regression, Bradley-Terry pairwise ranking models, Plackett-Luce listwise ranking models, Cox proportional hazards models, and multivariate Gaussian regression. Many of these approximations are new, and we note some interesting connections between them. Lastly, we demonstrate the logistic and multinomial logistic models in a simulation study and real-data benchmarking analysis.

CHAPTER 1

INTRODUCTION

In machine learning and statistics, there are few settings more commonly used than the general regression setting $y_i = f(\mathbf{x}_i) + \epsilon_i$. Here, $f(\ )$ is an unknown arbitrary function mapping a feature vector $\mathbf{x}$ 2 R^p to a conditional expectation $\mathsf{E}[y|\mathbf{x}]$ 2 R, and ϵ is independent mean-zero random noise (commonly assumed to be Gaussian). The primary goal of regression analysis is to learn an approximation $\hat{f}(\ )$ $f(\ )$ so that for a new input vector $\mathbf{x}$ we can use $\hat{f}(\mathbf{x})$ as an accurate prediction for its unobserved response y.

One popular class of methods to approximate $f(\mathbf{x})$ is ensemble methods, in particular those using a sum-of-trees model. This class of methods includes, among others, boosted decision trees (Freund and Schapire [1997], Friedman [2001], Chen and Guestrin [2016], Prokhorenkova et al. [2018]), random forests (Breiman [2001]), and Bayesian additive regression trees (BART) (Chipman et al. [2010]). These methods all approximate the unknown function as

$$f(\mathbf{x}) \quad \sum_{l=1}^{L} \hat{f}_l(\mathbf{x}), \tag{1.1}$$

where each $\hat{f}_l(\ )$ is a regression tree. In the boosting literature, these functions $\hat{f}_l$ are referred to as "weak learners," in that individually they are only weakly informative for the response, but together can be combined into a "strong learner." Despite the fact that some of these methods were developed over 20 years ago, they remain popular in many applications of machine learning where the practitioner's goal is good prediction accuracy in the regression context (see, e.g., Chen and Guestrin [2016]). In particular, these methods often yield state of the art performance on tabular datasets where there is no inherent structure in the observations that can be leveraged by complex and purpose-built deep learning architectures (Shwartz-Ziv and Armon [2022]).

In recent years, there has been an increased interest in Bayesian methods, including

BART and its extensions (Tan and Roy [2019]). The benefits of Bayesian methods are quite compelling; for example, they can provide estimates of uncertainty without compromising predictive performance (Tan and Roy [2019]). However, the computational cost of the Markov chain Monte Carlo (MCMC) schemes used to implement many of these methods are prohibitively expensive, especially given the rapidly expanding nature of data in the modern era. There have been some efforts to lessen the computational burden of BART, such as accelerated BART (XBART) (He et al. [2019]). However this strategy relies on very few samples (they recommend about 25) from an approximate sampling scheme, which could make uncertainty estimates less reliable.

The approach outlined in this dissertation, variational empirical Bayes boosting (VEB-Boost), fits a model similar to the ensemble model (1.1), but instead combines weak Bayesian learners using an arbitrary sequence of addition and multiplication rather than just addition. To fit the model, VEB-Boost relies on variational inference. Variational inference yields an approximation to a posterior distribution that offers computational benefits over sampling from the true posterior using MCMC schemes (Blei et al. [2017]). In addition, VEB-Boost is adaptive in the sense that it allows for an empirical Bayes procedure to estimate the prior distribution of each weak learner, which informs the algorithm as to which parts of the estimating function can improve and which cannot. VEB-Boost is also flexible and modular in that it can accommodate any prior distribution for the weak learners, provided the practitioner supplies a function that can approximate the posterior distribution in the context of a simpler problem outlined later in equation (3.1).

The VEB-Boost algorithm is implemented in an R-package that is available via github (`https://github.com/stephenslab/VEB.Boost`); the package's interface is stable, but the back-end code is in active development and subject to change.

This dissertation is organized as follows. The remainder of the introduction provides a brief review of variational Bayesian inference, empirical Bayes, and how the two can be

combined into a procedure dubbed "variational empirical Bayes" (VEB).

Chapter 2 presents what we call the variational additive model. This model bears a strong resemblance to the ensemble model from (1.1) in that we are adding together many weak learners. We show how to fit the model by iteratively fitting each weak learner to the residuals from the other weak learners in a process that is reminiscent of the fitting procedure in gradient boosting with a squared-error loss (Friedman [2001]) and the backfitting procedure used to fit generalized additive models (GAMs) (Hastie and Tibshirani [1990]). We introduce a very useful Bayesian weak learner that is used throughout the thesis in our numerical examples, called the single effect regression (SER). We then provide a brief simulation study showing the potential of this model to fit GAMs, comparing its performance with a few other algorithms.

Chapter 3 makes the leap from only adding together weak learners to also being able to multiply weak learners together in what we call the variational empirical Bayes boosting (VEB-Boost) model. We show that, using a different notion of residual, we can fit the model by iteratively fitting each weak learner on the residuals of the others. We also outline some methods to utilize the empirical Bayes aspect of the model to add more weak learners until the fit can no longer be improved. We wrap up chapter 3 with some simulated and real-data examples comparing the VEB-Boost method to other popular non-linear regression methods.

Chapter 4 shows how to extend the VEB-Boost model to many types of non-Gaussian data, and how to incorporate observation weights. In particular, we show how to extend the model to: binary data, multinomial data, negative binomial data, accelerated failure time data with log-logistic noise (accounting for left, right, and interval censoring), ranking data (both pairwise and listwise comparisons), ordinal data, survival data with the proportional hazards assumption, and multivariate Gaussian data. The approach we take to extend the VEB-Boost model is to approximate each data type with a Gaussian distribution by lower-bounding the log-likelihood of the model with a quadratic function. Although there are

known drawbacks to this approach (Minka [2001], Knowles and Minka [2011]), the benefit is that the practitioner only needs to derive and implement the solution for fitting their weak learner of choice to heteroskedastic Gaussian data. This substantially lowers the burden on the user and allows for more weak learners to be explored for any/all of these different types of data, without the need for any additional derivations or implementations. We conclude this chapter with some simulated and real-data examples comparing the VEB-Boost method to other popular non-linear logistic and multinomial logistic regression methods.

Throughout this dissertation, all proofs not presented in a chapter itself can be found in Section A.3. A zenodo repository containing the scripts used for the simulations and benchmarking studies performed in Chapters 3 and 4 can be found at `https://doi.org/10.5281/zenodo.7105767`.

1.1 Variational Inference: A Review

In this section, we offer a brief review of variational Bayes (VB) in Section 1.1.1, empirical Bayes (EB) in Section 1.1.2, and show how the two can be combined into a single procedure – dubbed variational empirical Bayes (VEB) – in Section 1.1.3. For a high-level overview of variational inference, see Blei et al. [2017].

1.1.1 Variational Bayes

Bayesian inference can be broken down into three components: the prior distribution of your parameters, the likelihood of the observed data under a particular model, and the posterior distribution of your parameters. These three components are related through Bayes' theorem:

$$p(\ \ \boldsymbol{j}\mathbf{x}) = \frac{g(\ \ )\ \ p(\mathbf{x}\boldsymbol{j}\ \ )}{p(\mathbf{x})}. \tag{1.2}$$

Here, $p(\ \ j\mathbf{x})$ is the posterior distribution of your model parameters given the observed data $\mathbf{x}$, $g(\ \ )$ is the prior distribution of these parameters, $p(\mathbf{x}j\ \ )$ is the likelihood of the observed data given the parameters, and $p(\mathbf{x})$ is the marginal likelihood of the observed data (sometimes referred to as the evidence).

The difficulty with Bayesian inference arises from the challenge of dealing with the marginal likelihood $p(\mathbf{x})$, since it involves the often intractable integral $p(\mathbf{x}) = \int g(\ \ )p(\mathbf{x}j\ \ )d\ \ $. Variational Bayesian inference, an alternative to exact Bayesian inference, was introduced to yield computationally tractable approximations to the exact posterior distribution $p(\ \ j\mathbf{x})$. Concretely, the goal of variational Bayes is to find

$$q\ (\ \ ) = \arg\min_{q2Q} D_{KL}\Big(q(\ \ )kp(\ \ j\mathbf{x})\Big), \tag{1.3}$$

where Q is referred to as a variational class of distributions, and $D_{KL}(qkp)$ is the KL-divergence from the density q to the density p, defined as

$$D_{KL}(qkp) \quad \mathsf{E}\quad {}_q\Big[\log\frac{q(\ \ )}{p(\ \ )}\Big]. \tag{1.4}$$

The variational class Q is often chosen to facilitate easy and/or closed-form updates when solving this optimization problem. If Q was the class of all distributions, then the minimizer would be equal to the true posterior, so the hope is that Q is expressive enough to contain distributions that are "close" to the true posterior while maintaining computational tractability.

The objective function in the optimization problem in (1.3) is not tractable to compute, since it relies on the unknown posterior distribution. Instead, we can perform some algebraic manipulations to show that

$$q\ (\ \ ) = \arg\max_{q2Q} \mathsf{E}\quad {}_q[\log p(\mathbf{x}j\ \ )]\quad D_{KL}\Big(q(\ \ )kg(\ \ )\Big). \tag{1.5}$$

The right-hand-side of this equation is referred to as the evidence lower bound (ELBO), so named because it is (unsurprisingly) a lower bound on the log-evidence, $\log p(\mathbf{x})$. It is straightforward to show that equivalent formulations of the ELBO $F(g, q; \mathbf{x})$ are

$$F(g, q; \mathbf{x}) = \mathsf{E} \quad _q[\log p(\mathbf{x}| \quad)] \quad D_{KL}\Big(q(\quad)\|g(\quad)\Big) \tag{1.6a}$$

$$= l(g; \mathbf{x}) \qquad\qquad D_{KL}\Big(q(\quad)\|p(\quad |\mathbf{x})\Big). \tag{1.6b}$$

The ELBO is typically a non-convex objective function and is thus difficult to maximize. In principle, any non-convex optimization method can be used to find a (local) maximum of the ELBO. But in practice, the most commonly-used algorithm is known as coordinate ascent variational inference (CAVI), which is essentially just exact (block) coordinate maximization of the ELBO (see Bishop [2006], Blei et al. [2017]).

1.1.2 Empirical Bayes

The goal of empirical Bayes is to perform Bayesian inference using a prior distribution $\hat{g}(\quad) \in \mathcal{G}$ learned from the data. Suppose we have the following Bayesian model

$$\mathbf{x} \quad p(\mathbf{x}| \quad , \quad) \tag{1.7a}$$

$$g \in \mathcal{G}, \tag{1.7b}$$

where we observe data $\mathbf{x}$ from a data generating process with likelihood $p(\mathbf{x}| \quad , \quad)$ that depends on (i) unobserved latent variables coming from prior distribution g living in some class of distributions $\mathcal{G}$, and (ii) additional parameters $\in \Theta$, either fixed or to be estimated.

Empirical Bayes is often framed as a two-step procedure. The steps are:

1. Estimating the prior by finding

$$(\hat{g}, \hat{\ }) := \underset{g \in \mathcal{G},\ \in \Theta}{\arg\max}\ l(g, \ ; \mathbf{x}),$$

where

$$l(g, \ ; \mathbf{x}) := \log \int p(\mathbf{x} | \ , \ ) g(\ ) d$$

is the marginal log-likelihood of the observed data;

2. Given the estimated prior distribution $\hat{g}$ and parameters $\hat{\ }$, compute the posterior distribution of $\ $:

$$\hat{p}_{post}(\ ) := p(\ | \mathbf{x}, \hat{g}, \hat{\ }) \propto p(\mathbf{x} | \ , \hat{\ }) \hat{g}(\ ).$$

However, this same EB procedure can be viewed as a one-step procedure, as outlined in appendix B of Wang et al. [2020]. As in (1.6b), define the ELBO as

$$F(q, g, \ ; \mathbf{x}) := l(g, \ ; \mathbf{x}) \quad D_{KL}\Big(q(\ ) \| \hat{p}_{post}(\ )\Big).$$

Using this formulation, it is easy to see that the two-step EB procedure can be performed in a single step:

$$(\hat{p}_{post}, \hat{g}, \hat{\ }) = \underset{q, g \in \mathcal{G},\ \in \Theta}{\arg\max}\ F(q, g, \ ; \mathbf{x}). \tag{1.8}$$

This can be seen by making two key observations:

1. The first term of the ELBO, $l(g, \ ; \mathbf{x})$, does not depend on q, and so

$$\begin{aligned}\hat{q} &= \arg\max_q F(q, g, \ ; \mathbf{x}) \\ &= \arg\min_q D_{KL}\Big(q(\ ) \| \hat{p}_{post}(\ )\Big) \\ &= \hat{p}_{post};\end{aligned}$$

2. Since $D_{KL}\Big(q(\ ) \| \hat{p}_{post}(\ )\Big) = 0$ when $q(\ ) = \hat{p}_{post}(\ )$, we have $\max_q F(q, g, \ ; \mathbf{x}) = l(g, \ ; \mathbf{x})$. And thus,

$$\begin{aligned}(\hat{g}, \hat{\ }) &= \arg\max_{g \in \mathcal{G}, \ \in \Theta} l(g, \ ; \mathbf{x}) \\ &= \arg\max_{g \in \mathcal{G}, \ \in \Theta} \max_q F(q, g, \ ; \mathbf{x}).\end{aligned}$$

1.1.3 Variational Empirical Bayes

The variational empirical Bayes (Braun and McAuliffe [2010], Wang et al. [2020]) approach makes one simple modification to the one-step EB procedure from (1.8); we only maximize over $q \in \mathcal{Q}$ for some variational class of distributions $\mathcal{Q}$. That is, we aim to find

$$(q \ , \hat{g}, \hat{\ }) = \arg\max_{q \in \mathcal{Q}, g \in \mathcal{G}, \ \in \Theta} F(q, g, \ ; \mathbf{x}). \tag{1.9}$$

The procedure can be thought of as being similar to EB, since we still learn a prior distribution $\hat{g}$ and parameters from the data, but we are now also finding an approximation to the true posterior $q \in \mathcal{Q}$.

CHAPTER 2

THE VARIATIONAL ADDITIVE MODEL

2.1 Introduction

In building up to the full VEB-Boost model, the first step is to make the leap from fitting data with a single weak learner to using a sum of weak learners. We refer to this sum as the variational additive model. This model is outlined in Appendix B of Wang et al. [2020]. We deviate slightly from their presentation, but the models are essentially the same.

The remainder of this chapter is organized as follows. Section 2.2 describes the Bayesian regression setting that is used as the building-block for the additive model, and we describe the useful example of the single effect regression (SER) in Section 2.2.1. We then present a generalization to the SER which allows for the fitting of non-linear relationships in Section 2.2.2. Section 2.3 then shows how we can combine weak learners that are solving the Bayesian regression problem in 2.2 into a strong learner by adding them together. Finally, we demonstrate the generalized additive model in a brief simulation study in Section 2.4.

2.2 Unweighted Bayesian Regression Problem

The building-block for the variational additive model, and what each weak learner aims to fit, is the unweighted Bayesian regression problem:

$$\mathbf{y} = \quad + \tag{2.1a}$$

$$N(\mathbf{0}, \sigma^2 I_n) \tag{2.1b}$$

$$h(\quad) \tag{2.1c}$$

$$g \in G. \tag{2.1d}$$

Here, $\mathbf{y} \in \mathbb{R}^n$ is our vector of responses, ϵ_i is iid Gaussian noise with given residual

variance $\sigma^2 > 0$, $\quad 2\ \mathsf{R}^p$ is a random vector with prior distribution g belonging to a prior family G, and $h : \mathsf{R}^p\ !\ \mathsf{R}^n$ is a fixed function that maps $\quad$ to a mean response vector $\quad$. For example, $h(\ \ )$ might be $\mathbf{X}\ \ $ for a fixed design matrix $\mathbf{X}\ 2\ \mathsf{R}^{n\ \ p}$. For terminology, we often refer to $\quad h(\ \ )$ as a Bayesian weak learner.

In order to "solve" this Bayesian regression problem, we require a function that can estimate a prior distribution $\hat{g}\ 2\ \mathsf{G}$ using an empirical Bayes procedure, and then return a variational approximation $q\ \ 2\ \mathsf{Q}$ to the true posterior distribution of $\quad$, where Q is an arbitrary variational class chosen by the practitioner. In fact, the VEB-Boost algorithm only needs access to $\mathsf{E}\ \ _q\ [h(\ \ )]$, $\mathsf{E}\ \ _q\ [h(\ \ )^2]$, and $D_{KL}(q\ \ k\hat{g})$.

Using the alternate formulations of the ELBO from (1.6), we denote the ELBO of model (2.1) as:

$$F_0(g, q; \mathbf{y}, h, \sigma^2) = \ \ \frac{n}{2}\log(2\pi)\ \ \ \frac{n}{2}\log(\sigma^2)\ \ \ \frac{1}{2\sigma^2}\mathsf{E}\ \ _q\Big[k\mathbf{y}\ \ \ h(\ \ )k_2^2\Big]\ \ \ D_{KL}\Big(q(\ \ )kg(\ \ )\Big). \tag{2.2}$$

For the purpose of compact notation, let

$$\begin{aligned} &FIT(\mathbf{y}, h, \sigma^2, \mathsf{G}, \mathsf{Q}) \\ &:= \left(\underset{g2\mathsf{G}, q2\mathsf{Q}}{\arg\max}\ F_0(g, q; \mathbf{y}, h, \sigma^2),\ \bar{\ }\ \ \mathsf{E}\ \ _q\ [h(\ \ )],\ \overline{\ 2\ }\ \ \mathsf{E}\ \ _q\ [h(\ \ )^2],\ D_{KL}(q\ \ k\hat{g})\right) \end{aligned} \tag{2.3}$$

be a function that maximizes the ELBO with respect to the prior distribution $g\ 2\ \mathsf{G}$ and variational approximation to the posterior $q\ 2\ \mathsf{Q}$ (with $\arg\max\ \hat{g}$ and $q\ \ $, respectively), and also returns the first and second posterior moments of the weak learner ($\bar{\ }$ and $\overline{\ 2\ }$), as well as the KL-divergence from $q\ \ $ to $\hat{g}$.

2.2.1 The Single E ect Regression

The single effect regression (SER) model, the foundational idea of which was introduced in Servin and Stephens [2007] and used as the building-block of Wang et al. [2020], is a very simple Bayesian linear model in which a single variable has a non-zero effect. Just like a decision stump or small decision tree, it is clear that such a model is too simple to be of much use on its own. But if we were to combine many such weak learners together, we should be able to explain more signal in a dataset.

The SER model can be formalized as:

$$\mathbf{y} = \mathbf{X} \quad + \tag{2.4a}$$

$$N(\mathbf{0}, \sigma^2 I_n) \tag{2.4b}$$

$$b \tag{2.4c}$$

$$b \quad N(0, \sigma_0^2) \tag{2.4d}$$

$$Mult(1, \quad). \tag{2.4e}$$

Here, $\mathbf{X}$ 2 $\mathsf{R}^{n \quad p}$ is a fixed design matrix, 2 R^p is an effect vector that is non-zero in only a single position (i.e. 1-sparse), b is the value of the non-zero entry in which comes from a mean-zero Gaussian prior distribution with variance σ_0^2, and is a vector of 0's with a 1 in a single position indicating which entry of is non-zero, which has a multinoulli prior distribution with a fixed prior probability vector 2 $\Delta^{p \quad 1}$ R^p.

One of the nice properties of this simple model is that the exact posterior distribution of is available in closed form.

Proposition 2.2.1 (SER Posterior Distribution)**.** *For a given prior variance $\sigma_0^2 > 0$, if we let $\tau_j := \frac{1}{\sigma_0^2} + \frac{\boldsymbol{X}_j^T \boldsymbol{X}_j}{\sigma^2}$ and $\nu_j := \frac{\boldsymbol{X}_j^T \boldsymbol{y}}{\sigma^2}$, then the posterior distribution of is given by:*

$$\boldsymbol{j}\boldsymbol{X}, \boldsymbol{y}, \sigma^2, \sigma_0^2 \quad Mult(1, \quad) \tag{2.5a}$$

$$\alpha_j \ / \ \pi_j \sqrt{\frac{1}{\tau_j}} \exp\left\{\frac{\nu_j^2}{2\tau_j}\right\} \tag{2.5b}$$

$$b\boldsymbol{j}\boldsymbol{X}, \boldsymbol{y}, \sigma^2, \sigma_0^2, \gamma_j = 1 \quad \boldsymbol{N}(\mu_j, \sigma_j^2) \tag{2.5c}$$

$$\mu_j = \frac{\nu_j}{\tau_j} \quad \textit{and} \quad \sigma_j^2 = \frac{1}{\tau_j}. \tag{2.5d}$$

Proof: Recalling that the prior places all of its mass on 1-sparse vectors (i.e. vectors with a single non-zero entry), we see that the posterior will also place all of its mass on such configurations. By Bayes' theorem, and using the definitions for τ_j and ν_j from above, and defining $\mathbf{e}_j$ to be the standard basis vector for dimension j, we have

$$\begin{aligned}
P(\quad = b\mathbf{e}_j \boldsymbol{j} \mathbf{y}, \mathbf{X}, \sigma_0^2, \quad, \sigma^2) \ / \ & P(\quad = b\mathbf{e}_j \boldsymbol{j} \sigma_0^2, \quad) \ P(\mathbf{y}\boldsymbol{j}\mathbf{X}, \sigma^2, \quad = b\mathbf{e}_j) \\
&= \pi_j \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\quad \frac{1}{2}\frac{2}{0} b^2} \prod_{i=1}^{n} \cancel{p} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\quad \frac{1}{2}\frac{2}{}(y_i \quad bX_{ij})^2} \\
&\ / \ \pi_j e^{\quad \frac{1}{2}\frac{2}{0} b^2 \quad \frac{1}{2}\frac{2}{} \sum_{i=1}^{n} (y_i \quad bX_{ij})^2} \\
&\ / \ \pi_j e^{\quad \frac{1}{2}\frac{2}{0} b^2 \quad \frac{1}{2}\frac{2}{} \sum_{i=1}^{n} \quad 2y_i bX_{ij} + b^2 X_{ij}^2} \\
&= \pi_j e^{\quad \frac{j}{2}(b \quad \nu_j/\tau_j)^2 + \frac{j}{2}(\nu_j/\tau_j)^2} \quad \frac{\sqrt{2\pi/\tau_j}}{\sqrt{2\pi/\tau_j}} \\
&\ / \ \pi_j \sqrt{\frac{1}{\tau_j}} e^{\frac{2}{2}\frac{2}{j}} \quad \frac{1}{\sqrt{2\pi/\tau_j}} e^{\quad \frac{j}{2}(b \quad \nu_j/\tau_j)^2}.
\end{aligned}$$

We can recognize this as being proportional to the posterior described above in Proposition 2.2.1. QED

It is also straightforward to perform an empirical Bayes step to estimate the prior distribution of $\quad \boldsymbol{2}\ \boldsymbol{G}$, which is fully parameterized by the prior variance $\sigma_0^2 > 0$.

Proposition 2.2.2. *The likelihood of the SER model as a function of $\sigma_0^2 > 0$ can be written as*

$$L(\sigma_0^2; \boldsymbol{y}, \boldsymbol{X}, \quad , \sigma^2) = \sum_{j=1}^{p} \pi_j \int_{1}^{1} p(b; \sigma_0^2) p(\boldsymbol{y}; b, \boldsymbol{X}_j, \sigma^2) db \tag{2.6a}$$

$$/ \sqrt{\frac{1}{\sigma_0^2}} \sum_{j=1}^{p} \pi_j \sqrt{\frac{1}{\tau_j}} \exp\left\{\frac{\nu_j^2}{2\tau_j}\right\}, \tag{2.6b}$$

where ν_j and τ_j are de ned in Proposition 2.2.1.

Proof: Starting from the expression for the likelihood, we get

$$\begin{aligned}
L(\sigma_0^2; \mathbf{y}, \mathbf{X}, \quad , \sigma^2) &= \sum_{j=1}^{p} \pi_j \int_{1}^{1} p(b; \sigma_0^2) p(\mathbf{y}; b, \mathbf{X}_j, \sigma^2) db \\
&= \sum_{j=1}^{p} \pi_j \int_{1}^{1} \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{1}{2}\frac{2}{0} b^2} \quad (2\pi\sigma^2)^{n/2} e^{\frac{1}{2}\frac{2}{2} k\mathbf{y} \quad b\mathbf{X}_j k_2^2} db \\
&/ \sqrt{\frac{1}{\sigma_0^2}} \sum_{j=1}^{p} \pi_j \int_{1}^{1} e^{\frac{j}{2}(b \quad \nu_j/\tau_j)^2 + \frac{j}{2}(\nu_j/\tau_j)^2} \quad \frac{\sqrt{2\pi/\tau_j}}{\sqrt{2\pi/\tau_j}} db \\
&/ \sqrt{\frac{1}{\sigma_0^2}} \sum_{j=1}^{p} \pi_j \sqrt{\frac{1}{\tau_j}} e^{\nu_j^2/2\tau_j}. \qquad QED
\end{aligned}$$

Using the above expression for the likelihood of the SER model, we can perform an empirical Bayes step by maximizing this likelihood with respect to $\sigma_0^2 > 0$ using the 1-dimensional numerical optimizer of our choice, such as the Brent method available in the base R function `optimize`.

We can also easily calculate the KL-divergence $D_{KL}(q \ k\hat{g})$ as:

$$D_{KL}(q \ k\hat{g}) = \sum_{j:\alpha_j > 0} \alpha_j \quad \left(\log\left(\frac{\alpha_j}{\pi_j} \quad \frac{\sigma_0}{\sigma_j}\right) \quad \frac{1}{2} + \frac{\sigma_j^2 + \mu_j^2}{2\sigma_0^2} \right). \tag{2.7}$$

2.2.2 Generalized Single E ect Regression

As an extension to the SER, suppose that instead of selecting one of a number of linear terms, we are selecting between one of a number of models. For example, perhaps each model corresponds to a non-linear function of a single variable, *a la* generalized additive models (Hastie and Tibshirani [1990]). From here on, I will assume that these models rely on only a single variable, but this applies to the more general case of arbitrary models. More concretely, consider the model

$$\mathbf{y} = \sum_{j=1}^{p} \gamma_j h_j(\ _j\ ;\ \mathbf{x}_j) + \tag{2.8a}$$

$$\epsilon \quad N(\mathbf{0}, \sigma^2 I_n) \tag{2.8b}$$

$$_j \ ^? \ g_j(\ ) \ 2 \ G_j \tag{2.8c}$$

$$Mult(1, \quad). \tag{2.8d}$$

Here, $h_j(\ ;\ \mathbf{x}_j)$ are fixed functions of a single variable $\mathbf{x}_j$ parameterized by $_j \ 2 \ \mathsf{R}^{pj}$ (e.g. $h_j(\ _j\ ;\ \mathbf{x}_j) = \mathbf{x}_j b_j$ recovers the SER), g_j is a prior distribution over $_j$ belonging to a prior family G_j, and is a vector of 0's with a 1 in a single position indicating which function $h_j(\ )$ has a non-zero weight, which has a multinoulli/categorical prior distribution with a fixed prior probability vector $\ 2 \ \Delta^p \ ^1 \ \mathsf{R}^p$.

It is very easy to see that since the prior distribution places all of its mass on configurations of a single non-linear function of a single variable, so too will the posterior distribution. We can calculate the posterior probabilities of each function being non-zero in the straightforward fashion:

$$\alpha_j := P(\gamma_j = 1 | \mathbf{y}) \ / \ P(\gamma_j = 1) \quad P(\mathbf{y} | \gamma_j = 1) = \pi_j P(\mathbf{y} | \gamma_j = 1). \tag{2.9}$$

Note that $P(\mathbf{y}|\gamma_j = 1)$ is simply the evidence of the model where there is only h_j, i.e.

$\mathbf{y} = h_j(\ \ _j\ ;\ \mathbf{x}_j) + \ \ .$

Continuing on, it is easy to calculate the posterior distribution of $\ \ _j|\gamma_j = 1$:

$$P(\ \ _j|\gamma_j = 1, \mathbf{y})\ /\ g_j(\ \ _j)\ \ P(\mathbf{y}|\gamma_j = 1, \ \ _j). \tag{2.10}$$

Note that this is proportional to the posterior probability of $\ \ _j$ under the model where there is only h_j, i.e. $\mathbf{y} = h_j(\ \ _j\ ;\ \mathbf{x}_j) + \ \ .$

Thus, the posterior distribution is a mixture distribution over the posteriors from the "univariate" models in which there is just a single function h_j, weighted by α_j. So algorithmically, we can find the posterior distributions of these "univariate" models individually, and then weight them by the appropriate weights α_j (which are proportional to the product of the prior probability that model j is the non-zero model, and the evidence for that model). This is exactly the procedure that is done in the SER.

As a slight modification, suppose that for each model j we have a variational class of distributions $\mathcal{Q}_j$, and our approximation to the posterior of $\ \ _j$ is a member of this class. In this case, we can find an approximation to the posterior by carrying out the same procedure, but instead we find an approximation to the posterior $q_j \in \mathcal{Q}_j$ for each univariate model j. We can then approximate the evidence of the model $P(\mathbf{y}|\gamma_j = 1)$ as $\exp\{ELBO_j\}$, where $ELBO_j$ is the ELBO from univariate model j. This is conceptually similar to using the ELBOs as an approximate log-likelihood, which is done in Carbonetto and Stephens [2012]. We formalize this intuition in the following proposition:

Proposition 2.2.3. *Consider the model*

$$\boldsymbol{y} = f(\boldsymbol{X}; \quad) + \tag{2.11a}$$

$$\epsilon \quad N(\boldsymbol{0}, \sigma^2 I_n) \tag{2.11b}$$

$$\sum_{j=1}^{p} \pi_j g_j(\ ), \quad g_j(\ ) \ 2 \ G_j \tag{2.11c}$$

$$\text{fsupport } g_j g \setminus \text{fsupport } g_k g = ;, \qquad j \ 6= k. \tag{2.11d}$$

Suppose that we wish to find a variational approximation to the true posterior that takes the form of the mixture model $q(\ _1, \ldots, \ _p) = \sum_{j=1}^{p} \alpha_j q_j(\ _j)$*, where* $2 \ \Delta^{p \ 1}$ *and each* $q_j \ 2 \ Q_j$ *for a variational class* Q_j*. Let* M_j *be the model where* g_j*, i.e.* $\pi_j = 1$*, and let* $ELBO_j(g_j, q_j)$ *be the ELBO we get from model* M_j *with prior* $g_j \ 2 \ G_j$ *and variational approximation* $q_j \ 2 \ Q_j$ *with* $\arg\max\ (\hat{g}_j, q_j)$ *and maximum* $ELBO_j$*. Then the empirical Bayes estimate for the prior is given by the mixture model*

$$\hat{g}(\ ) = \sum_{j=1}^{p} \pi_j \hat{g}_j(\ )$$

and the variational approximation to the posterior is given by the mixture model

$$q\ (\ ) = \sum_{j=1}^{p} \alpha_j q_j(\ ),$$

where $\alpha_j \ / \ \pi_j \exp\{ELBO_j\}$*.*

Proof: Writing out the objective function in the variational optimization, and substituting in the expectations and KL-divergences we get with our specific prior g and variational

approximation q, we aim to find

$$
\begin{aligned}
& \max_{\alpha \in \Delta^{p-1}, g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} \mathbb{E}_q\Big[\log P(\mathbf{y} \mid \cdot)\Big] - D_{KL}(q \| g) \\
= & \max_{\alpha \in \Delta^{p-1}, g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} \sum_{j=1}^{p} \alpha_j \mathbb{E}_{q_j}\Big[\log P(\mathbf{y} \mid \cdot)\Big] - D_{KL}\Big(\sum_{j=1}^{p} \alpha_j q_j \Big\| \sum_{j=1}^{p} \pi_j g_j\Big) \\
= & \max_{\alpha \in \Delta^{p-1}, g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} \sum_{j=1}^{p} \alpha_j \mathbb{E}_{q_j}\Big[\log P(\mathbf{y} \mid \cdot)\Big] - \sum_{j=1}^{p} \alpha_j \mathbb{E}_{q_j}\Big[\log \frac{\alpha_j q_j}{\sum_{k=1}^{p} \pi_k g_k}\Big] \\
= & \Big[\text{support } q_j \setminus \text{support } g_k = \emptyset \text{ for } j \neq k\Big] \\
= & \max_{\alpha \in \Delta^{p-1}, g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} \sum_{j=1}^{p} \alpha_j \mathbb{E}_{q_j}\Big[\log P(\mathbf{y} \mid \cdot)\Big] - \sum_{j=1}^{p} \alpha_j \Big[\log \frac{\alpha_j}{\pi_j} + D_{KL}(q_j \| g_j)\Big] \\
= & \max_{\alpha \in \Delta^{p-1}, g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} \sum_{j=1}^{p} \alpha_j \left[\mathbb{E}_{q_j}\Big[\log P(\mathbf{y} \mid \cdot)\Big] - D_{KL}(q_j \| g_j) - \log \frac{\alpha_j}{\pi_j}\right] \\
= & \max_{\alpha \in \Delta^{p-1}} \sum_{j=1}^{p} \alpha_j \left[\max_{g_j \in \mathcal{G}_j, q_j \in \mathcal{Q}_j} ELBO_j(g_j, q_j) - \log \frac{\alpha_j}{\pi_j}\right] \\
= & \max_{\alpha \in \Delta^{p-1}} \sum_{j=1}^{p} \alpha_j \left[ELBO_j - \log \frac{\alpha_j}{\pi_j}\right] \\
= & \min_{\alpha \in \Delta^{p-1}} \sum_{j=1}^{p} \alpha_j \log \frac{\alpha_j}{\pi_j \exp\{ELBO_j\}}.
\end{aligned}
$$

Noting that in this last line we are trying to minimize a discrete KL-divergence, we see that this is minimized when $\alpha_j \propto \pi_j \exp\{ELBO_j\}$.

Thus, the solution that maximizes the ELBO over the prior distribution and variational approximation is given by

$$
\begin{aligned}
\hat{g} &= \sum_{j=1}^{p} \pi_j \hat{g}_j \\
q &= \sum_{j=1}^{p} \alpha_j q_j. \qquad\qquad QED
\end{aligned}
$$

Corollary 2.2.3.1. *Since the model given by* (2.8) *can be re-written to satisfy the conditions of Proposition 2.2.3, we can find a variational approximation to the posterior by solving the univariate models individually, and then weighting them accordingly.*

2.3 The Variational Additive Model

The variational additive model is a direct extension of the unweighted Bayesian regression problem in Section 2.2, in which we add together multiple weak learners. The model is formalized as:

$$\mathbf{y} = \sum_{l=1}^{L} \quad _l + \tag{2.12a}$$

$$\quad N(\mathbf{0}, \sigma^2 I_n) \tag{2.12b}$$

$$_l \quad h_l(\quad _l) \tag{2.12c}$$

$$_l \quad ^{?} \; g_l(\;) \; 2 \; \mathsf{G}_l. \tag{2.12d}$$

As before, $\quad _l \; 2 \; \mathsf{R}^{p_l}$ is a random vector with prior distribution g_l belonging to a prior class G_l, all of the $\quad _l$ are independent *a priori*, and $h_l : \mathsf{R}^{p_l} \; ! \; \mathsf{R}^n$ are fixed functions that map $\quad _l$ to a vector $\quad _l$. As before, we refer to these $\quad _l \quad h_l(\quad _l)$ as Bayesian weak learners.

In order to "solve" this problem, we aim to find a variational approximation q to the true posterior distribution of $(\quad _1, \ldots, \quad _L)$. We restrict our approximation to belong to the variational class

$$\mathsf{Q} = \Big\{ q \quad \Big| \quad q(\quad _1, \ldots, \quad _L) = \prod_{l=1}^{L} q_l(\quad _l), \quad q_l \; 2 \; \mathsf{Q}_l \Big\}. \tag{2.13}$$

Here, each Q_l is an arbitrary variational class that restricts the approximation q_l.

A nice property of this model is that the ELBO, when viewed as a function of a particular (g_l, q_l), holding all other $(g \quad _l, q \quad _l)$ fixed, is equal to the ELBO in the unweighted Bayesian

regression subproblem (2.1) (up to a constant), where the response is a form of residual that takes into account the fixed distributions.

As an illustrative example, consider the simple additive model

$$\mathbf{y} = \quad _1 + \quad _2 + \quad , \qquad \quad N(\mathbf{0}, \sigma^2 I_n).$$

If we want to fit the weak learner $\quad _2$, we can subtract $\quad _1$ from both sides, which leaves

$$\mathbf{y} \qquad _1 = \quad _2 + \quad , \qquad \quad N(\mathbf{0}, \sigma^2 I_n).$$

It turns out that if we replace $\quad _1$ with $\mathsf{E}_{q_1}[\quad _1]$ on the left hand side, then this is exactly the correspondence we get between the ELBOs.

Theorem 2.3.1 (Variational Additive Model ELBO Equivalence)**.** *Let* $F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \sigma^2)$ *denote the ELBO of the additive model* (2.12)*, and let* $F_l(g_l, q_l; g \quad _l, q \quad _l, \boldsymbol{y}, h_1, \ldots, h_L, \sigma^2)$ *denote this ELBO with respect to* (g_l, q_l) *while holding all other distributions xed. And let* $F_0(g, q; \boldsymbol{y}, h, \sigma^2)$ *denote the ELBO from the simple regression model* (2.1) *with response* $\boldsymbol{y}$*, residual variance* σ^2*, and mean response* $\quad h(\quad)$*. Then*

$$F_l(g_l, q_l; g \quad _l, q \quad _l, \boldsymbol{y}, h_1, \ldots, h_L, \sigma^2) = F_0(g_l, q_l; \boldsymbol{y} \quad \sum_{k \neq l} \mathsf{E} \quad _k \quad _{q_k}[h_k(\quad _k)], h_l, \sigma^2) + c,$$

where c *is a constant term in* (g_l, q_l)*.*

The proof is a special case of the proof in Section A.3.1.

Corollary 2.3.1.1 (Block Coordinate Maximization of the Variational Additive Model

ELBO). *In the same setting as Theorem 2.3.1,*

$$\begin{aligned}&\underset{g_l \in \mathcal{G}_l, q_l \in \mathcal{Q}_l}{\arg\max}\ F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \sigma^2) \\ =\ &\underset{g_l \in \mathcal{G}_l, q_l \in \mathcal{Q}_l}{\arg\max}\ F_0(g_l, q_l; \boldsymbol{y} \quad \sum_{k \neq l} \mathbb{E}_{\quad k \quad q_k}[h_k(\quad_k)], h_l, \sigma^2).\end{aligned}$$

In other words, in order to perform a block coordinate maximization step of the full ELBO in the additive model (2.12) over the prior distribution $g_l \in \mathcal{G}_l$ and variational approximation $q_l \in \mathcal{Q}_l$, we simply need to be able to solve the unweighted Bayesian regression problem for a single weak learner using the response $\tilde{\mathbf{y}} := \mathbf{y} \quad \sum_{k \neq l} \mathbb{E}_{\quad k \quad q_k}[h_k(\quad_k)]$ and residual variance σ^2.

Corollary 2.3.1.2 (Gradient of the Variational Additive Model ELBO). *In the same setting as Theorem 2.3.1, suppose that the prior family $\mathcal{G}_l$ and variational family $\mathcal{Q}_l$ are finite-dimensional parametric families. Then*

$$\nabla_{g_l, q_l} F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \sigma^2) = \nabla_{g_l, q_l} F_0(g_l, q_l; \boldsymbol{y} \quad \sum_{k \neq l} \mathbb{E}_{\quad k \quad q_k}[h_k(\quad_k)], h_l, \sigma^2),$$

where ∇_{g_l, q_l} refers to the gradient with respect to the parameters of the distributions $g_l \in \mathcal{G}_l$ and $q_l \in \mathcal{Q}_l$.

This means that if one wanted to maximize the ELBO using a first-order method instead of (block) coordinate maximization, it is sufficient to be able to calculate the gradient in the context of the unweighted Bayesian regression problem.

Using Corollary 2.3.1.1, we can write a coordinate ascent algorithm for the variational additive model.

Algorithm 1: Variational Additive Model Coordinate Ascent Algorithm

Require: Data $\mathbf{y}$; functions $h_l(\,)$, prior classes $\mathcal{G}_l$, variational classes $\mathcal{Q}_l, \quad l = 1, \ldots, L$.

Require: initial residual variance σ^2.

Require: Functions $FIT : \left(\mathbf{y}, h_l, \sigma^2, \mathcal{G}_l, \mathcal{Q}_l\right) \; ! \; \left(\hat{g}_l, q_l\,, \overline{}_l, \overline{}^2_l, D_{KL}(q_l \,k \hat{g}_l)\right)$ that solve the unweighted Bayesian regression problem for each weak learner, $l = 1, \ldots, L$.

Initialize posterior means $\overline{}_l$, $\overline{}^2_l$, for $l = 1, \ldots, L$;

Initialize residual variance to $\hat{\sigma}^2 := \sigma^2$;

repeat

  for *l in* $1, \ldots, L$ **do**

    Compute $\tilde{\mathbf{y}} = \mathbf{y} \quad \sum_{k \neq l} \overline{}_k$;

    $(\hat{g}_l, q_l\,, \overline{}_l, \overline{}^2_l, D_{KL}(q_l \,k \hat{g}_l)) \quad FIT(\tilde{\mathbf{y}}, h_l, \hat{\sigma}^2, \mathcal{G}_l, \mathcal{Q}_l)$;

  Update $\hat{\sigma}^2$; // optional;

until *convergence criterion satis ed*;

return $q_1, \ldots, q_L$.

This algorithm is reminiscent of the fitting procedures in boosting (because we fit each weak learner to the residuals from the others), as well as backfitting in generalized additive models (because we iteratively return to earlier fits and update them).

2.4 Examples

Perhaps the most well-known application of the variational additive model is the **Su**m of **Si**ngle **E**ffects (SuSiE) model (Wang et al. [2020]). In this model, each effect $\quad_l$ is an independent vector with the prior from the single effect regression outlined in Section 2.2.1.

Concretely, the model is given by

$$\mathbf{y} = \sum_{l=1}^{L} \mathbf{X}\ _l + N(\mathbf{0}, \sigma^2 I_n)$$

$$_l = b_l\ _l$$

$$b_l \overset{?}{} N(0, \sigma_{0l}^2)$$

$$_l \overset{iid}{} Mult(1, \ ).$$

The SuSiE model has seen rapid interest and success in the context of, among other applications, genetic fine-mapping. This is due to SuSiE's ability to perform sparse linear regression and variable selection in highly correlated settings and yield uncertainty estimates about which effects are non-zero.

There are a number of other ways in which the variational additive model can be potentially useful. For example, one could add together a SuSiE model and a Bayesian ridge regression model; this could be useful if you believe there are a few strong signals in the data among correlated predictors, but that all predictors have some effect. A potential scenario where this assumption may approximately hold is that of the omnigenic model of complex traits (Boyle et al. [2017]), which maintains that while a few genetic variants will have a large effect on a trait of interest, all genetic variants have a non-zero effect.

In this section, I instead focus on demonstrating an extension of the SuSiE model, going from the sum of linear relationships of a single variable to the sum of non-linear relationships of a single variable. In fact, this is exactly the model you get if you add together independent instances of the model given in (2.8). There are many convenient ways to use variational inference to learn a non-linear function of a single variable to use as our h_j from model (2.8). For example, Cheng et al. [2022] took inspiration from the SuSiE model and investigated non-linear relationships using neural networks that are functions of a single input variable.

However, I have opted to use my own methods, which I will outline in Section 3.5.2. In short, saving the details for this later section, the method I used learns functions of a single variable which reduce to a piecewise polynomial, the degree of which is learned adaptively.

For this simulation study, I have used the function given in the example from Ravikumar et al. [2009]. The data is generated as:

$$
\begin{aligned}
y_i &= f_1(x_{i,10}) + f_2(x_{i,30}) + f_3(x_{i,50}) + f_4(x_{i,70}) + \epsilon_i \\
\epsilon_i &\overset{iid}{} N(0, \sigma^2) \\
f_1(x) &= \Big(\quad \sin(1.5x) \quad a_1 \Big)/b_1 \\
f_2(x) &= \Big(x^3 + 1.5(x \quad 0.5)^2 \quad a_2 \Big)/b_2 \\
f_3(x) &= \Big(\quad \phi(x; 0.5, 0.8^2) \quad a_3 \Big)/b_3 \\
f_4(x) &= \Big(\sin(\exp f \quad 0.5x g) \quad a_4 \Big)/b_4.
\end{aligned}
$$

Here, $\phi(x; \mu, \sigma^2)$ is the probability density function (pdf) of a Gaussian distribution with mean μ and variance σ^2 evaluated at x, and a_j and b_j are chosen so that for each simulation, the sample mean of $f_j(x)$ is 0 and the sample variance of $f_j(x)$ is 1.

The design matrix $X \; 2 \; \mathsf{R}^{1000 \;\; 100}$ is generated according to

$$
\mathbf{x}_i \overset{iid}{} N_{100}(\mathbf{0}, \mathbf{\Sigma}).
$$

In this simulation, we set $\mathbf{\Sigma}$ to the block-diagonal matrix:

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{S} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{S} \end{bmatrix}$$

Here, $S_{ij} = 0.95^{|i-j|}$ for $i, j \in \{1, \ldots, 20\}$. That is, there are 5 groups of 20 highly correlated predictors; the first four groups have a single non-null variable that has an effect on the response, and the fifth group contains all null variables.

We also set the residual variance σ^2 such that the proportion of variance (PVE) explained was either 0.1, 0.5, or 0.833, where PVE is defined as

$$PVE = \frac{Var\Big(f(\mathbf{x})\Big)}{Var\Big(f(\mathbf{x})\Big) + \sigma^2}.$$

I ran 20 replicates for each value of PVE.

The additive methods I considered in this simulation study were:

- **"SpAM"**: Sparse Additive Models (Ravikumar et al. [2009])

  I used the R package `SAM`. I performed 5-fold cross-validation to estimate the parameter `lambda`;

- **"FLAM"**: Fused Lasso Additive Model (Petersen et al. [2016])

  I used the R package `flam`. I performed 5-fold cross-validation to estimate the parameter `lambda`;

- **"SuSiE Stumps"**: Piecewise constant functions using a sum-of-decision-stumps model (Wang et al. [2020])

  I used the implementation in my R package. Each weak learner is a piecewise constant

function. Essentially, we fit the SuSiE model, but each predictor is an indicator variable for if $X_{ij} \quad c_j$ (which is equivalent to a decision stump); refer to Section 3.5.2, or the section on change point detection in Wang et al. [2020];

- **"SuSiE GAM"**: Piecewise polynomial functions

  I used the defaults in my R package to learn non-linear functions of a single variable (see chapter 3), and then combined them as per Section 2.2.2 and Proposition 2.2.3.

To evaluate the performance of the methods, I used the relative root-mean-squared-error (RRMSE). The root-mean-squared-error (RMSE) is defined as

$$\sqrt{\frac{1}{1000}\sum_{i=1}^{1000}\left(\hat{f}(\mathbf{x}_i^{new}) \quad f(\mathbf{x}_i^{new})\right)^2}, \tag{2.14}$$

where we are summing over unseen testing observations that were generated in the same way as the training observations, $\hat{f}(\ )$ is the fitted function returned by the algorithm, and $f(\ )$ is the true mean response function. For the Bayesian methods, the posterior mean is used as the estimated function value. Note that we are comparing with the true mean of the new observations and are not taking into account any noise; this is to highlight the methods' abilities to recover the true underlying mean.

The RRMSE for algorithm A is defined as

$$\frac{RMSE_{\mathsf{A}}}{\min_{\mathsf{A}'} RMSE_{\mathsf{A}'}}, \tag{2.15}$$

where the minimum in the denominator is taken over all algorithms tested for that particular simulation dataset.

The results are displayed below in Figure 2.1.

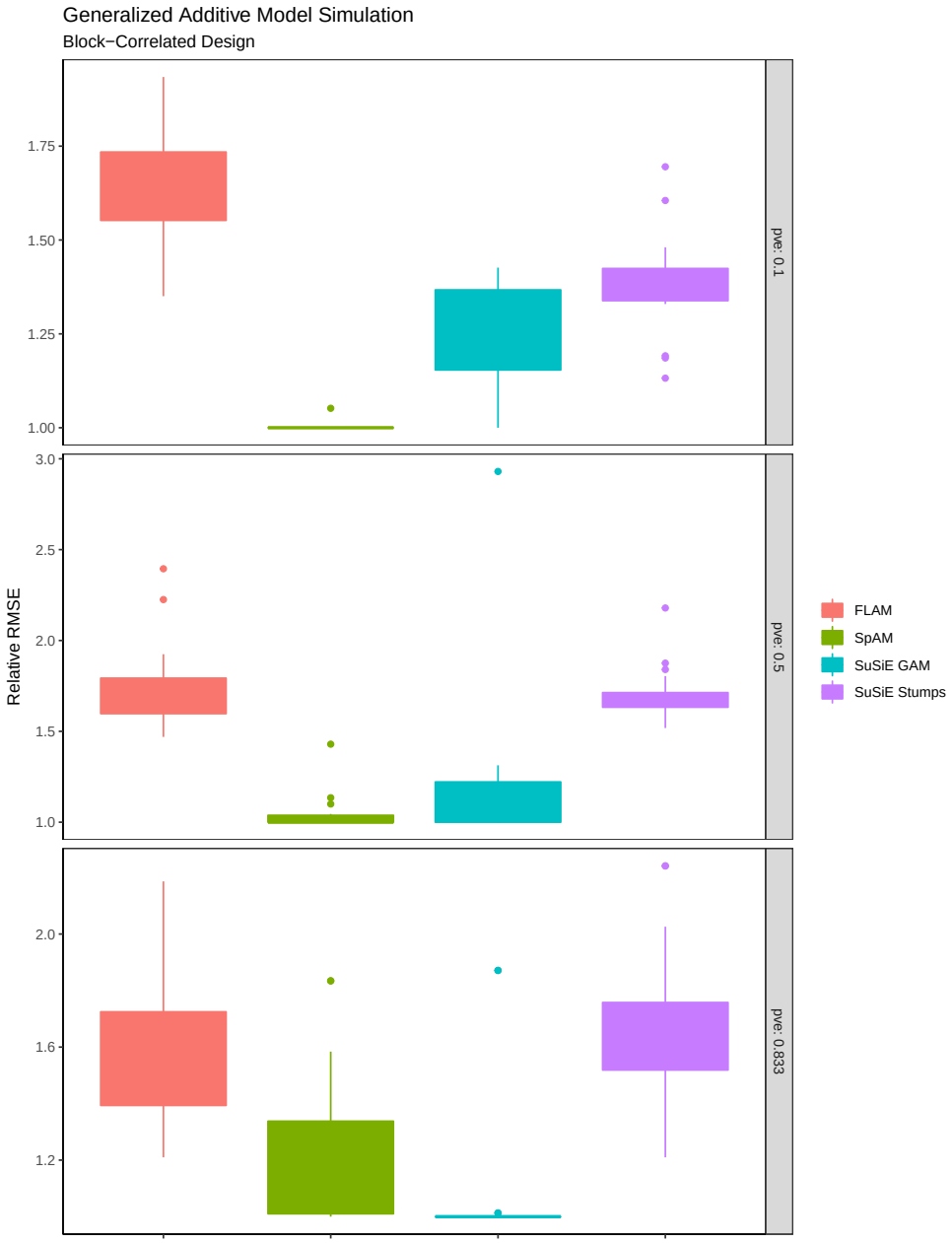

Figure 2.1: **GAM Simulation Relative RMSE** This plot shows the relative RMSE of the different GAM methods in this simulation. We see that the SpAM method appears to be the best in the higher noise settings, but the SuSiE GAM model pulls ahead in the lower noise setting.

2.5 Discussion

In this chapter, we have outlined the variational additive model. This model allows us to add together many Bayesian weak learners. The SuSiE model (Wang et al. [2020]) is a special case of this model, where each weak learner uses the SER from Section 2.2.1. The primary purpose of this chapter was to familiarize the reader with the notation, terminology, setting, etc, so that the next chapter is easier to follow. After introducing these concepts, I provided a brief example of the model's potential usefulness via a small simulation study. This study demonstrated that we can use the variational additive model to fit a generalized additive model in a Bayesian way. And what's more, we can do so quite easily by combining Bayesian weak learners that rely on a single variable, as outlined in Section 2.2.2.

While I have not done much investigation into the potential of additive models, the rapid popularity of the SuSiE model speaks to the promise for such methods. One potential avenue of exploration could be trying to fit a Bayesian sum-of-trees model by having each weak learner be a single Bayesian regression tree (e.g. fitting a BART model with a single tree with traditional MCMC methods). I have not explored the feasibility of fitting each weak learner with MCMC methods, but since we can still get first and second posterior moments from MCMC methods, and we only use the KL-divergence term to monitor convergence, I don't see any conceptual reason why such an approach wouldn't work. It would be interesting to see how such an approach compares with BART. It could also be interesting to try out other methods to fit a GAM in the context of the example from this section, such as using a Bayesian neural network as the non-linear univariate functions, much like in the work of Cheng et al. [2022].

CHAPTER 3

THE VEB-BOOST MODEL

3.1 Introduction

Now that we have introduced how to fit a model in which we add Bayesian weak learners together, we will introduce the ability to also multiply weak learners together. We show that such a model, with an arbitrary sequence of additions and multiplications of weak learners, can be fit by solving the weighted Bayesian linear regression sub-problem. Section 3.2 introduces this weighted Bayesian regression problem. Then, we introduce the full VEB-Boost model in Section 3.3. We discuss how to fit the model with the weighted Bayesian linear regression, how to adapt the complexity of the fit to the complexity of the signal, and briefly introduce the Bayesian weak learner that we use for all simulations and benchmarks. Section 3.4 outlines a few practical considerations: one approach on how to incorporate an intercept into your weak learners, and a way to use a solver for the homoskedastic Gaussian case in the heteroskedastic case. Section 3.5 outlines the R-package that implements the VEB-Boost model, and goes into more detail of the weak learner we used. Finally, Section 3.6 walks through simulated and real-data comparisons with other non-linear/non-parametric regression methods frequently used with tabular data.

3.2 Weighted Bayesian Regression Problem

Just as how the building-block of the variational additive model was the unweighted Bayesian regression problem from Section 2.2, the *weighted* Bayesian regression problem serves as the building-block for the VEB-Boost model. The weighted Bayesian regression problem is formalized as

$$\mathbf{y} = \quad + \tag{3.1a}$$

$$\epsilon \quad \mathcal{N}\left(\mathbf{0}, diag(\ ^2)\right) \tag{3.1b}$$

$$h(\ ) \tag{3.1c}$$

$$g(\ ) \in \mathcal{G}. \tag{3.1d}$$

The only difference between this model and the unweighted Bayesian regression model is that each observation has its own residual variance captured by $\ ^2 \in \mathbb{R}^n_{++}$. To keep the math simple, we will continue to focus on independent observations, but the VEB-Boost model can easily be extended to the case of correlated observations with known dependence structure, i.e. $\mathcal{N}(\mathbf{0}, \mathbf{\Lambda}\ ^{-1})$ for fixed and known precision matrix $\mathbf{\Lambda}$. For details, refer to Section A.3.1, which contains the main theorem and proof of this dissertation.

Analogous to the unweighted Bayesian model, we aim to "solve" this Bayesian regression problem with a function that can estimate a prior distribution $\hat{g} \in \mathcal{G}$ using an empirical Bayes procedure, and then return a variational approximation $q \in \mathcal{Q}$ to the true posterior distribution of , where $\mathcal{Q}$ is an arbitrary variational class. And just as before, VEB-Boost only needs access to $\mathbb{E}_{\ q}[h(\ )]$, $\mathbb{E}_{\ q}[h(\ )^2]$, and $D_{KL}(q\ \|\hat{g})$.

Denote the ELBO of this model as:

$$\begin{aligned} & F_{\mathcal{O}}(g, q; \mathbf{y}, h, \ ^2) \\ = \quad & \frac{n}{2}\log(2\pi) \quad \frac{1}{2}\sum_{i=1}^{n}\log(\sigma_i^2) \quad \frac{1}{2}\mathbb{E}_{\ q}\left[\left\| diag(1/\ )\left(\mathbf{y} \quad h(\ )\right)\right\|_2^2\right] \quad D_{KL}\left(q(\ )\|g(\ )\right). \end{aligned} \tag{3.2}$$

For the purpose of compact notation, let

$$FIT(\mathbf{y}, h, \ ^2, \mathcal{G}, \mathcal{Q})$$
$$:= \left(\underset{g \in \mathcal{G}, q \in \mathcal{Q}}{\arg\max}\ F_0(g, q; \mathbf{y}, h, \ ^2), \ ^- \quad \mathsf{E} \quad _q\ [h(\ )], \ \overline{\ ^2} \quad \mathsf{E} \quad _q\ [h(\ )^2], \ D_{KL}(q \ \| \hat{g}) \right) \quad (3.3)$$

be a function that maximizes the ELBO with respect to the prior distribution $g \in \mathcal{G}$ and variational approximation to the posterior $q \in \mathcal{Q}$ (with $\arg\max \hat{g}$ and q , respectively), and also returns the first and second posterior moments of the weak learner ($^-$ and $\overline{\ ^2}$), as well as the KL-divergence from q to $\hat{g}$.

3.3 The VEB-Boost Model

As in the variational additive model, let a Bayesian weak learner $\ _l \in \mathsf{R}^n$ be defined as $h_l(\ _l)$, where $\ _l \in \mathsf{R}^{p_l}$ is a random vector that we place a prior distribution on and for which we wish to approximate the posterior distribution, and $h_l : \mathsf{R}^{p_l} \to \mathsf{R}^n$ is a fixed function (e.g. $h_l(\ _l) := \mathbf{X}_l \ _l$ for a fixed $\mathbf{X}_l \in \mathsf{R}^{n \ p_l}$).

Using this definition of a Bayesian weak learner, we define a VEB-Boost ensemble learner recursively as: (i) a Bayesian weak learner, (ii) the element-wise sum of two ensemble learners, or (iii) the element-wise (i.e. Schur or Hadamard) product of two ensemble learners. The inclusion of multiplication of weak learners has been explored by others (see, e.g., Friedman and Popescu [2008], Kegl and Busa-Feketa [2009], Nalenz and Villani [2018]). We can leverage our recursive definition to conveniently describe an ensemble learner with a binary expression tree, where internal nodes represent binary operators to combine the ensemble learners defined by the left and right sub-trees ("+" or " " for addition or multiplication, respectively), and the leaf nodes represent weak learners. For instance, Figure 3.1 depicts the ensemble learner

$$\Big(\ _1 \quad (\ _2 + \ _3) \Big) + (\ _4 \quad \ _5).$$

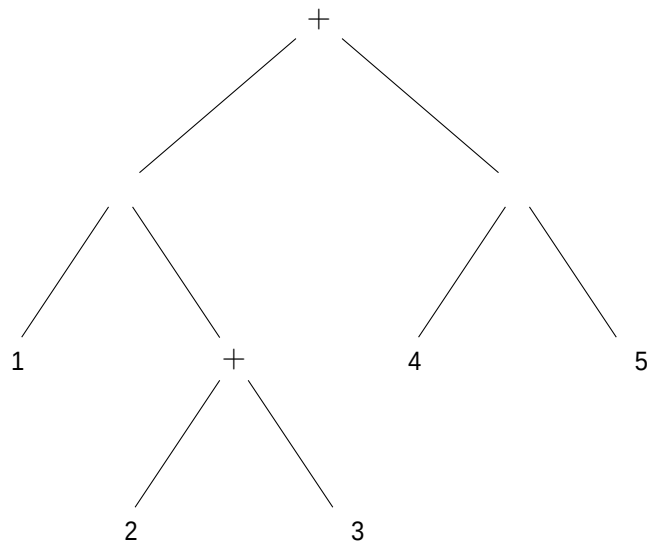

Figure 3.1: **VEB-Boost Ensemble Learner Example** This tree represents the VEB-Boost tree structure $T(\ _1, \ldots, \ _5) = \Big(\ _1 \quad (\ _2 + \ _3)\Big) + (\ _4 \quad \ _5)$.

Let $T(\ _1, \ldots, \ _L) : \mathsf{R}^{n \ L} \ ! \ \mathsf{R}^n$ be the function that combines the weak learners $_1, \ldots, \ _L$ using the appropriate sequence of additions and multiplications to yield the desired ensemble learner (e.g. for the ensemble learner in Figure 3.1, $T(\ _1, \ldots, \ _5) := (\ _1 \quad (\ _2 + \ _3)) + (\ _4 \quad \ _5)$). For now, we will assume $T(\ )$ is fixed, but in Section 3.3.3, we will discuss ways in which $T(\ )$ can be "grown" in an adaptive fashion.

Given our response $\mathbf{y} \ 2 \ \mathsf{R}^n$, functions $h_l : \mathsf{R}^{p_l} \ ! \ \mathsf{R}^n$, expression tree $T : \mathsf{R}^{n \ L} \ ! \ \mathsf{R}^n$, and residual variance vector $\ ^2$, our full model is as follows:

$$\mathbf{y} = T(\ _1, \ldots, \ _L) + \tag{3.4a}$$

$$N\left(\mathbf{0}, diag(\ ^2)\right) \tag{3.4b}$$

$$_l \quad h_l(\ _l), \qquad l = 1, \ldots, L \tag{3.4c}$$

$$_l \ ^? \ g_l(\ ) \ 2 \ \mathsf{G}_l, \qquad l = 1, \ldots, L. \tag{3.4d}$$

Here, $g_l(\ )$ is the prior distribution of $\ _l$, which belongs to a class of distributions G_l. We can either fix $g_l(\ )$ by having $\mathsf{G}_l \quad \mathsf{f} g_l(\ ) \mathsf{g}$ or estimate it with an empirical Bayes procedure.

Our goals are to estimate the prior distributions g_l using an empirical Bayes procedure to obtain $\hat{g}_l$, and use these estimated prior distributions to calculate the posterior distribution

$$p(\ _1, \ldots, \ _L | \mathbf{y}, T, \hat{g}_1, \ldots, \hat{g}_L, h_1, \ldots, h_L, \ ^2). \tag{3.5}$$

3.3.1 Variational Approximation to the Posterior Distribution

Rather than calculating the exact posterior distribution given in equation (3.5), we instead find a variational approximation. As in the variational additive model, we use the variational class of distributions that factorize over $\ _l$, where each factor in turn belongs to its own variational class $\mathcal{Q}_l$, i.e.

$$\mathcal{Q} = \left\{ q \ \middle| \ q(\ _1, \ldots, \ _L) = \prod_{l=1}^{L} q_l(\ _l), \quad q_l \in \mathcal{Q}_l \right\}. \tag{3.6}$$

In the VEB-Boost model outlined in equations (3.4a) – (3.4d), there are additional parameters that must be estimated, e.g. any parameters needed to characterize $g_1, \ldots, g_L$. Using the variational empirical Bayes framework outlined in Section 1.1.3, we can combine the empirical Bayes step and variational inference steps into a unified procedure of maximizing the ELBO. Note that the approaches here allow for global parameters $\in \Theta$ to be included and estimated. For example, if you are using the methods of Carbonetto and Stephens [2012] (so each weak learner corresponds to the effect of an individual variable) then the shared prior probability of the effect being 0 is a global parameter which you can estimate by maximizing the ELBO. However, for simplicity of notation, I exclude such a . Our ELBO now takes the form

$$F(g_1, q_1, \ldots, g_L, q_L; \mathbf{y}, h_1, \ldots, h_L, \ ^2, T). \tag{3.7}$$

Thus, our goal to approximate the posterior distribution (3.5) is accomplished by com-

puting

$$(\hat{g}_1, q_1, \ldots, \hat{g}_L, q_L, \ ^2) \ 2 \ \underset{g_l 2 G_l, q_l 2 Q_l, \ ^2 2 \Theta}{\arg\max} \ F(g_1, q_1, \ldots, g_L, q_L, \ ^2; \mathbf{y}, h_1, \ldots, h_L, T). \quad (3.8)$$

We refer to maximizing this objective function as "solving" the variational empirical Bayes problem associated with model (3.4).

Note that when optimizing over ^{2}, we typically either (i) fix it to a pre-specified value/vector (i.e. $\Theta = f \ ^2 \ : \ ^2 = \ ^2$ for fixed $^2 \ 2 \ R^n_{++} g$), or (ii) force $^2 = \sigma^2 \mathbf{1}$ for a $\sigma^2 > 0$ to be optimized over (i.e. $\Theta = f \ ^2 \ : \ ^2 = \sigma^2 \mathbf{1}$ for $\sigma^2 > 0 g$). But there are other options that I have considered but not yet explored, which I touch upon briefly in Section 3.7.

3.3.2 Fitting a VEB-Boost Ensemble Learner

The form of the ELBO in the VEB-Boost model is given below, which follows from (1.6):

$$\frac{n}{2}\log(2\pi) \quad \frac{1}{2}\sum_{i=1}^{n}\log(\sigma_i^2) \quad \frac{1}{2}E_q\Big[\Big\|diag(1/\ )\Big(\mathbf{y} \quad T(\ _1, \ldots, \ _L)\Big)\Big\|_2^2\Big] \quad \sum_{l=1}^{L} D_{KL}(q_l k g_l). \quad (3.9)$$

Just like how the ELBO of the additive model (when viewed as a function of (g_l, q_l) holding everything else fixed) is equal (up to a constant) to the ELBO in the unweighted Bayesian regression problem where the response is a form of residual, we can view the ELBO of the VEB-Boost model as being equal to the ELBO in the weighted Bayesian regression problem where the response and residual variance are a form of residual.

As an illustrative example, consider the simple learner

$$\mathbf{y} = \ _1 \quad _2 + \ , \qquad N(\mathbf{0}, \sigma^2 I_n).$$

If we want to fit the weak learner $\quad_2$, we can perform element-wise division by $\quad_1$ from both sides, which leaves

$$\mathbf{y}/\quad_1 = \quad_2 + \quad/\quad_1, \qquad \quad N(\mathbf{0}, \sigma^2 I_n).$$

It turns out that if we multiply by $\mathsf{E}_{q_1}[\quad_1]/\mathsf{E}_{q_1}[\quad_1^2]$ instead of dividing by $\quad_1$ on the left hand side, and divide by $\sqrt{\mathsf{E}_{q_1}[\quad_1^2]}$ instead of $\quad_1$ on the right hand side, then this is exactly the correspondence we get between the ELBOs.

Theorem 3.3.1 (VEB-Boost Model ELBO Equivalence)**.** *Let* $F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \quad^2, T)$ *denote the ELBO of the VEB-Boost model* (3.4)*, and let* $F_l(g_l, q_l; g\quad_l, q\quad_l, \boldsymbol{y}, h_1, \ldots, h_L, \quad^2, T)$ *denote this ELBO with respect to* (g_l, q_l) *while holding everything else xed. And let* $F_0(g, q; \boldsymbol{y}, h, \quad^2)$ *denote the ELBO from the weighted regression model* (3.1) *with response* $\boldsymbol{y}$*, residual variance* $\quad^2$*, and mean response* $\quad h(\quad)$*. Then*

$$F_l(g_l, q_l; g\quad_l, q\quad_l, \boldsymbol{y}, h_1, \ldots, h_L, \quad^2, T) = F_0(g_l, q_l; \tilde{\boldsymbol{y}}, h_l, \tilde{}^2) + c$$

for a particular $\tilde{\boldsymbol{y}}$ *and* $\tilde{}^2$ *(given in Theorem 3.3.2), and* c *is a constant term in* (g_l, q_l)*.*

Corollary 3.3.1.1 (Block Coordinate Maximization of the VEB-Boost Model ELBO)**.** *In the same setting as Theorem 3.3.1,*

$$\underset{g_l 2 \mathcal{G}_l, q_l 2 \mathcal{Q}_l}{\arg\max}\ F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \quad^2, T) = \underset{g_l 2 \mathcal{G}_l, q_l 2 \mathcal{Q}_l}{\arg\max}\ F_0(g_l, q_l; \tilde{\boldsymbol{y}}, h_l, \tilde{}^2).$$

In other words, in order to perform a block coordinate maximization step of the full ELBO in the VEB-Boost model (3.4) over the prior distribution $g_l \ 2 \ \mathcal{G}_l$ and variational approximation $q_l \ 2 \ \mathcal{Q}_l$, we simply need to be able to solve the weighted Bayesian regression problem for a single weak learner using the response $\tilde{\mathbf{y}}$ and residual variance $\tilde{}^2$.

Corollary 3.3.1.2 (Gradient of the VEB-Boost Model ELBO)**.** *In the same setting as Theorem 3.3.1, suppose that the prior family* $\mathcal{G}_l$ *and variational family* $\mathcal{Q}_l$ *are nite-dimensional*

parametric families. Then

$$\nabla_{g_l,q_l} F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \quad ^2, T) = \nabla_{g_l,q_l} F_0(g_l, q_l; \tilde{\boldsymbol{y}}, h_l, \tilde{} ^2),$$

where ∇_{g_l,q_l} *refers to the gradient with respect to the parameters of the distributions* $g_l \in G_l$ *and* $q_l \in Q_l$*.*

This means that if one wanted to maximize the ELBO using a first-order method instead of (block) coordinate maximization, it is sufficient to be able to calculate the gradient in the context of the weighted Bayesian regression problem.

To derive what exactly $\tilde{\mathbf{y}}$ and $\tilde{} ^2$ are, we must first introduce some notation. Recall how we can represent the arithmetic of the ensemble learner in the form of a binary tree (see Section 3.3, in particular Figure 3.1). Let $s = d_1 \quad d_K \in \{0,1\}^K$ represent the path string from the root of the tree to a node in the tree, where the directions d_k are either 0 (for going left in the tree) or 1 (for going right in the tree). Let the empty path string $s =$ refer to the root of the tree, which I sometimes refer to as d_0. Let $\bar{d}_k = 1 \quad d_k$ indicate switching directions (i.e. changing a left to a right, or a right to a left). Let $_s$ be the VEB-Boost ensemble learner whose root node is located at path string s. And let $_s \in \{+, \}$ denote the operator, $+$ or , at the internal node with path string s.

As a concrete example of this notation in action, in Figure 3.1, the weak learner $_2$ can be found at path string $s = 010$ (starting from the root, we go left-right-left to get to $_2$). Thinking in terms of the recursive definition of a VEB-Boost ensemble learner, we can represent the tree in terms of $_2$ as

$$_1 + \Big(\quad 00 \quad (\quad _2 + \quad 011) \Big).$$

In general, for a weak learner $_l$ at path string $s = d_1 \quad d_K$, we can describe the tree in

terms of $_l$ as

$$d_1 \quad \left(\quad d_1 d_2 \quad d_1 \left(\quad d_1 d_2 d_3 \quad d_1 d_2 \quad \left(\quad l \quad d_1 d_2 \quad d_K \quad 1 \quad d_1 d_2 \quad d_K \quad 1 d_K \right) \quad \right) \right). \tag{3.10}$$

This is easy to see once we see the correspondence between starting at a weak learner $_l$ and working our way up the tree (combining learners as we go), and working from the inside out of the expression above.

Adding one final piece of notation, for a given operator and current variational distribution over $_2$ (with first and second posterior moments $\overline{\ _2}$ and $\overline{\ \ _2^2}$), define

$$1 \overline{\quad} \ 2 = \begin{cases} 1 \quad \overline{\ \ 2}, & \text{if} \quad = + \\ 1 \quad \frac{\overline{\ \ 2}}{\overline{\ \ 2^2}}, & \text{if} \quad = \ . \end{cases}$$

With this notation established, we now have the ability to describe exactly what $\tilde{\mathbf{y}}$ and $\tilde{}^2$ are in Theorem 3.3.1:

Theorem 3.3.2 (VEB-Boost Residualized Response and Variance)**.** *In the context of Theorem 3.3.1, consider a weak learner $_l$ with path string $s = d_1 \quad d_K$ in the VEB-Boost ensemble learner $T(\ _1, \ldots, \ _L)$. Then the response $\tilde{\boldsymbol{y}}$ we need for equivalence between the full ELBO and the weighted Bayesian regression ELBO is*

$$\tilde{\boldsymbol{y}} = \left(\quad \left((\boldsymbol{y} \overline{\quad} \ _{d_1}) \overline{\ } d_1 \quad d_1 d_2 \right) \overline{\ } d_1 d_2 \quad \overline{\ } d_1 \quad d_K \quad 1 \quad d_1 \quad d_K \quad 1 d_K \right).$$

In order to get the residual variance to use for this equivalence to hold, let

$$= \quad \prod_{\substack{i=0 \\ d_0 \ d_i =}}^{K \ 1} \overline{\ \ _{d_0 \quad d_i d_{i+1}}^2}$$

be the Schur product of the second posterior moments of all ensemble learners you get by

traveling along path string s, and whenever you encounter an internal node with operator , take the child of that node that is in the opposite direction of $_l$. Then

$$\tilde{}^2 = \quad ^2 \quad \frac{1}{}.$$

For a proof of this theorem (in a more general form), refer to the Section A.3.1.

Under this framework, we can also estimate any global parameters $\in \Theta$. Holding $\hat{g}, q$ fixed, we can view

$$F\ (\ ; \mathbf{y}, T, h_1, \ldots, h_L, \hat{g}, q\ ) := F(\hat{g}, q\ ,\ ; \mathbf{y}, T, h_1, \ldots, h_L).$$

We can then add a coordinate ascent step to optimize the ELBO with respect to these global parameters to get $\hat{} \in \Theta$. This can be viewed as an EM algorithm in which the E-step is approximate (see section 3.1.3 of Wang et al. [2020]). The same can be done for ^{2}.

For each weak learner $_l$, let $FIT : \left(h_l, \mathbf{y}, \ ^2, G_l, Q_l\right) \ ! \ \left(\hat{g}_l, q_l\ , \overline{\ _l}, \overline{\ ^2_l}, D_{KL}(q_l \| \hat{g}_l)\right)$ be a function that takes in the required input for the weighted Bayesian regression problem (3.1) and returns the prior estimated by empirical Bayes $\hat{g}_l \in G_l$, the approximation to the posterior $q_l \in Q_l$, the first and second posterior moments of $_l$ under $_l \quad q_l$, and $D_{KL}(q_l \| \hat{g}_l)$. Combining these fitting functions and Theorems 3.3.1 and 3.3.2 leads to the coordinate ascent algorithm for maximizing the ELBO with a fixed tree structure T.

Algorithm 2: VEB-Boost Coordinate Ascent Algorithm (fixed $T(\ )$)

Require: Data $\mathbf{y}$; functions $h_l(\ )$, prior classes $\mathcal{G}_l$, variational classes $\mathcal{Q}_l, \quad l = 1, \ldots, L$.

Require: Tree structure $T(\ )$; initial variance $\ ^2$.

Require: Functions $FIT : \left(h_l, \mathbf{y}, \ ^2, \mathcal{G}_l, \mathcal{Q}_l\right) \ ! \ \left(\hat{g}_l, q_l\ , \overline{\ _l}, \overline{\ ^2_l}, D_{KL}(q_l \ k\hat{g}_l)\right)$ that solve the weighted Bayesian regression problem; see Corollary 3.3.1.1

Initialize posterior means $\overline{\ _l}$, $\overline{\ ^2_l}$, for $l = 1, \ldots, L$;

Initialize variance vector to $\hat{\ }^2 := \ ^2$;

repeat

  for *l in* $1, \ldots, L$ **do**

    Compute $(\tilde{\mathbf{y}}, \ \tilde{\ }^2)$ given $T, \mathbf{y}, \hat{\ }^2, \overline{\ _k}, \overline{\ ^2_k}, k \ 6= l$; // see Theorem 3.3.2

    $(\hat{g}_l, q_l\ , \overline{\ _l}, \overline{\ ^2_l}, D_{KL}(q_l \ k\hat{g}_l)) \quad FIT(h_l, \tilde{\mathbf{y}}, \ \tilde{\ }^2, \mathcal{G}_l, \mathcal{Q}_l)$;

  Update $\hat{\ }^2$; // optional;

until *convergence criterion satis ed*;

return $q_1, \ldots, q_L$.

Using the formulation of the ELBO given in equation (3.9), it is easy to see that if we are restricting $\ ^2 = \sigma^2\mathbf{1}$, then we can maximize the ELBO with respect to σ^2 by setting $\hat{\sigma}^2 := \frac{1}{n}\left(\overline{\mathbf{T}^2} \quad 2\overline{\mathbf{T}}\mathbf{y} \quad \mathbf{y}^2\right)$, where $\overline{\mathbf{T}}$ and $\overline{\mathbf{T}^2}$ are the first and second posterior moments of the entire ensemble learner.

3.3.3 Growing the VEB-Boost Ensemble Learner

Up to now, we have treated the tree structure T as fixed; this is anologous to fixing the number of decision trees and their depths in a boosting ensemble before fitting. However, in many cases it is beneficial to "grow" the tree T in such a way that the overall fit to the data is improved; this is analogous to sequentially adding more trees and splits to a boosting ensemble until some heuristic is met.

In the boosting framework, if the fit currently has L weak learners, $_1, \ldots, _L$, and you wish to add another, this can be viewed as splitting the last weak learner into the sum of two weak learners, i.e. changing $_L$ into $_L + _{L+1}$, and then fitting $_{L+1}$.

Using this view as motivation, we propose a similar weak learning splitting scheme to grow the tree structure T, but allow for the possibility of multiplying weak learners as well. At each step we wish to grow the tree T, we propose splitting each learner $_l$ into one of the following: (i) $_l + _{l,1}$; (ii) $_l \quad _{l,2}$; or (iii) $(_l \quad _{l,2}) + _{l,1}$, for new weak learners $_{l,1}$ (initialized to identically $\mathbf{0}$) and $_{l,2}$ (initialized to identically $\mathbf{1}$). We then proceed with Algorithm 2 using this new tree structure, but initialize the posterior moments from the existing weak learners with their moments at convergence from the previous tree structure. This can be viewed as warm-starting the optimization problem of maximizing the ELBO using the new tree structure obtained after this splitting procedure.

Another way to look at this is to note that our VEB ensemble learner is invariant to (i) adding a weak learner that is identically $\mathbf{0}$ to any weak learner $_l$, (ii) multiplying by a weak learner that is identically $\mathbf{1}$ to any weak learner $_l$, or (iii) any combination of (i) and (ii). So we can pretend that there are "hidden" learners that are adding 0 and multiplying by 1. Then, as we grow the tree, we "activate" these suppressed weak learners by extending their prior class from being a fixed constant (0 or 1) to a non-degenerate prior class like the other weak learners. Assuming that the constant fits of $\mathbf{0}$ and $\mathbf{1}$ are part of the non-degenerate prior class we assign to them (which is almost always the case), then this can be viewed as an ascent step that can only increase the ELBO.

When determining which weak learners we should consider splitting when growing the tree, the empirical Bayes portion of the framework suggests an intuitive rule. When we look at each weak learner's fitted prior distribution $\hat{g}_l$ at convergence, if the weak learner is close to being a constant value (e.g. $Var_{ \ \hat{g}_l}(_l) \quad 0$ or $Var_{ \ \hat{g}_l}(h_l(_l)) \quad 0$), then we can "lock" this weak learner and not split it when growing the tree, since the portion of the

overall fit that this area of the tree captures is likely not going to improve by making it more complex. For weak learners that are not essentially a constant, we propose splitting them since the fit is likely to improve by such a split. We can then continue to grow the tree in this way after each round of convergence in Algorithm 2 until each weak learner that we could add would be essentially a constant, or the ELBO does not appreciably increase by splitting the weak learners.

3.4 Some Practical Considerations

In this section, we outline a few practical considerations. These may prove useful for practitioners who wish to implement their own weighted linear regression weak learners. First, we talk about one way to include an intercept with each weak learner. And second, we show how to use a homoskedastic linear regression solver in the setting with arbitrary Gaussian errors.

3.4.1 Including an Intercept in a Linear Weak Learner

Most regression models include an intercept term and/or additional covariates that can be thought of as nuisance parameters (e.g. including the sex, age, top 10 genotype principal components, etc in a genome-wide association study, Price et al. [2006]). When using Bayesian regression methods, we often choose not to penalize these coefficients as strongly, and instead put a flat prior on them (Liseo [1993]). As a practical matter, when only dealing with an intercept, this can be accomplished by centering the response $\mathbf{y}$ and all columns of the design matrix $\mathbf{X}$; such an approach can be motivated by integrating out the intercept under a flat prior (Chipman et al. [2001]). However, doing this centering step once at the beginning is not suitable for our purposes, because

i) the residual variance for each weak learner is heteroskedastic; and

ii) the residual variance changes each time we fit a weak learner due to the other learners

changing.

My recommendation on how to incorporate an intercept and/or additional covariate effects in a linear weak learner is motivated by the above approach, but tailored to the VEB-Boost setting where the variance is heteroskedastic and constantly changing. I re-write the weighted Bayesian regression problem below, but now include an intercept and other effects we wish to put a flat prior on, $\in \mathbb{R}^q$, along with a covariate matrix $\mathbf{Z} \in \mathbb{R}^{n \times q}$ (e.g. for just an intercept, $\mathbf{Z}$ is a column of 1's).

$$\mathbf{y} = \mathbf{X} + \mathbf{Z} + \tag{3.11a}$$

$$g(\cdot) \in G \tag{3.11b}$$

$$N(0, \boldsymbol{\Lambda}^{-1}) \tag{3.11c}$$

$$\propto \delta_{\overline{\mathbf{y}} - \overline{\mathbf{X}}} \text{ (i.e. a point-mass at } \overline{\mathbf{y}} - \overline{\mathbf{X}}\text{)}, \tag{3.11d}$$

where

$$\overline{\mathbf{y}} = (\mathbf{Z}^T \boldsymbol{\Lambda} \mathbf{Z})^{-1} \mathbf{Z}^T \boldsymbol{\Lambda} \mathbf{y} \quad \text{and} \quad \overline{\mathbf{X}} = (\mathbf{Z}^T \boldsymbol{\Lambda} \mathbf{Z})^{-1} \mathbf{Z}^T \boldsymbol{\Lambda} \mathbf{X}. \tag{3.11e}$$

To serve as motivation, consider the model above, but instead, place a Jeffreys prior on $\propto$ (which will be an improper prior $\propto |\mathbf{Z}^T \boldsymbol{\Lambda} \mathbf{Z}|^{1/2} \propto 1$). Now, we can marginalize out of the likelihood. Doing so, and carrying out the algebra, we get:

$$
\begin{aligned}
P(\mathbf{y}|\mathbf{X}, \quad, \mathbf{\Lambda}) &= \int P(\mathbf{y}|\mathbf{X}, \quad, \mathbf{\Lambda}, \quad) P(\quad) d \\
&\propto \int (2\pi)^{\quad n/2} |\mathbf{\Lambda}|^{1/2} \exp\Big\{ \quad \frac{1}{2} (\mathbf{y} \quad \mathbf{X} \quad \mathbf{Z} \quad)^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad \mathbf{Z} \quad) \Big\} d \\
&= (2\pi)^{\quad n/2} |\mathbf{\Lambda}|^{1/2} \exp\Big\{ \quad \frac{1}{2} (\mathbf{y} \quad \mathbf{X} \quad)^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big\} \\
&\quad \int \exp\Big\{ \quad \frac{1}{2} \Big[\quad ^T (\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}) \quad 2 \quad ^T \mathbf{Z}^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big] \Big\} d \\
&= [\text{Let } \hat{\mathbf{\Lambda}} := \mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}] \\
&= (2\pi)^{\quad n/2} |\mathbf{\Lambda}|^{1/2} \exp\Big\{ \quad \frac{1}{2} (\mathbf{y} \quad \mathbf{X} \quad)^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big\} \\
&\quad \int \exp\Big\{ \quad \frac{1}{2} \Big[\Big(\quad \hat{\mathbf{\Lambda}} \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big)^T \hat{\mathbf{\Lambda}} \Big(\quad \hat{\mathbf{\Lambda}} \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big) \Big] \Big\} \\
&\quad \exp\Big\{ \frac{1}{2} (\mathbf{y} \quad \mathbf{X} \quad)^T \mathbf{\Lambda} \mathbf{Z} \hat{\mathbf{\Lambda}} \quad ^1 \hat{\mathbf{\Lambda}} \hat{\mathbf{\Lambda}} \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} (\mathbf{y} \quad \mathbf{X} \quad) \Big\} \quad \frac{|\mathbf{\Lambda}|^{1/2}}{|\hat{\mathbf{\Lambda}}|^{1/2}} d \\
&\propto (2\pi)^{\quad n/2} |\mathbf{\Lambda}|^{1/2} \exp\Big\{ \quad \frac{1}{2} (\mathbf{y} \quad \mathbf{X} \quad)^T \Big(\mathbf{\Lambda} \quad \mathbf{\Lambda} \mathbf{Z} \hat{\mathbf{\Lambda}} \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} \Big) (\mathbf{y} \quad \mathbf{X} \quad) \Big\}.
\end{aligned}
$$

Focusing on the matrix in the exponential term, we can simplify it as

$$
\begin{aligned}
\mathbf{\Lambda} \quad \mathbf{\Lambda} \mathbf{Z} \hat{\mathbf{\Lambda}} \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} &= \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z} \Big) \quad ^1 \mathbf{Z}^T \Big] \mathbf{\Lambda} \\
&= \Big[\text{Noting that } \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \Big] \text{ is idempotent} \Big] \\
&= \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \Big] \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \Big] \mathbf{\Lambda} \\
&= \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \Big] \Big[\mathbf{\Lambda} \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} \Big] \\
&= \Big[I_n \quad \mathbf{\Lambda} \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \Big] \mathbf{\Lambda} \Big[I_n \quad \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \mathbf{\Lambda} \Big] \\
&= [\mathbf{P} := I_n \quad \mathbf{Z} \Big(\mathbf{Z}^T \mathbf{\Lambda} \mathbf{Z}^T \Big) \quad ^1 \mathbf{Z}^T \mathbf{\Lambda}] \\
&= \mathbf{P}^T \mathbf{\Lambda} \mathbf{P}.
\end{aligned}
$$

Thus, turning back to simplifying the marginal likelihood, we get

$$
\begin{aligned}
&(2\pi)^{-n/2}|\boldsymbol{\Lambda}|^{1/2}\exp\left\{-\frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T\left(\boldsymbol{\Lambda}-\boldsymbol{\Lambda}\mathbf{Z}\hat{\boldsymbol{\Lambda}}^{-1}\mathbf{Z}^T\boldsymbol{\Lambda}\right)(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right\}\\
/\,&(2\pi)^{-n/2}|\boldsymbol{\Lambda}|^{1/2}\exp\left\{-\frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T\mathbf{P}^T\boldsymbol{\Lambda}\mathbf{P}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right\}\\
=&N\left(\mathbf{P}\mathbf{y}\,;\,\mathbf{P}\mathbf{X}\boldsymbol{\beta},\,\boldsymbol{\Lambda}^{-1}\right).
\end{aligned}
$$

So marginalizing out $\boldsymbol{\beta}$ leaves us with a likelihood that is proportional to the likelihood of observing $\mathbf{Py}$ coming from a Gaussian with mean $\mathbf{PX}\boldsymbol{\beta}$ and precision $\boldsymbol{\Lambda}$. Expanding $\tilde{\mathbf{y}} := \mathbf{Py}$ and $\tilde{\mathbf{X}} := \mathbf{PX}$, we get

$$\tilde{\mathbf{y}} = \mathbf{Py} = \mathbf{y}-\mathbf{Z}\left(\mathbf{Z}^T\boldsymbol{\Lambda}\mathbf{Z}^T\right)^{-1}\mathbf{Z}^T\boldsymbol{\Lambda}\mathbf{y} = \mathbf{y}-\mathbf{Z}\overline{\mathbf{y}}$$

and

$$\tilde{\mathbf{X}} = \mathbf{Xy} = \mathbf{X}-\mathbf{Z}\left(\mathbf{Z}^T\boldsymbol{\Lambda}\mathbf{Z}^T\right)^{-1}\mathbf{Z}^T\boldsymbol{\Lambda}\mathbf{X} = \mathbf{X}-\mathbf{Z}\overline{\mathbf{X}}.$$

This means that in order to fit the model, we can use response $\tilde{\mathbf{y}}$ and design matrix $\tilde{\mathbf{X}}$ and ignore $\boldsymbol{\beta}$ and $\mathbf{Z}$. We can also confirm that $\boldsymbol{\beta}\sim N\left(\overline{\mathbf{y}}-\overline{\mathbf{X}}\boldsymbol{\beta},(\mathbf{Z}^T\boldsymbol{\Lambda}\mathbf{Z})^{-1}\right)$.

Note that if $\mathbf{Z} = \mathbf{1}$ (i.e. there are no covariates, only an intercept) and $\boldsymbol{\Lambda}$ is diagonal, then this amounts to centering $\mathbf{y}$ by subtracting its weighted mean and centering each column of $\mathbf{X}$ by subtracting its weighted mean, where the weights are the diagonal elements of $\boldsymbol{\Lambda}$ (i.e. the precisions of the observations). And then the posterior mean of $\boldsymbol{\beta}$ is equal to the weighted mean of $\mathbf{y}$ - (vector of weighted column means from $\mathbf{X}$)$^T\boldsymbol{\beta}$.

With the above result, one might be tempted to use such a Jeffreys prior on $\boldsymbol{\beta}$. However, in my initial experiments, this occasionally resulted in convergence issues (e.g. the ELBO wasn't increasing at every step, and/or some infinite or NA variances arose in the fitting procedure). To circumvent this issue, I have taken the approach outlined in equation (3.11), which has the conditional distribution of $\boldsymbol{\beta}$ being a point-mass at $\overline{\mathbf{y}}-\overline{\mathbf{X}}\boldsymbol{\beta}$, which is the

conditional mean we get in the Jeffreys prior case. Doing so allows us to still slot in $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{X}}$ into our weighted Bayesian regression solver and ignore $\mathbf{Z}$. The only difference is how we compute the posterior second moment of our observations.

3.4.2 Using a Homoskedastic Linear Solver on Arbitrary Noise Gaussian Data

Since the VEB-Boost model requires a weak learner "solver" for the weighted Bayesian regression problem given in (3.1), it may seem that the practitioner has to fully derive a new solution entirely separate from a solver for the unweighted (i.e. homoskedastic) case. But it turns out that if we are fitting a linear model, the solver for the homoskedastic case can be used to find the solution! We can even use the homoskedastic solver in the context of arbitrary (possible correlated) Gaussian errors.

Suppose that our data follows the linear model

$$\mathbf{y} = \mathbf{X} \quad + \quad .$$

But now, let $\quad N_n(\mathbf{0}, \mathbf{\Lambda}^{\ 1})$ for an arbitrary full-rank precision matrix $\mathbf{\Lambda} \in S^n_{++}$. We can write the log-likelihood of this model as

$$\begin{aligned} l(\mathbf{y}; \mathbf{X}, \quad , \mathbf{\Lambda}) = & \quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\mathbf{\Lambda}| \quad \frac{1}{2}(\mathbf{y} \quad \mathbf{X} \quad)^T\mathbf{\Lambda}(\mathbf{y} \quad \mathbf{X} \quad) && \text{(3.12a)} \\ = & \quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\mathbf{\Lambda}| \quad \frac{1}{2}(\mathbf{y} \quad \mathbf{X} \quad)^T\mathbf{\Lambda}^{1/2}\mathbf{\Lambda}^{1/2}(\mathbf{y} \quad \mathbf{X} \quad) && \text{(3.12b)} \\ = & \quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\mathbf{\Lambda}| \quad \frac{1}{2}\log|\mathbf{I}_n| \\ & \quad \frac{1}{2}(\mathbf{\Lambda}^{1/2}\mathbf{y} \quad \mathbf{\Lambda}^{1/2}\mathbf{X} \quad)^T\mathbf{I}_n(\mathbf{\Lambda}^{1/2}\mathbf{y} \quad \mathbf{\Lambda}^{1/2}\mathbf{X} \quad). && \text{(3.12c)} \end{aligned}$$

Up to a constant in , we can recognize this as the likelihood for the model

$$\tilde{\mathbf{y}} = \tilde{\mathbf{X}} \quad + \tilde{},$$

where $\tilde{\mathbf{y}} := \mathbf{\Lambda}^{1/2}\mathbf{y}$, $\tilde{\mathbf{X}} := \mathbf{\Lambda}^{1/2}\mathbf{X}$, and $\tilde{}$ $N_n(\mathbf{0}, \mathbf{I}_n)$; in particular, if $\mathbf{\Lambda} = diag(1/\sigma_1^2, \ldots, 1/\sigma_n^2)$, then this corresponds to $\tilde{y}_i = y_i/\sigma_i$ and $\tilde{\mathbf{x}}_i^T = \mathbf{x}_i^T/\sigma_i$ (i.e. scaling observations and rows by their corresponding standard deviations). Thus, one can simply use their homoskedastic solver with response $\tilde{\mathbf{y}}$, design matrix $\tilde{\mathbf{X}}$, and residual variance σ^2 1 in order to solve this problem. In the literature, this is referred to as the "whitening" transformation (see, e.g., Kessy et al. [2018]).

While the current VEB-Boost package does not implement the case of correlated errors, it is straightforward to extend the full model to this case. However, in order for VEB-Boost to avoid taking matrix inverses and/or performing costly matrix factorizations, a solver that is designed to be able to handle correlated errors should take in as inputs: $\mathbf{X}$, $\mathbf{\Lambda}$, and $\mathbf{\Lambda y}$ (instead of $\mathbf{y}$). This is often a straight-forward task in the context of a linear model. To see this, note that

$$\begin{aligned} q(\quad = \mathbf{c}) \; &/ \; \exp\Big\{ \log g(\mathbf{c}) + \log p(\mathbf{y} j \mathbf{X}, \quad = \mathbf{c}) \Big\} \\ &/ \; \exp\Big\{ \log g(\mathbf{c}) \quad \frac{1}{2}(\mathbf{\Lambda}^{1/2}\mathbf{y} \quad \mathbf{\Lambda}^{1/2}\mathbf{X}\mathbf{c})^T \mathbf{I}_n (\mathbf{\Lambda}^{1/2}\mathbf{y} \quad \mathbf{\Lambda}^{1/2}\mathbf{X}\mathbf{c}) \Big\} \\ &/ \; \exp\Big\{ \log g(\mathbf{c}) \quad \frac{1}{2}\mathbf{c}^T\mathbf{X}^T\mathbf{\Lambda}\mathbf{X}\mathbf{c} + \mathbf{c}^T\mathbf{X}^T\mathbf{\Lambda}\mathbf{y} \Big\}, \end{aligned}$$

which we can calculate if we know $\mathbf{X}$, $\mathbf{\Lambda}$, and $\mathbf{\Lambda y}$.

This modification allows us to avoid taking the matrix inverses needed to calculate the necessary pseudo-response and -precision to use when fitting a weak learner (see the proof for Theorem A.3.1). The VEB-Boost machinery can, in principle, then keep track of $\mathbf{\Lambda}$ and $\mathbf{\Lambda y}$ for each node in the ensemble learner instead (although again, this is not currently implemented in the R package).

3.5 The VEB-Boost R Package

Much of my time has been spent developing and refining the VEB-Boost R package, available via github (`https://github.com/stephenslab/VEB.Boost`, version `0.0.0.9038` at the time of this writing). My efforts fall into two main categories: (i) implementing the VEB-Boost algorithm, and (ii) implementing a default Bayesian weak learner.

3.5.1 Implementing the VEB-Boost Algorithm

In my implementation, I chose to utilize the (block) coordinate ascent fitting procedure outlined in Corollary 3.3.1.1 and Algorithm 2. I did play around with the gradient-based version mentioned in Corollary 3.3.1.2, but the algorithm proved to be impractical. In particular, the step-sizes being taken were near 0, causing a slew of issues. Although one could definitely dig into this further and attempt to solve these issues, I have decided to stick with the coordinate ascent method, as it appears to perform fairly well.

As for the residual variance, I have taken the approach of assuming each observation has the same variance σ^2. And in each iteration of the algorithm, we update our estimate of σ^2. This approach may be too restrictive for real-world data, and I have considered a few other approaches (mentioned in 3.7), but have not yet implemented/tested them.

User-Supplied Inputs

Since such a large part of the appeal to the VEB-Boost algorithm is the modularity that it admits, I attempted to keep the package true to this feature. As a result, all the practitioner must provide is a Bayesian weak learner object. The requirements for such an object are:

1. A function that can solve the weighted Bayesian regression problem from (3.1) for their choice of function h : ! , prior class G, and variational class Q. Concretely, we require a function `fitFunction` that takes in as input:

- `X`: a "predictor" object that the function knows how to use, most likely a design matrix;
- `Y`: a vector in R^n of responses;
- `sigma2`: a vector in R^n_{++} of observation-specific variances;
- `currentFit`: the output from the previous call to `fit_function`, possibly used for warm-starting a part of the fitting procedure.

As output, the function must return a list containing:

- `mu1` : a vector in R^n containing the first posterior moments of each observation, i.e. $\mathsf{E}_q\ [h(\ \ )]$ for the newly fit approximation to the posterior q ;
- `mu2`: a vector in R^n containing the second posterior moments of each observation, i.e. $\mathsf{E}_q\ [h(\ \ )^2]$ for the newly fit approximation to the posterior q ;
- `KL_div`: $D_{KL}\Big(q\ \|\hat{g}\Big)$, i.e. the KL-divergence from the newly fit approximation to the posterior q to the estimated prior $\hat{g}$;
- any other values that must be saved in order to describe the fitted approximation q and predict on new data;

2. A function that can take a predictor object and fitted approximate posterior q , and calculate the first or second posterior moments of the observations. Concretely, we require a function `predFunction` that takes as input:

   - `X_new`: a "predictor" object that the function knows how to use, most likely a design matrix. This may or may not be the same as what was used for training, i.e. this can be "test data";
   - `currentFit`: the output from a call to `fit_function`;
   - `moment`: either 1 for calculating the first posterior moment, of 2 for calculating the second posterior moment.

As output, the function simply returns a vector in R^m, where m is the number of observations that `X_new` contains;

3. A function that can take a predictor object and fitted approximate posterior q , and says whether the fit is close enough to being a constant. Concretely, we require a function `constCheckFunction` that takes as input:

   - `currentFit`: the output from a call to `fit_function`;
   - Anything else required to test if the fit is a constant, e.g. a hyper-parameter.

   As output, the function simply returns `TRUE` or `FALSE`;

4. A predictor object `X` used for training. The only restriction with this is that the supplied `fitFunction` has to know how to use it as an input. In essence, this captures the data-specific components of the function h;

5. A predictor object `X_test` used for testing. This can be `NULL`;

6. A string `growMode` which determines how new nodes are grown in the tree. This can be one of:

   - `"+*"`: This changes each weak learner from to (2) + 1;
   - `"+"`: This changes each weak learner from to + 1;
   - `"*"`: This changes each weak learner from to 2;
   - `"NA"`: This is what's used when the weak learner is not to be grown;

7. A logical `changeToConstant`, which tells the algorithm whether we want to change weak learners that are essentially constant (as determined by the supplied `constCheckFunction`) to be fit as a constant rather than with the supplied `fitFunction`.

With all of these components packaged into a list, the practitioner simply needs to supply it, along with the response `Y`, to the R function `veb_boost`. There are a few other optional inputs, such as the initial tree structure $T(\ )$ of the ensemble learner, but I leave those in the package's documentation for interested parties. You can also tell it which type of data you're dealing with; see Chapter 4 for more details on which types of data are supported. The R-package handles the work that calculates the pseudo-response and pseudo-variance each weak learner needs to fit.

Memory Usage

One computational downside to the current VEB-Boost implementation is that it can be memory-hungry, especially with large sample sizes and strong signals in the data. The package stores the ensemble learner as a tree object, as depicted in Figure 3.1. In addition to the memory needed to store each variational approximation q_l, each node (both internal and leaf) stores the first and second posterior moments for the learner you get if you take that node to be the root of the learner. While this helps reduce computation time, it requires potentially a lot of memory. In particular, if you have L weak learners, then the tree has $2L \quad 1$ total nodes, and thus the storage requirements for storing these intermediate moments is $O(nL)$, where n is the sample size of the training data.

I briefly considered porting the meat of the package over to C++ and re-implementing it without storing these intermediate moments; and then when you need to calculate them, do so in parallel using OpenMP tasks in order to reduce the computation time. However, I have not done so at this time, and do not have any current plans to do so.

3.5.2 Implementing a Default Bayesian Weak Learner

Aside from implementing the VEB-Boost algorithm, I have implemented a default weak learner which seems to work fairly well, accessible in the R function `veb_boost_stumps`. This

weak learner is a slightly modified version of the single effect regression outlined in Section 2.2.1; we implement the single effect regression (see equation (2.4)) where our data is encoded as a catenation of the standard linear basis, as well as a step-function (i.e. "stumps") basis (see Wang et al. [2020] and Tibshirani [2014]).

In short, our weak learner can be described as being a distribution of selecting a single linear term or decision stump, where the stumps are defined by different cut-points in our linear predictors. Using sums/products of stumps has been attempted before in some forms (see, e.g., Friedman and Popescu [2008], Kegl and Busa-Feketa [2009], Nalenz and Villani [2018]). However, to the best of our knowledge, no one has allowed for an arbitrary sequence of additions and multiplications of stumps, and no one has done so using variational inference.

As further justification for this weak learner, it is easy to show that the product of stumps (each containing their own intercept term) results in a balanced decision tree, sometimes referred to as an "oblivious" decision tree (see Figure 3.2 for an example). This is the basic building-block of CatBoost (Prokhorenkova et al. [2018]), one of the three main gradient boosting packages used by machine learning practitioners. Our decision to also include linear terms stems from both ease of implementation, as well as experimental evidence that doing so typically improves performance.

It is worth noting that while the product of two stumps will necessarily be a balanced decision tree of depth two, not every balanced decision tree of depth two can be represented as the product of two stumps. For example, consider the balanced decision tree that describes the XOR function between two rules, R_1 and R_2. Looking at the structure of the balanced tree in Figure 3.2, this would mean that the predictions in each leaf node, going from left to right, would be: 0, 1, 1, 0. This would mean that either $\alpha_1 = 0$ or $\alpha_2 = 0$ for the first leaf to be 0, and either $\alpha_1 + \beta_1 = 0$ or $\alpha_2 + \beta_2 = 0$ for the last leaf to be 0. But this means that at least one of the second and third leaves must also be 0, whereas we wanted them to be 1. Thus, while the XOR function is a balanced decision tree, it cannot be represented as the

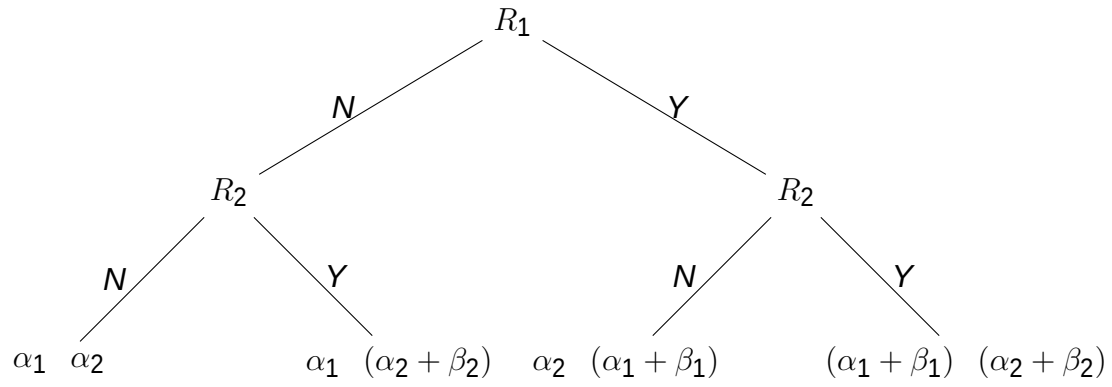

Figure 3.2: **Balanced Decision Tree as the Product of Decision Stumps** The decision tree above represents the product of two decision stumps: $(\alpha_1 + \beta_1 R_1)$ $(\alpha_2 + \beta_2 R_2)$. Here, R_1 and R_2 are the rules of the stump, e.g. $R_1 = \mathsf{I}_{X_{ij} \quad c_j}$. Going left in the decision tree means that the rule is not satisfied (so the indicator function evaluates to 0), and going right means that the rule is satisfied (so the indicator function evaluates to 1).

product of two stumps. However, if we allow additions as well, then we can easily represent the XOR function as $R_1 + R_2 \quad 2R_1R_2 = R_1(1 \quad 2R_2) + R_2$.

Suppose you have a design matrix $\mathbf{X}\ 2\ \mathsf{R}^{n \quad p}$. Focusing on a single variable $\mathbf{X}_{\ j}$, suppose for simplicity that there are n distinct values, and the matrix is sorted on those values, i.e. $X_{1j} < \quad < X_{nj}$. Then one could represent this variable in a basis of decision stumps, resulting in an $n \quad n$ matrix, call it X^j_{stumps}. Such a matrix will look something like

$$\begin{bmatrix} 0 & \dots & 0 & 1 \\ 0 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \dots & 1 & 1 \end{bmatrix}$$

where the first column is $\mathsf{I}_{X_{ij} \quad X_{nj}}$, the second column is $\mathsf{I}_{X_{ij} \quad X_{(n \ 1)j}}$, etc. Thus, each column represents a decision stump that splits on variable j at a particular cut-point. In actuality, since the matrix likely isn't sorted by variable j, you must also multiply this matrix with a permutation matrix, but such details obfuscate the high-level picture. We can

do this for all variables j, which lets us transform $\mathbf{X} \; ! \; \mathbf{X}_{stumps} = [\mathbf{X}^1_{stumps} j \quad j \mathbf{X}^p_{stumps}]$. Our matrix X_{stumps} is a binary matrix whose columns are indicator functions (i.e. decision stumps) splitting on a certain variable at a given cut-point.

As an aside, matrix-vector multiplications with a component on the stumps matrix (i.e. $\mathbf{X}^j_{stumps}$) can be performed in $O(n)$ time. Looking at the above matrix, it is clear that the product $\mathbf{X}^j_{stumps}\mathbf{b}$ is related to the cumulative sum of the reverse of $\mathbf{b}$, which can be computed in $O(n)$ time. And similarly, we do not need to actually store $\mathbf{X}^j_{stumps}$ as an $n \quad n$ matrix. This is outlined in Tibshirani [2014]. In practice, we also don't have to take every possible cut-point, and the default is to use far fewer than n ($d\min f n/5, \max f 100, p\overline{n}gge$). By not using all possible cut-points, we are able to save a bit on both computational cost and memory requirements, as well as ensure that both leaves in each of our stumps has a minimum number of observations. The curious reader can refer to `VEB.Boost` R package for more details.

Once we have our stumps matrix, the final design matrix we use is

$$\mathbf{X}_{combined} = [\mathbf{X} j \mathbf{X}_{stumps}],$$

i.e. we combine both the original design matrix (what I refer to as "linear terms") and the stumps matrix. And with this final design matrix, the weak learner is just the single effect regression from Section 2.2.1 using the design matrix $\mathbf{X}_{combined}$. The time complexity of fitting this SER model is $O(np)$.

We also specify the range of the log-prior variance $\log \sigma_0^2$ from the SER model using the `max_log_prior_var` argument, which has a default value of 0. When we perform the empirical Bayes step to estimate the prior (i.e. estimate σ_0^2), we perform the maximization only over $\log \sigma_0^2 \; 2 \; [\; 15, \; \texttt{max\_log\_prior\_var}]$. Restricting how large σ_0^2 can be can be viewed as an additional form of shrinkage beyond the Bayesian computation, and can keep each weak learner from being too strong. The prior probability vector in the SER is set

up such that there is a specified probability (called `lin_prior_prob` in the R function, which is a parameter the practitioner provides) that the non-zero effect is a linear term. And then among the linear terms, each term is equally likely, and among the stumps terms, each term is equally likely.

As an illustrative example, say there are two linear variables; from the first variable we generate two stumps terms and from the second variable we generate three stumps terms; in practice, most variables will have many more than just a few stumps, and will only have a different number of stumps if they have fewer unique values than cut-points. Then if we set `lin_prior_prob` to 0.5 (which is the default in the R package), then the prior probability vector will be

$$= (0.25\ 0.25\ \boldsymbol{|}\ 0.1\ 0.1\ \boldsymbol{|}\ 0.1\ 0.1\ 0.1)^T.$$

The vertical lines are just there to highlight that the first two probabilities are for the linear terms, the next two are for the stumps terms from the first variable, and the final three are for the stumps terms from the second variable.

In words, this weak learner places all of its mass on being either a linear function of a single variable or a decision stump. The justification for the decision stump portion is that it is a very simple non-linear function that can be combined to effectively approximate any function. As mentioned earlier, it is easy to see that the product of decision stumps yields a balanced/oblivious decision tree, the building block of CatBoost. The justification for the linear term is that it is still fairly simple and there's very little additional cost to include it. And as we will see in the simulation examples, having the linear term allows the VEB-Boost method with this weak learner to outperform other non-linear regression methods in a setting with a true linear relationship between the predictors and the response.

I have implemented this weak learner in C++, leveraging the OpenMP framework for its multi-platform shared memory parallel computation. In particular, I have parallelized the

matrix computations needed to solve the SER. For an arbitrary matrix partitioned as

$$\mathbf{X} = [\mathbf{X}_1\, j \quad j\, \mathbf{X}_k] \; 2 \; \mathsf{R}^{n \;\; p},$$

we can calculate

$$\mathbf{Xb} = [\mathbf{X}_1\, j \quad j\, \mathbf{X}_k] \begin{pmatrix} \mathbf{b}_1 \\ \hline \vdots \\ \hline \mathbf{b}_k \end{pmatrix} = \mathbf{X}_1\mathbf{b}_1 + \quad + \mathbf{X}_k\mathbf{b}_k$$

and

$$\mathbf{X}^T\mathbf{y} = [\mathbf{X}_1\, j \quad j\, \mathbf{X}_k]^T\mathbf{y} = \begin{pmatrix} \mathbf{X}_1^T\mathbf{y} \\ \hline \vdots \\ \hline \mathbf{X}_k^T\mathbf{y} \end{pmatrix}$$

for appropriately partitioned vector $\mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \hline \vdots \\ \hline \mathbf{b}_k \end{pmatrix} \; 2 \; \mathsf{R}^p$ and $\mathbf{y} \; 2 \; \mathsf{R}^n$.
We can thus calculate the components of these matrix products in parallel (split up by the matrix partitions) and then combine the results. This allows us to leverage the structure of the stumps matrices.

Another benefit of this weak learner is that we can provide measure of feature importance quite easily, since the posterior distribution includes a posterior probability that the effect is coming from a particular column in the design matrix (call this vector). The R function `stumpsVariableImportance` can take in a fitted VEB-Boost ensemble learner that uses the above weak learner and can provide some measures of variable importance. For each weak learner, we can take and add together the components that correspond to a particular variable (i.e. add together the posterior probability of the linear term and the stumps terms for each of the original variables). We now have a new probability vector $\begin{smallmatrix} 0 \\ l \end{smallmatrix} \; 2 \; \Delta^{p \;\; 1}$ for

each of our weak learner $_l$, $l = 1, \ldots, L$. Using these vectors, we can calculate feature importance measures as:

- Posterior inclusion probabilities (PIPs): This is the posterior probability that a variable has a non-zero effect anywhere, i.e.

$$1 \quad P(\text{variable } j \text{ is not an effect variable in any weak learners}) = 1 \quad \prod_{l=1}^{L}(1 \quad \alpha_{lj}^{\emptyset});$$

- Sum of probabilities: We can simply add up the vectors and try to get a sense of overall importance, i.e.

$$\sum_{l=1}^{L} \alpha_{lj}^{\emptyset}.$$

However, as written, there is one obvious improvement that can be made to the above definitions; not all weak learners are equally important. A quick and easy (and readily available) way to approximate the importance of a weak learner is to look at $D_{KL}(q_l \, k\hat{g}_l)$; a large value indicates that the weak learner is trying to fit more signal, and a small value indicates that there isn't much signal for the learner to fit. Let the KL-divergence for the l-th weak learner be KL_l. Then we can change the above formulae to

$$1 \quad \prod_{l=1}^{L}(1 \quad \alpha_{lj}^{\emptyset})^{KL_l}$$

and

$$\sum_{l=1}^{L} \alpha_{lj}^{\emptyset} \quad KL_l.$$

As a simple justification for including the KL-divergence in the exponent instead of as a multiplicative factor in the PIP case, a KL-divergence of 0 for a weak learner l would results in an unchanged PIP when exponentiating but a PIP of 0 when multiplying.

3.6 Examples

In this section, I present results from both a simulation study, as well as from a real dataset benchmarking analysis. All examples were run on the Midway2 high-performance computing platform at the University of Chicago, which uses Intel E5-2680 v4 CPUs. I set up each node with 8 CPUs and 48 GB of RAM. Version `0.0.0.9038` of `VEB.Boost` was used.

3.6.1 Simulation Study

In this subsection, I present the results from a small simulation study. I generate independent observations according to the following model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i.$$

Here, $f(\mathbf{x})$ is one of four functions:

1. Friedman's 5-dimensional test function

$$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 \quad 0.5)^2 + 10x_4 + 5x_5;$$

2. Max test function

$$f(\mathbf{x}) = \max\{x_1, x_2, x_3\};$$

3. Linear test function

$$f(\mathbf{x}) = \sum_{j=1}^{10} 4\left(\frac{j \quad 1}{9} \quad \frac{1}{2}\right); x_j$$

4. Null test function

$$f(\mathbf{x}) = 0.$$

And $\epsilon_i \quad N(0, \sigma^2)$, where σ^2 is set such that the proportional of variance explained

(PVE) is either 0.1 or 0.5. PVE is defined as

$$PVE = \frac{Var\Big(f(\mathbf{x})\Big)}{Var\Big(f(\mathbf{x})\Big) + \sigma^2}.$$

In the case of the null test function, since PVE is 0 regardless of the noise level, I use the noise level $\sigma^2 = 9$ instead of PVE $= 0.1$, and $\sigma^2 = 1$ instead of PVE $= 0.5$.

To generate the design matrix $\mathbf{X}$ 2 $\mathsf{R}^{n \ p}$, I simulated $x_{ij} \overset{iid}{} Unif(0,1)$ in the case of Friedman's function, and $x_{ij} \overset{iid}{} \mathsf{N}(0,1)$ otherwise.

The machine learning methods I considered in this simulation study were:

- **"BART"**: Bayesian Additive Regression Trees (Chipman et al. [2010])

  I used the R package `dbarts`, with a burn-in of 1000 samples and 5000 posterior samples. In the case where $n = p = 1000$, the R function gives an error unless I supply it with a residual standard deviation. In this case, I supplied the function with the true residual standard deviation. All other parameters were left at their default values;

- **"XBART"**: Accelerated Bayesian Additive Regression Trees (He et al. [2019])

  I used the R package `XBART` (available at `https://github.com/JingyuHe/XBART`). Following the recommendations in the paper and github code, I set `num_trees` to be $d0.25\log(n)^{\log\log n}e$, `tau` to 0.3Var(Y) / `num_trees`, `num_sweeps` to 40, `burnin` to 15, `alpha` to 0.95, `beta` to 1.25, and `num_cutpoints` to $\max f100, \frac{p}{n}g$;

- **"xgBoostCV"**: XGBoost with cross-validation (Chen and Guestrin [2016])

  I used the R package `xgboost`. I performed 5-fold cross-validation over a relatively small parameter grid: (`max_depth`, `eta`, `min_child_weight`) 2 $f2,4,8g$ $f0.1,0.3g$ $f1,5,10g$. I also set `nrounds` to 10000, `early_stopping_rounds` to 50, `gamma` to 0.1, `colsample_bytree` to 0.8, `subsample` to 0.8, and `nthread` to 8;

- **"VEBBoost"**: VEB-Boost with default settings

I simply set `nthreads` to 8;

- **"VEBBoostBig"**: VEB-Boost with a larger initial ensemble learner
  I set `nthreads` to 8, `k` to 16, and `d` to `sample(c(rep(1, 6), rep(2, 4), rep(4, 4), rep(8, 2)))`, i.e. I started the ensemble learner as the product of 8 learners in a randomized order, of which 6 were a single weak learner, 4 were the product of 2 weak learners, 4 were the product of 4 weak learners, and 2 were the product of 8 weak learners.

I ran 5 replicates each of all combinations of $(n, p, PVE, test\_function)$ $\in$ $\{1000, 10000, 100000\} \times \{10, 100, 1000\} \times \{0.1, 0.5\} \times \{friedman, max, linear, null\}$. One thing to note is that for $n = 100000, p = 1000$, XGBoost's built-in cross-validation function, `xgb.cv`, ran out of RAM when run with multiple threads and ran out of time when run with a single thread, and so there are no results for this combination of inputs. This could introduce some bias into the results and conclusions.

To evaluate the performance of the methods, I used the relative root-mean-squared-error (RRMSE). The root-mean-squared-error (RMSE) is defined as

$$\sqrt{\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \left(\hat{f}(\mathbf{x}_i^{new}) - f(\mathbf{x}_i^{new}) \right)^2}, \tag{3.13}$$

where we are summing over unseen testing observations that were generated in the same way as the training observations, $\hat{f}(\,)$ is the fitted function returned by the algorithm, and $f(\,)$ is the true mean response function. For the Bayesian methods, the posterior mean is used as the estimated function value. Note that we are comparing with the true mean of the new observations and are not taking into account any noise; this is to highlight the methods' abilities to recover the true underlying mean.

$$\frac{RMSE_{\mathsf{A}}}{\min_{\mathsf{A}'} RMSE_{\mathsf{A}'}}, \tag{3.14}$$

where the minimum in the denominator is taken over all algorithms tested for that particular simulation dataset.

I also use the running time of the algorithms to compare how fast they are. When the presented times are relative, they are relative to the fastest algorithm for that particular dataset.

I use two different types of plots to present the results. The first type is a boxplot; for a particular combination of $(n, p, PVE, test\_function)$, we make a boxplot from the 5 runs where each method gets its own box. We show boxplots for the Friedman test function as an illustrative example; boxplots for the other test functions are provided in Section A.2.1, Figures A.1 through A.6. The second type of plot shown is a profile plot (Dolan and Moré [2002]); this is just a plot of the empirical CDF of the metric for the given algorithm.

Results are presented below in Figures 3.3 through 3.6. In general, VEB-Boost appears to be quite competitive in these simulations, both in terms of performance and running times. We are also able to see how VEB-Boost's running time scales with the strength of the signal in the dataset.

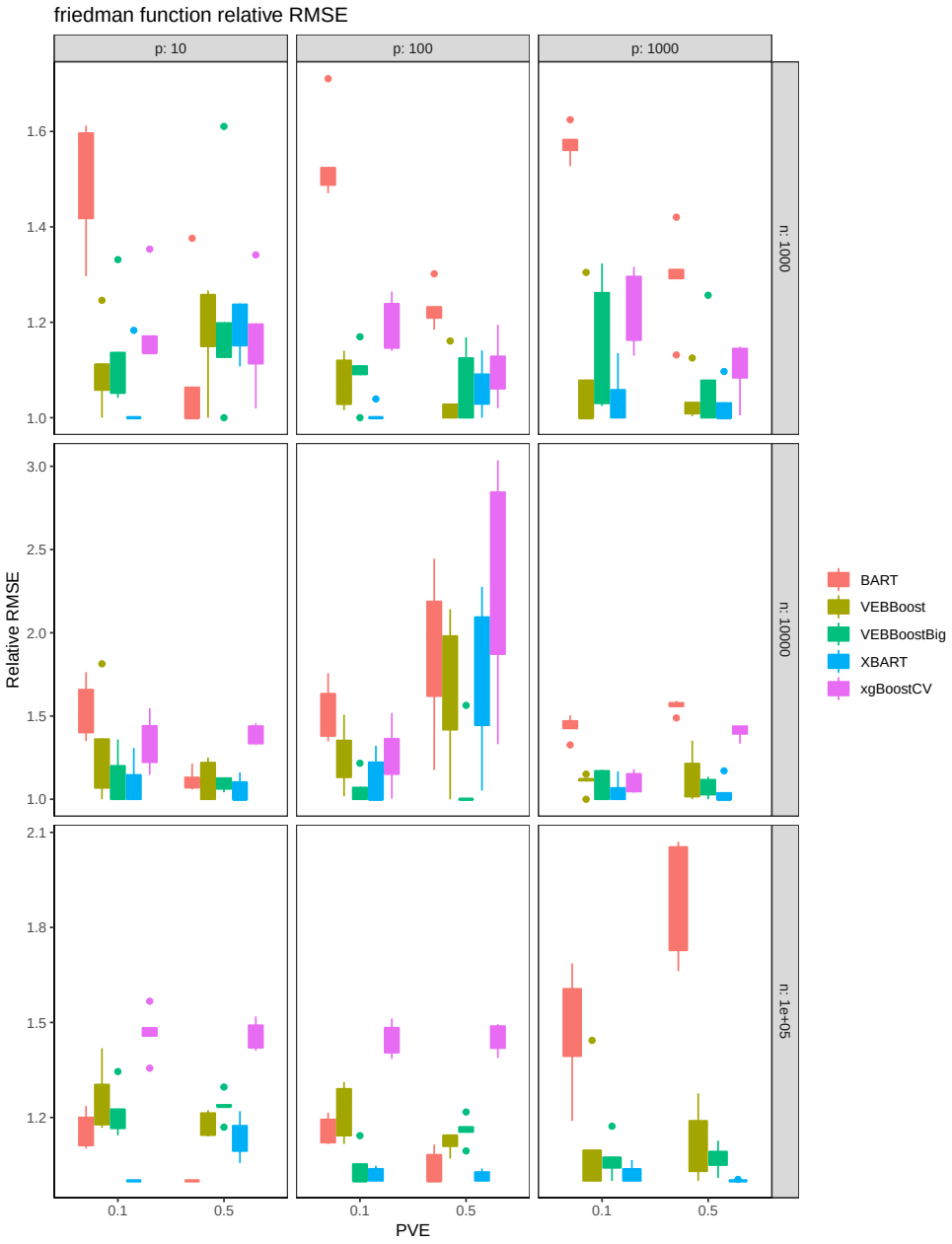

Figure 3.3: **Friedman's Function Relative RMSE** This plot shows the relative RMSE of the methods tested using Friedman test function. We can see that, on the whole, the VEB-Boost methods frequently outperform both BART and cross-validated XGBoost (and when they don't, they aren't much worse); only XBART appears to be a competitor in this simulation.

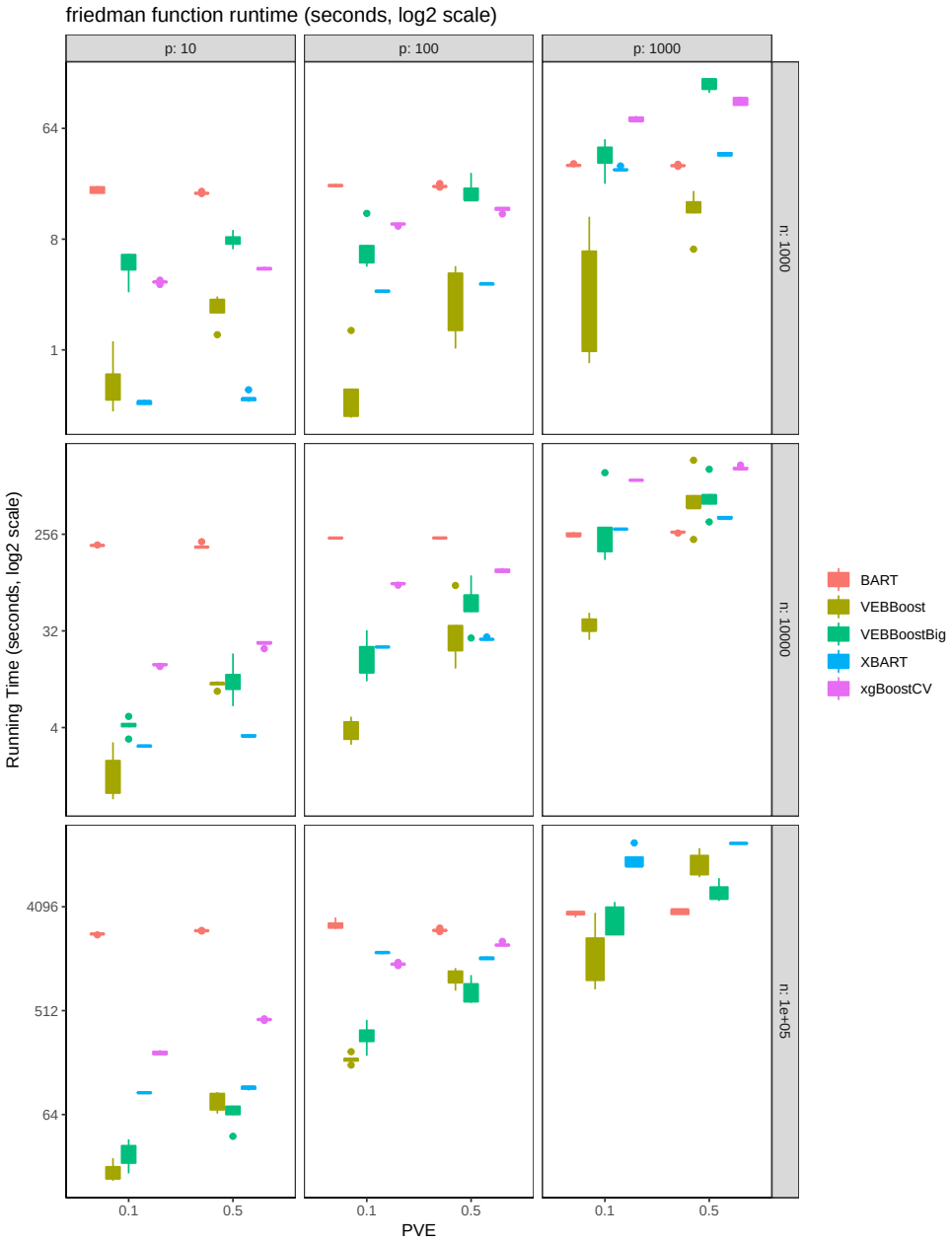

Figure 3.4: **Friedman's Function Running Time in Seconds,** $\log_2$ **scale** This plot shows the running time of the methods tested using Friedman's test function. We see that VEB-Boost is often the fastest method, and starting with a larger learner usually increases the overall running time. We also see that VEB-Boost is relatively faster when the PVE is 0.1 vs. when the PVE is 0.5. This makes sense, since BART and XBART are set *a priori* to run a certain number of iterations, whereas VEB-Boost keeps running and growing until it can't find any more signal. Thus, it will terminate faster in the higher noise setting, which is what we see above. This is also why there is a larger spread in the observed running times for VEB-Boost as compared with the others.

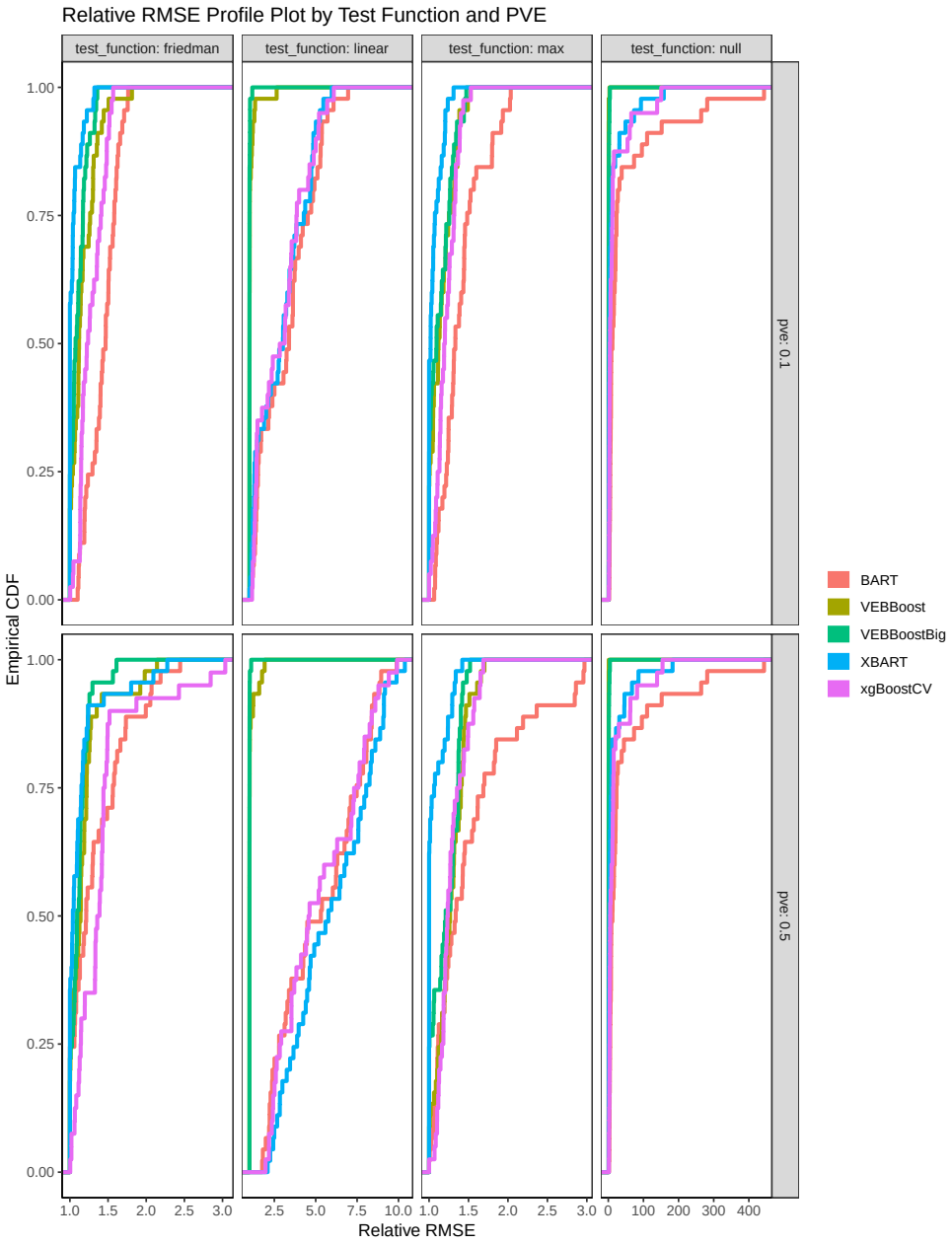

Figure 3.5: **Relative RMSE Profile Plot** This plot shows the empirical CDFs of the relative RMSE combining all values of n and p, but broken out by test function and PVE. We can see that in the linear and null test cases, VEB-Boost dominate the other methods. And in the Friedman and max test cases, XBART appears to have a slight edge over VEB-Boost, with the exception of the Friedman test function in the strong signal setting. We also see that, on the whole, VEB-Boost starting with a larger learner performs slightly better than starting with a single weak learner.

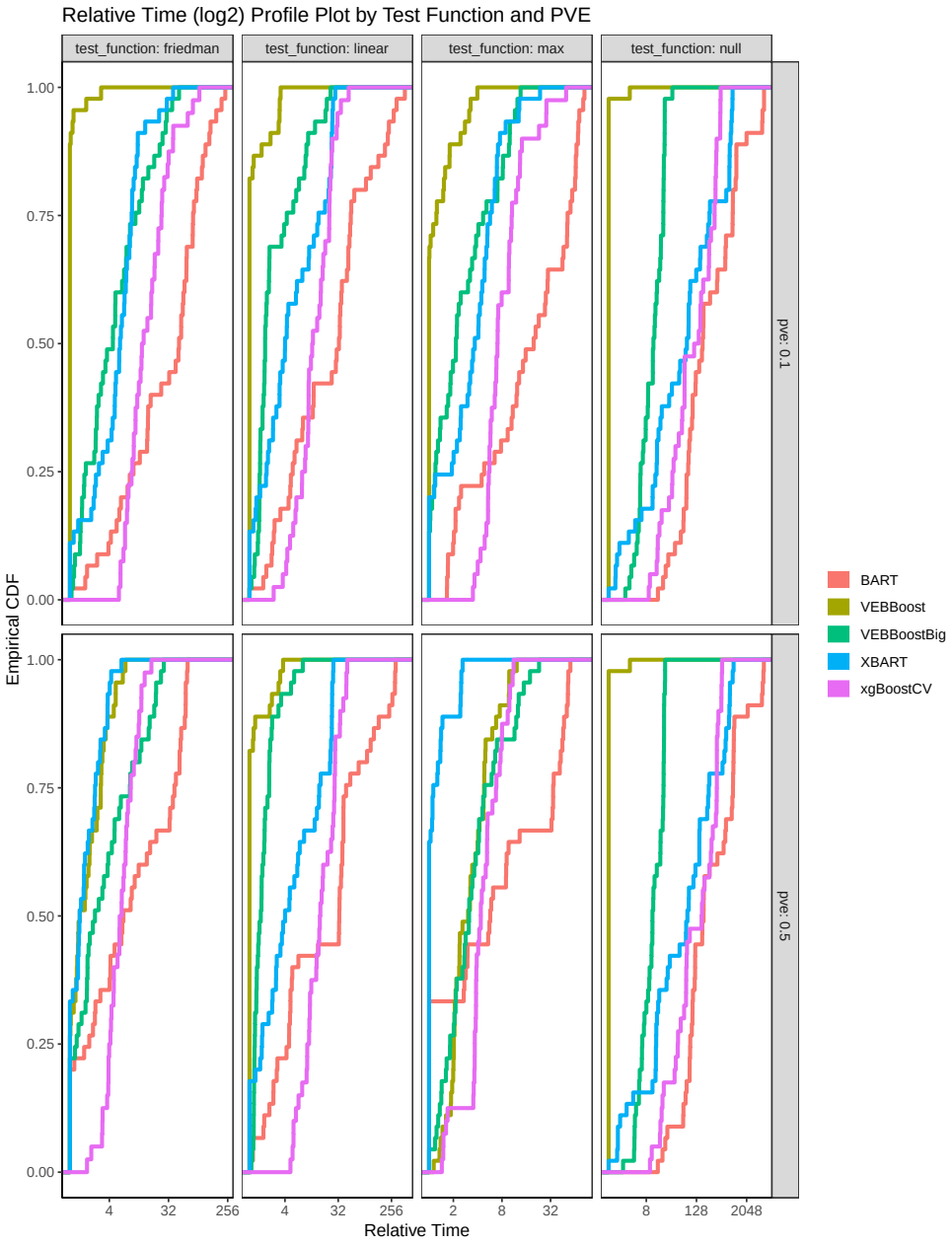

Figure 3.6: **Relative Time Profile Plot,** $\log_2$ **scale** This plot shows the empirical CDFs of the relative running times, on a $\log_2$ scale, combining all values of n and p, but broken out by test function and PVE. We can see that VEB-Boost wins for all cases except for the Friedman and max test functions in the strong signal setting, where it is roughly tied and loses to XBART, respectively. We also see that, on the whole, VEB-Boost starting with a larger learner is slower than starting with a single weak learner.

3.6.2 Real Dataset Comparison

To test out these methods on real data, I have used the AutoML regression benchmark datasets available through OpenML (`https://openml.org/search?type=benchmark&sort=tasks_included&study_type=task&id=269`). A total of 25 datasets from this list were used in the benchmarks; a full list can be found in the Appendix in Table A.1. For each dataset, I also added either 0, 100, or 1000 synthetic null variables simulated as iid $N(0, 1)$. This was to test out the methods in higher-dimensional/sparse settings as well, since some datasets had $n \quad p$. As before, some algorithms could not be run with additional variables due to the data being too large, so only results that include all algorithms are included.

Since the datasets did not come with pre-defined test/train splits, I opted to split each dataset into 5 folds and calculate the RMSE obtained by having each fold serve as the test set. That is, I partitioned each dataset into 5 disjoint folds, trained each model leaving a fold out, and then getting the RMSE when evaluated using that fold.

Results are presented below in Figures 3.7 through 3.10. VEB-Boost's performance is not quite as good as it appeared in the simulations. I hypothesize a few possible explanations in the discussion section to this chapter in Section 3.7.

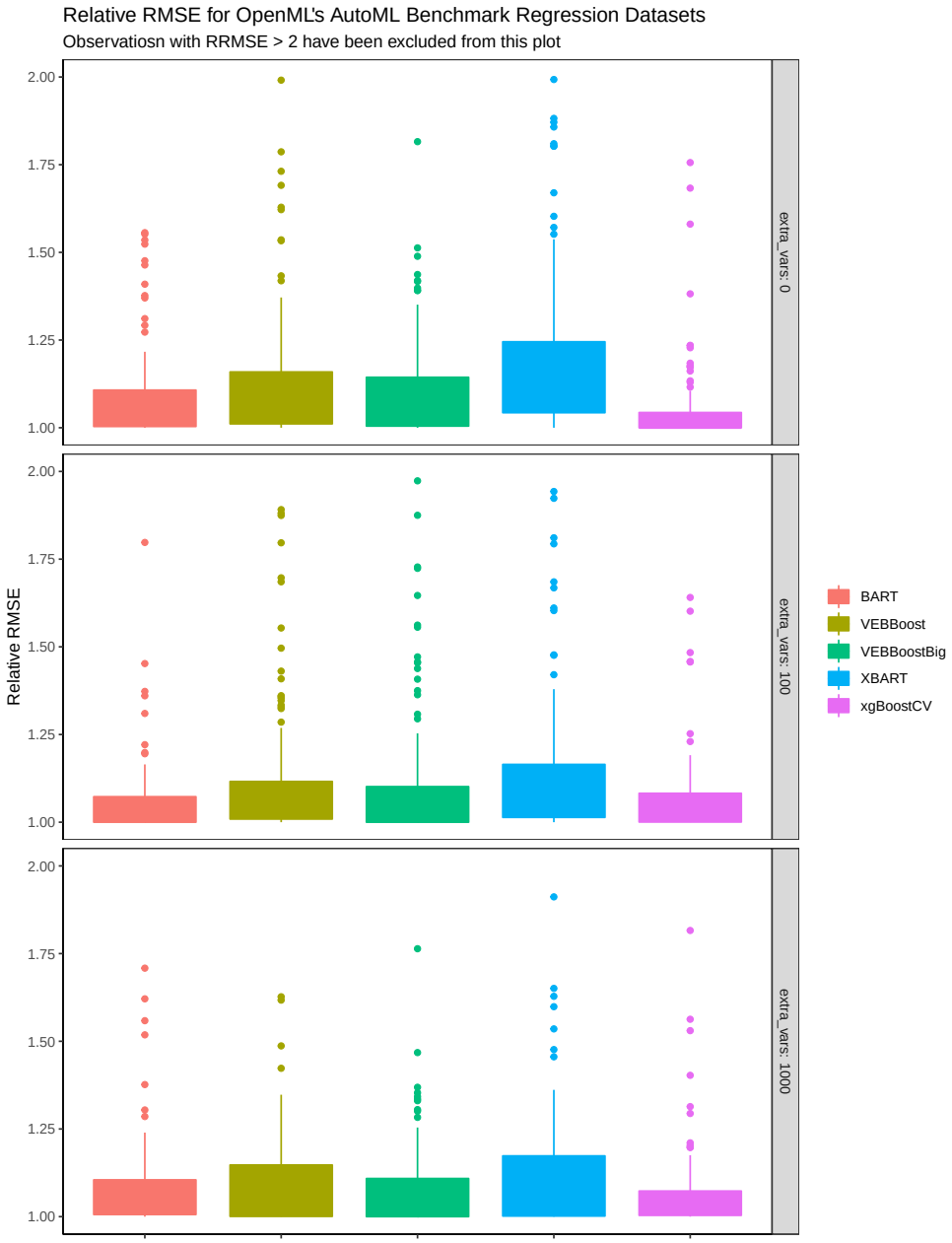

Figure 3.7: **OpenML AutoML Regression Benchmarks Relative RMSE** This plot shows the relative RMSEs for each method among all folds of all datasets, broken out by how many extra null variables were added. Observations with a RRMSE > 2 have been excluded for visual purposes. We can see that cross-validated XGBoost appears to be the best, followed closely by BART and VEB-Boost with a larger starging learner. We also see XGBoost's advantage start to disappear in the bottom row, where we've added 1000 null variables to each dataset.

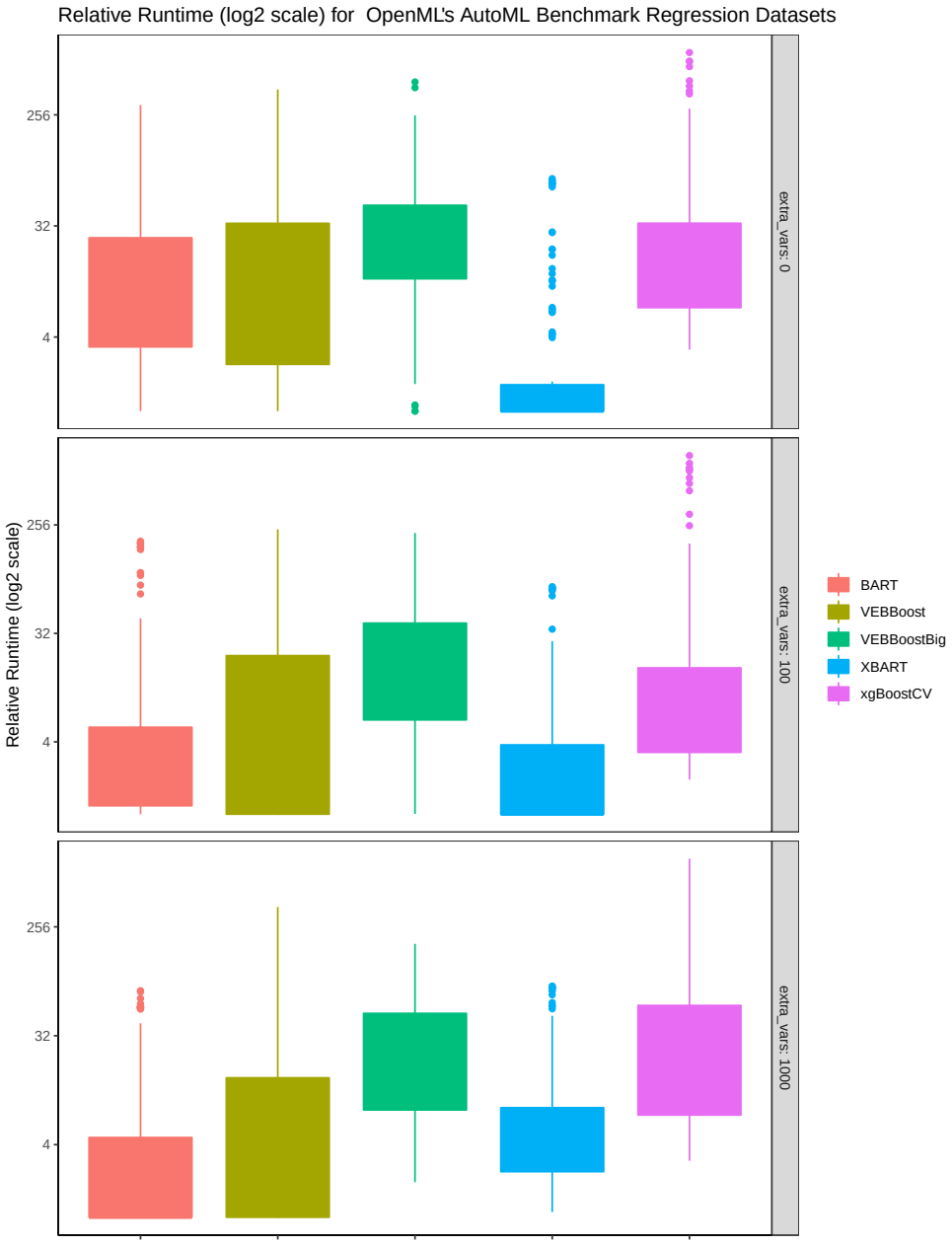

Figure 3.8: **OpenML AutoML Regression Benchmarks Relative Time,** $\log_2$ **scale** This plot shows the relative running times on a $\log_2$ scale for each method among all folds of all datasets, broken out by how many extra null variables were added. We can see that XBART is typically among the fastest, and that VEB-Boost is more competitive in the cases where we add more null variables.

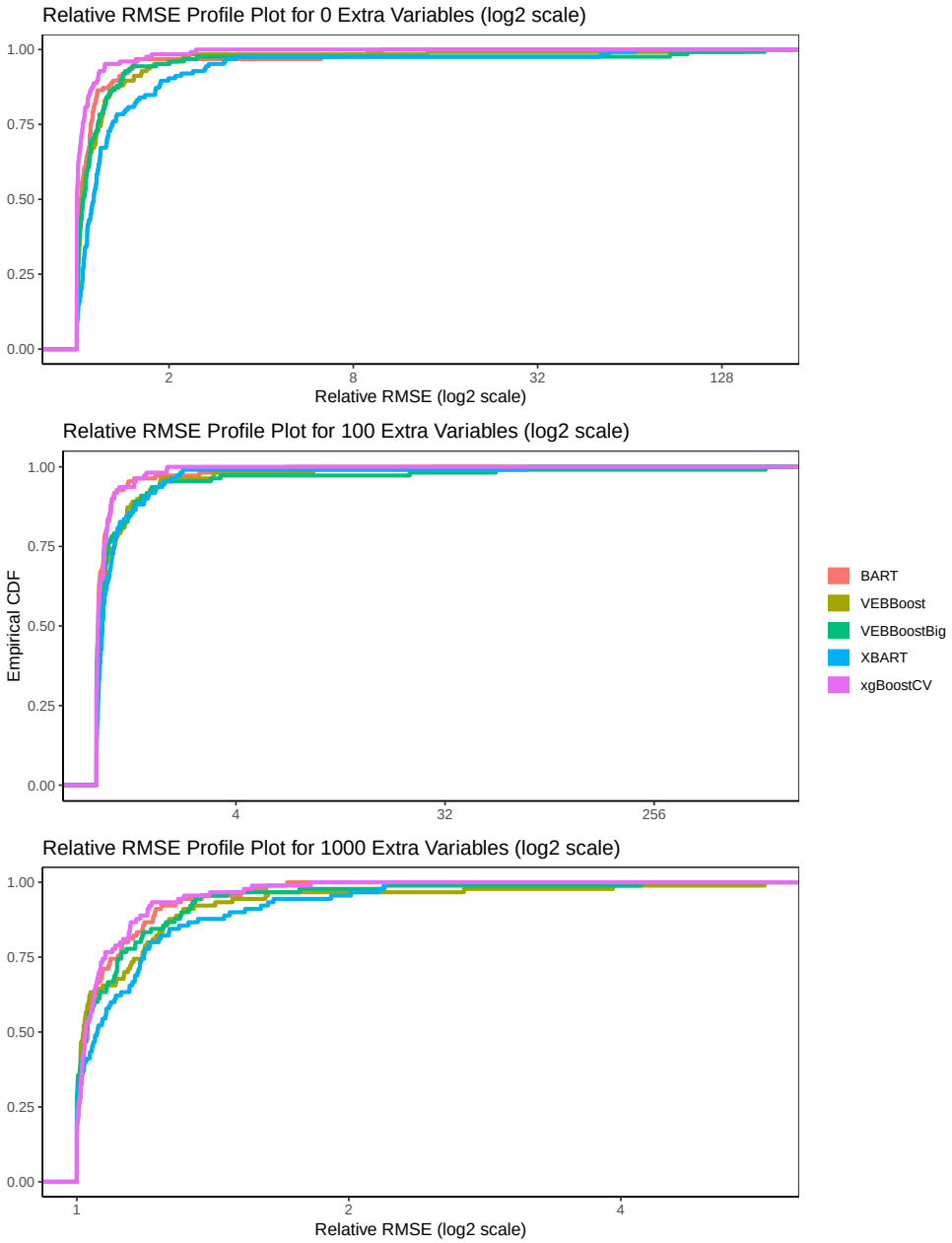

Figure 3.9: **OpenML AutoML Regression Benchmarks Relative RMSE Profile Plot** This plot shows the empirical CDFs of the relative RMSE on a $\log_2$ scale for each method among all folds of all datasets, broken out by how many extra null variables were added. Agreeing with Figure 3.7, we see that XGBoost is the winner is most cases. But as we add more null variables, VEB-Boost starts to become more competitive. We also see that there were a few cases of ***extremely*** poor relative performance; I briefly touch on this in Section 3.7.

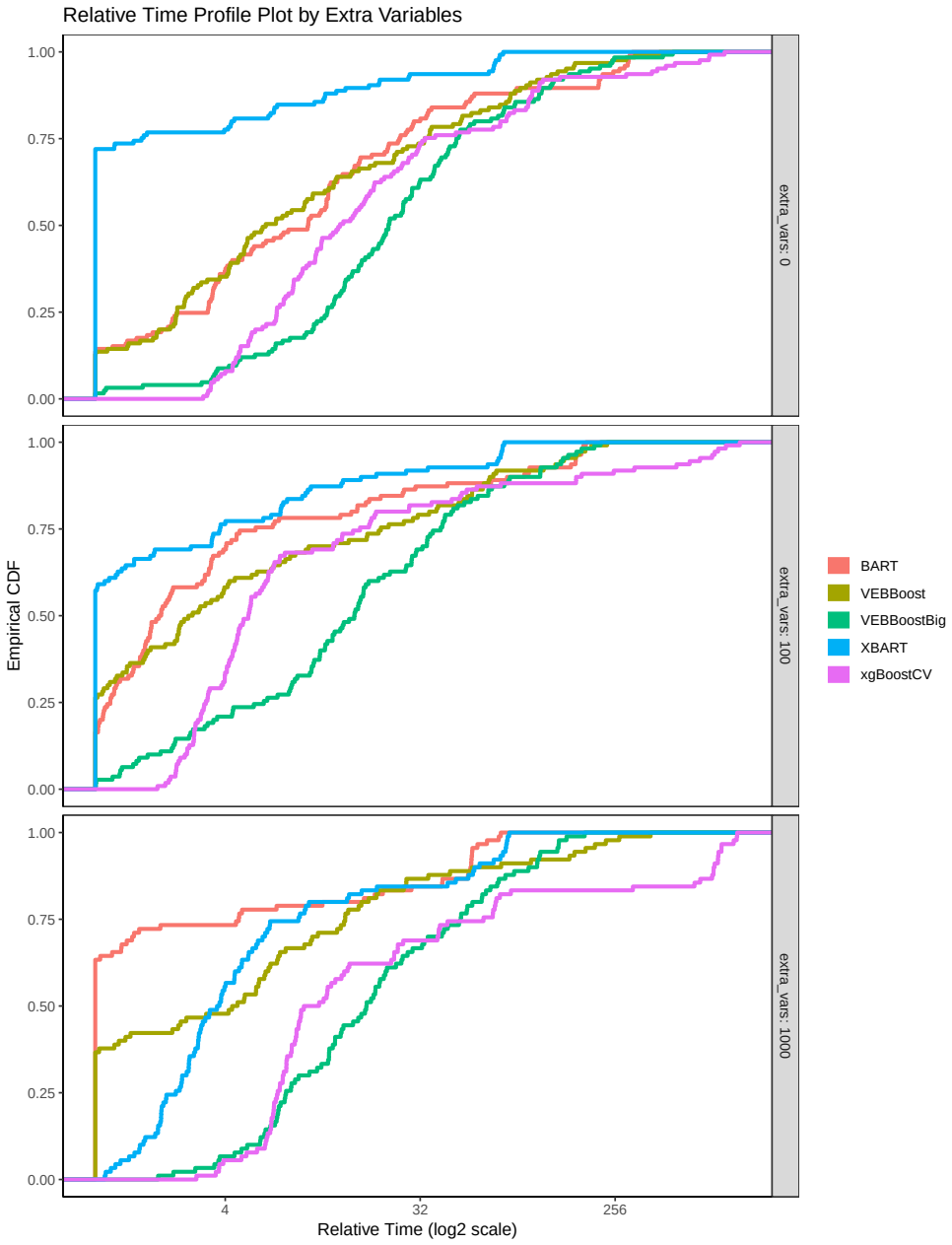

Figure 3.10: **OpenML AutoML Regression Benchmarks Relative Time Profile Plot** This plot shows the empirical CDFs of the relative running times on a $\log_2$ scale for each method among all folds of all datasets, broken out by how many extra null variables were added. We can see that VEB-Boost is in the middle of the pack, and starts to over-perform in the case with 1000 additional null variables. We also see that XGBoost can have some very long relative run-times with additional null variables added.

3.7 Discussion

In this chapter, we have laid out the core of this dissertation: the VEB-Boost model. We have built it up as an extension from the variational additive model from chapter 2 by allowing for the multiplication of weak learners. We have shown that performing (block) coordinate ascent updates can be achieved by solving the simpler weighted Bayesian regression subproblem from Section 3.2. We also briefly went over the R package that implements the VEB-Boost framework, as well as a few useful weak learners; in particular, we describe the SER learner that uses both linear and stumps terms. Finally, we concluded with a simulation study and some real-world examples.

Looking at the simulations, we see that there truly is no "free lunch," i.e. there is never going to be a single algorithm that consistently outperforms the others. In this idealized scenario, BART appears to be the loser. But this could just be due to MCMC convergence issues, which are hard to detect/diagnose, or because vanilla BART does not have any sense of sparsity in the prior. On the flip-side, we see that VEB-Boost and XBART are usually at the top, with cross-validated XGBoost not far behind. We also see that in some settings (e.g. the linear and null test functions), VEB-Boost's performance greatly exceeds that of the other methods tested. This could indicate that VEB-Boost is more conservative than the other methods, in the sense that it is more than happy to fit the data with a simpler fit (e.g. linear) or no fit at all (e.g. null).

In terms of run-time, we see that VEB-Boost compares quite favorably to the others. We also see that VEB-Boost is able to run faster in the higher noise settings, since the empirical Bayes nature allows it to scale with the signal strength. But due to the fact that VEB-Boost is an iterative algorithm that doesn't run for a fixed number of iterations, the run-times are much more variable, and can occasionally go on for quite some time. But since VEB-Boost uses CAVI (i.e. solves an optimization problem using coordinate ascent), then conceivably the algorithm could be terminated at any time to give an approximate solution.

Turning to the real-world datasets, we see that cross-validated XGBoost and BART make their way up to the top in terms of performance. Meanwhile, XBART appears to struggle in these examples. And VEB-Boost is slightly worse than BART and XGBoost, on average. I have a few theories as to why VEB-Boost's performance diminished compared to the simulation study:

- Many of these datasets contains categorical variables, for which I used one-hot encoding. It is possible that VEB-Boost suffers when there are categorical variables with many levels;
- VEB-Boost makes a somewhat-restrictive homoskedastic Gaussian assumption for the response, whereas XGBoost doesn't necessarily make/rely on such an assumption. If the data deviated from this assumption, any likelihood-based approach could certainly have issues;
- Some datasets had extreme outliers (some in the predictor space and some in the response) that I did not remove. It is possible that VEB-Boost is more sensitive to these extreme outliers than some of the other methods;
- There could be some bias in the datasets. In particular, for the larger datasets, some algorithms couldn't be run with additional variables (or even with no additional variables) with the computational resources provided, and thus these datasets are not included in the comparison. It's possible that VEB-Boost would have outperformed the other methods on the datasets in these settings, but we weren't able to include it in the comparison. For example, the implementation of BART I was using couldn't handle sparse matrices, which limited the size of the data it could handle.

There are a number of future directions that I would be interested in exploring. First, we only focused on VEB-Boost with our SER weak learner; I would be interested to test out other types of weak learners. In particular, I'm curious about:

- higher-order trend-filtering (Tibshirani [2014]): although easy to implement in homoskedastic cases, some computational difficulties arise in the heteroskedastic setting;
- the weak learner we used earlier for the examples with the variational additive model in Section 2.4: as a reminder, in this weak learner we pick a single variable and then fit a non-linear function in that variable alone. This is easy to implement using the existing software, but would benefit greatly from parallelization;
- a single Bayesian tree fit using MCMC: as I mentioned earlier, it would be interesting to compare a Bayesian sum-of-trees model made in this way with the other methods, BART in particular.

Second, I would like to explore alternative ways to estimate the residual variance. There are two different weak-points that I would like to address:

1. As hypothesized, it is possible that if the data deviates from the constant variance assumption, then VEB-Boost runs into problems. One idea I had to fix this was to move away from relying on this assumption, and instead try to fit multiple variances. There are a few obvious ways one could go about implementing this:
   (a) Place an inverse gamma prior of the residual variances for each observation, i.e. $\sigma_i^2 \overset{iid}{} IG(\alpha, \beta)$. We can then perform an empirical Bayes step to estimate α and β, and include the posterior distributions for σ_i^2 in the variational approximation. The inverse gamma prior is chosen to maintain conjugacy;
   (b) Tell the algorithm that there are k different possible residual variances that each observation can have, estimate what these variances are, and then assign them to observations. I believe that this could be achieved with some 1-d clustering methods. While NP-hard to solve exactly, it should easy to at least increase the ELBO each iteration;

2. In some cases, VEB-Boost terminates before the tree gains enough structure to see the signal. For example, in the noiseless case where $y_i \quad x_{i1} \quad x_{i2}$ (like a continuous version of XOR), VEB-Boost terminates at the null fit. This is because no variable has a marginal association with the response. However, if I force the residual variance to be extremely small, then the algorithm keeps running and is able to eventually learn the correct function. One idea I had was to start with a very small residual variance and update it more slowly (e.g. include a momentum term in the coordinate ascent step for the residual variance). This potentially runs the risk of exposing ourselves to over-fitting, since the residual variance term controls how much we over/under fit, but I think this is worth exploring.

And third, I would be interested in trying to optimize the `lin_prior_prob` parameter from the SER we used. Currently, this is a parameter that the user sets (defaults to 0.5). In some cases, I have observed that the fit can be drastically better (or worse) if this is set closer to 0 or 1. It would be relatively straight-forward to add this variable into the empirical Bayes step that estimates the prior distribution when fitting each weak learner.

CHAPTER 4

MODULARITY WITH NON-GAUSSIAN RESPONSE DATA AND WEIGHTED OBSERVATIONS

4.1 Introduction

The goal of variational inference – finding an approximation to a posterior distribution – is usually framed as an optimization problem in which we maximize a lower-bound to the marginal log-likelihood of the data called the evidence lower-bound (ELBO). Sometimes, maximizing this lower-bound proves to be too difficult, so instead the practitioner finds a lower-bound of the ELBO (referred to as a variational lower-bound) and maximizes that instead. Often times, this variational lower-bound is chosen so that the optimization is simple (e.g. the "likelihoods" become conjugate). For a brief description, see Blei et al. [2017].

In particular, if we can find a quadratic lower-bound for the log-likelihood of a given distribution, then we can utilize our weighted Bayesian regression solvers from Section 3.2. In other words, we can attempt to approximate data from a non-Gaussian distribution as coming from a Gaussian distribution, perform an iteration of the VEB-Boost model, and then update our Gaussian approximations. This is analogous to how weighted least squares is used in the context of the iteratively reweighted least squares algorithm for solving a generalized linear model (Nelder and Wedderburn [1972], McCullagh and Nelder [1983]).

Using a lower-bound to the log-likelihood has some desirable properties. Perhaps most desirably, it is guaranteed that each iteration will be non-decreasing in a lower-bound to an objective function (i.e. the ELBO), and thus the algorithm is guaranteed to terminate. However, one could choose to use other quadratic approximations to the log-likelihood, e.g. using a Taylor series expansion. This could have some benefits too, such as being able to include distributions with a log-likelihood that cannot be bounded by a quadratic (e.g.

Poisson with log-link), or having a better approximation in areas of high probability.

While using quadratic bounds has the appeal of being able to use Gaussian solvers, there are certainly some downsides. In particular, other bounds can often be tighter, and thus have have better performance (Minka [2001], Knowles and Minka [2011], Marlin et al. [2011]). While exploring alternate bounds/approaches could definitely be interesting in this setting, I believe that the modularity afforded by using quadratic approximations is too attractive a feature to sacrifice, at least at this stage in VEB-Boost's development.

On the topic of modularity, let $\mathbf{T} \quad T(h_1(\ _1), \ldots, h_L(\ _L))$ be our VEB-Boost learner, be a vector of variational parameters, and $l(\ _1, \ldots, \ _L; \mathbf{y}, h_1, \ldots, h_L, T)$ be the log-likelihood of our data. Then we want to find a matrix $\mathbf{A}(\mathbf{y}, \ ) \ 2 \ \mathbf{S}^n_{++}$, a vector $\mathbf{b}(\mathbf{y}, \ ) \ 2 \ \mathsf{R}^n$, and a constant $c(\mathbf{y}, \ ) \ 2 \ \mathsf{R}$ such that we have the following bound indexed by :

$$l(\ _1, \ldots, \ _L; \mathbf{y}, h_1, \ldots, h_L, T) \qquad \frac{1}{2}\mathbf{T}^T\mathbf{A}(\mathbf{y}, \ )\mathbf{T} + \mathbf{b}(\mathbf{y}, \ )^T\mathbf{T} + c(\mathbf{y}, \ ). \tag{4.1}$$

If we have such a matrix, vector, and constant, then we can approximate our data as being Gaussian distributed with a mean $\mathbf{T}$, precision matrix $\mathbf{A}(\mathbf{y}, \ )$, and response $\mathbf{A}(\mathbf{y}, \ ) \ ^1\mathbf{b}(\mathbf{y}, \ )$, which we can fit with the existing VEB-Boost machinery made for the Gaussian case. We can then maximize the lower bound to the ELBO with respect to the variational parameters at the end of each iteration.

The general CAVI algorithm is outlined below in Algorithm 3.

Algorithm 3: Coordinate Ascent Algorithm (Non-Gaussian data, fixed $T(\ )$)

Require: Data $\mathbf{y}$; functions $h_l(\ )$, prior classes $\mathcal{G}_l$, variational classes $\mathcal{Q}_l, \quad l = 1, \ldots, L$.

Require: Distribution data comes from (e.g. Binary, Ordinal, etc).

Require: Tree structure $T(\ )$; initial variational parameters .

Require: Functions $FIT : \left(h_l, \mathbf{y}, \ ^2, \mathcal{G}_l, \mathcal{Q}_l\right) \ ! \ \left(\hat{g}_l, q_l\ , \overline{\ }_l, \overline{\ ^2}_l, D_{KL}(q_l \| \hat{g}_l)\right)$ that solve the weighted Bayesian regression problem; see Corollary 3.3.1.1

Require: Function $FIT \ : \left(\mathbf{y}, T, h_l, \hat{g}_l, q_l\ , D_{KL}(q_l \| \hat{g}_l), l = 1, \ldots, L\right) \ ! \ \hat{}$ that updates the variational parameters for relevant distribution (handled internally by the VEB-Boost package).

Initialize posterior means $\overline{\ }_l$, $\overline{\ ^2}_l$, for $l = 1, \ldots, L$

Initialize variational parameters $\hat{} :=$

repeat

  Update Gaussian precision approximation $\mathbf{\Lambda}_N := \mathbf{A}(\mathbf{y}, \ )$;

  Update Gaussian response approximation $\mathbf{y}_N := \mathbf{\Lambda}_N{}^1\mathbf{b}(\mathbf{y}, \ )$;

  for *l in* $1, \ldots, L$ **do**

    Compute $(\tilde{\mathbf{y}}, \ \tilde{}^2)$ given $T, \mathbf{y}_N\ , \mathbf{\Lambda}_N\ , \overline{\ }_k, \overline{\ ^2}_k, k \neq l$; // see Theorem 3.3.2

    $(\hat{g}_l, q_l\ , \overline{\ }_l, \overline{\ ^2}_l, D_{KL}(q_l \| \hat{g}_l)) \quad FIT(h_l, \tilde{\mathbf{y}}, \ \tilde{}^2, \mathcal{G}_l, \mathcal{Q}_l)$;

  $\hat{} \quad FIT\ (\mathbf{y}, T, h_l, q_l\ , \hat{g}_l, D_{KL}(q_l \| \hat{g}_l), l = 1, \ldots, L)$;

until *convergence criterion satis ed*;

return $q_1, \ldots, q_L$.

The modularity of this approach is, in my opinion, quite powerful. As an example, say that you have binary data and you want to perform a non-parametric/non-linear logistic regression model using the VEB-Boost framework. Then as the practitioner, all you need to supply are the exact same solvers to the weighted Gaussian Bayesian regression from Section 3.2, and the R package takes care of the Gaussian approximation for you. This

substantially lowers the burden for the practitioner, since deriving and coding up a solver for the weighted Bayesian linear regression problem is often much simpler than a weighted Bayesian generalized model with non-Gaussian data.

4.2 Different Types of Non-Gaussian Data

This section outlines the different types of non-Gaussian data that can be handled by the VEB-Boost framework. Of particular note are the ranking models, accelerated failure time (AFT) model, ordinal logistic regression, and Cox proportional hazards model, as I am unaware of other instances in which these data types are approximated with Gaussian data in the context of variational inference. Note that not all of these are implemented in the R package, since some of them require dependent Gaussian approximations which is not currently supported.

4.2.1 Binary Data

First, let us consider the case of logistic regression in which the response is binary. The model we use is

$$\log\left(\frac{\mathbf{p}}{\mathbf{1}\quad\mathbf{p}}\right) = T(\ \ _1,\ldots,\ \ _L) \tag{4.2a}$$

$$y_i \ ^{?}\ Bern(p_i) \tag{4.2b}$$

$$_l \quad h_l(\ \ _l), \qquad l = 1,\ldots,L \tag{4.2c}$$

$$_l \ ^{?}\ g_l(\ ) \ 2 \ G_l, \qquad l = 1,\ldots,L. \tag{4.2d}$$

As in equation (3.4), $T(\ \ _1,\ldots,\ \ _L)$ is the tree structure of the VEB-Boost learner and $\ \ _l$ are the weak learners.

It is straightforward to show that the log-likelihood in this model is

$$l(\ \ _1,\ldots,\ \ _L;\mathbf{y},h_1,\ldots,h_L,T)=\sum_{i=1}^{n}\log\sigma\Big((2y_i\quad 1)T_i\Big),\tag{4.3}$$

where $\sigma(x)\quad\frac{1}{1+e^{\ \ x}}$ is the logistic sigmoid function and T_i is the ith value of the VEB-Boost output, $T(\ \ _1,\ldots,\ \ _L)_i$.

In order to find a quadratic (as a function of T_i) lower-bound to this log-likelihood, we utilize the bound proposed by Jaakkola and Jordan [1996].

Lemma 4.2.1 (Jaakkola-Jordan Bound)**.** *For all* $x, \xi \in \mathbb{R}$,

$\log\sigma(x)=\frac{x}{2}\quad\log(e^{x/2}+e^{\ \ x/2})\quad\frac{x}{2}\quad\frac{1}{2\xi}\Big(\sigma(\xi)\quad\frac{1}{2}\Big)(x^2\quad\xi^2)\quad\log(e^{\xi/2}+e^{\ \ \xi/2})$

This result allows us to find a quadratic lower-bound to the log-likelihood of the logistic model. In particular, we have

$$\sum_{i=1}^{n}\log\sigma\Big((2y_i\quad 1)T_i\Big)\qquad\frac{1}{2}\mathbf{T}^T\mathbf{A}(\mathbf{y},\ \ )\mathbf{T}+\mathbf{b}(\mathbf{y},\ \ )^T\mathbf{T}+c(\mathbf{y},\ \ ),\tag{4.4a}$$

where

$$\mathbf{A}(\mathbf{y},\ \ )=diag(\mathbf{d}),\quad d_i=\frac{1}{\xi_i}\Big(\sigma(\xi_i)\quad\frac{1}{2}\Big)\tag{4.4b}$$

$$\mathbf{b}(\mathbf{y},\ \ )=y_i\quad\frac{1}{2}\tag{4.4c}$$

$$c(\mathbf{y},\ \ )=\sum_{i=1}^{n}\log\sigma(\xi_i)+\frac{\xi_i}{2}(d_i\xi_i\quad 1).\tag{4.4d}$$

Here, we have introduced variational parameters $\xi_i,\ i=1,\ldots,n$.

This means that when fitting our model, we can approximate our data as Gaussian with response $\frac{1}{d_i}(y_i\quad\frac{1}{2})$ and variance $\frac{1}{d_i}$. We can then maximize the ELBO with respect to ξ_i, which we can analytically show gives us $\xi_i\ =+\sqrt{\mathbb{E}_q[T(\ \ _1,\ldots,\ \ _L)_i^2]}$. A derivation is provided in the appendix, Section A.3.2.

The Jaakkola-Jordan bound has been studied extensively since its introduction, mainly when/why it performs well. One of the more recent investigations shows that the bound amounts to a particular data augmentation strategy with Polya-Gamma distributed latent factors (Durante and Rigon [2019]); their work provides an alternate perspective of this approach.

4.2.2 Multinomial Data

Consider the setting of multinomial logistic regression with K classes. The model in this context is

$$\mathbf{s}_k = T^k(\ \overset{k}{1}, \ldots, \ \overset{k}{L_k}) \tag{4.5a}$$

$$p_{ki} := P(Y_i = k | s_{1i}, \ldots, s_{Ki}) = \frac{\exp\{s_{ki}\}}{\sum_{j=1}^{K} \exp\{s_{ji}\}} \tag{4.5b}$$

$$y_i \overset{?}{} Categorical(\mathbf{p}_i) \tag{4.5c}$$

$$\overset{k}{l} \quad h_l^k(\ \overset{k}{l}), \qquad k = 1, \ldots, K \quad l = 1, \ldots, L_k \tag{4.5d}$$

$$\overset{k}{l} \overset{?}{} g_l^k(\ ) \in G_l^k, \qquad k = 1, \ldots, K \quad l = 1, \ldots, L_k. \tag{4.5e}$$

The interpretation here is that for each class $k \in \{1, \ldots, K\}$, we estimate a separate vector $\mathbf{s}_k \in \mathbb{R}^n$. Each of these K models can have a different ensemble tree structure T^k, different weak learners $\overset{k}{l}$, etc.

The log-likelihood in this model is

$$l(\ \overset{1}{1}, \ldots, \ \overset{K}{L_K}; \mathbf{y}, h_1^1, \ldots, h_{L_k}^K, T^1, \ldots, T^K) = \sum_{i=1}^{n} \log\left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^{K} \exp\{T_i^k\}}\right). \tag{4.6}$$

In order to find a quadratic lower-bound to this log-likelihood, there are two known bounds we can use. The first bound is given by Bouchard [2008].

Lemma 4.2.2 (Bouchard Bound)**.** *For all* $\in \mathbb{R}^K, \alpha \in \mathbb{R}, \boldsymbol{x} \in \mathbb{R}^K$,

$\log \sum_{k=1}^K e^{x_k} \quad \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{b} + c,$

where

$\boldsymbol{A} = diag(\boldsymbol{d})$ *with* $d_k = \frac{1}{\xi_k}(\sigma(\xi_k) \quad \frac{1}{2})$

$\boldsymbol{b} = \alpha \boldsymbol{d} \quad \frac{1}{2}$

$c = \sum_{k=1}^K \left[\frac{\xi_k + \alpha}{2} \quad \frac{d_k}{2}(\alpha^2 \quad \xi_k^2) \quad \log(1 + e^{\xi_k})\right] \quad \alpha.$

Using this bound, we can construct a quadratic lower-bound of the log-likelihood of the multinomial model:

$$\begin{aligned} l(\ \frac{1}{1}, \ldots, \ \frac{K}{L_K}; \mathbf{y}, h_1^1, \ldots, h_{L_k}^K, T^1, \ldots, T^K) &= \sum_{i=1}^n \log\left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^K \exp\{T_i^k\}}\right) \\ &= \sum_{i=1}^n T_i^{y_i} \quad \log\Big(\sum_{k=1}^K \exp\{T_i^k\}\Big) \\ &\quad \sum_{i=1}^n T_i^{y_i} \quad \frac{1}{2}\mathbf{T}_i^T \mathbf{A}_i \mathbf{T}_i + \mathbf{T}_i^T \mathbf{b}_i + c_i \\ &= \sum_{i=1}^n \quad \frac{1}{2}\mathbf{T}_i^T \mathbf{A}_i \mathbf{T}_i + \mathbf{T}_i^T (\mathbf{b}_i + \mathbf{e}_{y_i}) + c_i \\ &= \sum_{k=1}^K \quad \frac{1}{2}\mathbf{T}^{kT} \mathbf{A}^k \mathbf{T}^k + \mathbf{T}^{kT} \mathbf{b}^k + \sum_{i=1}^n c_i, \end{aligned}$$

where

$$\begin{aligned} \mathbf{A}^k &= diag(d_1^k, \ldots, d_n^k) \\ \mathbf{b}^k &= (b_1^k + \mathbb{I}_{y_1 = k}, \ldots, b_n^k + \mathbb{I}_{y_n = k}). \end{aligned}$$

Here, we have introduced variational parameters $\xi_i^k,\ i = 1, \ldots, n$ and $k = 1, \ldots, K$, and $\alpha_i,\ i = 1, \ldots, n$.

This means that when fitting our model for class k, we can approximate our data as

Gaussian with response $\frac{1}{d_i^k}(\mathbf{1}_{y_i=k} - \frac{1}{2} + \alpha_i d_i^k)$ and variance $\frac{1}{d_i^k}$. We can then maximize the ELBO with respect to α_i and ξ_i^k, which can be done analytically. For fixed values for ξ_i^k (and hence, fixed d_i^k), we get that

$$\alpha_i = \frac{K/2 - 1 + \sum_{k=1}^K d_i^k \mathbb{E}_q[T_k(\overset{k}{1}, \ldots, \overset{k}{L_k})]_i}{\sum_{k=1}^K d_i^k}.$$

And then for fixed values of α_i, we get that

$$\xi_i^k = +\sqrt{\mathbb{E}_q[T_k(\overset{k}{1}, \ldots, \overset{k}{L_k})_i^2] - 2\alpha_i \mathbb{E}_q[T_k(\overset{k}{1}, \ldots, \overset{k}{L_k})_i] + \alpha_i^2}.$$

We can either alternate between updating the α_i's and ξ_i^k's until they converge, or just perform a single update. I walk through the algebra in a proof in the appendix, Section A.3.2.

The second bound is given by Titsias [2016].

Lemma 4.2.3 (Titsias Bound)**.** *$\forall s_k \in \mathbb{R}, k = 1, \ldots, K$*

$$\log\left(\frac{e^{s_k}}{\sum_{j=1}^K e^{s_j}}\right) \geq \sum_{j \neq k} \log \sigma(s_k - s_j)$$

This lemma can be combined with the Jaakkola-Jordan bound in Lemma 4.2.1 in order further bound each summand on the right-hand-side of this expression to get a quadratic lower-bound. Although I came up with this idea on my own, it unsurprisingly has been applied at least once before (see Snell and Zemel [2021]). Doing so will result in variational parameters ξ_i^m for $i = 1, \ldots, n$ and $m = 1, \ldots K$ ($m \neq y_i$), and as before we can define $d_i^m = \frac{1}{\xi_i^m}(\sigma(\xi_i^m) - \frac{1}{2})$.

Suppose we have an observation i with $y_i = k$. If we are fitting a model for class $m \neq k$, some arithmetic shows that we can approximate that observation as being Gaussian with response $\overline{T}_i^k - \frac{1}{2d_i^m}$ and variance $\frac{1}{d_i^m}$. If, instead, we are fitting a model for class k, some arithmetic shows that we can approximate that observation as being Gaussian with response

$\frac{1}{\sum_{m \in k} d_i^m} \sum_{m \in k} [\frac{1}{2} + d_i^m \overline{T}_i^m]$ and variance $\frac{1}{\sum_{m \in k} d_i^m}$. Again, we can maximize the ELBO over the ξ_i^m, which we can analytically show gives us $\xi_i^m = +\sqrt{\overline{T_i^{y_i 2}} - 2\overline{T_i^{y_i}}\overline{T_i^m} + \overline{T_i^{m2}}}$. A derivation of this result is given in the appendix, Section A.3.2.

4.2.3 Count Data

To handle count data, we utilize the negative binomial distribution. The reason we don't use the Poisson distribution is that if one wanted to use the typical log-link function used in Poisson regression (e.g. $\mathsf{E}[y|\mathbf{x}] = \exp(\mathbf{x}^T \beta)$), it is not possible to construct a quadratic lower-bound to the log-likelihood. It is conceivable that if one wanted to use quadratic approximations that aren't lower-bounds then this link function might be usable, however we did not explore this possibility. We did consider alternative link functions, in particular the soft-plus function: $\mathsf{E}[y|T] = \log(1 + \exp(T\beta))$. However, the lower-bound we were using (based on the work of Seeger and Bouchard [2012]) wasn't accurate enough to provide useful inference unless all observed counts were small.

It is worth noting that technically, the negative binomial distribution can be used to approximate the Poisson case. It is well-known that

$$\lim_{r \to \infty} NB\Big(r, \frac{\lambda}{r+\lambda}\Big) \stackrel{D}{=} Poisson(\lambda).$$

Thus, one could simply set a large value of r, estimate $p = \frac{\lambda}{r+\lambda}$, and then solve to get $\lambda = r\frac{p}{1-p}$. I've noticed that this sometimes works in practice, but it can take a long time to converge, and can sometimes be difficult. I suspect the challenge comes from the fact that we are trying to estimate a probability that approaches 0 as $r \to \infty$.

For negative binomial-distributed data, recall that the probability mass function is

$$NB(k; r, p) = \binom{k+r-1}{k} (1-p)^r p^k$$

for $k \in \{0, 1, 2, \ldots\}$, $r \in \{1, 2, 3, \ldots\}$, and $p \in (0, 1)$. The data generating process is the number of successes until the r-th failure among iid Bernoulli trials with success probability p. The negative binomial distribution is often used to model count data, especially when the variance doesn't follow the restrictive assumption of the Poisson distribution.

This leads to the full model

$$y_i \overset{?}{\sim} NB(r_i, p_i) \tag{4.8a}$$

$$\log\left(\frac{p_i}{1 - p_i}\right) = T(\beta_1, \ldots, \beta_L)_i \tag{4.8b}$$

$$\beta_l \sim h_l(\beta_l), \qquad l = 1, \ldots, L \tag{4.8c}$$

$$\beta_l \overset{?}{\sim} g_l(\beta) \in \mathcal{G}_l, \qquad l = 1, \ldots, L, \tag{4.8d}$$

where r_i is fixed and known.

The log-likelihood of this model is given by

$$\begin{aligned} l(\beta_1, \ldots, \beta_L; \mathbf{y}, h_1, \ldots, h_L, T) &= \sum_{i=1}^{n} \log \binom{y_i + r_i - 1}{y_i} + r_i \log \sigma(-\mathbf{T}_i) + y_i \log \sigma(\mathbf{T}_i) \\ &= \sum_{i=1}^{n} \log \binom{y_i + r_i - 1}{y_i} + y_i \mathbf{T}_i - (y_i + r_i) \log\left(1 + e^{\mathbf{T}_i}\right) \\ &= \sum_{i=1}^{n} \log \binom{y_i + r_i - 1}{y_i} + \frac{y_i - r_i}{2} \mathbf{T}_i \\ &\qquad - (y_i + r_i) \log\left(e^{\mathbf{T}_i/2} + e^{-\mathbf{T}_i/2}\right). \end{aligned}$$

From this expression of the log-likelihood, it is evident that we can apply the Jaakkola-Jordan bound in this context as well due to the $\log(e^{x/2} + e^{-x/2})$ term.

Performing some algebra, we get that we can bound the log-likelihood with

$$\sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + \frac{y_i \quad r_i}{2}\mathbf{T}_i \quad (y_i + r_i) \log \left(e^{\mathbf{T}_i/2} + e^{\quad \mathbf{T}_i/2} \right) \tag{4.9a}$$

$$\frac{1}{2}\mathbf{T}^T \mathbf{A}(\mathbf{y}, \quad)\mathbf{T} + \mathbf{b}(\mathbf{y}, \quad)^T \mathbf{T} + c(\mathbf{y}, \quad), \tag{4.9b}$$

where

$$\mathbf{A}(\mathbf{y}, \quad) = diag\Big(\mathbf{d}(\mathbf{y} + \mathbf{r})\Big), \quad d_i = \frac{1}{\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big) \tag{4.9c}$$

$$\mathbf{b}(\mathbf{y}, \quad) = \frac{\mathbf{y} \quad \mathbf{r}}{2} \tag{4.9d}$$

$$c(\mathbf{y}, \quad) = \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + (y_i + r_i)\Big[\log \sigma(\xi_i) + \frac{\xi_i}{2}(d_i \xi_i \quad 1)\Big]. \tag{4.9e}$$

This means that we can approximate our observations as coming from a Gaussian distribution with response $\frac{y_i \quad r_i}{2d_i(y_i + r_i)}$ and variance $\frac{1}{d_i(y_i + r_i)}$. We can then maximize the ELBO with respect to ξ_i, which analytically gives us $\xi_i \; = +\sqrt{\mathsf{E}_q[T(\quad_1, \ldots, \quad_L)_i^2]}$. A derivation for this result is given in the appendix, Section A.3.2.

4.2.4 Accelerated Failure Time Model (Log-Logistic Distribution)

This next extension deals with survival analysis. In particular, it deals with modeling survival times using an accelerated failure time (AFT) model using a log-logistic distributional assumption. As far as I know, the Gaussian lower-bound approximation in this context has not been done before.

The model in this setting is

$$\log \mathbf{y} = T(\ \ _1, \ldots, \ \ _L) + \tag{4.10a}$$

$$\epsilon_i \overset{iid}{\sim} logistic(0, s) \tag{4.10b}$$

$$_l \quad h_l(\ \ _l), \qquad\qquad l = 1, \ldots, L \tag{4.10c}$$

$$_l \overset{?}{\sim} g_l(\ ) \ 2 \ G_l, \qquad\qquad l = 1, \ldots, L, \tag{4.10d}$$

where y_i are the observed survival times and $s > 0$ is a scale parameter to be estimated. For reference, the logistic(0, 1) distribution has the probability density function $f(x) = \frac{e^{\ x}}{(1+e^{\ x})^2} = (e^{x/2} + e^{\ -x/2})^{\ -2}$ and cumulative distribution function $F(x) = \frac{1}{1+e^{\ -x}} = \sigma(x)$.

One wrinkle in survival analysis is that the survival time is not always observed, but rather we observe a censored time. A survival time could be left censored (failure occurred sometime in the interval $(0, t)$), right censored (failure occurred sometime in the interval $(t, \ \infty)$), or interval censored (failure occurred sometime in the interval (t_1, t_2)). In my derivations, I have made the standard assumption that censoring is non-informative and random (Patti et al. [2007]).

For censored data, the contributions to the likelihood can be shown to be $f(t)$ if there was no censoring, $F(t)$ if there was left-censoring at time t, $1 \quad F(t)$ if there was right-censoring at time t, or $F(t_2) \quad F(t_1)$ if there was interval censoring in the interval (t_1, t_2) (see, e.g., Kleinbaum and Klein [2012]). When finding a suitable quadratic lower-bound to the log-likelihood, we can deal with each type of censoring individually.

For the case with no censoring, we observe a failure at time e^{t_i}. Such an observation's

additive contribution to the log-likelihood is

$$\begin{aligned}\log f(t_i;\ T_i, s) &= \log \frac{e^{\ (t_i\ \ T_i)/s}}{s\Big(1+e^{\ (t_i\ \ T_i)/s}\Big)^2} \\ &= \quad \log(s) \quad 2\log\Big(e^{(t_i\ \ T_i)/2s} + e^{\ (t_i\ \ T_i)/2s}\Big),\end{aligned}$$

where T_i is the predicted log-survival time from the VEB-Boost model. Recall the Jaakkola-Jordan bound from Lemma 4.2.1; from the bound, it is clear that

$$\log(e^{x/2} + e^{\ x/2}) \qquad \frac{1}{2\xi}\big(\sigma(\xi) \quad \frac{1}{2}\big)(x^2 \quad \xi^2) \quad \log(e^{\xi/2} + e^{\ \xi/2}).$$

Thus, we can use this bound for our uncensored observations, resulting in the following lower-bound:

$$\begin{aligned}&\log(s) \quad 2\log\Big(e^{(t_i\ \ T_i)/2s} + e^{\ (t_i\ \ T_i)/2s}\Big) \\ &\log(s) + 2\Bigg[\quad \frac{1}{2\xi}\Big(\sigma(\xi) \quad \frac{1}{2}\Big)\Big(\big((t_i \quad T_i)/s\big)^2 \quad \xi^2\Big) \quad \log(e^{\xi/2} + e^{\ \xi/2})\Bigg].\end{aligned}$$

So we can approximate this observation as coming from a Gaussian distribution with response t_i and variance $\frac{s^2}{2d_i}$, where $d_i = \quad \frac{1}{\xi_i}\big(\sigma(\xi_i) \quad \frac{1}{2}\big)$.

For an observation that is right-censored at time e^{t_i} (i.e. the failure occurred in the interval $(e^{t_i},\ \infty)$), this observation's contribution to the log-likelihood is

$$\begin{aligned}\log\Big(1 \quad F(t_i; T_i, s)\Big) &= \log\Big(1 \quad \sigma\big((t_i \quad T_i)/s\big)\Big) \\ &= \log\sigma\big((T_i \quad t_i)/s\big).\end{aligned}$$

Using the Jaakkola-Jordan bound, it is easy to see that we can approximate this observation as coming from a Gaussian distribution with response $t_i + \frac{s}{2d_i}$ and variance $\frac{s^2}{d_i}$, where $d_i =$

$\frac{1}{\xi_i}\big(\sigma(\xi_i) \quad \frac{1}{2}\big)$.

For an observation that is left-censored at time e^{t_i} (i.e. the failure occurred in the interval $(0, e^{t_i})$), this observation's contribution to the log-likelihood is

$$\log F(t_i; T_i, s) = \log \sigma\big((t_i \quad T_i)/s)\big).$$

Using the Jaakkola-Jordan bound, it is easy to see that we can approximate this observation as coming from a Gaussian distribution with response $t_i \quad \frac{s}{2d_i}$ and variance $\frac{s^2}{d_i}$, where $d_i = \frac{1}{\xi_i}\big(\sigma(\xi_i) \quad \frac{1}{2}\big)$.

Lastly, for an observation that is interval-censored in $(e^{t_i^1}, e^{t_i^2})$, this observation's contribution to the log-likelihood is

$$\log\Big(F(t_i^2; T_i, s) \quad F(t_i^1; T_i, s)\Big) = \log\left(\frac{1}{1 + e^{\ (t_i^2 \ \ T_i)/s}} \quad \frac{1}{1 + e^{\ (t_i^1 \ \ T_i)/s}}\right).$$

Performing some algebra, we can show that this is equal to

$$\log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big) \quad \frac{t_i^1 + t_i^2}{2s} \quad \log\Big(e^{(t_i^2 \ \ T_i)/s} + e^{\ (t_i^2 \ \ T_i)/s}\Big) \quad \log\Big(e^{(t_i^1 \ \ T_i)/s} + e^{\ (t_i^1 \ \ T_i)/s}\Big).$$

We can apply the Jaakkola-Jordan bound separately for the last two terms in the above expression. Doing so leads to the conclusion that we can approximate this observation as coming from a Gaussian distribution with response $\frac{d_i^1 t_i^1 + d_i^2 t_i^2}{d_i^1 + d_i^2}$ and variance $\frac{s^2}{d_i^1 + d_i^2}$, where $d_i^1 = \frac{1}{\xi_i^1}\big(\sigma(\xi_i^1) \quad \frac{1}{2}\big)$ and $d_i^2 = \frac{1}{\xi_i^2}\big(\sigma(\xi_i^2) \quad \frac{1}{2}\big)$; the ξ_i^1 and ξ_i^2 come from the bounds obtained on the penultimate and ultimate terms in the above expression, respectively.

We can then simply add up each observation's contribution to the log-likelihood in order to have a quadratic lower-bound to the log-likelihood, and hence a lower-bound to the ELBO. This allows us to optimize the value of s by maximizing this lower-bound to the ELBO over s. Similarly, we can optimize over all variational parameters (e.g. ξ_i, ξ_i^1, ξ_i^2). A complete

derivation is provided in Section A.3.2.

4.2.5 Ordinal Logistic Regression

This next extension deals with the case of ordinal response data, i.e. the response is $y_i \in \{1, 2, \ldots, K\}$ where there exists some ordering $1 \quad 2 \quad K$. For example, the response could be the rating a viewer gives a movie, where a rating of 1 star is the worst and 5 stars is the best. We take the approach of ordinal logistic regression. Given $K+1$ "knots" $-\infty = \theta_0 < \theta_1 < \quad < \theta_{K-1} < \theta_K = \infty$, we have

$$P(y_i \quad k) = \sigma\big(\theta_k \quad T(\ _1, \ldots, \ _L)_i\big) \tag{4.11a}$$

$$P(y_i = k) = \sigma\big(\theta_k \quad T(\ _1, \ldots, \ _L)_i\big) \quad \sigma\big(\theta_{k-1} \quad T(\ _1, \ldots, \ _L)_i\big) \tag{4.11b}$$

$$= \frac{1}{1 + e^{-(\theta_k \quad T_i)}} \quad \frac{1}{1 + e^{-(\theta_{k-1} \quad T_i)}}. \tag{4.11c}$$

As far as I can tell, this model has not before been approximated with Gaussian data in the context of variational inference.

There is a clear connection between this model and the AFT model with scale parameter $s = 1$. For an observation with response $y_i = k \in \{2, \ldots, K-1\}$, this probability is the same as in the AFT model for an interval-censored observation with log-survival time censored in the interval $(\theta_{k-1}, \theta_k]$. For an observation with response $y_i = 1$, this probability is the same as the AFT model for a left-censored observation with log-survival time censored at θ_1. And for an observation with response $y_i = K$, this probability is the same as the AFT model for a right-censored observation with log-survival time censored at θ_{k-1}.

Thus, for fixed knots θ_k, we can simply fit the AFT model with scale parameter $s \quad 1$ and the log-survival times censored in the above manner. And after each iteration, we can incorporate a step to optimize the values of the knots, i.e. maximize the lower-bound of the ELBO over $-\infty = \theta_0 < \theta_1 < \quad < \theta_{K-1} < \theta_K = \infty$. A complete derivation is given in

Section A.3.2.

4.2.6 Ranking Data

Pairwise Comparisons

The Bradley-Terry model (Bradley and Terry [1952]) is one of, if not the most commonly used model when analyzing pairwise comparisons between n "players." The model associates a true rating $\pi_i > 0$ to each player i. For a pairwise comparison between players i and j, the probability that i "beats" j, denoted $i \quad j$, is given by

$$\frac{\pi_i}{\pi_i + \pi_j}.$$

When modelling these ratings, it is common to instead model the log of the true rating, $s_i \quad \log \pi_i$, sometimes referred to as a "skill." And thus,

$$\begin{aligned} P(i \quad j | s_i, s_j) &= \frac{e^{s_i}}{e^{s_i} + e^{s_j}} \\ &= \frac{1}{1 + e^{s_j \quad s_i}} \\ &= \sigma(s_i \quad s_j). \end{aligned}$$

The VEB-Boost model for pairwise comparison rankings thus becomes

$$Y_{ij} = \#\{i \quad j\} \tag{4.12a}$$

$$P(i \quad j | s_i, s_j) = \sigma(s_i \quad s_j) \tag{4.12b}$$

$$\mathbf{s} = T(\ _1, \ldots, \ _L) \tag{4.12c}$$

$$_l \quad h_l(\ _l), \qquad l = 1, \ldots, L \tag{4.12d}$$

$$_l \ \sim \ g_l(\ ) \in \mathcal{G}_l, \qquad l = 1, \ldots, L, \tag{4.12e}$$

where $\mathbf{Y}$ is a matrix whose (i, j)-th entry counts the number of times player i beat player j.

The log-likelihood of this model is given by

$$l(\ _1, \ldots, \ _L; \mathbf{Y}, h_1, \ldots, h_L, T) = \sum_{i=1}^{n} \sum_{j>i}^{1} Y_{ij} \sigma(T_i \quad T_j) + Y_{ji} \sigma(T_j \quad T_i).$$

Applying the Jaakkola-Jordan bound to each sigmoid function, we get a lower-bound of

$$\begin{aligned} l(\ _1, \ldots, \ _L; \mathbf{Y}, h_1, \ldots, h_L, T) = \sum_{i=1}^{n} \sum_{j>i}^{1} & Y_{ij} \sigma(T_i \quad T_j) + Y_{ji} \sigma(T_j \quad T_i) \\ & \sum_{i>j} \Bigg[Y_{ij} \Big[\quad \frac{d_{ij}}{2} (T_i \quad T_j)^2 + \frac{1}{2} (T_i \quad T_j) \\ & \qquad + \frac{d_{ij}}{2} (d_{ij} \xi_{ij} \quad 1) + \log \sigma(\xi_{ij}) \Big] \\ & \quad + Y_{ji} \Big[\quad \frac{d_{ji}}{2} (T_j \quad T_i)^2 + \frac{1}{2} (T_j \quad T_i) \\ & \qquad + \frac{d_{ji}}{2} (d_{ji} \xi_{ji} \quad 1) + \log \sigma(\xi_{ji}) \Big] \Bigg]. \end{aligned}$$

Knowing what we do about optimizing over ξ_{ij}, it is clear that at the end of our derivation, we will get

$$\xi_{ij} = +\sqrt{\mathbb{E}_q[(T_i \quad T_j)^2]} = +\sqrt{\mathbb{E}_q[(T_j \quad T_i)^2]} = \xi_{ji}.$$

Thus, for now, let us assume $\xi_{ij} = \xi_{ji}$ and $d_{ij} = d_{ji}$.

Careful accounting of combining like terms lets us simplify this expression as

$$\begin{aligned}
&\sum_{i=1}^{n} \frac{1}{2}T_i^2\left[\sum_{j \neq i} Y_{ij}d_{ij} + Y_{ji}d_{ji}\right] + \frac{1}{2}T_i\left[\sum_{j \neq i} Y_{ij} \quad Y_{ji}\right] \\
&\quad + \frac{1}{2}\sum_{j \neq i} T_i T_j\Big[Y_{ij}d_{ij} + Y_{ji}d_{ji}\Big] + const \\
= &\quad \frac{1}{2}\mathbf{T}^T\mathbf{L}(\mathbf{Y}, \ )\mathbf{T}^T + \mathbf{b}(\mathbf{Y}, \ ) + const,
\end{aligned}$$

where

$$\begin{aligned}
L_{ij} &= \begin{cases} \sum_{j \neq i} d_{ij}(Y_{ij} + Y_{ji}), & \text{if } i = j \\ d_{ij}(Y_{ij} + Y_{ji}), & \text{if } i \neq j \end{cases} \qquad d_{ij} = \frac{1}{\xi_{ij}}\Big(\sigma(\xi_{ij}) \quad \frac{1}{2}\Big) \\
b_i &= \frac{1}{2}\sum_{j \neq i} Y_{ij} \quad Y_{ji}.
\end{aligned}$$

The astute reader will notice that I have switched my notation for the precision matrix of the Gaussian approximation from $\mathbf{A}$ to $\mathbf{L}$. Although your first thought may be that I switched because $\mathbf{\Lambda}$ is the canonical symbol used for precision matrices, the reason I have switched notation in this case is because $\mathbf{L}$ is the canonical symbol used for the graph Laplacian. "But what does the graph Laplacian have to do with this setting?" you may be asking.

For a refresher on what the graph Laplacian is, consider an undirected weighted graph. The adjacency matrix of this graph, $\mathbf{A}$, is defined such that A_{ij} is the weight of the edge connecting nodes i and j. The degree matrix, $\mathbf{D}$, is a diagonal matrix defined such that D_{ii} is the sum of the weights of all edges connected to node i. The graph Laplacian is then defined as $\mathbf{L} = \mathbf{D} \quad \mathbf{A}$.

Now, let us define a graph $G = (V, E)$, where the vertices are the players, $1, \ldots, n$, and the

weight of the edge between nodes i and j is $d_{ij}(Y_{ij} + Y_{ji})$. Then the Laplacian of this graph is equal to the precision matrix in the Gaussian approximation we've derived above. Noting that $Y_{ij} + Y_{ji}$ is equal to the number of times players i and j competed against each other, this leads to the intuition that the precision of our approximation depends on how often these players competed; the more times they played against each other, the more certain we are about their relative skill. Graphs have long played a role in ranking problems; the class of methods known as spectral ranking methods, of which Google's famous PageRank algorithm is a member, center around graphs, adjacency matrices, etc (Vigna [2019]). So this is a neat little connection to the existing ranking literature.

Those familiar with Bradley-Terry models will recall that the likelihood is translation-invariant in $\mathbf{s}$, i.e. we can shift all of our skills s_i by a constant c and the resulting likelihood does not change. The complication that this additional degree of freedom brings manifests itself in the precision matrix $\mathbf{L}$. Assume that the graph G we defined above is connected, i.e. there exists a path from every node i to every other node j. Then the Laplacian of this graph, and hence our precision matrix, has rank n 1. This means that our Gaussian approximation is a degenerate Gaussian. There are two obvious ways that we can remedy the situation:

1. We can arbitrarily pick a player i and restrict s_i 0; or equivalently, we change what we are modeling from s_j to s_j s_i;

2. We can add a small multiple of the identity matrix to $\mathbf{L}$

I have tried option (1) in my experiments, but have not yet tried option (2).

The more complicated setting is where our graph G is not connected, but is instead comprised of K disjoint connected graphs. The interpretation is that there are K leagues of players who play within their league, but there are no "exhibition games" between leagues. In this setting, we now have K additional degrees of freedom, since we can shift the scores in league k by a constant c_k independently for all leagues, and the likelihood will not change.

This makes sense, since there is no way to compare the relative skills between teams in different leagues. In this case, the graph Laplacian will have rank $n - K$. Similarly, there are a few obvious ways that we can remedy the situation:

1. In each league $k \in \{1, \ldots, K\}$, we can arbitrarily pick a player i_k and restrict $s_{i_k} = 0$;
2. We can add a small multiple of the identity matrix to $\mathbf{L}$

I have not experimented with either option. Just at a glance, option 1 appears flawed, because it equalizes arbitrary players across leagues, even though they could have vastly different skill levels.

One way to side-step all of these issues entirely is to take a slightly different approach: instead of learning a mapping from $\mathbf{x}_i \to s_i$, learn a mapping from $(\mathbf{x}_i, \mathbf{x}_j) \to s_i - s_j$. To do this, we simply form a new input feature $\tilde{\mathbf{x}}_g$, which is simply the catenation of $\mathbf{x}_i$ and $\mathbf{x}_j$ for a "game" g between players i and j. The new response is $y_g = \mathbb{1}_{i \text{ beat } j \text{ in game } g}$, and we can just run a logistic regression with these new features and response. This is in the spirit of dyad ranking (Schäfer and Hüllermeier [2018]).

This approach introduces a new issue, since it is ambiguous if we should add an observation with feature vector $(\mathbf{x}_i, \mathbf{x}_j)$ and response $\mathbb{1}_{i \text{ beat } j \text{ in game } g}$, or if we should add an observation with feature vector $(\mathbf{x}_j, \mathbf{x}_i)$ and response $\mathbb{1}_{j \text{ beat } i \text{ in game } g}$. Consider the simple case of a linear model, where our coefficient vector is $(\frac{1}{2}, -\frac{1}{2})$, i.e. our model is $\beta_0 + (\mathbf{x}_i - \mathbf{x}_j)^T$. Then the intercept of this model, β_0, is the advantage of being placed first in the catenation, like a "home field advantage" in sports (see, e.g., Agresti [2003]). So if the observations have a symmetry-breaking feature such as which player was the home team, then this method could conceivable be explored. However, I have not performed any experiments with this approach.

Listwise Comparisons

The Plackett-Luce model (Plackett [1975]) is a commonly used model when analyzing listwise ranking between n "players," and is an extension of the Bradley-Terry model. Just as how the response for each observation in the Bradley-Terry model was a pairwise ranking between two players, the response for each observation in the Plackett-Luce model is a complete ranking of 2 players. And also like the Bradley-Terry model, the Plackett-Luce model associates a true rating $\pi_i > 0$ to each player i. For a listwise ranking between these players, the probability of observing the total ordering $i_1 \quad i_2 \quad \ldots \quad i_n$ is given by

$$\prod_{j=1}^{n} \frac{\pi_{i_j}}{\sum_{k \quad j} \pi_{i_k}}.$$

In words, this is the probability of observing the total ordering $i_1 \quad i_2 \quad \ldots \quad i_n$ in the process of first choosing i_1 as the overall best player, then choosing i_2 as the best from the remaining players, and so on until there are no players left.

The terminology of listwise rankings is taken from the world of document/information retrieval. Let the total number of players be n and their underlying ratings be π_i. Let the total number of ranked lists/queries be J, and let $q_j \quad \{1, \ldots, n\}$ be the subset of players seen in query j. And for query j, let the observed ranking be $r_j \in \sigma_{q_j}$, where σ_{q_j} is the set of all permutations of elements in q_j. So for a query j and observed ranking r_j, we have $r_{j1} \quad r_{j2} \quad \ldots \quad r_{jjq_jj}$. With this notation in place, we can now write the likelihood of the set of all rankings as

$$\prod_{j=1}^{J} \prod_{t=1}^{jq_jj \ 1} \frac{\pi_{r_{jt}}}{\sum_{v=t}^{jq_jj} \pi_{r_{jv}}} = \prod_{j=1}^{J} \prod_{t=1}^{jq_jj \ 1} \frac{e^{s_{r_{jt}}}}{\sum_{v=t}^{jq_jj} e^{s_{r_{jv}}}}].$$

Taking logs, we get a log-likelihood of

$$\sum_{j=1}^{J} \sum_{t=1}^{|q_j|} \log \left(\frac{e^{s_{r_{jt}}}}{\sum_{v=t}^{|q_j|} e^{s_{r_{jv}}}} \right).$$

Recognizing the log-sum-exponential in this expression, we can appeal to either the Bouchard bound or Titsias bound in order to derive a quadratic lower bound.

Starting with the Bouchard bound, we can derive the bound as

$$\sum_{j=1}^{J} \sum_{t=1}^{|q_j|} \log \left(\frac{e^{s_{r_{jt}}}}{\sum_{v=t}^{|q_j|} e^{s_{r_{jv}}}} \right) = \sum_{j=1}^{J} \sum_{t=1}^{|q_j|} s_{r_{jt}} \quad \log \left(\sum_{v=t}^{|q_j|} e^{s_{r_{jv}}} \right)$$

$$[Bouchard]$$

$$\sum_{j=1}^{J} \sum_{t=1}^{|q_j|} s_{r_{jt}} \quad \frac{1}{2} \mathbf{s}_{r_j^t}^T \mathbf{A}_j^t \mathbf{s}_{r_j^t}^T + \mathbf{s}_{r_j^t} \mathbf{b}_j^t + c_j^t,$$

where

$$\mathbf{s}_{r_j^t} = (s_{r_{jt}}, \ldots, s_{r_{j|q_j|}})^T \ 2 \ \mathsf{R}^{|q_j| \ t+1}$$

is the vector of scores for players in query q_j with a rank t

$$\mathbf{A}_j^t = diag(\mathbf{d}_j^t), \quad d_{jv}^t = \frac{1}{\xi_{jv}^t} \Big(\sigma(\xi_{jv}^t) \quad \frac{1}{2} \Big)$$

$$\mathbf{b}_j^t = \alpha_j^t \mathbf{d}_j^t \quad \frac{1}{2} \mathbf{1}$$

$$c_j^t = \sum_{v=t}^{|q_j|} \Big[\frac{\xi_{jv}^t + \alpha_j^t}{2} \quad \frac{d_{jv}^t}{2} (\alpha_j^{t\,2} \quad \xi_{jv}^{t\,2}) \quad \log(1 + e^{\xi_{jv}^t}) \Big] \quad \alpha_j^t.$$

As before, we can show that when optimizing over ξ_{jv}^t, we can perform a coordinate

maximization step by setting

$$\xi_{jv}^t = +\sqrt{\overline{s_{r_{jv}}^2} \quad 2\alpha_j^t \overline{s_{r_{jv}}} + \alpha_j^{t\,2}}.$$

And when optimizing over α_j^t, we can perform a coordinate maximization step by setting

$$\alpha_j^t = \frac{\left(\frac{jq_jj \quad t+1}{2} \quad 1\right) \sum_{v=t}^{jq_jj} d_{jv}^t \overline{s_{r_{jv}}}}{\sum_{v=t}^{jq_jj} .d_{jv}^t}$$

To reduce the above expression to a quadratic in $\mathbf{s} = \mathbf{T}$, consider a query q_j and a rank t 2 f1, . . . , jq_jj 1g. Define the matrix $\tilde{\mathbf{A}}_j^t$ 2 $\mathsf{R}^{n \quad n}$ to have its (jq_jj $t+1$) (($jq_j + t$ 1) sub-matrix corresponding to players in query q_j with rank t given by $\mathbf{A}_j^t$, and 0 elsewhere. And similarly, define the vector $\tilde{\mathbf{b}}_j^t$ 2 R^n to have its entries corresponding to the players in query q_j with rank t be given by $\mathbf{b}_j^t$, and 0 elsewhere. Then we can re-write the lower-bound as

$$\begin{aligned}
&\sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} s_{r_{jt}} \quad \frac{1}{2}\mathbf{s}_{r_j^t}^T \mathbf{A}_j^t \mathbf{s}_{r_j^t} + \mathbf{s}_{r_j^t}^T \mathbf{b}_j^t + c_j^t \\
=&\sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} s_{r_{jt}} \quad \frac{1}{2}\mathbf{s}^T \tilde{\mathbf{A}}_j^t \mathbf{s} + \mathbf{s}^T \tilde{\mathbf{b}}_j^t + c_j^t \\
=&\sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} \quad \frac{1}{2}\mathbf{s}^T \tilde{\mathbf{A}}_j^t \mathbf{s} + \mathbf{s}^T \left(\tilde{\mathbf{b}}_j^t + \mathbf{e}_{rjt}\right) + c_j^t \\
=& \quad \frac{1}{2}\mathbf{s}^T \left[\sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} \tilde{\mathbf{A}}_j^t\right] \mathbf{s} + \mathbf{s}^T \left[\sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} \left(\tilde{\mathbf{b}}_j^t + \mathbf{e}_{rjt}\right)\right] + \sum_{j=1}^{J} \sum_{t=1}^{jq_jj \quad 1} c_j^t.
\end{aligned}$$

The above expression is a quadratic in $\mathbf{s} = \mathbf{T}$, so we can use it to form our Gaussian approximation.

Just as a note, we could have instead applied the Bouchard bound to

$\log\left(\sum_{v=t}^{jq_jj} e^{s_{rjv}} \quad s_{rjt}\right)$. This would have given us a correlated Gaussian approximation, just like in the Bradley-Terry model. It is unclear to me which, if either, is preferable. But I suspect this other approach may have more potential, simply due to the increased flexibility afforded by the correlated Gaussian approximation. This is one area where more exploration is needed.

Turning to the Titsias bound, we stumble across an interesting connection between the bound and ranking data. In Titsias's paper, they hypothesize that there is a connection between the bound and Bradley-Terry models. Once we see the bound applied to the Plackett-Luce model, the supposed connection will be immediate.

Focusing on just the top ranking for a single query j in which $r_{j1} \quad r_{j2} \quad r_{jjq_jj}$, we can apply the Titsias bound to get a lower-bound to the likelihood of

$$\frac{e^{s_{rj1}}}{\sum_{v=1}^{jq_jj} e^{s_{rjv}}} \quad \prod_{v>1} \sigma(s_{rj1} \quad s_{rjv}).$$

We can immediately recognize the right-hand size of this bound as the probability in a Bradley-Terry model where we observe the pairwise rankings

$r_{j1} \quad r_{j2}$, $r_{j1} \quad r_{j3}$, $\ldots$, $r_{j1} \quad r_{jjq_jj}$. That is, we bound the probability of having player r_{j1} be the top-ranked item from the query with the probability of player r_{j1} being preferred to each other player in the query in a pairwise comparison. Intuitively, this bound makes sense; it seems much more likely that a player will be voted the best among a group of players when we're making a complete ordering, as opposed to that player winning every single head-to-head comparison without a single loss. The case in which these probabilities are close is when player r_{j1} is *signi cantly* better than all other players in the query, i.e. $s_{rj1} \quad s_{rjv}$ for $v = 2, \ldots, jq_jj$, since the chance of the top player losing in any one of the pairwise comparisons is negligible. This jibes with the math, which says that the Titsias bound will be tight when $s_{rj1} \quad s_{rjv}$ for $v = 2, \ldots, jq_jj$.

We can then repeat the above with the remaining items in the query. As an example, we would convert the ranking 1 2 3 4 into the pairwise rankings 1 2, 1 3, 1 4, 2 3, 2 4, and 3 4. And keeping with the above intuition, the bound we get would be tight if s_1 s_2 s_3 s_4, i.e. there is a clear separation between the skills of all players.

Now, applying this result to all top rankings in all queries, we get an overall lower-bound to the likelihood of

$$\prod_{j=1}^{J} \prod_{t=1}^{jq_jj \ 1} \frac{e^{s_{r_{jt}}}}{\sum_{v=t}^{jq_jj} e^{s_{r_{jv}}}} \quad \prod_{j=1}^{J} \prod_{t=1}^{jq_jj \ 1} \prod_{v=t+1}^{jq_jj} \sigma(s_{rjt} \quad s_{rjv}).$$

So we can simply convert all of our queries to pairwise comparisons, and then fit the resulting comparisons as a Bradley-Terry model. Casting listwise comparisons as pairwise comparisons is frequently done to fit listwise rankings data with pairwise methods (Cao et al. [2007]), and the Titsias bound appears to be at least one justification for doing so.

4.2.7 Cox Proportional Hazards Model

Similar to the AFT model, the Cox proportional hazards model deals with survival time data. However, the assumptions made aren't of a particular parametric data distribution, but rather the semi-parametric proportional hazards assumption:

$$\lambda(tjX_i) = \lambda_0(t)\theta_i.$$

Here, λ is the hazard function, λ_0 is the baseline hazard function, and $\log \theta_i = T_i$ is the quantity being modelled.

Let the observed survival times be the pairs (y_i, c_i), where y_i is the observed time (or right-censored time), and c_i is an indicator for if observation i is observed (i.e. not right-censored). Note that unlike the AFT model, I am only allowing for right-censored observations and not

left-censored or interval-censored observations.

Using Breslow's method for handling ties (Breslow [1974]), we can write the partial log-likelihood of the model as

$$l(\ \ _1, \ldots, \ \ _L; \mathbf{y}, \mathbf{c}, h_1, \ldots, h_L, T) = \sum_{i:c_i=1} \log \frac{e^{T_i}}{\sum_{j:y_j \ \ y_i, j \in i} e^{T_j}}.$$

Noting that this expression contains the log-sum-exponential function, we can utilize the Bouchard or Titsias bounds in order to approximate this data as Gaussian. I will note that my initial experiments of trying to use the Bouchard bound failed spectacularly for unknown reasons (but my best guess is that the bound performs too poorly, especially when the log-sum-exponential function contains many terms). However, the Titsias bound appears to provide better results, with the drawback that the Gaussian approximation we derive is not an independent Gaussian. While the VEB-Boost framework can handle correlated Gaussian data, this feature is not implemented in the R package. However, I will still state the results here, since this appears to be a new application of the Titsias bound.

When applying the Titsias bound and then Jaakkola-Jordan bound to this data and performing some algebra, we arrive at the result that

$$l(\ \ _1, \ldots, \ \ _L; \mathbf{y}, \mathbf{c}, h_1, \ldots, h_L, T) = \sum_{i:c_i=1} \log \frac{e^{T_i}}{\sum_{j:y_j \ \ y_i, j \in i} e^{T_j}} \\ \frac{1}{2}\mathbf{T}^T \mathbf{A}(\mathbf{y}, \mathbf{c}, \ \ )\mathbf{T} + \mathbf{b}(\mathbf{y}, \mathbf{c}, \ \ )^T \mathbf{T} + c(\mathbf{y}, \mathbf{c}, \ \ ),$$

where

$$A_{ij} = \begin{cases} c_i \quad \left[\sum_{k:y_k \quad y_i, k \neq i} d_{ik}\right] \quad \sum_{k:y_i \quad y_k, c_k = 1, k \neq i} d_{ki}, & \text{if } i = j \\ c_i \quad \mathsf{I}_{y_j \quad y_i} d_{ij} + c_j \quad \mathsf{I}_{y_i \quad y_j} d_{ji}, & \text{if } i \neq j \end{cases}$$

$$d_{ij} = \frac{1}{\xi_{ij}}\left(\sigma(\xi_{ij}) \quad \frac{1}{2}\right)$$

$$\mathbf{b}(\mathbf{y}, \mathbf{c}, \quad)_i = \frac{1}{2}\left[c_i \quad \sum_{j:c_j = 1, j \neq i} \mathsf{I}_{y_i \quad y_j}\right]$$

$$c(\mathbf{y}, \mathbf{c}, \quad) = \sum_{i:c_i = 1} \sum_{j:y_j \quad y_i, j \neq i} \log \sigma(\xi_{ij}) + \frac{\xi_{ij}}{2}\left(d_{ij}\xi_{ij} \quad 1\right).$$

And we would update the variational parameters with

$$\xi_{ij} = +\sqrt{\overline{T_i^2} \quad 2\overline{T_i T_j} + \overline{T_j^2}}, \quad \text{for } (i,j) \text{ s.t. } c_i = 1 \;\&\; y_j \quad y_i \;\&\; j \neq i.$$

This yields a Gaussian approximation with precision matrix $\mathbf{A}(\mathbf{y}, \mathbf{c}, \quad)$ and response $\mathbf{A}(\mathbf{y}, \mathbf{c}, \quad) \quad ^1\mathbf{b}(\mathbf{y}, \mathbf{c}, \quad)$. A complete derivation is given in Section A.3.2.

4.2.8 Multivariate Gaussian Data

While this last type of data is technically Gaussian, I believe that it is still different enough to include. Note that I have not attempted to implement this, so I haven't performed any experiments to gauge its potential.

Multiple response data is the scenario where each observation's response is vector-valued. The idea is that if you know a bit about the structure of the relationships between the different elements of the response, then you can potentially increase power/accuracy/etc by modelling them jointly. For simplicity, I will assume that the response is an $R \quad n$-dimensional

matrix Normal, i.e. the model is

$$\mathbf{Y} \quad MN_{R\ n}(\mathbf{T}, \mathbf{V}, \mathbf{U}) \tag{4.13a}$$

$$\mathbf{T} = T(\ _1, \ldots, \ _L) \in \mathbb{R}^{R\ \ n} \tag{4.13b}$$

$$_l = h_l(\ _l), \qquad l = 1, \ldots, L \tag{4.13c}$$

$$_l \overset{?}{\ } g_l(\ ) \in G_l, \qquad l = 1, \ldots, L. \tag{4.13d}$$

Here, $\mathbf{V} \in S^R_{++}$ is a covariance matrix that describes the relationship between the R response variables and $\mathbf{U} \in S^n_{++}$ is a covariance matrix that describes the relationship between the observations.

Equivalently, $vec(\mathbf{Y}) \quad N_{Rn}(vec(\mathbf{T}), \mathbf{U} \bigotimes \mathbf{V})$, where $vec(\ )$ is the vectorization operator of a matrix (i.e. stacking the columns on top of each other) and $\bigotimes$ is the Kronecker product. Written out for clarity, this gives

$$\begin{pmatrix} \mathbf{Y}_{\ 1} \\ \vdots \\ \mathbf{Y}_{\ n} \end{pmatrix} \quad N_{Rn}\left(\begin{pmatrix} \mathbf{T}_{\ 1} \\ \vdots \\ \mathbf{T}_{\ n} \end{pmatrix}, \mathbf{\Sigma} \right),$$

where

$$\mathbf{\Sigma} = \mathbf{U} \bigotimes \mathbf{V} = \begin{bmatrix} U_{11}\mathbf{V} & \ldots & U_{1n}\mathbf{V} \\ \vdots & \ddots & \vdots \\ U_{n1}\mathbf{V} & \ldots & U_{nn}\mathbf{V} \end{bmatrix} = \mathbf{\Lambda}^{\ 1}.$$

Just as a note, if the observations are independent, $\mathbf{U}$ will be diagonal, and so $\mathbf{\Sigma}$ will be block-diagonal, as will $\mathbf{\Lambda}$.

We can simply apply the same VEB-Boost machinery proved in A.3.1, with the proper vectorization of our now matrix-valued weak learners $_l \in \mathbb{R}^{R\ \ n}$. When I started doing the math, my hope was that we'd be able to have the building-block of multivariate VEB-

Boost be solving the multivariate Bayesian regression with matrix normal response, however the math ended up showing that we do not get a matrix normal when we (Schur) multiply learners together.

As a note, the multivariate single effect regression has a closed-form solution, which is a generalization of the regular SER. It involves some Kronecker products with identity matrices that reduce to 1 in the regular SER case and yields the same closed-form updates.

4.3 Examples

In this section, I walk through a simulation study using logistic regression, as well as a real-data benchmarking analysis using both the logistic and multinomial logistic models. In the multinomial case, I use both bounds in the VEB-Boost package: Bouchard and Titsias.

4.3.1 Simulation Study

In this subsection, I present the results from a small simulation study. I generate independent observations according to the following model:

$$y_i \overset{?}{} Bern\left(\frac{1}{1+e^{-f(\mathbf{x}_i)}}\right).$$

For $f(\mathbf{x})$, I use the same four functions as in Section 3.6.1: Friedman's 5-dimensional test function, max test function, linear test function, and null test function. However, I have modified the Friedman test function by subtracting 15 so that we get a more balanced response.

To generate the design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, I simulated $x_{ij} \overset{iid}{} Unif(0,1)$ in the case of Friedman's function, and $x_{ij} \overset{iid}{} \mathcal{N}(0,1)$ otherwise.

Since all methods included in the study from Section 3.6.1 can also handle binary data, I have included them in this simulation study as well. I have also added a few additional

methods:

- **"VEBBoostGaussian"**: VEB-Boost with homoskedastic Gaussian response

  I included this method to compare directly with the logistic model. In order to return a fitted probability, I used min$f$1 1e-8, max$f$1e-8, posterior meangg;

- **"SPORF"**: Sparse Projection Oblique Random Forests (Tomita et al. [2020])

  I opted to use the defaults and did not perform any cross-validation. However, it definitely could have benefited from such. I used the R package available at `https://github.com/neurodata/SPORF`.

For BART and XBART, the average posterior probability was used as the prediction, and for VEB-Boost the inverse-logit-transform of the posterior mean was used. I also looked into using the average of the inverse-logit-transform, which is technically more "proper," but the results were very similar so they are not included.

I ran 5 replicates each of all combinations of $(n, p, test\_function)$ 2 $f1000, 10000, 100000g$ $f10, 100, 1000g$ $f friedman, max, linear, nullg$. One thing to note is that for many runs with $n = 100000, p = 1000$, XGBoost, VEB-Boost with a Gaussian response, and XBART typically either ran out of RAM or ran out of time, so results are missing for them in those settings. This could certainly introduce bias into the results.

Using the same test functions as in the Gaussian simulation can help us diagnose issues with the Gaussian approximation. For example, in the Gaussian case, we know that VEB-Boost far out-performed the others with the linear test function. If VEB-Boost is now under-performing with the linear test function, this would certainly give us pause. To evaluate the performance of the methods, I used three relative performance metrics:

1. Relative logloss (relative to the minimum);
2. Relative AUC (relative to the maximum);
3. Relative $(1 + MCC)/2$ (relative to the maximum).

Since Matthews correlation coefficient (MCC) falls between $[\ \ 1, 1]$, performing this transformation puts everything on the $[0, 1]$ scale. We chose MCC over Cohen's κ in this comparison, since it has been shown to exhibit more desirable behavior in the cases when the two metrics qualitatively disagree (Delgado and Tibau [2019]). The MCC was calculated using predictions set with a thresholded probability of 0.5.

I also use the relative and absolute running times of the algorithms to compare how fast they are; the times are all relative to the fastest algorithm for that particular dataset.

Just like in the linear simulation study, I employ the use of boxplots and profile plots. The boxplots for the Friedman test function are presented below; boxplots for the other test functions are provided in Section A.2.2, Figures A.7 through A.15. In general, VEB-Boost still performs well, but not quite as well as the Gaussian simulations from Section 3.6.1.

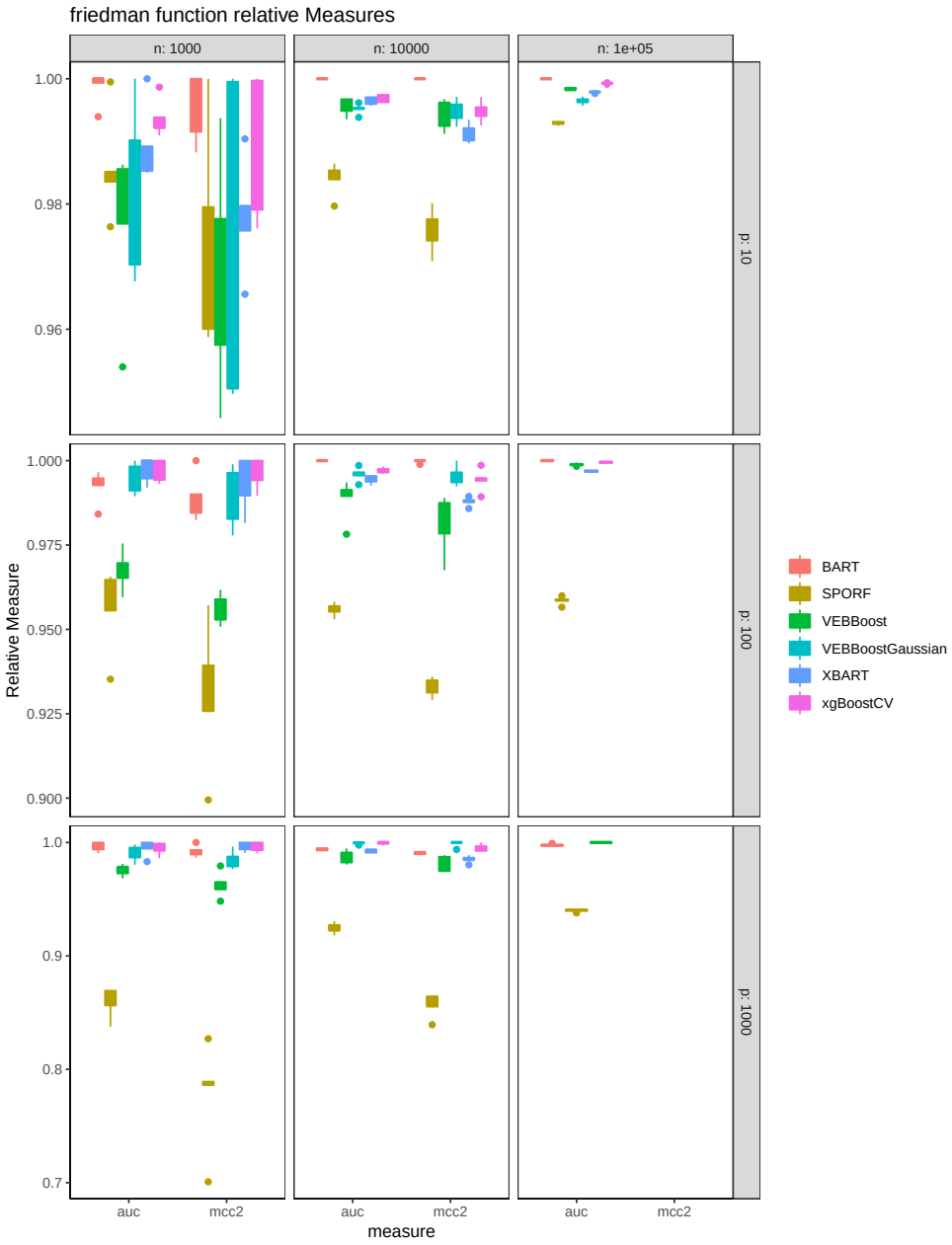

Figure 4.1: **Friedman Function Relative AUC and MCC** This plot shows the relative AUC and $(1 + MCC)/2$ for the logistic model using the Friedman test function (higher is better). We can see that BART and XGBoost are quite good, with VEB-Boost close behind. Somewhat interesting is that using the Gaussian VEB-Boost model seems to yield better values of AUC and MCC than the logistic VEB-Boost model. This is quite interesting, and is worth further exploration.

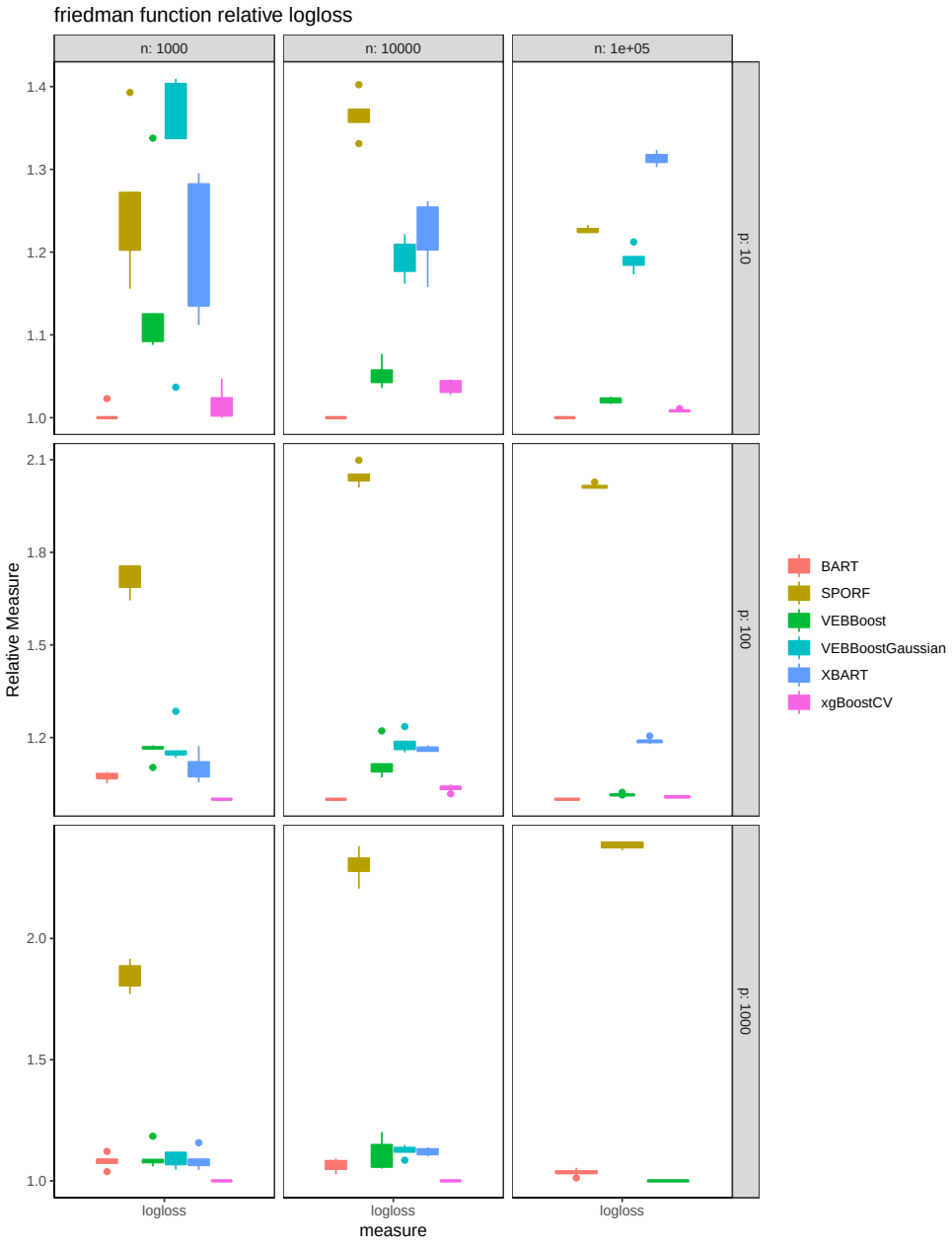

Figure 4.2: **Friedman Function Relative logloss** This plot shows the relative logloss for the logistic model using the Friedman test function (lower is better). Somewhat unsurprisingly, the Gaussian VEB-Boost model is not able to provide calibrated probabilities, especially around 0 and 1, so it suffers on the logloss metric. We see that XGBoost and BART still appear to be the best, with VEB-Boost not far behind.

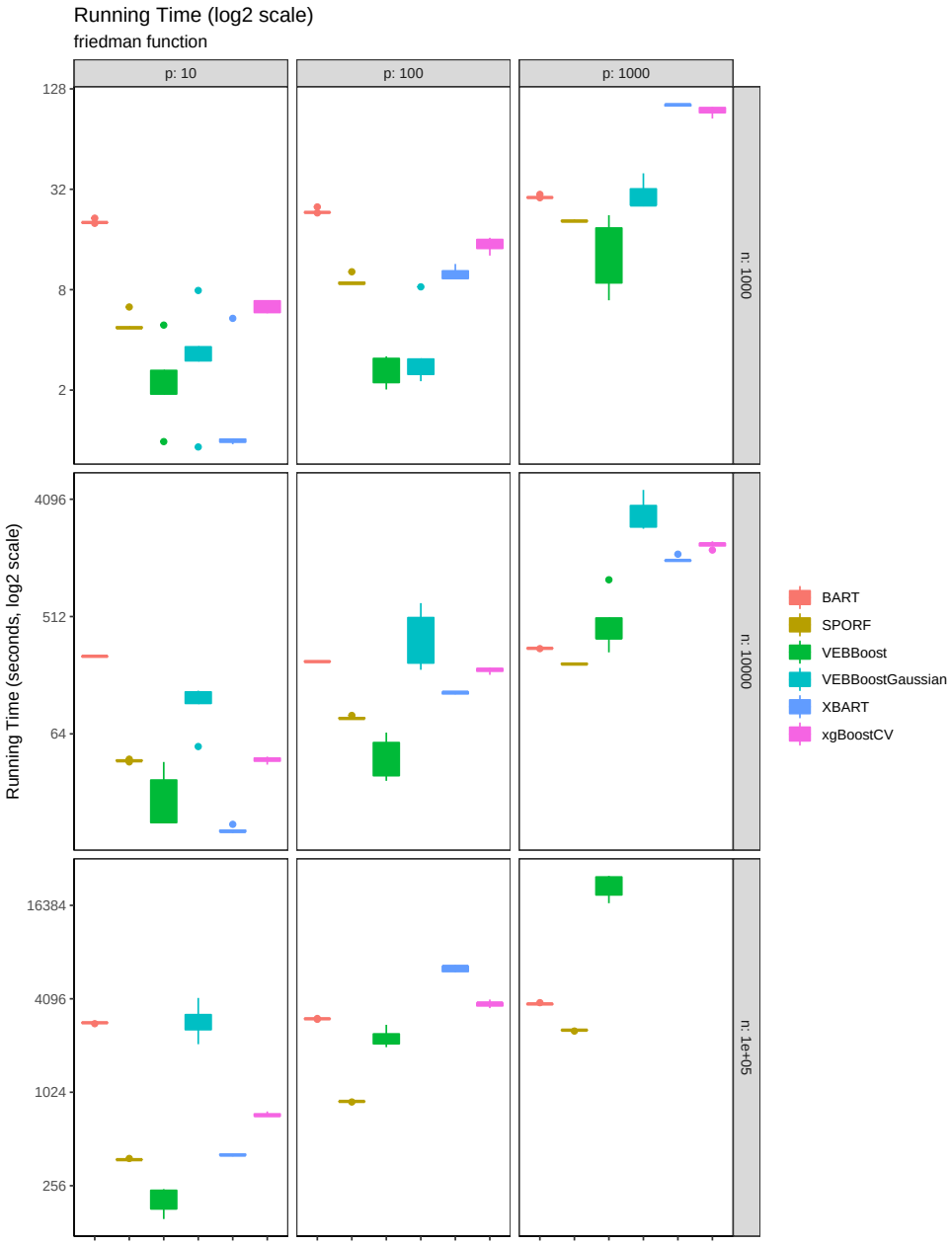

Figure 4.3: **Friedman Function Running Time ($\log_2$ scale)** This plot shows the running time for the logistic model using the Friedman test function (lower is better). We see that the running times for VEB-Boost are quite competitive in most cases. We also see the the Gaussian VEB-Boost model is much slower than the logistic VEB-Boost model.

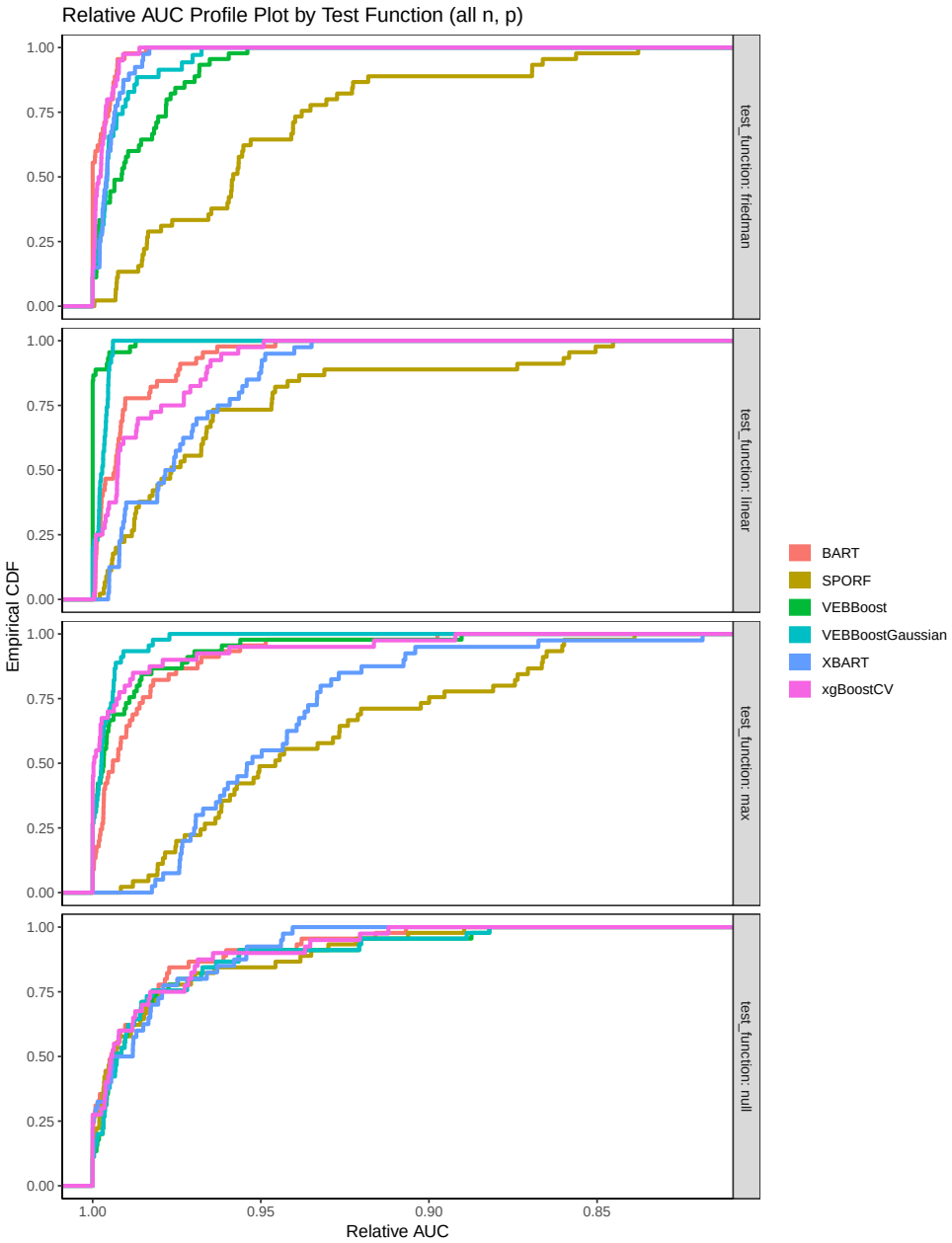

Figure 4.4: **Profile Plot of Relative AUC** This plot shows the empirical CDF of the relative AUC from the logistic simulation by test function. In general, the VEB-Boost models appear quite competitive. But as we saw in the boxplots above, the Gaussian VEB-Boost model appears to outperform the logistic VEB-Boost model.

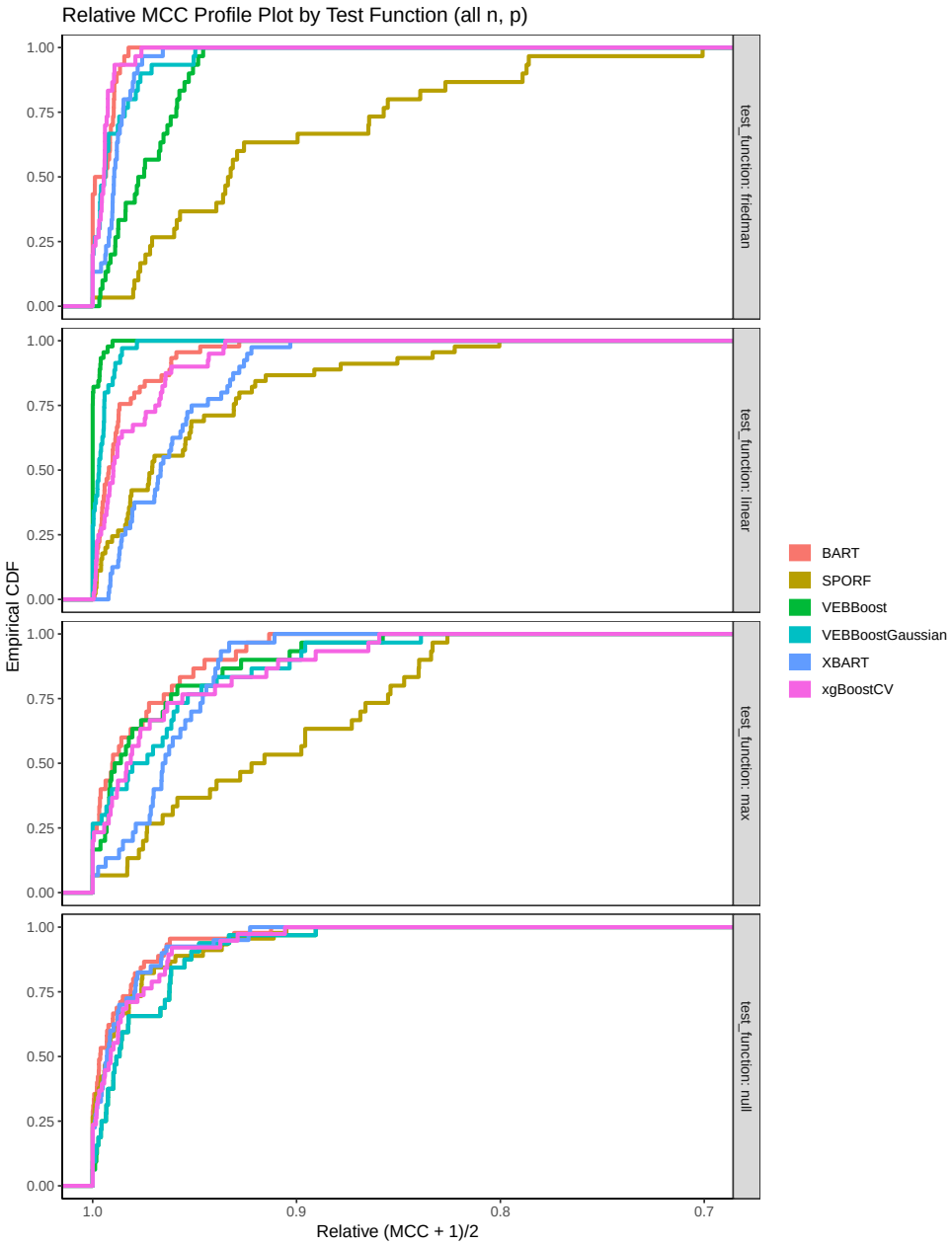

Figure 4.5: **Profile Plot of Relative** $(1 + MCC)/2$ This plot shows the empirical CDF of the relative MCC from the logistic simulation by test function. Again, the Gaussian VEB-Boost model appears to perform better than the logistic VEB-Boost model

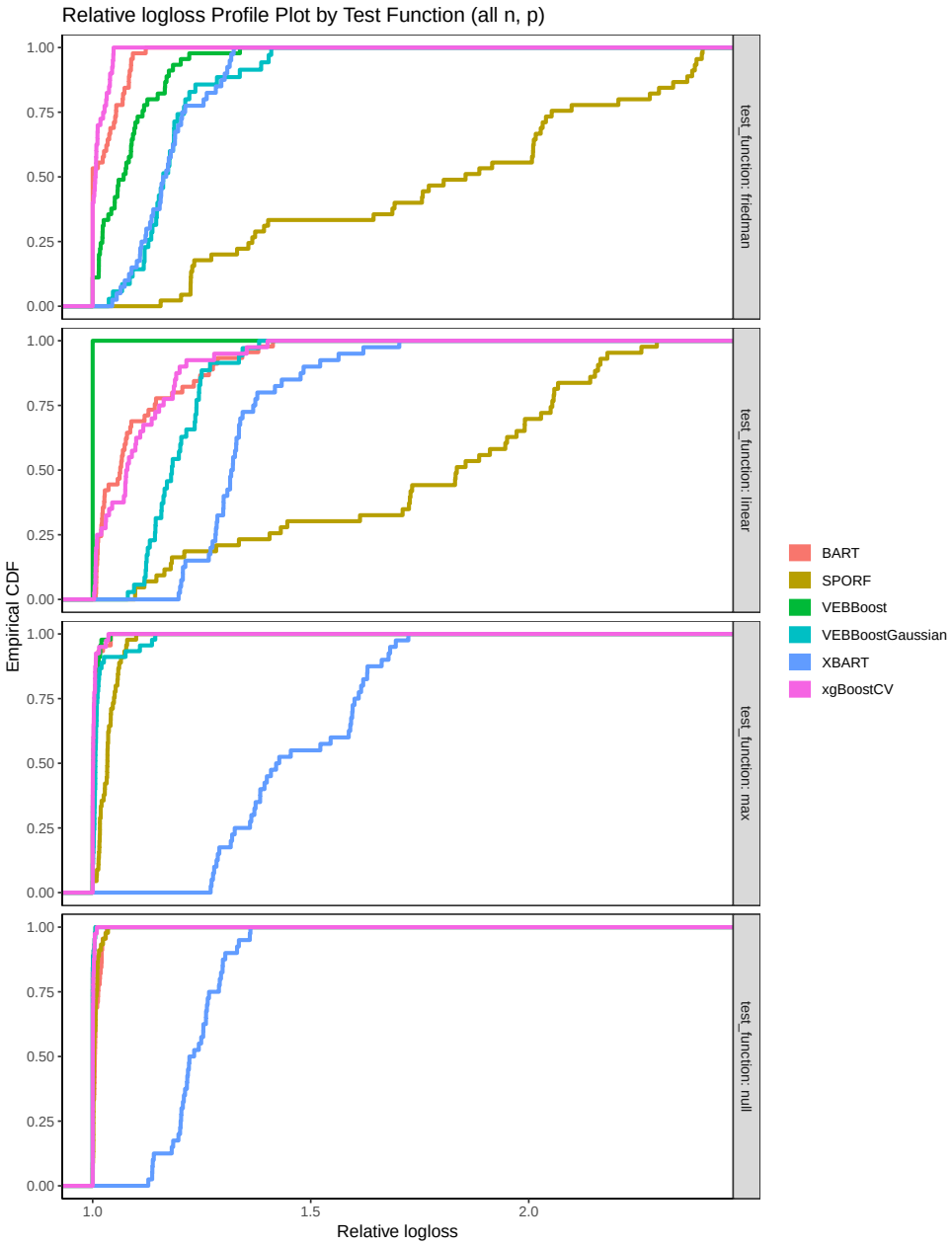

Figure 4.6: **Profile Plot of Relative logloss** This plot shows the empirical CDF of the relative logloss from the logistic simulation by test function. As we saw above, the Gaussian VEB-Boost model struggles to provide calibrated probabilities, and thus it suffers on the logloss metric. The logistic VEB-Boost model appears competitive with both BART and XGBoost.

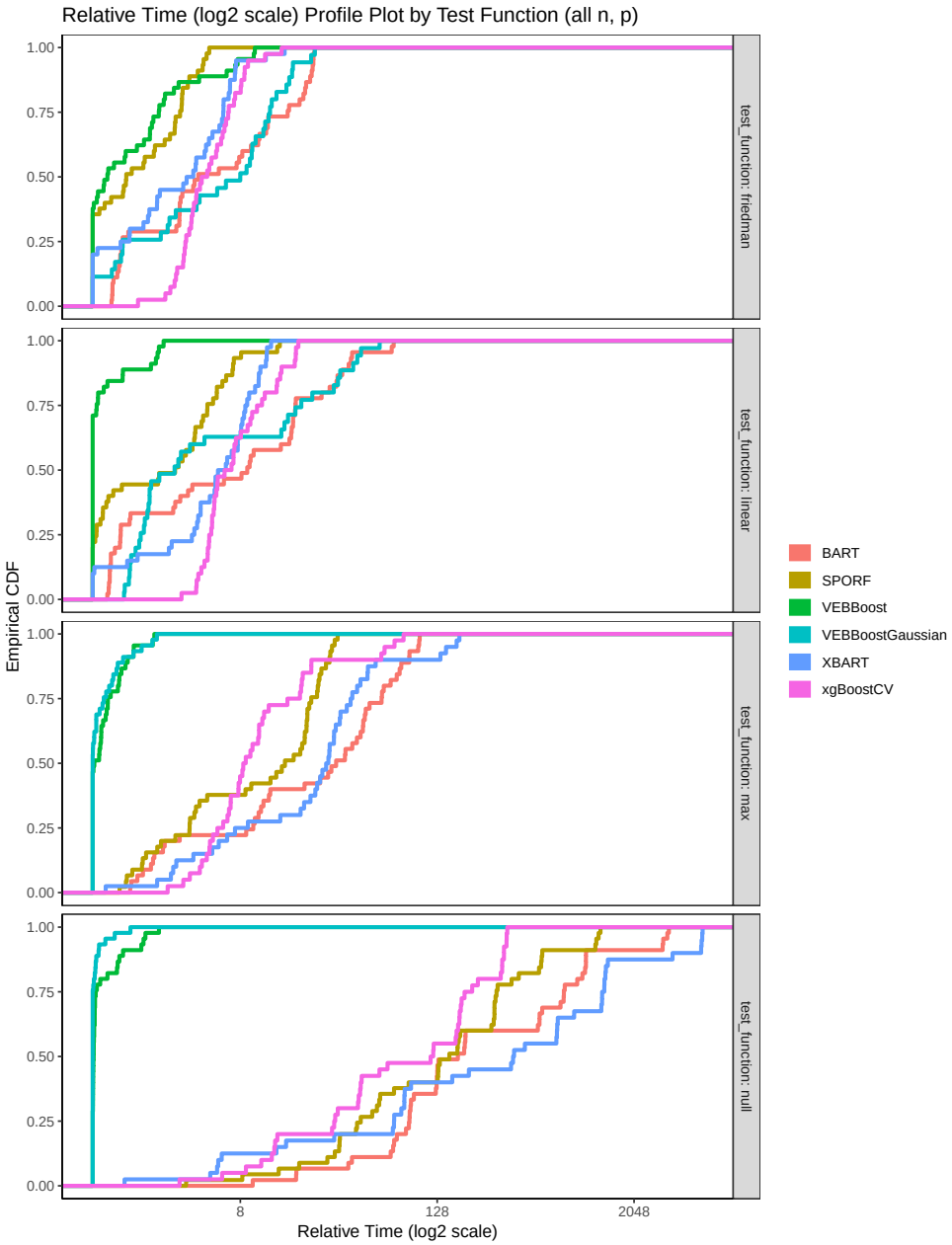

Figure 4.7: **Profile Plot of Relative Time ($\log_2$ scale)** This plot shows the empirical CDF of the relative time ($\log_2$ scale) from the logistic simulation by test function. We see that, pretty much across the board, the logistic VEB-Boost model has some of the fastest relative runtimes, and the Gaussian VEB-Boost model typically takes longer to run.

4.3.2 Real Data Examples

In this subsection, we display the results of comparing different methods on real binary and multi-class classification datasets available on the UCI machine learning repository (Dua and Graff [2017]). Specifically, we used the pre-processed datasets and test/train splits used by the original manuscript of Tomita et al. [2020], available at `https://github.com/ttomita/RandomerForest/tree/master/Data/Benchmarks`. In addition to the other models, I have included a logistic Lasso model (**"Lasso CV"**), fit with cross-validation, using the R package `glmnet`. I also note that for fitting BART models on the multi-class classification problems, we used the R package `BART`. And in the multi-class case, we ran VEB-Boost using both the Titsias bound (**"VEB-Boost Titsias"**) and the Bouchard bound (**"VEB-Boost Bouchard"**). We consider the same metrics as in the logistic simulation study:

1. Relative logloss (relative to the minimum);
2. Relative AUC (relative to the maximum);
3. Relative $(1 + MCC)/2$ (relative to the maximum).

As with the continuous response real data examples from Section 3.6.2, we added 0, 100, or 1000 null variables generated as iid $N(0, 1)$. And as with these earlier examples, some algorithms couldn't be run on these larger datasets. As a result, there is likely some bias that is introduced in the cases where we added null variables.

The results are provided below in Figures 4.8 through 4.16. When measured using the AUC and MCC metrics in both the binary and multi-class classification tasks, VEB-Boost (using the Titsias bound for the multi-class case) appears competitive in the higher-dimensional settings when we add many null variables; it suffers some when compared using logloss, suggesting an issue with the calibration of probabilities.

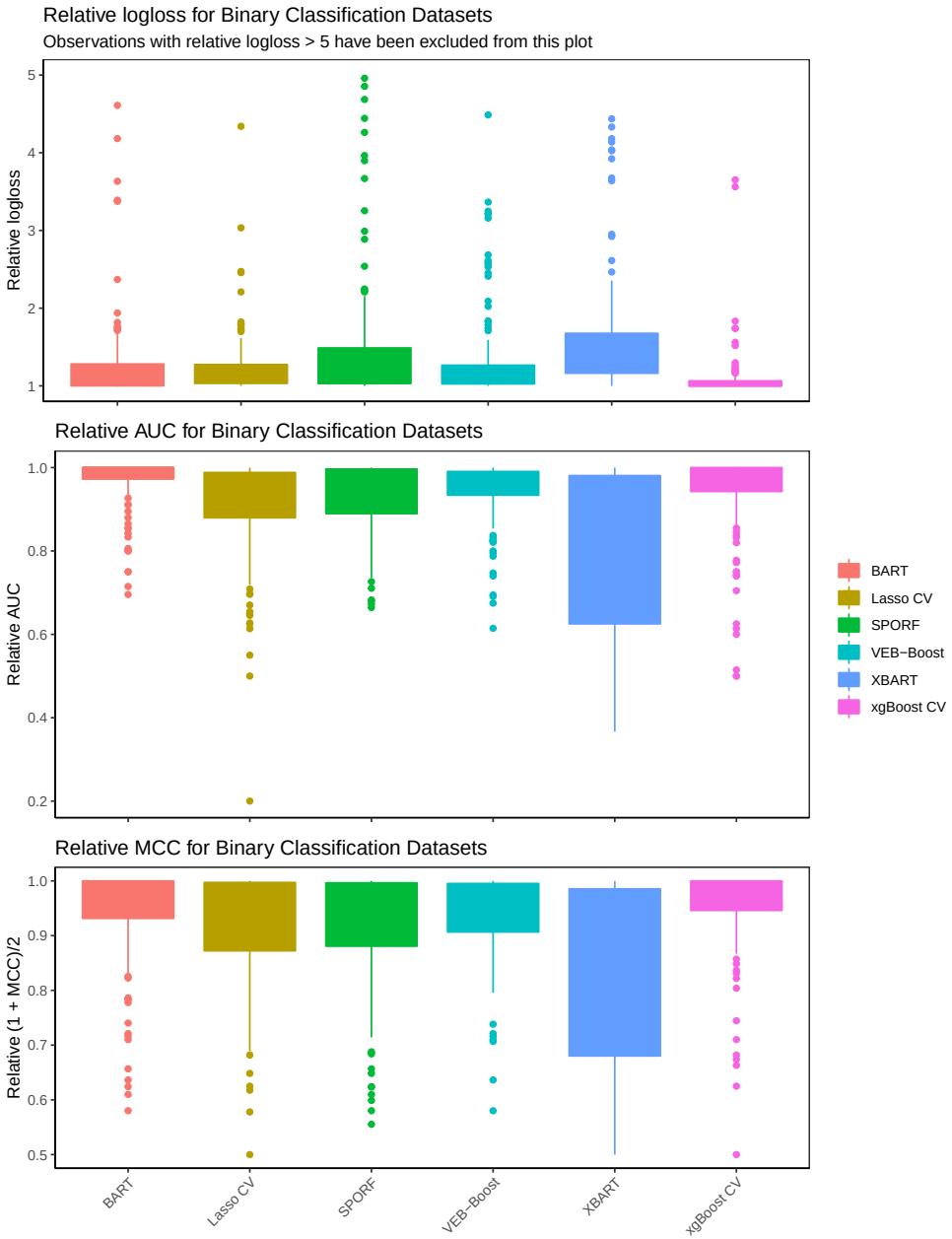

Figure 4.8: **Binary Classification Benchmarks** This plot shows the relative logloss, AUC, and $(1 + MCC)/2$ for the binary classification problems in this benchmarking study. We can see that XGBoost and BART are towards the top of the list, but VEB-Boost is not far behind.

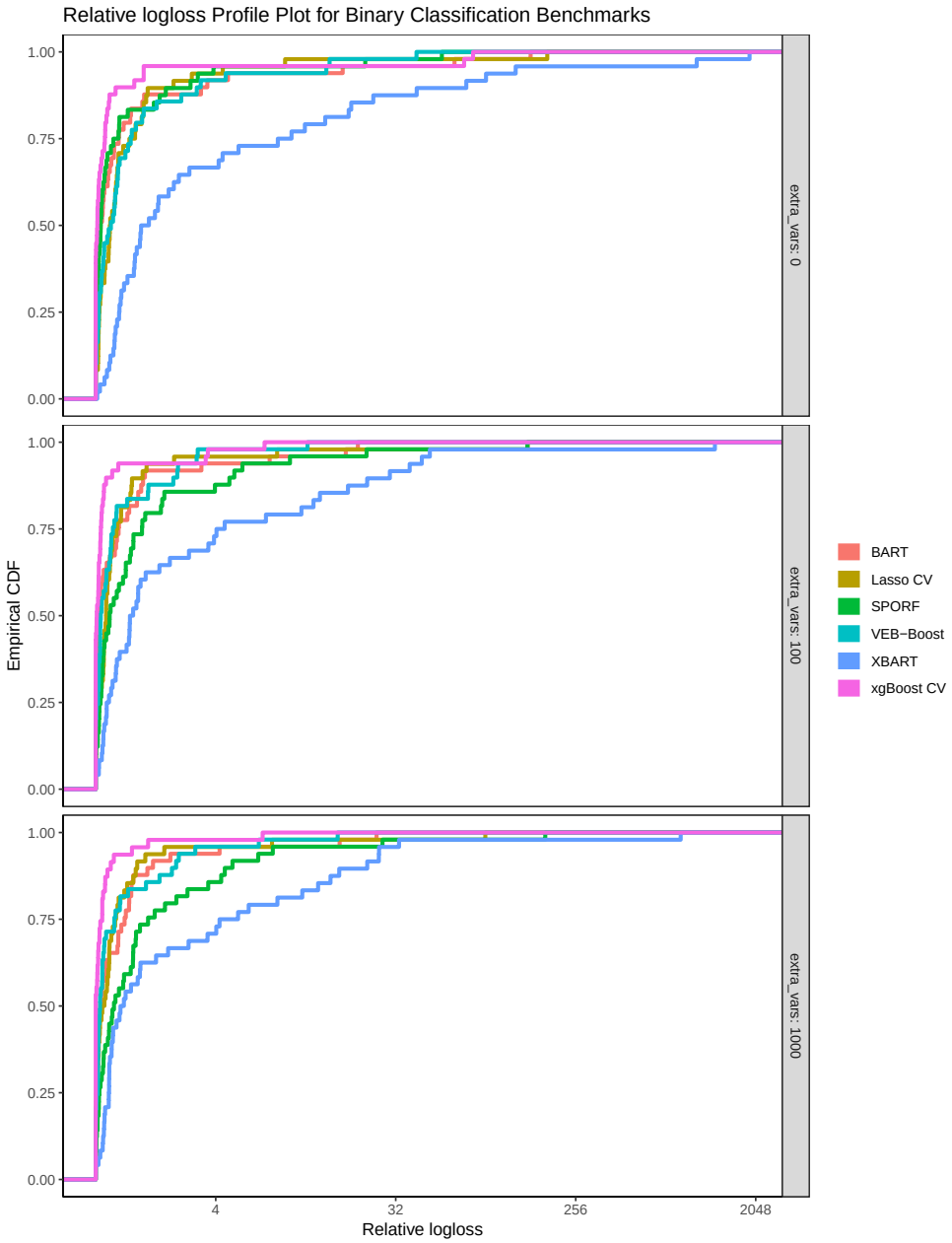

Figure 4.9: **Binary Classification logloss Profile Plot** This plot shows the logloss profile plots for the binary classification problems in the benchmarking study, broken up by how many null variables were added. We see that XGBoost appears to be at the top, with XBART performing quite poorly on these problems for some reason. We also see that VEB-Boost performs roughly on-par with Lasso; it would be interesting to see how a linear version of VEB-Boost (i.e. logistic SuSiE) would perform on these problems.

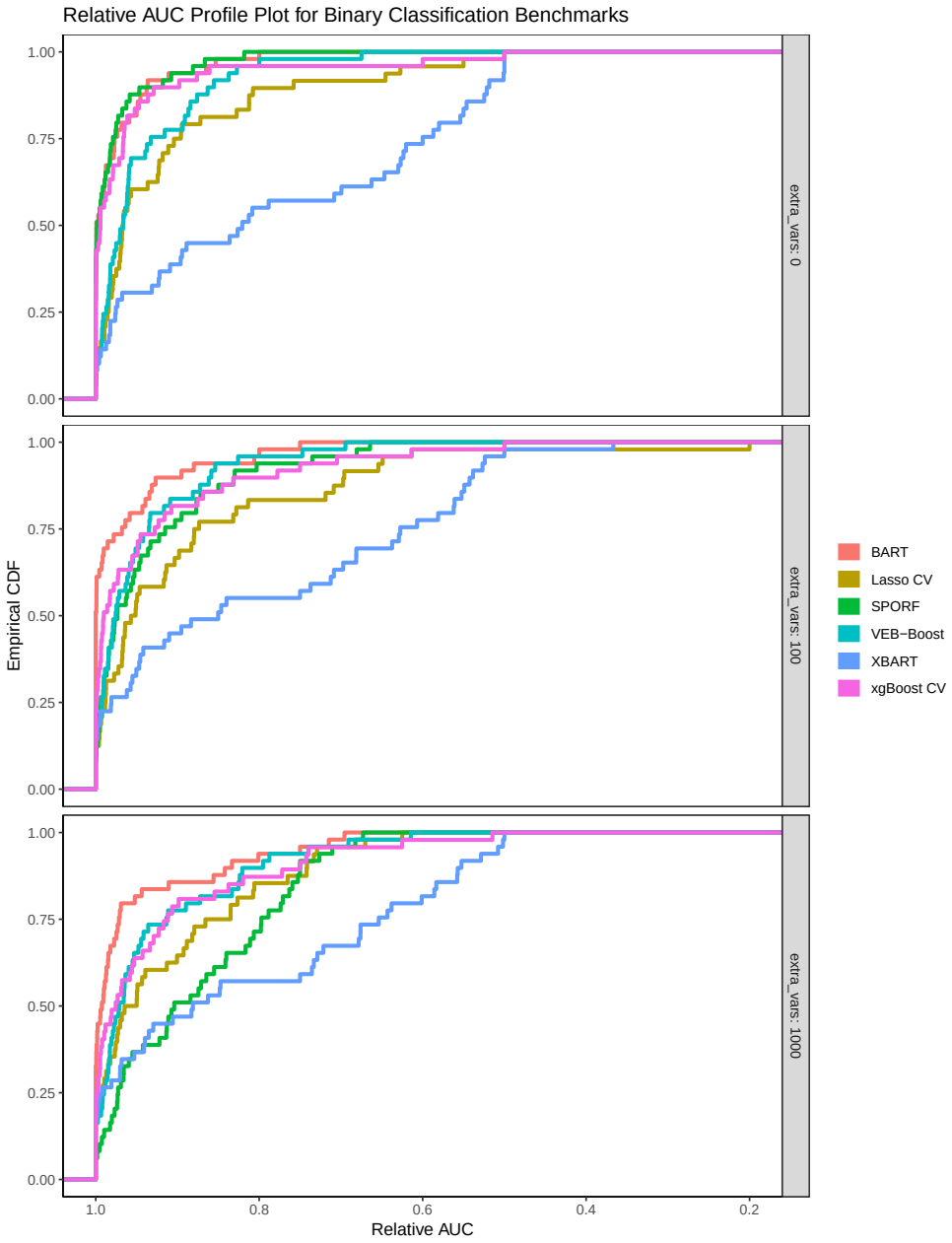

Figure 4.10: **Binary Classification AUC Profile Plot** This plot shows the AUC profile plots for the binary classification problems in the benchmarking study, broken up by how many null variables were added. With this metric, BART appears to perform the best. As opposed to the logloss metric, we see that XGBoost's relative performance degrades as we add more null variables. In the settings with many null variables, VEB-Boost is on-par with XGBoost. Also opposed to the logloss metric, we see that VEB-Boost outperforms Lasso; this suggests that VEB-Boost may not be yielding calibrated probabilities. This could be caused by the Gaussian approximation used.

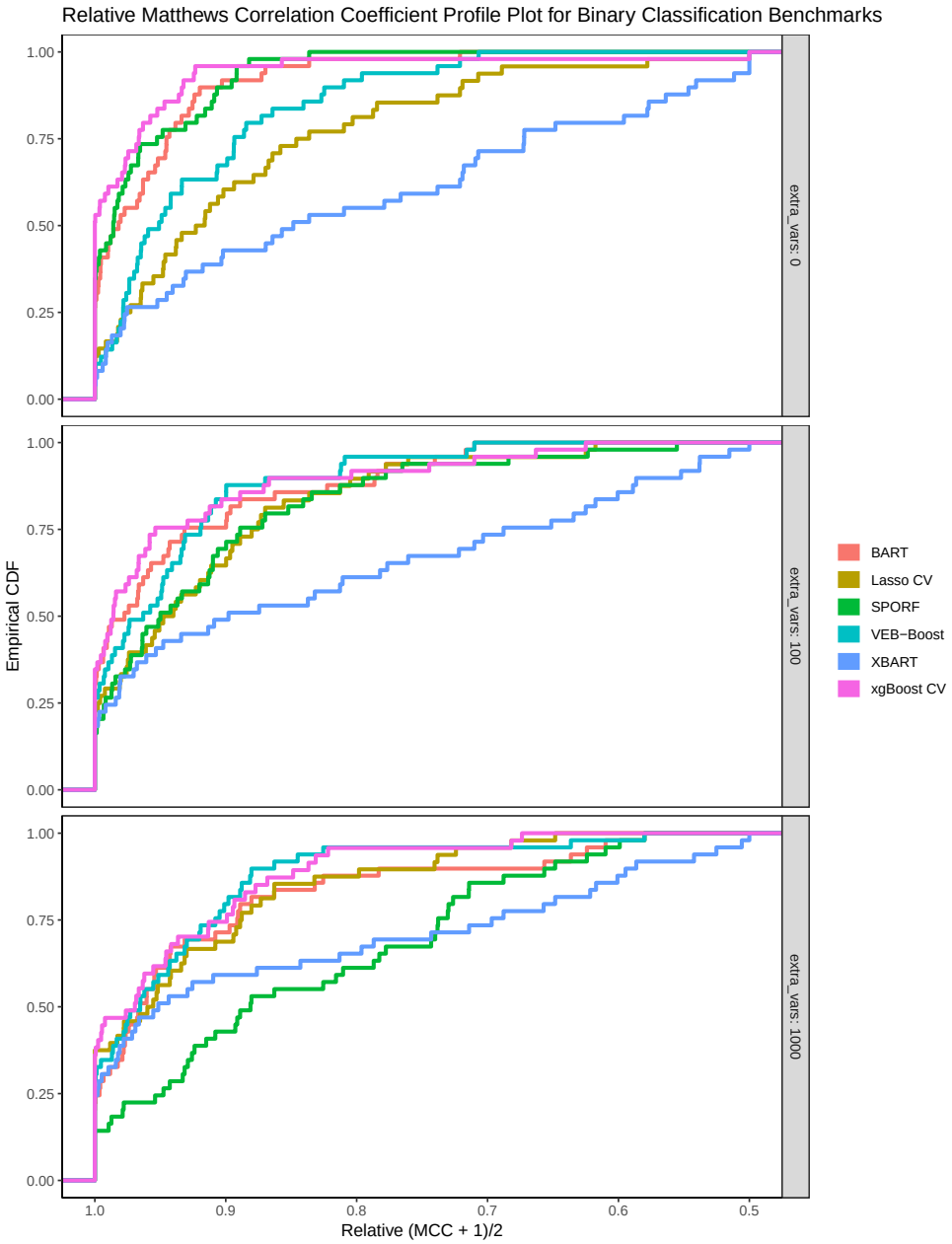

Figure 4.11: **Binary Classification MCC Profile Plot** This plot shows the MCC profile plots for the binary classification problems in the benchmarking study, broken up by how many null variables were added. We see that as null variables are added, VEB-Boost is able to become more competitive with XGBoost. We also see that VEB-Boost outperforms Lasso, as it did when compared using AUC.

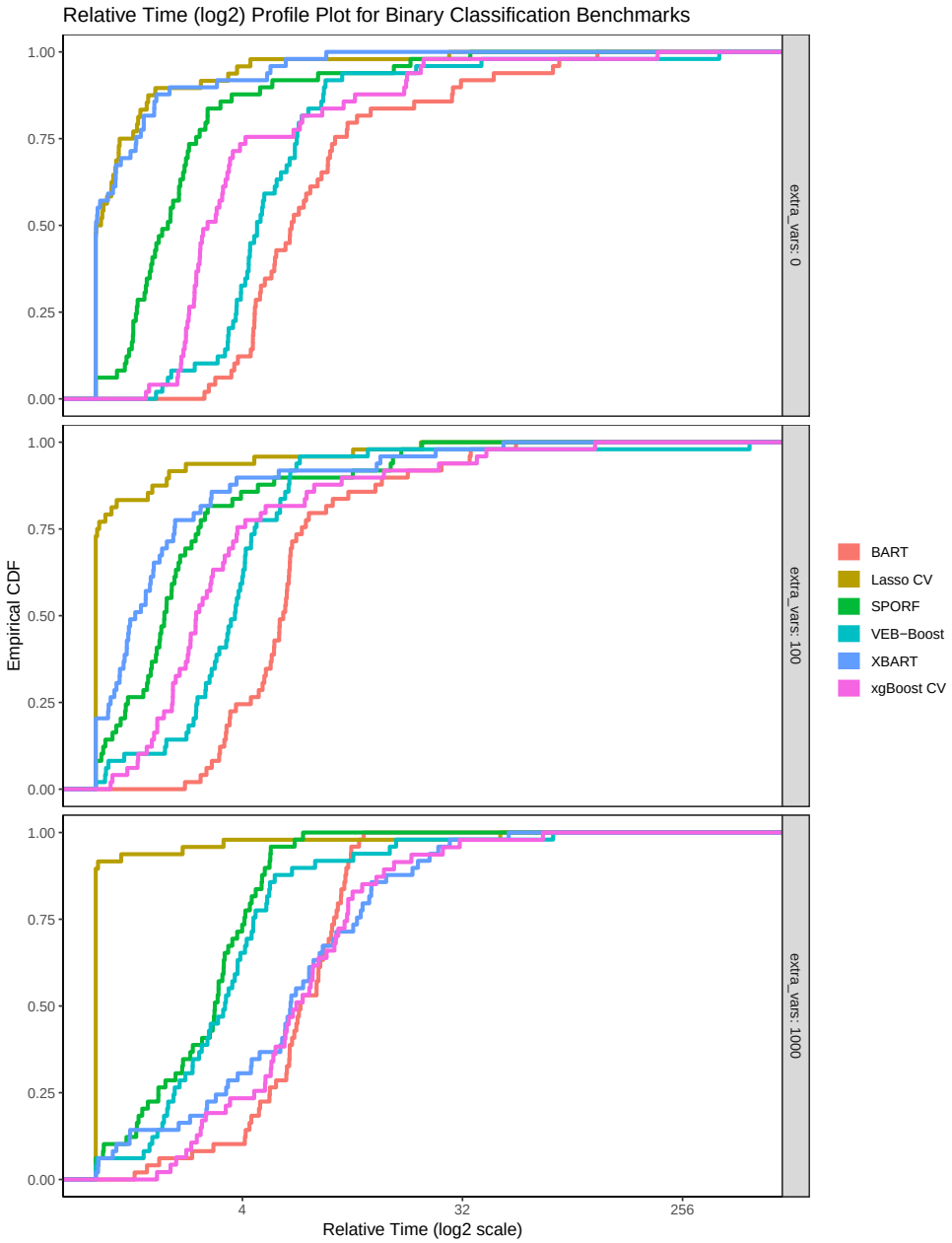

Figure 4.12: **Binary Classification Relative Time Profile Plot** This plot shows the relative time profile plots for the binary classification problems in the benchmarking study. Aside from Lasso being the fastest by a long shot (which is not too surprising), we see that VEB-Boost starts out relatively slow, but in the higher dimensional settings is able to far outperform XGBoost. Conversely, XBART is initially as fast as Lasso, but gets much slower as we add more null variables.

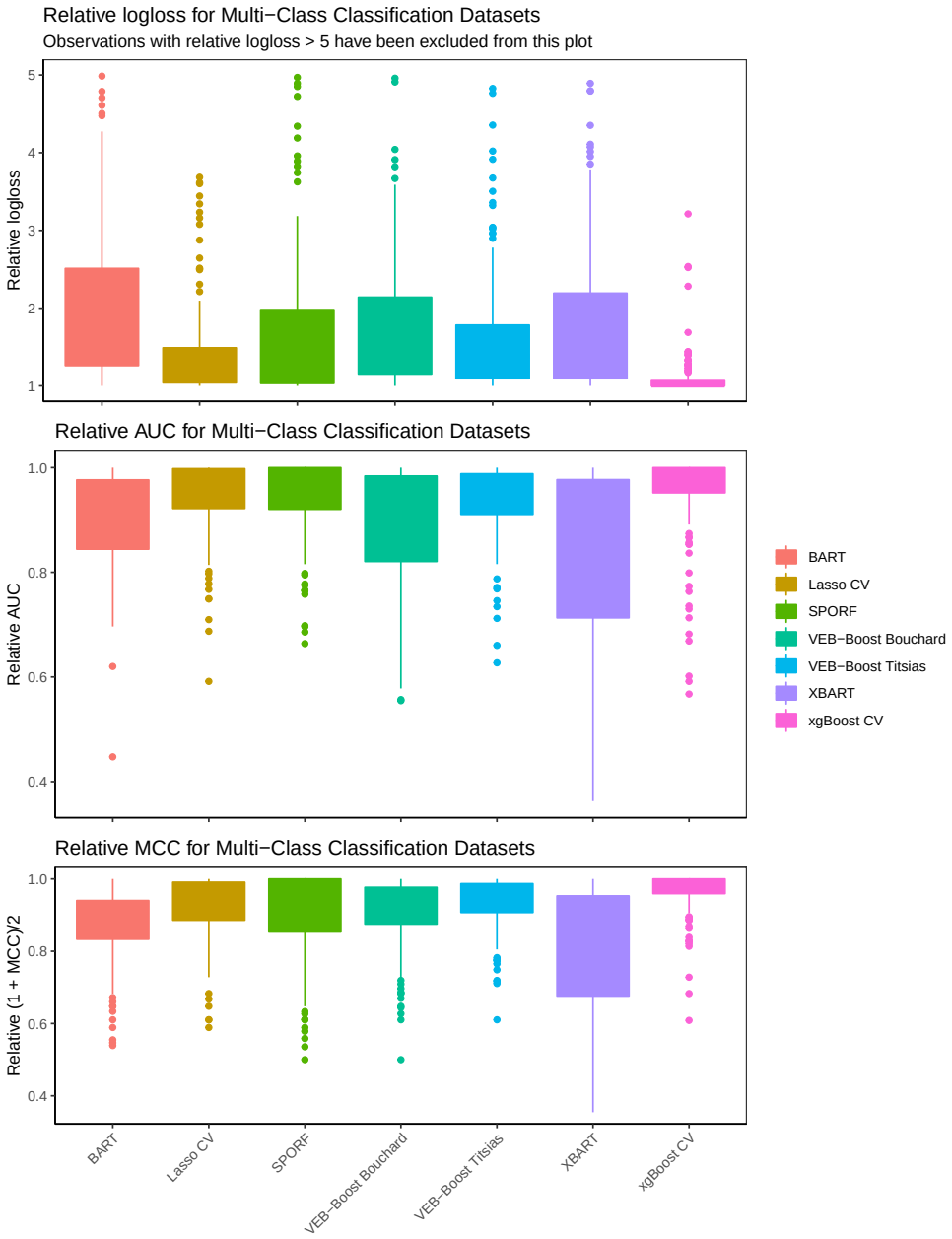

Figure 4.13: **Multi-Class Classification Benchmarks** This plot shows the relative logloss, AUC, and $(1 + MCC)/2$ for the multi-class classification problems in this benchmarking study. We see that XGBoost appears to perform quite well on all metrics. We also see that the VEB-Boost model using the Titsias bound is quite competitive when looking at the AUC and MCC metrics, and appears to outperforms the VEB-Boost model using the Bouchard bound.

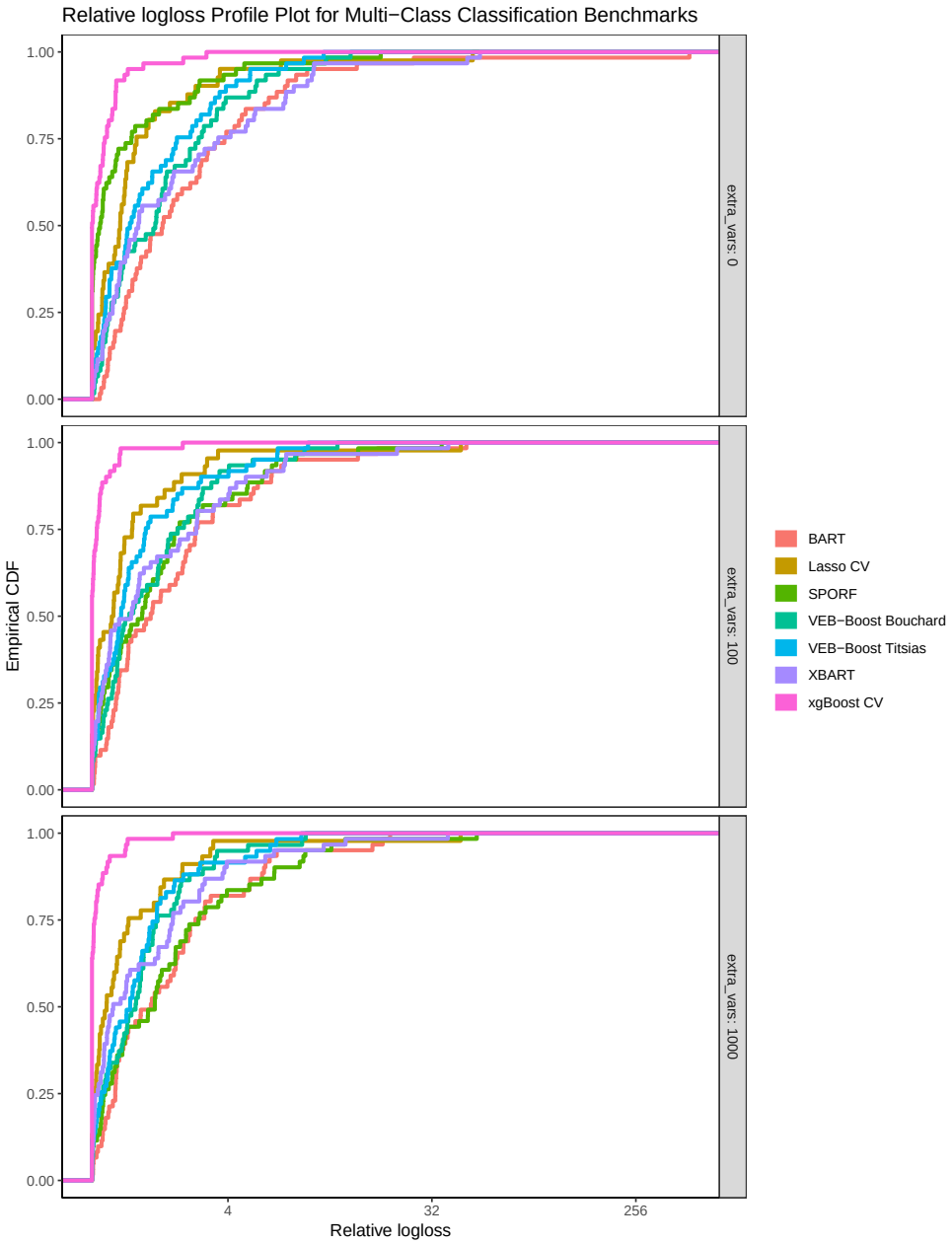

Figure 4.14: **Multi-Class Classification logloss Profile Plot** This plot shows the logloss profile plots for the multi-class classification problems in the benchmarking study, broken up by how many null variables were added. Right away, we see that XGBoost dominates the field. We also see that VEB-Boost using the Titsias bound appears a bit better than VEB-Boost using the Bouchard bound, and both perform relative better as more null variables are added. We also see that Lasso appears to perform better than both VEB-Boost models.

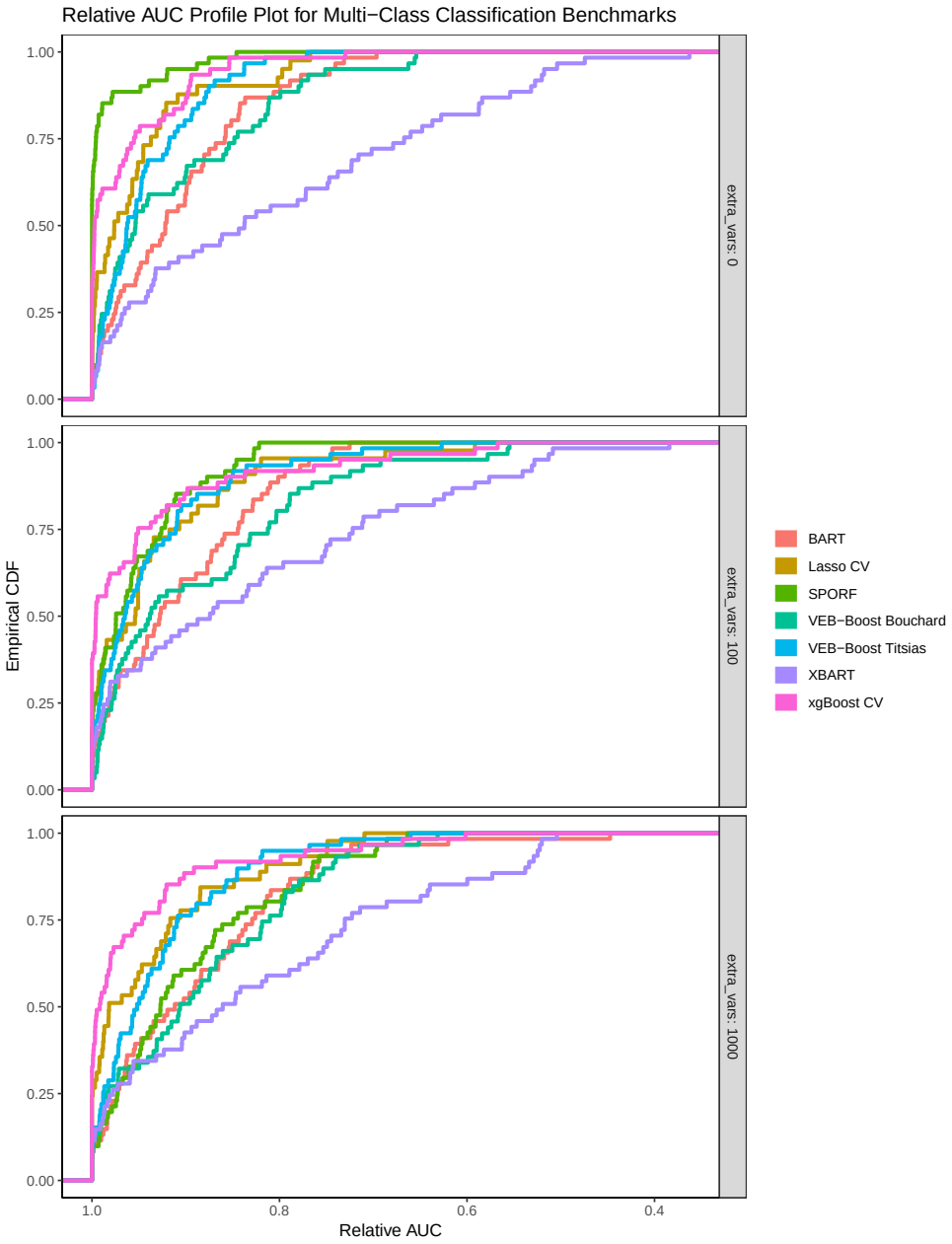

Figure 4.15: **Multi-Class Classification AUC Profile Plot** This plot shows the AUC profile plots for the multi-class classification problems in the benchmarking study, broken up by how many null variables were added. As opposed to the logloss metric, we see that VEB-Boost using the Titsias bound is roughly on-par with Lasso; this suggests that the VEB-Boost model is not providing calibrated probabilities. This could be due to the Gaussian approximations that are used.

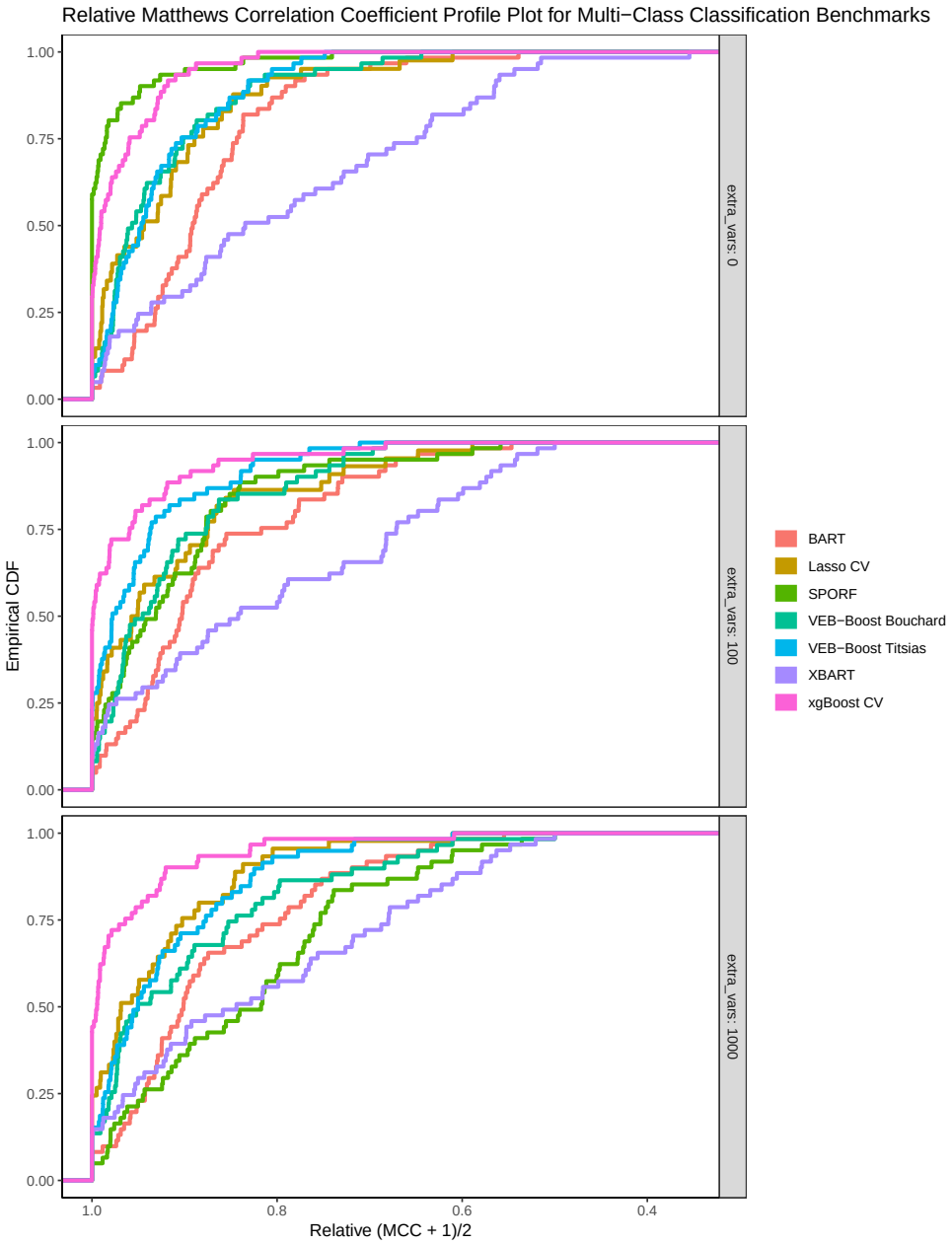

Figure 4.16: **Multi-Class Classification MCC Profile Plot** This plot shows the MCC profile plots for the multi-class classification problems in the benchmarking study, broken up by how many null variables were added. Here, we see that VEB-Boost using the Titsias bound appears a bit more competitive, especially with additional null variables. As with the AUC profile plot, we see that SPORF is quite good with no additional null variables. It would be interesting to add a cross-validation procedure to fitting SPORF in an attempt to yield better performance in the higher-dimensional settings.

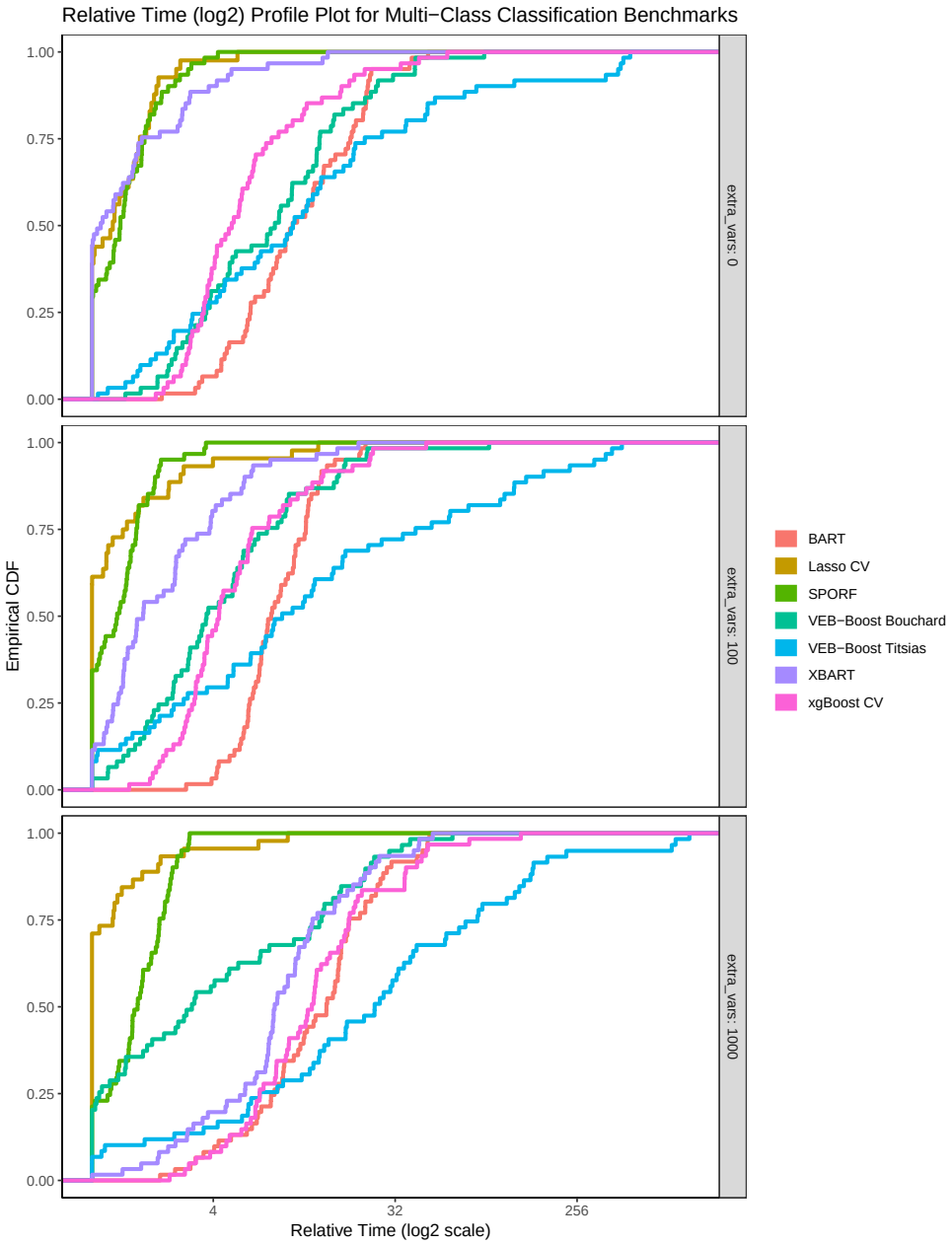

Figure 4.17: **Multi-Class Classification Relative Time Profile Plot** This plot shows the relative time profile plots for the multi-class classification problems in the benchmarking study. One interesting observation is that VEB-Boost using the Titsias bound appears to be much slower than VEB-Boost using the Bouchard bound. Since the Titsias bound appears to give better results, this could just be a consequence of the algorithm taking more time to fit the data.

4.4 Observation Weights

One final extension, which applies to both Gaussian and non-Gaussian data, is to add fixed observation weights $w_i > 0$. Observation weights can be useful in many application, such as logistic regression with imbalanced class proportions. For example, if a binary response $\mathbf{y}$ has 99% of its values as 0 and only 1% of its values as 1, we may wish to assign a relative weight to the 1 observations of 99, so that there is equal weight on the 0's and the 1's.

The most common way observation weights are incorporated into a model is by multiplying observation i's contribution to the log-likelihood by its weight w_i. It is easy to see that for Gaussian data, this corresponds to changing the variance of observation i from σ_i^2 to σ_i^2/w_i. Similarly, if we are using a Gaussian approximation with variance σ_i^2, we change the variance to σ_i^2/w_i. We must also modify the optimization step for the other parameters in the model (e.g. σ^2 if we have homoskedastic Gaussian data, or the knots in an ordinal logistic regression), but this is a simple adjustment.

There is one final decision to make when adding observation weights; whether we should normalize the weights to have an average of 1, or keep the weights un-normalized. The main consideration is that the size of the weights changes the relative importance between the expected log-likelihood term and KL-divergence term in the ELBO. If the weights have an average less than 1, the KL-divergence term matters more than it did before, so we will shrink more heavily towards the prior. And if the weights have an average greater than 1, the KL-divergence term matters less than it did before, so we will shrink less towards the prior. The VEB-Boost package allows for both, the user simply needs to specify if they want to normalize the weights (the default) or not.

4.5 Discussion

In this chapter, we have outlined our strategy for dealing with non-Gaussian data. In particular, we form a Gaussian approximation to the data, fit the Gaussian VEB-Boost model, update our approximation, and iterate to convergence. We chose our approximations so that we get a lower-bound to the ELBO. Taking this approach has some appealing properties (e.g. guaranteed to terminate), but some drawbacks as well (e.g. not possible for all data types). Our approximations make use of three bounds: the Jaakkola-Jordan bound, the Bouchard bound, and the Titsias bound. Using these bounds, we walked through the approximations to many types of models for non-Gaussian data, including:

- logistic model;
- multinomial logistic model;
- negative binomial model for count data;
- accelerated failure time model (with log-logistic noise), allowing for left-, right-, and interval-censoring;
- ordinal logistic regression;
- Bradley-Terry model (for pairwise comparison data) and Plackett-Luce model (for listwise ranking data);
- Cox proportional hazards model, allowing for right-censoring.

Of the models above, the logistic, multinomial logistic, and negative binomial models are well-known, since the aforementioned bounds were crafted with these models in mind. However, as far as I can tell, the bounds have not before been used in the context of the other models.

Not all of these models are implemented in the VEB-Boost package, since some of them require dependent Gaussians in their approximations. While the VEB-Boost framework can

easily handle these cases, it just isn't implemented yet. However, the logistic and multinomial logistic models are among those that are implemented, and due to the large number of methods that can handle these types of data, we chose them to participate in a simulation study and real-dataset benchmarking.

There are some things I would like to investigate in the future. Namely,

1. I would like to implement the dependent Gaussian case, which would allow for fitting the Bradley-Terry, Plackett-Luce, and Cox proportional hazards (using the Titsias bound) models. It would be interesting to explore how they compare with other methods, e.g. XGBoost and Lasso;

2. I would like to see how alternative quadratic bounds, such as a second-order Taylor series approximation, would perform, and see if the theoretical convergence issues are a problem or not in practice;

3. I would like to see if there is a way to incorporate piecewise quadratic bounds, which can be much more accurate (Marlin et al. [2011]). It is not clear to me how this could be done, but it might be worth looking at in more detail.

We then demonstrated the logistic model's potential in a simulation study. This study showed that logistic VEB-Boost certainly has potential. One interesting observation is that the VEB-Boost model using a Gaussian response outperformed the logistic VEB-Boost when compared using the AUC and MCC metrics, at the cost of increased running times. So if the practitioner cares more about these metrics, and less about a metric like logloss, then they should also consider fitting a Gaussian VEB-Boost model. However, just as a reminder, the simulation was set up so that the response would be fairly balanced between positive and negative instances; the most unbalanced was the case of the `max` test function, with about $\frac{2}{3}$ of the responses being positive instances. It would be interesting to see how these methods compare with more unbalanced datasets; this would be easy to accomplish simply by varying

the intercept when generating the log-odds. This could also be a way to demonstrate the usage of observations weights, as outlined in Section 4.4.

We then demonstrated the logistic and multinomial logistic models using real datasets. These further demonstrated the potential of the VEB-Boost model, especially in higher-dimensional settings. However, they also highlight the fact that these models can at times suffer when evaluated using the logloss metric as compared to AUC or MCC; this suggests that the fitted probabilities may not be calibrated. I suspect that this is a result of the Gaussian approximations used.

BIBLIOGRAPHY

Alan Agresti. *Categorical data analysis*. John Wiley & Sons, 2003.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv:1708.03731v2 [stat.ML]*, 2019.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, NY, 2006. ISBN 978-0-387-31073-2.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017. doi: 10.1080/01621459.2017.1285773. URL https://doi.org/10.1080/01621459.2017.1285773.

Guillaume Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models. *Presentation at the workshop for approximate bayesian inference in continuous/hybrid systems at NIPS-07*, 2008.

Evan A. Boyle, Yang I Li, and Jonathan K Pritchard. An expanded view of complex traits: from polygenic to omnigenic. *Cell*, 169(7):1177 – 1186, 2017. doi: 10.1016/j.cell.2017.05.038. URL https://www.cell.com/cell/fulltext/S0092-8674(17)30629-3.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. URL http://www.jstor.org/stable/2334029.

Michael Braun and Jon McAuliffe. Variational inference for large-scale models of discrete choice. *Journal of the American Statistical Association*, 105(489), 2010. doi: 10.1198/jasa.2009.tm08030. URL https://doi.org/10.1198/jasa.2009.tm08030.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://link.springer.com/article/10.1023/A:1010933404324.

N Breslow. Covariance analysis of censored survival data. *Biometrics*, 30(1):89 – 99, 1974.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. pages 129–136, 2007. URL https://dl.acm.org/doi/pdf/10.1145/1273496.1273513.

Peter Carbonetto and Matthew Stephens. Scalable variational inference for bayesian variable selection in regression, and its accuracy in genetic association studies. *Bayesian Analysis*, 7(1):73 – 108, 2012. doi: 10.1214/12-BA703. URL https://doi.org/10.1214/12-BA703.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785—794, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

Wei Cheng, Sohini Ramachandran, and Lorin Crawford. Uncertainty quantification in variable selection for genetic fine-mapping using bayesian neural networks. *iScience*, 25, 2022.

Hugh Chipman, Edward I. George, and Robert E. McCulloch. The practical implementation of bayesian model selection. *IMS Lecture Notes - Monograph Series*, 38, 2001. URL https://doi.org/10.1214/lnms/1215540964.

Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010. doi: 10.1214/09-AOAS285. URL https://projecteuclid.org/euclid.aoas/1273584455.

Rosario Delgado and Xavier-Andoni Tibau. Why cohen's kappa should be avoided as performance measure in classification. *PLOS ONE*, 14(9):1–26, 09 2019. URL https://doi.org/10.1371/journal.pone.0222916.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002. URL https://doi.org/10.1007/s101070100263.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Daniele Durante and Tommaso Rigon. Conditionally conjugate mean-field variational bayes for logistic models. *Statistical Science*, 34(3):472 – 485, 2019. doi: 10.1214/19-STS712. URL https://doi.org/10.1214/19-STS712.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(90):3133–3181, 2014. URL http://jmlr.org/papers/v15/delgado14a.html.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20:1 – 81, 2019. URL https://www.jmlr.org/papers/volume20/18-760/18-760.pdf.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. doi: 10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X".

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.

Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008. doi: 10.1214/07-AOAS148. URL https://projecteuclid.org/euclid.aoas/1223908046.

Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*. Routledge, 1990. URL https://doi.org/10.1201/9780203753781.

Jingyu He, Saar Yalov, and P. Richard Hahn. Xbart: Accelerated bayesian additive regression trees. *Proceedings of Machine Learning Research*, 89:1130–1138, 2019. URL http://proceedings.mlr.press/v89/he19a.html.

Tommi S. Jaakkola and Michael I. Jordan. A variational approach to bayesian logistic regression problems and their extensions. 1996.

Balazs Kegl and Robert Busa-Feketa. Boosting products of base classifiers. *Proceedings of the 26th International Conference on Machine Learning*, 2009. URL https://icml.cc/Conferences/2009/papers/231.pdf.

Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018. doi: 10.1080/00031305.2016.1277159. URL https://doi.org/10.1080/00031305.2016.1277159.

David G. Kleinbaum and Mitchel Klein. *Survival Analysis: A Self-Learning Text*. Springer New York, NY, 3 edition, 2012. URL https://doi.org/10.1007/978-1-4419-6646-9.

David Knowles and Tom Minka. Non-conjugate variational message passing for multinomial and binary regression. *Advances in Neural Information Processing Systems*, 24, 2011. URL https://proceedings.neurips.cc/paper/2011/file/5c936263f3428a40227908d5a3847c0b-Paper.pdf.

Brunero Liseo. Elimination of nuisance parameters with reference priors. *Biometrika*, 80(2):295–304, 1993. URL http://www.jstor.org/stable/2337200.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. pages 4768 – 4777, 2017. URL https://dl.acm.org/doi/10.5555/3295222.3295230.

Benjamin M. Marlin, Mohammad Emtiyaz Khan, and Kevin P. Murphy. Piecewise bounds for estimating bernoulli-logistic latent gaussian models. *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 633 – 640, 2011.

P. McCullagh and J.A. Nelder. *Generalized Linear Models*. 2 edition, 1983. ISBN 9780203753736. doi: https://doi.org/10.1201/9780203753736.

Thomas P Minka. Using lower bounds to approximate integrals. *Informal notes available at https://tminka.github.io/papers/minka-rem-tut.pdf*, 2001. URL https://tminka.github.io/papers/minka-rem-tut.pdf.

Malte Nalenz and Mattias Villani. Tree ensembles with rule structured horseshoe regularization. *Annals of Applied Statistics*, 12(4):2379–2408, 12 2018. doi: 10.1214/18-AOAS1157. URL https://doi.org/10.1214/18-AOAS1157.

J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. URL http://www.jstor.org/stable/2344614.

Juri Opitz and Sebastian Burst. Macro f1 and macro f1. 2019. URL http://arxiv.org/abs/1911.03347.

Stefano Patti, Elia Biganzoli, and Patrizia Boracchi. Review of the maximum likelihood functions for right censored data. a new elementary derivation. *Collection of Biostatistics Research Archive*, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.434&rep=rep1&type=pdf.

Ashley Petersen, Daniela Witten, and Noah Simon. Fused lasso additive model. *Journal of Computational and Graphical Statistics*, 25(4):1005–1025, 2016. doi: 10.1080/10618600.2015.1073155. URL https://doi.org/10.1080/10618600.2015.1073155.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975.

Nicholas G. Polson, James G. Scott, and Jesse Windle. Bayesian inference for logistic models using polya-gamma latent variables. *Journal of the American Statistical Association*, 108 (504):1339–1349, 2013. doi: 10.1080/01621459.2013.829001. URL https://doi.org/10.1080/01621459.2013.829001.

Alkes L. Price, Nick J. Patterson, Robert M. Plenge, Michael E. Weinblatt, Nancy A. Shadick, and David Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics*, 38:904 – 909, 2006. doi: https://doi.org/10.1038/ng1847.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Pradeep Ravikumar, John Lafferty, Han Liu, and Larry Wasserman. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, 2009. doi: https://doi.org/10.1111/j.1467-9868.2009.00718.x. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2009.00718.x.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. pages 1135 – 1144, 2016. URL https://doi.org/10.1145/2939672.2939778.

Dirk Schäfer and Eyke Hüllermeier. Dyad ranking using plackett–luce models based on joint feature representations. *Machine Learning*, 107:903–941, 2018. URL https://doi.org/10.1007/s10994-017-5694-9.

Matthias Seeger and Guillaume Bouchard. Fast variational bayesian inference for non-conjugate matrix factorization models. 22:1012–1018, 2012. URL https://proceedings.mlr.press/v22/seeger12.html.

Bertrand Servin and Matthew Stephens. Imputation-based analysis of association studies: Candidate regions and quantitative traits. *PLOS Genetics*, 2007. URL https://doi.org/10.1371/journal.pgen.0030114.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2021.11.011. URL https://www.sciencedirect.com/science/article/pii/S1566253521002360.

Jake Snell and Richard Zemel. Bayesian few-shot classification with one-vs-each pólya-gamma augmented gaussian processes. 2021. URL https://openreview.net/forum?id=lgNx56yZh8a.

Yaoyuan Vincent Tan and Jason Roy. Bayesian additive regression trees and the general bart model. *Statistics in Medicine*, 38(25):5048–5069, 2019. doi: https://doi.org/10.1002/sim.8347. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.8347.

Ryan J. Tibshirani. Adaptive piecewise polynomial estimation via trend filtering. *The Annals of Statistics*, 42(1):285 – 323, 2014. doi: 10.1214/13-AOS1189. URL https://doi.org/10.1214/13-AOS1189.

Michalis K. Titsias. One-vs-each approximation to softmax for scalable estimation of probabilities. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016. URL https://dl.acm.org/doi/abs/10.5555/3157382.3157563.

Tyler M. Tomita, James Browne, Cencheng Shen, Jaewon Chung, Jesse L. Patsolic, Benjamin Falk, Carey E. Priebe, Jason Yim, Randal Burns, Mauro Maggioni, and Joshua T. Vogelstein. Sparse projection oblique randomer forests. *Journal of Machine Learning Research*, 21(104):1–39, 2020. URL http://jmlr.org/papers/v21/18-664.html.

Sebastiano Vigna. Spectral ranking. 2019. URL https://arxiv.org/abs/0912.0238.

Gao Wang, Abhishek Sarkar, Peter Carbonetto, and Matthew Stephens. A simple new approach to variable selection in regression, with application to genetic fine mapping.

Journal of the Royal Statistical Society: Series B (Statistical Methodology), 82(5):1273–1300, 2020. doi: https://doi.org/10.1111/rssb.12388. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12388.

APPENDIX A

APPENDIX

A.1 Real Dataset Information

The OpenML regression datasets used in Section 3.6.2 are described in the table below.

	Task ID	Task Name	n	p	Extra Variables
1	167210	Moneyball	1232	14	0, 100, 1000
2	233211	diamonds	53940	9	0
3	233214	Santander_transaction_value	4459	4992	0, 100, 1000
4	233215	Mercedes_Benz_Greener_Manufacturing	4209	377	0, 100, 1000
5	359930	quake	2178	3	0, 100, 1000
6	359931	sensory	576	11	0, 100, 1000
7	359932	socmob	1156	5	0, 100, 1000
8	359933	space_ga	3107	6	0, 100, 1000
9	359934	tecator	240	124	0, 100, 1000
10	359935	wine_quality	6497	11	0, 100, 1000
11	359936	elevators	16599	18	0, 100, 1000
12	359937	black_friday	166821	9	0, 100
13	359938	Brazilian_houses	10692	12	0, 100
14	359939	topo_2_1	8885	266	0, 100, 1000
15	359940	yprop_4_1	8885	251	0, 100, 1000
16	359942	colleges	7063	47	0
17	359944	abalone	4177	8	0, 100, 1000
18	359945	us_crime	1994	127	0, 100, 1000
19	359946	pol	15000	48	0
20	359948	SAT11-HAND-runtime-regression	4440	120	0, 100, 1000
21	359949	house_sales	21613	22	0, 100
22	359950	boston	506	13	0, 100, 1000
23	359951	house_prices_nominal	1460	80	0, 100, 1000
24	359952	house_16H	22784	16	0, 100
25	360945	MIP-2016-regression	1090	147	0, 100, 1000

Table A.1: **List of OpenML Regression Datasets** This table provides summary information about the benchmark datasets used in Section 3.6.2. It lists the task ID and task name used by OpenML, the sample size n, the number of predictor variables p, and the number of extra variables we were able to include in our analysis.

The pre-processing for each dataset was minimal. The steps were:

1. Remove observations with any missing data;

2. If $\log(\mathbf{y})$ appeared more Gaussian than $\mathbf{y}$, transform $\mathbf{y} \mapsto \log(\mathbf{y})$. The normality was gauged by the r^2 value of the regression line fit to the normal QQ-plot of either $\mathbf{y}$ or $\log(\mathbf{y})$;

3. Use one-hot encoding for all categorical predictors;

4. Optionally add 0, 100, or 1000 columns of iid $N(0, 1)$ noise;

5. Shuffle the rows of the response and the design matrix, and shuffle the columns of the design matrix;

6. Partition the data into 5 disjoint folds to be used to calculate the cross-validated RMSE.

A.2 Supplemental Figures

In this section, we include supplemental figures from the simulation studies given in Sections 3.6.1 and 4.3.1.

A.2.1 VEB-Boost Simulation Supplemental Figures

This subsection contains supplemental figures from the simulation study in Section 3.6.1. It contains boxplots showing the relative RMSE and runtimes of the max, linear, and null test functions.

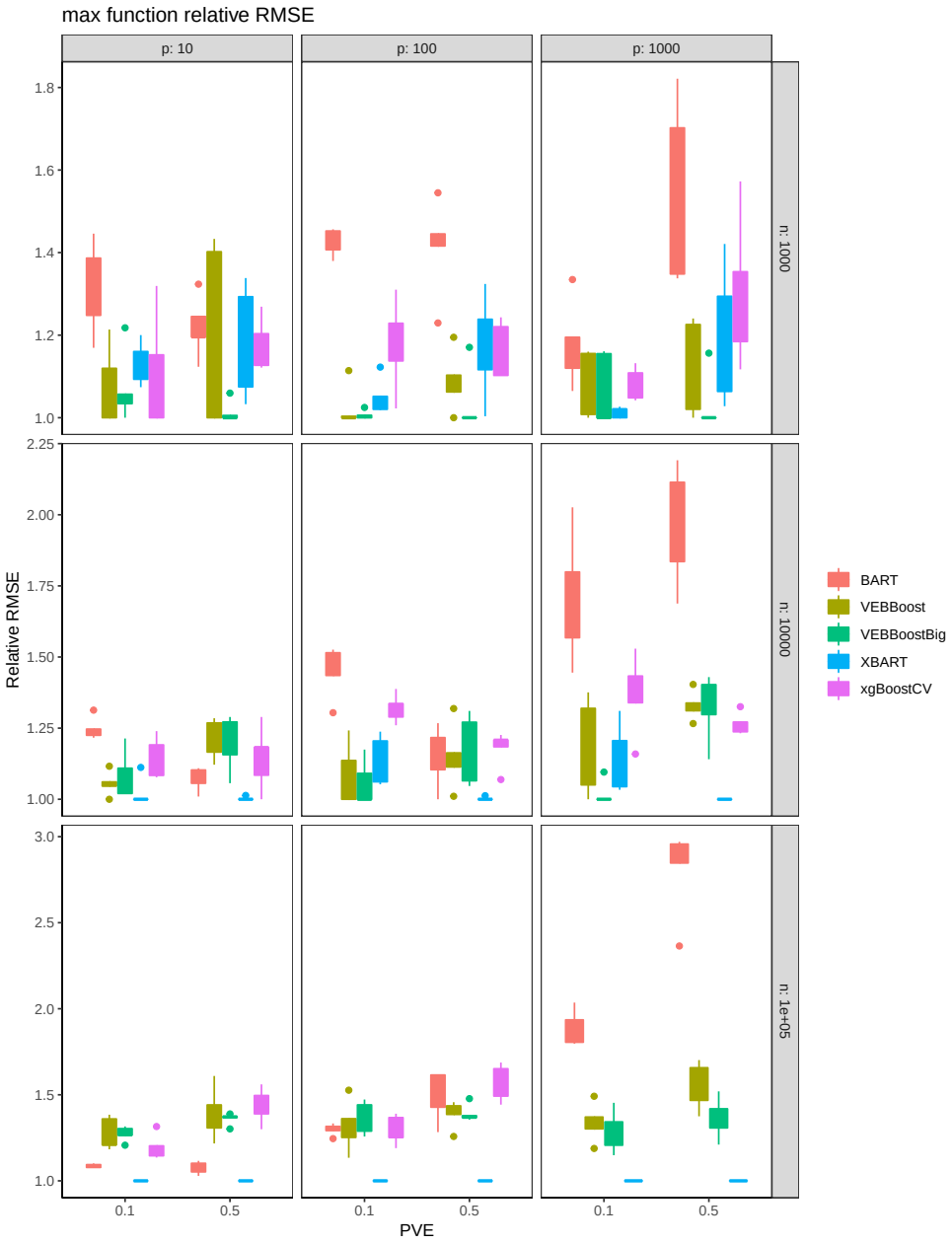

Figure A.1: **Max Function Relative RMSE** This plot shows the relative RMSE of the methods tested using the max test function. We see that XBART appears to be the winner here pretty much across the board, with the only exception being in the small sample-size setting (the top row), where the VEB-Boost method with a large starting learner comes out on top. Even though VEB-Boost isn't the winner, we can see that it's rarely much worse than XBART.

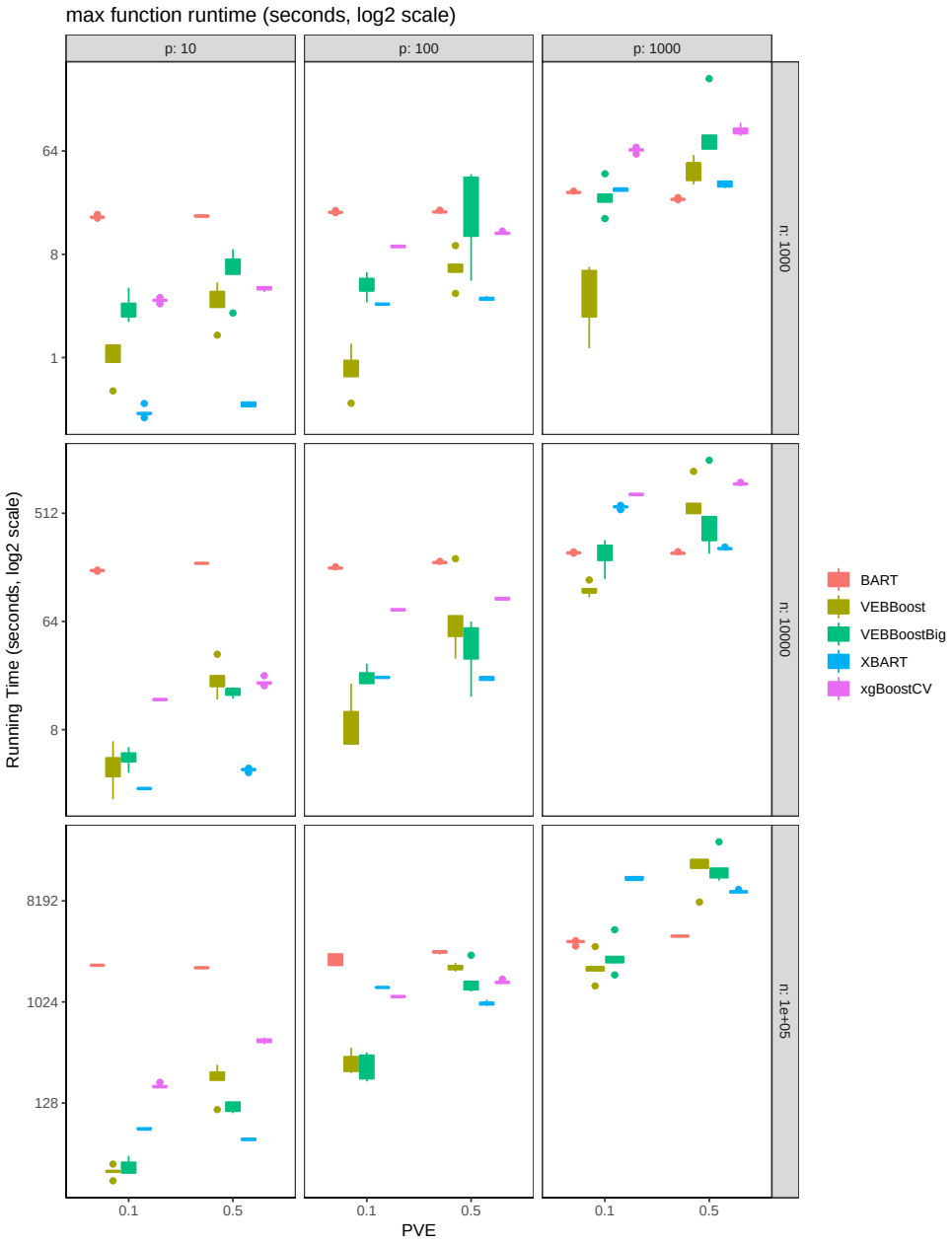

Figure A.2: **Max Function Running Time in Seconds, $\log_2$ scale** This plot shows the running time of the methods tested using the max test function. Just as in the case of Friedman's test function, VEB-Boost appears to be quite competitive, particularly in the higher noise setting. We still see similar trends for VEB-Boost in terms of longer run-times for stronger signals, and more variable run-times overall.

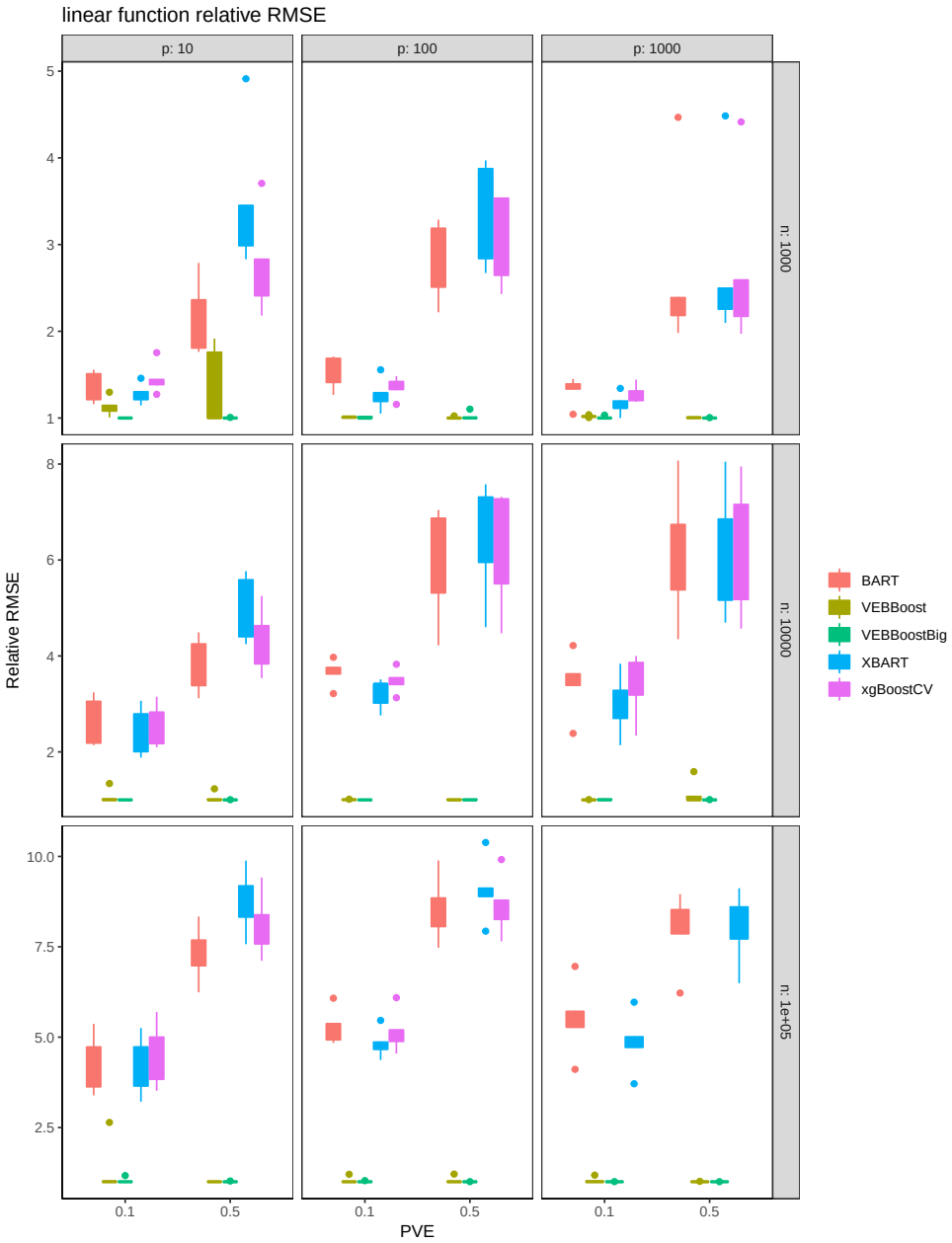

Figure A.3: **Linear Function Relative RMSE** This plot shows the relative RMSE of the methods tested using the linear test function. It is clear that VEB-Boost outperforms the other methods. This is likely due to the inclusion of the linear terms in the SER. In contrast to the Friedman and max settings, XBART performs quite poorly here.

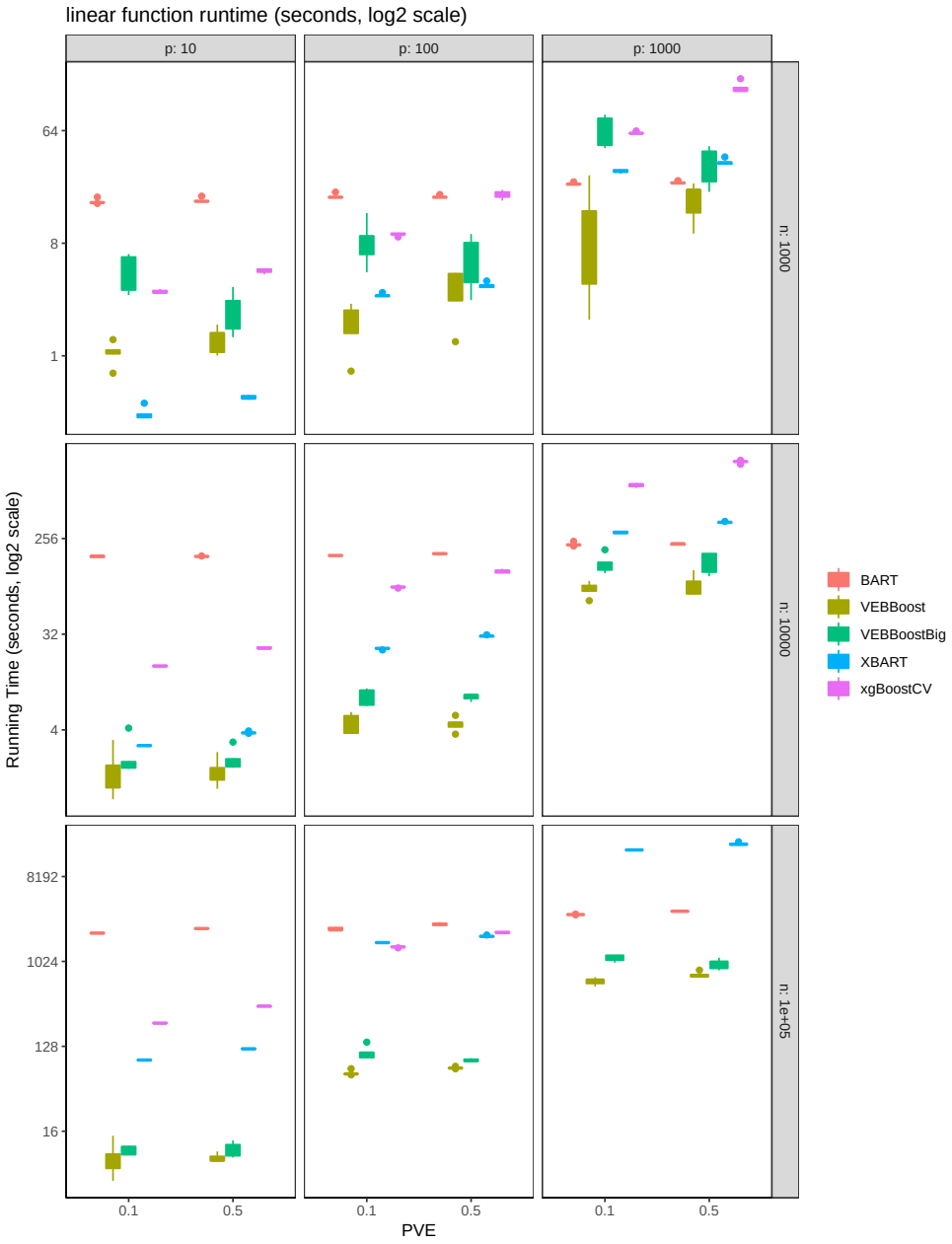

Figure A.4: **Linear Function Running Time in Seconds, $\log_2$ scale** This plot shows the running time of the methods tested using the linear test function. We see that VEB-Boost is the clear winner. We also see that on an absolute scale, VEB-Boost is able to run much faster than it did with the other test functions. This is due to the simplicity of the fit, and how few learners are needed to adequately explain the signal due to the inclusion of the linear terms in the SER.

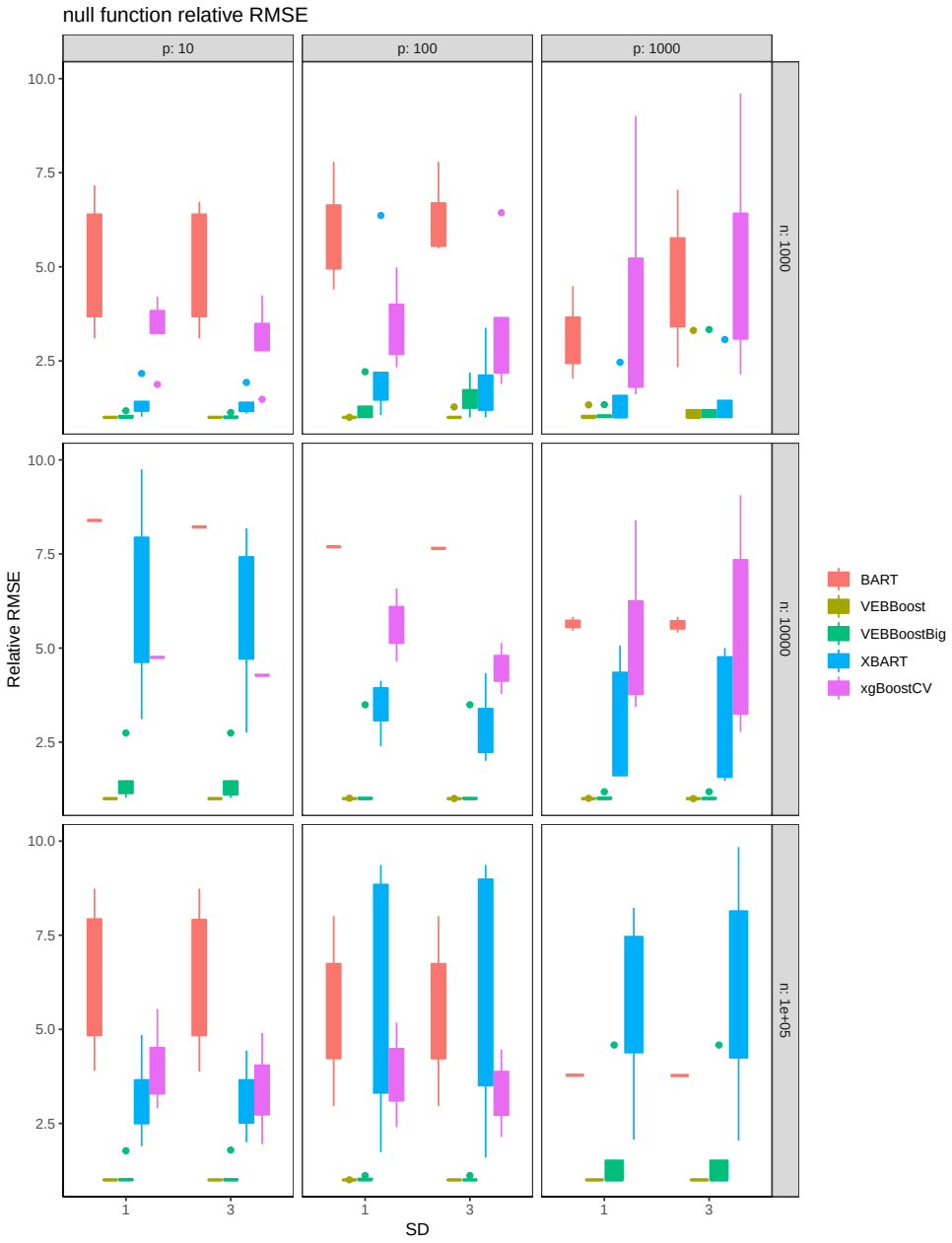

Figure A.5: **Null Function Relative RMSE** This plot shows the relative RMSE of the methods tested using the null test function. As with the linear case, VEB-Boost is the clear winner. This is likely at least partly due to the fact that VEB-Boost starts with a single weak learner initialized to the sample mean. However, we see that the VEB-Boost method starting with a larger learner still performs quite well despite its more complicated initial structure. But while the structure is more complicated, it's still initialized to the sample mean, which likely helps in the null setting.

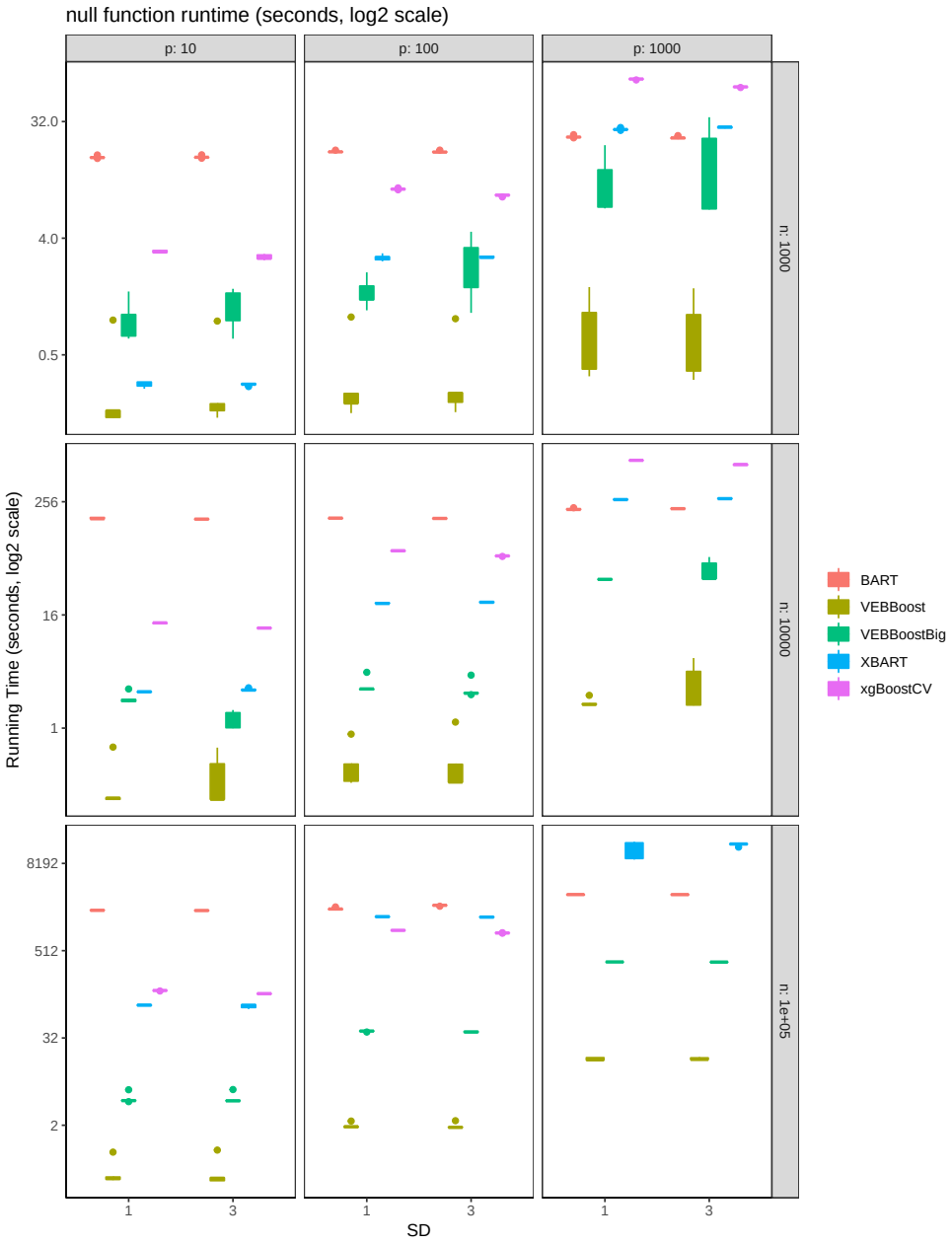

Figure A.6: **Null Function Running Time in Seconds, $\log_2$ scale** This plot shows the running time of the methods tested using the null test function. Again, VEB-Boost comes out on top. This is the benefit of the empirical Bayes aspect of VEB-Boost; instead of being forced to run for a fixed number of iterations, it can learn that there is no more signal to fit and terminate.

A.2.2 Logistic Simulation Supplemental Figures

This subsection contains supplemental figures from the logistic simulation study given in 4.3.1. It contains boxplots showing the relative logloss, AUC, MCC, and running times for the max, linear, and null test functions.

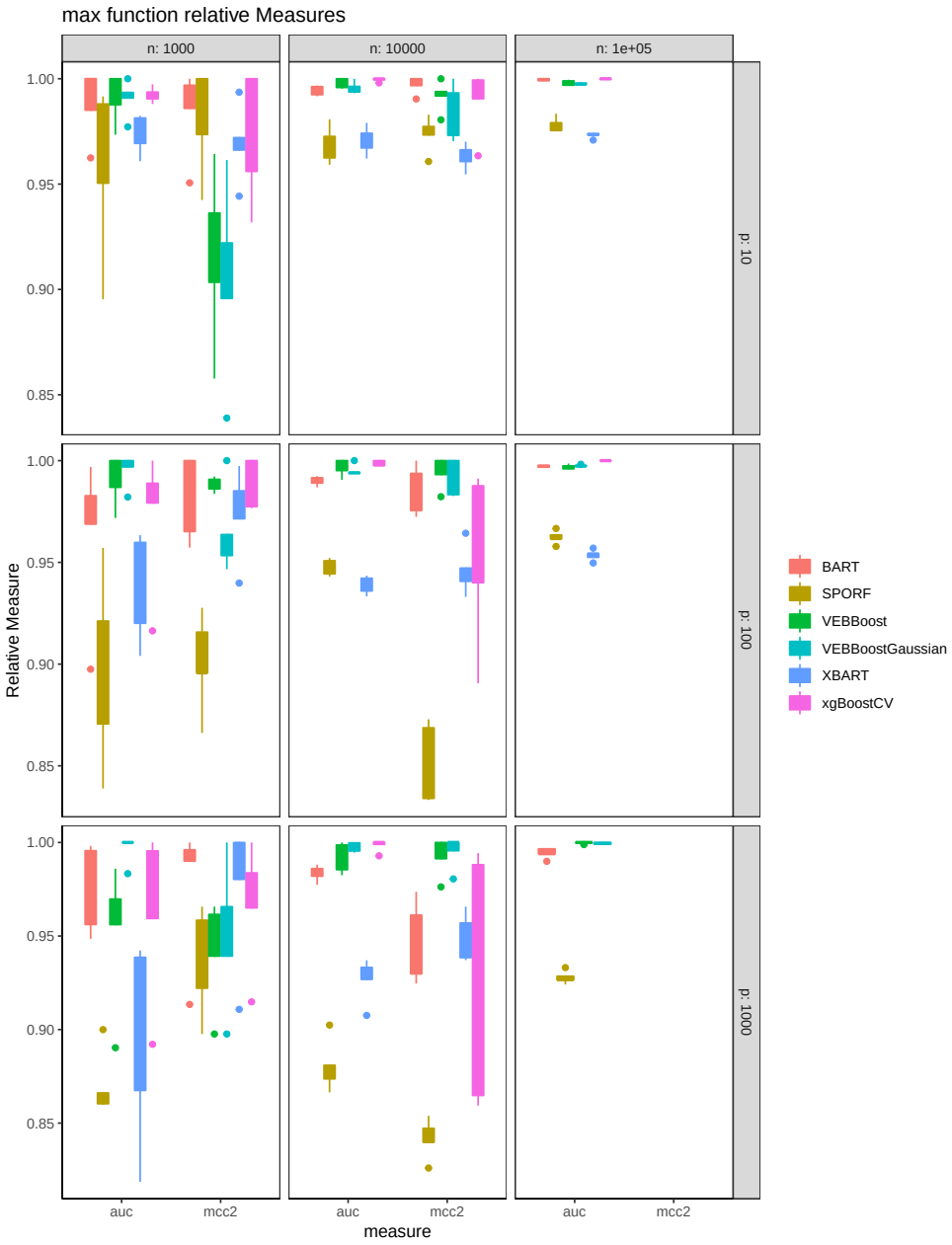

Figure A.7: **Max Function Relative AUC and MCC** This plot shows the relative AUC and $(1 + MCC)/2$ for the logistic model using the max test function (higher is better). XGBoost appears to have issues in some cases, especially when evaluated using the MCC metric. VEB-Boost also appears to perform quite well here, as long as the sample size isn't too small.

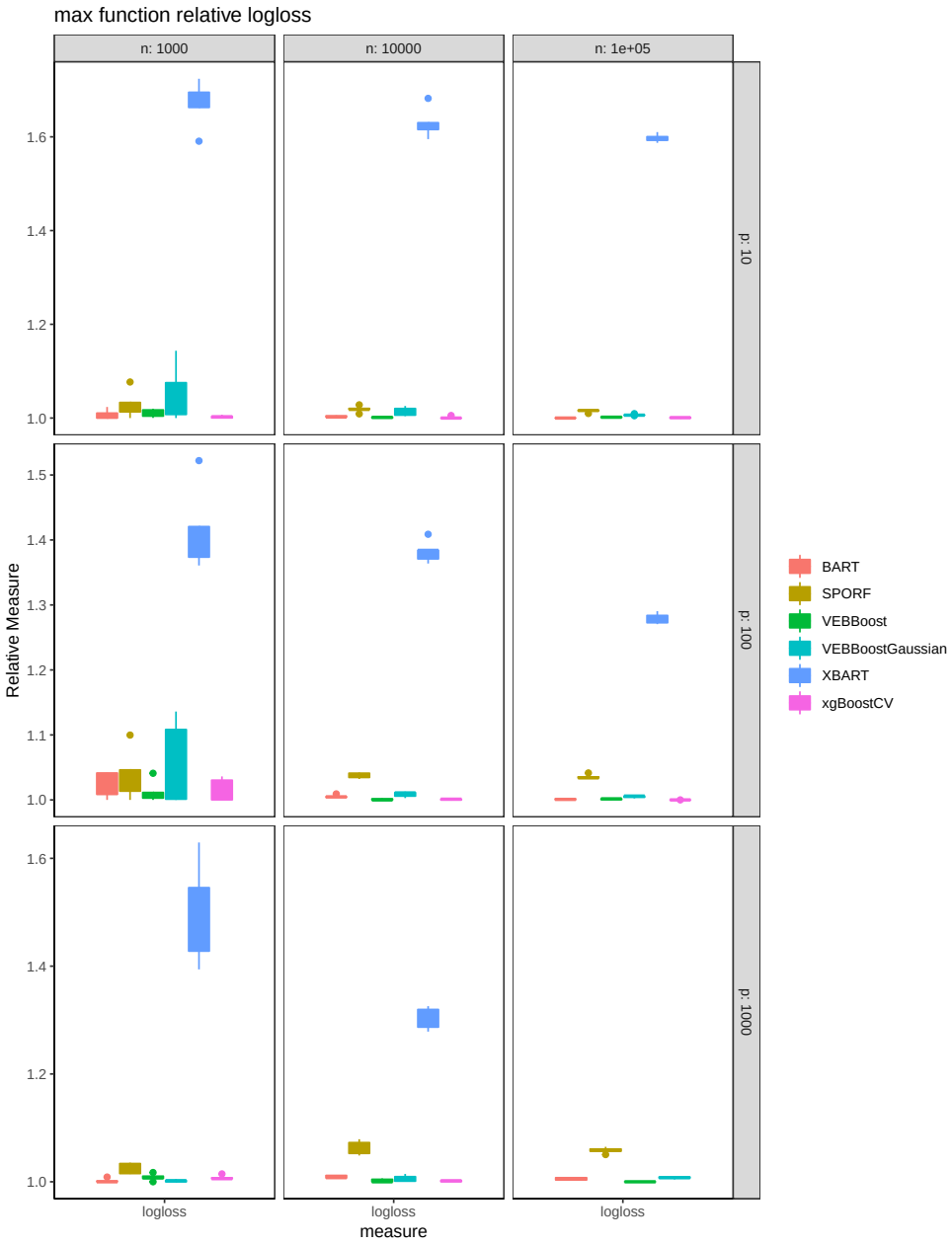

Figure A.8: **Max Function Relative logloss** This plot shows the relative logloss for the logistic model using the max test function (lower is better). With the exception of XBART, all other methods appear competitive with each other.

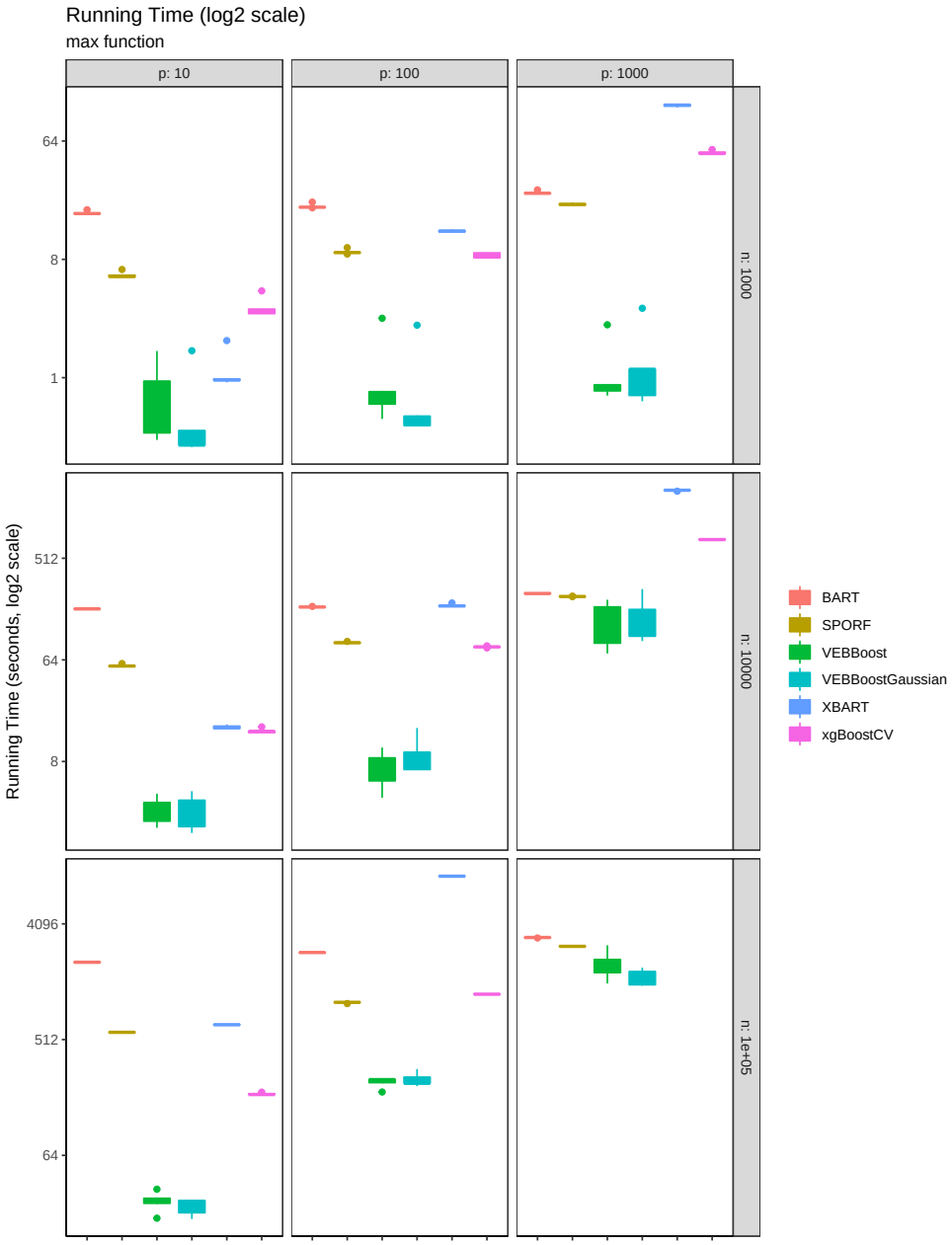

Figure A.9: **Max Function Running Time ($\log_2$ scale)** This plot shows the running time for the logistic model using the max test function (lower is better). We see that the VEB-Boost methods are typically the fastest, sometimes by a wide margin.

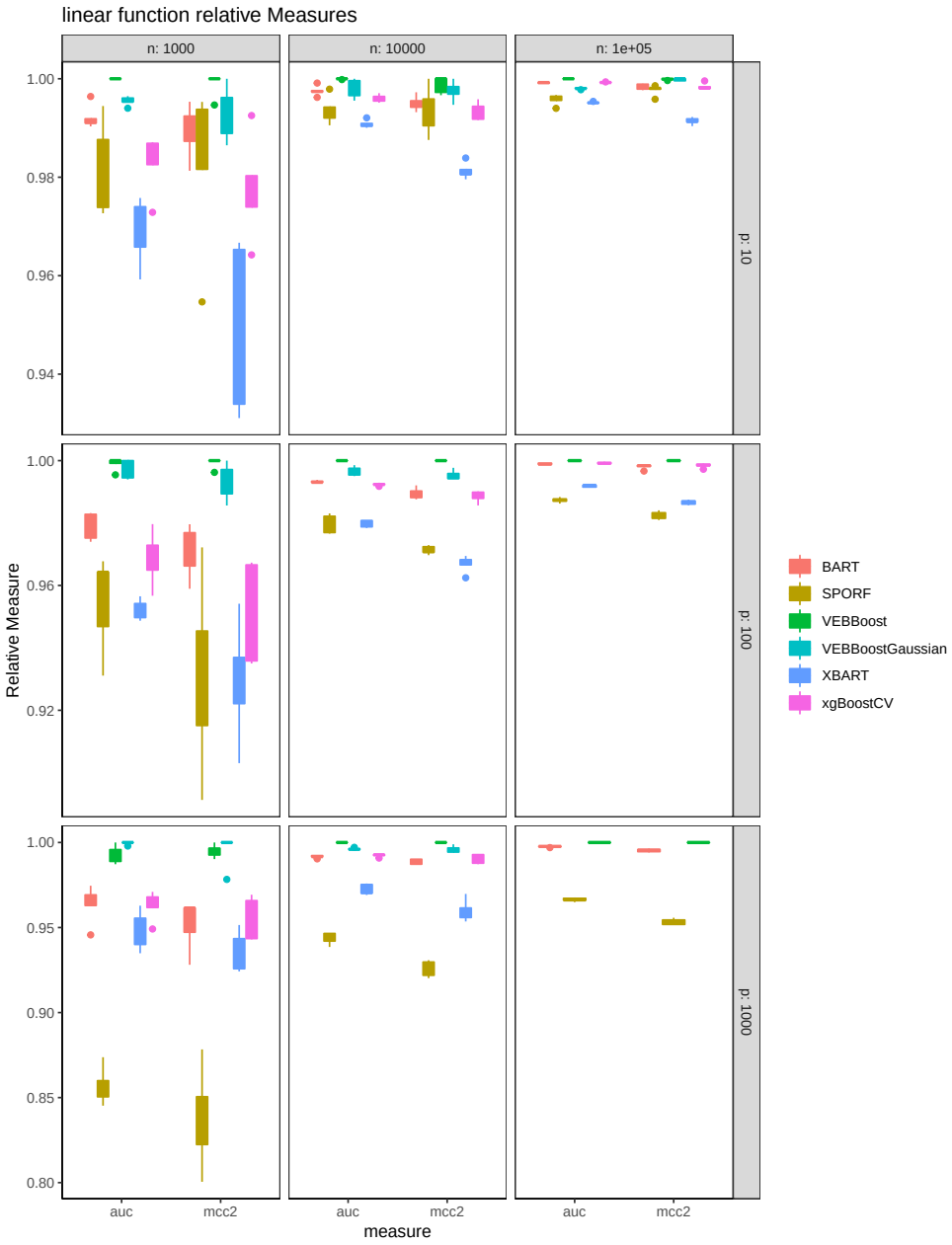

Figure A.10: **Linear Function Relative AUC and MCC** This plot shows the relative AUC and $(1 + MCC)/2$ for the logistic model using the linear test function (higher is better). Happily, VEB-Boost still outperforms the other methods, as it did in the Gaussian simulations with this test function.

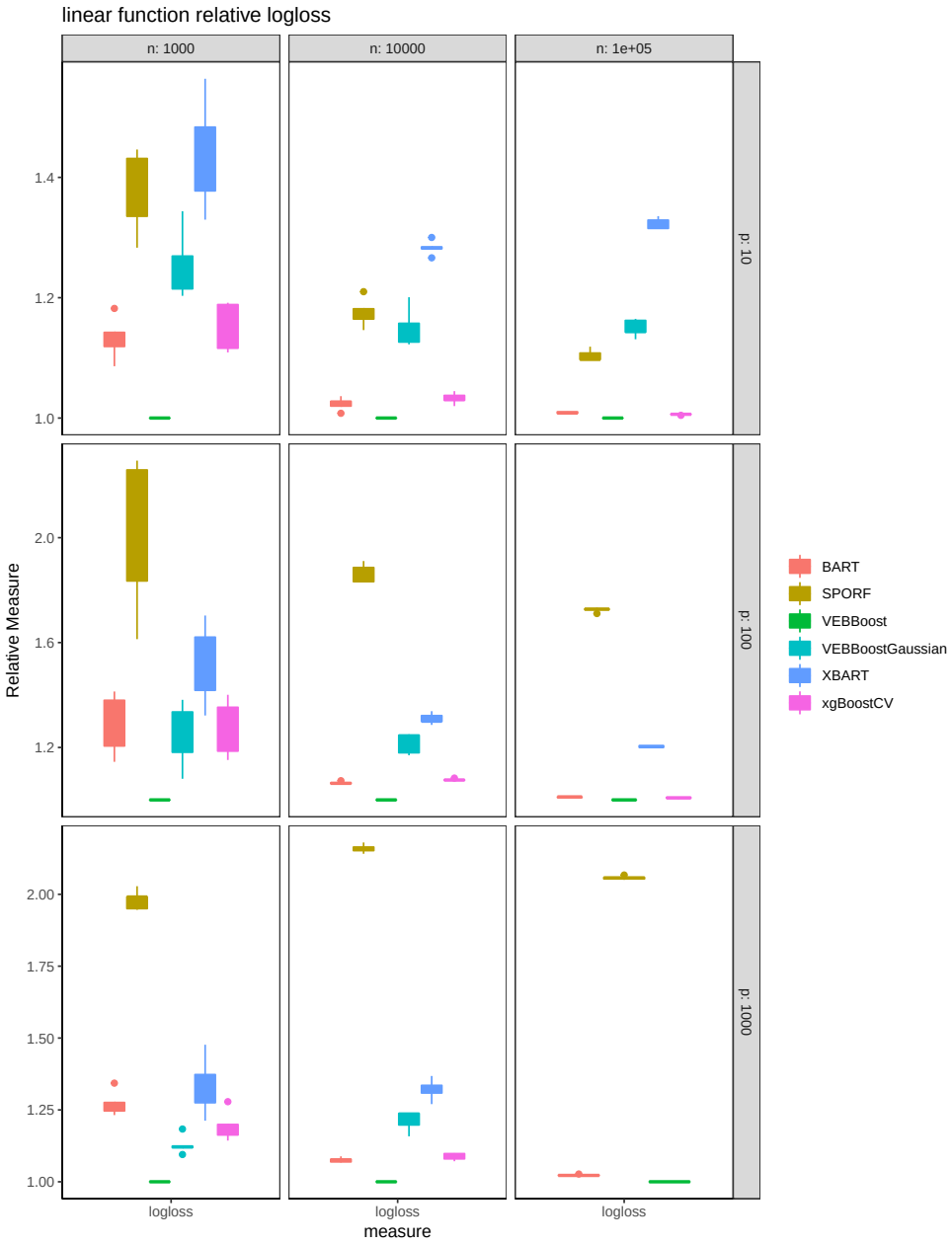

Figure A.11: **Linear Function Relative logloss** This plot shows the relative logloss for the logistic model using the linear test function (lower is better). VEB-Boost still outperforms the other methods. And we see that the Gaussian VEB-Boost model performs worse than the logistic VEB-Boost model.

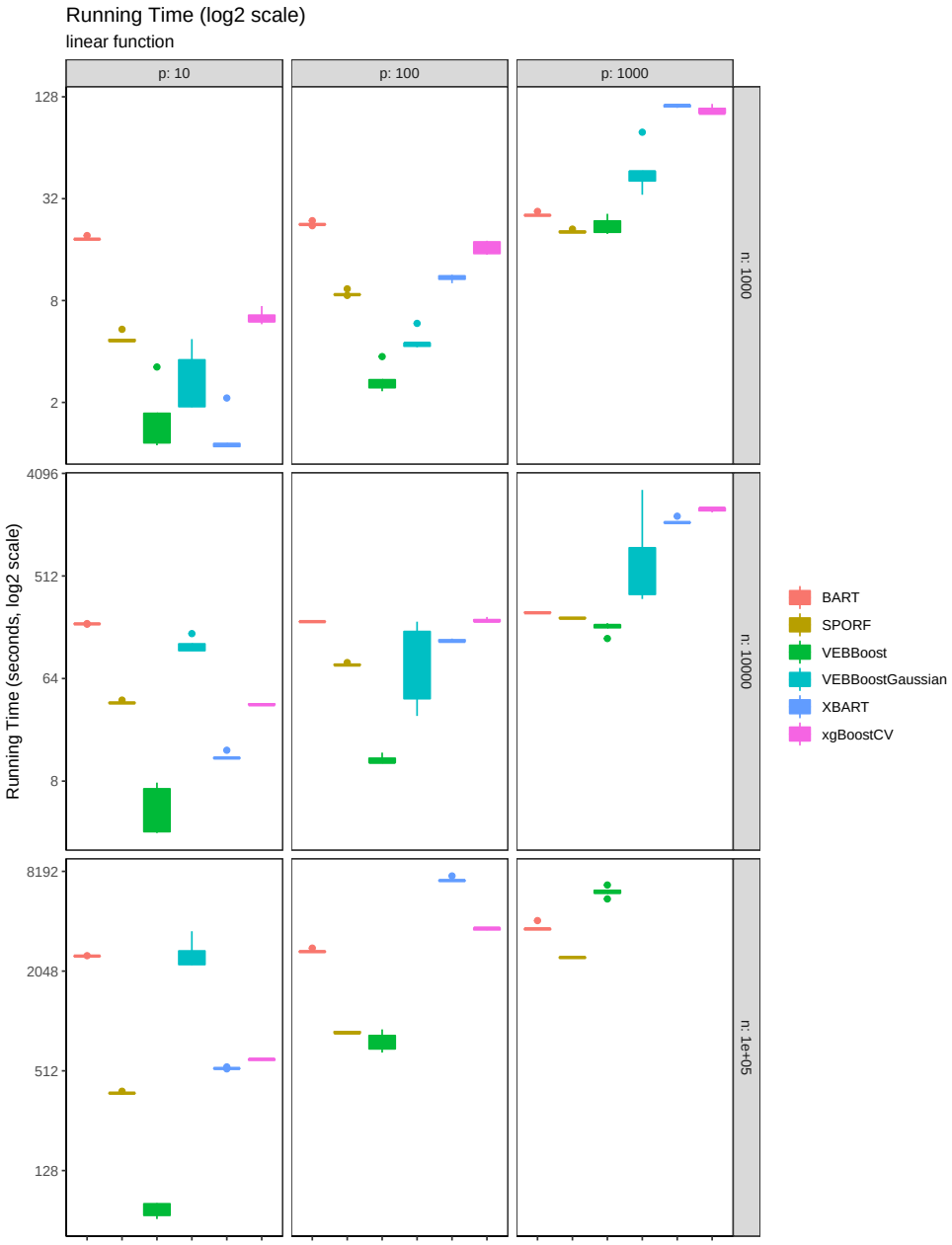

Figure A.12: **Linear Function Running Time ($\log_2$ scale)** This plot shows the running time for the logistic model using the linear test function (lower is better). We see that VEB-Boost is the fastest, and the Gaussian VEB-Boost model ends up being quite slow.

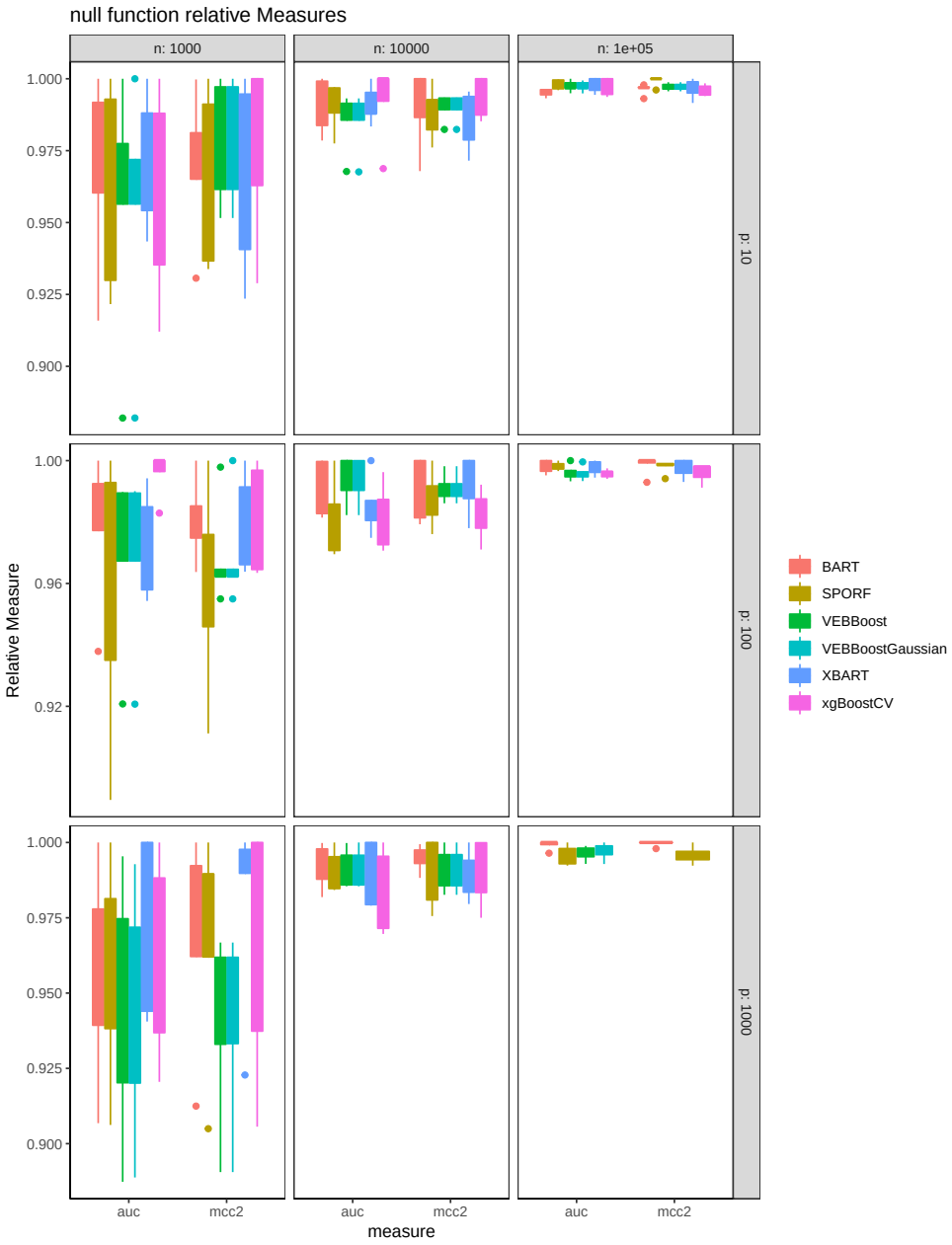

Figure A.13: **Null Function Relative AUC and MCC** This plot shows the relative AUC and $(1 + MCC)/2$ for the logistic model using the null test function (higher is better). All methods are more-or-less on-par with each other.

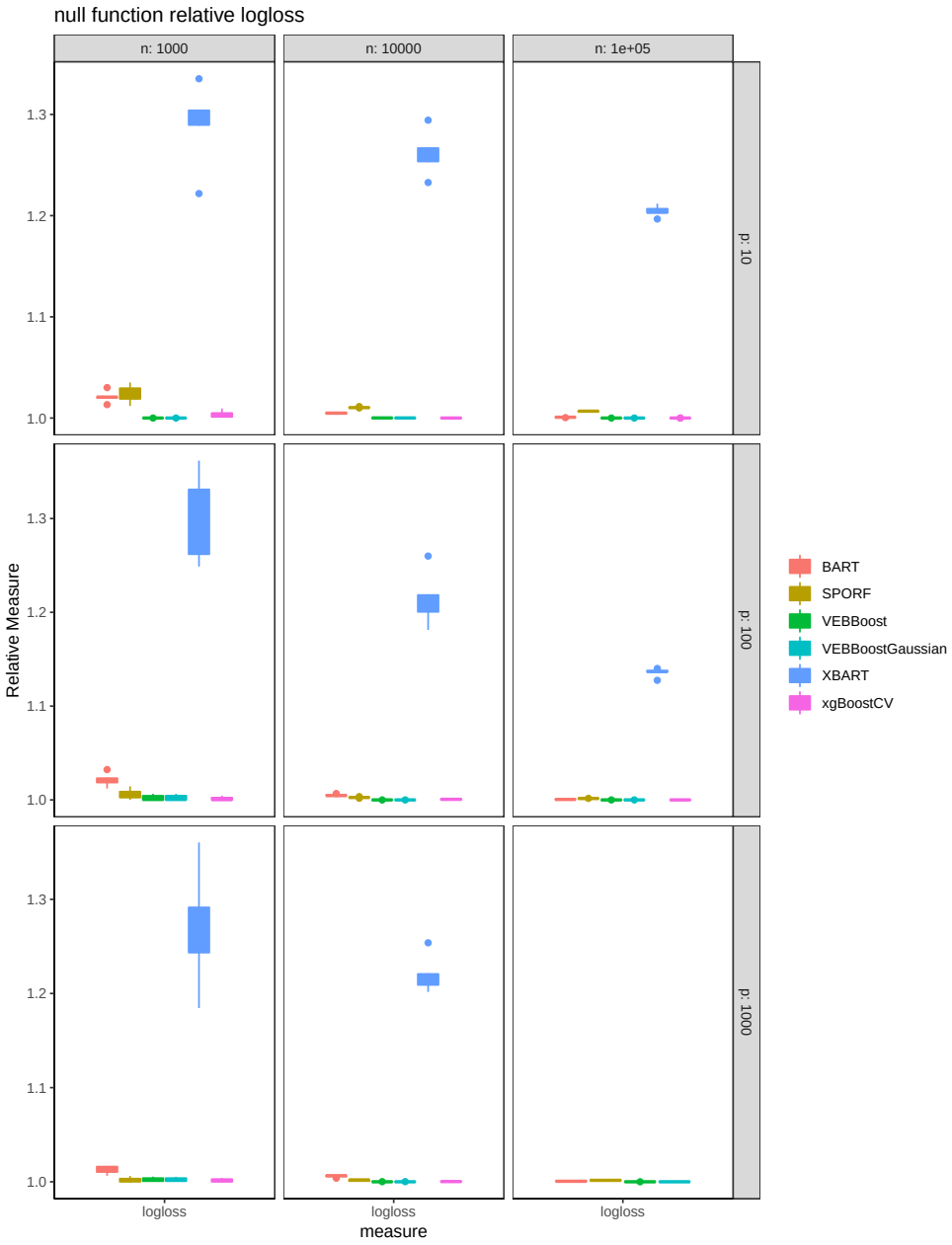

Figure A.14: **Null Function Relative logloss** This plot shows the relative logloss for the logistic model using the null test function (lower is better). Aside from XBART, all methods appear on-par with each other.

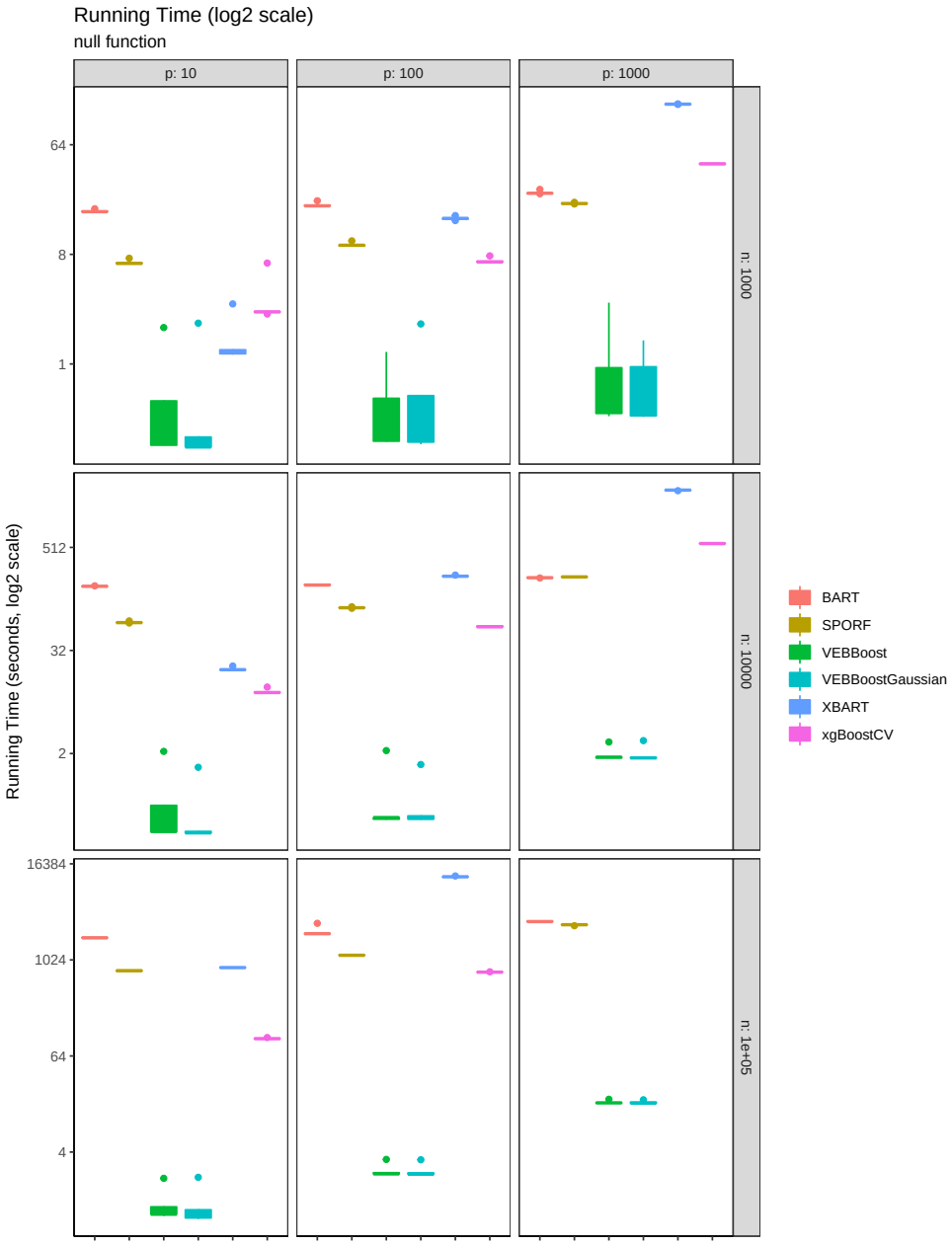

Figure A.15: **Null Function Running Time ($\log_2$ scale)** This plot shows the running time for the logistic model using the null test function (lower is better). As in the Gaussian response simulation study, VEB-Boost is orders of magnitude faster than the other methods in the null case.

A.3 Proofs

A.3.1 Proof of Theorems 2.3.1, 3.3.1, and 3.3.2

In order to prove Theorems 2.3.1, 3.3.1, and 3.3.2, I will re-state the theorem for the VEB-Boost model with an arbitrary precision matrix for the Gaussian errors. From this, we can derive the results of the theorems as special cases.

Theorem A.3.1. *Consider the VEB-Boost model with arbitrary dependence structure among the errors:*

$$\boldsymbol{y} = T\Big(h_1(\ \ _1), \ldots, h_L(\ \ _L)\Big) + \tag{A.1}$$

$$N(\boldsymbol{0}, \boldsymbol{\Lambda}\ ^{1}) \tag{A.2}$$

$$_l\ ^{?}\ g_l\ 2\ G_l \qquad\qquad l = 1, \ldots, L. \tag{A.3}$$

Denote the ELBO of this model using the variational class that factorizes over $_l$ *as*

$$F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \boldsymbol{\Lambda}, T).$$

Denote the ELBO for this model where there is only a single weak learner, i.e. $\mathsf{E}[\boldsymbol{y}] = h(\ \ )$ *as*

$$F_0(g, q; \boldsymbol{y}, h, \boldsymbol{\Lambda}).$$

For $_1$ *and* $_2$*, where* $_1$ *has precision* $\boldsymbol{\Lambda}$ *and* $_2$ *has current posterior variational approximation* q*, let* $\overline{\ }_2 = \mathsf{E}_{\ _2\ q}[\ _2]$ *and let* $\boldsymbol{V} = \mathsf{E}_{\ _2\ q}[\ _2\ \ _2^T]$*. Finally, let*

$$_1\overline{\ \ }\ _2 = \begin{cases} _1\ \ \overline{\ }_2, & \text{if} \quad = + \\ (\boldsymbol{\Lambda}\ \ \boldsymbol{V})\ ^{1} diag(\overline{\ }_2)\boldsymbol{\Lambda}\ _1, & \text{if} \quad = \ . \end{cases}$$

Consider a weak learner $_l = h_l(\ _l)$ *that has path string* $s = d_1 \quad d_K$ *in the ensemble learner* T*. Then*

$$F(g_1, q_1, \ldots, g_L, q_L; \boldsymbol{y}, h_1, \ldots, h_L, \boldsymbol{\Lambda}, T) = F_0(g_l, q_l; \tilde{\boldsymbol{y}}, h_l, \tilde{\boldsymbol{\Lambda}}) + (const\ in\ g_l, q_l),$$

where

$$\tilde{\boldsymbol{y}} = \left(\quad \left((\boldsymbol{y}^{-} \quad _{d_1})^{-}{}_{d_1 \quad d_1 d_2} \right)^{-}{}_{d_1 d_2} \quad {}^{-}{}_{d_1 \quad d_K \ 1 \quad d_1 \quad d_K \ 1 d_K} \right)$$

and

$$\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda}_{\ i=0,\ d_0 \ d_i =}^{K \quad 1} \quad \boldsymbol{V}_{d_0 \quad d_i d_{i+1}}.$$

N.B. As a technical consideration, note that since $\boldsymbol{\Lambda} \quad \mathbf{0}$ and $\mathbf{V}_l \quad \mathbf{0}$, then as long as no $\mathbf{V}_l$ has a row/column of all 0's, then by the Schur product theorem $\tilde{\boldsymbol{\Lambda}} \quad \mathbf{0}$. A $\mathbf{V}_l$ will have a row/column of all 0's $(\ )$ $_{l,i}$ $^{\prime}$ ^{q_l} 0 (i.e. an entry of the ensemble learner $_l$ is a point-mass on 0 under the variational approximation q_l). Barring a degenerate case, this should almost never happen in practice.

Proof: As outlined in Section 3.3.2, define the learner T in terms of weak learner $_l$ with path string $s = d_1 \quad d_K$ as

$$_{d_1} \quad \left(\ _{d_1 d_2} \quad _{d_1} \left(\ _{d_1 d_2 d_3} \quad _{d_1 d_2} \quad \left(\ _l \quad _{d_1 d_2} \quad _{d_K \ 1} \quad _{d_1 d_2} \quad _{d_K \ 1 d_K} \right) \quad \right) \right).$$

We will proceed with an inductive proof over K.

Base case: $\mathbf{K = 1}$ In this case, the ELBO F is

$$\mathsf{E}_{q_l} \left[\quad \frac{n}{2} \log(2\pi) + \frac{1}{2} \log |\boldsymbol{\Lambda}| \quad \frac{1}{2} \mathsf{E}_{q \ _l} \left[\| \boldsymbol{\Lambda}^{1/2} (\mathbf{y} \quad (\ _l \quad _{d_1})) \|_2^2 \right] \right] \quad \sum_{j=1}^{L} D_{KL}(q_j \| g_j).$$

i) $\boxed{\quad = +}$ Starting from the full ELBO, we have

$$
\begin{aligned}
&\mathsf{E}_{q_l}\Bigg[\quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log \jmath \mathbf{\Lambda} \jmath \quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\kappa \mathbf{\Lambda}^{1/2}(\mathbf{y} \quad (\ \ _l + \ \ _{d_1}))\kappa_2^2\Big]\Bigg] \quad \sum_{j=1}^{L} D_{KL}(q_j \kappa g_j) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\kappa \mathbf{\Lambda}^{1/2}((\mathbf{y} \quad \ \ _{d_1}) \quad \ \ _l)\kappa_2^2\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\quad 2(\mathbf{y} \quad \ \ _{d_1})^T \mathbf{\Lambda} \ \ _l + \ \ _l^T \mathbf{\Lambda} \ \ _l\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\Big[\quad 2(\mathbf{y} \quad \overline{\ \ _{d_1}})^T \mathbf{\Lambda} \ \ _l + \ \ _l^T \mathbf{\Lambda} \ \ _l\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\Big[\kappa \mathbf{\Lambda}^{1/2}((\mathbf{y} \quad \overline{\ \ _{d_1}}) \quad \ \ _l)\kappa_2^2\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&F_{\mathsf{0}}(g_l, q_l; \mathbf{y} \quad \overline{\ \ _{d_1}}, h_l, \mathbf{\Lambda}) + (const\ in\ g_l, q_l). \qquad \mathsf{X}
\end{aligned}
$$

ii) $\boxed{\quad =}$ Starting from the full ELBO, we have

$$
\begin{aligned}
&\mathsf{E}_{q_l}\Bigg[\quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log \jmath \mathbf{\Lambda} \jmath \quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\kappa \mathbf{\Lambda}^{1/2}(\mathbf{y} \quad (\ \ _l \quad \ \ _{d_1}))\kappa_2^2\Big]\Bigg] \quad \sum_{j=1}^{L} D_{KL}(q_j \kappa g_j) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\kappa \mathbf{\Lambda}^{1/2}(\mathbf{y} \quad diag(\ \ _{d_1}) \ \ _l)\kappa_2^2\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\kappa \mathbf{\Lambda}^{1/2} diag(\ \ _{d_1})(diag(1/\ \ _{d_1})\mathbf{y} \quad \ \ _l)\kappa_2^2\Big]\Bigg] \quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[\quad 2(diag(1/\ \ _{d_1})\mathbf{y})^T diag(\ \ _{d_1}) \mathbf{\Lambda} diag(\ \ _{d_1}) \ \ _l + \ \ _l^T diag(\ \ _{d_1}) \mathbf{\Lambda} diag(\ \ _{d_1}) \ \ _l\Big]\Bigg] \\
&\quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\Big[\quad 2\mathbf{y}^T \mathbf{\Lambda} diag(\overline{\ \ _{d_1}}) \ \ _l + \ \ _l^T \mathsf{E}_{q\ \ l}[diag(\ \ _{d_1}) \mathbf{\Lambda} diag(\ \ _{d_1})] \ \ _l\Big]\Bigg] \\
&\quad D_{KL}(q_l \kappa g_l) + (const\ in\ g_l, q_l).
\end{aligned}
$$

Note that

$$[diag(\quad)\mathbf{\Lambda} diag(\quad)]_{ij} = \Lambda_{ij}\nu_i\nu_j$$

$$) \ \mathsf{E}_{q\ \ l}[diag(\quad_{d_1})\mathbf{\Lambda} diag(\quad_{d_1})]_{ij} = \begin{cases} \Lambda_{ii}\overline{\quad^2_{d_1;i}} & \text{if } i = j, \\ \Lambda_{ij}\overline{\quad_{d_1;i}\quad d_{1;j}} & \text{if } i \neq j. \end{cases}$$

So if we let

$$\mathbf{V} = \mathsf{E}_{q\ \ l}[\quad_{d_1}\quad^T_{d_1}] \ ) \ V_{ij} = \mathsf{E}_{q\ \ l}[\quad_{d_1;i}\quad d_{1;j}] = \overline{\quad_{d_1,i}\quad d_{1,j}},$$

we get

$$\mathsf{E}_{q\ \ l}[diag(\quad_{d_1})\mathbf{\Lambda} diag(\quad_{d_1})] = \mathbf{\Lambda} \quad \mathbf{V}.$$

Thus, the ELBO further reduces to

$$\mathsf{E}_{q_l}\left[\quad \frac{1}{2}\left[k(\mathbf{\Lambda} \quad \mathbf{V})^{1/2}((\mathbf{\Lambda} \quad \mathbf{V}) \ \ ^1 diag(\overline{\quad_{d_1}})\mathbf{\Lambda}\mathbf{y} \quad\quad _l)k_2^2\right]\right] \quad D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$= F_0(g_l, q_l; (\mathbf{\Lambda} \quad \mathbf{V}) \ \ ^1 diag(\overline{\quad_{d_1}})\mathbf{\Lambda}\mathbf{y}, h_l, \mathbf{\Lambda} \quad \mathbf{V}) + (const\ in\ g_l, q_l). \qquad \mathsf{X}$$

Inductive step: Assume true for K-1 In this case, the ELBO F is

$$\mathsf{E}_{q_l}\Bigg[\quad \frac{n}{2}\log(2\pi) + \frac{1}{2}\log j\mathbf{\Lambda} j$$
$$\frac{1}{2}\mathsf{E}_{q\ \ l}\Big[k\mathbf{\Lambda}^{1/2}(\mathbf{y} \quad\quad _{d_1}$$
$$\left(\quad_{d_1 d_2} \quad _{d_1}\left(\quad_{d_1 d_2 d_3} \quad _{d_1 d_2} \quad (\quad _l \quad _{d_1 d_2} \quad _{d_K\ 1} \quad _{d_1 d_2} \quad _{d_K\ 1 d_K})\right)\right))k_2^2\Big]\Bigg]$$
$$\sum_{j=1}^{L} D_{KL}(q_j k g_j).$$

Let

$$_{d_1} = \Big(\quad _{d_1 d_2} \quad _{d_1} \Big(\quad _{d_1 d_2 d_3} \quad _{d_1 d_2} \quad (\quad _{l} \quad _{d_1 d_2} \quad _{d_K \ 1} \quad _{d_1 d_2} \quad _{d_K \ 1 d_K}) \quad \Big) \Big).$$

Then we can simplify the ELBO as

$$\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{q \ _l}\Big[k\mathbf{\Lambda}^{1/2}(\mathbf{y} \quad _{d_1} \quad _{d_1}) k_2^2 \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l).$$

Note that $_{d_1}$ is an ensemble learner in which $_{l}$ has a path string length (i.e. depth) of $K \quad 1$.

In the below, let $\mathsf{E}_{d_1}[\,]$ be the expectation with respect to the distributions of all components of $_{d_1}$ excluding $_{l}$, and let $\mathsf{E}_{d_1}[\,]$ be the expectation with respect to the distributions of all components of $_{d_1}$.

i) $\boxed{\quad = +}$ Starting from the above simplification of the ELBO, we have

$$\begin{aligned}
&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{q \ _l}\Big[k\mathbf{\Lambda}^{1/2}(\mathbf{y} \quad (\quad _{d_1} + \quad _{d_1})) k_2^2 \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{q \ _l}\Big[k\mathbf{\Lambda}^{1/2}((\mathbf{y} \quad _{d_1}) \quad _{d_1}) k_2^2 \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{d_1}\mathsf{E}_{d_1}\Big[k\mathbf{\Lambda}^{1/2}((\mathbf{y} \quad _{d_1}) \quad _{d_1}) k_2^2 \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{d_1}\mathsf{E}_{d_1}\Big[\quad 2(\mathbf{y} \quad _{d_1})^T \mathbf{\Lambda} \quad _{d_1} + \quad _{d_1}^T \mathbf{\Lambda} \quad _{d_1} \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{d_1}\Big[\quad 2(\mathbf{y} \quad \overline{\quad _{d_1}})^T \mathbf{\Lambda} \quad _{d_1} + \quad _{d_1}^T \mathbf{\Lambda} \quad _{d_1} \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2}\mathsf{E}_{d_1}\Big[k\mathbf{\Lambda}^{1/2}((\mathbf{y} \quad \overline{\quad _{d_1}}) \quad _{d_1}) k_2^2 \Big] \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l) \\
=&\Big[\text{Inductive step, using response } \mathbf{y} \quad \overline{\quad _{d_1}} \text{ and precision } \mathbf{\Lambda}\Big] \\
=&\mathsf{E}_{q_l}\Big[\quad \frac{1}{2} k\tilde{\mathbf{\Lambda}}^{1/2}(\tilde{\mathbf{y}} \quad _{l}) k_2^2 \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l). \qquad \mathsf{X}
\end{aligned}$$

ii) $\boxed{\quad=\quad}$ Starting from the above simplification of the ELBO, we have

$$\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[k\mathbf{\Lambda}^{1/2}(\mathbf{y}\quad(\ \ _{d_1}\quad\ \ _{d_1}))k_2^2\Big]\Bigg]\quad D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{q\ \ l}\Big[k\mathbf{\Lambda}^{1/2}diag(\ \ _{d_1})(diag(1/\ \ _{d_1})\mathbf{y}\quad\ \ _{d_1})k_2^2\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\mathsf{E}_{d_1}\Big[k\mathbf{\Lambda}^{1/2}diag(\ \ _{d_1})(diag(1/\ \ _{d_1})\mathbf{y}\quad\ \ _{d_1})k_2^2\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\mathsf{E}_{d_1}\Big[$$

$$2(diag(1/\ \ _{d_1})\mathbf{y})^T diag(\ \ _{d_1})\mathbf{\Lambda} diag(\ \ _{d_1})\ \ _{d_1} +\ \ _{d_1}^{T} diag(\ \ _{d_1})\mathbf{\Lambda} diag(\ \ _{d_1})\ \ _{d_1}\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\mathsf{E}_{d_1}\Big[\quad 2\mathbf{y}^T\mathbf{\Lambda} diag(\ \ _{d_1})\ \ _{d_1} +\ \ _{d_1}^{T} diag(\ \ _{d_1})\mathbf{\Lambda} diag(\ \ _{d_1})\ \ _{d_1}\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\Big[\quad 2\mathbf{y}^T\mathbf{\Lambda} diag(\overline{\ \ _{d_1}})\ \ _{d_1} +\ \ _{d_1}^{T}\mathsf{E}_{d_1}[diag(\ \ _{d_1})\mathbf{\Lambda} diag(\ \ _{d_1})]\ \ _{d_1}\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\Big[\text{Let } \mathbf{V}_{d_1} := \mathsf{E}_{d_1}[\ \ _{d_1}\ \ _{d_1}^{T}]\Big]$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\Big[\quad 2\mathbf{y}^T\mathbf{\Lambda} diag(\overline{\ \ _{d_1}})\ \ _{d_1} +\ \ _{d_1}^{T}(\mathbf{\Lambda}\quad\mathbf{V}_{d_1})\ \ _{d_1}\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$=\mathsf{E}_{q_l}\Bigg[\quad \frac{1}{2}\mathsf{E}_{d_1}\Big[k(\mathbf{\Lambda}\quad\mathbf{V}_{d_1})^{1/2}((\mathbf{\Lambda}\quad\mathbf{V}_{d_1})\ \ ^{1} diag(\overline{\ \ _{d_1}})\mathbf{\Lambda}\mathbf{y}\quad\ \ _{d_1})k_2^2\Big]\Bigg]$$

$$D_{KL}(q_l k g_l) + (const\ in\ g_l, q_l)$$

$$= \Big[\text{Inductive step, using response } (\mathbf{\Lambda} \quad \mathbf{V}_{d_1}) \quad {}^{1} diag(\overline{\quad}_{d_1}) \mathbf{\Lambda y} \text{ and precision } \mathbf{\Lambda} \quad \mathbf{V}_{d_1} \Big]$$

$$= \mathsf{E}_{q_l} \Big[\quad \frac{1}{2} k \tilde{\mathbf{\Lambda}}^{1/2} (\tilde{\mathbf{y}} \qquad {}_{l}) k_2^2 \Big] \quad D_{KL}(q_l k g_l) + (const \ in \ g_l, q_l). \qquad \mathsf{X}$$

Thus, we have shown the desired result QED

Note that from this, we recover the proofs for the additive model Theorem 2.3.1 (only let $= +$ in the tree structure T and force $\mathbf{\Lambda} \quad \frac{1}{\sigma^2} I_n$) and the VEB-Boost model Theorems 3.3.1 and 3.3.2 (force $\mathbf{\Lambda} \quad diag(1/ \ {}^{2})$).

A.3.2 Proofs of Gaussian Approximations

In this section, I walk through the algebra for the Gaussian approximations to the non-Gaussian data. To keep this subsection self-contained, I will repeat some important bounds here.

The Jaakkola-Jordan bound, given in Lemma 4.2.1, says that for all $x, \xi \ 2 \ \mathsf{R}$,

$$\log \sigma(x) = \frac{x}{2} \quad \log(e^{x/2} + e^{\ x/2}) \quad \frac{x}{2} \quad \frac{1}{2\xi} \Big(\sigma(\xi) \quad \frac{1}{2} \Big) (x^2 \quad \xi^2) \quad \log(e^{\xi/2} + e^{\ \xi/2}).$$

This bound will be used in the cases of binary data, multinomial data (using the Titsias bound), count data, accelerated failure time survival data, ordinal data, pairwise ranking data, listwise ranking data (using the Titsias bound), and proportional hazards survival data (using the Titsias bound).

The Bouchard bound, given in Lemma 4.2.2, says that for all $\quad 2 \ \mathsf{R}^K, \alpha \ 2 \ \mathsf{R}, \mathbf{x} \ 2 \ \mathsf{R}^K$,

$$\log \sum_{k=1}^{K} e^{x_k} \qquad \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c,$$

where

$$\mathbf{A} = diag(\mathbf{d}), \quad d_k = \frac{1}{\xi_k}(\sigma(\xi_k) \quad \frac{1}{2})$$
$$\mathbf{b} = \alpha\mathbf{d} \quad \frac{1}{2}$$
$$c = \sum_{k=1}^{K} \Big[\frac{\xi_k + \alpha}{2} \quad \frac{d_k}{2}(\alpha^2 \quad \xi_k^2) \quad \log(1 + e^{\xi_k})\Big] \quad \alpha.$$

And the Titsias bound, given in Lemma 4.2.3, says that for all $s_k \in \mathbb{R}, k = 1, \ldots, K$,

$$\log\left(\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}\right) \quad \sum_{j \neq k} \log \sigma(s_k \quad s_j).$$

These two bounds will be used in the case of multinomial data, (listwise) ranking data, and proportional hazards survival data.

Proof for Logistic Model for Binary Data

Restating the logistic model for binary response data in equation (4.2), we have

$$\log\left(\frac{\mathbf{p}}{\mathbf{1} \quad \mathbf{p}}\right) = T(\ _1, \ldots, \ _L)$$
$$y_i \overset{?}{} Bern(p_i)$$
$$_l \quad h_l(\ _l), \qquad l = 1, \ldots, L$$
$$_l \overset{?}{} g_l(\ ) \in \mathcal{G}_l, \qquad l = 1, \ldots, L.$$

As in equation (3.4), $T(\ _1, \ldots, \ _L)$ is the tree structure of the VEB-Boost learner and $_l$ are the weak learners.

As stated in equation (4.3), the log-likelihood of this model is

$$l(\ \ _1,\ldots,\ \ _L;\mathbf{y},h_1,\ldots,h_L,T)=\sum_{i=1}^{n}\log\sigma\Big((2y_i\quad 1)T_i\Big),$$

where $\sigma(x)\quad\frac{1}{1+\exp f\ \ xg}$ is the logistic sigmoid function and T_i is the ith value of the VEB-Boost output, $T(\ \ _1,\ldots,\ \ _L)_i$.

Proposition A.3.2. *Given VEB-Boost output $\boldsymbol{T}$ 2 R^n, variational parameters 2 R^n_+, and observations $\boldsymbol{y}$ 2 $f0,1g^n$ we can bound the log-likelihood of the logistic model with*

$$\sum_{i=1}^{n}\log\sigma\Big((2y_i\quad 1)T_i\Big)\qquad\frac{1}{2}\boldsymbol{T}^T\boldsymbol{A}(\boldsymbol{y},\ \ )\boldsymbol{T}+\boldsymbol{b}(\boldsymbol{y},\ \ )^T\boldsymbol{T}+c(\boldsymbol{y},\ \ ),$$

where

$$\begin{aligned}\boldsymbol{A}(\boldsymbol{y},\ \ )&=diag(\boldsymbol{d}),\quad d_i=\frac{1}{\xi_i}\Big(\sigma(\xi_i)\quad\frac{1}{2}\Big)\\ \boldsymbol{b}(\boldsymbol{y},\ \ )&=y_i\quad\frac{1}{2}\\ c(\boldsymbol{y},\ \ )&=\sum_{i=1}^{n}\log\sigma(\xi_i)+\frac{\xi_i}{2}(d_i\xi_i\quad 1).\end{aligned}$$

Proof: Using the Jaakkola-Jordan bound, we have

$$\begin{aligned}
\log \sigma\Big((2y_i \quad 1)T_i\Big) \quad & \frac{(2y_i \quad 1)T_i}{2} \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)\Big(\big((2y_i \quad 1)T_i\big)^2 \quad \xi_i^2\Big) \\
& \log\Big(e^{\xi_i/2} + e^{\quad \xi_i^2}\Big) \\
&= \Big[y_i \in \{0,1\} \Rightarrow 2y_i \quad 1 \in \{1,1\} \Rightarrow (2y_i \quad 1)^2 = 1\Big] \\
&= \Big(y_i \quad \frac{1}{2}\Big)T_i \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)\Big(T_i^2 \quad \xi_i^2\Big) \quad \log\Big(e^{\xi_i/2} + e^{\quad \xi_i^2}\Big) \\
&= \quad \frac{d_i}{2}T_i^2 + \Big(y_i \quad \frac{1}{2}\Big)T_i + \frac{d_i}{2}\xi_i^2 \quad \frac{\xi_i}{2} \quad \log\Big(e^{\xi_i/2} + e^{\quad \xi_i^2}\Big) \\
&= \Big[\log\sigma(x) = \frac{x}{2} \quad \log\Big(e^{x/2} + e^{\quad x^2}\Big)\Big] \\
&= \quad \frac{d_i}{2}T_i^2 + \Big(y_i \quad \frac{1}{2}\Big)T_i + \log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1).
\end{aligned}$$

Therefore, we get

$$\begin{aligned}
\sum_{i=1}^{n} \log \sigma\Big((2y_i \quad 1)T_i\Big) \qquad & \frac{d_i}{2}T_i^2 + \Big(y_i \quad \frac{1}{2}\Big)T_i + \log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1) \\
&= \quad \frac{1}{2}\mathbf{T}^T\mathbf{A}(\mathbf{y}, \ )\mathbf{T} + \mathbf{b}(\mathbf{y}, \ )^T\mathbf{T} + c(\mathbf{y}, \ ),
\end{aligned}$$

where $\mathbf{A}(\mathbf{y}, \ )$, $\mathbf{b}(\mathbf{y}, \ )$, and $c(\mathbf{y}, \ )$ are as defined above. QED

Corollary A.3.2.1. *Given the results from the introduction to chapter 4, we can approximate our data as Gaussian with response* $\frac{1}{d_i}(y_i \quad \frac{1}{2})$ *and variance* $\frac{1}{d_i}$.

Proposition A.3.3. *For xed variational distributions* $q_1, \ldots, q_L$*, we can maximize the lower-bound to the ELBO with respect to* ξ_i *by setting*

$$\xi_i = +\sqrt{\mathsf{E}_q\Big[T(\ _1, \ldots, \ _L)_i^2\Big]}.$$

Proof: First, note that the lower-bound to the log-likelihood factorizes into a sum of terms, each depending on ξ_i. And furthermore, neither the variational approximation q, nor

the KL-divergence term in the ELBO, depends on ξ_i. Thus, we can optimize with respect to each ξ_i individually. For a given ξ_i, recall that the lower-bound to the log-likelihood is

$$\frac{d_i}{2}T_i^2 + \Big(y_i \quad \frac{1}{2}\Big)T_i + \log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1).$$

Thus, the contribution of this term to the ELBO is

$$\begin{aligned} & \mathsf{E}_q\Big[\quad \frac{d_i}{2}T_i^2 + \Big(y_i \quad \frac{1}{2}\Big)T_i + \log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1)\Big] \\ = \quad & \frac{d_i}{2}\overline{T_i^2} + \Big(y_i \quad \frac{1}{2}\Big)\overline{T_i} + \log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1) \\ = \quad & \frac{1}{2\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)(\overline{T_i^2} \quad \xi_i^2) + \log\sigma(\xi_i) \quad \frac{\xi_i}{2} + const. \end{aligned}$$

By analytically maximizing this function with respect to ξ_i using the first and second order optimality conditions (or plugging it into Wolfram Alpha), we get that $\xi_i = \sqrt{\overline{T_i^2}}$. We just take the positive solution, so we get $\xi_i = +\sqrt{\mathsf{E}_q\Big[T(\ _1, \ldots, \ _L)_i^2\Big]}$. *QED*

Proof for Multinomial Data

Restating the multinomial logistic model for categorical response data in equation (4.5), we have

$$\begin{aligned} \mathbf{s}_k &= T^k(\ _1^k, \ldots, \ _{L_k}^k) \\ p_{ki} := P(Y_i = k \mid s_{1i}, \ldots, s_{Ki}) &= \frac{\exp\{s_{ki}\}}{\sum_{j=1}^K \exp\{s_{ji}\}} \\ y_i \ ^? \ & Multi(1, \mathbf{p}_i) \\ _l^k \quad & h_l^k(\ _l^k), \qquad k = 1, \ldots, K \quad l = 1, \ldots, L_k \\ _l^k \ ^? \ & g_l^k(\ )\ 2\ G_l^k, \qquad k = 1, \ldots, K \quad l = 1, \ldots, L_k. \end{aligned}$$

The interpretation here is that for each class $k \in \{1, \ldots, L\}$, we estimate a score vector $\mathbf{s}_k \in \mathbb{R}^n$. Each of these K models can have a different ensemble tree structure T^k, different weak learners $\quad$ etc.

As stated in equation (4.6), the log-likelihood in this model is

$$l(\quad; \mathbf{y}, h_1^1, \ldots, h_{L_k}^K) = \sum_{i=1}^{n} \log \left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^{K} \exp\{T_i^k\}} \right).$$

Proposition A.3.4. *Given VEB-Boost outputs $\boldsymbol{T}^1, \ldots, \boldsymbol{T}^K \in \mathbb{R}^n$ for our K classes, variational parameters $\quad_i \in \mathbb{R}_+^K,\ i = 1, \ldots, n$ and $\quad \in \mathbb{R}^n$, and observations $\boldsymbol{y} \in \{1, 2, \ldots, K\}^n$, we can use the Bouchard bound to bound the log-likelihood of the multinomial model with*

$$l(\quad; \boldsymbol{y}, h_1^1, \ldots, h_{L_k}^K, T^1, \ldots, T^K) = \sum_{i=1}^{n} \log \left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^{K} \exp\{T_i^k\}} \right)$$
$$\sum_{k=1}^{K} \quad \frac{1}{2} \boldsymbol{T}^{kT} \boldsymbol{A}^k \boldsymbol{T}^k + \boldsymbol{T}^{kT} \boldsymbol{b}^k + \sum_{i=1}^{n} c_i$$

where

$$\boldsymbol{A}^k = diag(d_1^k, \ldots, d_n^k), \quad d_i^k = \frac{1}{\xi_i^k} \left(\sigma(\xi_i^k) \quad \frac{1}{2} \right)$$
$$\boldsymbol{b}^k = (b_1^k + \mathbb{1}_{y_1 = k}, \ldots, b_n^k + \mathbb{1}_{y_n = k}), \quad b_i^k = \alpha_i(d_i^k) \quad \frac{1}{2}$$
$$c_i = \sum_{k=1}^{K} \Big[\frac{\xi_i^k + \alpha_i}{2} \quad \frac{d_i^k}{2} (\alpha_i^2 \quad \xi_i^{k2}) \quad \log(1 + e^{\xi_i^k}) \Big] \quad \alpha_i.$$

Proof: For notation, let $\mathbf{T}_i := (T_i^1, \ldots, T_i^K)^T \in \mathbb{R}^K$ and let $\mathbf{T}^k := (T_1^k, \ldots, T_n^k)^T \in \mathbb{R}^n$.

Using the Bouchard bound, for fixed variational parameters $_i$ and , we have

$$\log\left(\frac{\exp\!f T_i^{y_i} g}{\sum_{k=1}^{K}\exp\!f T_i^{k} g}\right) = T_i^{y_i} \quad \log\Big(\sum_{k=1}^{K}\exp\!f T_i^{k} g\Big)$$
$$T_i^{y_i} \quad \frac{1}{2}\mathbf{T}_i^T\mathbf{A}_i\mathbf{T}_i + \mathbf{T}_i^T\mathbf{b}_i + c_i,$$

where

$$\mathbf{A}_i = diag(\mathbf{d}_i)\ 2\ \mathsf{R}^{K\ \ K}, \quad d_i^k = \frac{1}{\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)$$
$$\mathbf{b}_i = \alpha_i\mathbf{d}_i \quad \frac{1}{2}\mathbf{1}\ 2\ \mathsf{R}^K$$
$$c_i = \sum_{k=1}^{K}\Big[\frac{\xi_i^k + \alpha_i}{2} + \frac{d_i^k}{2}(\alpha_i^2 \quad \xi_i^{k2}) + \log(1+e^{\xi_i^k})\Big] \quad \alpha_i.$$

Thus,

$$\begin{aligned} l(\ {}^{1}_{1},\ldots,\ {}^{K}_{L_K};\mathbf{y},h^{1}_{1},\ldots,h^{K}_{L_k},T^1,\ldots,T^K) &= \sum_{i=1}^{n}\log\left(\frac{\exp\!f T_i^{y_i} g}{\sum_{k=1}^{K}\exp\!f T_i^{k} g}\right) \\ &= \sum_{i=1}^{n} T_i^{y_i} \quad \log\Big(\sum_{k=1}^{K}\exp\!f T_i^{k} g\Big) \\ &\quad \sum_{i=1}^{n} T_i^{y_i} \quad \frac{1}{2}\mathbf{T}_i^T\mathbf{A}_i\mathbf{T}_i + \mathbf{T}_i^T\mathbf{b}_i + c_i \\ &= \sum_{i=1}^{n} \quad \frac{1}{2}\mathbf{T}_i^T\mathbf{A}_i\mathbf{T}_i + \mathbf{T}_i^T(\mathbf{b}_i + \mathbf{e}_{y_i}) + c_i \\ &= \sum_{k=1}^{K} \quad \frac{1}{2}\mathbf{T}^{kT}\mathbf{A}^k\mathbf{T}^k + \mathbf{T}^{kT}\mathbf{b}^k + \sum_{i=1}^{n} c_i, \end{aligned}$$

where

$$\mathbf{A}^k = diag(d_1^k, \ldots, d_n^k)$$
$$\mathbf{b}^k = (b_1^k + \mathsf{I}_{y_1=k}, \ldots, b_n^k + \mathsf{I}_{y_n=k}).$$

This follows simply from switching the order of summation of $i = 1, \ldots, n$ and $k = 1, \ldots, K$, which we can do since all matrices here are diagonal. QED

Corollary A.3.4.1. *Given the results from the introduction to chapter 4, when tting our model for class k, we can use the Bouchard bound to approximate our data as Gaussian with response $\frac{1}{d_i^k}(\mathsf{I}_{y_i=k} \quad \frac{1}{2} + \alpha_i d_i^k)$ and variance $\frac{1}{d_i^k}$.*

Proposition A.3.5. *For xed variational distributions and xed α_i, we can maximize the lower-bound to the ELBO with respect to ξ_i^k by setting*

$$\xi_i^k \;= +\sqrt{\mathsf{E}_q[T^k(\;{}_1^k, \ldots, \;{}_{L_k}^k)_i^2] \quad 2\alpha_i \mathsf{E}_q[T_k(\;{}_1^k, \ldots, \;{}_{L_k}^k)_i] + \alpha_i^2}.$$

And for xed variational distributions and xed $_i$, we can maximize the lower-bound to the ELBO with respect to α_i be setting

$$\alpha_i \;= \frac{K/2 \quad 1 + \sum_{k=1}^K d_i^k \mathsf{E}_q[T^k(\;{}_1^k, \ldots, \;{}_{L_k}^k)]_i}{\sum_{k=1}^K d_i^k}.$$

We can either alternate between updating the α_i's and ξ_i^k's until they converge, or just perform a single update.

Proof: First, note that the lower-bound to the log-likelihood factorizes into a sum of terms, each depending on ξ_i^k. And furthermore, neither the variational approximation q, nor the KL-divergence term in the ELBO, depends on ξ_i^k. Thus, we can optimize with respect to each ξ_i^k individually. For a given ξ_i^k, note that the lower-bound to the log-likelihood can

be expressed as

$$\frac{1}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)T_i^{k2} + \Big(\frac{\alpha_i}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big) \quad \frac{1}{2} + \mathsf{I}_{y_i=k}\Big)T_i^k$$
$$+ \frac{\xi_i^k + \alpha_i}{2} + \frac{1}{2\xi_i}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)(\alpha_i^2 \quad \xi_i^{k2}) + \log(1 + e^{\xi_i^k}) \quad \alpha_i + const.$$

Thus, the contribution of this term to the ELBO is

$$\begin{aligned}
&\mathsf{E}_q\Big[\quad \frac{1}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)T_i^{k2} + \Big(\frac{\alpha_i}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big) \quad \frac{1}{2} + \mathsf{I}_{y_i=k}\Big)T_i^k \\
&\quad + \frac{\xi_i^k + \alpha_i}{2} \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)(\alpha_i^2 \quad \xi_i^{k2}) \quad \log(1 + e^{\xi_i^k}) \quad \alpha_i + const\Big] \\
= \quad &\frac{1}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)\overline{T_i^{k2}} + \Big(\frac{\alpha_i}{2\xi_i^k}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)\Big)\overline{T_i^k} \\
&\quad + \frac{\xi_i^k + \alpha_i}{2} \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i^k) \quad \frac{1}{2}\Big)(\alpha_i^2 \quad \xi_i^{k2}) \quad \log(1 + e^{\xi_i^k}) + const.
\end{aligned}$$

By analytically maximizing this function with respect to ξ_i^k using the first and second order optimality conditions (or plugging it into Wolfram Alpha), we get that

$$\xi_i^k \quad = \quad \sqrt{\mathsf{E}_q[T_k(\ {}^k_1, \ldots, \ {}^k_{L_k})_i^2] \quad 2\alpha_i \mathsf{E}_q[T_k(\ {}^k_1, \ldots, \ {}^k_{L_k})_i] + \alpha_i^2}.$$

We just take the positive solution, so we get

$$\xi_i^k \quad = + \sqrt{\mathsf{E}_q[T_k(\ {}^k_1, \ldots, \ {}^k_{L_k})_i^2] \quad 2\alpha_i \mathsf{E}_q[T_k(\ {}^k_1, \ldots, \ {}^k_{L_k})_i] + \alpha_i^2}.$$

Now, focusing on α_i, the ELBO as a function of α_i can be written as

$$\alpha_i + \sum_{k=1}^{K} \alpha_i d_i^k \overline{T_i^k} + \frac{\alpha_i}{2} \quad \frac{d_i^k}{2} \alpha_i^2 + const$$
$$= \Big(\frac{K}{2} \quad 1 + \sum_{k=1}^{K} d_i^k \overline{T_i^k} \Big) \alpha_i \quad \frac{\alpha_i^2}{2} \sum_{k=1}^{K} d_i^k.$$

Again, by analytically maximizing this function with respect to α_i, we get

$$\alpha_i = \frac{K/2 \quad 1 + \sum_{k=1}^{K} d_i^k \mathsf{E}_q[T_k(\ {}^k_1, \ldots, \ {}^k_{L_k})]_i}{\sum_{k=1}^{K} d_i^k}.$$

QED

Proposition A.3.6. *Given VEB-Boost outputs $\boldsymbol{T}^1, \ldots, \boldsymbol{T}^K \in \mathbb{R}^n$ for our K classes, variational parameters $\ {}_i \in \mathbb{R}_+^K,\ i = 1, \ldots, n$, and observations $\boldsymbol{y} \in \{1, 2, \ldots, K\}^n$, we can use the Titsias bound in conjunction with the Jaakkola-Jordan bound to bound the log-likelihood of the multinomial model with*

$$l(\ {}^1_1, \ldots, \ {}^K_{L_K}; \boldsymbol{y}, h_1^1, \ldots, h_{L_k}^K, T^1, \ldots, T^K) = \sum_{i=1}^{n} \log \left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^{K} \exp\{T_i^k\}} \right)$$
$$\sum_{i=1}^{n} \sum_{m \neq y_i} \frac{1}{2\xi_i^m} \Big(\sigma(\xi_i^m) \quad \frac{1}{2} \Big) \Big((T_i^{y_i} \quad T_i^m)^2 \quad \xi_i^{m2} \Big) + \frac{T_i^{y_i} \quad T_i^m}{2} \quad \log \Big(e^{\xi_i^m/2} + e^{\ \xi_i^m/2} \Big).$$

Proof: With the same notation as in the proof for the Bouchard multinomial bound,

focusing on a single observation i, we have

$$\log\left(\frac{\exp\{T_i^{y_i}\}}{\sum_{k=1}^{K}\exp\{T_i^{k}\}}\right) \quad \Big[\text{Titsias bound}\Big]$$

$$\sum_{m\neq y_i}\log\sigma(T_i^{y_i} - T_i^{m}) \quad \Big[\text{Jaakkola-Jordan bound applied to each term in the sum}\Big]$$

$$\sum_{m\neq y_i}\frac{1}{2\xi_i^m}\Big(\sigma(\xi_i^m) - \frac{1}{2}\Big)\Big[(T_i^{y_i} - T_i^{m})^2 - \xi_i^{m2}\Big] + \frac{T_i^{y_i} - T_i^{m}}{2} - \log\Big(e^{\xi_i^m/2} + e^{-\xi_i^m/2}\Big).$$

Thus, when combing all n observations, we get the desired result. QED

Corollary A.3.6.1. *Focusing on the quadratic approximation for a given observation i and class k, when fitting our model for class k, we can use the Titsias bound to approximate our data as Gaussian with response $\overline{T_i^{y_i}} - \frac{1}{2d_i^k}$ and variance $\frac{1}{d_i^k}$ for $k \neq y_i$, and as a Gaussian with response $\frac{1}{\sum_{m\neq k} d_i^m}\sum_{m\neq k}\frac{1}{2} + d_i^m\overline{T_i^m}$ and variance $\frac{1}{\sum_{m\neq k} d_i^m}$ for $k = y_i$.*

Proposition A.3.7. *For fixed variational distributions, we can maximize the lower-bound to the ELBO with respect to ξ_i^m by setting*

$$\xi_i^m = +\sqrt{\overline{T_i^{y_i 2}} - 2\overline{T_i^{y_i}}\,\overline{T_i^m} + \overline{T_i^{m2}}}.$$

Proof: As a function of ξ_i^m for $m \neq y_i$, it is easy to see that the ELBO can be expressed as

$$\frac{1}{2\xi_i^m}\Big(\sigma(\xi_i^m) - \frac{1}{2}\Big)\Big[\overline{(T_i^{y_i} - T_i^{m})^2} - \xi_i^{m2}\Big] - \log\Big(e^{\xi_i^m/2} + e^{-\xi_i^m/2}\Big) + const.$$

With the same optimization results as in the logistic case, we see that

$$
\begin{aligned}
\xi_i^m &= +\sqrt{\overline{(T_i^{y_i} \quad T_i^m)^2}} \\
&= +\sqrt{\overline{T_i^{y_i 2}} \quad 2\overline{T_i^{y_i} T_i^m} + \overline{T_i^{m2}}} \\
&= \left[T_i^{y_i} \ ? \ T_i^m \text{ because of our choice of variational family } Q \right] \\
&= +\sqrt{\overline{T_i^{y_i 2}} \quad 2\overline{T_i^{y_i}}\,\overline{T_i^m} + \overline{T_i^{m2}}}.
\end{aligned}
$$

And for $m = y_i$, there is no ξ_i^m QED

Proof for Negative Binomial Count Data

Restating the negative binomial count data model given in equation (4.8), we have

$$
\begin{aligned}
y_i \ &^? \ NB(r_i, p_i) \\
\log\left(\frac{p_i}{1 \quad p_i}\right) &= T(\ \ _1, \ldots, \ \ _L)_i \\
\ _l \quad &h_l(\ \ _l), \qquad\qquad l = 1, \ldots, L \\
\ _l \ &^? \ g_l(\ ) \ 2 \ \mathsf{G}_l, \qquad l = 1, \ldots, L,
\end{aligned}
$$

where r_i is fixed and known.

Proposition A.3.8. *Given VEB-Boost output $\boldsymbol{T}$ 2 R^n, variational parameters 2 R_+^n, and observations $\boldsymbol{y}$ 2 N^n, we can bound the log-likelihood of the negative binomial model with*

$$
\begin{aligned}
& l(\ \ _1, \ldots, \ \ _L; \boldsymbol{y}, h_1, \ldots, h_L, T) \\
=& \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + r_i \log \sigma(\ \ T_i) + y_i \log \sigma(T_i) \\
& \frac{1}{2} \boldsymbol{T}^T \boldsymbol{A}(\boldsymbol{y}, \ \ ) \boldsymbol{T} + \boldsymbol{b}(\boldsymbol{y}, \ \ )^T \boldsymbol{T} + c(\boldsymbol{y}, \ \ ),
\end{aligned}
$$

where

$$\boldsymbol{A}(\boldsymbol{y}, \ ) = diag(\boldsymbol{d}(\boldsymbol{y}+\boldsymbol{r})), \quad d_i = \frac{1}{\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)$$
$$\boldsymbol{b}(\boldsymbol{y}, \ ) = \frac{\boldsymbol{y} \quad \boldsymbol{r}}{2}$$
$$c(\boldsymbol{y}, \ ) = \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + (y_i + r_i)\Big[\log\sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1)\Big].$$

Proof: Performing some algebra on the log-likelihood, we get

$$\sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + r_i \log\sigma(\quad T_i) + y_i \log\sigma(T_i)$$
$$= \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} \quad r_i \log(1 + e^{T_i}) + y_i \log\frac{e^{T_i}}{1 + e^{T_i}}$$
$$= \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} \quad r_i \log(1 + e^{T_i}) + y_i T_i \quad y_i \log(1 + e^{T_i})$$
$$= \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + y_i T_i + (y_i + r_i)\log\sigma(\quad T_i)$$
$$= \Big[\log\sigma(x) = \frac{x}{2} \quad \log(e^{x/2} + e^{\ -x/2}) \ \textit{)} \ \log\sigma(\quad x) = \quad \frac{x}{2} \quad \log(e^{x/2} + e^{\ -x/2})\Big]$$
$$= \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + y_i T_i + (y_i + r_i)\Big(\quad \frac{T_i}{2} \quad \log(e^{T_i/2} + e^{\ -T_i/2})\Big)$$
$$\Big[\text{Jaakkola-Jordan}\Big]$$
$$\sum_{i=1}^{n} \Bigg[\log \binom{y_i + r_i \quad 1}{y_i} + y_i T_i + (y_i + r_i)\Big(\quad \frac{T_i}{2} \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)(T_i^2 \quad \xi_i^2)$$
$$\log(e^{\xi_i/2} + e^{\ -\xi_i/2})\Big)\Bigg]$$
$$= \quad \frac{1}{2}\mathbf{T}^T\mathbf{A}(\mathbf{y}, \ )\mathbf{T} + \mathbf{b}(\mathbf{y}, \ )^T\mathbf{T} + c(\mathbf{y}, \ ),$$

where

$$\mathbf{A}(\mathbf{y}, \ ) = diag(\mathbf{d}(\mathbf{y}+\mathbf{r})), \quad d_i = \frac{1}{\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)$$
$$\mathbf{b}(\mathbf{y}, \ ) = \frac{\mathbf{y} \quad \mathbf{r}}{2}$$
$$c(\mathbf{y}, \ ) = \sum_{i=1}^{n} \log \binom{y_i + r_i \quad 1}{y_i} + (y_i + r_i)\Big[\log \sigma(\xi_i) + \frac{\xi_i}{2}(d_i\xi_i \quad 1)\Big].$$

QED

Corollary A.3.8.1. *Given the results from the introduction to chapter 4, we can approximate our data as Gaussian with response* $\frac{y_i \quad r_i}{2d_i(y_i+r_i)}$ *and variance* $\frac{1}{d_i(y_i+r_i)}$.

Proposition A.3.9. *For xed variational distributions* $q_1, \ldots, q_L$*, we can maximize the lower-bound to the ELBO with respect to* ξ_i *by setting*

$$\xi_i \ = +\sqrt{\mathsf{E}_q[T(\ _1, \ldots, \ _L)_i^2]}.$$

Proof: This follows directly from the results of the analogous proposition for the case of the logistic model. QED

Proof for AFT (log-logistic) Survival Data

Restating the model in this setting, we have

$$\log \mathbf{y} = T(\ _1, \ldots, \ _L) + \epsilon_i \overset{iid}{} logistic(0, s),$$

where the $logistic(\mu, s)$ distribution has pdf

$$f(x; \mu, s) = \frac{e^{\ (x \ \ \mu)/s}}{s(1 + e^{\ (x \ \ \mu)/s})^2}$$

and CDF

$$F(x;\mu,s) = \frac{1}{1+e^{-(x-\mu)/s}} = \sigma\Big(\frac{x-\mu}{s}\Big).$$

In the setting of survival data, we can have censored observations. An observation is uncensored if we observe y_i exactly, left-censored if we know $y_i \in (0, e^{t_2})$, right-censored if we know $y_i \in (e^{t_1}, \infty)$, and interval-censored if we know $y_i \in (e^{t_1}, e^{t_2})$, for $t_1 > t_2 \in \mathbb{R}$. We make the assumption of non-informative and random censoring (Patti et al. [2007]). Thus, the contribution to the likelihood of uncensored observations is $f(\log y_i; T_i, s)$, the contribution to the likelihood of left-censored observations is $F(t_2; T_i, s)$, the contribution of right-censored observations if $1 - F(t_1; T_i, s)$, and the contribution for interval-censored observations is $F(t_2; T_i, s) - F(t_1; T_i, s)$.

Denote the log of an uncensored time for observation i as t_i, the log of a left-censored time as t_i^2, the log of a right-censored time as t_i^1, and the log of the left and right endpoints of an interval-censored time as t_i^1 and t_i^2, respectively. With this in mind, we can express the log-likelihood of the model (up to a constant) as

$$\begin{aligned}
&\sum_{i:\text{uncensored}} \log \frac{e^{-(t_i - T_i)/s}}{s(1+e^{-(t_i - T_i)/s})^2} \\
&+ \sum_{i:\text{left-censored}} \log \sigma\Big(\frac{t_i^2 - T_i}{s}\Big) \\
&+ \sum_{i:\text{right-censored}} \log \Big[1 - \sigma\Big(\frac{t_i^1 - T_i}{s}\Big)\Big] \\
&+ \sum_{i:\text{interval-censored}} \log \Big[\sigma\Big(\frac{t_i^2 - T_i}{s}\Big) - \sigma\Big(\frac{t_i^1 - T_i}{s}\Big)\Big].
\end{aligned}$$

Performing some simple algebra, along with the facts that $1 - \sigma(x) = \sigma(-x)$ and

$$\begin{aligned}
&\log \Big[\sigma\Big(\frac{t_i^2 - T_i}{s}\Big) - \sigma\Big(\frac{t_i^1 - T_i}{s}\Big)\Big] \\
=&\frac{T_i}{s} - \log\Big(e^{T_i/s} + e^{t_i^2/s}\Big) - \log\Big(e^{T_i/s} + e^{t_i^1/s}\Big) + \log\Big(e^{t_i^2/s} - e^{t_i^1/s}\Big),
\end{aligned}$$

we can simplify this expression as

$$\begin{aligned}
&\sum_{i:\text{uncensored}} \frac{T_i \quad t_i}{s} \quad \log(s) + 2\log\sigma\Big(\frac{t_i \quad T_i}{s}\Big) \\
&+ \sum_{i:\text{left-censored}} \log\sigma\Big(\frac{t_i^2 \quad T_i}{s}\Big) \\
&+ \sum_{i:\text{right-censored}} \log\sigma\Big(\frac{T_i \quad t_i^1}{s}\Big) \\
&+ \sum_{i:\text{interval-censored}} \frac{T_i}{s} \quad \log\Big(e^{T_i/s} + e^{t_i^2/s}\Big) \quad \log\Big(e^{T_i/s} + e^{t_i^1/s}\Big) + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big).
\end{aligned}$$

Focusing on the interval-censored observations, we can further manipulate the expression to get

$$\begin{aligned}
&\sum_{i:\text{interval-censored}} \frac{T_i}{s} \quad \log\Big(e^{T_i/s} + e^{t_i^2/s}\Big) \quad \log\Big(e^{T_i/s} + e^{t_i^1/s}\Big) + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big) \\
=&\sum_{i:\text{interval-censored}} \frac{T_i}{s} \quad \log\Big(e^{t_i^2/s}(1 + e^{(T_i \quad t_i^2)/s})\Big) \quad \log\Big(e^{t_i^1/s}(1 + e^{(T_i \quad t_i^1)/s})\Big) \\
&\qquad + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big) \\
=&\sum_{i:\text{interval-censored}} \frac{T_i}{s} \quad \frac{t_i^2}{s} + \log\sigma((t_i^2 \quad T_i)/s) \quad \frac{t_i^2}{s} + \log\sigma((t_i^1 \quad T_i)/s) + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big) \\
=&\Big[\log\sigma(x) = \frac{x}{2} \quad \log(e^{x/2} + e^{\ -x/2})\Big] \\
=&\sum_{i:\text{interval-censored}} \Bigg[\frac{T_i \quad t_i^2 \quad t_i^1}{s} + \frac{t_i^2 \quad T_i}{2s} \quad \log\Big(e^{(t_i^2 \ - \ T_i)/2s} + e^{\ (t_i^2 \ - \ T_i)/2s}\Big) \\
&\quad + \frac{t_i^1 \quad T_i}{2s} \quad \log\Big(e^{(t_i^1 \ - \ T_i)/2s} + e^{\ (t_i^1 \ - \ T_i)/2s}\Big) + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big)\Bigg] \\
=&\sum_{i:\text{interval-censored}} \Bigg[\quad \frac{t_i^2 + t_i^1}{2s} \quad \log\Big(e^{(t_i^2 \ - \ T_i)/2s} + e^{\ (t_i^2 \ - \ T_i)/2s}\Big) \\
&\qquad \log\Big(e^{(t_i^1 \ - \ T_i)/2s} + e^{\ (t_i^1 \ - \ T_i)/2s}\Big) + \log\Big(e^{t_i^2/s} \quad e^{t_i^1/s}\Big)\Bigg].
\end{aligned}$$

We can then apply the Jaakkola-Jordan bound separately to the uncensored observations, left-censored observations, right-censored observations, and each of the two terms in the expression above for the interval-censored observations.

Uncensored Observations: For the uncensored observations, an application of the Jaakkola-Jordan bound gives

$$\begin{aligned}
&\frac{T_i - t_i}{s} \quad \log(s) + 2\log\sigma\Big(\frac{t_i - T_i}{s}\Big) \\
&\frac{T_i - t_i}{s} \quad \log(s) + 2\Big[\frac{t_i - T_i}{2s} - \frac{1}{2\xi_i}\Big(\sigma(\xi_i) - \frac{1}{2}\Big)\Big(\frac{(T_i - t_i)^2}{s^2} - \xi_i^2\Big) - \log\Big(e^{\xi_i/2} + e^{-\xi_i/2}\Big)\Big] \\
=&\log(s) - \frac{1}{2}\frac{2d_i}{s^2}(T_i - t_i)^2 + d_i\xi_i^2 - 2\log\Big(e^{\xi_i/2} + e^{-\xi_i/2}\Big).
\end{aligned}$$

Thus, we can approximate the uncensored observations as being Gaussian with response t_i and variance $\frac{s^2}{2d_i}$. We can also use the previous results to show that when updating ξ_i, we set

$$\xi_i = +\sqrt{\mathsf{E}_q\Big[\frac{(T_i - t_i)^2}{s^2}\Big]} = +\frac{1}{s}\sqrt{\overline{T_i^2} - 2t_i\overline{T_i} + t_i^2}.$$

Left-censored Observations: For the left-censored observations, an application of the Jaakkola-Jordan bound gives

$$\begin{aligned}
\log\sigma\Big(\frac{t_i^2 - T_i}{s}\Big) \quad & \frac{t_i^2 - T_i}{2s} - \frac{1}{2\xi_i}\Big(\sigma(\xi_i) - \frac{1}{2}\Big)\Big(\frac{(t_i^2 - T_i)^2}{s^2} - \xi_i^2\Big) - \log\Big(e^{\xi_i/2} + e^{-\xi_i/s}\Big) \\
= \quad & \frac{d_i}{2s^2}\Big(T_i^2 - 2t_i^2 T_i + \frac{s}{d_i}T_i\Big) + const \\
= \quad & \frac{d_i}{2s^2}\Big(T_i - (t_i^2 - \frac{s}{2d_i})\Big)^2 + const.
\end{aligned}$$

Thus, we can approximate the left-censored observations as being Gaussian with response $t_i^2 - \frac{s}{2d_i}$ and variance $\frac{s^2}{d_i}$. And as for optimizing with respect to the variational parameter ξ_i, we set

$$\xi_i = +\sqrt{\mathsf{E}_q\Big[\frac{(t_i^2 - T_i)^2}{s^2}\Big]} = +\frac{1}{s}\sqrt{\overline{T_i^2} - 2t_i^2\overline{T_i} + t_i^{2\,2}}.$$

Right-censored Observations: For the left-censored observations, an application of the Jaakkola-Jordan bound gives

$$\begin{aligned}
\log\sigma\Big(\frac{T_i \quad t_i^1}{s}\Big) \quad & \frac{T_i \quad t_i^1}{2s} \quad \frac{1}{2\xi_i}\Big(\sigma(\xi_i) \quad \frac{1}{2}\Big)\Big(\frac{(T_i \quad t_i^1)^2}{s^2} \quad \xi_i^2\Big) \quad \log\Big(e^{\xi_i/2} + e^{\ \xi_i/s}\Big) \\
&= \frac{d_i}{2s^2}\Big(T_i^2 \quad 2t_i^1 T_i \quad \frac{s}{d_i}T_i\Big) + const \\
&= \frac{d_i}{2s^2}\Big(T_i \quad (t_i^1 + \frac{s}{2d_i})\Big)^2 + const.
\end{aligned}$$

Thus, we can approximate the right-censored observations as being Gaussian with response $t_i^1 + \frac{s}{2d_i}$ and variance $\frac{s^2}{d_i}$. And as for optimizing with respect to the variational parameter ξ_i, we set

$$\xi_i = +\sqrt{\mathsf{E}_q\Big[\frac{(T_i \quad t_i^1)^2}{s^2}\Big]} = +\frac{1}{s}\sqrt{\overline{T_i^2} \quad 2t_i^1\overline{T_i} + t_i^{12}}.$$

Interval-censored Observations: Finally, focusing on interval-censored observations, we can apply the Jaakkola-Jordan bound separately to each of the two terms. The the first term use variational parameter ξ_i^2, and for the second term use variational parameter ξ_i^1. This yields

$$\begin{aligned}
& \log\Big(e^{(t_i^2 \ \ T_i)/2s} + e^{\ (t_i^2 \ \ T_i)/2s}\Big) \quad \log\Big(e^{(t_i^1 \ \ T_i)/2s} + e^{\ (t_i^1 \ \ T_i)/2s}\Big) + const \\
& \frac{1}{2\xi_i^2}\Big(\sigma(\xi_i^2) \quad \frac{1}{2}\Big)\Big(\frac{(T_i \quad t_i^2)^2}{s^2} \quad \xi_i^{22}\Big) \quad \log\Big(e^{\xi_i^2/2} + e^{\ \xi_i^2/2}\Big) \\
& \frac{1}{2\xi_i^1}\Big(\sigma(\xi_i^1) \quad \frac{1}{2}\Big)\Big(\frac{(T_i \quad t_i^1)^2}{s^2} \quad \xi_i^{12}\Big) \quad \log\Big(e^{\xi_i^1/2} + e^{\ \xi_i^1/2}\Big) + const \\
= \quad & \frac{1}{2}\frac{d_i^2}{s^2}(T_i \quad t_i^2)^2 \quad \frac{1}{2}\frac{d_i^1}{s^2}(T_i \quad t_i^1)^2 + const \\
= \quad & \frac{1}{2}\frac{d_i^1 + d_i^2}{s^2}\Big[T_i \quad \frac{1}{d_i^1 + d_i^2}(d_i^1 t_i^1 + d_i^2 + t_i^2)\Big]^2 + const.
\end{aligned}$$

Thus, we can approximate the interval-censored observations as being Gaussian with response $\frac{d_i^1 t_i^1 + d_i^2 t_i^2}{d_i^1 + d_i^2}$ and variance $\frac{s^2}{d_i^1 + d_i^2}$. And as for optimizing with respect to the variational parameters ξ_i^1 and ξ_i^2, we set

$$\xi_i^1 \quad = +\sqrt{\mathsf{E}_q\Big[\frac{(T_i \quad t_i^1)^2}{s^2}\Big]} = +\frac{1}{s}\sqrt{\overline{T_i^2} \quad 2t_i^1\overline{T_i} + t_i^{12}}$$

and

$$\xi_i^2 \quad = +\sqrt{\mathsf{E}_q\Big[\frac{(T_i \quad t_i^2)^2}{s^2}\Big]} = +\frac{1}{s}\sqrt{\overline{T_i^2} \quad 2t_i^2\overline{T_i} + t_i^{22}}.$$

Proof for Ordinal Data

Restating the likelihood for ordered response data, given $K+1$ "knots" $\quad \theta_0 < \theta_1 <$ $< \theta_{K \ 1} < \theta_K =$, we have

$$\begin{aligned} & P(y_i \quad k) = \sigma(\theta_k \quad T(\ _1, \ldots, \ _L)_i) \\ ( ) \quad & P(y_i = k) = \sigma(\theta_k \quad T(\ _1, \ldots, \ _L)_i) \quad \sigma(\theta_{k \ 1} \quad T(\ _1, \ldots, \ _L)_i) \\ & = \frac{1}{1 + e^{\ (\theta_k \ T_i)}} \quad \frac{1}{1 + e^{\ (\theta_{k \ 1} \ T_i)}}. \end{aligned}$$

It is immediately obvious that, given these knots, this is equivalent to the likelihood of the AFT model with log-logistic noise, where:

- $s = 1$

  N.B. This model could be generalized as $P(y_i \quad k) = \sigma\Big(\big(\theta_k \quad T(\ _1, \ldots, \ _L)_i\big)/s\Big)$ *for* $s > 0$. *This could be advisable if the prior on* T_i *is scale-dependent*;

- for an observation with response $y_i = k \not\in \{1, K\}$, this probability is the same as in the AFT model for an interval-censored observation with log-survival time censored in the interval $(\theta_{k \ 1}, \theta_k]$;

- for an observation with response $y_i = 1$, this probability is the same as the AFT model for a left-censored observation with log-survival time censored at θ_1;
- for an observation with response $y_i = K$, this probability is the same as the AFT model for a right-censored observation with log-survival time censored at θ_{k-1}.

Thus, the only remaining thing to show is the lower-bound to the ELBO as a function of these knots so that we can maximize over these knots.

Using the Jaakkola-Jordan bound in the AFT model, for an observation with $y_i = k \notin \{1, K\}$, we get a lower-bound to the log-likelihood of

$$\begin{aligned}
&\frac{\theta_k + \theta_{k-1}}{2} + \log\left(1 + e^{\theta_{k-1} - \theta_k}\right) \\
&\frac{d_i}{2}\left[(T_i - \theta_k)^2 - \xi_i^{22}\right] - \log\left(e^{\xi_i^2/2} + e^{-\xi_i^2/2}\right) \\
&\frac{d_i}{2}\left[(T_i - \theta_{k-1})^2 - \xi_i^{12}\right] - \log\left(e^{\xi_i^1/2} + e^{-\xi_i^1/2}\right).
\end{aligned}$$

And for an observation with $y_i = 1$, we get a lower-bound to the log-likelihood of

$$\frac{\theta_1 + T_i}{2} + \frac{d_i}{2}\left[(\theta_1 - T_i)^2 - \xi_i^2\right] - \log\left(e^{\xi_i/2} + e^{-\xi_i/2}\right).$$

And for an observation with $y_i = K$, we get a lower-bound to the log-likelihood of

$$\frac{T_i + \theta_K}{2} + \frac{d_i}{2}\left[(T_i - \theta_K)^2 - \xi_i^2\right] - \log\left(e^{\xi_i/2} + e^{-\xi_i/2}\right).$$

Putting these together, we can write the lower-bound to the ELBO as a function of the θ_k's as

$$
\begin{aligned}
&\sum_{i:y_1=1} \frac{\theta_1}{2} \quad \frac{d_i}{2}(\theta_1^2 \quad 2\theta_1\overline{T_i}) \\
+\sum_{k=2}^{K}\sum_{i:y_i=k} &\frac{\theta_k \quad \theta_{k\ 1}}{2} + \log\left(1 \quad e^{\theta_{k\ 1}\ \theta_k}\right) \quad \frac{d_i^2}{2}(\theta_k^2 \quad \theta_k\overline{T_i}) \quad \frac{d_i^1}{2}(\theta_{k\ 1}^2 \quad 2\theta_{k\ 1}\overline{T_i}) \\
&+ \sum_{i:y_i=K} \frac{\theta_K}{2} \quad \frac{d_i}{2}(\theta_K^2 \quad 2\theta_K\overline{T_i}).
\end{aligned}
$$

Combining like-terms and counting carefully, we can simplify this as

$$
\begin{aligned}
&\frac{n_1}{2}\theta_1 \quad \theta_1^2 \sum_{i:y_i=1}\frac{d_i}{2} + 2\theta_1 \sum_{i:y_i=1}\frac{d_i}{2}\overline{T_i} \\
+ \quad &\sum_{k=2}^{K}\left[n_k\left[\frac{\theta_k \quad \theta_{k\ 1}}{2} + \log\left(1 \quad e^{k\ 1\ \theta_k}\right)\right] \quad \theta_k^2 \sum_{i:y_i=k}\frac{d_i^2}{2} + 2\theta_k \sum_{i:y_i=k}\frac{d_i^2}{2}\overline{T_i}\right. \\
&\qquad \left. \theta_{k\ 1}^2 \sum_{i:y_i=k}\frac{d_i^1}{2} + 2\theta_{k\ 1} \sum_{i:y_i=k}\frac{d_i^1}{2}\overline{T_i}\right] \\
&\frac{n_K}{2}\theta_{K\ 1} \quad \theta_{K\ 1}^2 \sum_{i:y_i=K}\frac{d_i}{2} + 2\theta_{K\ 1} \sum_{i:y_i=K}\frac{d_i}{2}\overline{T_i},
\end{aligned}
$$

where $n_k = \#\{i : y_i = k\}$.

Finally, combining terms for the same θ_k and θ_k^2, we get

$$
\begin{aligned}
\sum_{k=1}^{K\ 1}\Bigg[&\theta_k^2\Big[\sum_{i:y_i=k}\frac{d_i^2}{2} + \sum_{i:y_i=k+1}\frac{d_i^1}{2}\Big] + \theta_k\Big[\frac{n_k \quad n_{k+1}}{2} + \sum_{i:y_i=k} d_i^2\overline{T_i} + \sum_{i:y_i=k+1} d_i^1\overline{T_i}\Big] \\
&+ n_k \log\left(1 \quad e^{\theta_{k\ 1}\ \theta_k}\right)\Bigg],
\end{aligned}
$$

with the convention that $\theta_0 = \quad \infty$.

With this expression in place, we can use off the shelf numerical optimization methods to maximize this function over $\theta_1 < \theta_2 < \quad < \theta_{K\ 1}$.

Proof for Pairwise Ranking Data

Section 4.2.6 contains sufficient details.

Proof for Listwise Ranking Data

Section 4.2.6 contains sufficient details.

Proof for Cox Proportional Hazards Model

Restating the partial log-likelihood for the Cox proportional hazards model using Breslow's method for handling ties, we have

$$l(\ \ _1, \ldots, \ \ _L; \mathbf{y}, \mathbf{c}, h_1, \ldots, h_L, T) = \sum_{i:c_i=1} \log \frac{e^{T_i}}{\sum_{j:y_j \ \ y_i, j \in i} e^{T_j}},$$

where, as before, y_i is the survival time, and c_i is an indicator for if observation i is uncensored ($c_i = 1$) or right-censored ($c_i = 0$).

Performing some algebra, along with the same combination of the Titsias and Jaakkola-Jordan bounds used in the multinomial case, we can lower-bound this partial log-likelihood with:

$$
\begin{aligned}
&\sum_{i:c_i=1} \log \frac{e^{T_i}}{\sum_{j:y_j = y_i, j \neq i} e^{T_j}} \\
&\Big[\text{Titsias bound}\Big] \\
&\sum_{i:c_i=1} \sum_{j:y_j = y_i, j \neq i} \log \sigma(T_i - T_j) \\
&\Big[\text{Jaakkola-Jordan bound}\Big] \\
&\sum_{i:c_i=1} \sum_{j:y_j = y_i, j \neq i} \frac{T_i - T_j}{2} - \frac{d_{ij}}{2}\Big(T_i^2 - 2T_iT_j + T_j^2 - \xi_{ij}^2\Big) - \log\Big(e^{\xi_{ij}/2} + e^{-\xi_{ij}/2}\Big) \\
= \quad & \frac{1}{2}\mathbf{T}^T\mathbf{A}\mathbf{T} + \mathbf{b}^T\mathbf{T} + c,
\end{aligned}
$$

where

$$
A_{ij} = \begin{cases} c_i - \Big[\sum_{k:y_k = y_i, k \neq i} d_{ik}\Big] - \sum_{k:y_i = y_k, c_k = 1, k \neq i} d_{ki}, & \text{if } i = j \\ c_i - \mathbf{I}_{y_j = y_i} d_{ij} + c_j - \mathbf{I}_{y_i = y_j} d_{ji}, & \text{if } i \neq j \end{cases} \qquad d_{ij} = \frac{1}{\xi_{ij}}\Big(\sigma(\xi_{ij}) - \frac{1}{2}\Big)
$$

$$
\mathbf{b}_i = \frac{1}{2}\Big[c_i - \sum_{j:c_j=1, j \neq i} \mathbf{I}_{y_i = y_j}\Big]
$$

$$
c = \sum_{i:c_i=1} \sum_{j:y_j = y_i, j \neq i} \log \sigma(\xi_{ij}) + \frac{\xi_{ij}}{2}\Big(d_{ij}\xi_{ij} - 1\Big).
$$

This last step, going from the double-sum to the matrix formula, simply involves careful accounting and grouping of terms relating to T_i^2 and T_iT_j.

We can also use the results for the optimal ξ_{ij} from the binomial case in order to see that

$$
\xi_{ij} = +\sqrt{\mathsf{E}_q[T_i^2 - 2T_iT_j + T_j^2]} = +\sqrt{\overline{T_i^2} - 2\overline{T_iT_j} + \overline{T_j^2}}.
$$

Note that the middle term is the expectation of the product of T_i and T_j, and not the

product of their expectations.

Proof for Multivariate Gaussian Data

Section 4.2.8 contains sufficient details.